



# **Bienvenue à la Sfeir School**

## **ANGULAR 200**

WIFI : SFEIRGUEST



# Présentation



[sf≡ir]

**Bruno Bellenoue**

Developer & Team Leader  
@BrunoBellenoue

# Présentation



[sf≡ir]

**Jean-Rodolphe  
Poinso**

Developer & Team Leader

# Déroulement de la formation

C'est quand la pause ?  
Quand est-ce qu'on mange ?  
Tour de table...

**Feuille de présence (obligatoire)**

# Slides de la formation

<http://bit.ly/sfeir-school-angular>

(accès restreint)

## Déroulement de la formation

# Github de la formation

[github.com/Sfeir/angular-200](https://github.com/Sfeir/angular-200)

Angular2 People 200

Wassim (GDE)

localhost:4200/#/people

people

MapsList

Found 100 people

Leanne Woodard


BIOSPAN

Leanne.Woodard@BIOSPA...

0784112248

Manager : Erika

Location : SFEIR



Castaneda Salinas


METROZ

Castaneda.Salinas@METR...

0145652522

Manager : Erika

Location : SFEIR



Phyllis Donovan


PEARLESSA

Phyllis.Donovan@PEARLES...

0685230125

Manager : Erika

Location : SFEIR



Erika Guzman


CIRCUM

Erika.Guzman@CIRCUM.com

0678412587

Manager : Mercedes

Location : SFEIR



Moody Prince


TRIPSCH

Moody.Prince@TRIPSCH.c...

0662589632

Manager : Mercedes

Location : SFEIR



Mercedes Hebert


QUINTITY

Mercedes.Hebert@QUINTIT...

0125878522

Manager : Mclaughlin

Location : SFEIR



8



# Déroulement de la formation

```
$ git clone github.com/Sfeir/angular-200.git
```

# Déroulement de la formation

```
$ npm install -g @angular/cli
```

```
$ npm install
```

```
$ npm run server
```

```
$ npm run client -- <step>
```

```
client -> http://localhost:4200/
```

```
server -> http://localhost:9000/
```

*(cf. [ndm](#) & [angular console](#) si vous préférez les interfaces graphiques)*

# Extensions VS Codes



**Angular Language Service** 0.1.10

Editor services for Angular templates

**Angular**



**TSLint** 1.0.40

TSLint for Visual Studio Code

**egamma**



**Prettier - Code formatter** 1.7.2

VS Code plugin for prettier/prettier

**Esben Petersen**



**EditorConfig for VS Code** 0.12.5

EditorConfig Support for Visual Studio Code

**EditorConfig**

# Déroulement de la formation

Un concept clé d'Angular

Un TP

- un projet d'exercice : `steps/<project>`
- une projet de solution : `steps/<project>-solution`

# Angular next

2

.

3

.

1

**Major**

*Breaking  
change*

**Minor**

*New features,  
not breaking*

**Patch**

*Bugfixes,  
not breaking*

# Quickstart

# Angular CLI

```
angular-200 on master [$] is v0.0.0 via v10.13.0 with A v7.1.0 ✖ v7.0.6
```

```
i76% at 18:16 → ng help
```

Available Commands:

- `add` Adds support for an external library to your project.
- `build` (b) Compiles an Angular app into an output directory named dist/ at the given output path. Must be executed from within a workspace directory.
- `config` Retrieves or sets Angular configuration values.
- `doc` (d) Opens the official Angular documentation (angular.io) in a browser, and searches for a given keyword.
- `e2e` (e) Builds and serves an Angular app, then runs end-to-end tests using Protractor.
- `generate` (g) Generates and/or modifies files based on a schematic.
- `help` Lists available commands and their short descriptions.
- `lint` (l) Runs linting tools on Angular app code in a given project folder.
- `new` (n) Creates a new workspace and an initial Angular app.
- `run` Runs a custom target defined in your project.
- `serve` (s) Builds and serves your app, rebuilding on file changes.
- `test` (t) Runs unit tests in a project.
- `update` Updates your application and its dependencies. See <https://update.angular.io/>
- `version` (v) Outputs Angular CLI version.
- `xi18n` Extracts i18n messages from source code.

For more detailed help run "ng [command name] --help"

[github.com/angular/angular-cli](https://github.com/angular/angular-cli)  
[angular.io/cli](https://angular.io/cli)

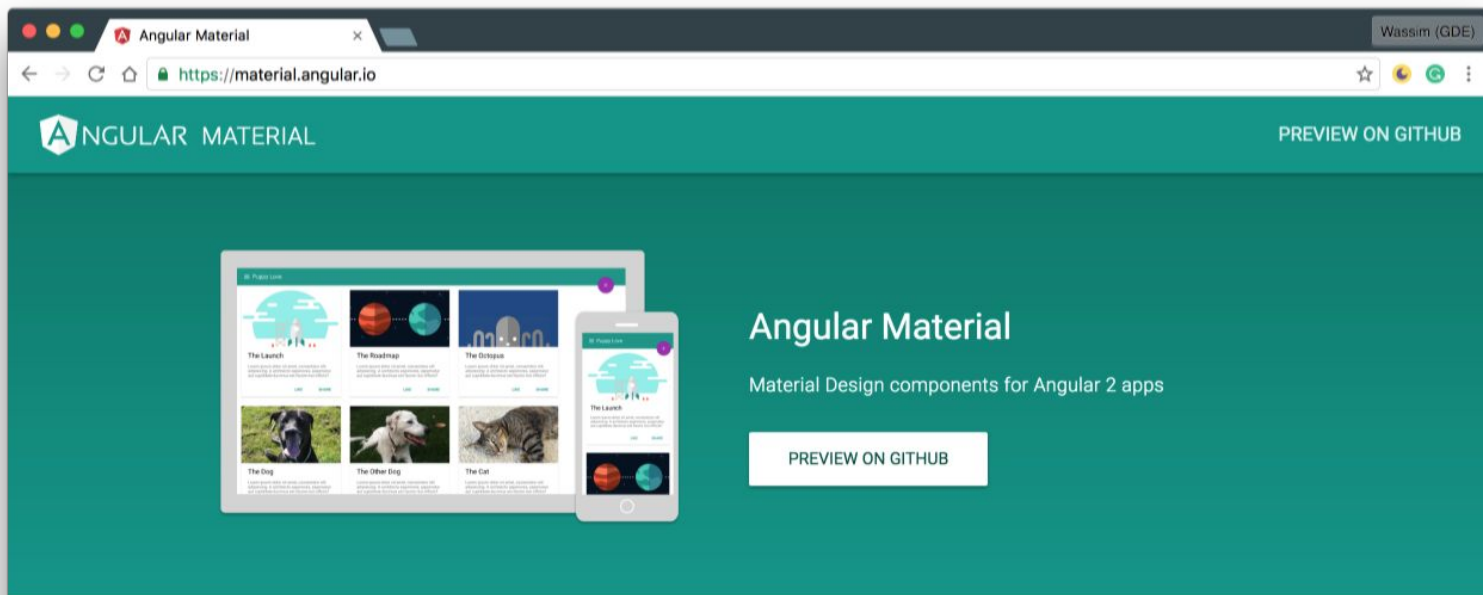
# Configuration

```
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "version": 1,
  "newProjectRoot": "projects",
  "projects": {
    "angular-200": {
      "root": "",
      "sourceRoot": "src",
      "projectType": "application",
      "architect": {
        "build": {...
      }
    }
  },
  "e2e": {...
}
...
}
```

**angular.json**



# Un mot sur le Material Design



## Sprint from Zero to App

Hit the ground running with comprehensive, modern UI components that work across web, mobile and desktop.

# Un mot sur le Material Design

- Le TP utilise des composants [Material Design](#)
  - `mat-toolbar`
  - `button[mat-fab] , button[mat-button]`
  - `mat-card`
  - `mat-checkbox`
  - ...
- Ce n'est pas une formation sur Material Design
- Vous n'êtes pas obligé d'utiliser ces composants

# Les langages supportés

- **TypeScript**

- ES2018+
- types (optionnels)
- annotations

- **Javascript**

- ES6
- ES5

- **Dart**

```
@Component({  
  selector: 'sfeir-app',  
  template: '<h1>My First Angular App</h1>'  
})  
class AppComponent { }
```

