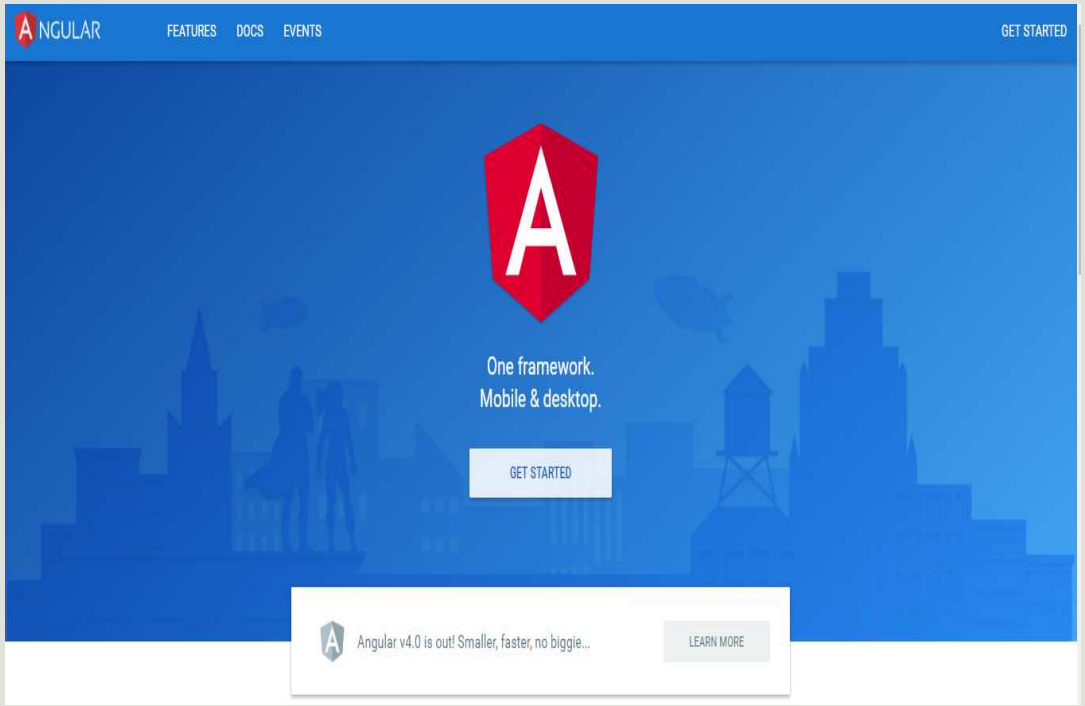
Angular Introduction

AYMEN SELLAOUTI

aymen.sellaouti@gmail.com

Références


ANGULAR
FEATURES
DOCS
EVENTS
GET STARTED
One framework.
Mobile & desktop.
GET STARTED
Angular v4.0 is out! Smaller, faster, no biggie...
LEARN MORE


Learn Angular
Gravissez tous les concepts jusqu'au sommet
Maîtriser les concepts du Framework Angular
pour développer des applications robustes
Auteur : William KOZA

Plan du Cours

1. Introduction
2. Les composants
3. Les directives

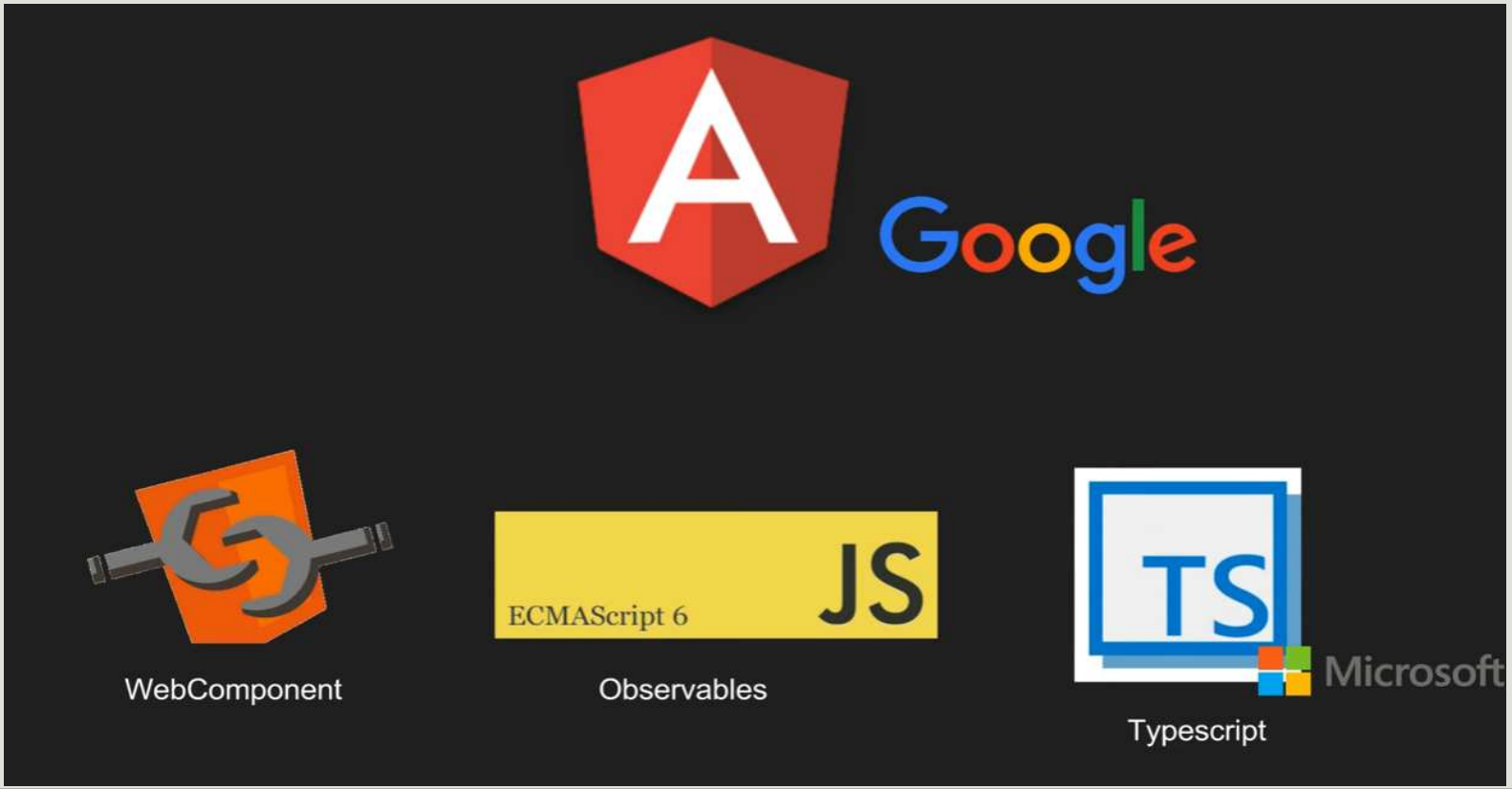
3 Bis. Les pipes

4. Service et injection de dépendances
5. Le routage
6. Form
7. HTTP

Objectif

1. Définir Angular
2. Comprendre l'architecture d'Angular
3. Avoir un aperçu sur les différents éléments d'Angular
4. Installer Angular et créer notre première application

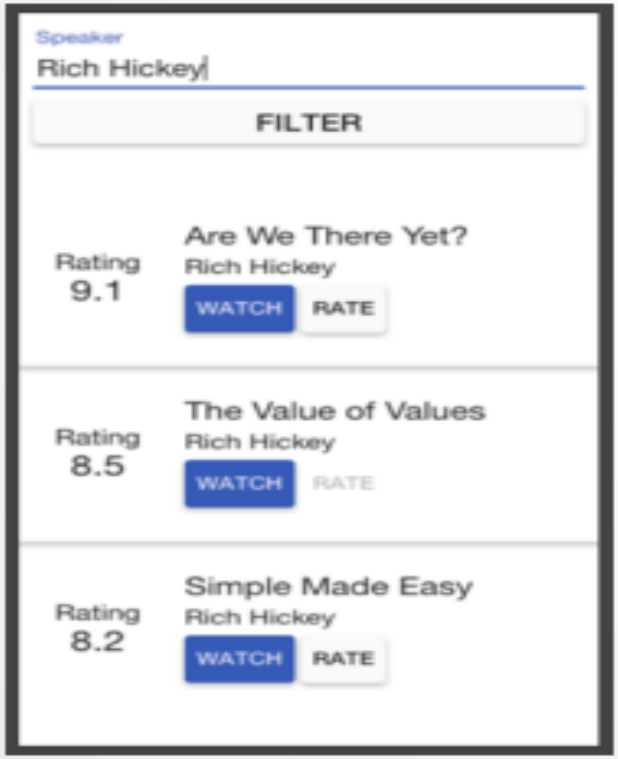
C’est quoi Angular?

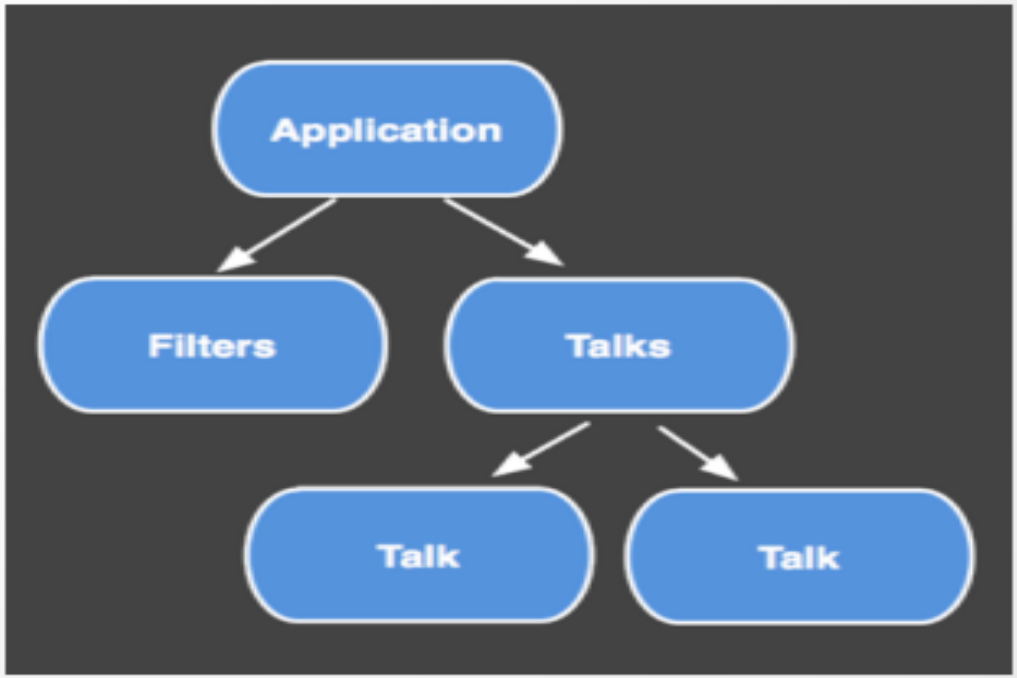


C’est quoi Angular?

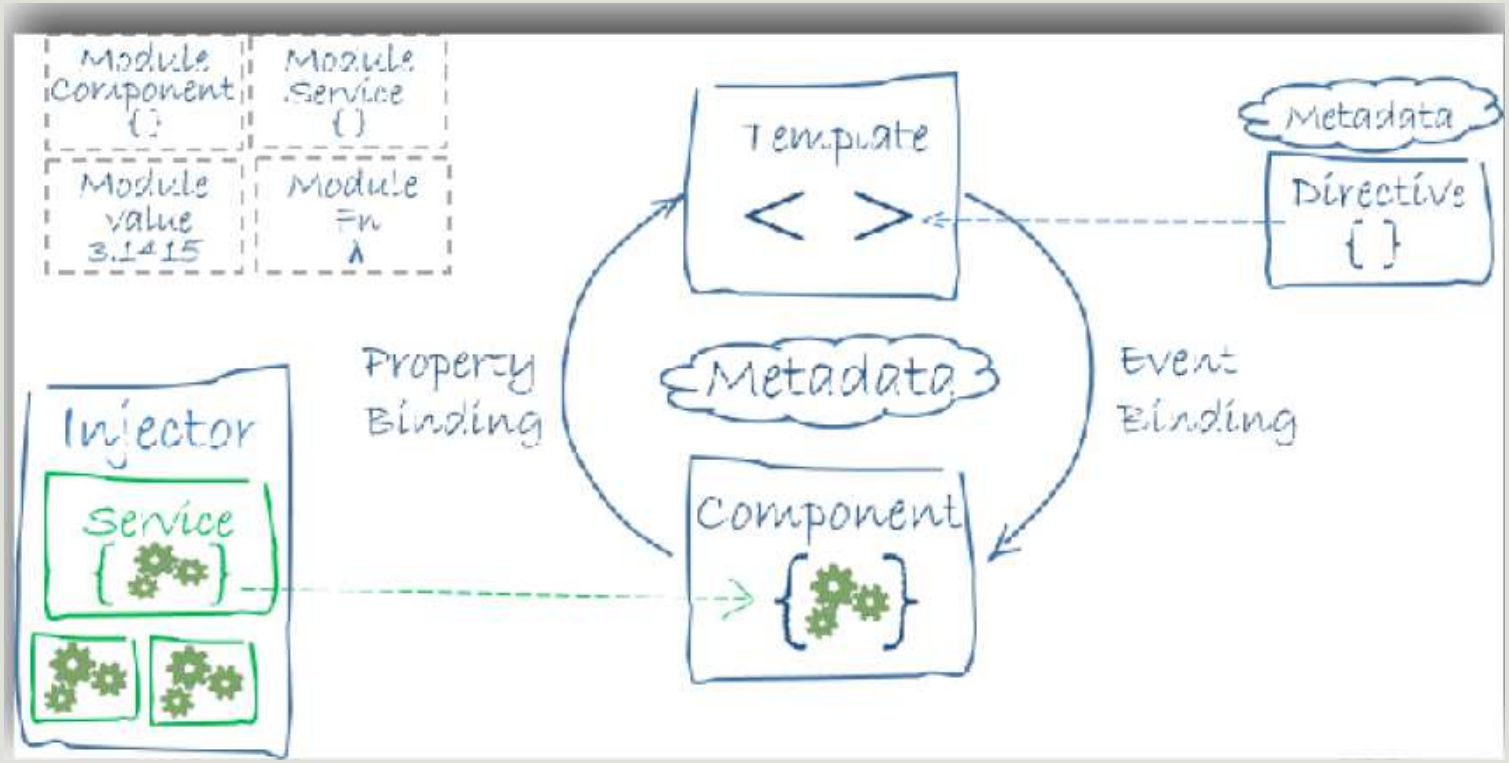
- Framework JS
- SPA
- Supporte plusieurs langages : ES5, EC6, TypeScript, Dart
- Modulaire (organisé en composants et modules)
- Rapide
- Orienté Composant

Angular : Arbre de composants





Architecture Angular



Module

Angular est modulaire

- Chaque application va définir Angular Modules or NgModules
- Chaque module Angular est une classe avec une annotation @NgModule
- Chaque application a au moins un module, c'est le module principale.

AppModule : le module principal

- le module principal est le module qui permet de lancer l'application de la bootstraper.
- Le nom par convention est AppModule.
- L'annotation (decorator) @NgModule identifie

AppModule comme un module Angular.

- L'annotation prend en paramètre un objet spécifiant à

Angular comment compiler et lancer l'application.

- imports : tableau contenant les modules utilisés.
- declarations : tableau contenant les composants, directives et pipes de l'application.
- bootstrap : indique le composant exécuter au lancement de l'application.

Il peut y avoir aussi d'autres attributs dans cet objet

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

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

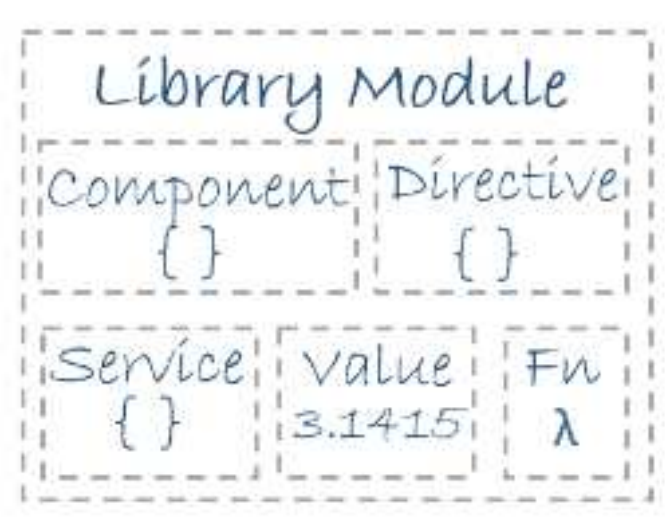
Les librairies d'Angular

- Ensemble de modules JS

- Des librairies qui contiennent un ensemble de fonctionnalités.

- Toutes les librairies d'Angular sont préfixées par @angular

- Récupérable à travers un import JavaScript.



- Exemple pour récupérer l'annotation component : import { Component } from '@angular/core';

Les composants

- Le composant est la partie principale d'Angular.

- Un composant s'occupe d'une partie de la vue.

- L'interaction entre le composant et la vue se fait à travers une API.

Template

- Un Template est le complément du composant.
- C'est la vue associée au composant.
- Elle représente le code HTMl géré par le composant.

Les méta data

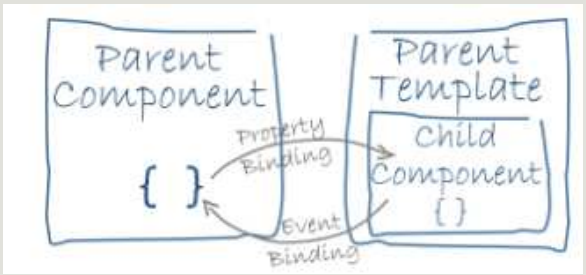
- Appelé aussi « decorator », ce sont des informations permettant de décrire les classes.

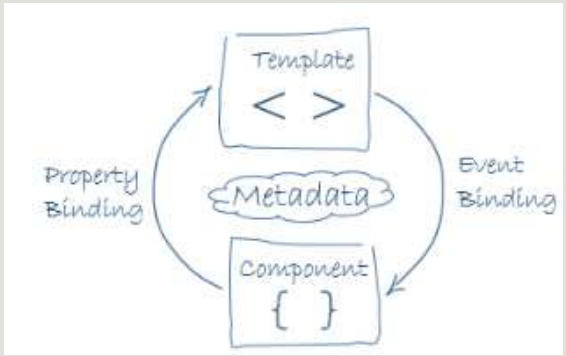
- @Component permet d'identifier la classe comme étant un composant angular.

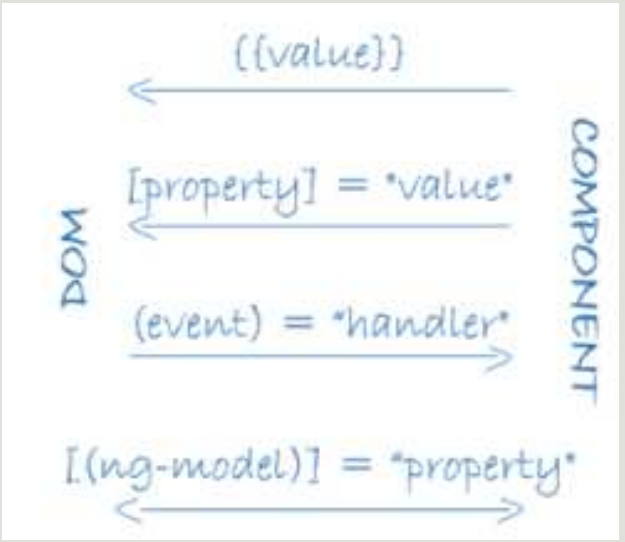
Le Data Binding

- Le data binding est le mécanisme qui permet de mapper des éléments du DOM avec des propriétés et des méthodes du composant.

- Le Data Binding permettra aussi de faire communiquer les composants.







Les directives

- Les directives Angular sont des classes avec la métadata @Directive. Elle permettent de modifier le DOM et de rendre les Template dynamiques.

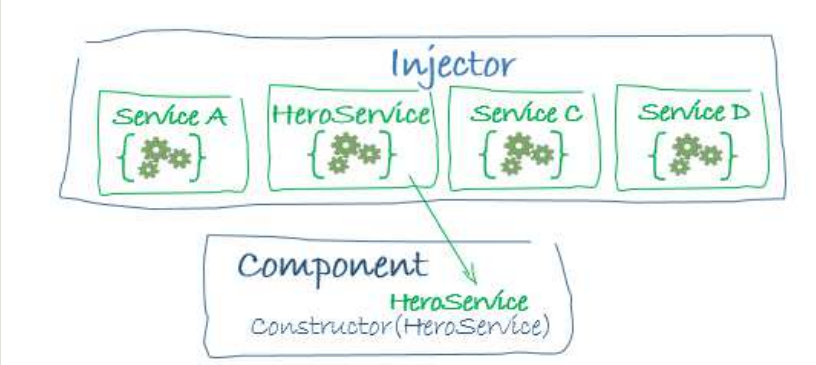
- Apparaissent dans des éléments HTML comme les attributs.
- Un composant est une directive à laquelle Angular a associé un Template.
- Ils existe deux autres types de directives :
  - Directives structurelles
  - Directive d'attributs

Les services

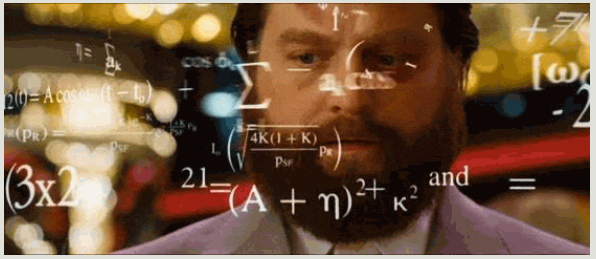
- Classes permettant d'encapsuler des traitements métiers.

- Doivent être légers.

- Associées aux composants et autres classes par injection de dépendances.

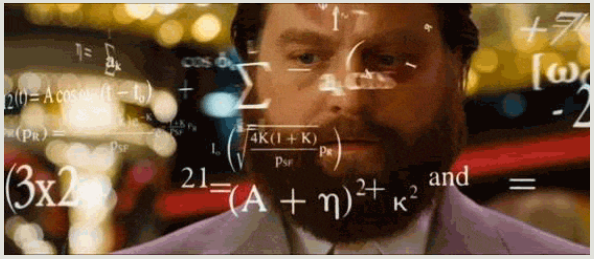


Installation d'Angular
Angular Cli



- Nous allons installer notre première application en utilisant angular Cli.
- Si vous avez Node c'est bon, sinon, installer NodeJs sur votre machine. Vous devez avoir une version de (Angular 7 node >= 8.9.0, Angular 8 node >= 10.9.0)
- Une fois installé vous disposez de npm qui est le Node Package Manager. Afin de vérifier si vous avez NodeJs installé, tapez npm –v.
- Installer ensuite TypeScript. Pour ce faire, tapez npm install -g typescript.
- Installer maintenant le Cli en tapant la : npm install -g @angular/cli
  - npm install -g @angular/cli@6.0.0 installe la version 6.0.0
  - npm view @angular/cli affiche la liste des versions de la cli
- Installer un nouveau projet à l'aide de la commande ng new nomProjet
- Afin d'avoir du help pour le cli tapez ng help
- Lancer le projet en utilisant la commande ng serve

Installation d'Angular
Angular Cli



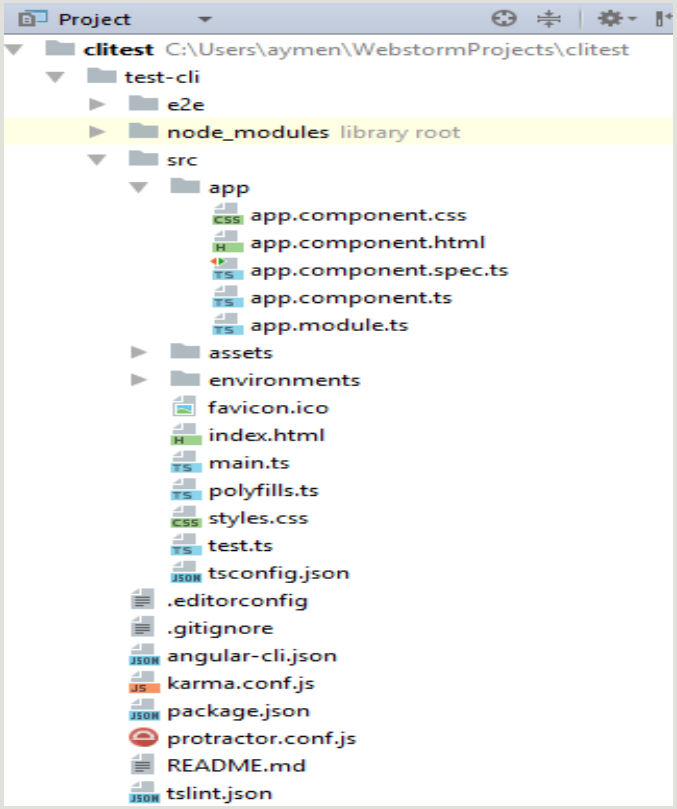
- Positionnez vous maintenant dans le dossier
- Tapez la commande suivante : ng new nomNewProject
- lancez le projet à l'aide de npm : ng serve
- Naviguez vers l'adresse mentionnée.
- Vous pouvez configurer le Host ainsi que le port avec la commande suivante : ng serve --host leHost --port lePort

- Pour plus de détails sur le cli visitez https://cli.angular.io/

Quelques commandes du Cli

Commande	Utilisation
Component	ng g component my-new-component
Directive	ng g directive my-new-directive
Pipe	ng g pipe my-new-pipe
Service	ng g service my-new-service
Class	ng g class my-new-class
Interface	ng g interface my-new-interface
Module	ng g module my-module

Arborescence d'un projet Angular



E2e : end to end pour réaliser des tests d'intégration
Node_modules : les dépendences

Src : dossier contenant l'index le code source les styles, main.ts qui est le bootstrap d'angular ainsi que d'autres fichiers

App : Les sources du projet par défaut ainsi que le module appModule, le fichier app.component.ts ainsi que son template (app.component.html), son style (app.component.css) et app.component.spec.ts pour les tests unitaires.

Ajouter Bootstrap

On peut ajouter Bootstrap de plusieurs façons :

1- Via le CDN à ajouter dans le fichier index.html

2- En le téléchargeant du site officiel

3- Via npm

- Npm install bootstrap --save

Ajouter Bootstrap

Pour ajouter les dossiers téléchargés on peut le faire de deux façons :

1- En l'ajoutant dans index.html

2- En ajoutant le chemin des dépendances dans les tableaux styles et scripts dans le fichier angular.json:

"styles": [

"../node_modules/bootstrap/dist/css/bootstrap.min.css",

"styles.css",

],

"scripts": [

"../node_modules/jquery/dist/jquery.min.js",

"../node_modules/popper.js/dist/umd/popper.min.js",

"../node_modules/bootstrap/dist/js/bootstrap.min.js"

],

Ajouter Bootstrap

Ajouter dans le fichier src/style.css un import de vos bibliothèques.

```
@import "~bootstrap/dist/css/bootstrap.css";
```

Essayer la même chose avec font-awesome.

Angular
Les composants

AYMEN SELLAOUTI

Plan du Cours

1. Introduction
2. Les composants
3. Les directives

3 Bis. Les pipes

4. Service et injection de dépendances
5. Le routage
6. Form
7. HTTP

Objectifs

1. Comprendre la définition du composant
2. Assimiler et pratiquer la notion de Binding
3. Gérer les interactions entre composants.

Qu'est ce qu'un composant (Component)

- Un **composant** est une **classe** qui permet de **gérer une vue**. Il se charge **uniquement** de cette vue la.

