



Bienvenue à la Sfeir School
ANGULAR 200

**WIFI : SFEIRGUEST
MDP : GUEST\$2014**



Présentation



[sf≡ir]

Cyril Balit

CTO front
@cbalit



Présentation



[sf≡ir]

Noël Macé

Developer Advocate
@noel_mace



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-ng-200>

(accès restreint)

Déroulement de la formation

Github de la formation

<https://github.com/Sfeir/angular2-200>

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

Déroulement de la formation

```
$ git clone -b step-01  
https://github.com/Sfeir/angular2-200.git
```

Déroulement de la formation

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

```
$ npm run client  
$ npm run server
```

npm run client: <http://localhost:4200/>
npm run server: <http://localhost:9000/>

Déroulement de la formation

Un concept clé d'Angular

Un TP

- une branche d'exercice : step-00
- une branche de solution : step-00-solution

Angular next

2

3

1

Major

*Breaking
change*

Minor

*New features,
not breaking*

Patch

*Bugfixes,
not breaking*

Quickstart

La stack “officielle”

Angular CLI

<https://github.com/angular/angular-cli>

```
1. wchegham@wchegham-mbp-2: ~/Sandbox/dev/oss (zsh)
wchegham 192.168.1.16 ➜ ~/Sandbox/dev/oss
$ ng
Usage: ng <command> (Default: help)

Available commands in angular-cli:

ng addon <addon-name> <options...>
  Generates a new folder structure for building an addon, complete with test harness.
  --dry-run (Boolean) (Default: false)
    aliases: -d
  --verbose (Boolean) (Default: false)
    aliases: -v
  --blueprint (String) (Default: addon)
    aliases: -b <value>
  --skip-npm (Boolean) (Default: false)
    aliases: -sn
  --skip-bower (Boolean) (Default: false)
    aliases: -sb
  --skip-git (Boolean) (Default: false)
    aliases: -sg

ng build <options...>
  Builds your app and places it into the output path (dist/ by default).
  aliases: b
```

Configuration: .angular-cli.json

```
{  
  "project": {  
    "version": "1.0.0-beta.20-4",  
    "name": "angular2-200"  
  },  
  "apps": [  
    {  
      "root": "src",  
      "outDir": "dist",  
      "assets": "assets",  
      "index": "index.html",  
      "main": "main.ts",  
      "test": "test.ts",  
      "tsconfig": "tsconfig.json",  
      "prefix": "sfeir",  
      "mobile": false,  
      "styles": [  
        "styles.css"  
      ],  
      "scripts": []  
    },  
    {  
      "environments": {  
        "source": "environments/environment.ts",  
        "dev": "environments/environment.ts",  
        "prod": "environments/environment.prod.ts"  
      }  
    },  
    {  
      "addons": [],  
      "packages": [],  
      "e2e": {  
        "protractor": {  
          "config": "./protractor.conf.js"  
        }  
      },  
      "test": {  
        "karma": {  
          "config": "./karma.conf.js"  
        }  
      },  
      "defaults": {  
        "styleExt": "css",  
        "prefixInterfaces": false  
      }  
    }  
  ]  
}
```

La stack “officielle”

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

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

La stack “officielle”

```
$ 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

La stack “officielle”