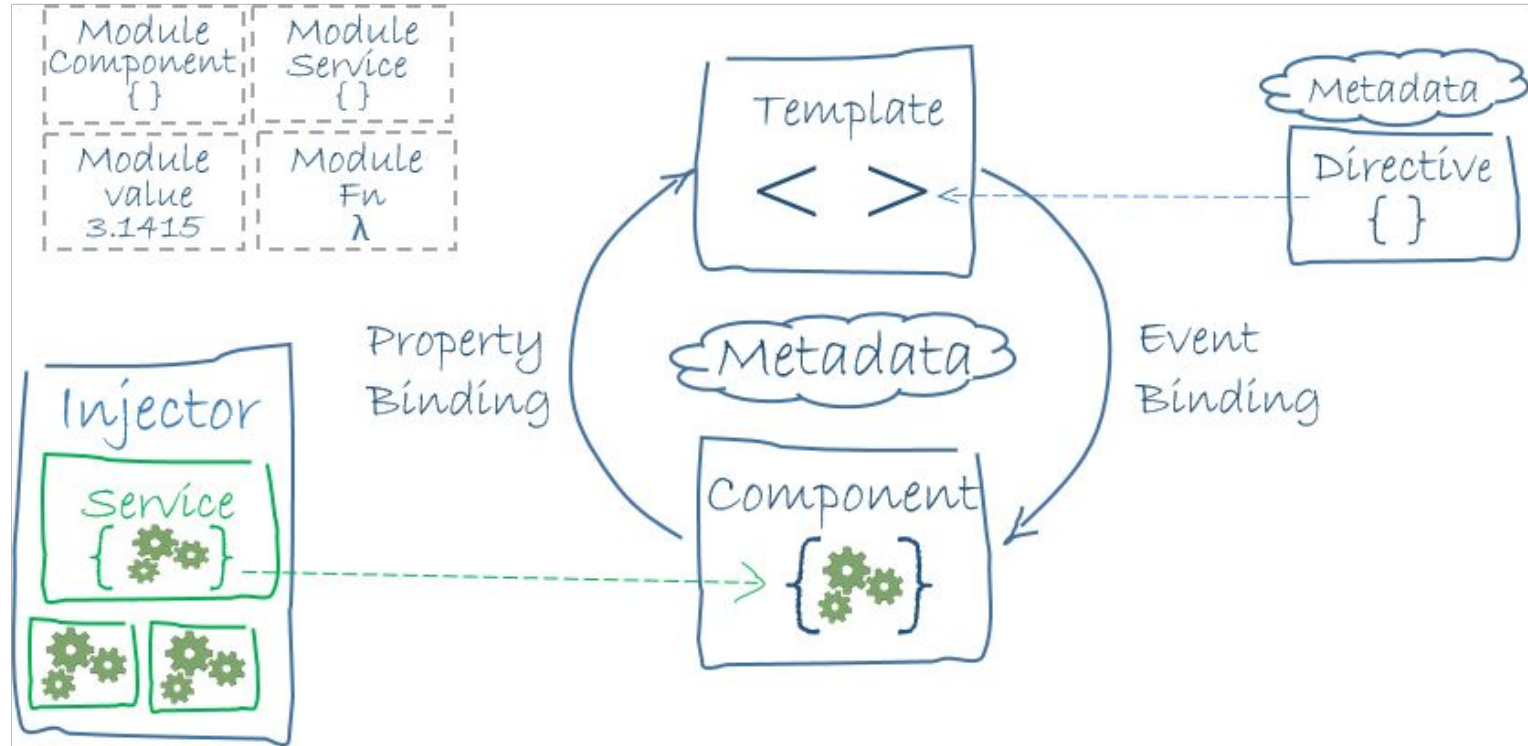
```
var AppComponent = ng.core.Component({  
  selector: 'sfeir-app',  
  template: '<h1>My First Angular App</h1>'  
})  
.Class({  
  constructor: function () { }  
});
```

# index.html

```
<html>  
<head></head>  
<body>  
  <sfeir-app>Loading...</sfeir-app>  
</body>  
</html>
```

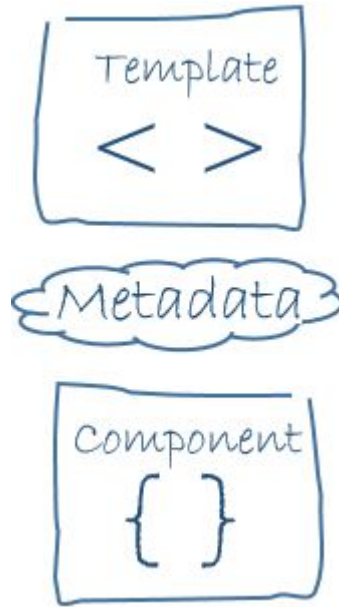
# Architecture Globale

# Architecture globale



# Votre application : un composant

- 3 concepts de base



# Un composant : annotation + classe

- La logique du composant: utilise la syntaxe de classe de ES2015

```
export class AppComponent {  
  
    name: string;  
  
    constructor(){  
        this.name = 'Angular';  
    }  
  
}
```



# Un composant : annotation + classe

- les annotations (comment afficher le composant dans la page)

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

```
@Component({  
  selector: 'sfeir-app',  
  templateUrl: './app.component.html'  
})
```

# Un composant : annotation + classe

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

```
@Component({  
  selector: 'sfeir-app',  
  templateUrl: './app.component.html'  
})  
export class AppComponent {  
  name: string;  
  constructor(){  
    this.name = 'Angular';  
  }  
}
```

# NgModule

# Un module...

- Permet de regrouper des fonctionnalités
- Au moins un module par application
- Peut être chargé de façon asynchrone
- Différents types de modules
  - Root (App) module
  - Feature Module
  - Shared module
  - Core module

# Exemple d'un module

```
import { NgModule } from '@angular/core';  
import { BrowserModule } from '@angular/platform-browser';  
import { AppComponent } from '../app/app.component';
```

```
@NgModule({  
  imports: [ BrowserModule, /*...*/ ],  
  declarations: [ AppComponent, /*...*/ ],  
  providers: [],  
  bootstrap: [ AppComponent ]  
})  
export class AppModule { }
```

# Bootstrap

- Pour charger l'application dans la page

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';  
import { AppModule } from './app.module';  
  
platformBrowserDynamic().bootstrapModule(AppModule);
```

# Soit notre application

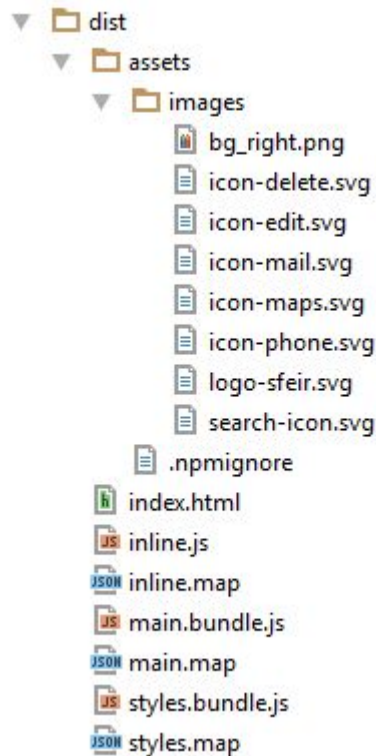
```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

@NgModule({
  imports: [ BrowserModule, /*...*/ ],
  declarations: [ AppComponent, /*...*/ ],
  providers: [],
  bootstrap: [ AppComponent ]
})
export class AppModule { }

platformBrowserDynamic().bootstrapModule(AppModule);
```

# Webpack

- Bundle en javascript
- hot reload
- Choix par défaut d'angular





# Exercice 1 : prise en main



1

## steps/hands-on

Mise en place de votre premier composant (à la main)

- Créer un composant **PeopleAppComponent** (à la main)
  - `src/app/app.component.ts`
  - `<sfeir-app></sfeir-app>`
- Utiliser le template et le CSS fournis
  - `src/app/app.component.html`
  - `src/app/app.component.css`
- Configurer le module de l'application: `src/app/app.module.ts`
- Afficher du texte dans la variable `{{ name }}`

# SOLUTION

steps/hands-on-solution

# Initialiser un projet / workspace

```
$ ng new my-awesome-app
```

- génère l'arborescence de l'application
- initialise un repo Git + 1er commit
- installe les deps NPM

# Initialiser un composant

**\$ ng generate component user**

- génère les fichiers d'un composant
  - `src/app/user/user.component.css`
  - `src/app/user/user.component.html`
  - `src/app/user/user.component.spec.ts`
  - `src/app/user/user.component.ts`

# Initialiser un service

**\$ ng generate service user**

- génère les fichiers d'un service
  - `src/app/user.service.spec.ts`
  - `src/app/user.service.ts`

**\$ ng generate service shared/user**

- génère les fichiers d'un service
  - `src/app/shared/user.service.spec.ts`
  - `src/app/shared/user.service.ts`

# Générer...

```
angular-200 on master [$] is v0.0.0 via v10.13.0 with A v7.1.0 X v7.0.6  
73% at 18:24 → ng generate --help  
Generates and/or modifies files based on a schematic.  
usage: ng generate <schematic> [options]
```

arguments:

**schematic**

The schematic or collection:schematic to generate.

options:

**--defaults**

When true, disables interactive input prompts for options with a default.

**--dry-run (-d)**

When true, run through and report activity without writing out results.

**--force (-f)**

When true, force overwriting of existing files.

**--help**

Shows a help message for this command in the console.

**--interactive**

When false, disables interactive input prompts.

Available Schematics:

Collection "@schematics/angular" (default):

- appShell
- application
- class
- component
- directive
- enum
- guard
- interface
- library
- module
- pipe
- service
- serviceWorker
- universal

# 2

## Exercice 2 : Utilisez le CLI

### steps/ngg

Mise en place de votre premier composant avec le CLI

- Créer un composant **HomeComponent** avec le CLI
  - `cd steps/ngg/src`
  - `ng g c home`
- Examiner les fichiers générés dans `src/app/home/`
- Compléter `src/app/home.component.html` et `src/app/home.component.ts`
  - afficher par exemple la chaîne “Hello {{ name }}”
- Importer **HomeComponent** dans `src/app.module.ts`
  - Ajouter le dans “declarations” et “bootstrap”
  - remplacer l’ancien composant **PeopleAppComponent** dans “bootstrap”
- Changer le nom de l’élément HTML dans `src/index.html`
  - utiliser le sélecteur de **HomeComponent**

# SOLUTION

steps/ngg-solution



# Databinding & template

# JavaScript