Plus simplement, un composant est un fragment HTML géré par une classe JS (component en angular et controller en angularJS)

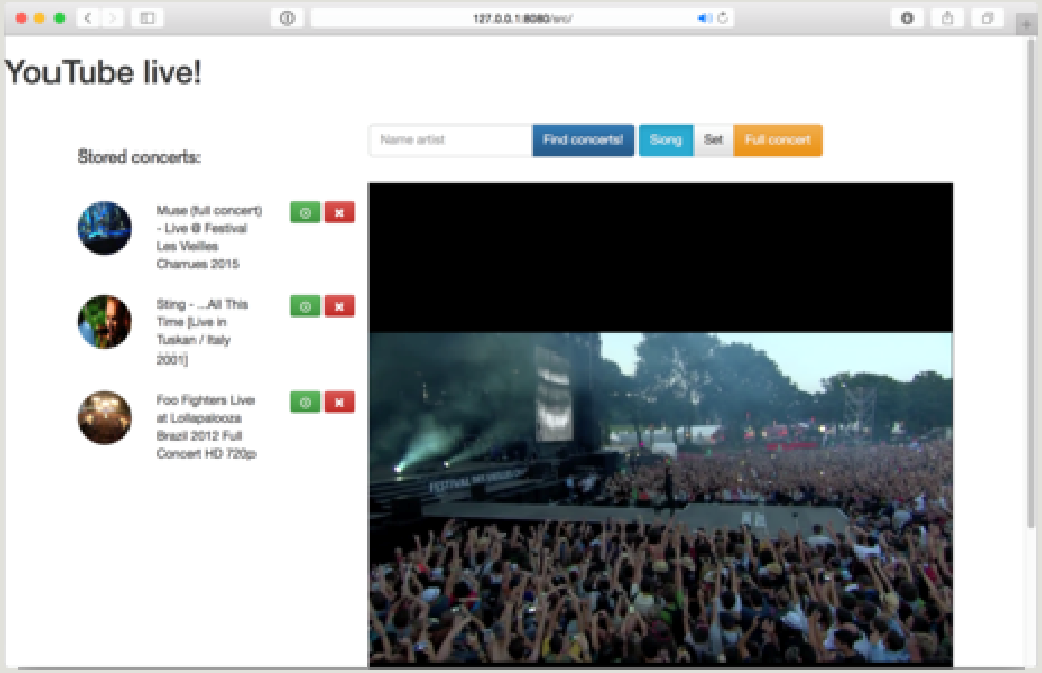
- Une application Angular est un **arbre de composants**

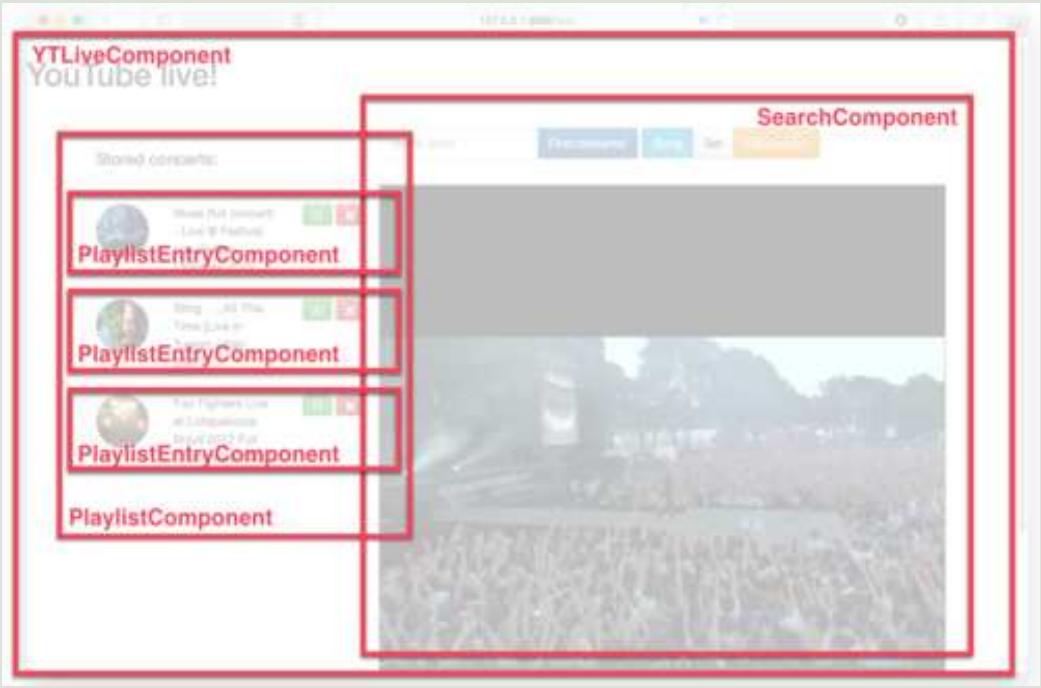
- La racine de cet arbre est l'application lancée par le navigateur au lancement.

- Un composant est :
  - Composable (normal c'est un composant)
  - Réutilisable
  - Hiérarchique (n'oublier pas c'est un arbre)

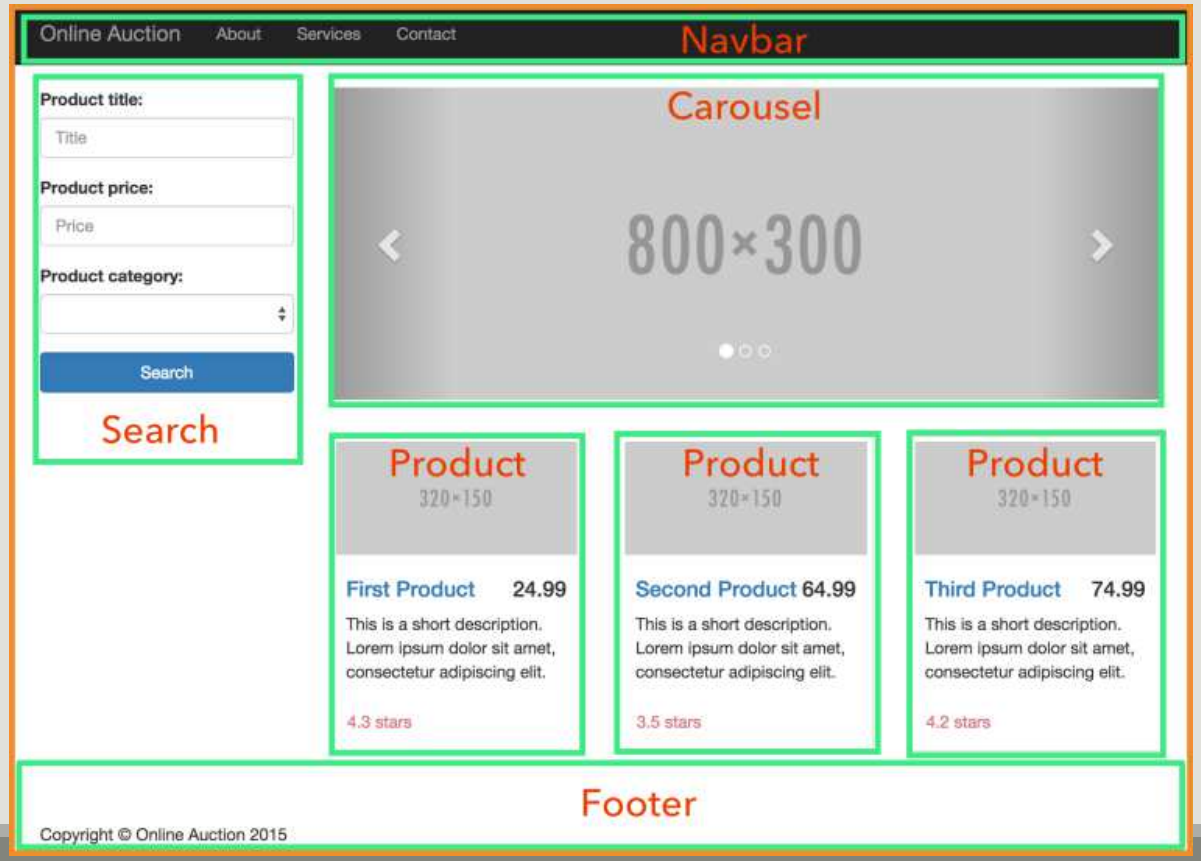
NB : Dans le reste du cours les mots composant et component représentent toujours un composant Angular.

Quelques exemples

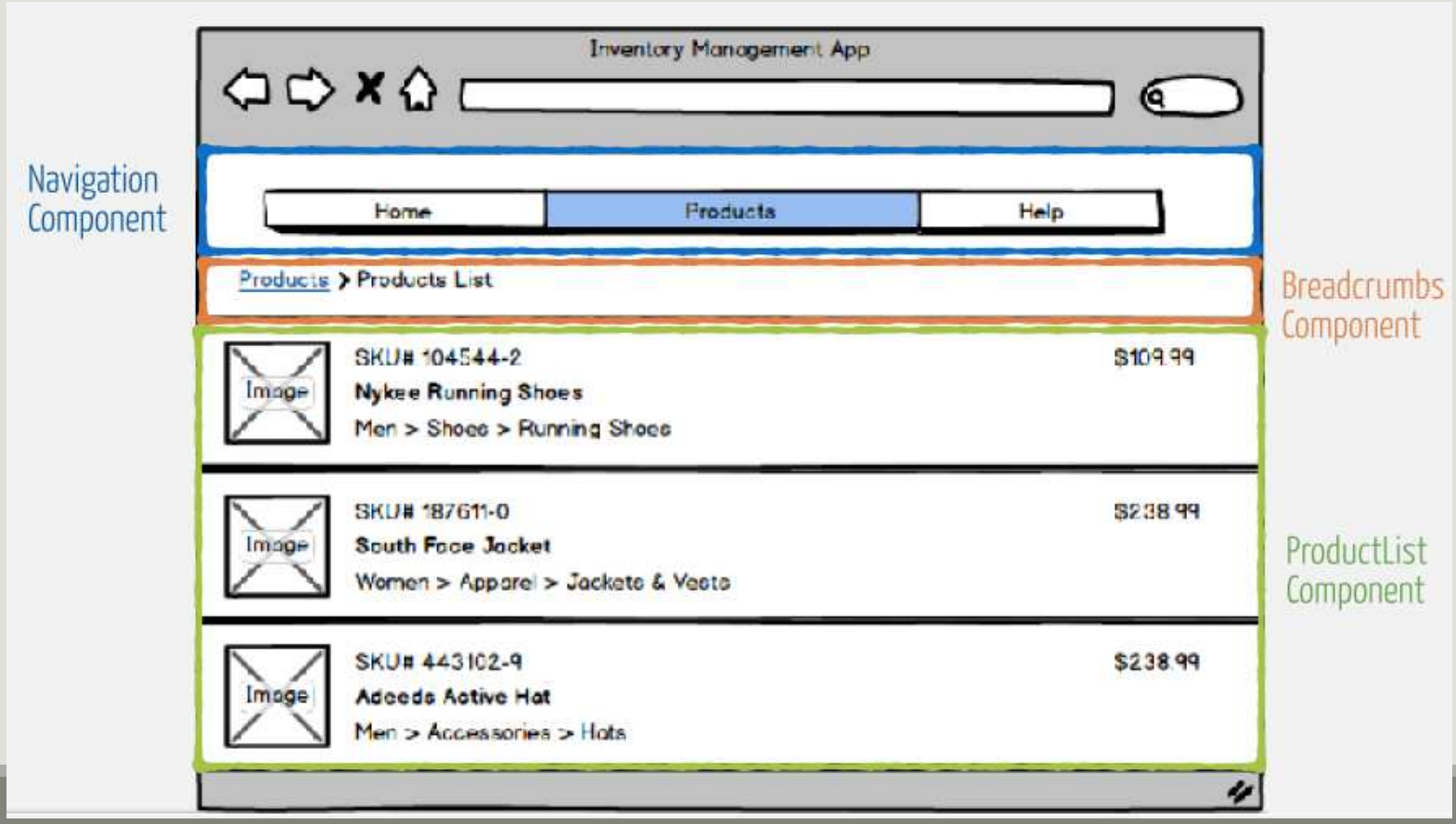




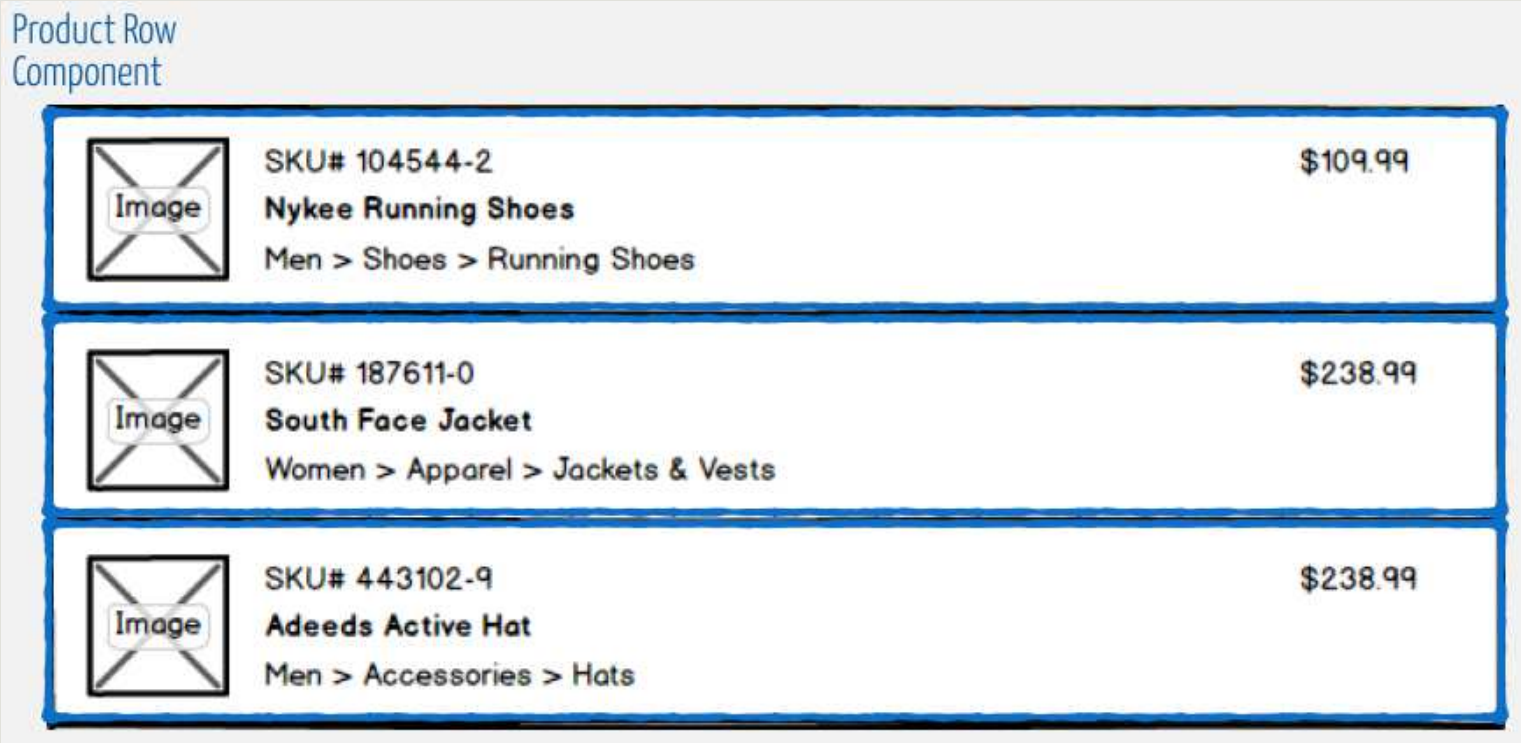
Quelques exemples



Quelques exemples



Quelques exemples



Quelques exemples

Premier Composant

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app works for tekup people !';
}
```

Chargement de la classe Component

Le décorateur @Component permet d'ajouter un comportement à notre classe et de spécifier que c'est un Composant Angular.
selector permet de spécifier le tag (nom de la balise) associé ce composant
templateUrl: spécifie l'url du template associé au composant
styleUrls: tableau des feuilles de styles associé à ce composant

Export de la classe afin de pouvoir l'utiliser

Création d'un composant

- Deux méthodes pour créer un composant
  - Manuelle
  - Avec le Cli

- Manuelle
  - Créer la classe
  - Importer Component
  - Ajouter l'annotation et l'objet qui la décore
  - Ajouter le composant dans le **AppModule(app.module.ts)** dans l'attribut **declarations**
- Cli
  - Avec la commande ng **g**enerate component my-new-component ou son raccourci ng g c my-new-component

Property Binding

Balises

Propriétés

HTML

Attributs

Méthodes

TS

Property Binding

```html
<div [style.backgroundColor]="color">
  Color
</div>

<input [(ngModel)]="color"
       type="text"
       class="form-control"
>
le contenu de la propriété color est {{color}}
<button (click)="loggerMesData()">log data</button>
<br>
<a (click)="goToCv()" >Go to Cv</a>
```

HTML

```ts
@Component({
  selector: 'app-color',
  templateUrl: './color.component.html',
  styleUrls: ['./color.component.css'],
  providers: [PremierService]
})
export class ColorComponent implements OnInit {
  color = 'red';
  constructor() { }

  ngOnInit() {}
  processReq(message: any) {
    alert(message);
  }
  loggerMesData() {
    this.premierService.logger('test');
  }
  goToCv() {
    const link = ['cv'];
    this.router.navigate(link);
  }
```

TS

Property Binding

- Binding unidirectionnel.
- Permet aussi de récupérer dans le DOM des propriétés du composant.
- La propriété liée au composant est interprétée avant d'être ajoutée au Template.
- Deux possibilités pour la syntaxe:
  - [propriété]
  - **bind-**propriété

Event Binding

- Binding unidirectionnel.
- Permet d'interagir du DOM vers le composant.
- L'interaction se fait à travers les événements.
- Deux possibilités pour la syntaxe :
  - (evenement)
  - on-evenement

Property Binding et Event Binding

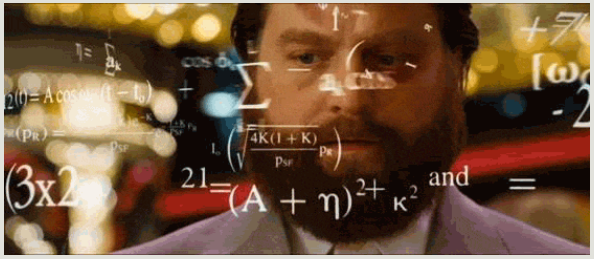
```typescript
import { Component } from '@angular/core';

@Component({
  selector: 'inter-interpolation',
  template : `interpolation.html` ,
  styles: []
})
export class InterpolationComponent {
  nom:string ='Aymen Sellaouti';
  age:number =35;
  adresse:string ='Chez moi ou autre part :)';
  getName(){
    return this.nom;
  }
  modifier(newName){
    this.nom=newName;
  }
}
```

Component

```html
<hr>
Nom : {{nom}}<br>
Age : {{age}}<br>
Adresse : {{adresse}}<br>
//Property Binding
<input #name
[value]="getName()">
//Event Binding
<button
(click)="modifier(name.value)"
>Modifier le nom</button>
<hr>
```

Template

Exercice



- Créer un nouveau composant. Ajouter y un Div et un input de type texte.
- Fait en sorte que lorsqu'on écrive une couleur dans l'input, ca devienne la couleur du Div.
- Ajouter un bouton. Au click sur le bouton, il faudra que le Div reprenne sa couleur par défaut.
- Ps : pour accéder au contenu d'un élément du Dom utiliser #nom dans la balise et accéder ensuite à cette propriété via le nom.
- Pour accéder à une propriété de style d'un élément on peut binder la propriété `[style.nomPropriété]` exemple `[style.background-color]`

Two way Binding

- Binding Bi-directionnel
- Permet d'interagir du Dom ver le composant et du composant vers le DOM.
- Se fait à travers la directive **ngModel** (on reviendra sur le concept de directive plus en détail)
- Syntaxe :
  - **[(ngModel)]=property**
- Afin de pouvoir utiliser ngModel vous devez importer le FormsModule dans app.module.ts

Property Binding et Event Binding

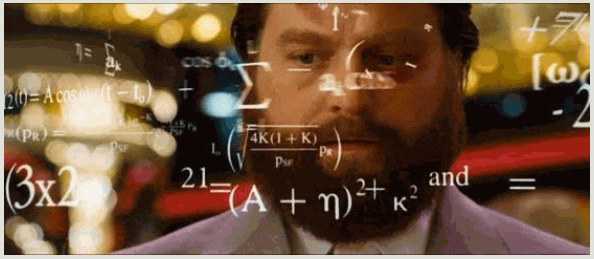
```
import { Component} from
'@angular/core';

@Component({
  selector: 'app-two-way',
  templateUrl: './two-
way.component.html',
  styleUrls: ['./two-
way.component.css']
})
export class TwoWayComponent  {
  two:any="myInitValue";
}
```

Component

```
<hr>
Change me if u can
<input
[(ngModel)]="two">
<br>
it's always me :d
{{two}}
```

Template

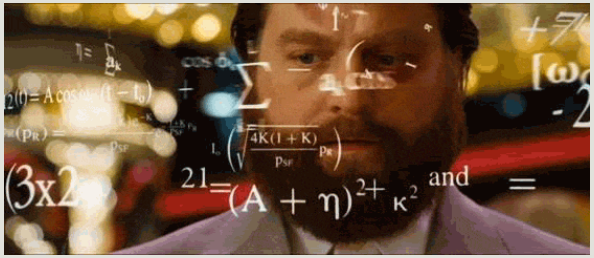
Exercice



- Le but de cet exercice est de créer un aperçu de carte visite.
- Créer un composant
- Préparer une interface permettant de saisir d'un coté les données à insérer dans une carte visite. De l'autre coté et instantanément les données de la carte seront mis à jour.
- Préparer l'affichage de la carte visite. Vous pouvez utiliser ce thème gratuit :

https://www.creative-tim.com/product/rotating-css-card

Exercice



Two Way Binding

Nom :
Sellaouti

Prénom :
aymen

Job :
Enseignant

image :
as.jpg

Citation Favorite :
tant qu'il y a de la vie il y a de l'espoir

Décrivez nous votre travail :
J enseigne aux étudiants les technos du Web

Mots clé de votre travail :
HTML CSS JS PHP Symfony Angular

Two Way Binding

Nom :
Sellaouti

Prénom :
aymen

Job :
Enseignant

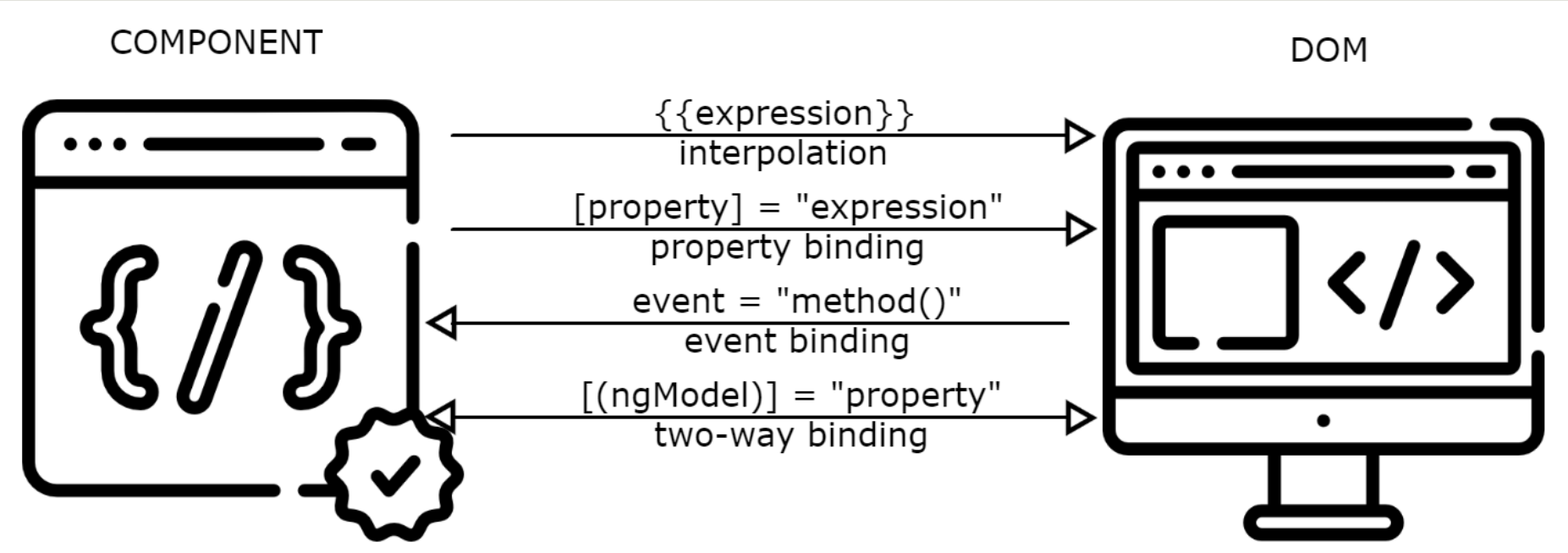
image :
as.jpg

Citation Favorite :
tant qu'il y a de la vie il y a de l'espoir

Décrivez nous votre travail :
J enseigne aux étudiants les technos du Web

Mots clé de votre travail :
HTML CSS JS PHP Symfony Angular

Résumé : Property Binding



Cycle de vie d'un composant

- Le composant possède un cycle de vie géré par Angular. En effet Angular :
  - Crée le composant
  - L'affiche
  - Crée ses fils
  - Les affiche
  - Ecoute le changement des propriétés
  - Le détruit avant de l'enlever du DOM

https://medium.com/bhargav-bachina-angular-training/angular-understanding-angular-lifecycle-hooks-with-a-sample-project-375a61882478

Comment ca marche réellement

Exemple :