```
$ 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
```

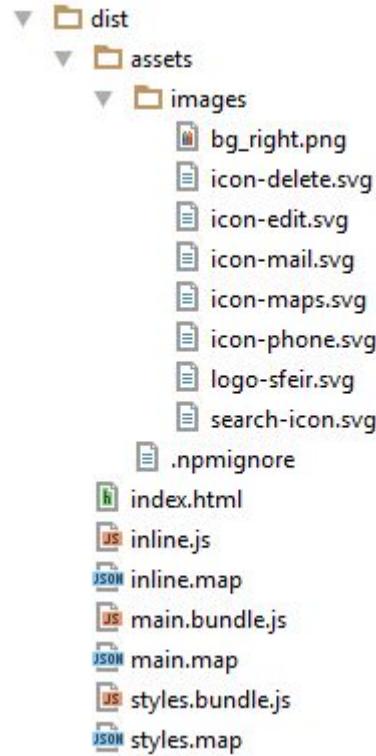
- génère les fichiers d'un service
 - src/app/**shared**/user.service.spec.ts
 - src/app/**shared**/user.service.ts

La stack “officielle”

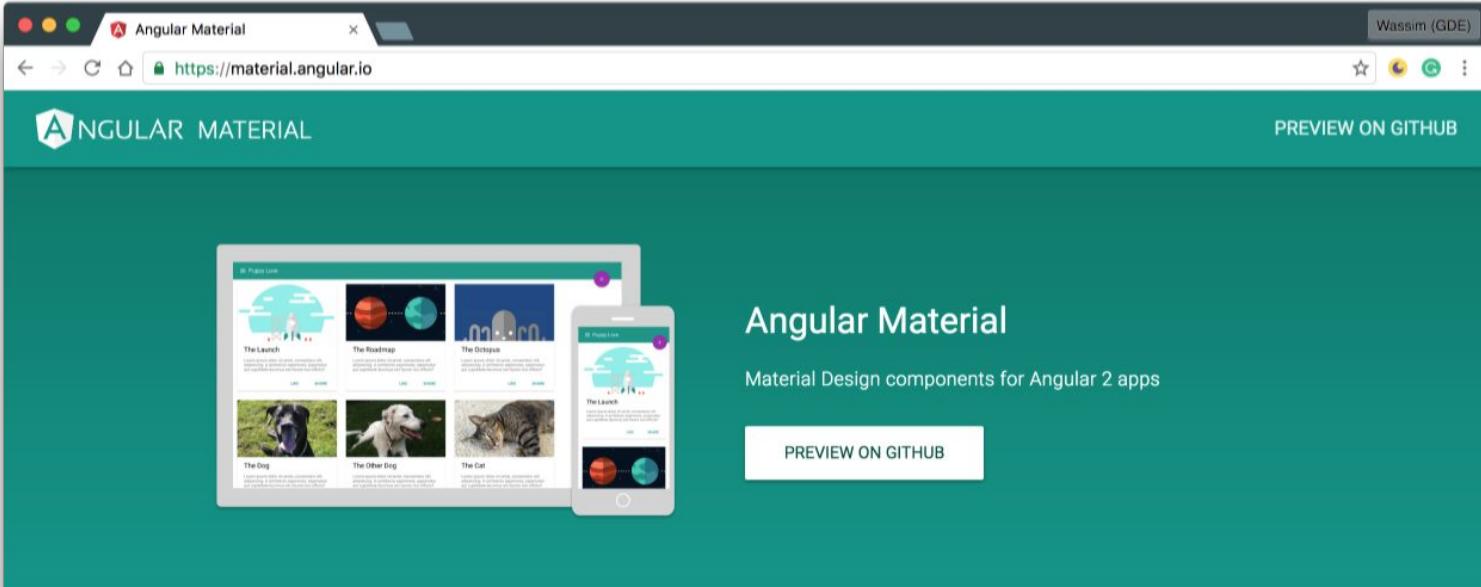
- \$ ng g directive
- \$ ng g pipe
- \$ ng g class
- \$ ng g interface
- \$ ng g enum
- ...
- ng build --prod
- ng lint
- ng help
- ...

Webpack

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



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**

- ES6+
- types (optionnels)
- annotations

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

- Javascript

- ES6
- ES5

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

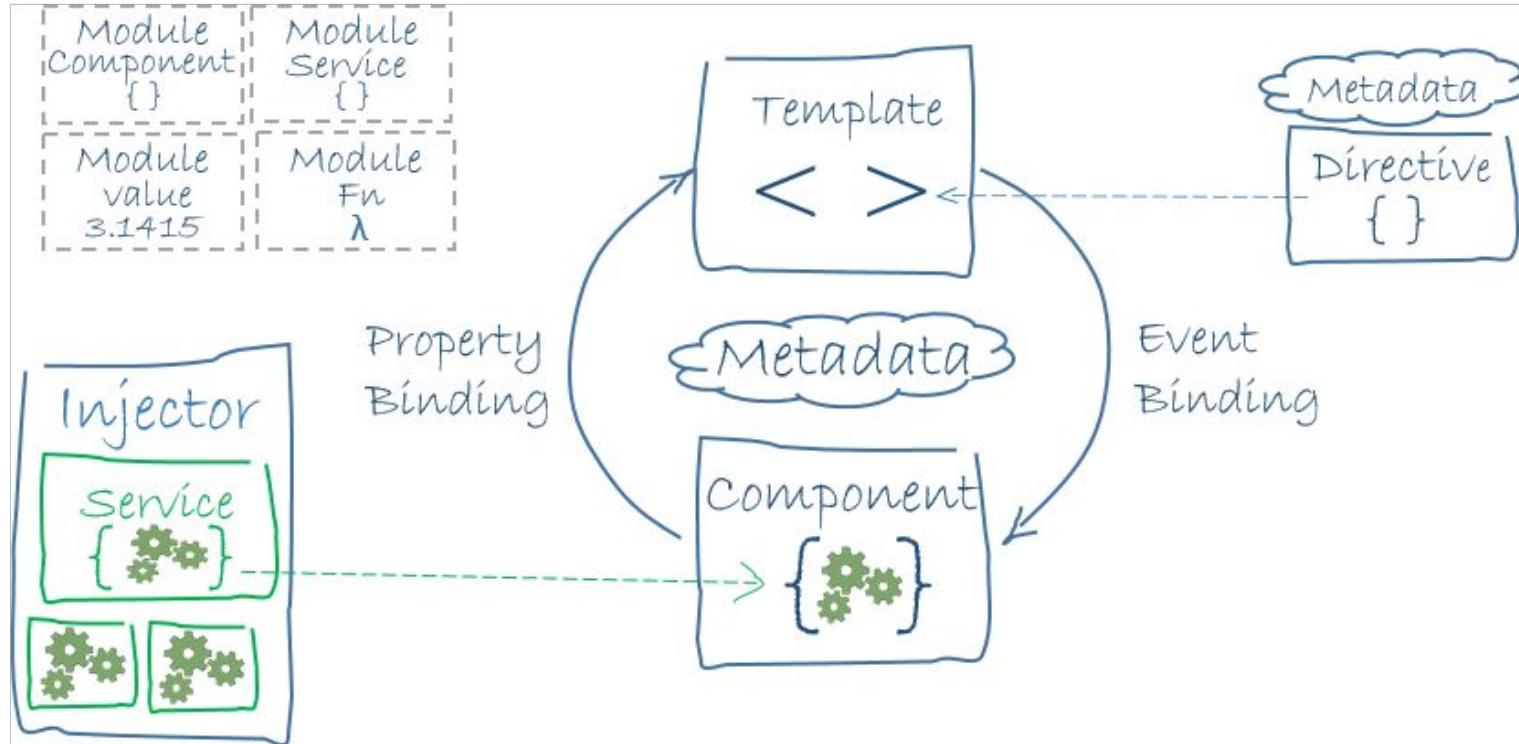
- Dart

index.html

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

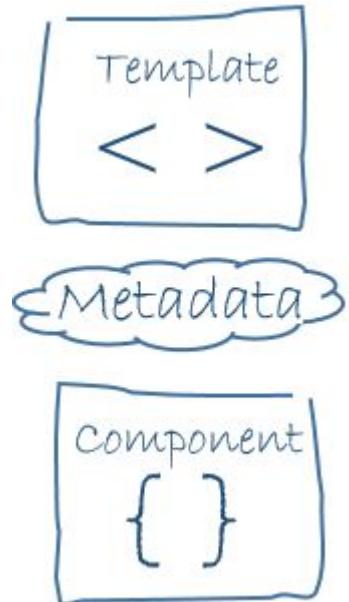
Architecture Globale

Architecture globale



Votre application : un composant

- 3 concepts de base



Un composant : annotation + classe

- La logique du composant: utilise la syntaxe de classe de ES6

