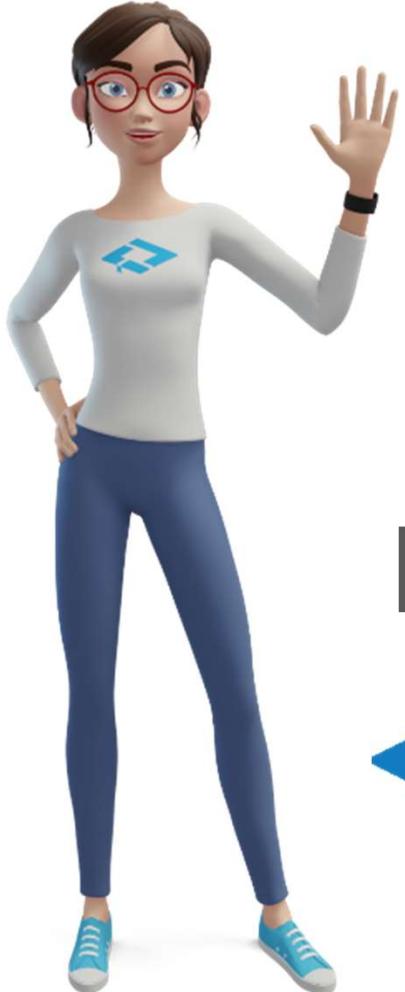




Formation Angular : Développer une application web



Bienvenue !

Formation mixte présentiel et distanciel



plb consultant

L'OFFRE DE FORMATION PLB

Près de 25 ans d'existence et plus de 2 000 formations au catalogue !

PLB dispose d'un catalogue de formation qui répond à la quasi-totalité des besoins de formation IT des entreprises.

Nous proposons des formations 100% développées par PLB

PLB est Organisme ATO (Accredited Training Organisation) et Centre d'examen



PLB est partenaires de centres ATO



Notre offre est classique et digitale dans ses modalités : présentiel, distanciel, blended, @learning, MOOC...

Rechercher une autre formation ?



www.plb.fr

The screenshot shows the homepage of www.plb.fr. A blue callout box points to the search bar, which contains the placeholder text "Intitulé, mot-clé, référence...". Another blue callout box points to the "Toutes nos formations" button in the top left corner of the sidebar. The main content area features a large image of a smiling man wearing glasses. Below the image, the text "+ de 2000 formations Informatique et Management" and "Présentiel, à distance, Blended, e-Learning" is displayed, along with the question "Qu'allez-vous choisir ?". The "Top formations" section displays four cards with training programs:

Formation	Durée	Description
Angular : Développer une application Web	3 jrs	Niveau : Intermédiaire
Préparer la certification DevOps Foundation	2 jrs	Niveau : Intermédiaire Certification : DevOps Foundation Cours officiel : DevOps Institute
Scrum Master - Niveau 1 (PSM1)	2 jrs	Niveau : Fondamental Certification : Professional Scrum Master - Niveau 1 Eligible CPF : Oui
Power BI - Initiation	3 jrs	Niveau : Fondamental Eligible CPF : Oui

 plb consul

Quelques chiffres



FORMATIONS INFORMATIQUE
ET MANAGEMENT DISPONIBLES
À CE JOUR



DE NOS COURS SONT
RÉALISABLES À DISTANCE



STAGIAIRES DANS NOS SALLES
DE COURS PAR AN



DE PARTICIPANTS SATISFAITS
OU TRÈS SATISFAITS
DE NOS FORMATIONS



FORMATEURS EXPERTS
VALIDÉS PLB



FORMATIONS ÉLIGIBLES
AUX DIFFÉRENTS LEVIERS
DE FINANCEMENT (CPF, AIF...)



90 %

DE RÉUSSITE AUX EXAMENS
DE CERTIFICATION



Déroulement de votre formation

▪ Horaires

9h /12h30 ○ Pause déjeuner 1h30 ○ 14h / 17h30

Pause : matin (≈11h) + après midi (≈15h30)

▪ Présentation

Votre formateur

Tour de table : votre profil, vos attentes, vos besoins

▪ Détail du programme

▪ Les objectifs de votre formation

▪ Plateforme formation PLB

Signatures matin + après midi

Avant de commencer : auto - évaluez vos connaissances

Evaluation en fin de formation



VOS CONTACTS CHEZ PLB

Equipe IT assistance@plb.fr

01 43 34 34 10

pour toute question technique
(VM, Teams, login, etc.)

Référent stagiaire rs@plb.fr

pour tout autre sujet ... et vous
accompagner durant votre formation

Service qualité qualite@plb.fr

Pour toute question ou remarque liée à
votre satisfaction

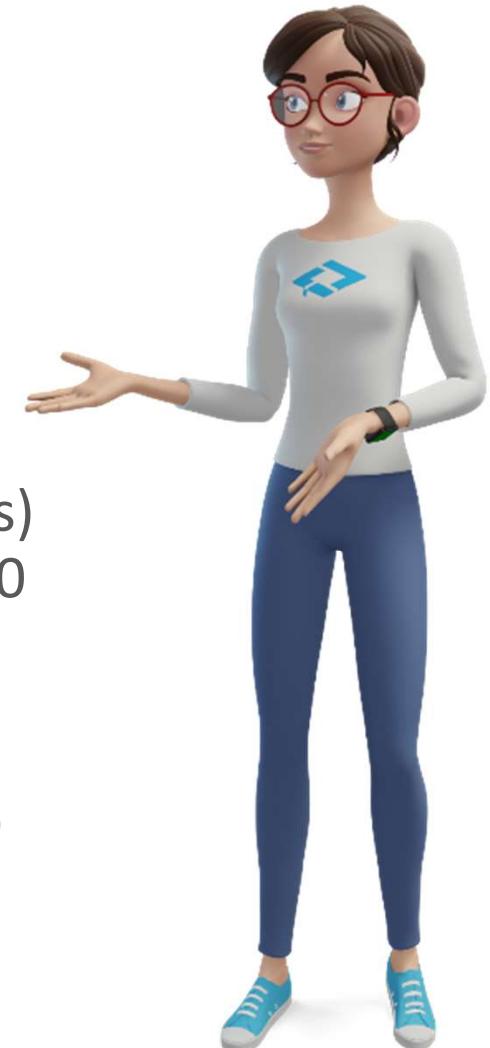
Votre formateur

Aymen SELLAOUTI

- Docteur en informatique
- Enseignant universitaire
- Créeur de contenu (PHP, Symfony, Angular, NestJs) et Fondateur de la chaine Youtube Techwall (+20000 abonnés)

<https://www.youtube.com/c/TechWall>

- Formateur en Fullstack Javascript (Angular, Nest JS)
- Consultant Fullstack Javascript et PHP Symfony



Tour de table

Votre parcours / votre métier

Votre expérience dans le domaine

Vos attentes de la formation

Programme de la formation

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
8. i18n
9. Tests



Objectifs pédagogiques

- Appréhender le Framework Angular
- Comprendre l'architecture d'Angular
- Voir les différentes couches composant le framework
- Créer une application qui permet de regrouper toutes les connaissances acquises le long de la formation

Votre espace formation PLB

- Vous avez reçu le lien pour créer votre identifiant et accéder à votre espace en ligne

The screenshot displays the PLB consultant LMS interface. On the left, a sidebar titled "Le menu de votre espace PLB" shows navigation options: Accès, Mon calendrier, Planning, and Informations. A green box highlights the "Accès" button. Below this, a section titled "Accès à votre espace formation" contains a "Connexion" button, also highlighted with a green box. The main content area starts with a "Bonjour Bertrand!" message and a photo of two people at a desk. It then lists "Votre formation en cours" and "Maîtriser Power BI". Further down, there's a "Dernier message" section with a "Dernier message" button, and a "Tour de table avec découverte des profils et des citoyennetés" section.

Ici les tâches à effectuer
Signature, auto-évaluation, évaluation

The right side of the interface shows a "Tâches à effectuer" section with three buttons: "Signature", "Evaluation auto", and "Evaluation". Below this is a "Demande d'aide" section with a "Demander une réponse" button. The bottom right corner features a large screenshot of a map-based application with various data layers and a search bar.

Vous trouverez également le lien vers le règlement intérieur et toutes les infos pratiques¹¹ si votre formation est en présentiel

Auto-évaluez vos connaissances avant / après la formation sur le LMS

▪ Objectif de ces auto-évaluation individuelles

- Permettre à votre formateur de mieux connaitre votre niveau de connaissance avant de commencer
- Valider la qualité de votre progression en fin de formation

Pour chaque objectif pédagogique de votre formation, **choisissez votre niveau entre 0 et 9 avant la formation**

Vous pourrez remplir votre auto-évaluation à la fin de la formation

Ressources Formation à distance Travaux pratiques **Auto-évaluation** Engagement Evaluation

Les modifications sont automatiquement enregistrées.

L'auto-évaluation vous permet de mesurer votre progression tout au long de la formation. Avant de commencer, vous attribuez une note de 0 à 9 à chaque objectif de la formation. Une fois la formation terminée, vous reévaluez l'objectif en leur attribuant à nouveau une note, ce qui vous permet de constater vos progrès.