```
<mon-app>
    <compo-fils [binded]='prop'>
```

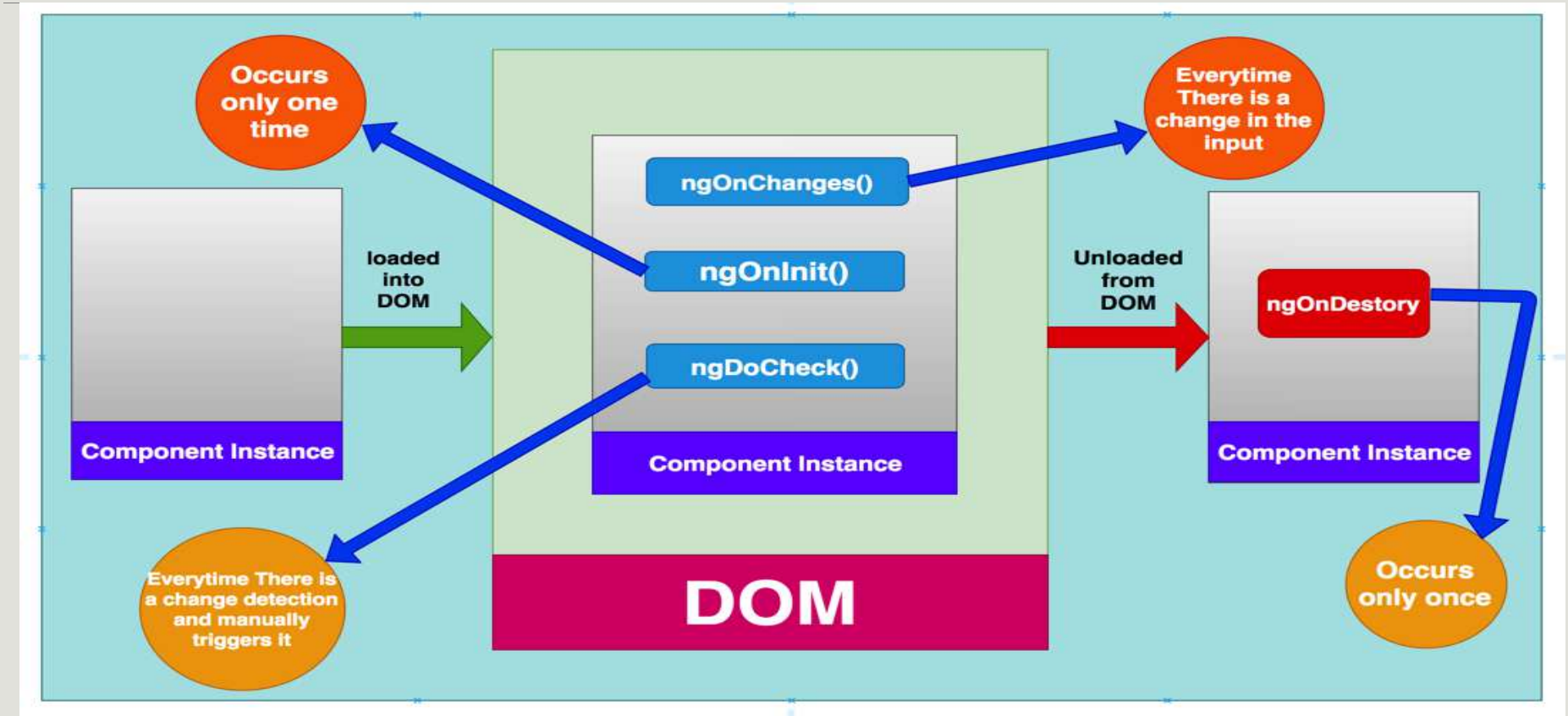
1. Angular lance l'application
2. Ils crée les classes pour chaque composant, il lance donc le Constructeur de mon-app component.
3. Il gère toutes les dépendances injectées au niveau du constructeur et les définit comme des paramètres
4. Il crée le nœud du Dom qui va héberger le composant
5. Il commence ensuite la création du composant fils et appelle son constructeur. Ici la propriété binded n'est pas prise en considération par Angular. Le Life Hook cycle n'est pas encore déclenché.
6. A ce moment Angular lance le processus de détection des changements. C'est ici qu'il mettra à jour le binding de mon-App et lancera l'appel à ngOnInit, suivi de la gestion du binding de compo-fils puis l'appel de son ngOnInit.

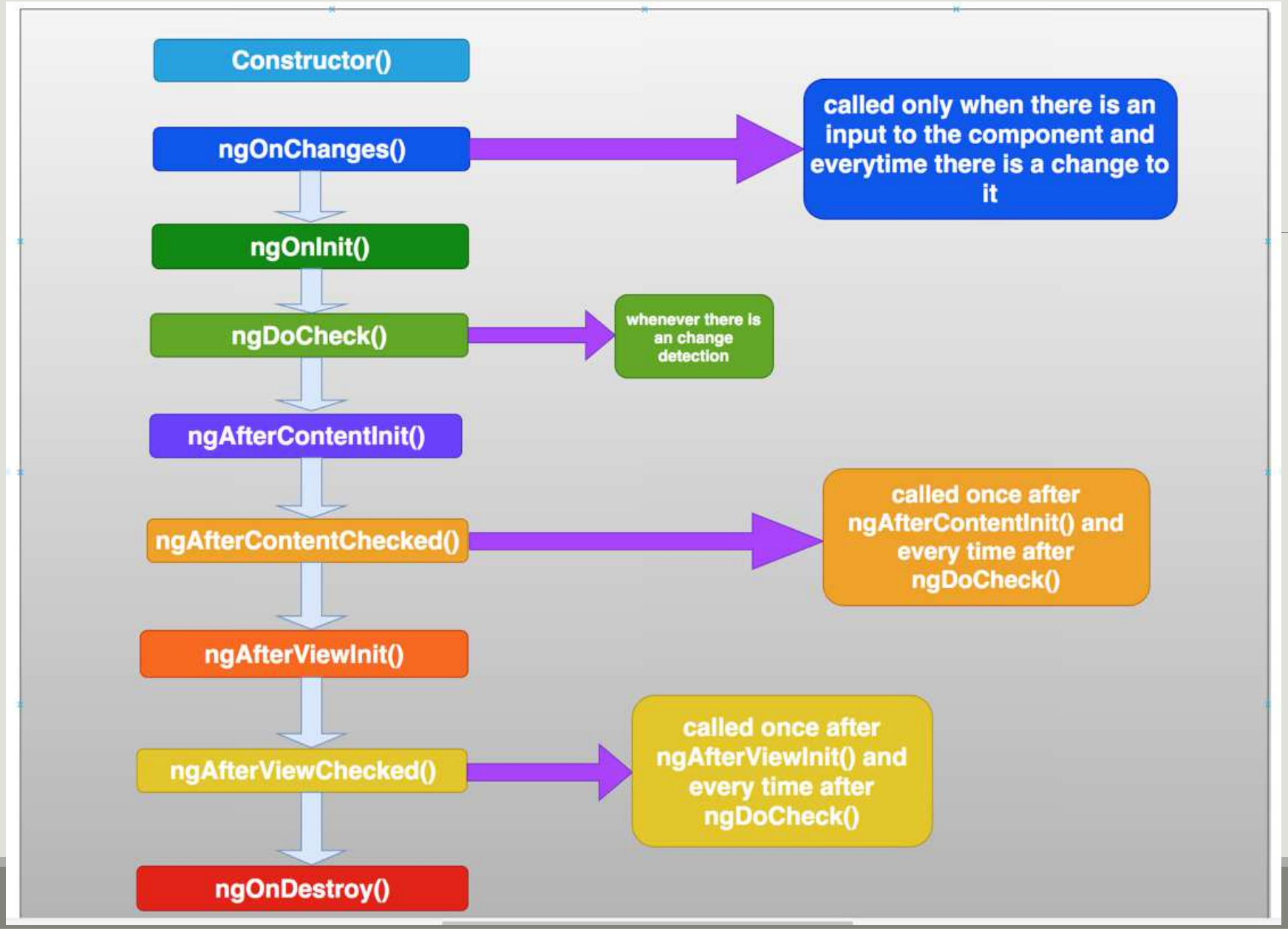
Interfaces de gestion du cycle de vie d'un composant

- Afin de gérer le cycle de vie d'un composant, Angular nous offre un ensemble d'interfaces à implémenter pour les différentes étapes du cycle de vie.
- L'ensemble des interfaces est disponible dans la librairie core d'Angular
- Chaque interface offre une seule méthode dont le nom suit la convention suivante :

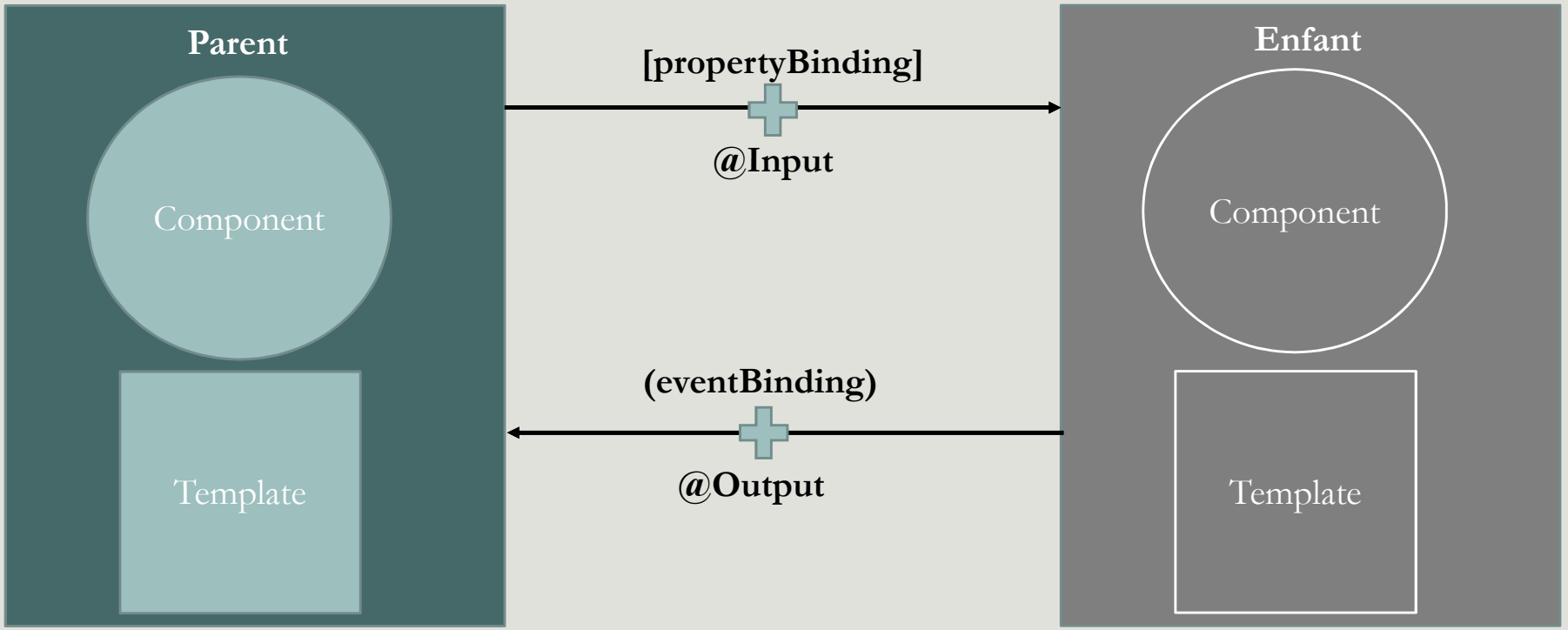
ng + NomDeL'Interface

Exemple : Pour l'interface OnInit nous avons la méthode ngOnInit


Occurs only one time
loaded into DOM
Component Instance
ngOnChanges()
ngOnInit()
ngDoCheck()
Component Instance
DOM
Everytime There is a change detection and manually triggers it
Everytime There is a change in the input
Unloaded from DOM
ngOnDestory
Component Instance
Occurs only once

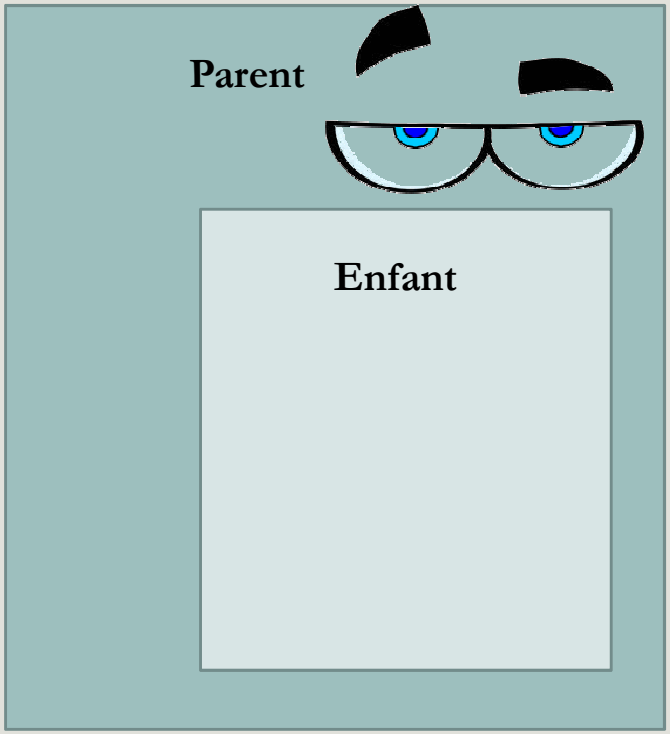

Constructor()
ngOnChanges()
called only when there is an input to the component and everytime there is a change to it
ngOnInit()
ngDoCheck()
whenever there is an change detection
ngAfterContentInit()
ngAfterContentChecked()
called once after ngAfterContentInit() and every time after ngDoCheck()
ngAfterViewInit()
ngAfterViewChecked()
called once after ngAfterViewInit() and every time after ngDoCheck()
ngOnDestroy()

Interaction entre composants



Pourquoi ?

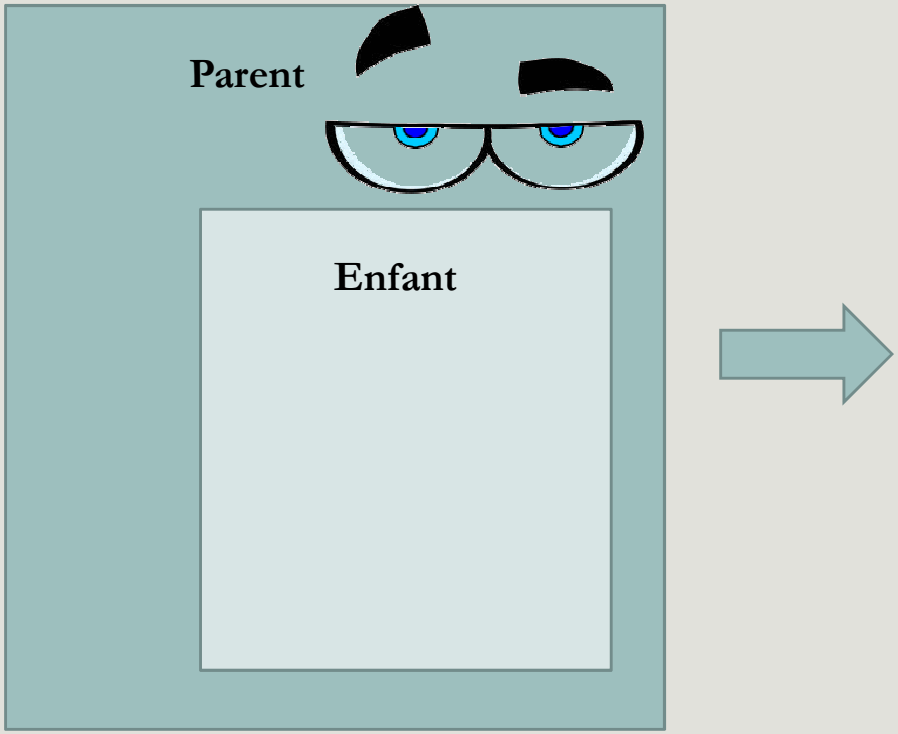
Le père voit le fils, le fils ne voit pas le père




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

@Component({
  selector: 'app-root',
  template: `
    <p>Je suis le composant père </p>
    <forma-fils></forma-fils>
`,
  styles:
})
export class AppComponent {
  title = 'app works !';
}
```

Interaction du père vers le fils

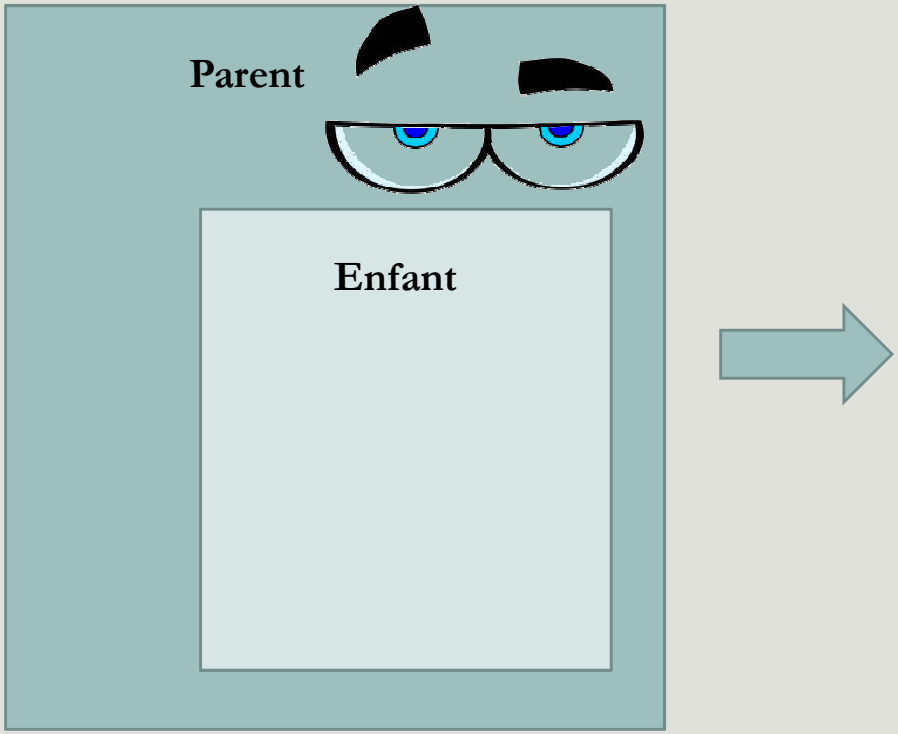
Le père peut directement envoyer au fils des données par Property Binding




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

@Component({
  selector: 'app-root',
  template: `
    <p>Je suis le composant père </p>
    <forma-fils></forma-fils>
`,
  styles:
})
export class AppComponent {
  title = 'app works !';
}
```

Interaction du père vers le fils

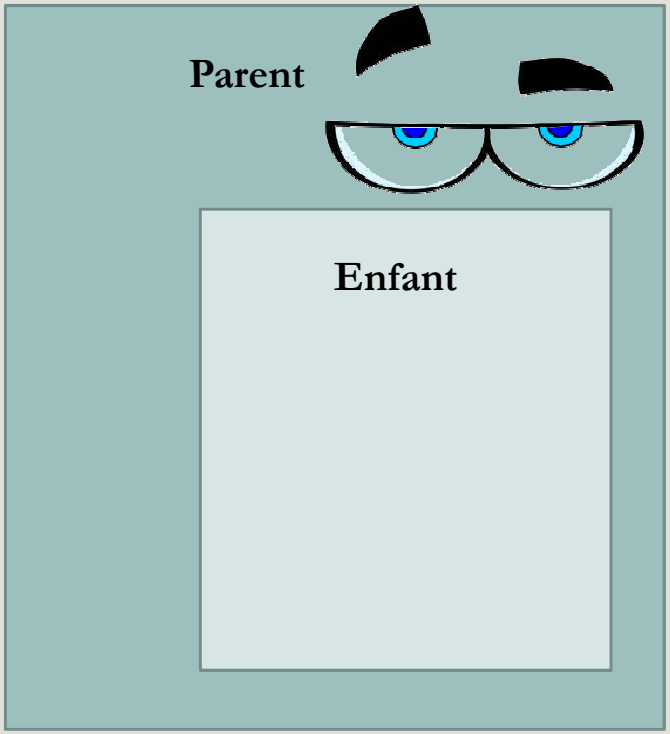
Problème : Le père voit le fils mais pas ces propriétés !!! **Solution : les rendre visible avec Input**




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

@Component({
  selector: 'app-root',
  template: `
    <p>Je suis le composant père </p>
    <forma-fils></forma-fils>
`,
  styles:
})
export class AppComponent {
  title = 'app works !';
}
```

Interaction du père vers le fils

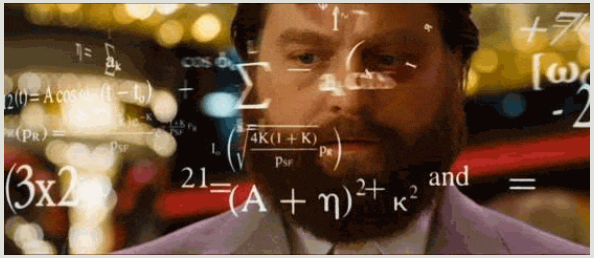
Problème : Le père voit le fils mais pas ces propriétés !!! **Solution : les rendre visible avec Input**




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

@Component({
  selector: 'app-root',
  template: `
    <p>Je suis le composant père </p>
    <forma-fils [external]="title">
</forma-fils>
`,
  styles:
})
export class AppComponent {
  title = 'app works !';
}
```

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

@Component({
  selector: 'app-input',
  templateUrl:
'./input.component.html',
  styleUrls:
['./input.component.css']
})
export class InputComponent
{
  @Input() external:string;
}
```

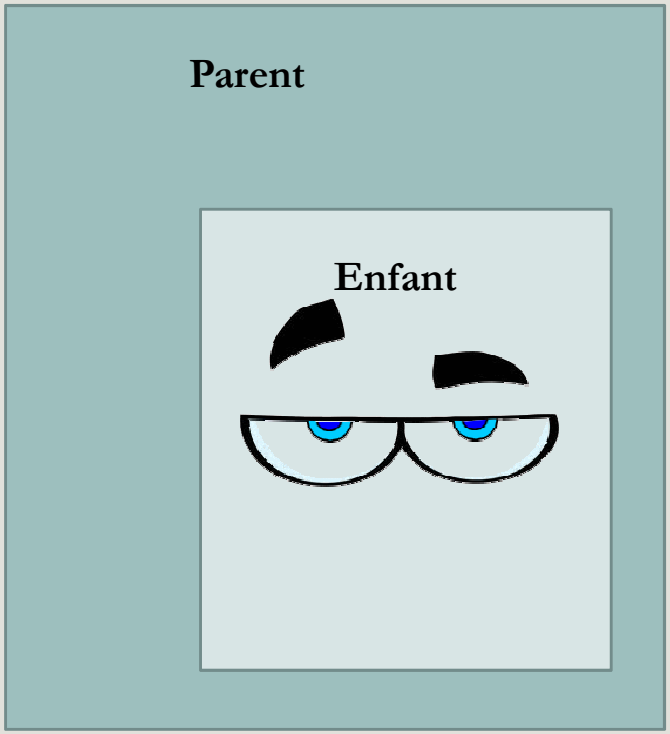
Exercice



- Créer un composant fils
- Récupérer la couleur du père dans le composant fils
- Faite en sorte que le composant fils affiche la couleur du background de son père

Interaction du fils vers le père

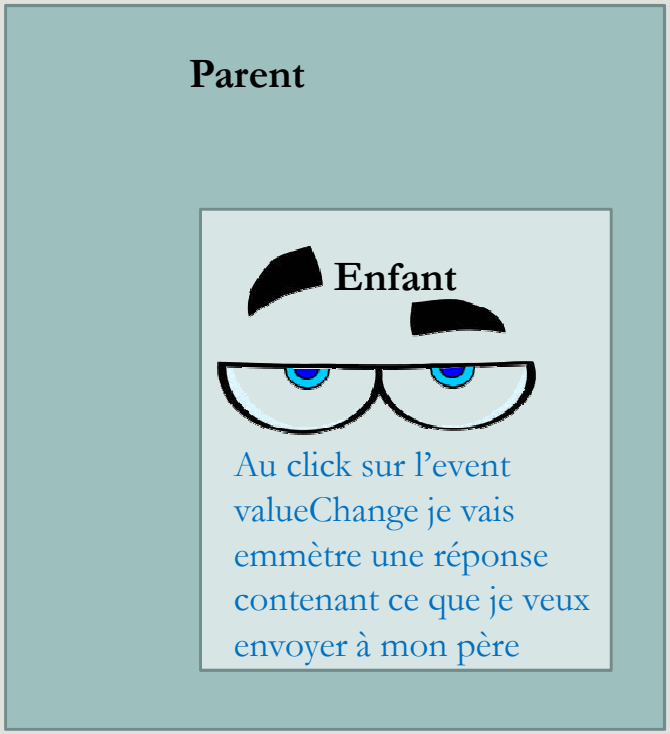
L'enfant ne peut pas voir le parent. Appel de l'enfant vers le parent. Que faire ?




```
import {Component} from '@angular/core';
@Component({
  selector: 'bind-output',
  template: `Bonjour je suis le fils`,
  styleUrls: ['./output.component.css']
})
export class OutputComponent {
  valeur:number=0;
  }
}
```

Interaction du fils vers le père

Solution : Pour entrer c'est un input pour sortir c'est surement un output. Externaliser un évènement en utilisant l'Event Binding.




```
import {Component, EventEmitter, Output} from
'@angular/core';
@Component({
  selector: 'bind-output',
  template: `
<button (click)="incrementer()">+</button> `,
  styleUrls: ['./output.component.css']
})
export class OutputComponent {
  valeur:number=0;
  // On déclare un evenement
  @Output() valueChange=new EventEmitter();
  incrementer(){
    this.valeur++;
    this.valueChange.emit
       this.valeur);
}
}}
```

Interaction du père vers le fils

La variable $event est la variable utilisée pour faire transiter les informations.

Parent

Mon père va ensuite intercepter l'event et récupérer ce que je lui ai envoyé à travers la variable $event et va l'utiliser comme il veut

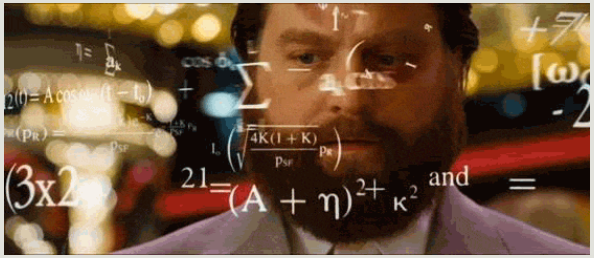
```
import {Component, EventEmitter, Output} from
'@angular/core';
@Component({
  selector: 'bind-output',
  template: `
<button (click)="incrementer()">+</button> `,
  styleUrls: ['./output.component.css']
})
export class OutputComponent {
  valeur:number=0;
  // On déclare un evenement
  @Output() valueChange=new EventEmitter()
  incrementer(){
    this.valeur++;
    this.valueChange.emit(
       this.valeur
    );
}
}}
```

Enfant

```
import { Component } from
'@angular/core';
@Component({
  selector: 'app-root',
  template: `
    <h2> {{result}}</h2>
<bind-output
(valueChange)="showValue($event)
"></bind-output>
 ,
  styles:  [``],
})
export class AppComponent {
  title = 'app works !';
  result:any='N/A';
  showValue(value){
    this.result=value;
  }
}
```

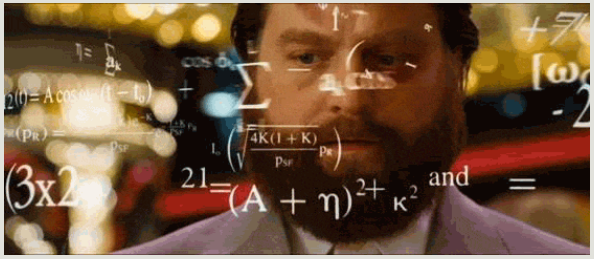
Parent

Exercice



- Ajouter une variable myFavoriteColor dans le composant du fils.
- Ajouter un bouton dans le composant Fils
- Au click sur ce bouton, la couleur de background du Div du père doit prendre la couleur favorite du fils.

Exercice

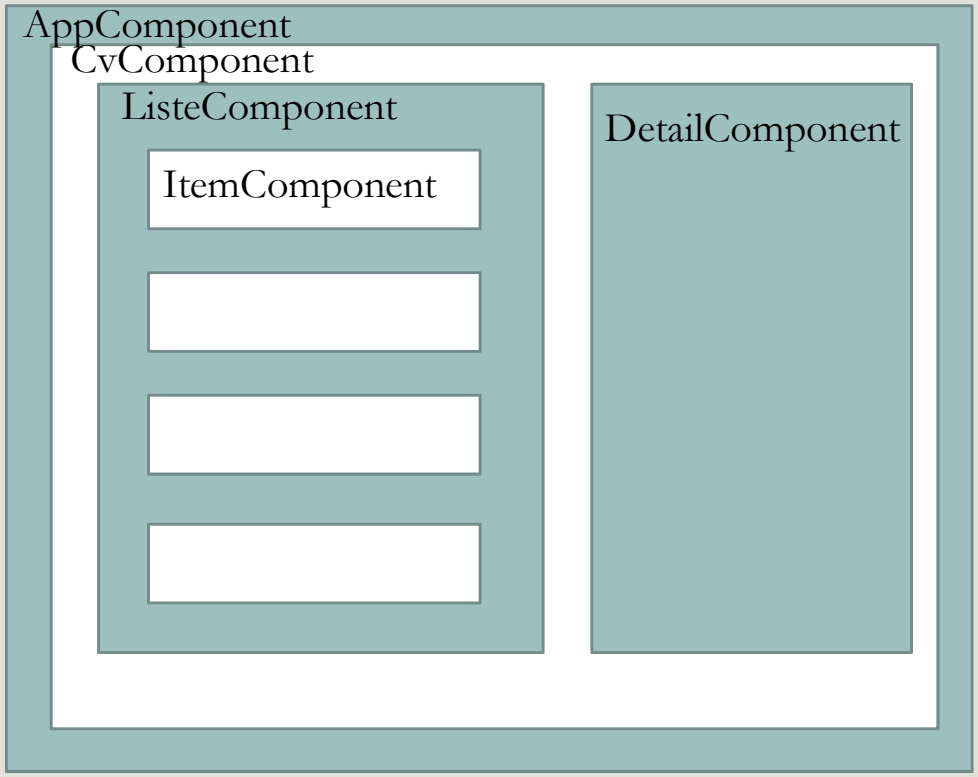


- Le but de cet exercice est de créer une mini plateforme de recrutement.