```
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);
```

1

Exercice 1 : prise en main



git checkout -f step-01

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 }}
- Mettre à jour le fichier src/app/index.ts avec les bons exports
- Compléter le fichier index.html: src/index.html

SOLUTION

git checkout -f

step-01-solution

2

Exercice 2 : Utilisez le CLI



git checkout -f step-02

Mise en place de votre premier composant avec le CLI

- Créer un composant **HomeComponent** avec le CLI
 - `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

git checkout -f

step-02-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	<code>{{ expression }}</code> <code>[targetFooBar] = "expression"</code> <code>bindTargetFooBar = "expression"</code>	Interpolation Propriétés Classe Attribut Style
Unidirectionnel depuis la vue vers le model	<code>(targetFooBar) = "expression"</code> <code>onTargetFooBar = "expression"</code>	Evénements
bidirectionnel	<code>[(targetFooBar)] = "expression"</code> <code>bindonTargetFooBar = "expression"</code>	bidirectionnel

Mais avant : attributs vs propriétés

- Les attributs c'est du **HMTL**, 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></code> <code><my-component [data]="currentData"></my-component></code> <code><div [ngClass]="{{selected: isSelected}}></div></code>

- Forme canonique: bind**CapitalAttr**
- 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><button (click)="onSave()"></button></code> <code><hero-detail (deleted)="onDeleted(\$event)">...</code> <code><input (change)="firstName = \$event" /></code>

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

2 way binding

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

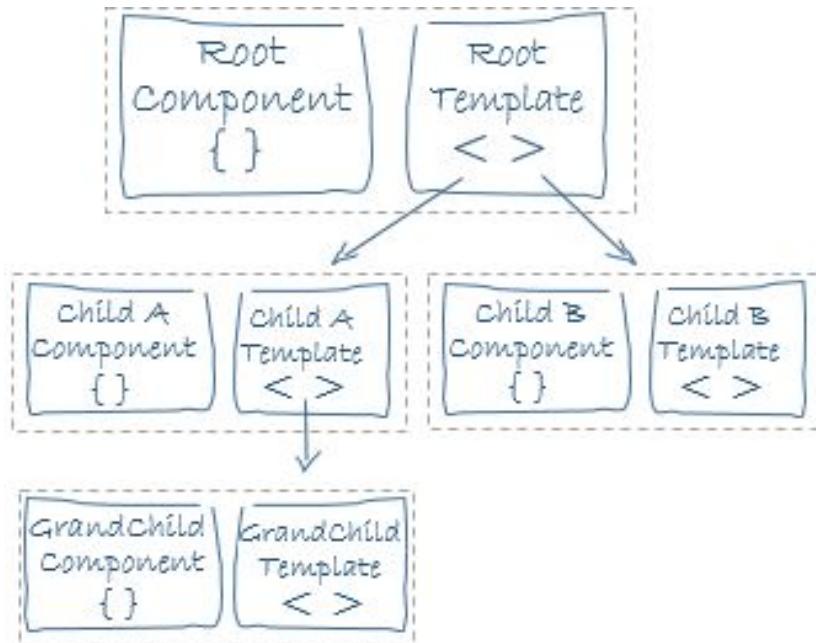
Composants

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

`git checkout -f step-03`

- Faites en sorte que le composant **AppComponent** 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

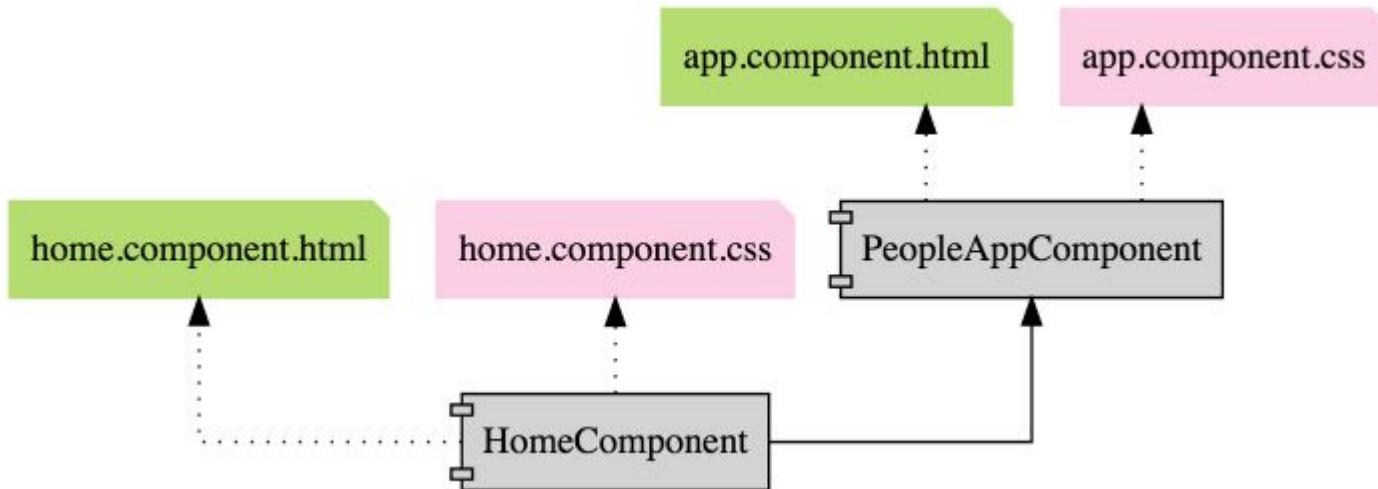
git checkout -f

step-03-solution

3

Exercice 3 : Imbriquer les composants

git checkout -f step-03



4

Exercice 4 : Afficher une personne

git checkout -f step-04

- 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

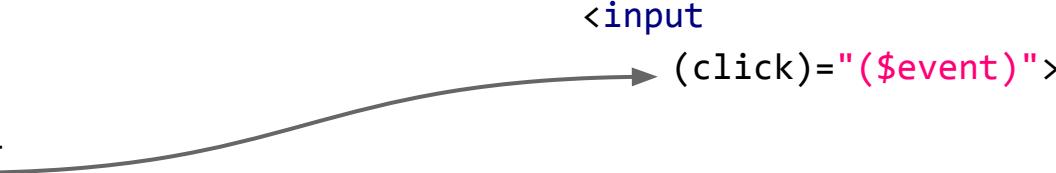
The screenshot shows a web browser window titled "Angular2 People 200". The address bar displays "localhost:4200/#/home". The page has a blue header with the word "people" on the left and "Maps" and "List" buttons on the right. Below the header, there is a card for a person named "Vargas Shaffer" from "Sfeir-Luxembourg". The card includes an email icon and the address "vargasshaffer@sfeir.com", a phone icon and the number "+33197542822", and a location section stating "Manager : Bruno" and "Location : SFEIR". To the right of the card is a circular profile picture of a man. At the bottom of the card are three icons: a person with a location pin, a pen, and a trash can.

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'event : **\$event**