Avant la formation

- Automatiser des tests fonctionnels dans de multiples environnements techniques
0 1 2 3 4 **5** 6 7 8 9
- Perfectionner le script de test en plaçant des points de synchronisation et de contrôle
0 1 2 3 4 **5** 6 7 8 9
- Paramétrier le script de test avec des jeux de données
0 1 2 3 4 **5** 6 7 8 9

Après la formation

- Automatiser des tests fonctionnels dans de multiples environnements techniques
0 1 2 3 4 5 6 7 **8** 9
- Perfectionner le script de test en plaçant des points de synchronisation et de contrôle
0 1 2 3 4 5 6 7 **8** 9
- Paramétrier le script de test avec des jeux de données
0 1 2 3 4 5 6 7 **8** 9

Signer

* L'auto-évaluation ne sera plus modifiable une fois signée.



Signatures, documents, évaluations sur le LMS

- Vous avez reçu le lien pour créer votre identifiant et accéder à votre espace en ligne
Les signatures sont obligatoires pour le suivi de votre session de formation

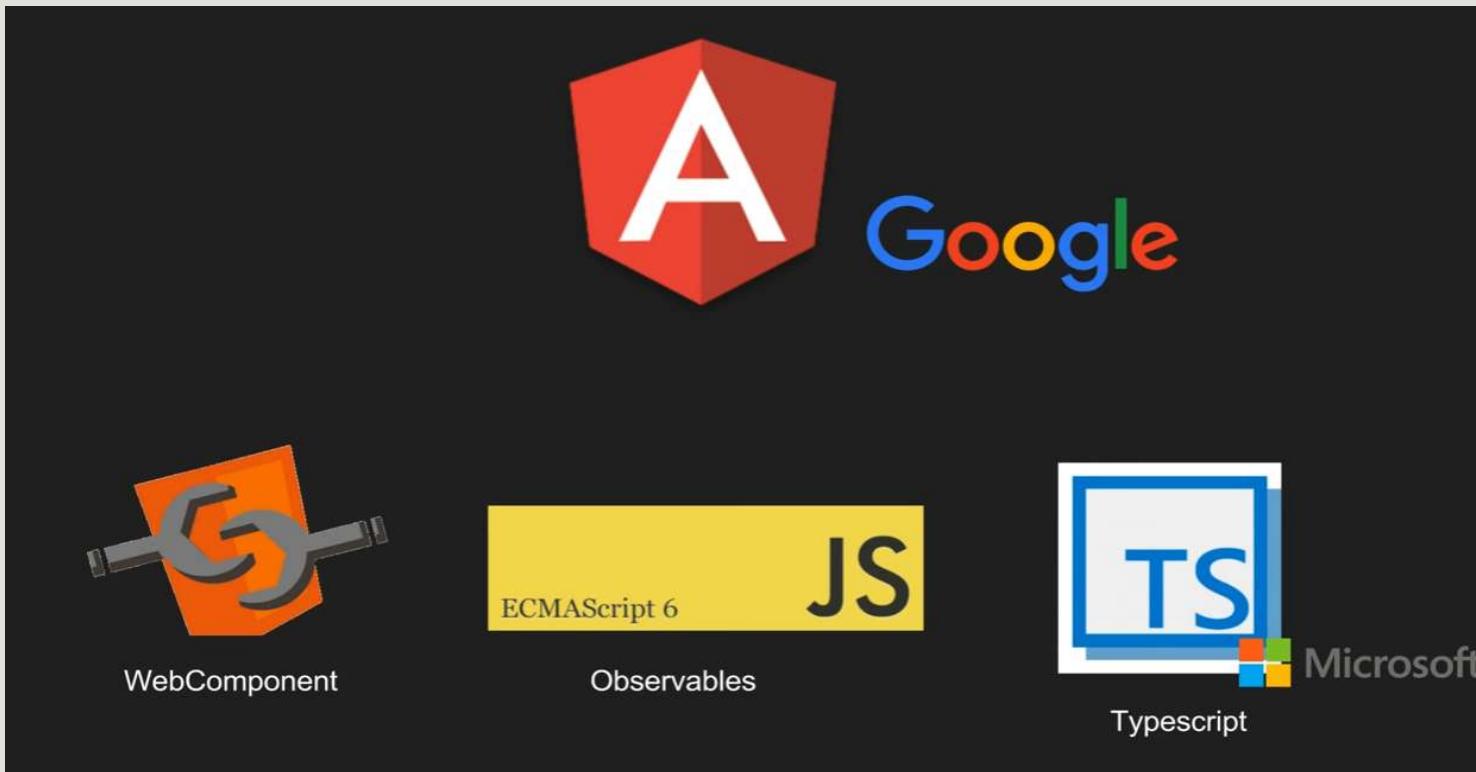
Vous pouvez signer chaque matin + après midi



Vous pourrez remplir votre évaluation le dernier jour

A screenshot of a LMS interface for a course titled "HPUFT - Micro Focus UFT 12 prise en main". The interface includes a sidebar with navigation links like Ressources, Formation à distance, Travaux pratiques, Auto-évaluation, Émplacement, and Evaluation. A green box highlights the "Evaluation" button in the "Evaluation" section. Below the button, there is a message about the evaluation being essential to collect feedback on the course content, trainer, and environment. There is also a section for global satisfaction with five rating options: Très satisfait, Satisfait, Moyen, Peu satisfait, and Insatisfait.

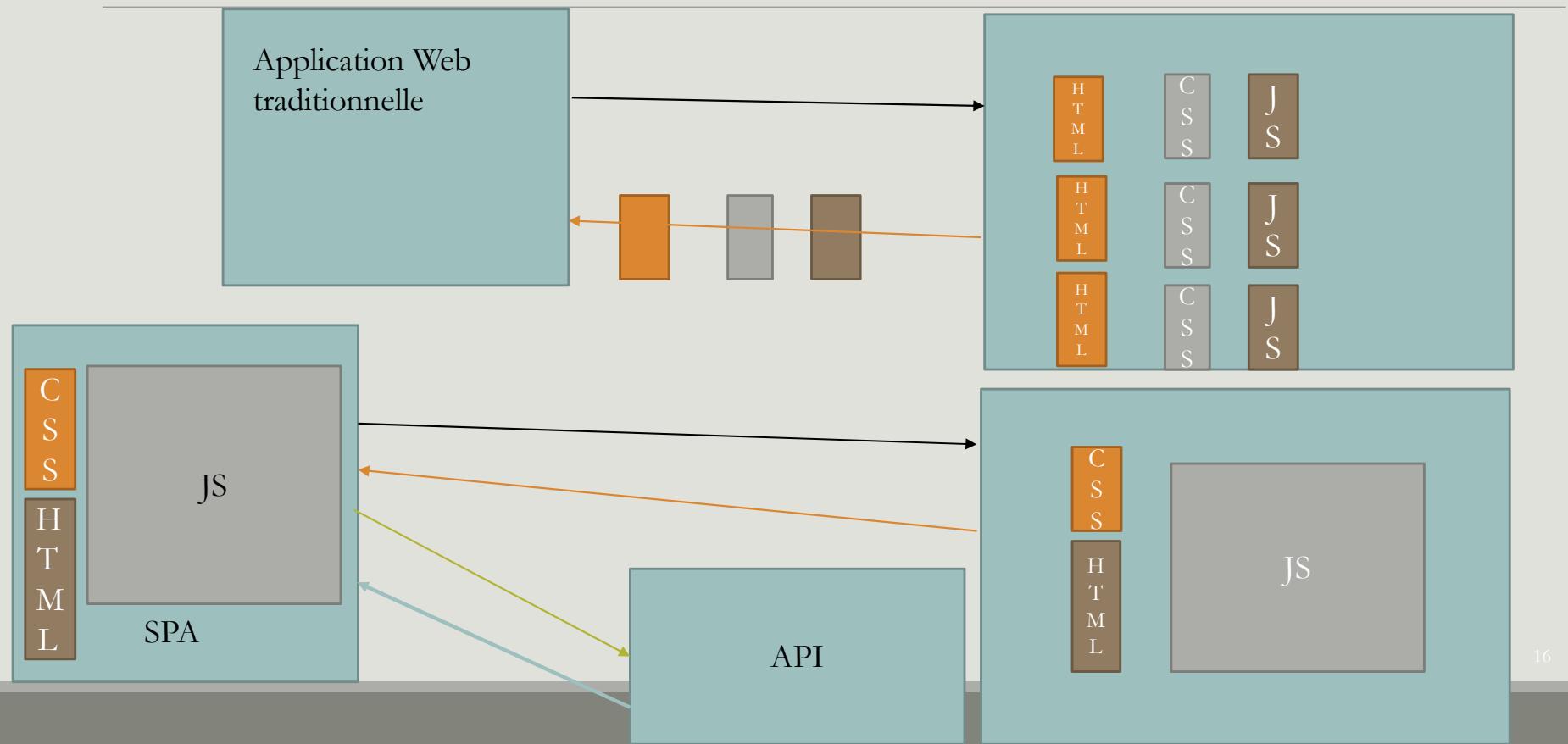
C'est quoi Angular?