```
<html>
  Bonjour <span id="name"></span>
  <input type="text"/>
</html>
```

```
window.onload = function(){
  var span = document.querySelector('name');
  var input = document.getElementsByTagName('input')[0];

  input.onkeyup = function(){
    if (span.textContent || span.textContent === "") {
      span.textContent = input.value;
    } else if(span.innerText || span.innerText === "") { // IE
      span.innerText = input.value;
    }
  };
};
```

# jQuery

```
<html>
  Bonjour <span id="name"></span>
  <input type="text"/>
</html>
```

```
$(document).ready(function() {

    var $input = $('input');
    var $span = $('#name');

    $input.keyup(function (event) {
        $span.text(event.target.value);
    });
});
```

# Angular

```
<div>  
  <input type="text" name="myName" [(ngModel)]="myName">  
  <p>Bonjour {{myName}}</p>  
</div>
```

# Syntaxe

## Properties, events et références

```
<div> My name is {{ name }} </div>
<div>
  <input      #newname      type="text">

  <button (click)="changeName(newname.value)"
    [disabled]="newname.value == 'Angular 2'">Change Name

</button>
</div>
```

# anatomie d'un binding

target="expression"

# Interpolation et expression

- Interpolation

```
<div>Hello {{name}}</div>  

```

- Les expressions

- dans le contexte du composant
- du JS mais
  - pas d'affectation (sauf pour les events)
  - pas d'accès aux variables globales (window, document..)
  - Pour les opérateurs logiques, tout est évalué
  - Pas de new, ++, --

# 3 catégories de binding

Direction	Syntaxe	Type
unidirectionnel depuis le modèle vers la vue	<pre>{{ expression }} [ targetFooBar ] = "expression" bindTargetFooBar = "expression"</pre>	Interpolation Propriétés Classe Attribut Style
Unidirectionnel depuis la vue vers le model	<pre>( targetFooBar ) = "expression" onTargetFooBar = "expression"</pre>	Événements
bidirectionnel	<pre>[( targetFooBar )] = "expression" bindonTargetFooBar = "expression"</pre>	bidirectionnel



# Mais avant : attributs vs propriétés

- Les attributs c'est du **HTML**, les propriétés c'est du **DOM**
  - mapping strict (id)
  - attribut sans propriété (colspan)
  - propriété sans attribut (textContent)
  - les 2 mais....
- La plupart du temps, les attributs servent à initialiser les propriétés mais ne sont pas modifiés si la propriété change
- Des attributs sans valeurs : `<bouton disabled >Click!!</bouton>`
- Un attribut: une chaîne de caractère

# Que des propriétés ...

- Un monde sans attributs
- Avec le binding, nous travaillons sur les **propriétés** (des éléments, composants ou directives)

```
<bouton [disabled]="true" >Click!!</bouton>
```

- Permet de passer des objets

# Property binding

Type	cible	exemple
propriété	Attribut d'élément Attribut de component Attribut de directive	<code>&lt;img [src]="someUrl" /&gt;</code> <code>&lt;my-component [data]="currentData"&gt;&lt;/my-component&gt;</code> <code>&lt;div [ngClass]="{selected: isSelected}"&gt;&lt;/div&gt;</code>

- Forme canonique: **bindCapitalAttr**
- Constantes

`<show-title title="Some Title"></show-title>`

`<show-title [title]=" 'Some Title' "></show-title>`

# Ok mais si je veux un attribut....

- Les éléments n'ont pas forcément la propriété (ex: aria, svg, colspan)

- On peut cibler un attribut en précédant le nom de **attr**.

```
<td [attr.colspan]="1+1">a cell!!</td>
```

- pour les classes précède le nom de la classe par **class**.

```
<div [class.isSpecial]="isSpecial">special class</div>
```

- pour les styles précède le nom de la propriété par **style**.

```
<div [style.color]="isSpecial ? 'red' : 'green'">Special class</div>
```

# Event binding

Type	cible	exemple
Evènement	Évènement d'élément Évènement de composant Évènement de directive	<code>&lt;button (click)="onSave()"&gt;&lt;/button&gt;</code> <code>&lt;hero-detail (deleted)="onDelete(\$event)"&gt;...</code> <code>&lt;input (change)="firstName = \$event" /&gt;</code>

- Forme canonique: **onCapitalAttr**
- Référence à l'évènement grâce à **\$event**

# 2 way binding

Type	cible	exemple
bidirectionnel	Propriétés Événement de directive	<code>&lt;input name="firstName" [(ngModel)]="firstName" /&gt;</code>

- équivalent à

```
<input [(ngModel)]="firstName" (ngModelChange)="firstName=$event">
```

- Note : ***ngModel*** est fourni par le package ***@angular/forms***

# Variables locales

- Variables :

- une valeur (**let**)

```
<movie-detail  
  *ngFor="let movie of movies">  
</movie-detail>
```

- Références :

- l'élément (**#** ou **ref-XXX**)
- disponible dans :
  - tout le template
  - le composant

```
<input #phone >  
<button (click)="click(phone.value)">Call</button>
```

```
<input ref-fax >  
<button (click)="click(fax.value)">Fax</button>
```

# Composants

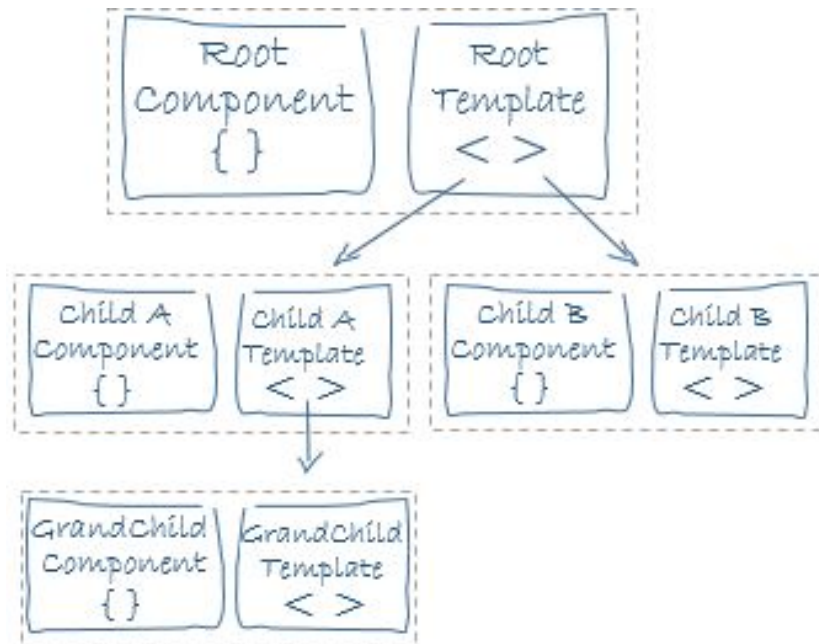


## 2 types de composants

- La "**Directive**" permet d'enrichir un élément HTML...
- Le "**Component**" est une **directive** avec une vue et des styles CSS

# Composants : arbre

- Arbre de composant
- Les “enfants” sont ajouté au parent s'ils apparaissent dans son template
- Les composants doivent être déclarés dans le module



# Composant : @Component()

- Component
  - selector
  - template et templateUrl
  - providers
  - ...

```
@Component({  
  selector: 'sfeir-app',  
  templateUrl: 'home.component.html',  
  ...  
})
```

# Imbriquer les composants

- Lorsqu'un composant parent utilise des composants enfants
  - Ils doivent être référencés
  - Ils doivent être déclarés dans les déclarations du **@NgModule()**

```
import { HomeComponent } from './app/home/';  
import { FooDirective } from './app/shared/';  
  