```
export class MyComponent {  
  values: string = '';  
  
  constructor(){}
  
  
  updateValue(event){  
    this.values += event.target.value + ' | ';  
  }
}
```



The diagram illustrates the flow of an event from the component's method to the DOM. A curved arrow originates from the 'updateValue(event)' call in the code and points to the '(click)=\"\$event\"' attribute of the input element.

5

Exercice 5 : gérer un clic

`git checkout -f step-05`

- 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

git checkout -f

step-05-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

git checkout -f step-06

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

git checkout -f

step-06-solution

Navigation

Angular2 People 200 Wassim (GDE)

localhost:4200/#/home

pe@ople

Maps List

Do you know?

Mercedes Hebert
QUINTITY

Mercedes.Hebert@QUINTITY...
 0125878522

Manager : McLaughlin
Location : SFEIR

Angular2 People 200 Wassim (GDE)

localhost:4200/#/people/5763cd4d979b62a209809160

pe@ople Maps List

Fiche personnelle

Mercedes Hebert
QUINTITY

 [Mercedes.Hebert@QUINTITY...](mailto:Mercedes.Hebert@QUINTITY.COM)

 [0125878522](tel:0125878522)

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

Skills

ex commodo pariatur sit aute



Angular2 People 200 Wassim (GDE)

localhost:4200/#/people

pe@ople

Maps List

Found 10 people

Leanne Woodard
BIOSPAN

 [Email](#) [Leanne.Woodard@BIOSPA...](#) [Phone](#) [0784112248](#)

Manager : Erika
Location : SFEIR

[Edit](#) [Delete](#)

Castaneda Salinas
METROZ

 [Email](#) [Castaneda.Salinas@METR...](#) [Phone](#) [0145652522](#)

Manager : Erika
Location : SFEIR

[Edit](#) [Delete](#)

Phyllis Donovan
PEARLESSA

 [Email](#) [Phyllis.Donovan@PEARLES...](#) [Phone](#) [0685230125](#)

Erika Guzman
CIRCUM

 [Email](#) [Erika.Guzman@CIRCUM.com](#) [Phone](#) [0678412587](#)

82

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

School

Angular2 People 200

localhost:4200/#/edit/5763cd4d979b62a209809160

Maps List

Update Mercedes Hebert

ID (disabled)
5763cd4d979b62a209809160

First name *
Mercedes

Last Name *
Hebert

Email *
Mercedes.Hebert@QUINTITY.com

Address
Laurel Avenue

City
Northchase

Postal Code
85752

Phone * (ex: 0612345678)
0125878522

Manager

Cancel Save



Wassim (GDE)

A screenshot of a web application titled "Angular2 People 200". The main view is a map of Paris and the surrounding Seine-Saint-Denis and Val-de-Marne departments. The map shows major roads, including the périphérique, and various arrondissements of Paris. Several red location markers are placed on the map, primarily in the 1st, 7th, 8th, and 15th arrondissements of Paris. A modal window is open, featuring a portrait photo of a woman with long brown hair and the name "Mercedes Hebert" below it. The top navigation bar includes the title "Angular2 People 200" and the URL "localhost:4200/#/locator". The bottom right corner of the map area contains a small note: "Map data ©2016 Google Terms of Use Report a map error".

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

Exemple simple

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

Exemple : définitions des routes

```
//app.routes.ts
import { 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/edit/:id', component: PersonComponent},
];
export const AppRoutes = RouterModule.forRoot(ROUTES);
```

Exemple : Utilisation des routes

```
//app.module.ts
import { RouterModule } from '@angular/router';
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

git checkout -f step-07

- 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

git checkout -f

step-07-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

git checkout -f step-08

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

git checkout -f

step-08-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

git checkout -f step-09

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

git checkout -f

step-09-solution

Exercice 10 : La même chose pour les people

10

`git checkout -f step-10`

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

SOLUTION

git checkout -f

step-10-solution

Exercice 11 : supprimer une personne

11

git checkout -f step-11

- 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

git checkout -f

step-11-solution



Si vous appréciez la formation, Envoyez un Tweet !

#sfeirschool #angular #ItsJustAngular
@sfeir @cbalit

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
Model Driven 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)

git checkout -f step-12

- 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)

git checkout -f step-12

- 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/people>
 - retourne la personne créée
- Pensez à ajouter **FormComponent** dans les déclarations du module

SOLUTION

git checkout -f

step-12-solution

13

Exercice 13 : modifier une personne (1/2)

git checkout -f step-13

- 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)

git checkout -f step-13

- 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/people/:id (PUT)

SOLUTION

git checkout -f

step-13-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>
```

Exercice 14 : valider votre formulaire

git checkout -f step-14

- Valider les champs
 - Firstname : **required + min 2 lettres**
 - Lastname : **required + min 2 lettres**
 - Email : **required**
 - Phone: **required** et 10 digits ⇒ **\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

git checkout -f

step-14-solution

Model Driven Forms

Model Driven Forms : ReactiveFormsModule

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

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

Model Driven Forms : Template

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

```
<form [formGroup]="editForm">
  <input type="text" formControlName="name">

  <div [hidden]="!editForm.controls.name.valid">...</div>
  <button type="submit" [disabled]="!editForm.valid">
    Modifier
  </button>
</form>
```

Model Driven Forms : Syntaxe

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

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

Model Driven 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>
```

Exercice 15 : valider votre formulaire

git checkout -f step-15

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

SOLUTION

git checkout -f

step-15-solution

Model Driven 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

`git checkout -f step-16`

- 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

SOLUTION

git checkout -f

step-16-solution

Template Driven Model Driven Rappel

Template-Driven vs Model-Driven

- La classe expose le modèle de données
- La classe expose le modèle du formulaire

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

```
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

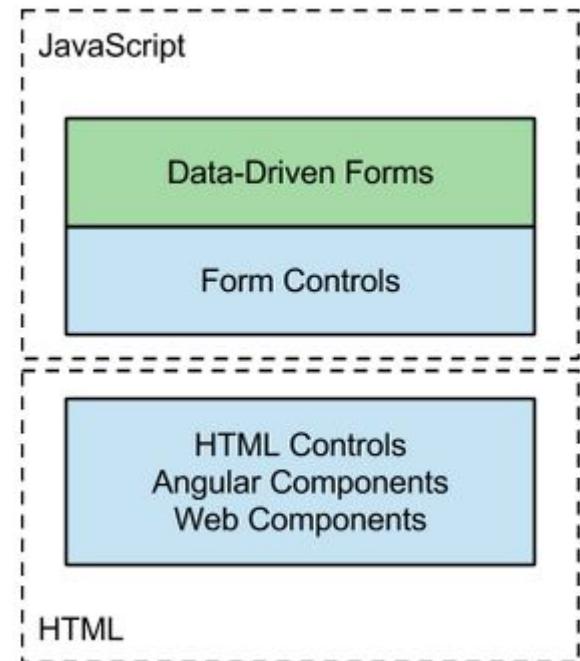
- La classe expose le modèle de données
 - Binding et validation se font dans la vue
 - la vue contient le data binding
- La classe expose le modèle du formulaire
 - Binding et validation se font dans la classe
 - La vue contient le mapping

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