C'est quoi Angular?

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

SPA

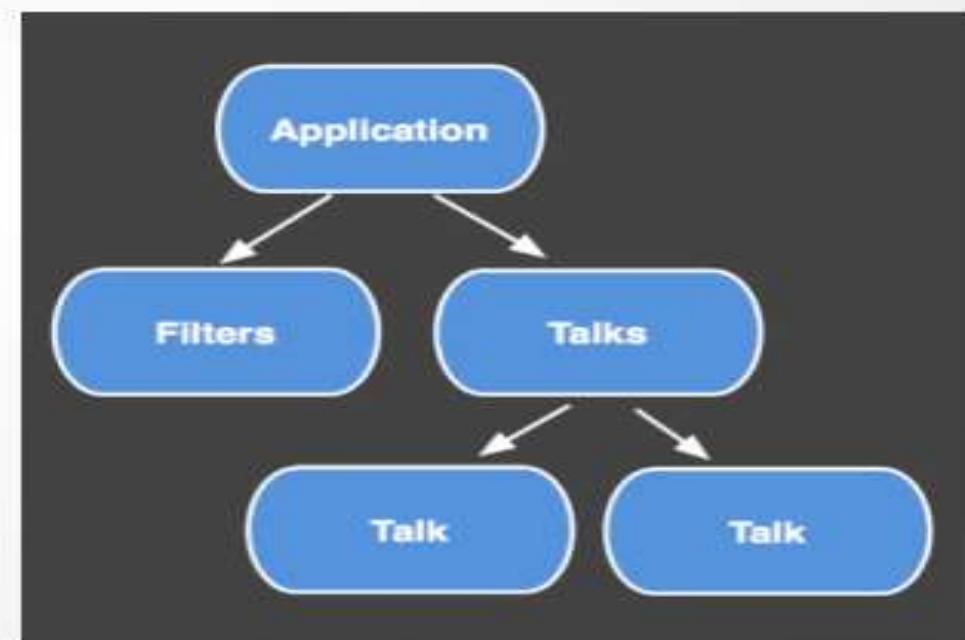


Angular : Arbre de composants

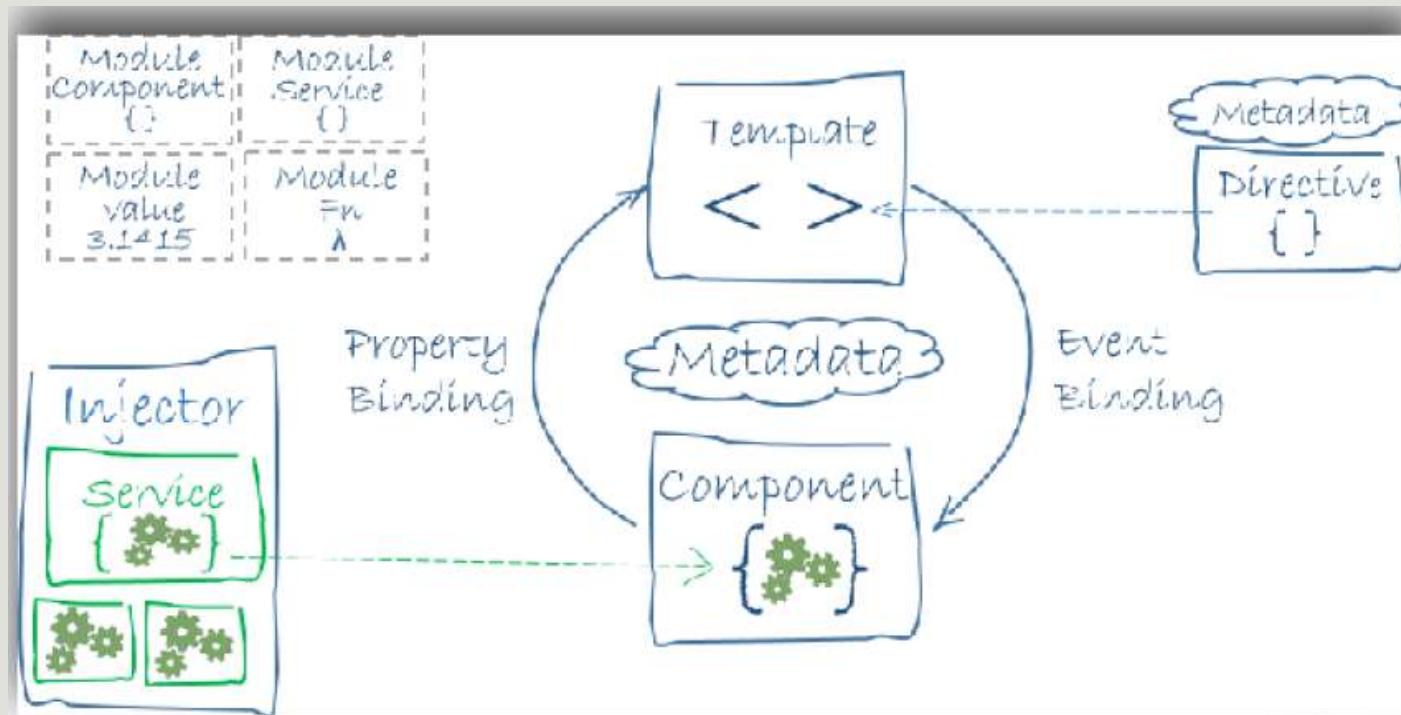
Speaker
Rich Hickey

FILTER

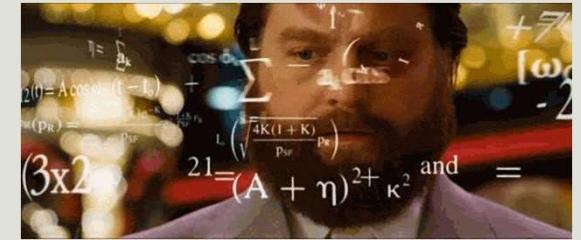
Rating	Title	Speaker	Action
9.1	Are We There Yet?	Rich Hickey	WATCH RATE
8.5	The Value of Values	Rich Hickey	WATCH RATE
8.2	Simple Made Easy	Rich Hickey	WATCH RATE



Architecture Angular



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 [node nécessaire pour la version Angular que vous installez](#).
- 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 maintenant le Cli en tapant la : [npm install -g @angular/cli](#)
 - [npm install -g @angular/cli@16.0.0](#) installe la version 16.0.0
 - [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](#)
- [npx @angular/cli@16.2.15 new nomProjet](#)
- Afin d'avoir du help pour le cli tapez [ng help](#)
- Lancer le projet en utilisant la commande [ng serve](#)

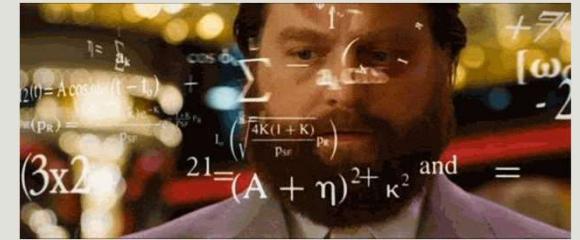
Angular dépendances

	Angular CLI version	Angular version	Node.js version	TypeScript version	RxJS version
29	~10.1.7	~10.1.6	^10.13.0 ^12.11.1	>= 3.9.4 <= 4.0.8	^6.5.5
30	~10.2.4	~10.2.5	^10.13.0 ^12.11.1	>= 3.9.4 <= 4.0.8	^6.5.5
31	~11.0.7	~11.0.9	^10.13.0 ^12.11.1	~4.0.8	^6.5.5
32	~11.1.4	~11.1.2	^10.13.0 ^12.11.1	>= 4.0.8 <= 4.1.6	^6.5.5
33	~11.2.19	~11.2.14	^10.13.0 ^12.11.1	>= 4.0.8 <= 4.1.6	^6.5.5
34	~12.0.5	~12.0.5	^12.14.1 ^14.15.0	~4.2.4	^6.5.5
35	~12.1.4	~12.1.5	^12.14.1 ^14.15.0	>= 4.2.4 <= 4.3.5	^6.5.5
36	~12.2.0	~12.2.0	^12.14.1 ^14.15.0	>= 4.2.4 <= 4.3.5	^6.5.5 ^7.0.1
37	~13.0.4	~13.0.3	^12.20.2 ^14.15.0 ^16.10.0	~4.4.4	^6.5.5 ^7.4.0
38	~13.1.4	~13.1.3	^12.20.2 ^14.15.0 ^16.10.0	>= 4.4.4 <= 4.5.5	^6.5.5 ^7.4.0
39	~13.2.6	~13.2.7	^12.20.2 ^14.15.0 ^16.10.0	>= 4.4.4 <= 4.5.5	^6.5.5 ^7.4.0
40	~13.3.0	~13.3.0	^12.20.2 ^14.15.0 ^16.10.0	>= 4.4.4 < 4.7.0	^6.5.5 ^7.4.0
41	~14.0.7	~14.0.7	^14.15.0 ^16.10.0	>= 4.6.4 < 4.8.0	^6.5.5 ^7.4.0
42	~14.1.3	~14.1.3	^14.15.0 ^16.10.0	>= 4.6.4 < 4.8.0	^6.5.5 ^7.4.0
43	~14.2.0	~14.2.0	^14.15.0 ^16.10.0	>= 4.6.4 < 4.9.0	^6.5.5 ^7.4.0
44	~15.0.0	~15.0.0	^14.20.0 ^16.13.0 ^18.10.0	~4.8.4	^6.5.5 ^7.4.0