@NgModule({  
  declarations: [HomeComponent, FooDirective]  
})
```

# 3

## Exercice 3 : Imbriquer les composants

### steps/cpt-hierarchy

- Faites en sorte que le composant **PeopleAppComponent** utilise **HomeComponent**
- Voir le contenu du fichier `src/app/home/home.component.html`
- Mettre à jour les fichiers suivants :
  - `src/index.html`
  - `src/app/app.module.ts`
  - `src/app/app.component.html`
  - `src/app/app.component.ts`

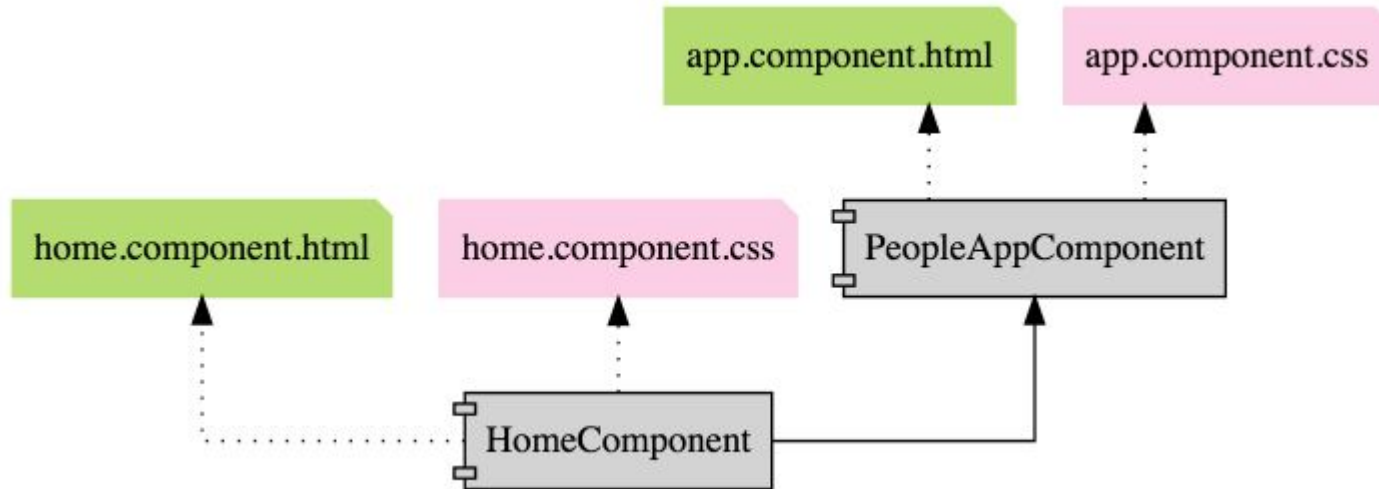
# SOLUTION

steps/cpt-hierarchy-solution

# 3

## Exercice 3 : Imbriquer les composants

steps/cpt-hierarchy-solution



# 4

## Exercice 4 : Afficher une personne

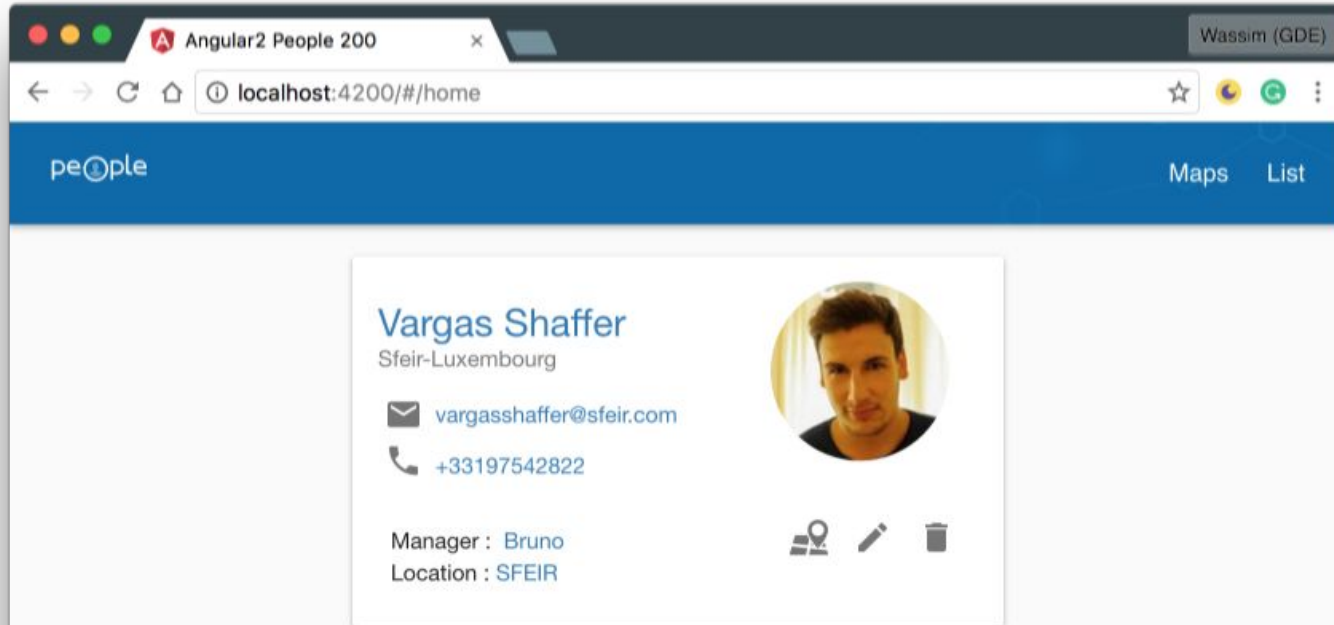
### steps/binding

- Dans le composant **HomeComponent** nous allons afficher les détails d'une personne
- Utiliser les fichiers suivants comme exemple pour le contenu :
  - src/app/\_static/home.component.html
  - src/app/\_static/home.component.css
  - src/app/\_static/people.ts
- Utiliser les données de people.ts dans src/app/home.component.ts
  - `import { PEOPLE } from '../_static/people'`



# 4

## Exercice 4 : Afficher une personne



# Gestion des événements DOM

# Les événements

- Nom de l'événement entre ( )
- Fait référence à une fonction de la classe
- Pour récupérer les détails de l'événement : *\$event*

```
export class MyComponent {  
  values: string = '';
```

```
  constructor(){}  
  
  updateValue(event){
```

```
    this.values += event.target.value + ' | ';
```

```
  }
```

<input

(click)="updateValue(\$event)">

# 5

## Exercice 5 : gérer un clic

### steps/events

- Un bouton “random” a été ajouté dans `src/app/home.component.html`
- Ajouter un clic sur ce bouton pour afficher une personne au hasard
  - exploiter le tableau `PEOPLE` fourni

# SOLUTION

steps/events-solution

# Communication Serveur

# Utilisation du Service HTTP

```
import { NgModule } from '@angular/core';  
import { HttpClientModule } from '@angular/common/http';  
  
@NgModule({  
  imports: [ HttpClientModule ],  
  ...  
});
```

- Intégrer `HttpClientModule` dans notre **NgModule**

# Utilisation dans un composant

Pour utiliser le service il faut l'injecter dans le composant

- Importer le service Http

```
import {Component} from '@angular/core';  
import {HttpClient} from '@angular/common/http';
```

```
@Component()  
export class MyClass {  
  myHttp:HttpClient;  
  
  constructor(http: HttpClient) {  
    this.myHttp = http;  
  }  
}
```

## ASTUCE TYPESCRIPT

```
import {Component} from '@angular/core';  
import {HttpClient} from '@angular/common/http';  
  
@Component()  
export class MyClass {  
  constructor(  
    public http: HttpClient  
  ) {}  
}
```



# Usages

Les méthodes disponibles:

```
this.http.get(url, options);
```

```
this.http.post(url, data, options);
```

```
this.http.put(url, data, options);
```

```
this.http.delete(url, options);
```

# Envoyer des données (POST/PUT)

- Les données sont envoyées en JSON
- Possibilité d'envoyer des entêtes en 3eme argument

```
http.post(  
    url,  
    datas,  
    {headers: new HttpHeaders().set('Authorization', 'my-auth-token')}}  
);
```

# Subscribe

Les méthodes renvoient un observable. Il faut **souscrire** pour déclencher la requête

```
this.http.get(url, options)  
  .subscribe( datas=>{  
    //Do something with datas  
  });
```

# Gérer les retours

- Par défaut en JSON
- {responseType: 'text'}

```
http.get(url, options)  
  .subscribe((data) => {  
    /* do something */  
  });
```

# Le réponse

- On peut accéder à la réponse complète

```
this.http.get(url, {observe: 'response'}).subscribe(  
  resp=>{  
    console.log(resp.headers.get('X-Custom-Header'));  
    console.log(resp.body.someField);  
  });
```

# Exercice 6 : fini les données en dur

## 6

### steps/http

Récupérer les personnes à partir du serveur

- Utiliser Http pour récupérer les données depuis le serveur
- Pensez à ajouter le module HttpClientModule dans **NgModule**
- GET <http://localhost:9000/api/peoples>
  - retourne une collection de PEOPLE
- GET <http://localhost:9000/api/peoples/random>
  - retourne une personne au hasard

# SOLUTION

steps/http-solution

# Navigation



http://localhost:4200/#/home

Angular2 People 200

Wassim (GDE)

← → ↻ 🏠

localhost:4200/#/home

☆ 🌙 🟢 ⋮

people

MapsList

Do you know?

Mercedes Hebert

QUINTITY


✉

Mercedes.Hebert@QUINTIT...

📞

0125878522

Manager : [Mclaughlin](#)  
Location : [SFEIR](#)



📍

✎

🗑

⌕

🔄

⌕

🔄

http://localhost:4200/#/people/5763cd4d979b62a209809160

Angular2 People 200

Wassim (GDE)

← → ↻ 🏠

localhost:4200/#/people/5763cd4d979b62a209809160

☆ 🌙 🟢 ⋮

people

MapsList

Fiche personnelle

Mercedes Hebert

QUINTITY

✉


Mercedes.Hebert@QUINTIT...

📞

0125878522

Manager : [Mclaughlin](#)

Location : [SFEIR](#)



📍

✎

🗑

Skills

ex

commodo

pariatur

sit

aute

[≡] Sch

http://localhost:4200/#/people

Angular2 People 200

Wassim (GDE)

localhost:4200/#/people

☆ 🌙 🟢 ⋮

people

MapsList

Found 10 people


Leanne Woodard

BIOSPAN

✉ Leanne.Woodard@BIOSPA...

📞 0784112248

Manager : [Erika](#)  
Location : [SFEIR](#)



📍

✎

🗑


Castaneda Salinas

METROZ

✉ Castaneda.Salinas@METR...

📞 0145652522

Manager : [Erika](#)  
Location : [SFEIR](#)



📍

✎

🗑


Phyllis Donovan

PEARLESSA

✉ Phyllis.Donovan@PEARLES...

📞 0685230125

Manager : [Erika](#)  
Location : [SFEIR](#)



📍

✎

🗑


Erika Guzman

CIRCUM

✉ Erika.Guzman@CIRCUM.com

📞 0678412587

Manager : [Erika](#)  
Location : [SFEIR](#)



📍

✎

🗑

+

http://localhost:4200/#/edit/5763cd4d979b62a209809160

Angular2 People 200 x Wassim (GDE)

← → ↻ 🏠 ⓘ localhost:4200/#/edit/5763cd4d979b62a209809160 ☆ 🌙 🌐 ⋮

peOple Maps List

### Update Mercedes Hebert

ID (disabled)  
5763cd4d979b62a209809160

First name \*  
Mercedes

Last Name \*  
Hebert

Email \*  
Mercedes.Hebert@QUINTITY.com


Address  
Laurel Avenue

City Postal Code  
Northchase 85752

Phone \* (ex: 0612345678)  
0125878522

☒ Manager

Cancel Save



<http://localhost:4200/#/locator>

Angular2 People 200 x Wassim (GDE)

localhost:4200/#/locator

people Maps List

Map Satellite

Paris

Mercedes Hebert

Google

Map data ©2016 Google Terms of Use Report a map error

The screenshot shows a web browser window with the title 'Angular2 People 200' and a user 'Wassim (GDE)'. The address bar shows 'localhost:4200/#/locator'. The application has a blue header with the 'people' logo and 'Maps' and 'List' tabs. The main content is a map of Paris with several red location pins. A profile card for 'Mercedes Hebert' is overlaid on the map, showing a photo of a woman. The map includes labels for various Parisian districts and landmarks.

# Configuration (simple)

- **path** : l'URL de route (ex: /people/:id)
- **component** : le composant associé à cette route (ex: PeopleComponent)
- **redirectTo** : le fragment d'URL vers lequel rediriger route courante (ex: '/home')
- **pathMatch** : stratégie de redirection (full / prefix)
  - full: tente une reconnaissance depuis la racine de la route
  - prefix: tente une reconnaissance partielle de la route

# Configuration (complète)

- **path** : l'URL de route (ex: /people/:id)
- **component** : le composant associé à cette route (ex: AppComponent)
- **redirectTo** : le fragment d'URL vers lequel rediriger la route courante
- **pathMatch** : stratégie de redirection (full / prefix)
- **outlet** : le nom de l'emplacement dans lequel le composant doit s'afficher
- **data** : données passées à la route via ActivatedRoute
- **canActivate / canDeactivate** : permet d'activer ou non la route
- **resolver** : récupère des données avant de naviguer vers la route
- **children** : un tableau de définition des sous-routes



# Example : routes definition