```
<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()  
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';

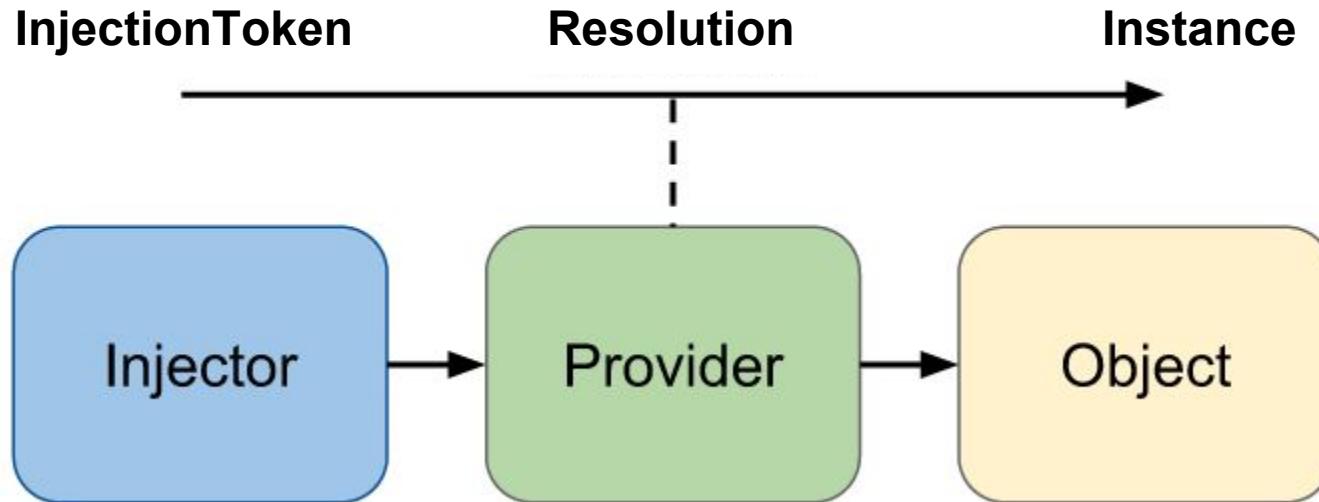
@Component({...})
class Car {

    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

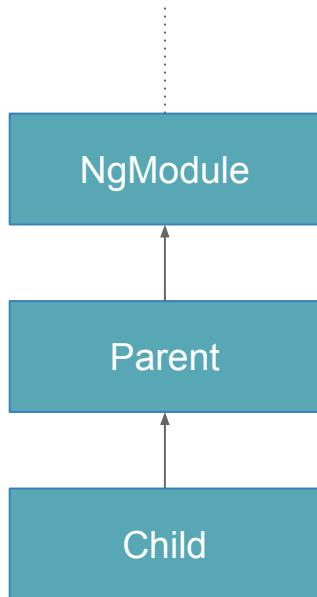
Factory : useFactory

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

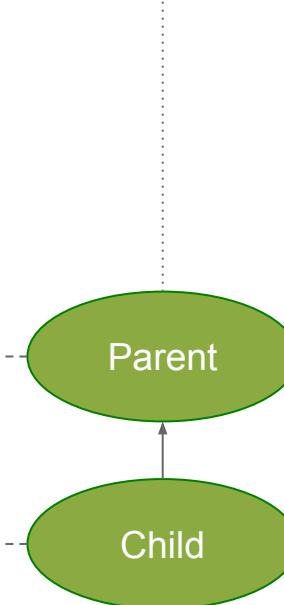
```
providers: [  
    V8Engine,  
    {  
        provide: Engine,  
        useFactory: createEngineFactory,  
        deps: [V8Engine]  
    }  
]
```

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

git checkout -f step-17

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/`
 - `ng g s shared/people-service/people`
- 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

git checkout -f

step-17-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

Exemples

hello

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

HELLO

Exemples

1368730200

```
{ { "1368730200" | date } }
```

Jeudi 16 mai 2013 20H50

Exemples

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 : <https://angular.io/docs/ts/latest/api/common/DatePipe.html>

18

Exercice 18 : Améliorer la lisibilité des données

```
git checkout -f step-18
```

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

SOLUTION

git checkout -f

step-18-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

`git checkout -f step-19`

- 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

git checkout -f

step-19-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]="expression">  
  
  <p *ngSwitchCase="whenExpression1">...</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, Nglf 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

git checkout -f step-20

- 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 **Nglf**

SOLUTION

git checkout -f

step-20-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
- **attribut** : 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

git checkout -f step-21

- 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**:
 - ``

SOLUTION

git checkout -f

step-21-solution

Bonus 1

Introduction aux Tests

Outils

L'Outillage pour le TDD



Angular CLI

- ZERO configuration avec Angular CLI :

ng test

```
1. wchegham@wchegham-mbp-2: ~/Sandbox/dev/oss (zsh)
wchegham 192.168.1.16 > ~/Sandbox/dev/oss >
$ ng
Usage: ng <command (Default: help)>

Available commands in angular-cli:

ng addon <addon-name> <options...>
  Generates a new folder structure for building an addon, complete with test harness.
  --dry-run (Boolean) (Default: false)
    aliases: -d
  --verbose (Boolean) (Default: false)
    aliases: -v
  --blueprint (String) (Default: addon)
```

Angular CLI : des tas d'options

- ng test --browsers
- ng test --code-coverage
- ng test --progress
- ng test --reporters
- ng test --single-run
- ng test --sourcemap
- ng test --watch
- ...

Concepts

Concepts généraux de TDD

- **describe(string, function)** : un scénario de specs à exécuter
- **it(string, function)** : une spec contient un ou plusieurs vérifications
- **expect(actual)...** : une vérification repose sur des comparateurs
- **toXXXX(expected) ⇒ toBe(expected)**: un comparateur compare le résultat obtenu avec un résultat attendu
- **beforeAll/afterAll, beforeEach/afterEach** : Permet d'exécuter du code avant/après chaque scénario/test

Concepts TDD (exemple)

```
describe('scenario description...', () => {  
    // setup  
    beforeEach(() => ...);  
  
    // test  
    it('should return some value', () => {  
        expect(service.getSomeValue()).toContain('some value');  
    });  
});
```

API Angular pour TDD

- **TestBed** : module de configuration des tests (similaire à NgModule)

```
TestBed.configureTestingModule({
  declarations: [ UserComponent ],
  providers: [ UserService ]
});
```