<https://gist.github.com/LayZeeDK/c822cc812f75bb07b7c55d07ba2719b3>

Installation d'Angular Angular Cli

- 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

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

Objectifs

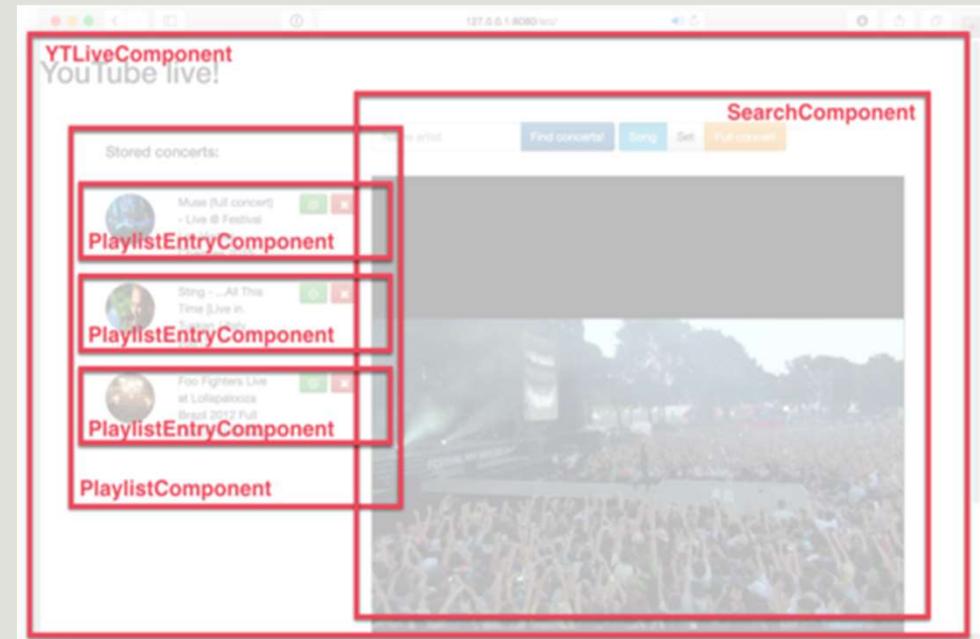
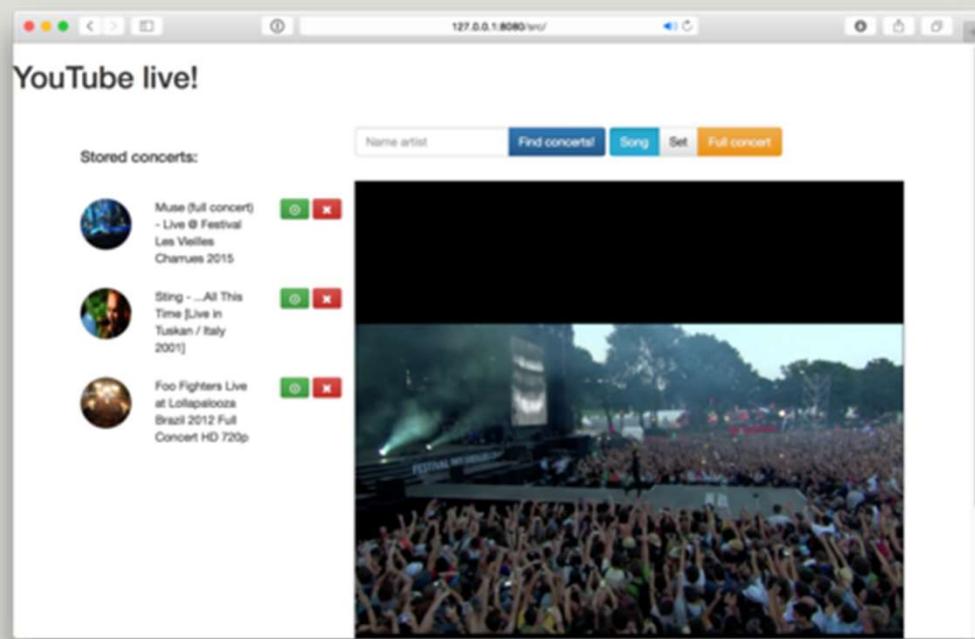
1. Comprendre la définition du composant
2. Assimiler et pratiquer la notion de Binding
3. Gérer les interactions entre composants.

Qu'est ce qu'un composant (Component)

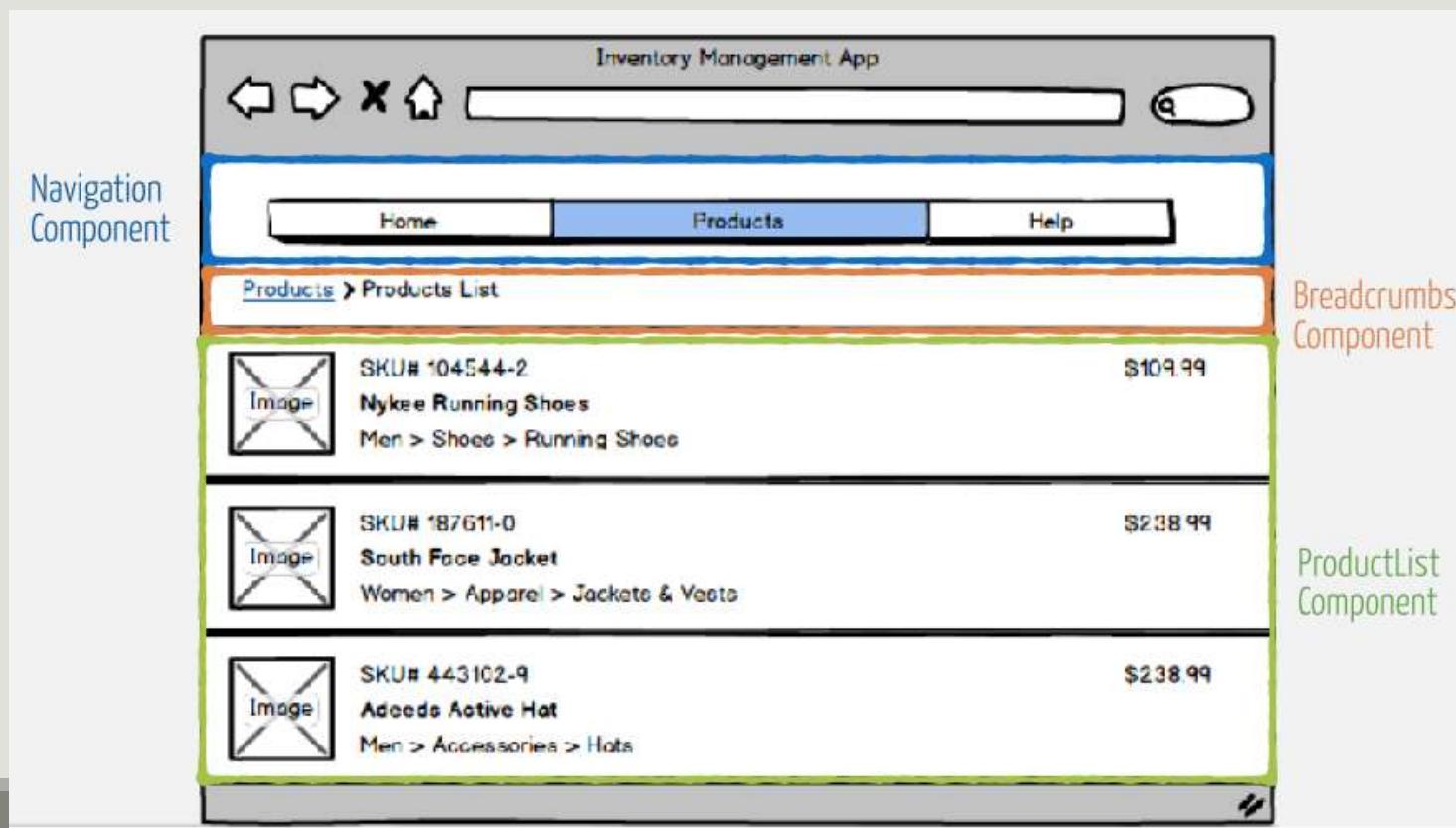
- Un **composant** est une **classe** qui permet de **gérer une vue**. Il se charge **uniquement** de cette vue là.
Plus simplement, un composant est un fragment HTML géré par une classe JS (component en angular et controller en angularJS)
- Une application Angular est un **arbre de composants**
- La 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