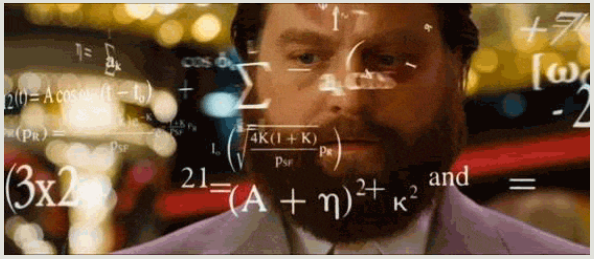
- La première étape est de créer la structure suivante avec une vue contenant deux parties :
  - Liste des Cvs inscrits
  - Détail du Cv qui apparaitra au click

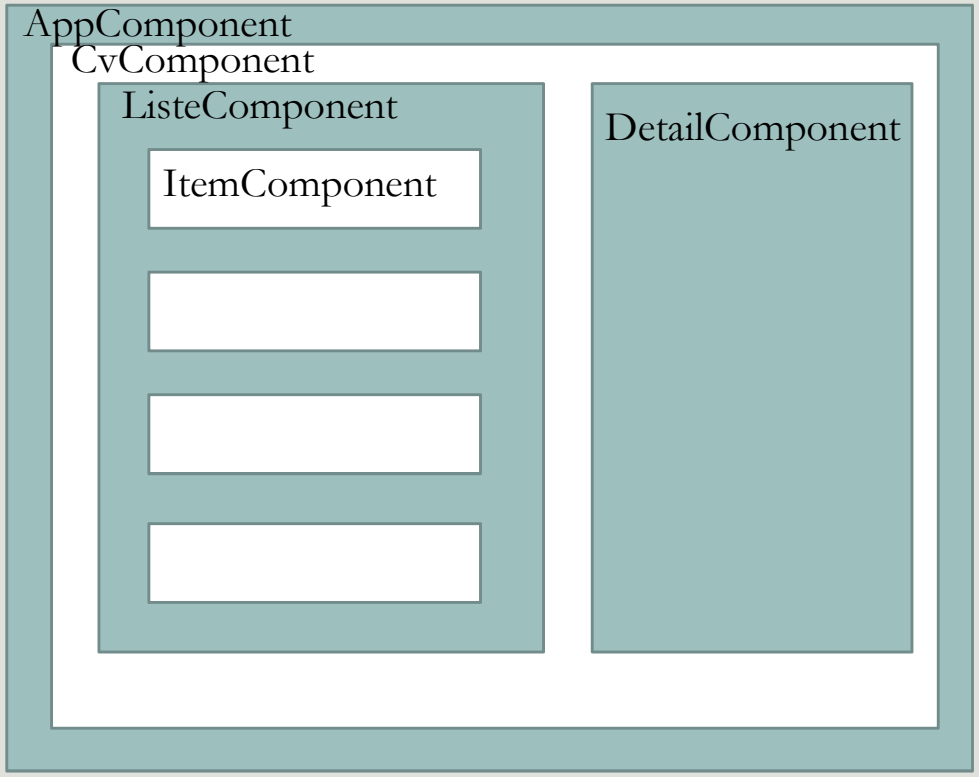
- Il vous est demandé juste d'afficher un seul Cv et de lui afficher les détails au click.

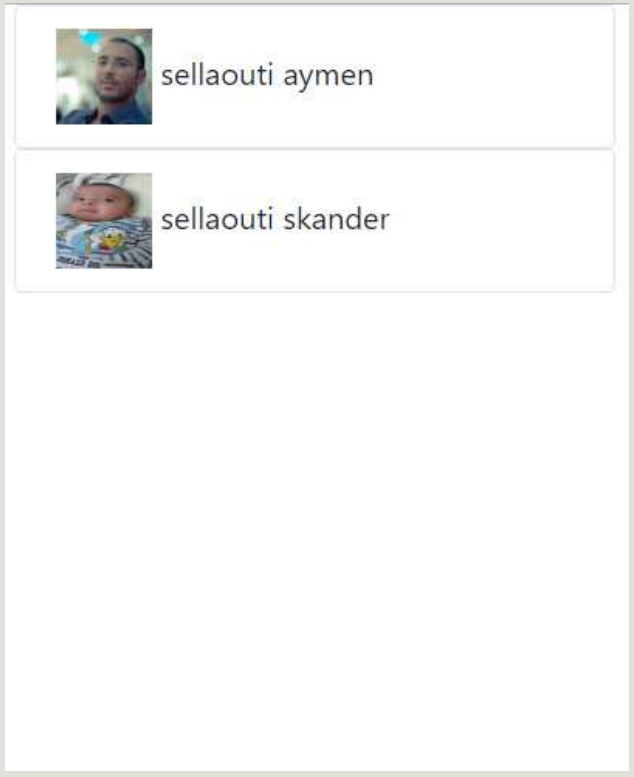
Il faudra suivra cette architecture.



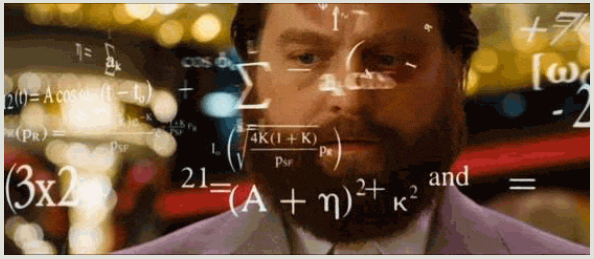
Exercice

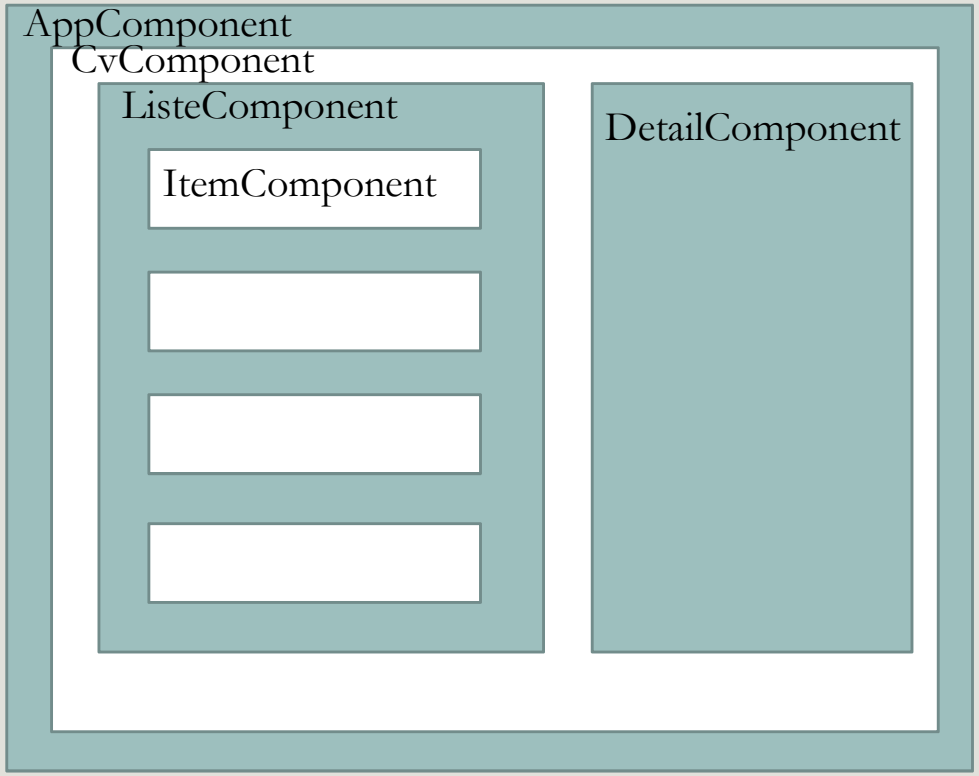



AppComponent
CvComponent
ListeComponent
ItemComponent
DetailComponent


sellaouti aymen
sellaouti skander

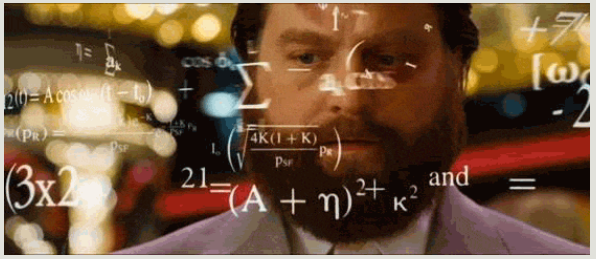
Exercice





Au click sur le Cv les détails sont affichés

Exercice



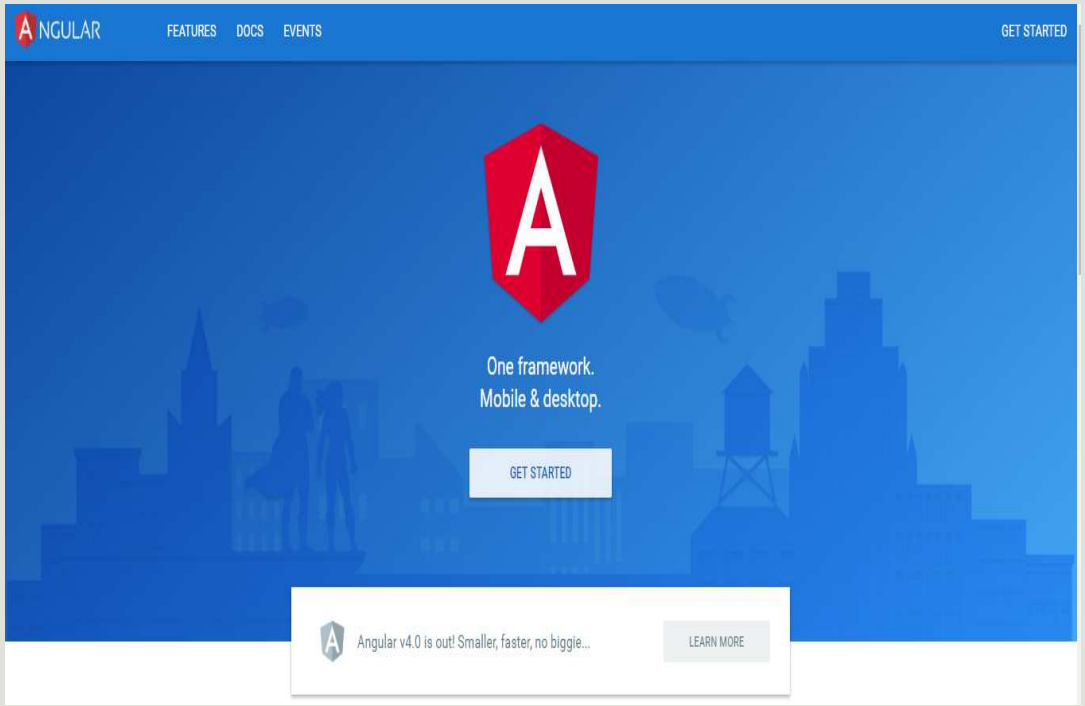
Un cv est caractérisé par :
id
Nom
Prenom
Age
Cin
Job
path

Au click sur le Cv les détails sont affichés

Angular
Les directives

AYMEN SELLAOUTI

Références


ANGULAR
FEATURES
DOCS
EVENTS
GET STARTED
One framework.
Mobile & desktop.
GET STARTED
Angular v4.0 is out! Smaller, faster, no biggie...
LEARN MORE


Learn Angular
Gravissez tous les concepts jusqu'au sommet
Maîtriser les concepts du Framework Angular
pour développer des applications robustes
Auteur : William KOZA

Plan du Cours

1. Introduction
2. Les composants
3. Les directives

3 Bis. Les pipes

4. Service et injection de dépendances
5. Le routage
6. Form
7. HTTP

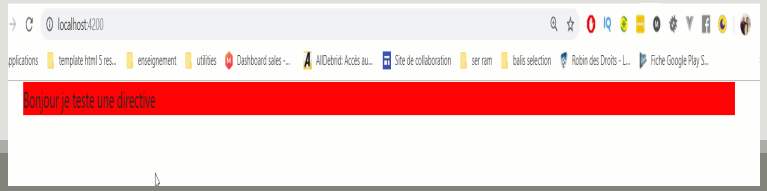
Objectifs

1. Comprendre la définition et l'intérêt des directives.
2. Voir quelques directives d'attributs offertes par angular et savoir les utiliser
3. Créer votre propre directive d'attributs
4. Voir quelques directives structurelles offertes par angular et savoir les utiliser

Qu'est ce qu'une directive

- Une **directive** est une **classe** permettant **d'attacher** un **comportement** aux **éléments** du **DOM.** Elle est décorée avec l'annotation **@Directive.**
- Apparait dans un élément comme un **tag** (comme le font les **attributs**).
- La command pour créer une directive est
  - **ng g d nomDirective**

```
import {Directive, HostBinding, HostListener}
from '@angular/core';
@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
  @HostBinding('style.backgroundColor') bg = '';
  constructor() { }
  @HostListener('mouseenter') mouseenter() {
    this.bg = 'yellow';
  }
  @HostListener('mouseleave') mouseleave() {
    this.bg = 'red';
  }
}
```




```
<div appHighlight>
  Bonjour je teste une directive
</div>
```

Qu'est ce qu'une directive

- La documentation officielle d'Angular identifie trois types de directives :
  - Les **composants** qui sont des directives avec des templates.
  - Les **directives structurelles** qui changent **l'apparence** du **DOM** en ajoutant et supprimant des éléments.
  - Les **directives d'attributs** (attribute directives) qui permettent de changer **l'apparence** ou le **comportement** d'un **élément.**

Les directives d'attribut (ngStyle)

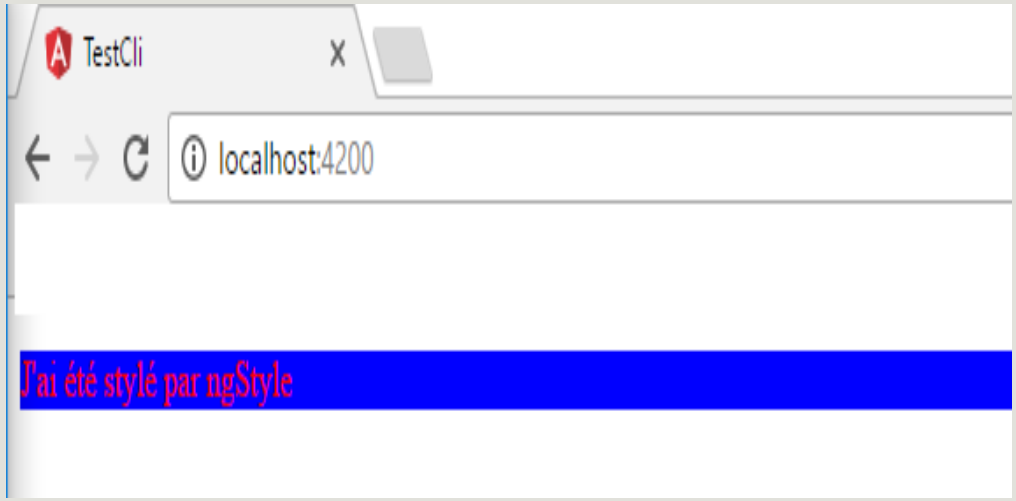
- Cette directive permet de modifier **l'apparence** de **l'élément cible**.
- Elle est placé entre [ ] **[ngStyle]**
- Elle prend en paramètre un **attribut** représentant un **objet** décrivant le **style** à appliquer.
- Elle utilise le **property Binding.**

Les directives d'attribut (ngStyle)

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

@Component({
  selector: 'direct-direct',
  template: `
    <p [ngStyle]="{'color':'red',
       'font-family':'garamond',
       'background-color' : 'yellow'}">
    <ng-content></ng-content>
    </p>
  `,
  styleUrls: ['./direct.component.css']
})
export class DirectComponent{
}
```

```
import { Component} from
'@angular/core';
@Component({
  selector: 'direct-direct',
  template: `
  <p [ngStyle]="{'color':myColor,'font-
family':myfont,'background-color' :
myBackground}">
    <ng-content></ng-content>
    </p>`,
  styleUrls: ['./direct.component.css']
})
export class DirectComponent{
  private myfont:string="garamond";
  private myColor:string="red";
  private myBackground:string="blue"
}
```

Les directives d'attribut (ngStyle)

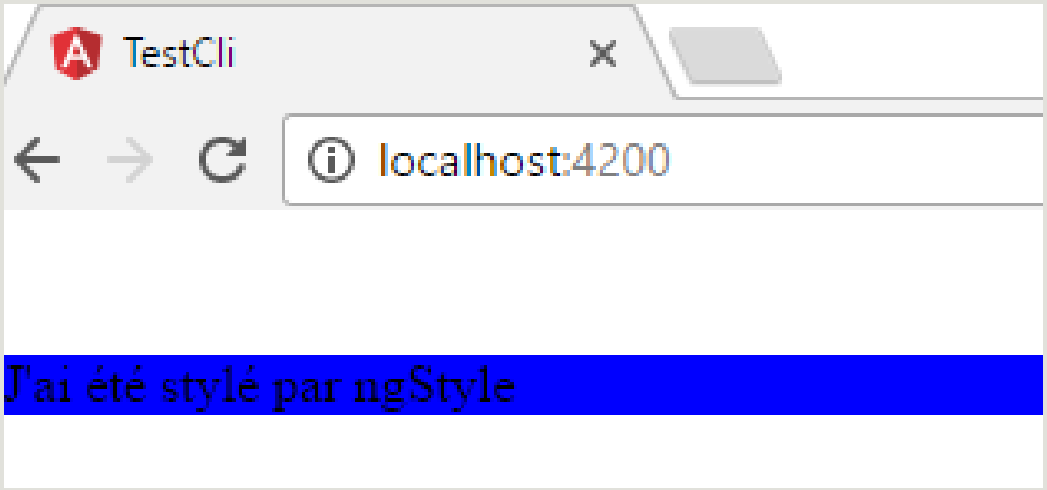


Les directives d'attribut (ngStyle)

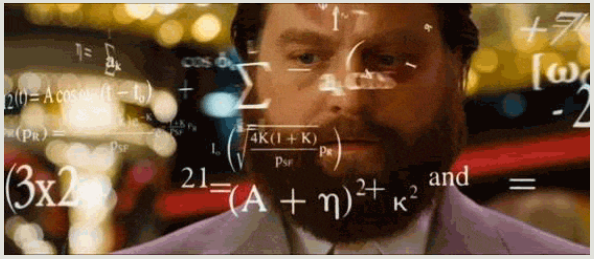
```
@Component({
  selector: 'app-root',
  template: `
    <direct-direct [myColor]="gray">J'ai
été stylé par ngStyle</direct-direct>
  `,
  styles:  [`
    h1 { font-weight: normal; }
    p{color:yellow;background-color: red}
  `],
})
export class AppComponent {

}
```

```
import {Component, Input} from
'@angular/core';
@Component({
  selector: 'direct-direct',
  template: `
    <p [ngStyle]="{'color':myColor,
   'font-familly':myfont,
   'background-color' : myBackground}">
    <ng-content></ng-content>
    </p>
  `,
  styleUrls: ['./direct.component.css']
})
export class DirectComponent{
  private myfont:string="garamond";
  @Input() private myColor:string="red";
  private myBackground:string="blue"
}
```

Les directives d'attribut (ngStyle)



Exercice



- Nous voulons simuler un Mini Word pour gérer un paragraphe en utilisant ngStyle.
- Préparer un input de type texte, un input de type number, et un select box.
- Faite en sorte que lorsqu'on écrive une couleur dans le texte input, ca devienne la couleur du paragraphe. Et que lorsque on change le nombre dans le number input la taille de l'écriture.
- Finalement ajouter une liste et mettez y un ensemble de police. Lorsque le user sélectionne une police dans la liste, la police dans le paragraphe change.

Les directives d'attribut (ngClass)

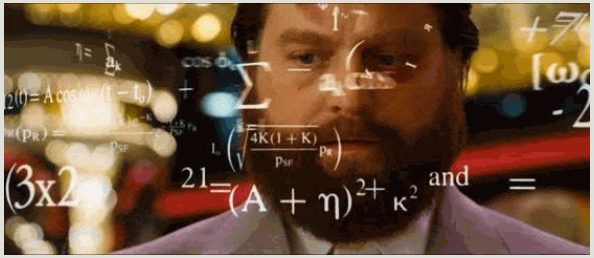
- Cette directive permet de modifier **l'attribut class**.
- Elle cohabite avec l'attribut **class**.
- Elle prend en paramètre
  - Une chaine (string)
  - Un tableau (dans ce cas il faut ajouter les [ ] donc [ngClass]
  - Un objet (dans ce cas il faut ajouter les [ ] donc [ngClass]
- Elle utilise le **property Binding.**

Les directives d'attribut (ngClass)

```
import {Component, Input} from '@angular/core';
@Component({
  selector: 'direct-direct',
  template: `

    <div ngClass="colorer arrierplan"  class="encadrer">
      test ngClass
    </div>
  `,
  styles: [`
    .encadrer{  border: inset 3px black;  }
    .colorer{   color: blueviolet; }
    .arrierplan{background-color: salmon; }
  `]
})
export class DirectComponent{
  private myfont:string="garamond";
  @Input()  private myColor:string="red";
  private myBackground:string="blue«
  private isColoree:boolean=true;
  private isArrierPlan:boolean=true
}
```

```
// Tableau
<div [ngClass]="['colorer', 'arrierplan'] "
class="encadrer">
// Objet

<div [ngClass]="{ colorer: isColoree,
arrierplan: isArrierPlan} "
class="encadrer">
```

Exercice



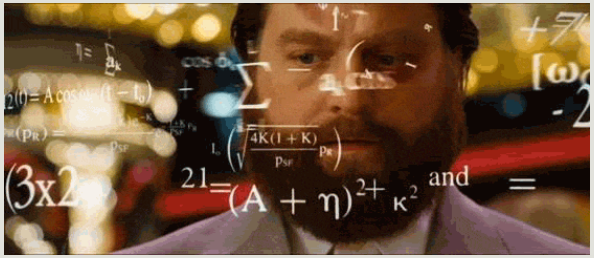
- Préparer 3 classes présentant trois thèmes différents (couleur font-size et font-police)
- Au choix du thème votre cible changera automatiquement

Customiser un attribut directive

- Afin de créer sa propre « attribut directive » il faut utiliser un `HostBinding` sur la **propriété** que vous voulez **binder**.
  - **Exemple :**
```
@HostBinding('style.backgroundColor')
bg:string="red";
```

- Si on veut associer un **événement** à notre directive on utilise un `HostListner` qu'on associe à un **événement** déclenchant une `méthode.`
  - **Exemple :**
```
@HostListener('mouseenter') mouseover() {
    this.bg =this.highlightColor;
}
```
- Afin d'utiliser le HostBinding et le HostListner il faut les importer du core d'angular

Exercice



- Créer une directive appelée exergue
- Cette directive devra pouvoir faire deux choses. A l'entrée de la cible, elle devra modifier la couleur du background. A la sortie, elle devra récupérer la valeur par défaut.

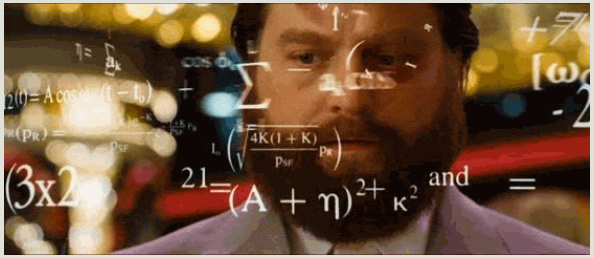
Un truc plus sympas on va créer un simulateur d'écriture arc en ciel.

- Créer une directive
- Créer un hostbinding sur la couleur et la couleur de la bordure.
- Créer un tableau de couleur dans votre directive.
- Faite en sorte qu'en appliquant votre directive à un input, à chaque écriture d'une lettre (event keydown) la couleur change en prenant aléatoirement l'une des couleurs de votre tableau. Pensez à utiliser Math.random() qui vous retourne une valeur entre 0 et 1.

Customiser une attribut directive

- Nous pouvons aussi utiliser le @Input afin de rendre notre directive paramétrable
- Tous les paramètres de la directive peuvent être mises en @Input puis récupérer à partir de la cible.
- Exemple
  - Dans la directive `@Input()` `private myColor:string="red";`
  - `<direct-direct [myColor]="gray">`

Exercice



- Reprenez la directive exergue et faite en sorte qu'elle permette aussi à l'utilisateur de personnaliser ces paramètres ou d'utiliser les paramètres offerts par la directive.

Les directives structurelles

- Une **directive** structurelle permet de modifier le DOM.
- Elles sont appliquées sur **l'élément HOST**.
- Elles sont généralement précédées par le **préfix *.**
- Les directives les plus connues sont :
  - *ngIf
  - *ngFor
  - [ngSwitch]

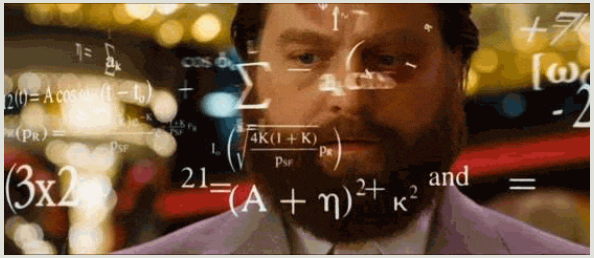
Les directives structurelles *ngIf

- Prend un booléen en paramètre.
- Si le booléen est true alors l'élément host est visible
- Si le booléen est false alors l'élément host est caché

Exemple

```
<p *ngIf="true">
  Je suis visible :D</p>
<p *ngIf="false">
  Le *ngIf c'est faché contre
moi et m'a caché :(
</p>
```

Exercice



- Teston *ngIf en créant un composant contenant un bouton et un paragraphe.
- Le bouton s'appellera 'Click moi'
- Un paragraphe contenant une phrase
- Au click, si le paragraphe est caché on l'affiche, s'il est affiché, on le cache

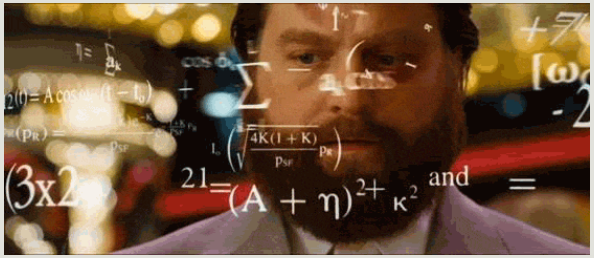
Les directives structurelles *ngFor

- Permet de répéter un élément plusieurs fois dans le DOM.
- Prend en paramètre les entités à reproduire.
- Fournit certaines valeurs :
  - index : position de l'élément courant
  - first : vrai si premier élément
  - last vrai si dernier élément
  - even : vrai si l'indice est paire
  - odd : vrai si l'indice est impaire

```
<ul>
  <li *ngFor="let episode of episodes">
  {{episode.title}}
  </li>
</ul>
```