- **Inject** : injection des dépendances

```
inject([ UserService ], (user) => {
  // sync test
});
```

Composant

Test des Composants

TestBed

- **TestBed.createComponent** : crée une instance du composant (fixture)
- **TestBed.overrideComponent** : surcharge une instance d'un composant

ComponentFixture

- **fixture.componentInstance** : accès à l'instance du composant
- **fixture.nativeElement** : accès au DOM du composant
- **fixture.debugElement** : fonction helper
- **fixture.detectChanges** : déclenche la détection du changement

Test des Composants

```
beforeEach(() => {  
  
  TestBed.configureTestingModule({  
    declarations: [ UserProfileComponent ]  
  });  
  
  const fixture = TestBed.createComponent(UserProfileComponent);  
  const instance = fixture.componentInstance;  
  const element = fixture.nativeElement;  
  const debug    = fixture.debugElement;  
  
  //...  
  fixture.detectChanges();  
  
});
```

Test des Composants (Astuce)

- Masquer les erreurs liées aux composants imbriqués !

```
import { NO_ERRORS_SCHEMA } from '@angular/core';
beforeEach(() => {

  TestBed.configureTestingModule({
    declarations: [ ... ],
    schemas: [ NO_ERRORS_SCHEMA ]
  });
});
```

Test des Composants (Exemple)

```
@Component({
  selector: 'user-profile',
  template: `<h1>Hi {{name}}!</h1>`
})
export class UserComponent {
  @Input() name;
}
```

Test des Composants (Exemple)

```
// setup
beforeEach(() => ... );

it('should render `Hi Igor!`', inject(() => {
  instance.name = 'Igor';

  fixture.detectChanges();

  fixture.whenStable().then(() => {
    expect(element.querySelector('h1').innerText).toBe('Hi Igor!');
  });
}));
```

Directive

Test des Directives

- Même API que pour les Composants : TestBed, ComponentFixture...
- Nécessite la création d'un composant de test pour accueillir la directive

```
@Component({  
  selector: 'test-my-directive',  
  template: ``  
})  
export class HostComponentForDirective {}
```

Test des Directives

- Renseigner la directive et le composant dans TestBed

```
beforeEach(() => {
  TestBed.configureTestingModule({
    declarations: [
      HostComponentForDirective,
      MyDirective
    ]
  });
});
```

Test des Directives

```
@Component({  
  selector: 'test-my-directive',  
  template: ``  
})  
export class HostComponentForDirective {}
```

TestBed

```
.overrideComponent(HostComponentForDirective, {  
  set: {  
    template: `<div my-directive></div>`  
  }  
})  
.createComponent(HostComponentForDirective);
```

Test des Directives (Exemple)

```
it('should ...', () => {  
  
  let fixture = TestBed  
    .overrideComponent(HostComponentForDirective, {  
      set: {  
        template: '<div my-badge></div>'  
      }  
    })  
    .createComponent(HostComponentForDirective);  
  
  fixture.detectChanges();  
  
  const divElement = fixture.nativeElement.querySelector('div');  
  expect(divElement.innerHTML).toBe('^^');  
  
});
```

Service

Test des Services

```
@Injectable()  
export class UserService {  
  
  getName(){  
    return 'Igor';  
  }  
  
}  
}
```

Test des Services

```
// setup
beforeEach(() => TestBed.configureTestingModule({ providers:[ UserService ] }));

it('should return a valid name', inject([ UserService ], (service) => {
  let name = service.getName();
  expect(name).toContain('Igor');
  expect(name.length).toEqual(4);
}));
```

Test des Services (Http)

```
@Injectable()
export class UserService {

    constructor(private http:Http) { }

    getNameFromServer(){
        return this.http.get('api/users/123?info=username')
            .map(response => response.json());
    }
}
```

Test des Services (Http)

```
//setup
const service: UserService;
beforeEach(() => {
  TestBed.configureTestingModule({
    imports: [ HttpClientModule ],
    providers: [ UserService ]
  })
  service = TestBed.get(UserService);
});

it('should return a valid name', () => {
  service.getNameFromServer().subscribe(name => {
    expect(name).toContain('Igor');
    expect(name.length).toEqual(4);
  });
});
```

Attention : Exécute des vraies requêtes HTTP !

Test des Services (avec HttpClientTestingModule)

```
//setup
beforeEach(() => {
  TestBed.configureTestingModule({
    imports: [ HttpClientTestingModule ],
    providers: [
      UserService,
    ]
  })
});

it('should...', inject([XHRBackend, UserService], (mockbackend, service) => {
  // ...
})
```

Test des Services (avec HttpClientTestingModule)

```
it('should return mocked username', () => {
  let response = 'Brad';

  TestBed.get(UserService).getNameFromServer().subscribe(name => {
    expect(name).toContain('Brad');
    expect(name.length).toBe(4);
  });

  const req = httpTestingController.expectOne('/service-url');

  req.flush(response);

  httpTestingController.verify();
});
```

Attention : N'exécute pas les requêtes HTTP !

Test des Services (avec MockBackend)

```
//setup
beforeEach(() => {
  TestBed.configureTestingModule({
    imports: [ HttpClientModule ],
    providers: [
      UserService,
      { provide: XHRBackend, useClass: MockBackend }
    ]
  })
});
// ...
```

Test des Services (avec MockBackend)

```
it('should return mocked username',
  inject([XHRBackend, UserService], (mockbackend, service) => {
    let response = 'Brad';

    mockbackend.connections.subscribe(connection => {
      connection.mockRespond(new Response(
        new ResponseOptions({body: JSON.stringify(response)})
      ));
    });

    service.getNameFromServer().subscribe(name => {
      expect(name).toContain('Brad');
      expect(name.length).toBe(4);
    });
  });
});
```

Attention : N'exécute pas les requêtes HTTP !