Product Row Component

 Image	SKU# 104544-2 Nykee Running Shoes Men > Shoes > Running Shoes	\$109.99
 Image	SKU# 187611-0 South Face Jacket Women > Apparel > Jackets & Vests	\$238.99
 Image	SKU# 443102-9 Adeeds Active Hat Men > Accessories > Hats	\$238.99

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 generate component my-new-component** ou son raccourci **ng g c my-new-component**

Création d'un composant

- La commande `generate` possède plusieurs options

OPTION	DESCRIPTION
<code>--inlineStyle=true false</code>	Inclus les styles css dans le composant Aliases: <code>-s</code>
<code>--inlineTemplate=true false</code>	Inclus le template dans le composant Aliases: <code>-t</code>
<code>--prefix=prefix</code>	Le préfixe à appliquer pour la génération des composants Valeur par défaut: app Aliases: <code>-p</code>

Property Binding



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.
- syntaxe: [propriété]="**varOuCte**"

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

Event Binding

- Binding unidirectionnel.
- Permet d'interagir du DOM vers le composant.
- L'interaction se fait à travers les événements.
- Syntaxe : (evenement)="fct()">>

```
a (click)="goToCv()" >Go to Cv</a>
```

Property Binding et Event Binding

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

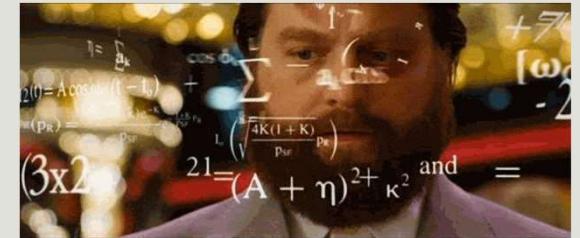
@Component({
  selector: 'inter-interpolation',
  template : `interpolation.html` ,
  styles: []
})
export class InterpolationComponent {
  nom:string ='Aymen Sellaouti';
  age:number =35;
  adresse:string ='Chez moi ou autre part :)';
  getName() {
    return this.nom;
  }
  modifier(newName) {
   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

Component

```
<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.backgroundColor]**

Two way Binding

- Binding Bi-directionnel
- Permet d'interagir du Dom vers 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';

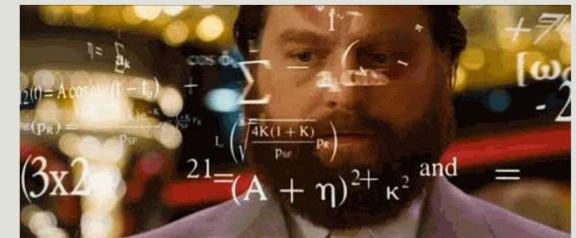
@Component({
  selector: 'app-two-way',
  templateUrl: './two-
way.component.html',
  styleUrls: ['./two-
way.component.css']
})
export class TwoWayComponent {
  two:any="myInitValue";
}
```

Component

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

Template

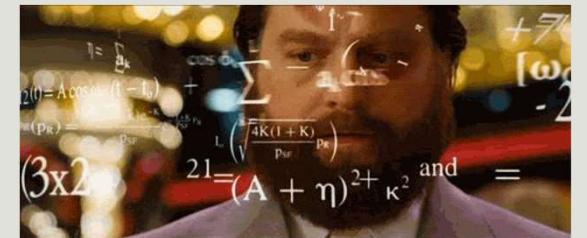
Exercice



- 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



A profile card for Aymen Sellaouti, trainer. It features a circular profile picture of a man in a black jacket and striped shirt. Below the picture, the name "Aymen Sellaouti" and title "trainer" are displayed. A quote at the bottom reads: "I'm the new Sinatra, and since I made it here I can make it anywhere, yeah, they love me everywhere". At the bottom of the card is a "Auto Rotation" button.

name :
sellouati

firstname :
aymen

job :
trainer

path :
rotating_card_profile3.png

Exercice

Two Way Binding

Two Way Binding Example



Sellaouti Aymen
Enseignant

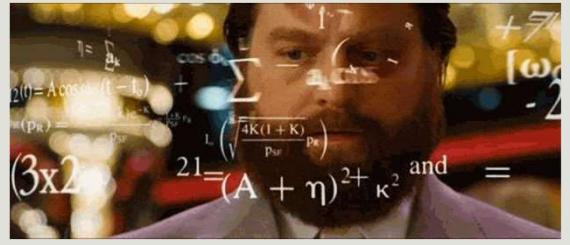
tant qu'il y a de la vie il y a de l'espoir

Auto Rotation

Nom :	<input type="text" value="Sellaouti"/>
Prénom :	<input type="text" value="aymen"/>
Job :	<input type="text" value="Enseignant"/>
image :	<input type="text" value="as.jpg"/>
Citation Favorite :	<input type="text" value="tant qu'il y a de la vie il y a de l'espoir"/>
Décrivez nous votre travail :	<input type="text" value="J enseigne aux étudiants les technos du Web"/>
Mots clé de votre travail :	<input type="text" value="HTML CSS JS PHP Symfony Angular"/>

Two Way Binding

Two Way Binding Example



"To be or not to be, this is my awesome motto!"

J enseigne aux étudiants les technos du Web

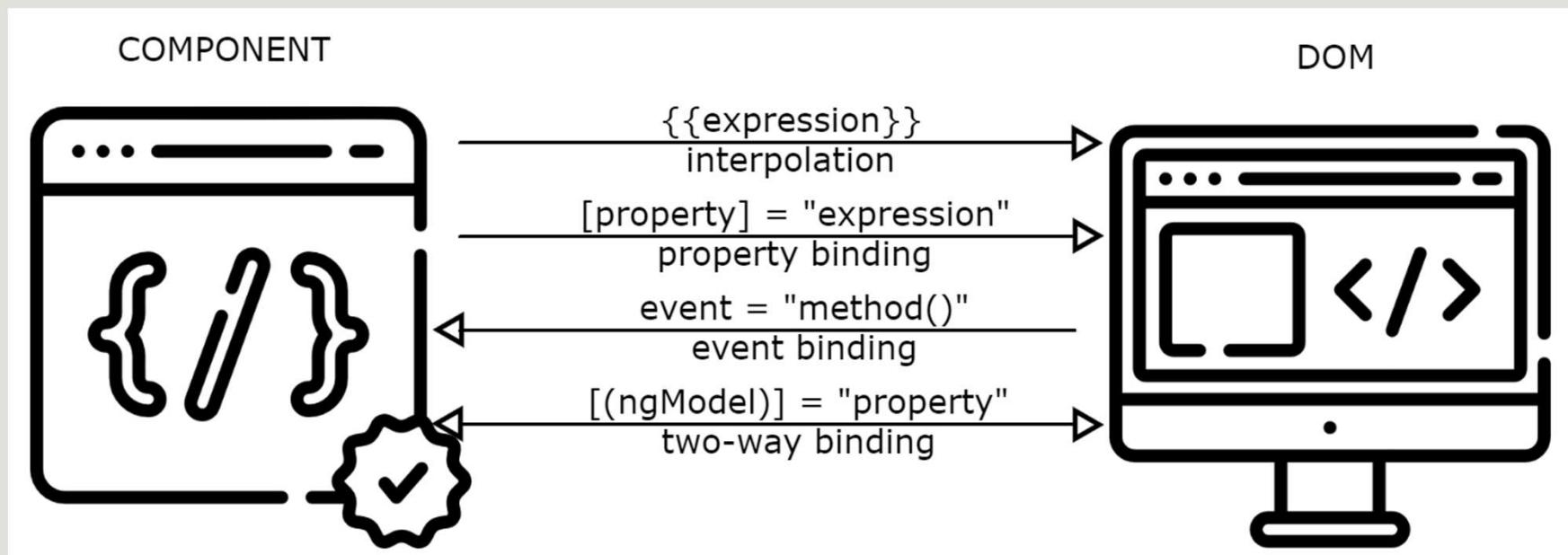
HTML CSS JS PHP Symfony Angular

235 Followers 114 Following 35 Projects

f g+ t

Nom :	<input type="text" value="Sellaouti"/>
Prénom :	<input type="text" value="aymen"/>
Job :	<input type="text" value="Enseignant"/>
image :	<input type="text" value="as.jpg"/>
Citation Favorite :	<input type="text" value="tant qu'il y a de la vie il y a de l'espoir"/>
Décrivez nous votre travail :	<input type="text" value="J enseigne aux étudiants les technos du Web"/>
Mots clé de votre travail :	<input type="text" value="HTML CSS JS PHP Symfony Angular"/>