```
<ul>
  <li  *ngFor="let episode of episodes; let i = index;
let isOdd = odd; let isFirst=first"
       [ngClass]="{ odd: isOdd , bgfonce: isFirst}"
  >
    Episode {{i+1}}{{episode.title}}
  </li>
</ul>
```

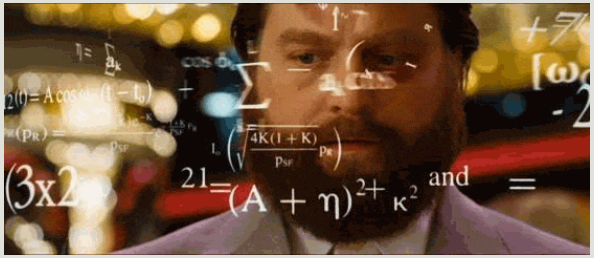
Exercice



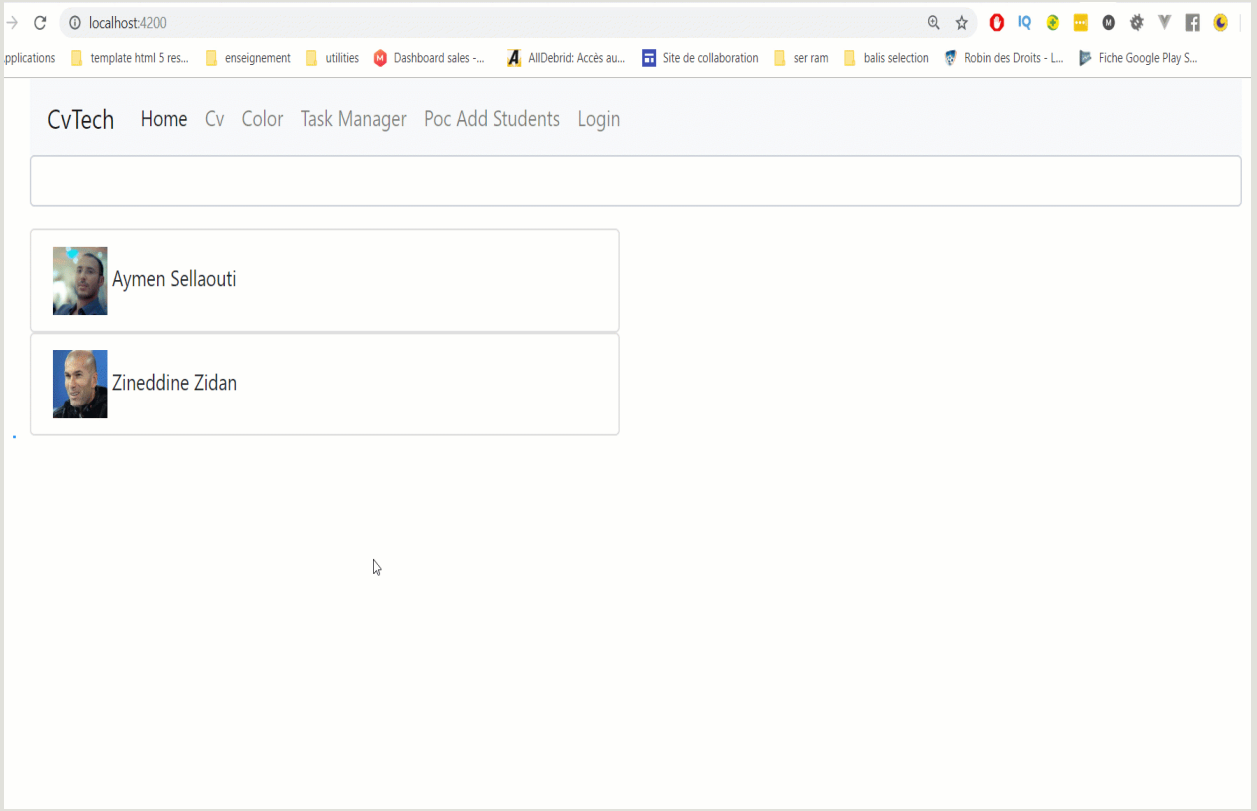
- Teston *ngFor en créant un composant contenant un tableau d'objets personnes. Chaque personne est caractérisée par son nom, prénom, âge et métier.

- Utiliser *ngFor pour afficher la liste de ces personnes.

Exercice (Notre Projet)



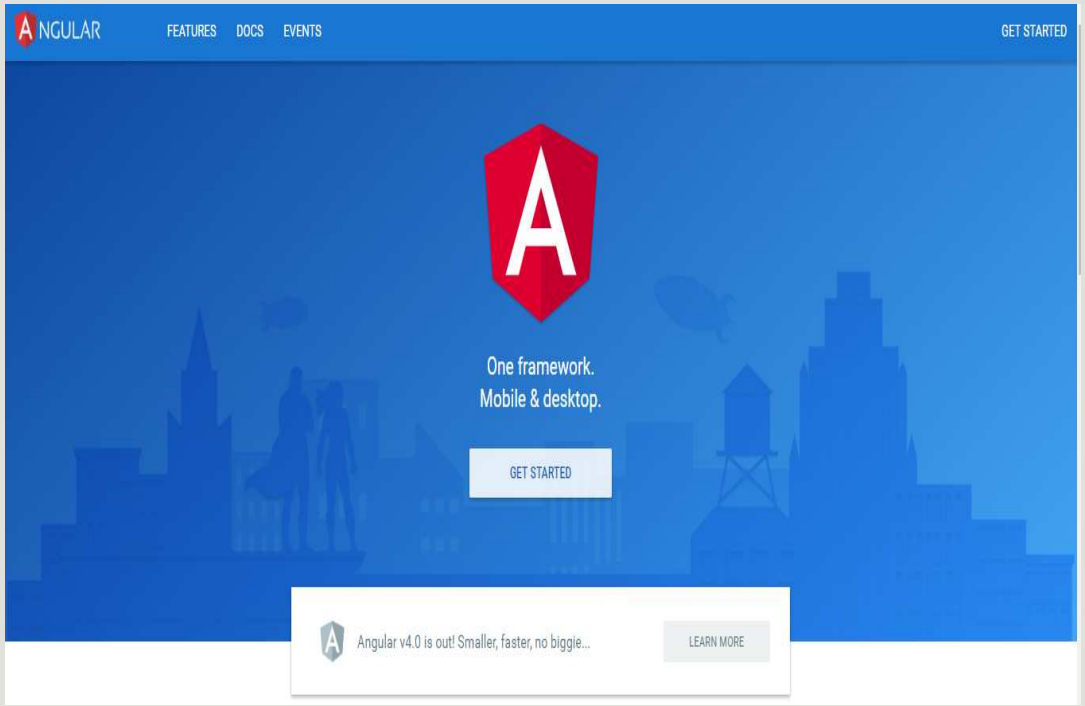
- Reprenons notre plateforme d'embauche.
- Utilisez les directives vues dans ce cours pour afficher une liste de Cv et pour améliorer l'affichage.
- Les détails ne sont affichés qu'au click sur un des cvs.



Angular
Les pipes

AYMEN SELLAOUTI

Références


ANGULAR
FEATURES
DOCS
EVENTS
GET STARTED
One framework.
Mobile & desktop.
GET STARTED
Angular v4.0 is out! Smaller, faster, no biggie...
LEARN MORE


Learn Angular
Gravissez tous les concepts jusqu'au sommet
Maîtriser les concepts du Framework Angular
pour développer des applications robustes
Auteur : William KOZA

Plan du Cours

1. Introduction
2. Les composants
3. Les directives

3 Bis. Les pipes

4. Service et injection de dépendances
5. Le routage
6. Form
7. HTTP

Objectifs

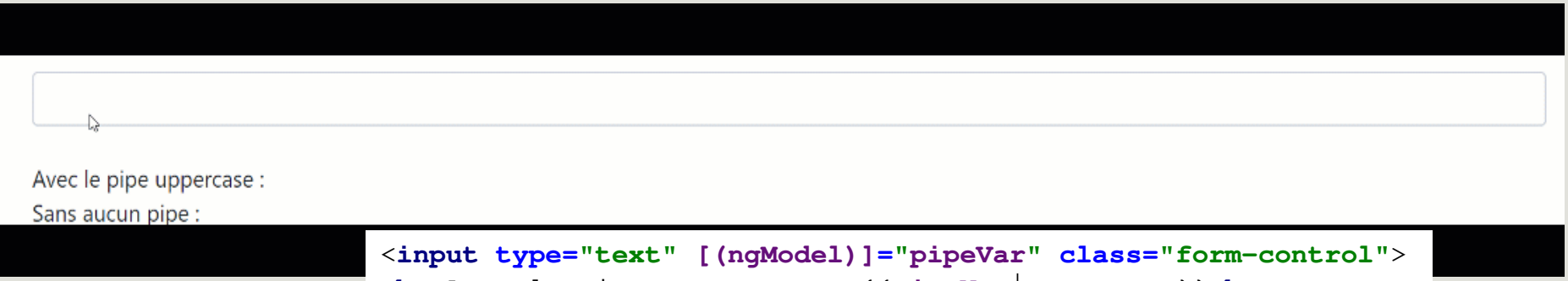
1. Définir les pipes et l'intérêt de les utiliser

2. Vue globale des pipes prédéfinies

3. Créer un pipe personnalisé

Qu'est ce qu'un pipe

- Un pipe est une fonctionnalité qui permet de formater et de transformer vos données avant de les afficher dans vos Templates.
- Exemple l'affichage d'une date selon un certain format.
- Il existe des pipes offerts par Angular et prêt à l'emploi.
- Vous pouvez créer vos propres pipes.




```
<input type="text" [(ngModel)]="pipeVar" class="form-control">
<br>Avec le pipe uppercase : {{pipeVar|uppercase}}<br>
Sans aucun pipe : {{pipeVar}}
```

Syntaxe

- Afin d'utiliser un pipe vous utilisez la syntaxe suivante :
  - {{ variable | **nomDuPipe** }}
- Exemple : {{ maDate | **date** }}
- Afin d'utiliser plusieurs pipes combinés vous utilisez la syntaxe suivante :
  - {{ variable | **nomDuPipe1** | **nomDuPipe2** | **nomDuPipe3** }}
- Exemple : {{ maDate | **date** | **uppercase** }}

Les pipes disponibles par défaut (Built-in pipes)

- La documentation d'angular vous offre la liste des pipes prêt à l'emploi.

https://angular.io/api?type=pipe

- uppercase
- lowercase
- titlecase
- currency
- date
- json
- percent
- …

Paramétrer un pipe

- Afin de paramétrer les pipes ajouter ‘:’ après le pipe suivi de votre paramètre.
  - {{ maDate | date:"MM/dd/yy" }}
- Si vous avez plusieurs paramètres c’est une suite de ‘:’
  - {{ nom | slice:1:4 }}

Pipe personnalisé

- Un pipe personnalisé est une classe décoré avec le décorateur @Pipe.
- Elle implémente l'interface PipeTransform
- Elle doit implémenter la méthode transform qui prend en paramètre la valeur cible ainsi qu'un ensemble d'options.
- La méthode transform doit retourner la valeur transformée
- Le pipe doit être déclaré au niveau de votre module de la même manière qu'une directive ou un composant.
- Pout créer un pipe avec le cli : ng g p nomPipe

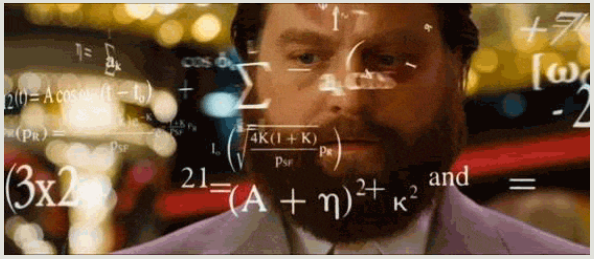
Exemple de pipe

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

@Pipe({
  name: 'team'
})
export class TeamPipe implements PipeTransform {

  transform(value: any, args?: any): any {
    switch (value) {
      case 'barca' : return ' blaugrana';
      case 'roma' : return ' giallorossa';
      case 'milan' : return ' rossoneri';
    }
  }

}
```

```
<li>
  <ol *ngFor="let team of
teams">
    {{team | team}}
  </ol>
</li>
```

```
ngOnInit() {
this.teams = ['milan', 'barca', 'roma'];
}
```

Exercice

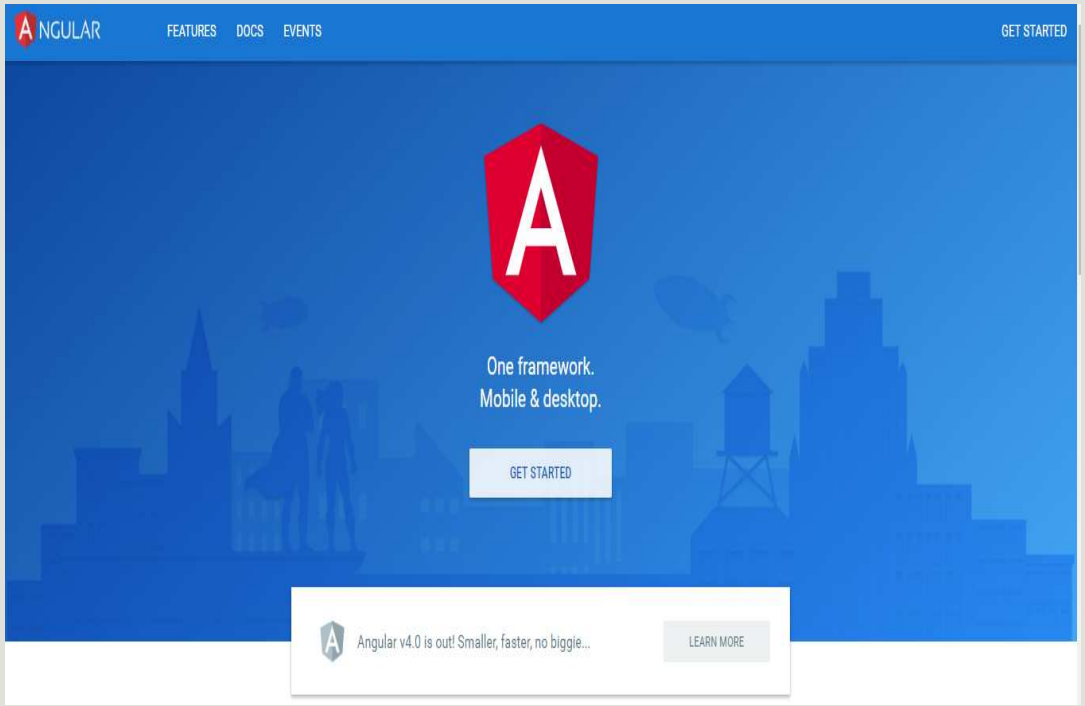


Créer un pipe appelé defaultImage qui retourne le nom d'une image par défaut que vous stockerez dans vos assets au cas ou la valeur fournie au pipe est une chaine vide.

Angular
Service et injection de dépendances

AYMEN SELLAOUTI

Références


ANGULAR
FEATURES
DOCS
EVENTS
GET STARTED
One framework.
Mobile & desktop.
GET STARTED
Angular v4.0 is out! Smaller, faster, no biggie...
LEARN MORE


Learn Angular
Gravissez tous les concepts jusqu'au sommet
Maîtriser les concepts du Framework Angular
pour développer des applications robustes
Auteur : William KOZA

Plan du Cours

1. Introduction
2. Les composants
3. Les directives

3 Bis. Les pipes

4. Service et injection de dépendances
5. Le routage
6. Form
7. HTTP

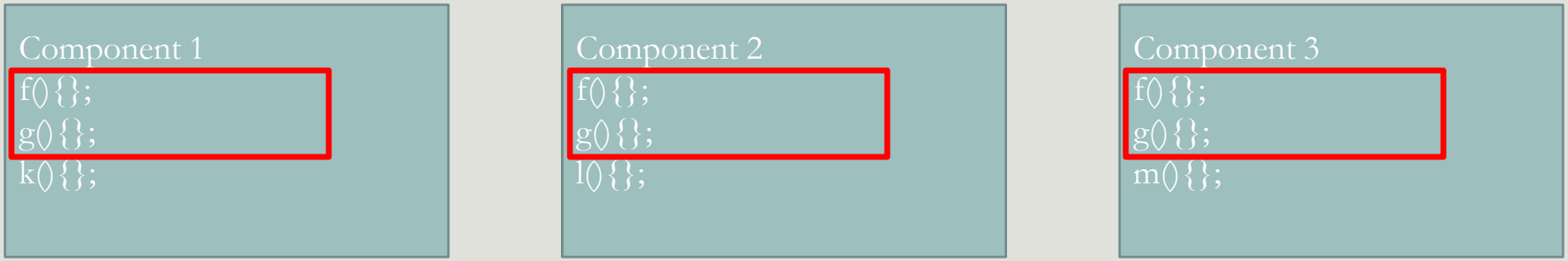
Objectifs

1. Définir un service
2. Définir ce qu'est l'injection de dépendance
3. Injecter un service
4. Définir la portée d'un service
5. Réordonner son code en utilisant les services

Qu'est ce qu'un service ?

- Un service est une classe qui permet d'exécuter un traitement.

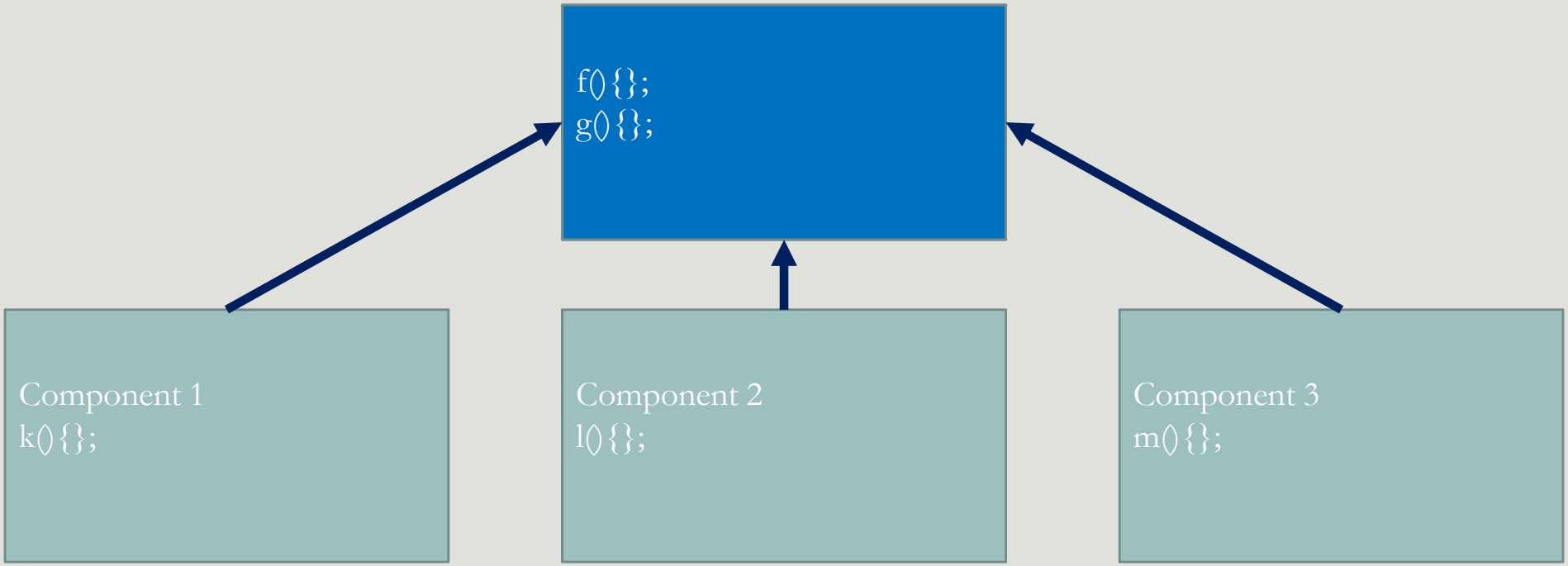
- Permet d'encapsuler des fonctionnalités redondantes permettant ainsi d'éviter la redondance de code.



Redondance de code

Maintenabilité difficile

Qu'est ce qu'un service ?



Qu'est ce qu'un service ?

- Un service est un médiateur entre la vue et la logique
- Ce qui n'est pas trivial doit être écrit sous forme d'un composant
- Un service est associé à un composant en utilisant l'injection de dépendance

Qu'est ce qu'un service ?

- Un service peut donc :
  - Interagir avec les données (fournit, supprime et modifie)
  - Interaction entre classes et composants
  - Tout traitement métier (calcul, tri, extraction …)

Création d'un service

- Via CLI
  - ng generate service nomDuService
  - ng g s nomDuService

Premier Service

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

@Injectable()
export class FirstService {

  constructor() { }

}
```

```
//…Other import
import {FirstService} from
"./first.service";
@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule
  ],
  providers: [FirstService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Injection de dépendance (DI)

- L'injection de dépendance est un patron de conception.

```
Classe A1{
 ClasseB b;
 ClasseC c;
 …
}
```

```
Classe A2{
 ClasseB b;
…
}
```

```
Classe A3{
ClasseC c;
 …
}
```

Que se passera t-il si on change quelque chose dans le constructeur de B ou C ?
Qui va modifier l'instanciation de ces classes dans les différentes classes qui en dépendent?

Injection de dépendance (DI)

- Déléguer cette tache à une entité tierce.

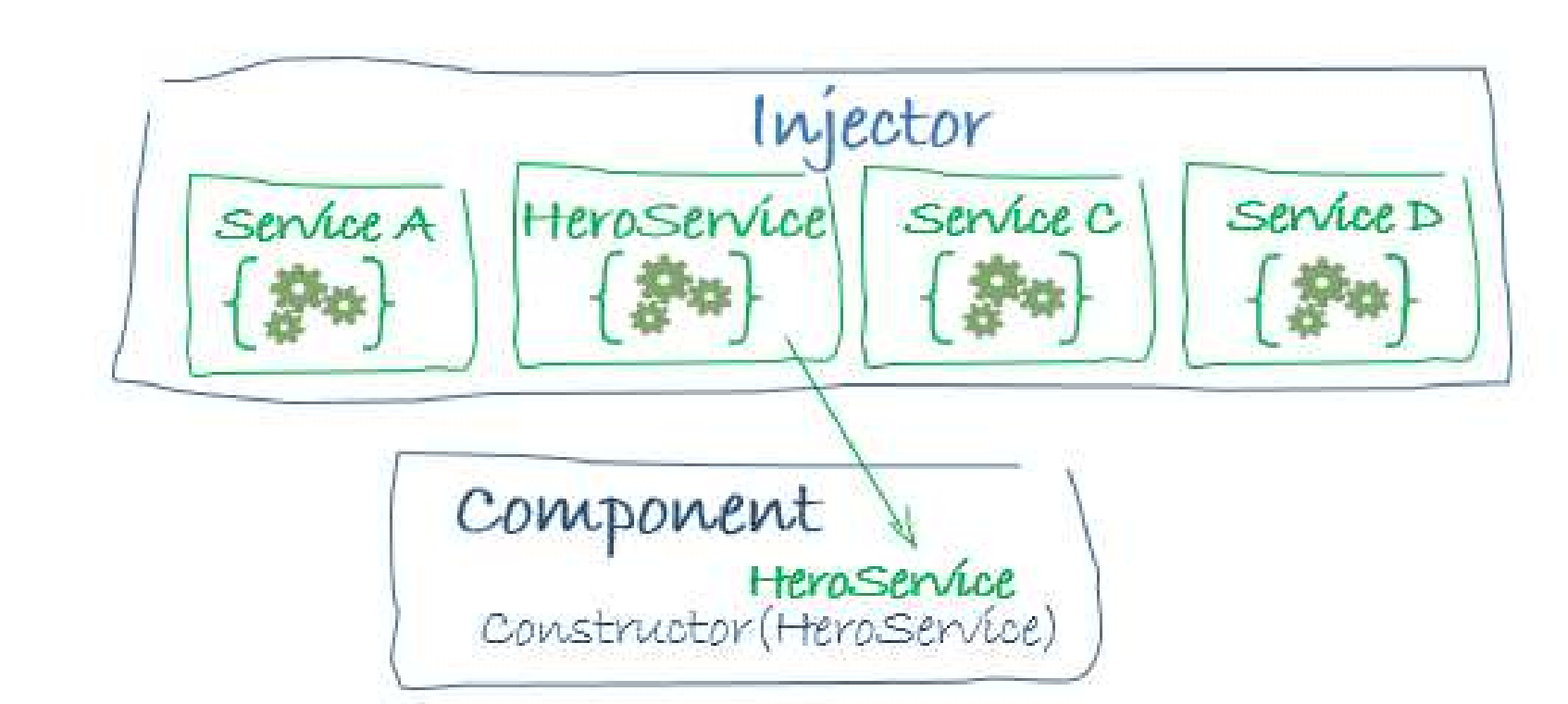
```
Classe A1{
 Constructor(B b, C c)
 …
}
```

```
Classe A2{
Constructor(B b)
…
}
```

```
Classe A3{
Constructor(C c)
 …
}
```

INJECTOR

Injection de dépendance (DI)



Injection de dépendance (DI)

- Comment les injecter ?
- Comment spécifier à l'injecteur quel service et ou est-il visible ?

Injection de dépendance (DI)

- L'injection de dépendance utilise les étapes suivantes :
  - Déclarer le service dans le provider du module ou du composant
  - Passer le service comme paramètre du constructeur de l'entité qui en a besoin.

Injection de dépendance (DI)

```
import { BrowserModule, } from '@angular/platform-browser';
import {CUSTOM_ELEMENTS_SCHEMA, NgModule} from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';
import { AppComponent } from './app.component';
import {CvService} from "./cv.service";
@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule
  ],
  providers: [CvService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

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

@Injectable(
export class CvService {

  constructor() { }

}
```

Injection de dépendance (DI)

```
import { Component, OnInit } from '@angular/core';
import {Cv} from './cv';
import {CvService} from "../cv.service";
@Component({
  selector: 'app-cv',
  templateUrl: './cv.component.html',
  styleUrls: ['./cv.component.css'],
  providers:[CvService] // on peut aussi l'importer ici
})
export class CvComponent implements OnInit {
  selectedCv : Cv;
  constructor(private monPremierService:CvService) { }
  ngOnInit() {
  }
}
```

Chargement automatique du service

- A partir de Angular 6 vous pouvez ne plus utiliser le provider du module afin de charger votre service mais le faire directement au niveau du service à travers l'annotation @Injectable et sa propriété providedIn. Vous pouvez charger le service dans toute l'application via le mot clé root.
- Si vous voulez charger le service dans un module particulier vous l'importer et vous le mettez à la place de 'root'.

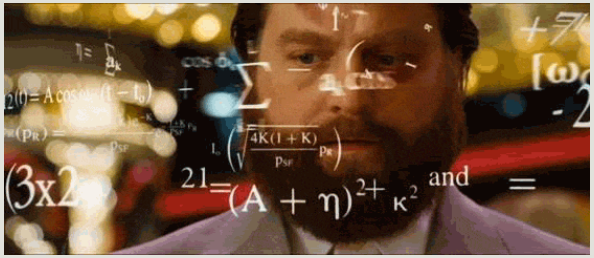
```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class CvService {
  constructor() { }
}
```

Avantage de l'utilisation du providedIn

- Lazy loading : Ne charger le code des services qu'à la première injection

- Permettre le **Tree-Shaking** des services non utilisés : Si le service n'est jamais utilisé, son code ne sera entièrement retiré du build final.