Pipe

Test des Pipes

```
@Pipe({
  name: 'capitalise'
})
export class UpPipe implements PipeTransform {
  transform(value: string): string {
    if (typeof value !== 'string') {
      throw new Error('Requires a String as input');
    }
    return value.toUpperCase();
  }
}
```

Test des Pipes

```
it('should capitalise', () => {  
  const pipe = new UpPipe();  
  
  expect( pipe.transform('sfEiR') ).toEqual('SFEIR');  
});
```

Test des Pipes (Catch des erreurs)

```
it('should throw with invalid values', () => {  
  expect(()=> pipe.transform(123) ).toThrow();  
  expect(()=> pipe.transform(null) ).toThrow();  
  expect(()=> pipe.transform() ).toThrowError('Requires a String as input');  
});
```

22

Exercice 22: TDD

`git checkout -f step-22`

- Étudier et Compléter les tests de :
 - UpdateComponent
 - SfeirBadgeDirective
 - PeopleService
 - NaPipe

SOLUTION

git checkout -f

step-22-solution

BONUS 2

Communication avancée

3 façons de communiquer

- entre parent- enfant:
 - Input/Output
- entre autres éléments
 - bus de communication
 - architecture flux

Pour notifier

- Notification via un EventEmitter
- Global ou dans des services
- Utiliser ces méthodes
 - subscribe et emit

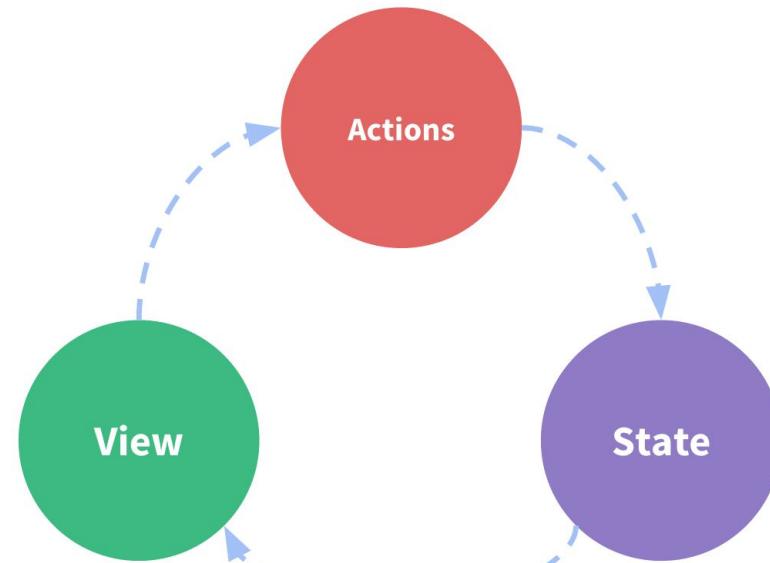
```
public bus$:EventEmitter<Object>;
constructor() {
    this.bus$= new EventEmitter();
}
notify(data) {
    this.bus$.emit(data);
}
```

```
import bus from '../path/toBus';
bus.subscribe((datas) =>{...});
bus.notify(datas);
```

State Management

- Architecture dataflow: like Redux :)
- librairie tiers à installer

```
npm install @ngrx/store --save  
npm install @ngxs/store --save
```

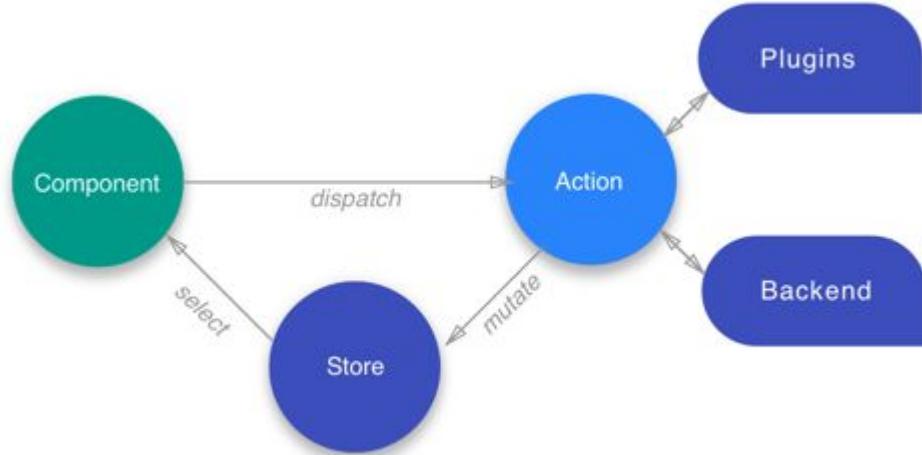


A blurred background image showing a group of people in what appears to be a conference room or meeting space. They are seated around a large table, facing towards the front where a presentation might be taking place. The overall atmosphere is professional and focused.

NGXS

Concepts

- les éléments:
 - store
 - state
 - actions
 - sélecteurs



Le Store

- Service de base qui va permettre de :
 - dispatcher des actions
 - sélectionner une partie de l'état
- On doit injecter dans le module le NgxsModule, en lui passant une liste de State

```
NgxsModule.forRoot([AppState]),  
NgxsModule.forFeature([FeatureState])
```

```
import { Store } from '@ngxs/store';  
import { Observable } from 'rxjs/Observable';  
import { AddAnimal } from './animal.events';  
  
@Component({ ... })  
export class ZooComponent {  
  @Select(state => state.animals) animals$;  
  constructor(private store: Store) {}  
  
  addAnimal(name) {  
    this.store.dispatch(new AddAnimal(name));  
  }  
}
```

Le State

- Une classe avec le décorateur **@State**
 - name: le nom de l'état
 - defaults: les valeurs par défaut
 - children: des states enfants
 -
- Peut utiliser des services

```
import { State } from '@ngxs/store';

@State<string[]>({
  name: 'animals',
  defaults: []
})
export class AnimalsState {

  constructor(private zooService: ZooService) {}

}
```

Mutations et actions

- **SEULE** façon de modifier le state
- déclenchée par une **action**
- **synchrones et asynchrones**
- Méthodes du State avec le decorateur **@Action()**
- **StateContext**
 - **setState**
 - **getState**
 - **patchState**
 - **dispatch**

```
@Action(LoadAnimals)
LoadAnimals({ dispatch }: StateContext<ZooStateModel>, { payload
}: LoadAnimals) {
    return this.animalService.load(payload)
        .pipe((result) => dispatch(new AnimalsLoaded(result)));
}

@Action(AnimalsLoaded)
AnimalsLoaded({ getState, setState }: StateContext<ZooStateModel>, { payload }: AnimalsLoaded) {
    const state = getState();
    setState({
        ...state,
        animals: [ ...payload ]
    });
}
```

Les actions

- déclenchée par un **dispatch**

Dans le State:

```
export class SetPeople {  
    constructor (public payload) { }  
}
```