Résumé : Property Binding



Récap Binding

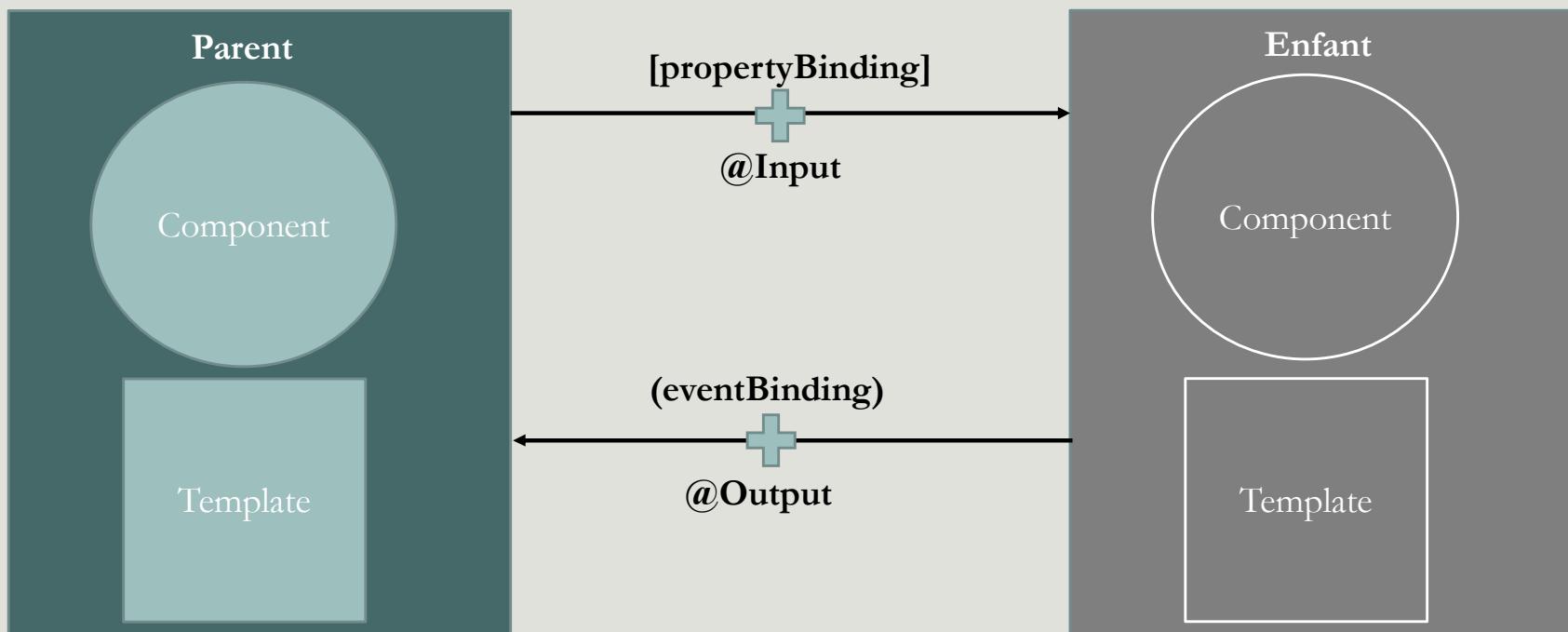
```
<div [style.backgroundColor]="color">  
  Color  
</div>  
  
<input [(ngModel)]="color"  
        type="text"  
        class="form-control"  
>  
le contenu de la propriété color est {{color}}  
<button (click)="loggerMesData()">log data</button>  
<br>  
<a (click)="goToCv()">Go to Cv</a>
```

HTML

```
@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);  
  }  
}
```

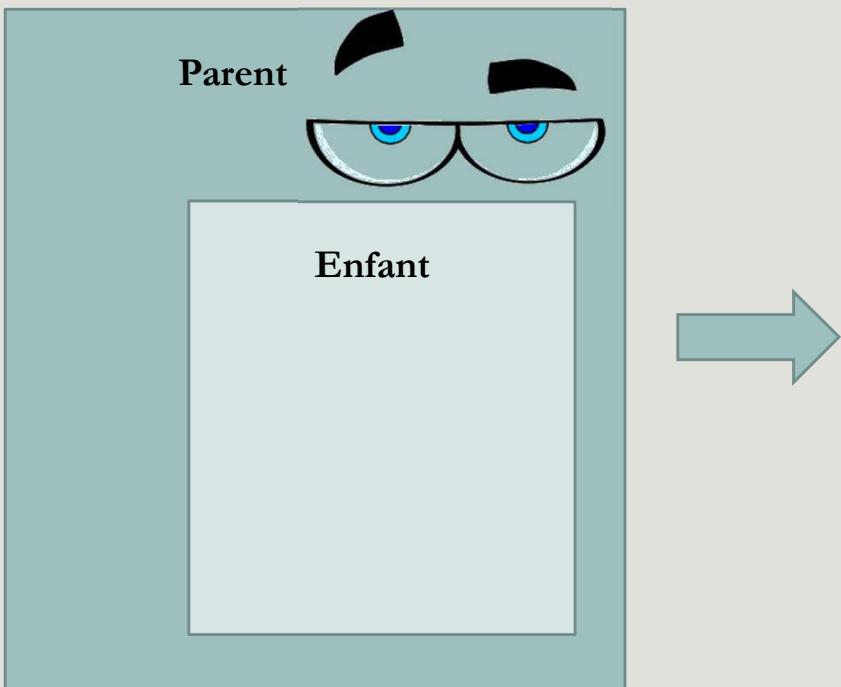
TS

Interaction entre composants



Pourquoi ?

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

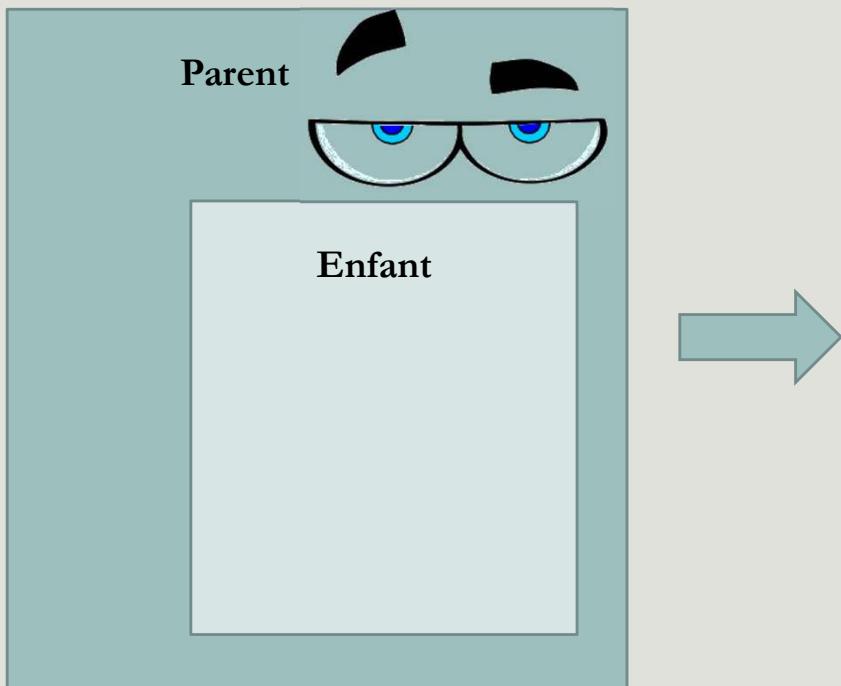


```
import { Component } from '@angular/core';

@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

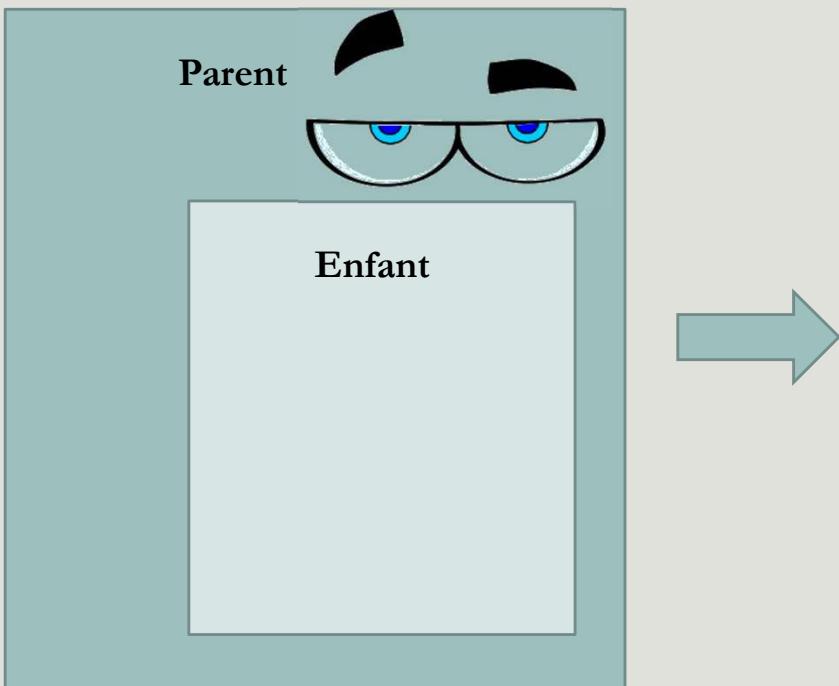


```
import { Component } from '@angular/core';