Exercice



- Créons un service de Todo, le Model et le composant qui va avec. Un Todo est caractérisé par un nom et un contenu.
- Ce service permettra de faire les fonctionnalités suivantes :
  - Loger un Todo
  - Ajouter un Todo
  - Récupérer la liste des Todos
  - Supprimer un Todo

@Injectable

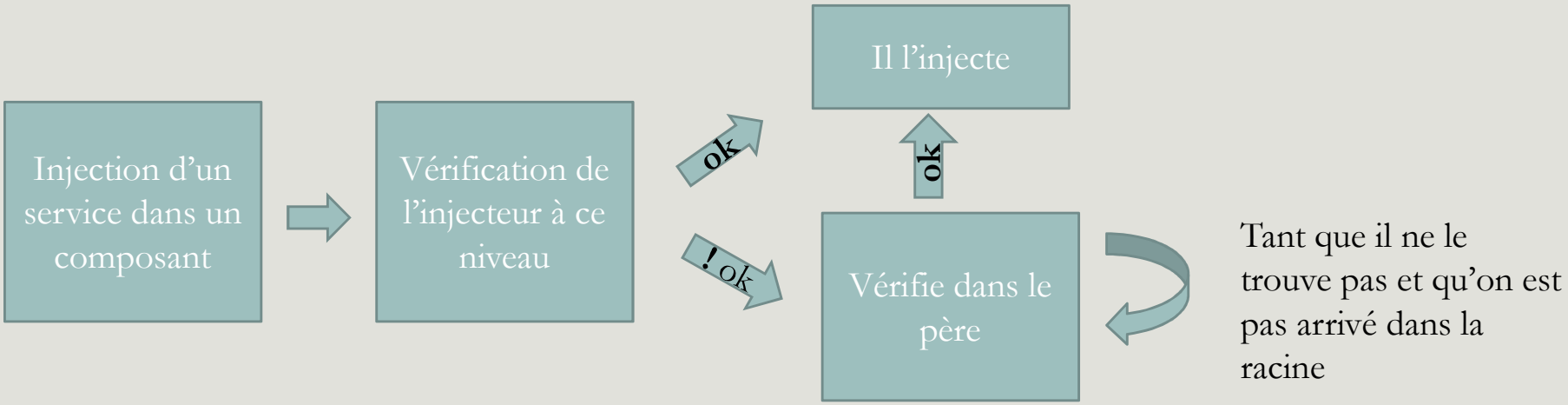
- C'est un décorateur permettant de rendre une classe injectable
- Une classe est dite injectable si on peut y injecter des dépendances
- @Component, @Pipe, et @Directive sont des sous classes de @Injectable(), ceci explique le fait qu'on peut y injecter directement des dépendances.
- Si vous n'aller injecter aucun service dans votre service, cette annotation n'est plus nécessaire.
- **Remarque** : Angular conseille de toujours mettre cette annotation.

Exemple

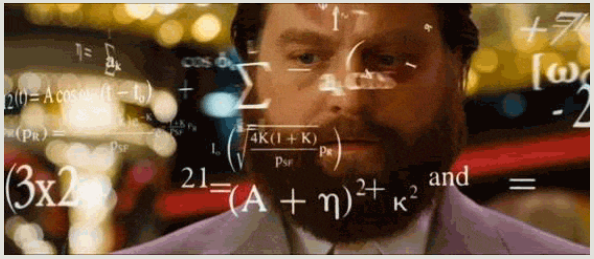
```
import { Injectable } from '@angular/core';
@Injectable()
export class LoggerService {
  constructor() { }
  Logs: string[]=[];
  loger(message:string){
    this.Logs.push(message); console.log(message);
  }
  info(message:string){
    this.Logs.push(message); console.info(message);
  }
  debuger(message:string){
    this.Logs.push(message); console.debug(message);
  }
  avertir(message:string){
    this.Logs.push(message); console.warn(message);
  }
  erreur(message:string){
    this.Logs.push(message); console.error(message);
  }
}
```

DI Hiérarchique

- Le système d'injection de dépendance d'Angular est hiérarchique.
- Un arbre d'injecteur est crée. Cet arbre est // à l'arbre de composant.
- L'algorithme suivant permet la détection de l'injecteur adéquat :



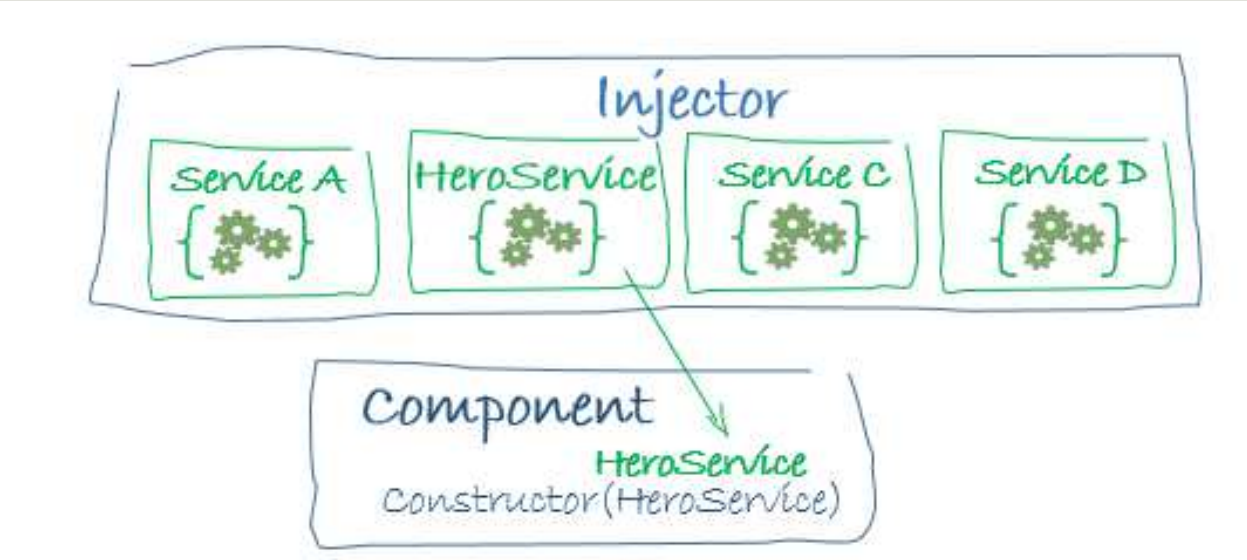
Exercice



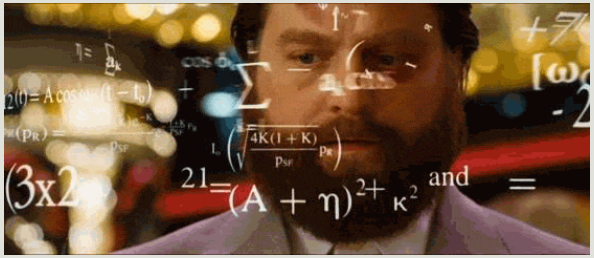
- Reprenez le service Logger en y ajoutant un getter qui retourne le tableau de logs
- Créer deux composants compo1 et compo2
- Injecter y le service logger
- Dans les deux composants créer un input texte et un bouton. Au click sur le bouton il va logger le contenu de l'input.
- Déclarer le directement dans le provider des deux composants.
- Vérifier est ce qu'il travaille sur la même instance du service ou non.

DI Hiérarchique

- Si un service est déclaré au niveau du Module et qu'il est déclaré dans le provider d'un composant c'est la déclaration la plus spécifique qui l'emporte.



Exercice

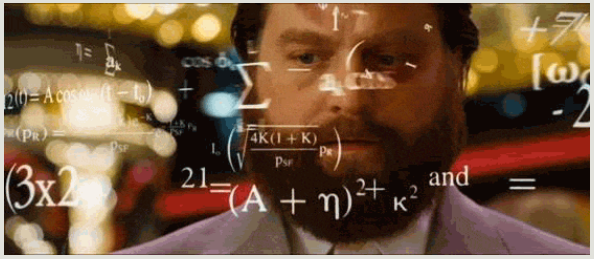


- Ajouter un troisième composant.
- Déclarer le service dans le module
- Enlever le provider du service dans le composant 1
- Garder le dans le composant 2
- Reprenez les mêmes fonctionnalités du composant 1 dans le composant 3
- Testez. Que remarquez vous?

Injecter un service dans un autre

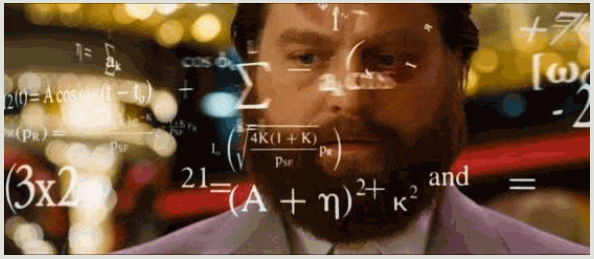
- L'injection d'un service dans un autre est la même que pour un composant.
- Les seuls différences sont :
  - Le service à injecter doit être visible pour le service cible.
  - Le service cible doit obligatoirement avoir la décoration @Injectable.

Exercice



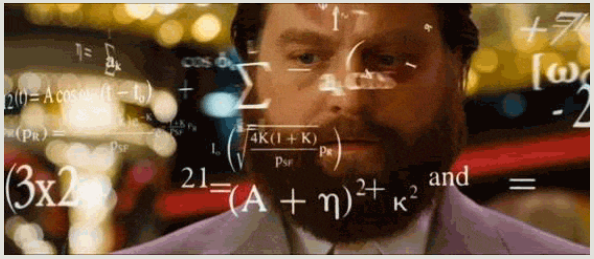
- Créer un nouveau service et utiliser le LoggerService dedans

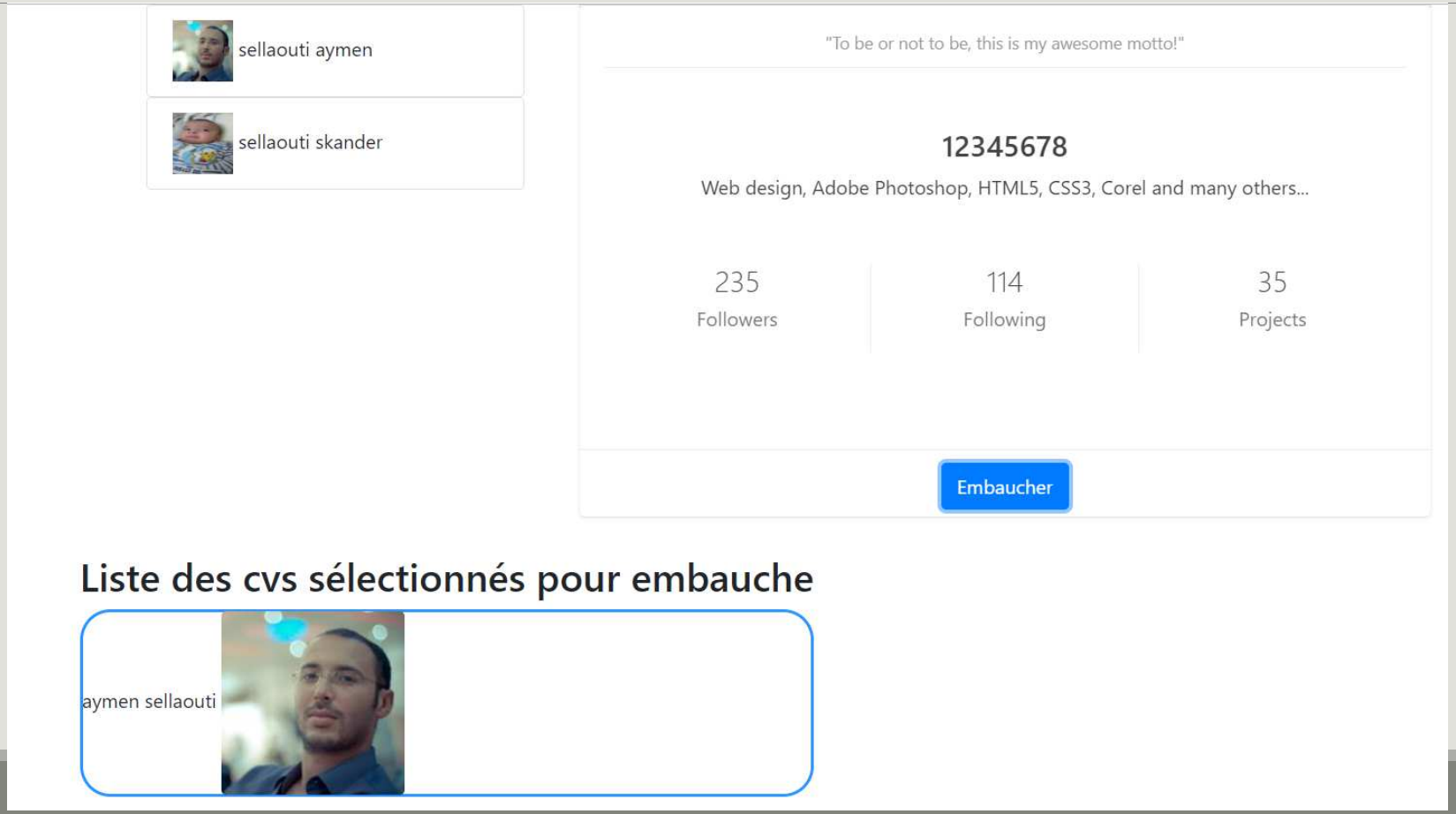
Exercice



- Ajouter les services suivants afin d'améliorer la gestion de notre plateforme d'embauche.
  - Un premier service CvService qui gérera les Cvs. Pour le moment c'est lui qui contiendra la liste des cvs que nous avons.
  - Ajouter aussi un composant pour afficher la liste des cvs embauchées ainsi qu'un service EmbaucheService qui gérer les embauches.
  - Au click sur le bouton embaucher d'un Cv, le cv est ajoutés à la liste des personnes embauchées et une liste des embauchées apparait.

Exercice



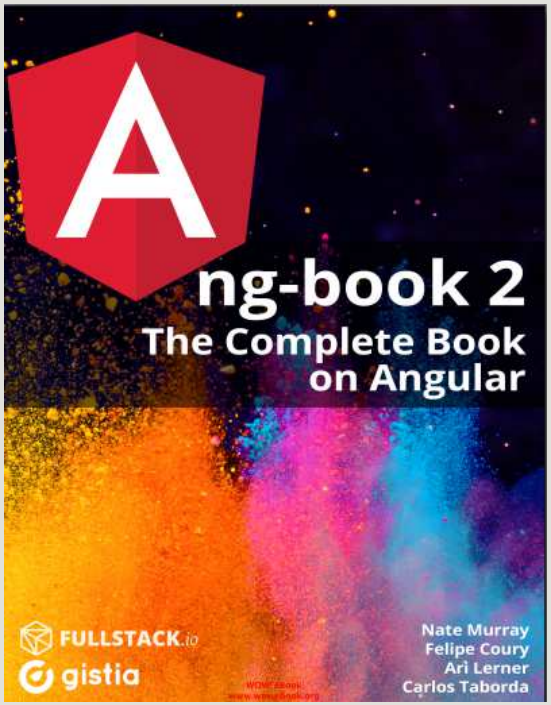

sellaouti aymen
sellaouti skander
"To be or not to be, this is my awesome motto!"
12345678
Web design, Adobe Photoshop, HTML5, CSS3, Corel and many others...
235
Followers
114
Following
35
Projects
Embaucher
Liste des cvs sélectionnés pour embauche
aymen sellaouti

Angular Routing

AYMEN SELLAOUTI

Références


ANGULAR
FEATURES
DOCS
EVENTS
GET STARTED
One framework.
Mobile & desktop.
GET STARTED
Angular v4.0 is out! Smaller, faster, no biggie...
LEARN MORE


ng-book 2
The Complete Book on Angular
FULLSTACK.io
gistia
Nate Murray
Felipe Coury
Ari Lerner
Carlos Taborda


Learn Angular
Gravissez tous les concepts jusqu'au sommet
Maîtriser les concepts du Framework Angular
pour développer des applications robustes
Auteur : William KOZA

Plan du Cours

1. Introduction
2. Les composants
3. Les directives

3 Bis. Les pipes

4. Service et injection de dépendances
5. Le routage
6. Form
7. HTTP

Objectifs

1. Définir le routeur d'Angular
2. Définir une route
3. Déclencher une route à partir d'un composant
4. Ajouter des paramètres à une route
5. Récupérer les paramètres d'une route à partir du composant.
6. Préfixer un ensemble de routes
7. Gérer les routes innexistantes

Qu'est ce que le routing

- Tout système de routing permet d'associer une route à un traitement

- Angular SPA. Pourquoi parle-on de route ??
  - Séparer différentes fonctionnalités du système
  - Maintenir l'état de l'application
  - Ajouter des règles de protection
- Que risque t-on d'avoir si on n'utilise pas un système de routing ?
  - On ne peut plus rafraichir notre page
  - Plus de Favoris ☹
  - Comment partager vos pages ????

Création d'un système de Routing

1. Indiquer au routeur comment composer les urls en ajoutant dans le head la balise suivante : <base href="/">

2. Créer un fichier 'app.routing.ts' Importer le service de routing d'Angular
   - import { RouterModule, Routes } from '@angular/router';
   - Le RouterModule va permettre de configurer les routes dans votre projet
   - Le Routes va permettre de créer les routes

Création d'un système de Routing

1

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Cv</title>
  <base href="/">

  <meta name="viewport" content="width=device-
width, initial-scale=1">
  <link rel="icon" type="image/x-icon"
href="favicon.ico">
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap
/3.3.7/css/bootstrap.min.css"
integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+P
mSTsz/K68vbdEjh4u"
crossorigin="anonymous"></head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

2

```
import {Routes, RouterModule} from
"@angular/router";
import {CvComponent} from "./cv/cv.component";
import {HeaderComponent} from
"./header.component";
```

App.routing.ts

Création d'un système de Routing

3. Créer la constante qui est un tableau d'objet de type Routes représentant chacun la route à décrire.

4. Intégrer les routes à notre application dans le app module à travers le RouterModule et sa méthode forRoot

Création d'un système de Routing

2

```
import {Route, RouterModule} from
"@angular/router";
import {CvComponent} from "./cv/cv.component";
import {HeaderComponent} from
"./header.component";

const APP_ROUTES : Routes = [
  {path: '',component:CvComponent},
  {path:'onlyHeader',component:HeaderComponent}
];

export const ROUTING =
RouterModule.forRoot(APP_ROUTES);
```

3

4

App.routing.ts

```
import { BrowserModule, } from
'@angular/platform-browser';
import {CUSTOM_ELEMENTS_SCHEMA, NgModule} from
'@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';
import { AppComponent } from './app.component';
import {routing} from "./app.routing";
@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    ROUTING
  ],
  providers: [CvService,EmbaucheService],
  schemas: [CUSTOM_ELEMENTS_SCHEMA],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

4

Préparer l'emplacement d'affichage des vues correspondantes aux routes

- Pour indiquer à Angular ou est ce qu'il doit charger les vues spécifiques aux routes nous utilisons le router outlet.
- Router outlet est une directive qui permet de spécifier l'endroit ou la vue va être chargée.
- Sa syntaxe est <router-outlet></router-outlet>

Préparer l'emplacement d'affichage des vues correspondantes aux routes

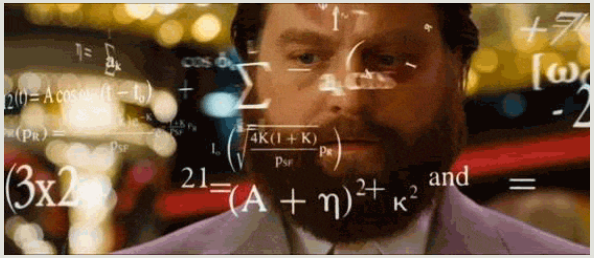
```
<as-header></as-header>

<div class="container">
  <router-outlet></router-outlet>
</div>
```

Syntaxe minimaliste d'une route

- Une route est un objet.
- Les deux propriétés essentielles sont path et component.
- path permet de spécifier l'URI. Cette url ne doit pas commencer par un /
- component permet de spécifier le composant à exécuter.

```
{path: '',component:CvComponent},
{path:'onlyHeader',component:HeaderComponent}
```

Exercice



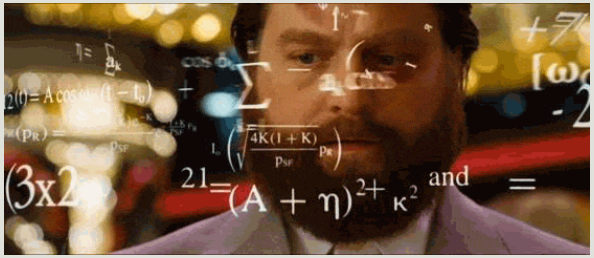
- Configurer votre routing
- Créer deux composants
- Créer deux routes qui pointent sur ces deux composants
- Vérifier le fonctionnement de votre routing

Déclencher une route routerLink

- L'idée intuitive pour déclencher une route est d'utiliser la balise **a** et son attribut **href**. Est-ce que ca risque de poser un problème ?
- L'utilisation de <a href > va déclencher le **chargement** de la page ce qui est inconcevable pour une SPA.
- La solution proposée par le router d'Angular est l'utilisation de la **directive routerLink** qui comme son nom l'indique liera la directive à la route que nous souhaitons déclencher **sans recharger la page**.
- Exemple :

```
<li ><a [routerLink]="[‘’]" routerLinkActive="active">Gérer les cvs</a></li>
```

Exercice



- Faites en sorte d'avoir un composant dans votre application qui permet d'afficher l'ensemble de vos liens.

- En cliquant sur un lien, le composant qui lui est associé doit être affiché.

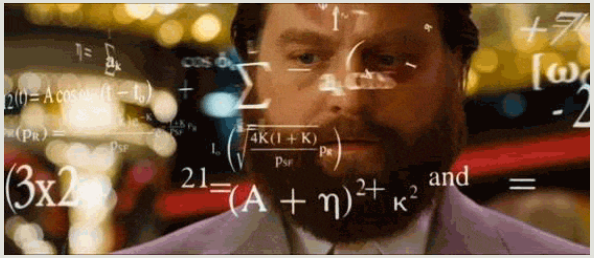
Déclencher une route à partir du composant

- Afin de déclencher une route à travers le composant on utilise l'objet **Router** et sa méthode **navigate**.
- Cette méthode prend le **même paramètre** que le routerLink, à savoir un tableau contenant la description de la route.
- Afin d'utiliser le Router, il faut l'importer de l'@angular/router et l'injecter dans votre composant.

Déclencher une route à partir du composant

```
import { Component} from '@angular/core';
import {Router} from "@angular/router";
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent{
  constructor(private router:Router) { }
  onNaviger(){
    this.router.navigate(['/about/10']);
  }

}
```

Exercice



- Créer un composant appelé RouerSimulator
- Dans ce composant créer une liste déroulante contenat le nom des différentes routes de votre application.
- Ajouter ce composant au même niveau que le header et que le <router-outlet>.
- En sélectionnant le nom d'un composant, il doit apparaitre dans le <router-outlet> simulant ainsi le fonctionnement d'un routeur.

Les paramètres d'une route

- Afin de spécifier à notre router qu'un segment d'une route est un paramètre, il suffit d'y ajouter ':' devant le nom de ce segment.

- Exemple
  - /cv/:id permet de dire que la root contient au début cv ensuite un paramètre de root appelé id.

Récupérer les paramètres d'une route

- Afin de récupérer les paramètres d'une root au niveau d'un composant on doit procéder comme suit :
  1. Importer **ActivatedRoute** qui nous permettra de récupérer les paramètres de la root.
  2. **Injecter** ActivatedRoute au niveau du composant.
  3. Affecter le paramètre à une variable du composant en s'inscrivant avec la méthode **subscribe** à l'observable **params** de notre **ActivatedRoute**. Cette variable retourne un tableau de l'ensemble des paramètres.

Syntaxe :

```
activatedRoute.params.subscribe(params=>{this.monParam=params['param']});
```

Récupérer les paramètres d'une route

```
import {RouterModule, Routes} from
'@angular/router';
import {HomeComponent} from
"./app/home/home.component";
import {AboutComponent} from
"./app/about/about.component";

const APP_Routes:Routes =[
{path:'',component:HomeComponent},

{path:'about/:param',component:AboutCo
mponent},
  ]
;

export const routing =
RouterModule.forRoot(APP_Routes);
```

4

App.routing.ts

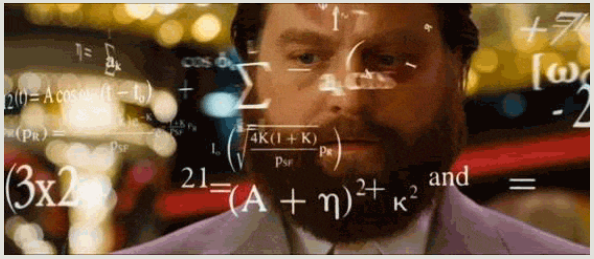
```
import { Component, OnInit } from '@angular/core';
import {ActivatedRoute} from '@angular/router';
@Component({
  selector: 'app-about',
  templateUrl: './about.component.html',
  styleUrls: ['./about.component.css']
})
export class AboutComponent {

  monParam:any;
  constructor(private router:ActivatedRoute) {
router.params.subscribe(params=>{this.monParam=par
ams['param']});
  }
}
```

1

2

3

Exercice



- Reprendre le composant qui permet de changer la couleur de la DIV
- Ajouter lui une route
- Faite en sorte que cette route soit de cette forme /color/:couleur et qui permettra d'affecter la couleur récupérée par la route comme couleur par défaut du DIV.

Passer le paramètre à travers le tableau de routerLink

- Une autre méthode permet de passer le paramètre de la route est en l'ajoutant comme un autre attribut du tableau associé au routerLink

```
import { Component, OnInit } from '@angular/core';
import {Router} from "@angular/router";
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent{
  constructor(private router:Router) { }
  id:number=10;
  onNaviger(){this.router.navigate(['/about',this.id]);}
}
```

Les queryParameters

- Les **queryParameters** sont les paramètres envoyé à travers une requête **GET**.
- Identifié avec le **?**.
- Afin d'insérer un **queryParameters** on dispose de deux méthodes
- On ajoute dans la méthode **navigate** du **Router** un second paramètre de type objet.
- L'une des propriétés de cet objet est aussi un objet dont la clé est **queryParams** dont le contenu est aussi un objet content les identifiants des queryParams et leurs valeurs.