```
//routes.module.ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

import { HomeComponent } from './home/';
import { PeopleComponent } from './+people/';
import { PersonComponent } from './+person/';

const ROUTES: Routes = [
  {path: '', redirectTo: 'home', pathMatch: 'full'},
  {path: 'home', component: HomeComponent},
  {path: 'people', component: PeopleComponent},
  {path: 'people/:id', component: PersonComponent},
];

export const RoutesModule = RouterModule.forRoot(ROUTES);
```



# Exemple : Utilisation des routes

```
//app.module.ts
```

```
import { NgModule } from '@angular/core';  
import { AppRoutes } from './app.routes';
```

```
@NgModule({  
  imports: [  
    AppRoutes,  
    //...  
  ],  
  ...  
})  
export class AppModule { }
```

# Stratégie de navigations

- Par “Path”: PathLocationStrategy (Mode HTML5 et pushState=>Par défaut)
  - ex: localhost/people/1
    - `{useHash: false}`
- Par “Hash”: HashLocationStrategy
  - ex: localhost/#/people/1
    - `{useHash: true}`

```
RouterModule.forRoot(ROUTES, {useHash: true});
```

# Utilisation dans un composant : TypeScript

```
import { Component, OnInit } from '@angular/core';  
import { ActivatedRoute, Router } from '@angular/router';
```

```
@Component({})
```

```
export class FooComponent implements OnInit {
```

```
  constructor(private route: ActivatedRoute, private router: Router ) { }
```

```
  ngOnInit() {  
    this.route.params.subscribe(params => {  
      let id = params['id'];  
      //...  
    });  
  }
```

```
  go() { this.router.navigate(['/people/']); }
```

```
}
```

# Utilisation dans un composant : HTML

```
<a class="btn btn-info" routerLink="/people">Movies Liste</a>
<a class="btn btn-info" [routerLink]="['/people']">Movies Liste</a>
<a class="btn btn-info" [routerLink]="['/people/edit/', person.id]">Edit</a>

<section class="container">

  <router-outlet></router-outlet>

</section>
```

# 7

## Exercice 7 : une navigation côté client

### steps/router

- Compléter le fichier `src/app/app.routes.ts` avec la configuration des routes
- Mettre à jour le fichier `src/app/app.module.ts`
- Utiliser le `<router-outlet></router-outlet>` dans le fichier :
  - `src/app/app.component.html`

# SOLUTION

steps/router-solution

# Ajoutons des fonctionnalités

# Itérer sur une collection avec \*ngFor

- Itère dans une collection et génère un template par élément
- *index, odd, even, last* à utiliser en alias dans des variables

```
<ul>
  <li *ngFor="let fruit of fruits; let i=index">
    <!-- répétition du template li par élément fruit -->
    {{ i }} : {{ fruit.name }}
  </li>
</ul>
```



# 8

## Exercice 8 : Répéter les personnes

### steps/ngfor

Afficher la liste des personnes en utilisant la directive \*ngFor

- Créer un composant **PeopleComponent** pour afficher la liste des personnes
  - ng g c people
- Lui associer une route `#/people` accessible depuis le lien List (en haut dans la toolbar)
- Appeler le serveur pour récupérer la listes des personnes
- Pour la liste vous pouvez répéter le contenus de :
  - `src/app/home/home.component.html`
    - note: il est inutile de copier le bouton “random”
  - `src/app/home/home.component.css`
- Nous verrons comment améliorer cela lors de la prochaine question

# SOLUTION

step/ngfor-solution

# Composant : In and Out

- Des annotations
    - `@Input()`
    - `@Output()`
    - ...
- ```
class FormComponent {  
    @Input() name: string;  
    @Output('personAdd') personAdd$: EventEmitter<any>;  
  
    constructor(){  
        this.personAdd$ = new EventEmitter<any>();  
    }  
}
```
- ```
<parent-form>  
  <app-form (personAdd)="addPerson($event)" [name]="formTitle"></app-form>  
</parent-form>
```

# Communication entre composants par événements

```
import { Component, Output, EventEmitter } from '@angular/core';
@Component({
  selector: "app-child"
})
export class ChildComponent {
  @Output() childEvent$: EventEmitter<string>;
  constructor() {
    this.childEvent$ = new EventEmitter<string>();
  }
  raiseEvent(){
    this.childEvent$.emit("event from child");
  }
}
```

# Exercice 9 : Réutilisation des composants

## 9

### steps/input

Créer un composant pur pour éviter la duplication

- Créer un composant **CardComponent** qui affichera les détails d'une personne
  - ng g c shared/card
    - note: Ce composant sera créé dans le répertoire src/app/shared car il sera utilisé à plusieurs endroits
- Y transférer les contenus HTML et CSS du composant **HomeComponent**
- Faire en sorte que le composant **HomeComponent** utilise **CardComponent**  
`<sfeir-card [person]="person"></sfeir-card>`
- Ajouter **CardComponent** dans les déclarations du **NgModule**

# SOLUTION

steps/input-solutio  
n

# Exercice 10 : La même chose pour les people

10

steps/input-b

- Même question que précédemment...
- Faire en sorte que le composant **PeopleComponent** utilise **CardComponent**

# SOLUTION

steps/input-b-solution



# Exercice 11 : supprimer une personne

11

## steps/output

- Ajouter un événement clic sur le bouton de suppression dans **CardComponent**
- Propager l'événement
  - l'événement devra s'appeler **personDelete**
  - astuce : utiliser `@Output()`
  - dans **PeopleComponent** supprimer l'élément de la liste
  - dans **HomeComponent** changer de card (comme le random)
- L'API à utiliser est celle-ci:
  - DELETE <http://localhost:9000/api/peoples/:id>
  - retourne une collection de personnes à jour

# SOLUTION

steps/output-solution



**Si vous appréciez la formation, Envoyez un Tweet !**

**#sfeirschool #angular #ItsJustAngular  
@sfeir @noel\_mace**

# Vers des concepts plus avancés

# Formulaires et validations

# Les formulaires: 2 problématiques

- Collecter les données
- validation des données saisies

# Les formulaires avec Angular

Template Driven Forms  
dynamic forms

# Template Driven Forms



# Template Driven Forms : FormsModule

```
import { NgModule } from '@angular/core';  
import { FormsModule } from '@angular/forms';
```

```
@NgModule({  
  imports: [  
    BrowserModule, FormsModule  
  ],  
  declarations: [ ],  
  providers: [],  
  bootstrap: []  
})  
export class AppModule { }
```

# Template Driven Forms : Syntaxe

- **#f="ngForm"**
  - déclarer une référence sur un formulaire (#f est un exemple de réf)
- **f.value**
  - JSON avec les valeurs de tous les champs de ce formulaire

```
<form #f="ngForm"  
      (ngSubmit)="onSubmit(f.value)">  
  ...  
</form>
```

# Template Driven Forms : Syntaxe

- **ngModel** : le binding d'un contrôle
- **name** : associer un contrôle au champ (obligatoire avec un **ngModel**)

- **Exemple 1** : binding View → Model

```
<input name="title" ngModel />
```

- **Exemple 2** : binding Model → View

```
<input [ngModel]="firstname" name="firstname" />
```

- **Exemple 3** : binding bi-directionnel

```
<input [(ngModel)]="postalCode" name="postalCode" />
```

# Template Driven Forms : Syntaxe

**ngModelGroup** : regrouper des contrôles dans un sous-objet (optionnel)

```
<p ngModelGroup="address">  
  <input ngModel name="city"></input>  
</p>
```