@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

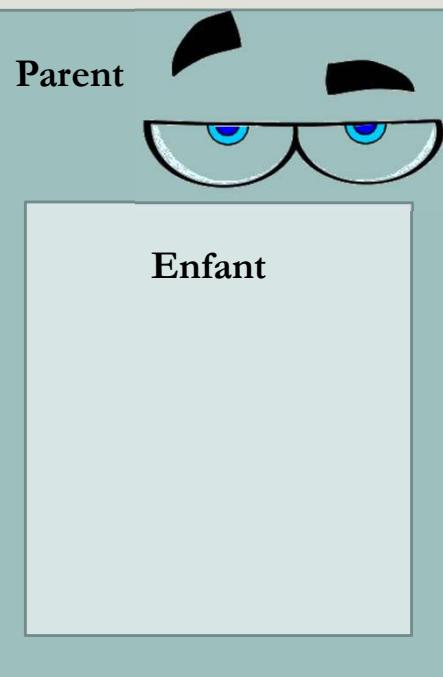


```
import { Component } from '@angular/core';

@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';

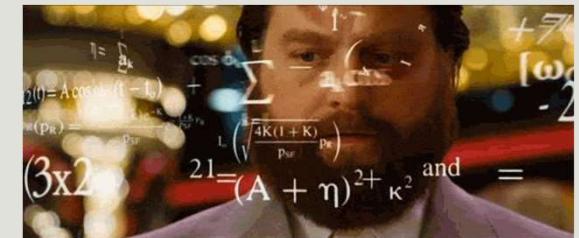
@Component({
  selector: 'app-root',
  template: `
    <p>Je suis le composant père </p>
    <forma-fils [external]="title">
    </forma-fils>
  `,
  styles: []
})
export class AppComponent {
  title = 'app works !';
}
```

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

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

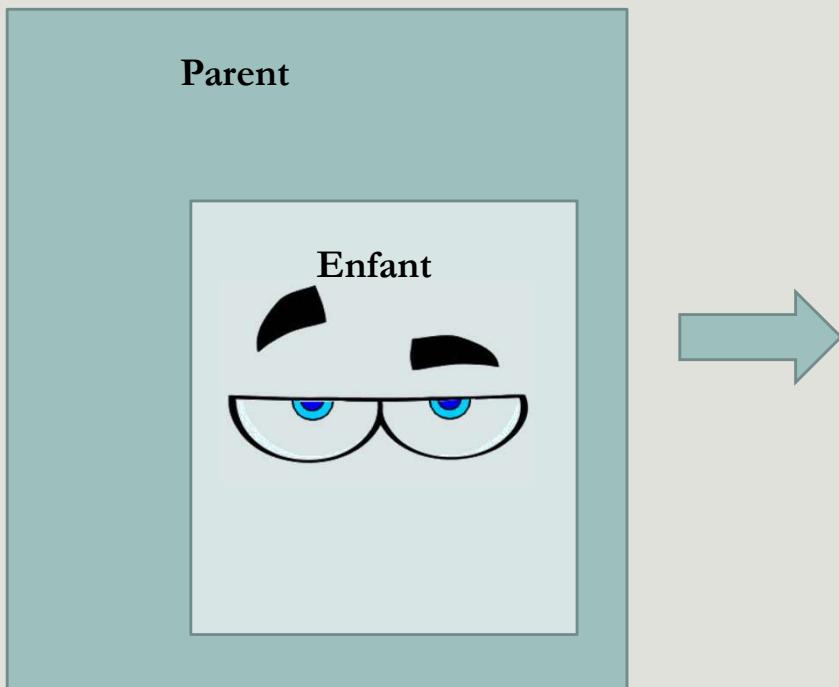
Exercice

- 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

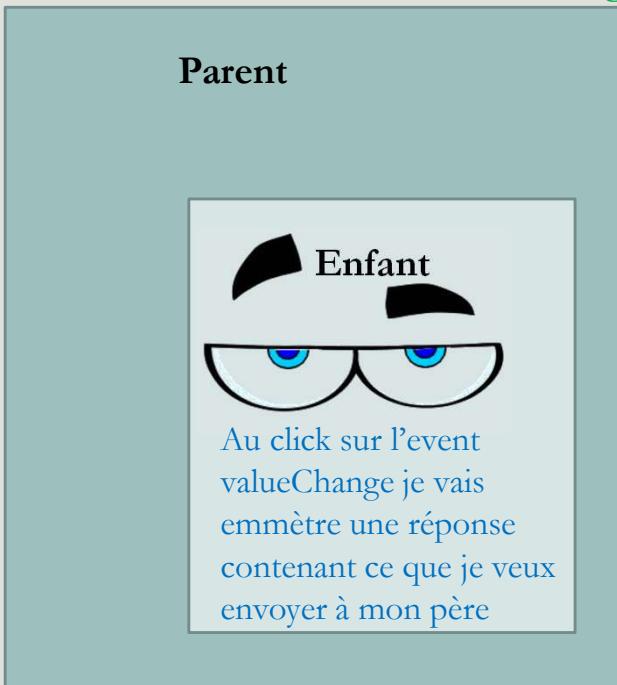
L'enfant ne peut pas voir le parent. Appel de l'enfant vers le parent. Que faire ?



```
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 sûrement 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 passer les informations.

Parent



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

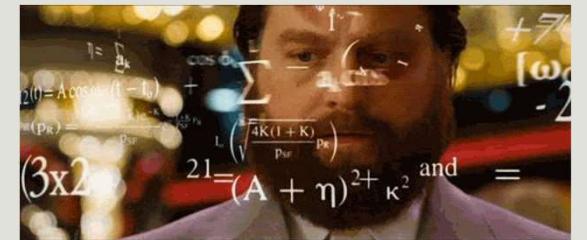
```
import {Component, EventEmitter, Output} from
'@angular/core';
@Component({
  selector: 'bind-output',
  template: `
<button (click)="incrementer()">+</button> `,
  styleUrls: ['./output.component.css']
})
export class OutputComponent {
  valeur:number=0;
  // On déclare un événement
  @Output()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

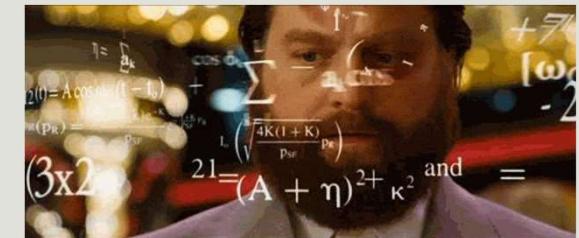
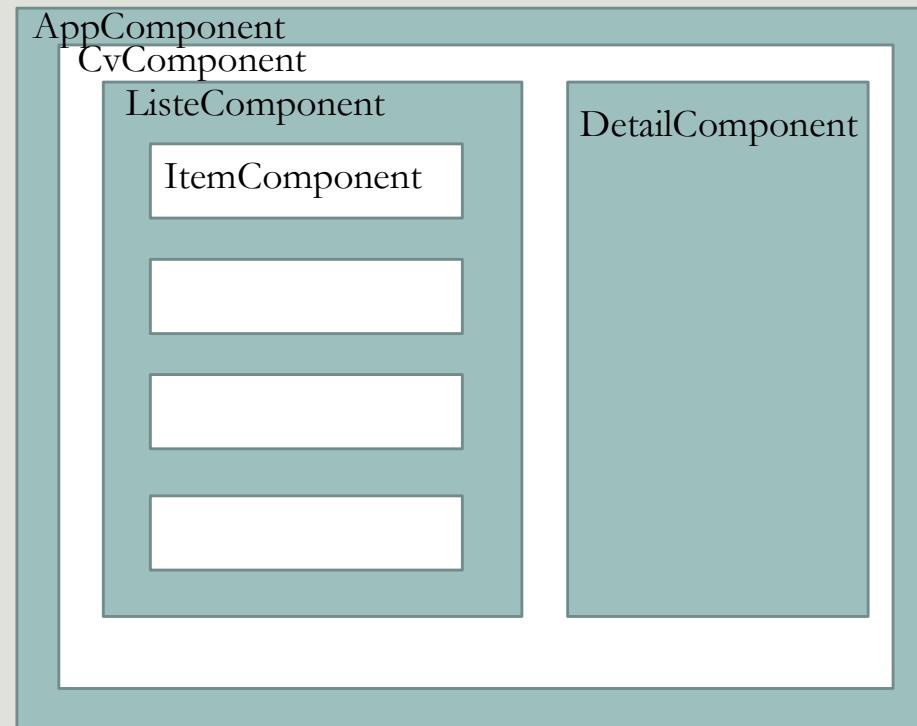
Exercice



- 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.