Dans le composant

```
this.store.dispatch( new SetPeople(people));
```

Les selecteurs

- Pour extraire une partie de l'état
- Au niveau du state
 - réutilisable
 - decorator **@Selector()**
- Au niveau du composant
 - decorator **@Select()**
 - **store.select()**

```
@State({  
  name: 'animals',  
  defaults: []  
})  
export class ZooState {  
  @Selector()  
  static animals(state) {  
    return state.animals;  
  }  
}
```

```
@Component({ ... })  
export class ZooComponent {  
  // Reads the name of the store from the store class  
  @Select(ZooState) animals$: Observable<string[]>;  
  // Reads the name of the property minus the $  
  @Select() animals$: Observable<string[]>;  
  // Also accepts a function like our select method  
  @Select(state => state.animals) animals$: Observable<string[]>;  
  
  constructor(private store: Store) {  
    this.animals$ = this.store.select(state => state.zoo.animals);  
  }  
}
```

23

Exercice 23 : Je branche NGXS

`git checkout -f step-23`

- Un champs de saisie a été ajouté pour filtrer le tableau
- Vous disposez d'un squelette de Store
- Nous allons mettre en place une architecture NgXs pour
 - récupérer la liste des utilisateurs
 - Filtrer le tableau

SOLUTION

git checkout -f

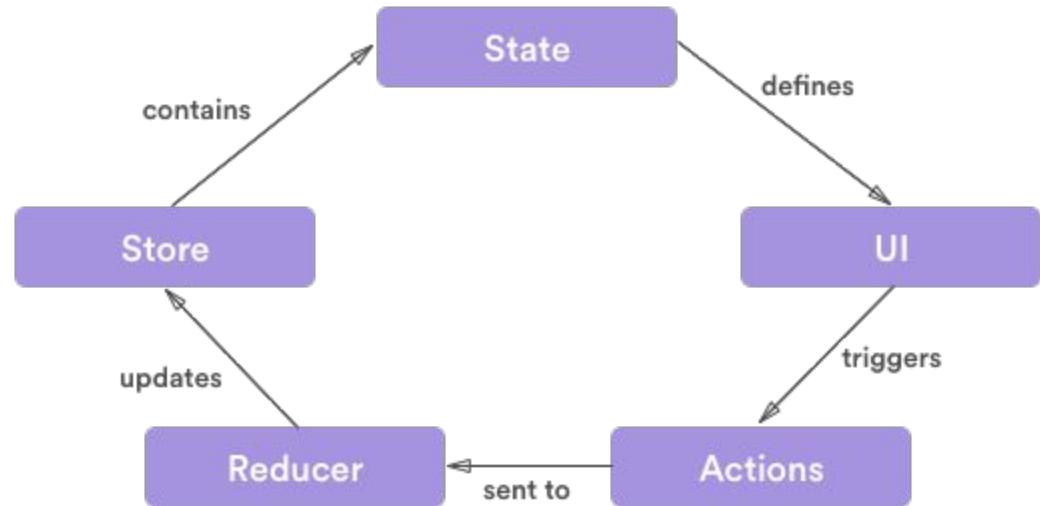
step-23-solution

The background of the slide is a blurred photograph of a group of people in what appears to be a conference room or meeting space. They are seated around a large table, looking towards the front where a presentation might be taking place. The overall atmosphere is professional and focused.

NGRX

Architecture

- 3 éléments
 - state et store
 - reducer
 - actions



Le Store

- Service de base qui va permettre de :
 - dispatcher des actions
 - sélectionner une partie de l'état
- On doit injecter dans le module le storeModule
 - en donnant le **nom** de notre Store et notre **reducer**

```
storeModule.forRoot({people: reducer}),  
storeModule.forFeature('people', reducer)
```

```
import { Store } from '@ngrx/store';  
  
@Component({ ... })  
export class MyComponent {  
  
  constructor(private store: Store) {  
  
    this.count$ = store.pipe(select('count'));  
  }  
  
  onClick(name) {  
    this.store.dispatch(new MyAction(name));  
  }  
}
```

Le State

- map de clé valeurs
- récupérer grâce à nos sélecteurs
- Retourne un observable

Dans le Store:

```
const state = {  
    users: [],  
    search: ''  
}
```

Dans le composant

```
this.store.select('people');  
this.store.select(fromRoot.getSearch);
```

Les reducers

- **SEULE** façon de modifier le state
- déclenchée par une **action**
- Immutable
- les mutations sont **synchrones**

Dans le Store:

```
export function reducer(state: State = initialState,  
action) {  
  switch (action.type) {  
    case PeopleAction.SET_PEOPLE:  
      const users = action.payload;  
      return { ...state, users };  
    default:  
      return state;  
  }  
}
```

Dans le composant

```
this.store.dispatch(new PeopleAction.SetPeople([]));
```

Les actions

- déclenchée par un **dispatch**
- Actions typées

Dans le Store:

```
export class SetPeople implements Action {  
  readonly type = 'SET_PEOPLE';  
  constructor (public payload) {}  
}
```

Dans le composant

```
this.store.dispatch(new PeopleAction.SetPeople(people));
```

Les selector

- Feature selector
 - pour extraire une portion dédié à une feature
- Selector
 - partie du store

```
// Feature selector

export const getPeopleState = createFeatureSelector<State>('people');

// Selector

export const getSearch = createSelector(getPeopleState ,
  (state: State) => {...});
```

Mais aussi

- State composition
- effects
- meta-reducers
- router-store
- store-devtools
- entity
- ...

ill
ndance.Levesque@ag2rlamondiale.fr
ce.Paquette@ag2rlamondiale.fr

Filter...	Commit	Action	State	Diff	Test
@ngrx/store/init	5:21:37.21				
@ngrx/effects/init	5:21:37.21				
ROUTER_NAVIGATION	5:21:37.21				
ROUTE_ERROR	5:21:37.21				
ROUTER_NAVIGATION	5:21:37.22				
LOAD_PEOPLE	5:21:37.22				
SET_PEOPLE	5:21:37.22				
FILTER_PEOPLE	5:21:37.22				

Diff

Tree Raw

+ people (pin)
+ list (pin): { search: "q" }

24

Exercice 24 : Je branche NGRX

`git checkout -f step-24`

- Un champs de saisie a été ajouté pour filtrer le tableau
- Vous disposez d'un squelette de Store
- Nous allons mettre en place une architecture NgRx pour
 - récupérer la liste des utilisateurs
 - Filtrer le tableau

SOLUTION

git checkout -f

step-24-solution

Ressources

Ressources

- Site Officiel : <https://angular.io/>
- Victor Savkin : <http://vsavkin.com/>
- Thoughtram: <http://blog.thoughtram.io/categories/angular-2/>
- Test:<https://github.com/victormejia/angular-testing-workshop#module-2-configuring-terminal-reporting>
- Ngxs: <https://ngxs.gitbooks.io/ngxs>
- Ngrx: <https://github.com/ngrx/platform>