```
<p ngModelGroup="address">  
  <input ngModel name="postalCode"></input>  
</p>
```

# 12

## Exercice 12 : ajouter une personne (1/2)

### steps/form

- Petite contrainte : affichage dans une modale
- Le composant **PeopleComponent** a été complété avec deux méthodes :
  - **showDialog()** : permet d'afficher la modale
  - **hideDialog()** : permet de cacher la modale
- Le template a été complété avec :
  - un bouton pour ajouter une personne (affiche la modale d'ajout)
  - l'HTML de la modale
  - l'emplacement du formulaire d'ajout... (voir slide suivant)

# 12

## Exercice 12 : ajouter une personne (2/2)

### steps/form

- Créer un composant **FormComponent** (dans shared)
  - ng g c shared/form
  - vous trouverez dans app/static les fichiers HTML et CSS à utiliser
  - Sorties: ( *cancel* ), ( *save* )
- Mettre à jour le composant **AddDialogComponent** en y intégrant **FormComponent**
- implémenter dans le **PeopleComponent** la méthode **add** qui ajoute un contact
  - L'API à utiliser est celle-ci:
    - POST <http://localhost:9000/api/peoples>
    - retourne la personne créée

# SOLUTION

steps/form-solution

# 13

## Exercice 13 : modifier une personne (1/2)

### steps/form-b

- Créer un composant **UpdateComponent**
  - ng g c update
- Ce composant doit être accessible via l'url `#/edit/:id`
  - pensez à mettre à jour `src/app/app.routes.ts`
  - ainsi que `src/app/shared/card/card.component.html`
- Récupérer le paramètre `id` depuis la route (**ActivatedRoute**)
- Utiliser l'API `/api/peoples/:id` (GET)



# 13

## Exercice 13 : modifier une personne (2/2)

### steps/form-b

- Utiliser **FormComponent** dans **UpdateComponent** pour afficher le formulaire
  - Entrée: [ *model* ]
  - Sorties: ( *cancel* ), ( *save* )
- Mettre à jour **FormComponent** en ajoutant un “mode édition”
  - L'idée est d'utiliser le même formulaire pour l'édition et la création
    - Si **model** alors **isUpdateMode=TRUE** sinon **isUpdateMode=FALSE...**
- Utiliser le Template Driven Forms, *ngModel*
- La mise à jour se fait sur l'API /api/peoples/:id (PUT)

# SOLUTION

steps/form-b-solution

# Template Driven Forms Validation

# Les états d'un contrôle (ou groupe)

- **control.pristine** : l'utilisateur n'a pas interagi avec le contrôle
- **control.dirty** : l'utilisateur a déjà interagi avec le contrôle
- **control.valid** : le contrôle est valide
- **control.invalid** : le contrôle n'est pas valide
- **control.touched** : le contrôle a perdu le focus
- **control.untouched** : le contrôle n'a pas encore perdu le focus

# Les classes CSS

- Class name disponible pour le skin
  - .ng-valid / .ng-invalid
  - .ng-pristine / .ng-dirty
  - .ng-touched / .ng-untouched

# La gestion des erreurs

- Pour un **groupe de contrôles** ou un **contrôle**
  - `control.valid`, `control.invalid ...`
  - `control.errors`
- Par exemple
  - `f.valid`
  - `f.errors.minlength`
  - Astuce : `f.errors?.minlength`

# Exemple avec gestion des erreurs

```
<input name="user" ngModel #userRef="ngModel" required>  
  
<div [hidden]="!userRef.errors?.required">  
  <span class="help-block">Ce champ est obligatoire</span>  
</div>
```

# 14

## Exercice 14 : valider votre formulaire

### steps/form-validation

- Valider les champs
  - Firstname : **required** + min 2 lettres
  - Lastname : **required** + min 2 lettres
  - Email : **required**
  - Phone: **required** et 10 digits  $\Rightarrow \backslash d\{10\}$
- Afficher des messages en fonction des erreurs
- Utiliser `[class.errors]="control.errors"` pour appliquer une classe CSS "errors"
  - par ex: `.errors{color:red;}`



# SOLUTION

steps/form-validation-solution

# dynamic forms

# dynamic forms : ReactiveFormsModule

```
import { NgModule } from '@angular/core';
import { ReactiveFormsModule } from '@angular/forms';

@NgModule({
  imports: [
    ..., ReactiveFormsModule
  ],
  declarations: [ ],
  providers: [],
  bootstrap: []
})
export class AppModule { }
```

# dynamic forms : Template

- Référence au modèle de formulaire via **formGroup**
- Mapping de controls via **formControlName**

```
<form [formGroup]="editForm">  
  <input type="text" formControlName="firstname">  
  
  <div [hidden]="!editForm.controls.firstname.valid">...</div>  
  <button type="submit" [disabled]="!editForm.valid">  
    Modifier  
  </button>  
</form>
```

# dynamic forms : Syntaxe

- **[formGroup]="editForm"**
  - déclarer une référence sur un modèle de formulaire "editForm"

```
<form [formGroup]="editForm">
```

```
...
```

```
</form>
```

# dynamic forms : Syntaxe

- **formControlName** : le binding d'un contrôle

```
<form [formGroup]="editForm">
```

```
  <input formControlName="id">
```

```
  <input formControlName="firstname">
```

```
  <div formGroupName="address">...</div>
```

```
</form>
```

# Dans la classe

```
import { Validators, FormControl, FormGroup } from '@angular/forms';

@Component({...})
export class FormComponent {
  editForm: FormGroup;

  constructor() {
    this.editForm = new FormGroup({
      id: new FormControl(''),
      firstname: new FormControl('', Validators.compose([
        Validators.required, Validators.minLength(2)
      ])),
      ...
    });
  }
}
```

# Exemple avec gestion des erreurs

```
<form [formGroup]="editForm">
  <div>
    <input formControlName="firstname">

    <div [hidden]="!editForm.controls.firstname.errors?.required">
      <span class="help-block">Ce champ est obligatoire</span>
    </div>

  </div>
</form>
```



# 15

## Exercice 15 : valider votre formulaire

### steps/form-model-driven

- Transformer **FormComponent** en mode **Model-Driven**
- Tenez compte du mode “création” aussi
- Astuces :
  - **Validators.pattern('\\d{10}')**
  - **Validators.minLength(2)**

# SOLUTION

steps/form-model-driven-solution



# **dynamic forms**

## **Validation personnalisée**

# Créer ses propres validateurs

- Créer sa fonction de validation
  - Si OK : retourne **NULL**
  - Si NOK: retourne un objet sous la forme
    - { **nomErreur: true** }

```
function CustomEmailValidator (c: FormControl) {  
    return (c.value.indexOf('@') !== -1)  
        ? null :  
        {email: true};  
}
```

# Utilisation du validateur

- Lors de la création du control

```
this.formModel = new FormGroup({  
  email: new FormControl('', CustomEmailValidator),  
  // or  
  
  email: new FormControl('', Validators.compose([  
    Validators.required, CustomEmailValidator  
  ]))  
});
```

# 16

## Exercice 16 : votre validateur

### steps/form-custom-validator

- Générer une classe **CustomValidators** dans **shared/form**
  - `ng g class shared/form/CustomValidators`
- Créer un validateur (méthode statique) qui vérifie l'adresse e-mail
  - respectant le format "**nom.p@sfeir.com**"
- Associer ce validateur au control "mail"
- Afficher dans le HTML le message correspondant au type d'erreur
- `const regex = /^\\w+\\.\\w@sfeir\\.com$/;`

# SOLUTION

steps/form-custom-validator-solution

# Template Driven Model Driven Rappel



# Template-Driven vs Model-Driven

- La classe expose le modèle de données

```
class MyComponent {  
  model: any;
```

- La classe expose le modèle du formulaire

```
class MyComponent {  
  formModel: FormGroup;
```

# Template-Driven vs Model-Driven

- La classe expose le modèle de données
- Binding et validation se font dans la vue
- La classe expose le modèle du formulaire
- Binding et validation se font dans la classe

```
<input type="text"  
  name="name"  
  ngModel  
  required>
```

```
class MyComponent {  
  formModel: FormGroup;  
  constructor() {  
    this.formModel = new FormGroup({  
      name: new FormControl('', Validators.required)  
    });  
    //...
```

# Template-Driven vs Model-Driven

- La classe expose le modèle de données
- Binding et validation se font dans la vue
- la vue contient le data binding

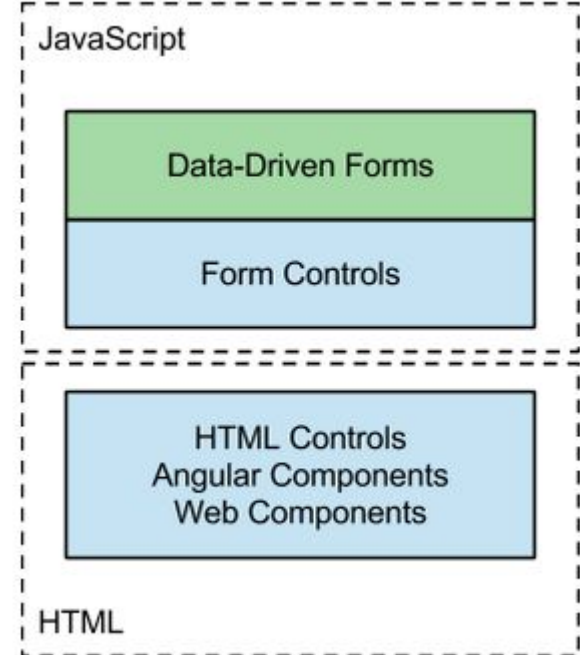
```
<input type="text"  
      name="name"  
      ngModel  
      required>
```

- La classe expose le modèle du formulaire
- Binding et validation se font dans la classe
- La vue contient le mapping