```
this.router.navigate(['/about',this.id],{queryParams:{'qpVar':'je suis un qp'}});
```

Les queryParameters

- La deuxième méthode est en l'intégrant à notre routerLink de la manière suivante :

```
<a [routerLink]="['/about/10']" [queryParams]="{qpVar:'je suis un qp bindé avec le routerLink'}">About</a>
```

Récupérer Les queryParameters

- Les queryParameters sont récupérable de la même facon que les paramètres.
- Afin de récupérer les paramètres d'une root au niveau d'un composant on doit procéder comme suit :
  1. Importer **ActivatedRoute** qui nous permettra de récupérer les paramètres de la root.
  2. **Injecter** ActivatedRoute au niveau du composant.
  3. Affecter le paramètre à une variable du composant en utilisant la méthode **subscribe** pour se souscrire à **params** de notre **ActivatedRoute**.

  Syntaxe :

```
activatedRouter.queryParams.subscribe(
   (queryParam:any)=>(this.monQp=queryParam['qpVar'])
);
```

Parenthèse

- La méthode **subscribe** permet de s'inscrire à un **observable**.
- **Problème :** Cette souscription reste valide même après la disparition de la variable ce qui sature la mémoire pour rien.
- **Solution : Se Désinscrire à la mort du composant donc dans le ngOnDestroy().**

Route Fils

- Certains composants ne sont visible qu'à l'intérieur d'autres composants.
- Prenons l'exemple d'un objet Personne. En accédant à la route /personne/:id nous avons l'affichage de la personne et nous aimerions avoir deux boutons. Un pour éditer la Personne (route /personne/:id/editer). L'autre pour afficher ces détails (route /personne/:id/apercu).
- L'idée est de préfixer nos routes.

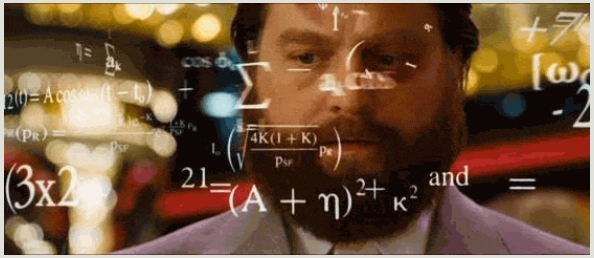
Route Fils

- Afin de mettre en place ce processus nous procédons comme suit :
  - Nous définissons le préfixe avec la propriété **path**.
  - Nous y ajoutons la propriété children qui contiendra le tableau des routes. Chaque route de ce tableau sera préfixé avec la route définie dans **path**.

Route Fils

```
const CV_ROUTE: Routes = [
  {
    path: 'cv',
    children: [
      {path: '', component: CvComponent },
      {path: 'detail/:id', component: DetailCvComponent },
      {path: 'addPersonne', component: FormPersonneComponent },
    ]
  }
];
```

Exercice



- Modéliser un système de routage qui utilise ses propriétés.

Route fils / définition dans un parent

- Supposons que nous voulons avoir un Template central avec des données fixe et des parties variables dans le même template.

- En changeant les routes, le contenu principal doit rester le même et la partie variable doit changer selon la route.

Route Fils

- Afin de mettre en place ce processus nous procédons comme suit :
  - Nous définissons le préfixe avec la propriété **path**. On lui associe le composant Père.
  - Nous y ajoutons la propriété children qui contiendra le tableau des routes. Chaque route de ce tableau sera préfixé avec la route définie dans **path**.
  - Nous ajoutons la balise <router-outlet></router-outlet> dans le Template père.

Route Fils

```
const CV_ROUTE: Routes = [
  {
    path: 'cv',
    component: CvComponent,
    children: [
      {path: 'detail/:id', component: DetailCvComponent },
      {path: 'addPersonne', component: FormPersonneComponent },
    ]
  }
];
```

Redirection

- Afin de rediriger une route il suffit d'ajouter une propriété dans l'objet route qui est **redirectTo.** Cette propriété permet d'indiquer vers quelle route le path doit être redirigé**.** Si la route n'a pas encore été matché, alors les routes commençant par ce path seront redirigées.

- Une autre propriété peut être utilisé qui est la propriété **pathMatch**. Cette propriété permet de définir comment le matching des path est exécuté. Avec la **valeur 'full'**, elle spécifie au routeur de ne faire la redirection que si le path exact est matché.

Redirection : exemple

```
const APP_Routes:Routes =[
   {path:'',component:HomeComponent},
   {path:'about',redirectTo:'/', pathMatch:'full'},
   {path:'about/:param',component:AboutComponent},
   {path:'about/:param',component:AboutComponent,children:FILS_ROUTE},
   ]
```

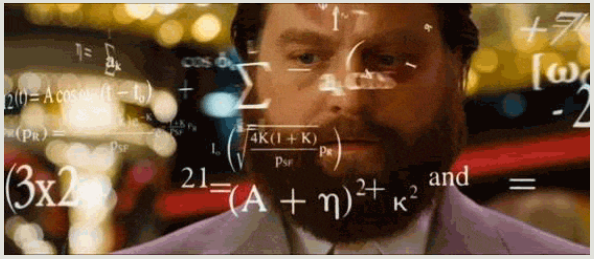
Redirection : gestion d'erreurs de rooting

- Afin de rediriger une route inexistante vers une page d'erreur, il suffit de garder la même syntaxe de redirection et de mettre dans la **propriété path '**'**.

Exemple

```
const APP_ROUTE: Routes = [
  {path: '', redirectTo: 'cv', pathMatch: 'full'},
  {path: `cv', component: CvComponent},
  {path: 'lampe', component: ColorComponent},
  {path: 'login', component: LoginComponent},
  {path: 'error', component: ErrorPageComponent},
  {path: '**', component: ErrorPageComponent }
];
```

Exercice



- Créer un composant HeaderComponent contenant votre navbar.

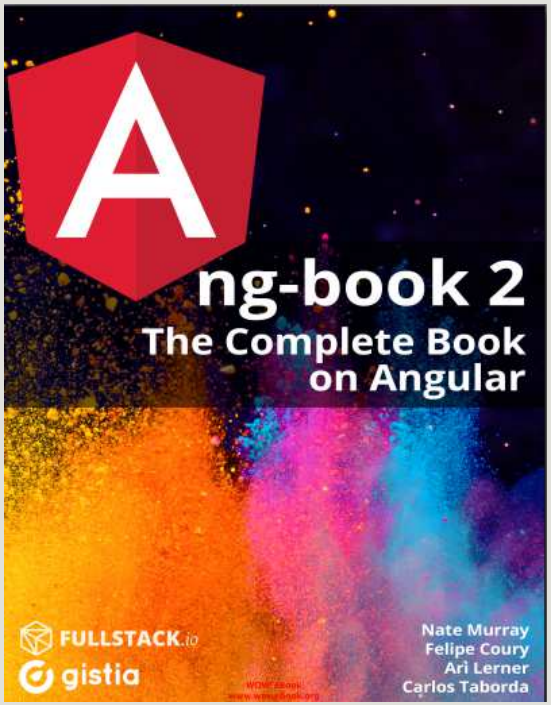
- Ajouter les fonctionnalités suivante à votre cvTech:
  - Une page détail qui va afficher les détails d'un cv.
  - Un bouton dans chaque cv qui au click vous envoi vers la page détails.
  - Dans la page détail, un bouton delete qui au click supprime ce cv et vous renvoi à la liste des cvs.

Angular Form

AYMEN SELLAOUTI

Références


ANGULAR
FEATURES
DOCS
EVENTS
GET STARTED
One framework.
Mobile & desktop.
GET STARTED
Angular v4.0 is out! Smaller, faster, no biggie...
LEARN MORE


ng-book 2
The Complete Book on Angular
FULLSTACK.io
gistia
Nate Murray
Felipe Coury
Ari Lerner
Carlos Taborda


Learn Angular
Gravissez tous les concepts jusqu'au sommet
Maîtriser les concepts du Framework Angular
pour développer des applications robustes
Auteur : William KOZA

Plan du Cours

1. Introduction
2. Les composants
3. Les directives

3 Bis. Les pipes

4. Service et injection de dépendances
5. Le routage
6. Form
7. HTTP

Approche de gestion de FORM

1. Approche basée Template
2. Approche réactive

Objetctifs

1. Créer un formulaire
2. Ajouter des validateurs
3. Appréhender les classes Css générées par le formulaire
4. Manipuler l'objet ngForm
5. Manipuler les controles du formuliare

Approche basée Template/ Template Driven Approach

1. Importer le module FormsModule dans app.module.ts

2. Angular détecte automatiquement un objet form à l'aide de la balise FORM. Cependant, il ne détecte aucun des éléments (inputs).

3. Spécifier à Angular quel sont les éléments (contrôles) à gérer.
   - Pour chaque élément ajouter la directive angular **ngModel**.
   - Identifier l'élément avec un nom permettant de le détecter et de l'identifier dans le composant.

4. Associer l'objet représentant le formulaire à une variable et la passer à votre fonction en utilisant le référencement interne # et la directive **ngForm**

```
<input
  type="text"
  id="username"
  class="form-
control"
  ngModel
  name="username"
>
```

Approche basée Template/ Template Driven Approach

```
<form
(ngSubmit)="onSubmit(formula
ire)" #formulaire="ngForm">
```

Template

```
export class
TmeplateDrivenComponent{
   onSubmit(formulaire:
NgForm){

console.log(formulaire);
   }
}
```

Component.ts

Approche basée Template Validation

Afin de valider les propriétés des différents contrôles, Angular utilise des attributs et des directives

- required
- email

La propriété valid de ngForm permet de vérifier si le formulaire est valid ou non en se basant sur les validateurs qu'ils contient.

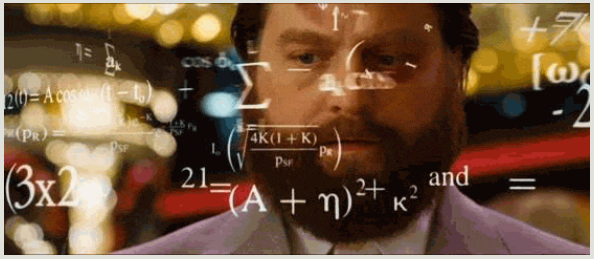
https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5/Constraint_validation

Approche basée Template
NgForm

En détectant le formulaire, Angular décore les différents éléments du formulaire avec des classes qui informe sur leur état :

- dirty : informe sur le fait que l'une des propriétés du formulaire a été modifié ou non
- Valid : informe si le formulaire est valide ou non
- untouched : informe si le formulaire est touché ou non
- pristine : le formulaire n'a pas été touché, c'est l'opposé du dirty

Exercice



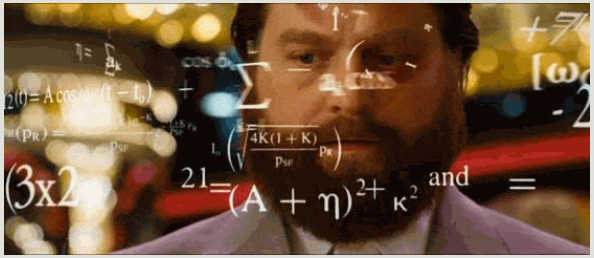
- Créer un formulaire d'authentification contenant les champs suivants :
  - Email
  - Password
  - Envoyer
- Si un champ est invalide alors il devra avoir une bordure rouge.
- Un champ vide et non encore modifié ne peut avoir de bordure rouge que s'il a été touché. Le bouton « envoyer » ne doit être cliquable que si le formulaire est valide. Utiliser le binding sur la propriété disabled.

Approche basée Template
Accéder aux propriétés d'un champ (contrôle) du formulaire

- Pour accéder à l'objet form et ces propriétés nous avons utilisé
  #notreForm=«ngForm »

- Pour les champs du formulaire c'est la même chose mais au lieu du ngForm c'est un **ngMdoel**
  #notreChamp=« ngModel »

Exercice

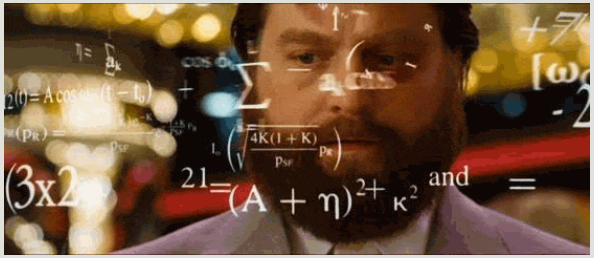


- Ajouter un petit message d'erreur qui devra s'afficher sous le champs de l'email s'il est invalide. Ce champ ne devra apparaitre que si l'utilisateur accède ou modifie le champ email.
- Le password devra avoir au moins 6 caractères. Ajouter un champ d'erreur pour signaler l'erreur à l'utilisateur.

Approche basée Template
Associer des valeurs par défaut aux champs

- Pour associer des valeurs par défaut au champs d'un formulaire associé à Angular il faut le faire à partir du composant.
- Afin de gérer les valeur du formulaire à partir du composant il faut du binding.
- Au lieu d'avoir juste la primitive ngModel associé au contrôle d'un élément on ajoute le property binding avec [ngModel]

Exercice



- Ajouter la valeur par défaut « myUserName » au champ username.

Approche basée Template
Grouping form

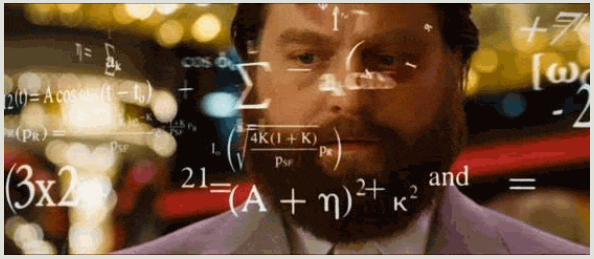
- Afin de grouper l'ensemble des contrôles (propriétés/champs) d'un formulaire, on peut utiliser la technique du « grouping form controls ».

- Il suffit d'ajouter la directive ngModelGroup dans la div qui englobe les propriétés à grouper.

```
<div
  ngModelGroup= "user"
  userData= "ngModelGroup"
>
```

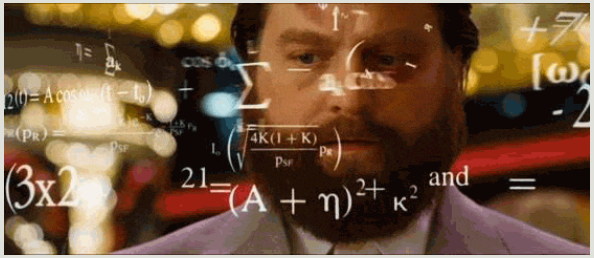
- Afin d'accéder à cet objet vous pouvez le référence localement en utilisant le mot clé ngModelGroup

Exercice



- Grouper les données de votre utilisateur dans un ngModelGroup
- Tester l'objet généré
- Essaye de voir s'il contient les mêmes classes qu'un contrôle simple, e.g. ng-dirty, ng-valid.
- Ajouter un message d'erreur qui apparait si votre groupe n'est pas valide.

Exercice



- Ajouter dans votre cvTech un composant contenant un formulaire. Ce formulaire devra vous permettre d'ajouter un utilisateur.

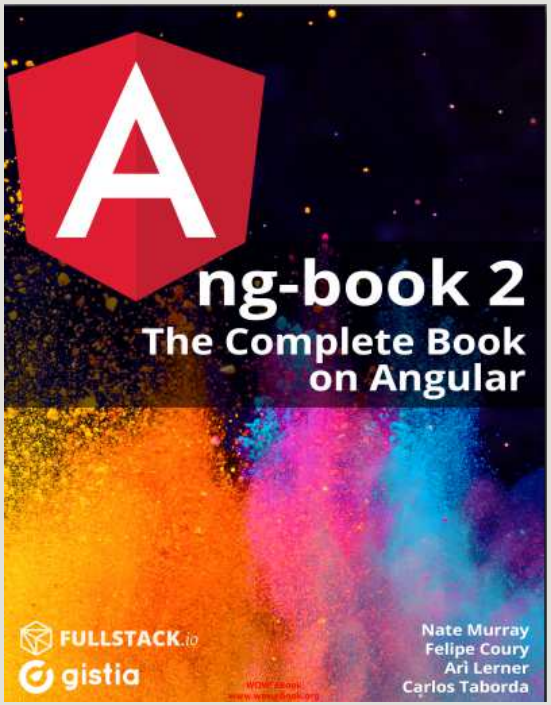
- Après l'ajout fowarder le user vers la liste des cvs.

Angular
HTTP et Déploiement

AYMEN SELLAOUTI

Références


ANGULAR
FEATURES
DOCS
EVENTS
GET STARTED
One framework.
Mobile & desktop.
GET STARTED
Angular v4.0 is out! Smaller, faster, no biggie...
LEARN MORE


ng-book 2
The Complete Book on Angular
FULLSTACK.io
gistia
Nate Murray
Felipe Coury
Ari Lerner
Carlos Taborda


Learn Angular
Gravissez tous les concepts jusqu'au sommet
Maîtriser les concepts du Framework Angular
pour développer des applications robustes
Auteur : William KOZA

Plan du Cours

1. Introduction
2. Les composants
3. Les directives

3 Bis. Les pipes

4. Service et injection de dépendances
5. Le routage
6. Form
7. HTTP

Objectifs

1. Comprendre le design pattern Observable et son implémentation avec RxJs
2. Appréhender le Module HTTPClientModule d'Angular
3. Utiliser les différents services du module HTTPClientModule
4. Comprendre le principe d'authentification via les tokens
5. Utiliser les protecteurs de routes (les guards)
6. Utiliser les Interceptors afin d'intercepter les requêtes Http
7. Déployer votre application en production

HTTP

- Angular est un Framework FrontEnd
- Pas d'accès à la BD
- Pas de possibilité de persistance des données
- Pas de puissance permettant des traitements lourds.

Le Moule HTTPClient

Programmation Asynchrone

Programmation non bloquante.

Qu'est ce que la programmation réactive

1. Nouvelle manière d'appréhender les appels asynchrones
2. Programmation avec des flux de données asynchrones

Programmation reactive =
Flux de données (observable) + écouteurs d'événements(observer).

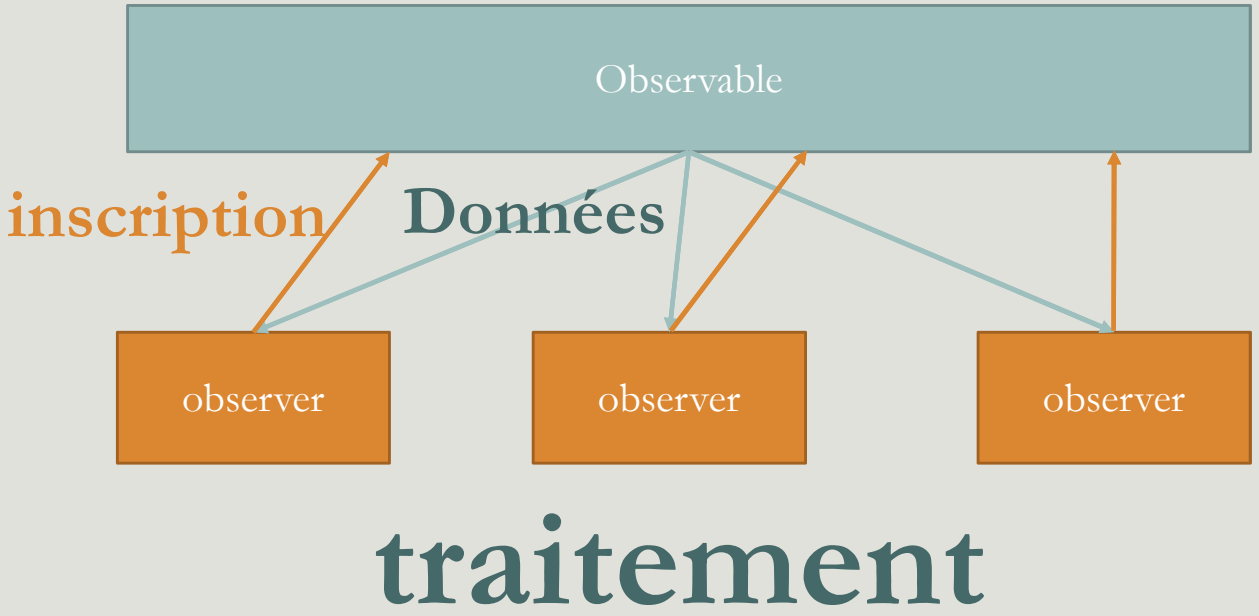
Le pattern « Observable »

- Le patron de conception **Observable** permet à un objet de garder la trace d'autres objets, intéressés par l'état de ce dernier.

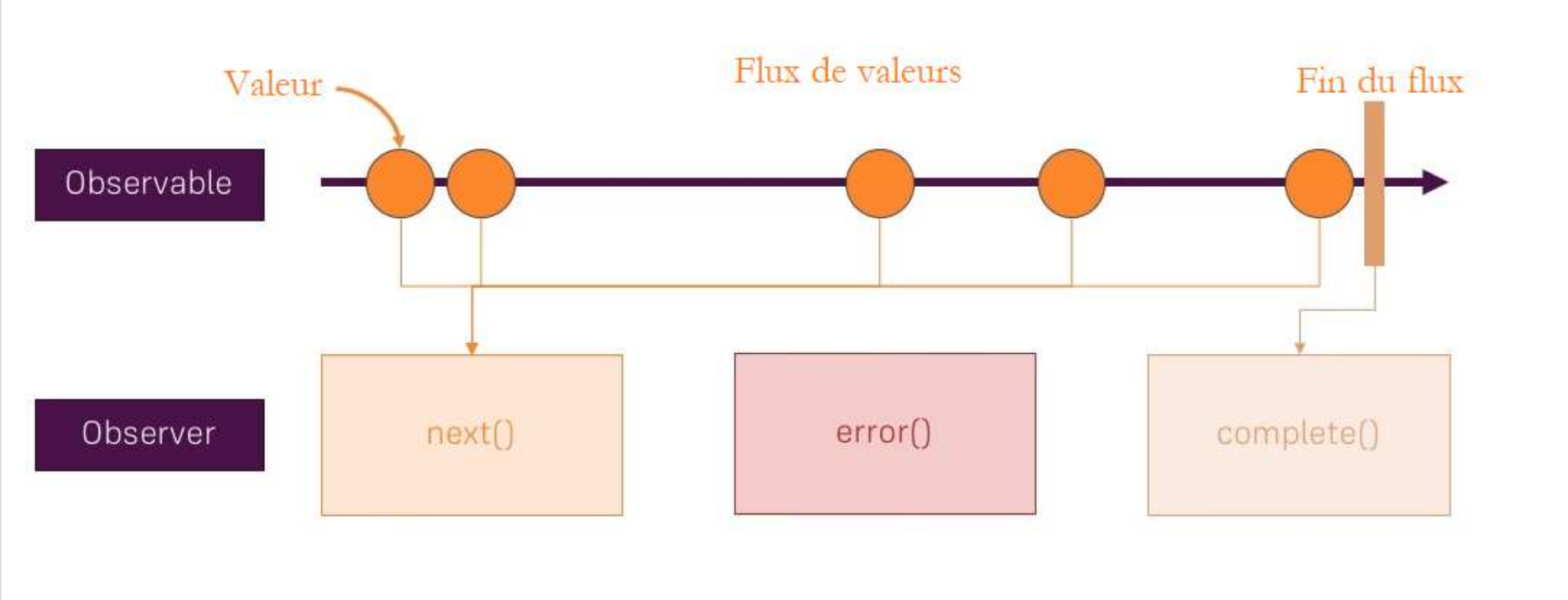
- **Il définit une relation entre objets de type un-à-plusieurs.**

- Lorsque l'état de cet objet **change**, il **notifie** ces **observateurs**.

Observables, Observers et subscriptions



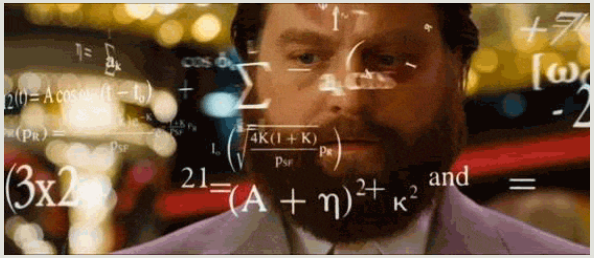
Fonctionnement



Observable

```
const observable = new Observable(
  (observer) => {
    let i = 5;
    setInterval(() => {
      if (!i) {
        observer.complete();
      }
      observer.next(i--);
    }, 1000);
  });
observable.subscribe(
  (val) => {
    console.log(val);
  }
);
```

Exercice



- Ecrire un composant qui affiche une suite d'images non stop.
- Utiliser un observable comme la source des images.

asyncPipe

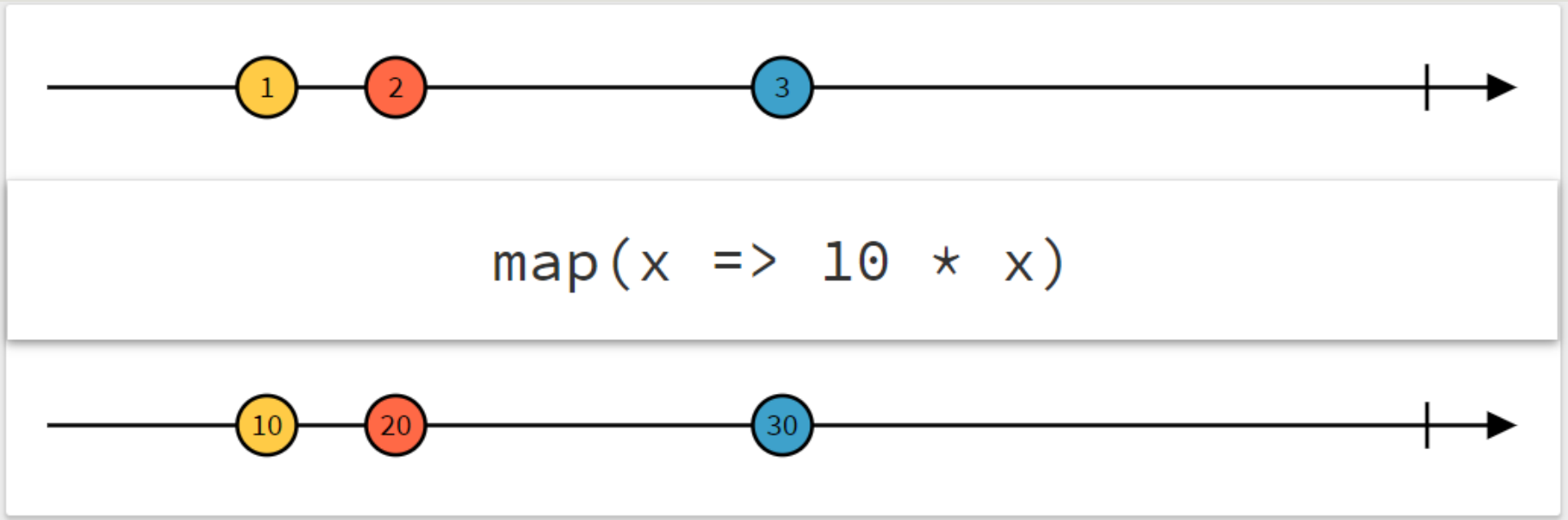
- asyncPipe est un pipe qui permet d'afficher directement un observable.
- {{ valeurSourceAsynchrone | async }}
- L'asyncPipe s'inscrit automatiquement à l'observable et affiche le dernier résultat envoyé.
- Quand le composant est détruit l'asyncPipe se désinscrit automatiquement de l'observable.

Les operateurs de l'observable

- Les opérateurs sont des fonctions. Il y a deux types d'opérateurs :
- **Un opérateur pipeable** est une fonction qui prend un observable comme entrée et renvoie un autre observable. C'est une opération pure : le précédent Observable reste inchangé.
  - Syntaxe : monObservable.pipe(opertaeur1(), operateur2(), …).
- **Les opérateurs de création** sont l'autre type d'opérateur, qui peut être appelé comme fonctions autonomes pour créer un nouvel Observable. Par exemple : of(1, 2, 3) crée un observable qui va émettre 1, 2, et 3, l'un après l'autre.

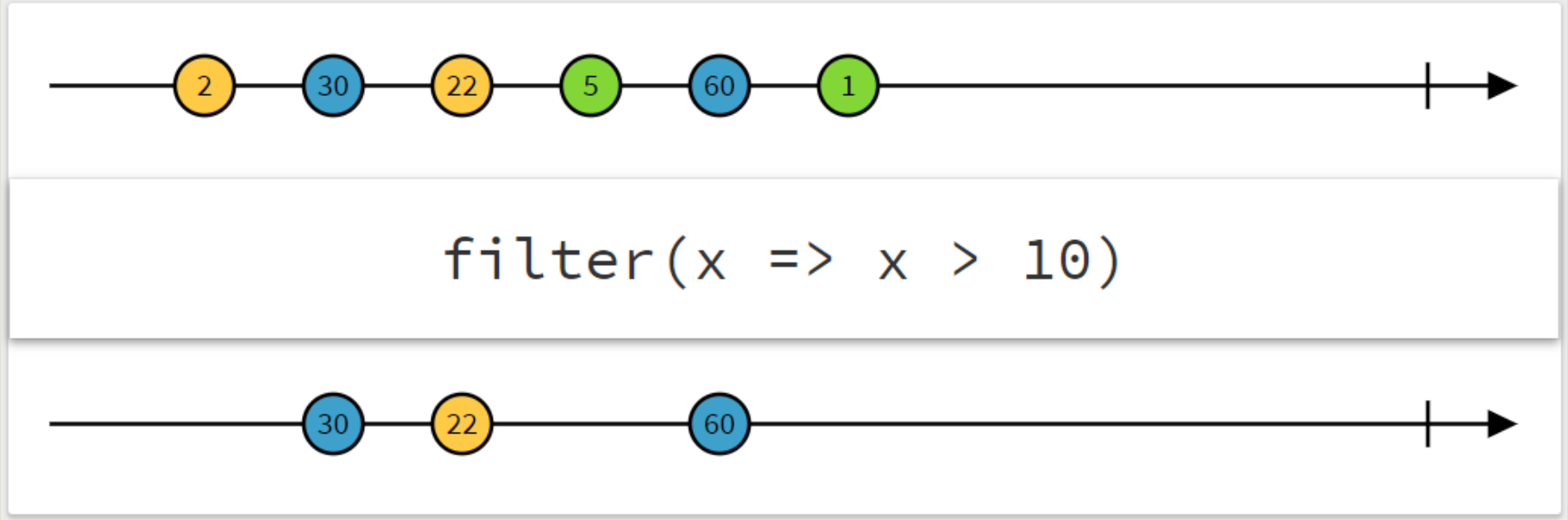
Quelques opérateurs utiles de l'Observable

map

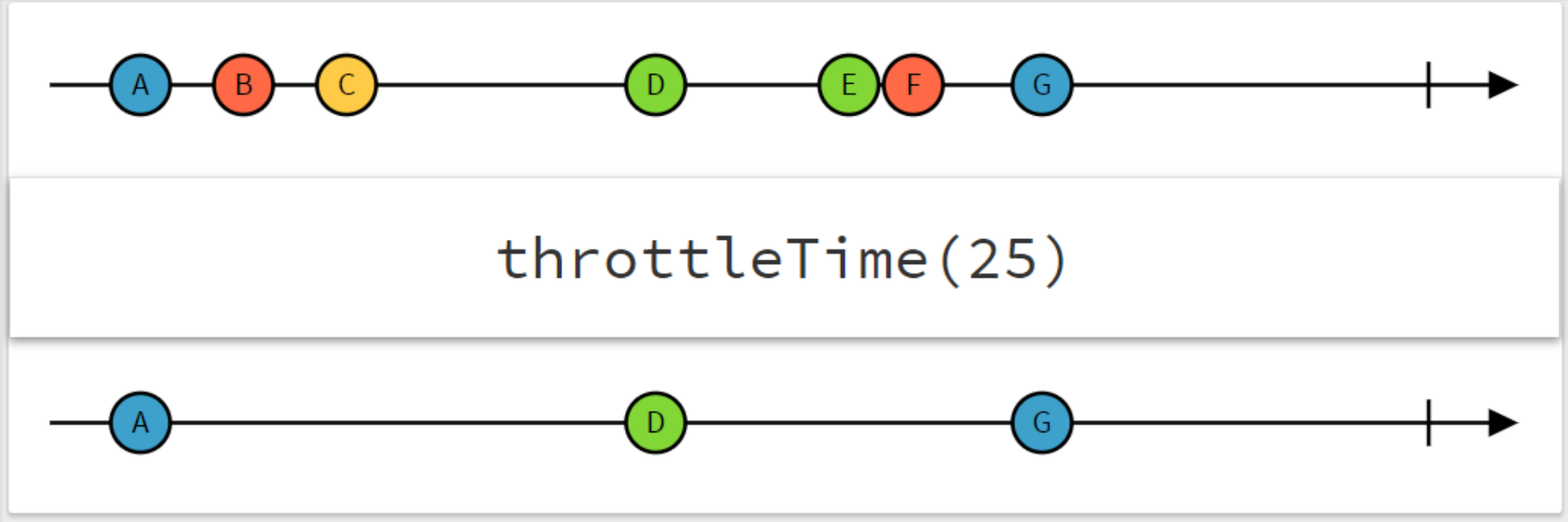


Quelques opérateurs utiles de l'Observable

filter



Quelques opérateurs utiles de l'Observable



Quelques opérateurs utiles de l'Observable

https://angular.io/guide/rx-library

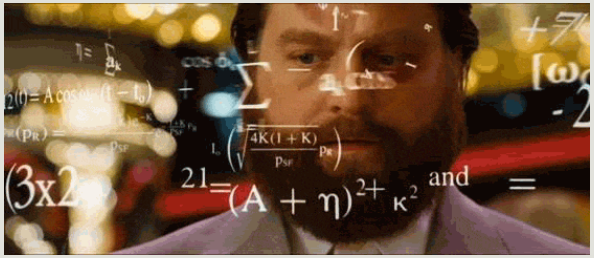
http://reactivex.io/rxjs/manual/overview.html#operators

http://rxmarbles.com/

Les subjects

- Un subject est un type particulier d'observable. En effet Un subject est en même temps un observable et un observer, il possède donc les méthodes next, error et complete.
- Pour broadcaster une nouvelle valeur, il suffit d'appeler la méthode next, et elle sera diffusé aux Observateurs enregistrés pour écouter le Subject.

Exercice



- Modifier l'affichage des détails d'une personne au click. Enlever tous les outputs et remplacer les par l'utilisation d'un subject.

Installation de HTTP

- Le module permettant la consommation d'API externe s'appelle le HTTP MODULE.
- Afin d'utiliser le module HTTP, il faut l'importer de @angular/common/http (@angular/http dans les anciennes versions)