```
<input type="text"  
      formControlName="name">
```

# Avantages du Model-Driven

- La logique est dans le code et non dans le template
- Plus facile à tester
- Prêt pour de futurs scénarios (Data-Driven-Form)



# Créer vos propres services

# Un service Angular

- Une simple classe exportée
- Un décorateur @Injectable()

```
@Injectable({  
  provideIn: AdminModule  
})  
export class NameService {  
  
  constructor() {  
    this.name = 'Hello';  
  }  
  
  getName() {  
    return this.name;  
  }  
}
```

# Injection globale : @NgModule()

```
class NameService {  
  constructor() {  
    this.name = 'Hello';  
  }  
  getName() {  
    return this.name;  
  }  
}
```

```
import { NameService } from './shared/';  
import { AppComponent } from './shared/';  
  
@NgModule({  
  declarations: [AppComponent],  
  providers: [NameService],  
  ...  
})  
class AppModule {}
```

```
@Component() class AppComponent {  
  constructor(nameService: NameService) {  
    this.name = nameService.getName();  
  }  
}
```

# Injection locale : @Component()

```
class NameService {  
    constructor() {  
        this.name = 'Hello';  
    }  
    getName() {  
        return this.name;  
    }  
}
```

```
import { NameService } from './shared/';  
  
@Component({  
    providers: [NameService]  
})  
class AppComponent {  
    constructor(public nameService: NameService) {  
        this.name = nameService.getName();  
    }  
}
```



# L'injection de dépendances (DI)

# Principe

Pour une classe, il y a 3 façons de gérer une dépendance

1. L'instancier (new)
2. La récupérer de façon définie (variable globale, singleton)
3. **Se la faire fournir**

```
class Car {  
  
    constructor() {  
        this.engine = new Engine();  
        this.tires = Tires.getInstance();  
        this.doors = app.get('doors');  
    }  
}
```

# Exemple en Angular (avec TypeScript)

```
import { EngineService, TiresService, DoorsService } from './shared';
```

```
@Component({...})
```

```
class Car {
```

```
    constructor(
```

```
        public engine: EngineService,
```

```
        public tires: TiresService,
```

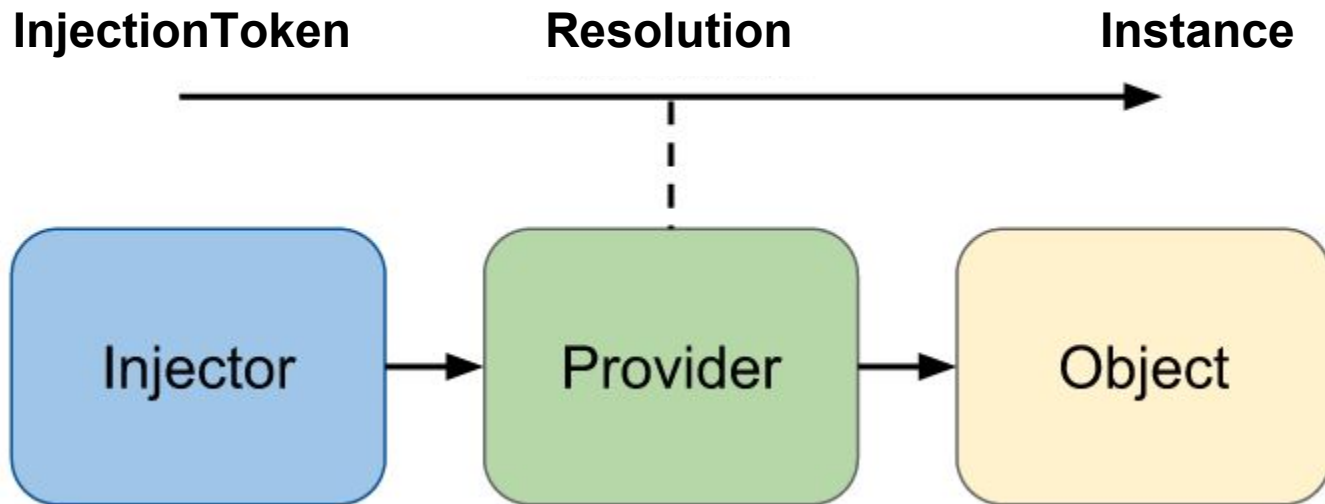
```
        public doors: DoorsService
```

```
    ) {}
```

```
}
```

# Principe de la DI en Angular

- L'**Injector** expose l'API pour **créer** des instances de dépendances
- Le **Provider** indique à l'**Injector** comment créer la dépendance
- La dépendance est le **type** d'objet à créer



# Le rôle du Provider

- Fait le lien entre un **InjectionToken** (token) et une **Factory**
- Permet de découpler la dépendance et son implémentation
- API pour:
  - lier à une simple valeur :
    - faire des alias de token
    - créer des factory synchrones ou pas ( toFactory, toAsyncFactory)

# Différents types de résolutions

- Valeur
- Classe alternative
- Classe aliasée
- Factory

## Par valeur : useValue

```
providers: [ {provide: V8, useValue: 8} ]
```

```
providers: [ {provide: V8, useValue: 'V8'} ]
```

```
providers: [ {provide: V8, useValue: false} ]
```

```
providers: [ {provide: V8, useValue: { cylinder: 8 } } ]
```

# Classe alternative : useClass

```
providers: [ {provide: V8, useClass: V8} ]
```

```
providers: [V8]
```

```
providers: [ {provide: V8, useClass: V8Mock} ]
```



# Classe aliasée : useExisting

```
providers: [  
  V8, {provide: V8Engine, useClass: V8}  
]
```

NOTE : Création de 2 instances de V8 !!

```
providers: [  
  V8,  
  {provide: V8Engine, useExisting: V8}  
]
```

NOTE : Réutilisation de l'instance V8

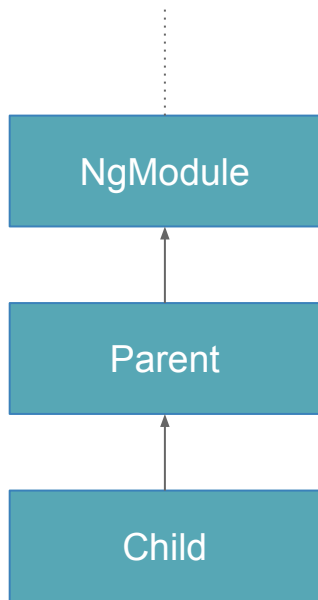
# Factory : useFactory

```
export const function createEngineFactory(dep) {  
  return new Engine(dep.cylinders);  
}
```

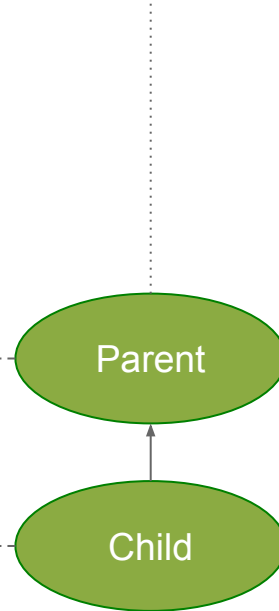
```
providers: [  
  V8Engine,  
  {  
    provide: Engine,  
    useFactory: createEngineFactory,  
    deps: [V8Engine]  
  }  
]
```

# Injection hiérarchique

Injection Tree



Component Tree



# Singleton ou pas...

- Par défaut
- Pour avoir des instances
  - Utiliser une **Factory**
  - Ou créer un **Provider** fils

# 17

## Exercice 17 : Créez votre propre service

### steps/service

Actuellement, nous utilisons l'API **Http** dans les composants et nous dupliquons à chaque fois l'URL de l'API.

Améliorons ceci en créant notre propre service pour nous connecter au back-end pour les opérations de CRUD sur les personnes.

- Créer un service **PeopleService** dans le répertoire `src/app/shared/people-service/`
  - Dans le composant, au lieu d'appeler `http.get("/api/peoples")`
  - On voudrait appeler `peopleService.fetch().subscribe(...)`;
- De manière analogue, idem pour les autres opérations CRUD
- Pensez à ajouter ce service dans les providers du **@NgModule**

# SOLUTION

steps/service-solution

# Transformer les données avant de les afficher avec les Pipes

# Syntaxe

- A la suite d'une expression

```
{{ expression | filter1 }}
```

- On peut les chaîner

```
{{ expression | filter1 | filter2 }}
```

- On peut leur passer des paramètres

```
{{ expression | filter1:param1:param2 }}
```



# Pipes existants

- currency
- date
- lowercase
- uppercase
- async
- json
- decimal
- percent

# Examples

hello

```
{{ "hello" | uppercase }}
```

HELLO

# Exemples

1368730200

`{{ "1368730200" | date }}`

Jeudi 16 mai 2013 20H50

# Examples

1234.562342

`{{ "1234.562342" | currency }}`

1 234,56 €

# Syntaxe

```
this.amount = 1234.56;
```

```
<!-- 1234.56 -->
```

```
<div>{{amount}}</div>
```

```
<!-- USD1,234.56 -->
```

```
<div>{{amount | currency}}</div>
```

```
<!-- USD$1,234.56 -->
```

```
<div>{{amount | currency:"USD$"}}</div>
```

```
<!-- €1,234.56 -->
```

```
<div>{{amount | currency:"EUR":true}}</div>
```

# Pipe date

- Formate une date selon un certain format et selon une locale

```
<div>{{uneDate | date:format}}</div>
```

- Ce filtre accepte un format (string) en argument.

Doc : [angular.io/docs/ts/latest/api/common/DatePipe.html](https://angular.io/docs/ts/latest/api/common/DatePipe.html)

# Exercice 18 : Améliorer la lisibilité des données

18

steps/pipe

- Une date de naissance a été ajoutée
- Afficher cette date sous le format : **dd/MM/yyyy**

# SOLUTION

steps/pipe-solution



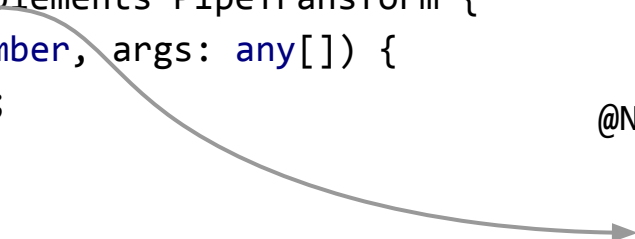
# Créer ses propres Pipes

# Créer ses propres pipes

- importer le **Pipe** decorator
- Un pipe est une classe décorée avec **@Pipe()**
- **@Pipe()** définit une propriété name qui sera utilisé dans les template
- La classe implémente une méthode transform qui prend en paramètre une valeur et éventuellement un tableau d'arguments (string)
- retourne la nouvelle valeur

# Créer ses propres pipes

```
import {Pipe} from '@angular/core';
@Pipe({
  name: 'mypipe'
})
export class MyPipe implements PipeTransform {
  transform(value: number, args: any[]) {
    return newValue;
  }
}
```



```
@NgModule({
  ..
  declarations: [MyPipe],
  ..
})
```

# 19

## Exercice 19 : Je veux des N/A

### steps/pipe-b

- Créer un **NaPipe** avec le CLI dans le répertoire shared
  - ng g p shared/na-pipe/na
- Afficher "**N/A**" s'il n'y a pas de manager ou d'entité associés

# SOLUTION

steps/pipe-b-solution

# Les directives

# Directive : \*NgFor (rappel)

- Itère dans une collection et génère un template par élément
- ***index, odd, even, last*** à utiliser en alias dans des variables

```
<ul>  
  <li *ngFor="let fruit of fruits; let i=index">  
    <!-- répétition du template li par élément fruit -->  
    {{ i }} : {{ fruit.name }}  
  </li>  
</ul>
```

# Directive : \*NgSwitch

- Change la structure du DOM de manière conditionnel

```
<div [ngSwitch]="display">
```

```
  <p *ngSwitchCase="list">...</p>
```

```
  <p *ngSwitchCase="whenExpression1">...</p>
```

```
  <p *ngSwitchDefault>...</p>
```

```
</div>
```



# Directive : \*NgIf

- Change la structure du DOM de manière conditionnel

```
<div *ngIf="errorCount > 0" class="error">  
  {{ errorCount }} errors detected  
</div>
```

# C'est quoi ces étoiles ?

- pour NgFor, NgIf et NgSwitch
- sucre syntaxique
- indique que l'on utilise un `<ng-template>`
- Attention à ne pas oublier les `[]` si vous utilisez les *templates*

```
<div *ngIf="errorCount > 0"></div>
```

```
//SAME AS
```

```
<ng-template [ngIf]="errorCount > 0">  
  <div></div>  
</ng-template>
```

# 20

## Exercice 20: une liste plus compacte

### steps/directive-use

- Ajouts :
  - Un bouton ainsi qu'une nouvelle liste viennent d'être ajoutés dans **PeopleComponent** (la vue HTML)
- Utiliser la directive **NgSwitch** pour changer le mode d'affichage : "card", "list"
- Modifier les icônes à l'aide d'un **NgIf**

# SOLUTION

steps/directive-use-solution

# Créer vos propres directives

# Rappel...

- Le **composant** est une **directive** avec une vue
- La **directive** est une classe sans vue



# 3 types de directives

- **structurelle** : qui modifie le layout en ajoutant, supprimant ou remplaçant des éléments du DOM
  - NgIf, NgFor, NgSwitch
- **attribute** : qui altère l'apparence ou le comportement d'un élément
  - ex : NgStyle, NgClass
- **composant** : qui est juste une directive avec un template



# Définition

```
import { Directive } from '@angular/core';  
  
@Directive({ ... })  
export class MyDirective {}
```



# Plusieurs types d'invocations possibles

## juste un sélecteur CSS3

- **element-name**: pour restreindre à un élément
- **[attribute]**: pour restreindre à un attribut
- **.class**: pour restreindre à une classe
- **[attribute=value]**: restreint à un attribut avec une certaine valeur
- **:not(sub\_selector)**: si l'élément ne match pas le sous-sélecteur

```
import {Directive} from '@angular/core';

@Directive({
  selector: '[foobar]',
})
export class MyDirective {}
```

# Properties (rappel)

- listées dans les inputs ou grâce à l'annotation @Input()
- peuvent être aliasées

- **Comme pour les Composants**

```
import {Directive, Input} from '@angular/core';  
@Directive({  
  selector: '[foobar]'  
})  
export class MyDirective {  
  
  myProp: string;  
  @Input('alias') myProp2: string;  
  
}
```

# Elements DOM

- **this.element** référence directe de l'élément DOM
- **this.renderer** pour interagir avec le DOM

```
import {  
    Directive, ElementRef, Renderer2  
} from '@angular/core;  
  
@Directive({  
    selector: '[foobar]'  
})  
export class MyDirective {  
    constructor(  
        element: ElementRef,  
        renderer: Renderer2  
    ) {}  
}
```

# Interaction avec le DOM

- Préférez l'utilisation du **Renderer** au lieu du **ElementRef**
- Pas de dépendance direct au DOM
- Permet d'exécuter l'application dans d'autres environnements

# Interaction avec le DOM

X

```
document.querySelector('#someId').innerHTML = 'X';
```

:/

```
this.element.nativeElement.style.color = 'orange';
```

```
this.element.nativeElement.innerHTML = ':/';
```

:)

```
this.renderer.setStyle(this.element.nativeElement, 'color', '#0f0');
```

```
this.renderer.setProperty(  
    this.element.nativeElement, 'innerHTML', ':)'  
);
```

# Évènements (rappel)

- **@HostListener()** pour écouter des évènements sur l'élément host
- **@Output()** pour propager des évènements
- Comme pour les Composants

```
@Directive({})
export class MyDirective {

  @Output() somethingChange$: EventEmitter<any>;

  constructor(){
    this.somethingChange$ = new EventEmitter();
  }

  @HostListener('click', '$event')
  onClick($event) {
    this.somethingChange$.emit({$event, data});
  }
}
```

# 21

## Exercice 21: manipuler le DOM

### steps/directive-create

- Créer une directive **SfeirBadge** dans le répertoire `shared/badge`
  - `ng g d shared/badge/badge`
- Cette directive affiche une icône si une personne est un manager
  - `<i class="material-icons">supervisor_account</i>`
- Elle doit pouvoir s'utiliser comme ceci dans la **vue liste** du template du **PeopleComponent**:
  - `<span sfeirBadge [person]="person"></span>`

# SOLUTION

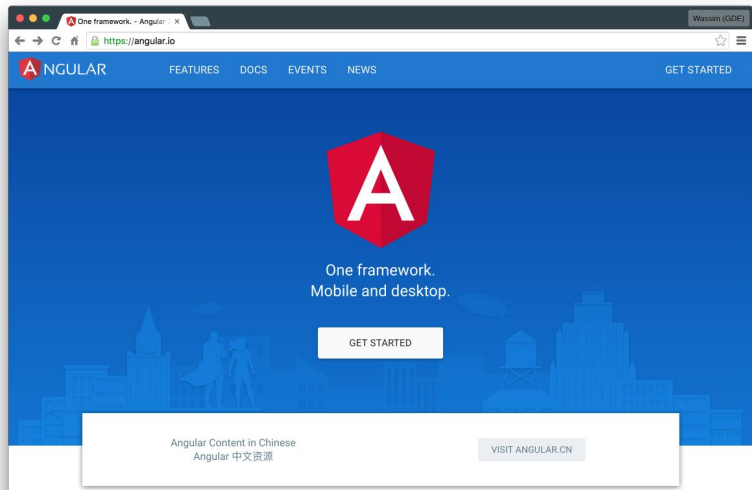
steps/directive-create-solution



# Ressources

# Approfondissements

- Site Officiel : [angular.io](https://angular.io)
- Angular Blog: [blog.angular.io](https://blog.angular.io)
- [angularindepth.com](https://angularindepth.com): where advanced Angular concepts are explained
- [hirez.io](https://hirez.io): Professional and entertaining Angular courses (en particulier pour le TDD)
- <http://rxmarbles.com/>



# Outils

- Ngxs: [ngxs.gitbooks.io/ngxs](https://ngxs.gitbooks.io/ngxs)
- NgRx: [github.com/ngrx/platform](https://github.com/ngrx/platform)
- Ng-bootstrap: [ng-bootstrap.github.io](https://ng-bootstrap.github.io)
- Ngx-Build-Plus: [github.com/manfredsteyer/ngx-build-plus](https://github.com/manfredsteyer/ngx-build-plus)
- Augury: [augury.rangle.io](https://augury.rangle.io)
- Storybook: [storybook.js.org](https://storybook.js.org)