```
import {HttpClientModule} from "@angular/common/http";
```

- Il faudra aussi l'ajouter dans le fichier module.ts dans le tableau d'imports.

```
imports: [
  BrowserModule,
  FormsModule,
  HttpClientModule,
],
```

Installation de HTTP

- Afin d'utiliser le module HTTP, il faut l'injecter dans le composant ou le service dans lequel vous voulez l'utiliser.

```
constructor(private http:HttpClient) { }
```

Interagir avec une API Get Request

- Afin d'exécuter une requête **get** le module http nous offre une méthode **get**.
- Cette méthode retourne un **Observale**.
- Cette observable a 3 callback function comme paramètres.
  - Une en cas de réponse
  - Une en cas d'erreur
  - La troisième en cas de fin du flux de réponse.

Interagir avec une API Get Request

```
this.http.get(API_URL).subscribe(
   (response:Response)=>{
     //ToDo with DATA
     },
(err:Error)=>{
   //ToDo with error
},
   () => {
     console.log('Data transmission complete');
}
);
```

Interagir avec une API POST Request

- Afin d'exécuter une requête POST le module http nous offre une méthode post.
- Cette méthode retourne un Observale.
- Diffère de la méthode get avec un attribut supplémentaire : body
- Cette observable a 3 callback function comme paramètres.
  - Une en cas de réponse
  - Une en cas d'erreur
  - La troisième en cas de fin du flux de réponse.

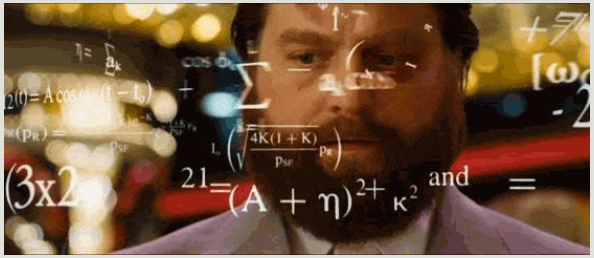
Interagir avec une API POST Request

```
this.http.post(API_URL,dataToSend).subscribe(
   (response:Response)=>{
     //ToDo with response
     },
(err:Error)=>{
   //ToDo with error
},
   () => {
     console.log('complete');
}
);
```

Documentation

https://angular.io/guide/http

Exercice



- Accéder au site https://jsonplaceholder.typicode.com/

- Utiliser l'API des posts pour afficher la liste des posts. En attendant le chargement des données afficher un message « loading… ».

- Ajouter un input. A chaque fois que vous écrivez un élément dans cet input il sera ajouté dans la liste.

Les headers

- Afin d'ajouter des headers à vos requêtes, le HttpClient vous offre la classe HttpHeaders.
- Cette classe est une classe immutable (read Only).

https://angular.io/guide/http#immutability

- Elle propose une panoplie de méthode helpers permettant de la manipuler.
- set(clé,valeur) permet d'ajouter des headers. Elle écrase les anciennes valeurs.
- append(clé,valeur) concatène de nouveaux headers.

Toutes les méthodes de modification retourne un HttpHeaders permettant un chainage d'appel.

https://angular.io/api/common/http/HttpHeaders

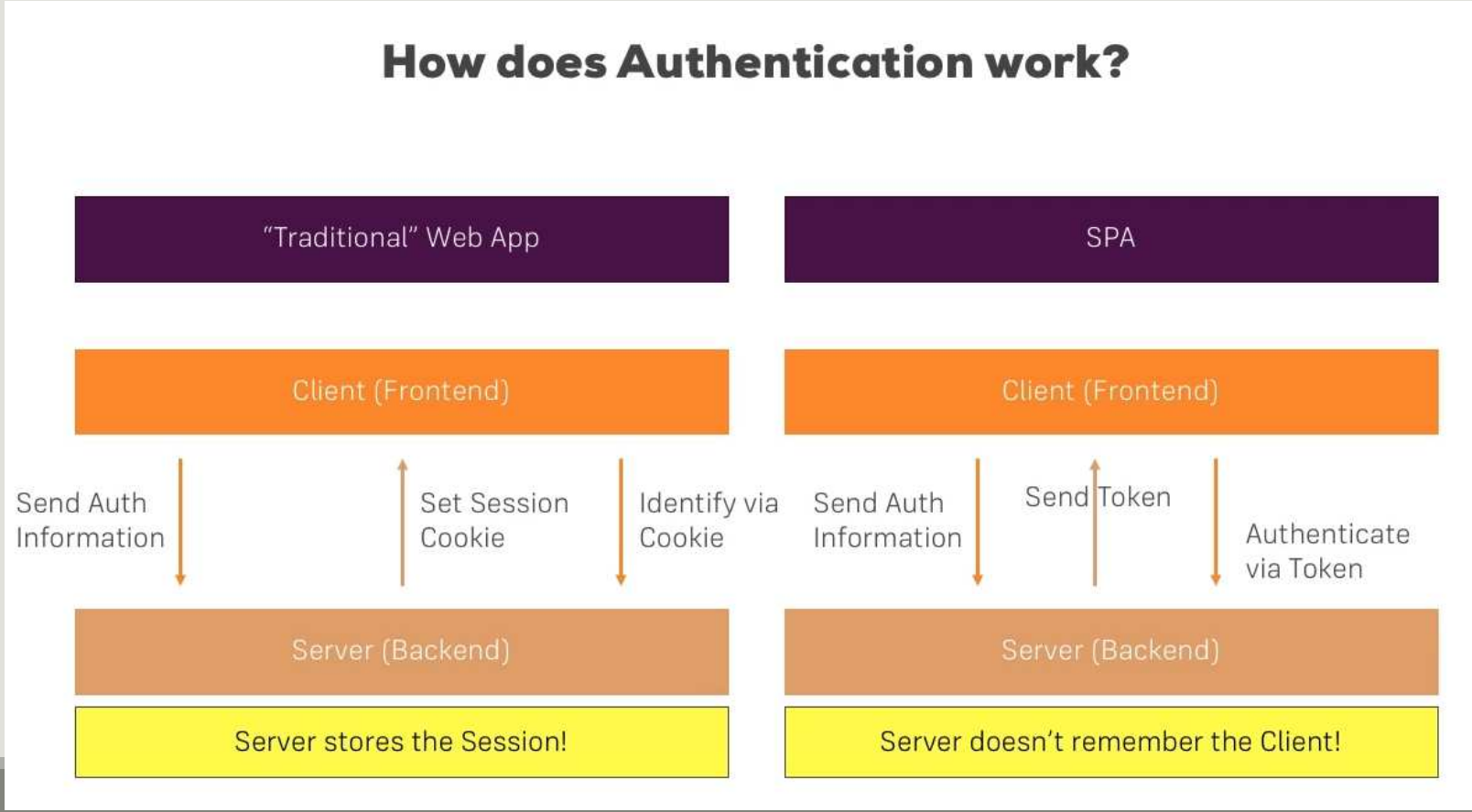
Les paramètres

- Afin d'ajouter des paramètres à vos requêtes, le HttpClient vous offre la classe HttpParams.
- Cette classe est une classe immutable (read Only).
- Elle propose une panoplie de méthode helpers permettant de la manipuler.
- set(clé,valeur) permet d'ajouter des headers. Elle écrase les anciennes valeurs.
- append(clé,valeur) concatène de nouveaux headers.

Toutes les méthodes de modification retourne un HttpParams permettant un chainage d'appel.

https://angular.io/api/common/http/HttpParams

Authentification



S’authentifier avec Loopback

- Loopback offre un mécanisme d”authentification prêt à l’emploi.
- Avec l’api de la classe user il vous permet d’ajouter des users et de vous connectez. Il permet aussi avec son module Loopback acl de créer des restrictions sur l’api.
- Une fois vos uri protégée, vous devez vous connecter pour pouvoir les utiliser.
- En vous connectant, il vous offrira un token. Vous devez l’utiliser à chaque appel de votre api.

Installer loopback

- Nous allons installer la version lts :
- **npm install -g loopback-cli**
- Vérifier votre version avec **lb –v**
- Créer votre première application avec la commande **lb**
- Renseigner le nom de votre projet
- Renseigner le type de votre projet
- C'est fait

Loopback : création du modèle

- Créer votre modèle avec **lb model**

- Suivez les étapes, ajouter votre modèle et ajouter les champs qui le compose.

- Dans notre exemple nous allons utiliser un model « Personne »

Loopback : création du modèle

```
PS E:\FormaAngular\gestionPersonnes> lb model
? Entrer le nom du modèle : personne
? Sélectionner la source de données à laquelle associer personne : (n
o datasource)
? Sélectionner la classe de base du modèle PersistedModel
? Exposer personne via l'API REST ? Yes
? Forme plurielle personnalisée (utilisée pour générer l'URL REST) :
? Modèle commun ou serveur uniquement ? commun
```

Modèle généré

- Dans ce cas nous allons utiliser une base de données MySql avec un id en auto-increment.

```
{
  "name": "personne",
  "base": "PersistedModel",
  "idInjection": true,
  "options": {
    "validateUpsert": true
  },
  "properties": {
    "cin": {
      "type": "number",
      "required": true
    },
    "name": {
      "type": "string",
      "required": true
    },
    "firstname": {
      "type": "string",
      "required": true
    },
    "age": {
      "type": "number",
      "required": true
    },
    "path": {
      "type": "string",
      "required": true
    },
    "job": {
      "type": "string",
      "required": true
    }
  },
```

Associer votre modèle à une base de données MySql

- Installer le mysql-connector

```
npm install loopback-connector-mysql --save
```

- Configurer votre datasource dans le fichier datasource.json et ajouter la configuration de votre base de données mySql.

- Associer la datasource à votre modèle dans le fichier model-config.json.

Associer votre modèle à une base de données MySql

```
{
  "db": {
    "name": "db",
    "connector": "memory"
  },
  "personneDb": {
    "host": "localhost",
    "port": 3306,
    "url": "",
    "database": "test_pdo",
    "password": "",
    "name": "personneDb",
    "user": "root",
    "connector": "mysql"
  }
}
```

datasource.json

```
                    .
                    .
personneDb          .

  "Note": {
    "dataSource": "db"
  },
  "personne": {
    "dataSource": "personneDb",
    "public": true
  }
```

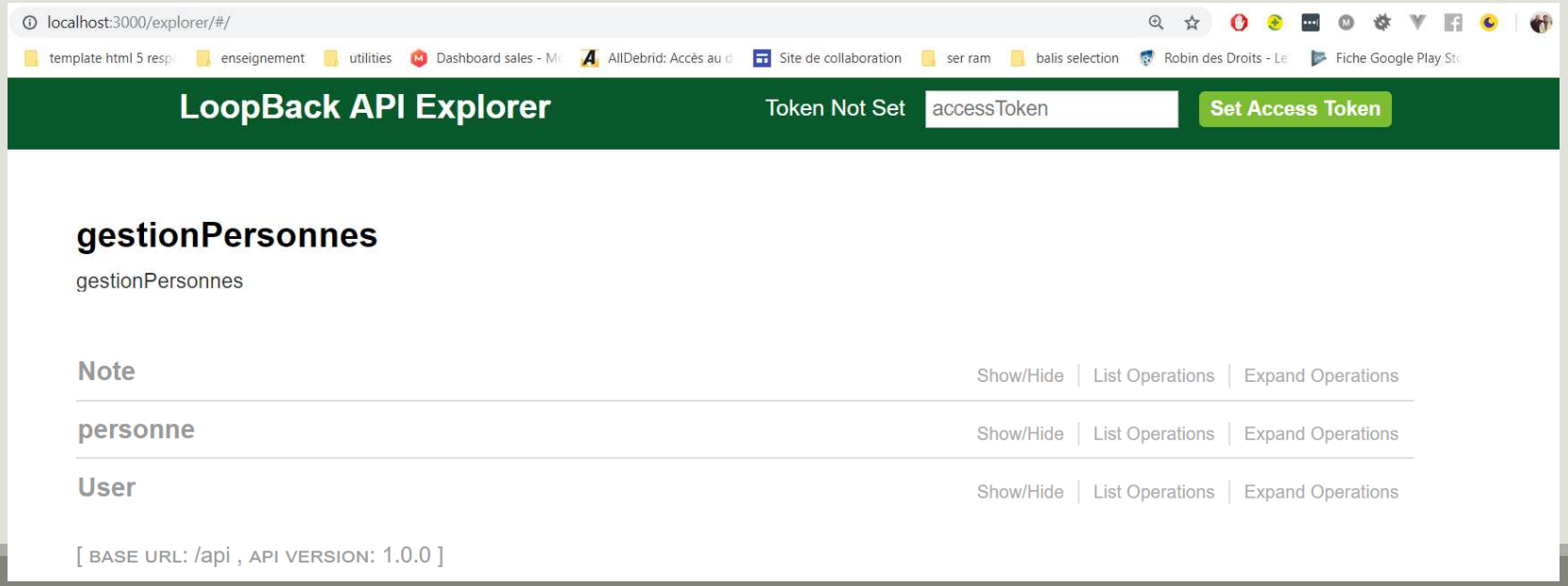
Model-config.json

Lancer votre application

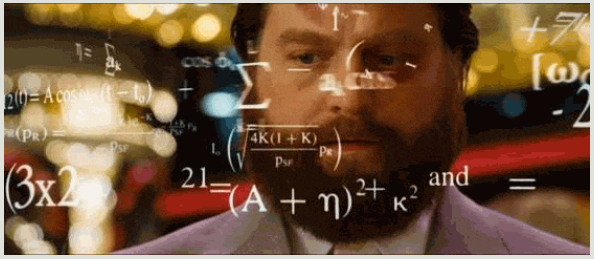
- Pour lancer votre application exécuter la commande **node .**

- Accéder à votre swagger en utilisant l'adresse http://localhost:3000/explorer

- Tester votre api.



Exercice



- Associer votre application à votre API Loopback. Tester les fonctionnalités d'ajout de suppression et de modification et de sélection de l'ensemble des personnes.

- Sachant que pour sélectionner une personne dont le nom contient une chaine donnée, loopback utilise la syntaxe suivante :

```
{"where":{"name":{"like":"%${name}%"}}}
```

- Ceci doit être fourni dans les paramètres de votre requête avec la clé filter. Tester le sur votre swagger.

- Ajouter un champ input. A chaque caractère saisie, la liste des choix doit automatiquement changer et n'afficher que les cvs qui contiennent la chaine saisie. En sélectionnant un des choix, rediriger l'utilisateur vers les détails du cv sélectionné.

Ajouter une acl avec loopback

- Nous voulons ajouter des contraintes d'accès à notre application.
- Nous voulons que seul les utilisateurs connectés puissent ajouter et supprimer des personnes.
- Ajouter les ACL en utilisant la commande lb acl.

```
PS E:\FormaAngular\gestionPersonnes> lb acl
? Sélectionner le modèle auquel a
ppliquer l'entrée ACL : personne
? Sélectionner la portée ACL : To
utes les méthodes et propriétés
? Sélectionner le type d'accès :
Ecrire
? Sélectionner le rôle Tous les u
tilisateurs non authentifiés
? Sélectionner le droit à appliqu
er Refuser explicitement l'accès
PS E:\FormaAngular\gestionPersonnes>
```

Ajouter une acl avec loopback

- Vérifier que votre modèle a été modifié :

```
"acls": [
  {
    "accessType": "WRITE",
    "principalType": "ROLE",
    "principalId": "$unauthenticated",
    "permission": "DENY"
  }
```

Ajouter une acl avec loopback

- Essayer d'ajouter une personne avec une requête de type POST. Utiliser directement votre swagger.

- Que remarquer vous ?

Ajouter le token dans la requête

- Si la ressource demandé est contrôlé avec un token, vous devez y insérer le token afin d'être authentifié au niveau du serveur.
- Pour ajouter un token vous pouvez le faire via un objet HttpParams. Cet objet possède une méthode set à laquelle on passe le nom du token 'access_token' suivi du token.
- Vous devez ensuite l'ajouter comme paramètre à votre requête.

```
const params = new HttpParams()
  .set('access_token', localStorage.getItem('token'));
return this.http.post(this.apiUrl, personne, {params});
```

https://angular.io/guide/http#immutability

Ajouter le token dans la requête

- Une seconde méthode consiste à ajouter dans le header de la requête avec comme name 'Authorization' et comme valeur 'bearer' à laquelle on concatène le Token.
- Pour se faire, créer un objet de type HttpHeaders.
- Utiliser sa méthode append afin d'y ajouter ses paramètres.
- Ajouter la à la requête.

```
const headers = new HttpHeaders();
headers.append('Authorization', 'Bearer ${token}');
return this.http.post(this.apiUrl, personne, {headers});
```

Sécuriser vos routes

- Dans vos applications, certaines routes ne doivent être accessibles que si vous êtes authentifié. Ce cas d'utilisation se répète souvent et s'est un sous cas de la sécurisation de vos routes.

- Angular a pris en considération ce cas en fournissant un mécanisme via l'utilisation des **Guard**.

Guard

- Ce sont des classes qui permettent de gérer l'accès à vos routes.
- Un guard informe sur la validité ou non de la continuation du process de navigation en retournant un booléen, une promesse d'un booléen ou un observable d'un booléen.
- Le routeur supporte plusieurs types de guards, par exemple :
  - CanActivate permettre ou non l'accès à une route.
  - CanActivateChild permettre ou non l'accès aux routes filles.
  - CanDeactivate permettre ou non la sortie de la route.

Guard / canActivate

- Afin d'utiliser le guard canActivate (de même pour les autres), vous devez créer un classe qui implémente l'interface CanActivate et donc qui doit implémenter la méthode canActivate de sorte qu'elle retourne un booléen permettant ainsi l'accès ou non à la route cible. 1

- Vous devez ensuite ajouter cette classe dans le provider. 2

- Finalement pour l'appliquer à une route, ajouter la dans la propriété canActivate. Cette propriété prend un tableau de guard. Elle ne laissera l'accès à la route qu esi la totalité des guard retourne true. 3

Guard / canActivate

1

```
import { Injectable } from '@angular/core';
import {ActivatedRouteSnapshot, CanActivate, RouterStateSnapshot} from '@angular/router';
import {Observable} from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor() {
  }
  // route contient la route appelé
  // state contiendra la futur état du routeur de l'application qui devra passer la validation du guard
  // https://vsavkin.com/routeur-angular-comprendre-l%C3%A9tat-du-routeur-5e15e729a6df
    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean> |
  Promise<boolean> | boolean {
      if (// your condition) {
        return true;
      }
      return false;
    }
  }
```

Guard / canActivate

2

```
providers: [
  TodoService,
  CvService,
  LoginService,
  AuthGuard,
 ],
```

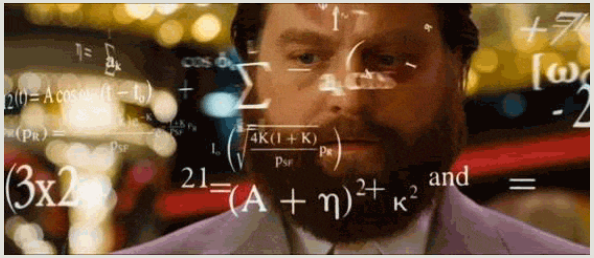
App.module.ts

Guard / canActivate

3

```
{
  path: 'lampe',
  component: ColorComponent,
  canActivate: [AuthGuard]
},
```

Exercice



- Ajouter les guards nécessaire afin de sécuriser vos routes.

- Une personne déjà connectée ne peut pas accéder au composant de login.

Les intercepteurs

- A chaque fois que nous avons une requête à laquelle nous devons ajouter le token, nous devons refaire toujours le même travail.
- Pourquoi ne pas intercepter les requêtes HTTP et leur associer le token s'il est la à chaque fois ?
- Un intercepteur Angular (fournit par le client HTTP) va nous permettre d'intercepter une requête à l'entrée et à la sortie de l'application.
- Un intercepteur est une classe qui **implémente l'interface HttpInterceptor.**
- En implémentant cette interface, chaque intercepteur va devoir implémenter la méthode **intercept**.

Les intercepteurs : changer la requête

- Par défaut la requête est immutable, on ne peut pas la changer.

- Solution : la cloner, changer les headers du clone et le renvoyer.

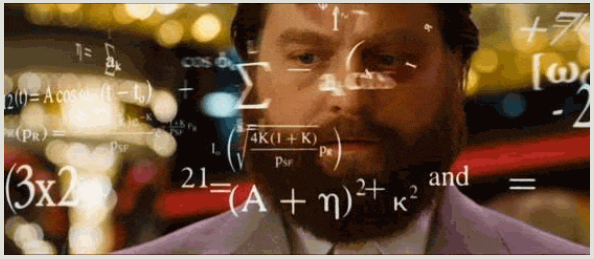
```
export const
AuthentificationInterceptorProvider = {
  provide: HTTP_INTERCEPTORS,
  useClass: AuthentificationInterceptor,
  multi: true,
};
```

```
providers: [
AuthentificationInterceptorProvider
],
```

Cloner une requête

```
 const newReq = req.clone({
    headers: new HttpHeaders()// faites ce que vous voulez ici ajouter des
headers, des params …
  });
// Chainer la nouvelle requete avec next.handle
return next.handle(newReq);
```

Exercice



- Créer un intercepteur qui injecte un token à chaque requête. Si le token n'existe pas on ne faite rien.

Déploiement

- Afin de déployé votre application, il vous suffit d'utiliser la commande suivante :

```
ng build --prod
```

- Un dossier dist sera créer contenant votre projet
- Pour tester localement votre projet, télécharger un serveur HTTP virtuel avec la commande suivante :

```
Npm install http-server –g
```

- Lancer maintenant votre projet à l'aide de cette commande :

```
http-server dist/NomDeVotreProjet
```

Déployer votre application sur GitHub Pages (angularCli >= 8.3

- Créer un repository et mettez y votre projet
- installer angular-cli-ghpages : ng add angular-cli-ghpages
- Ajouter cette configuration dans votre fichier **angular.json**

```
"deploy": {
    "builder": "angular-cli-ghpages:deploy",
}
```

- Vérifier que vous avez déjà effectuer le build de votre applicaction avec ng build --prod
- Lancer command ng build --prod --base-href https://USERNAME.github.io/REPOSITORY_NAME/
- Lancer la commande ng deploy --base-href=/the-repositoryname/
- Accéder à votre page https://USERNAME.github.io/REPOSITORY_NAME
- En cas de mise à jour, relancer le même code.

https://github.com/angular-schule/angular-cli-ghpages

Déployer votre application sur GitHub Pages (angularCli >= 8.3

Ajouter cette configuration dans votre fichier angular.json et utiliser uniquement ng deploy

```
"deploy": {
  "builder": "angular-cli-ghpages:deploy",
  "options": {
    "baseHref": "https://username.github.io/repoName/",
   }
}
```

aymen.sellaouti@gmail.com