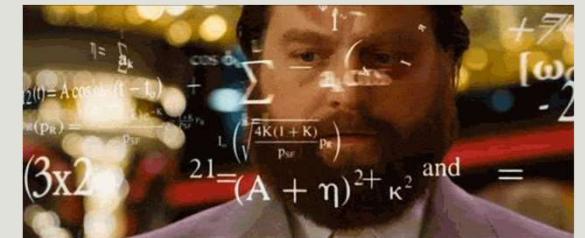
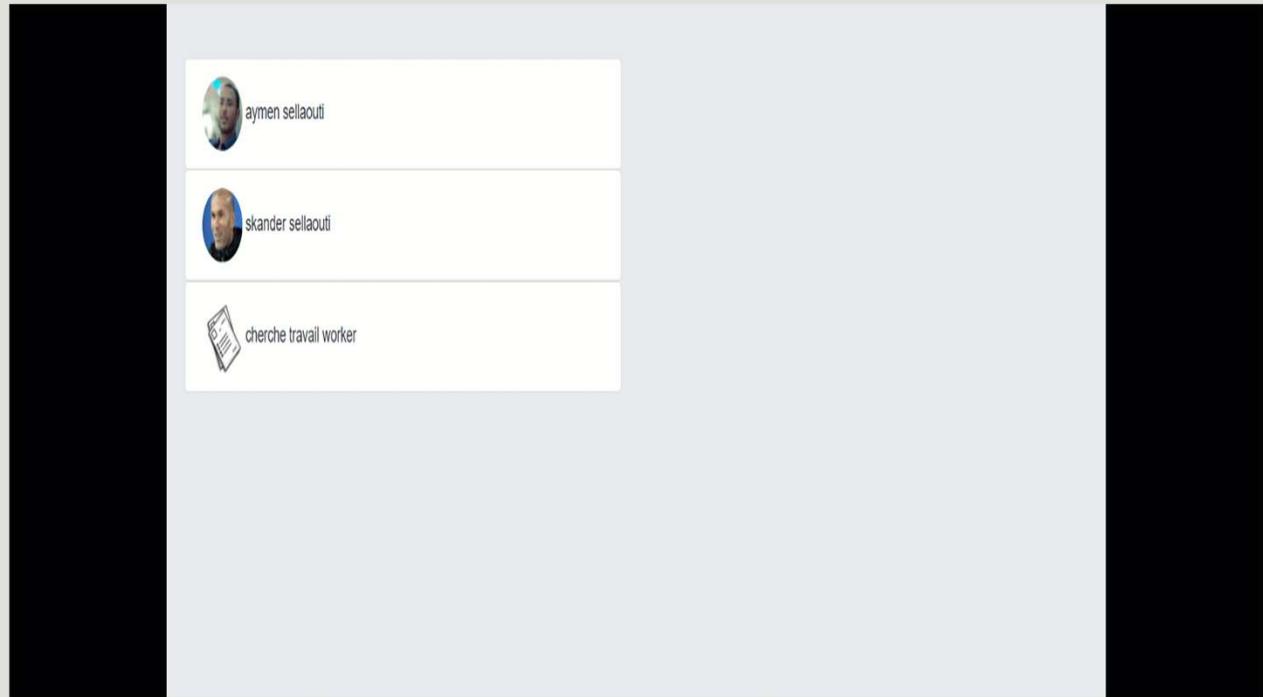
Exercice

- 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 suivre cette architecture.

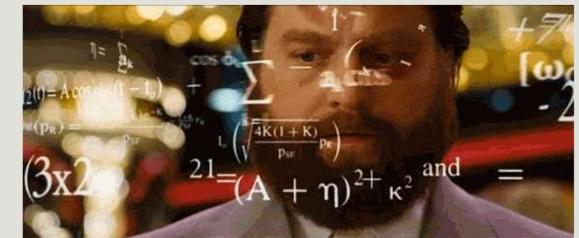
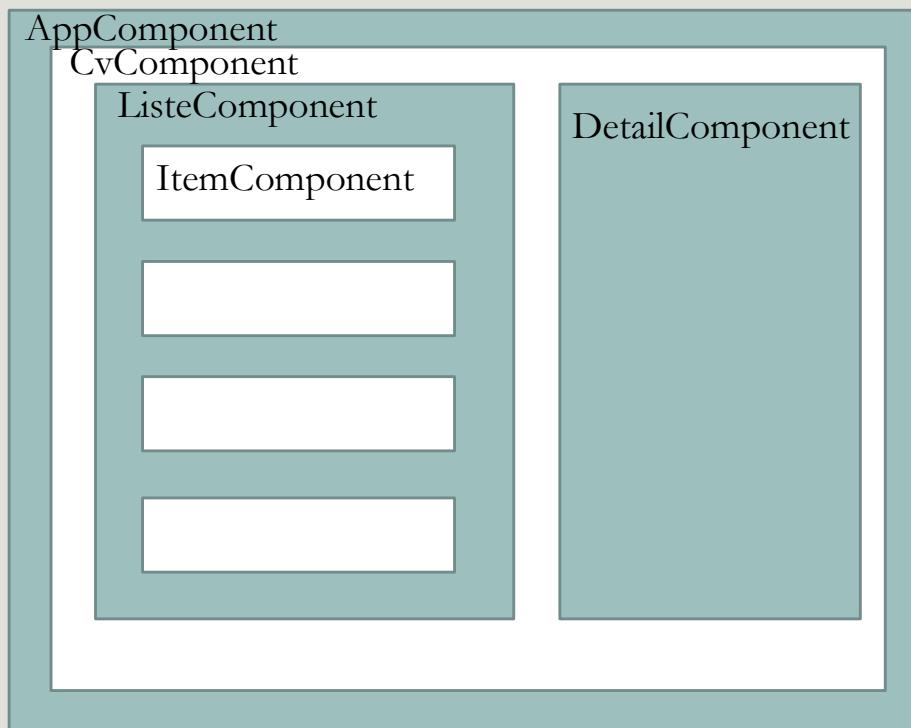


Exercice

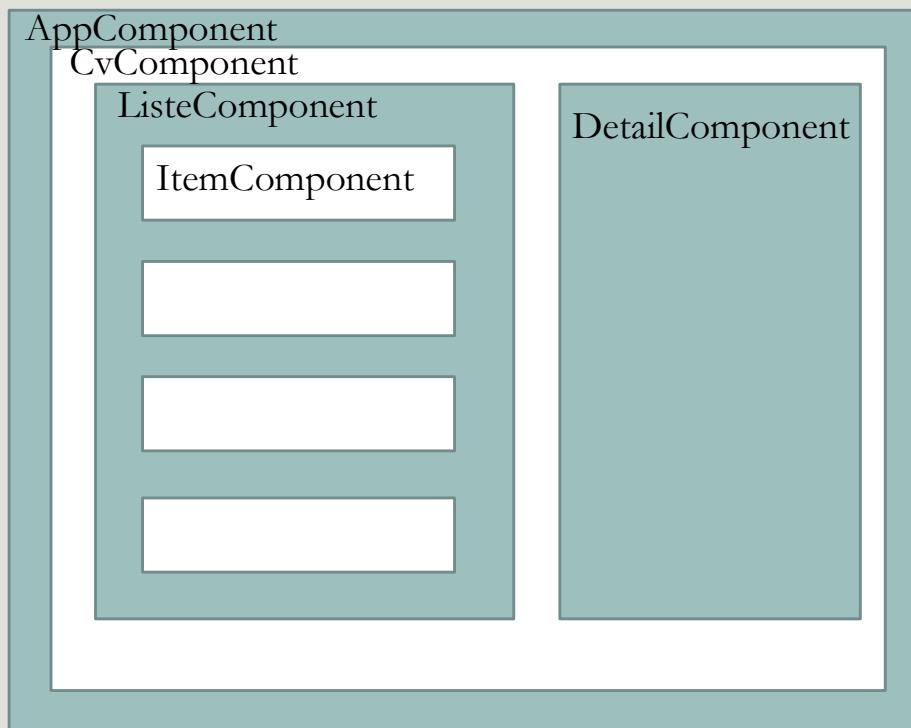
- Le but de cet exercice est de créer une mini plateforme de recrutement.
- La première étape est de créer la structure suivante avec une vue contenant deux parties :
 - Liste des Cvs inscrits
 - Détail du Cv qui apparaitra au click
- Il vous est demandé juste d'afficher un seul Cv et de lui afficher les détails au click.
Il faudra suivre cette architecture.



Exercice



Exercice

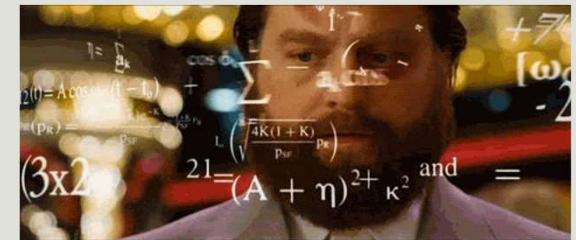


A screenshot of a mobile application interface. On the left, there is a list of users with their profile pictures and names: "sellaouti aymen" and "sellaouti skander". On the right, there is a larger circular profile picture of a man, identified as "Aymen Sellaouti Teacher". Below the list, a red text message says "Au click sur le Cv les détails sont affichés". At the bottom right, there is a "Auto Rotation" button.

Exercice

Un cv est caractérisé par :

- id
- name
- firstname
- Age
- Cin
- Job
- path





sellaouti aymen



sellaouti skander



Aymen Sellaouti
Teacher

Au click sur le Cv les détails sont affichés

 Auto Rotation

Angular Les directives

AYMEN SELLAOUTI

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**.
- Apparaît dans un élément comme un **tag** (comme le font les **attributs**).
- La commande 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**.

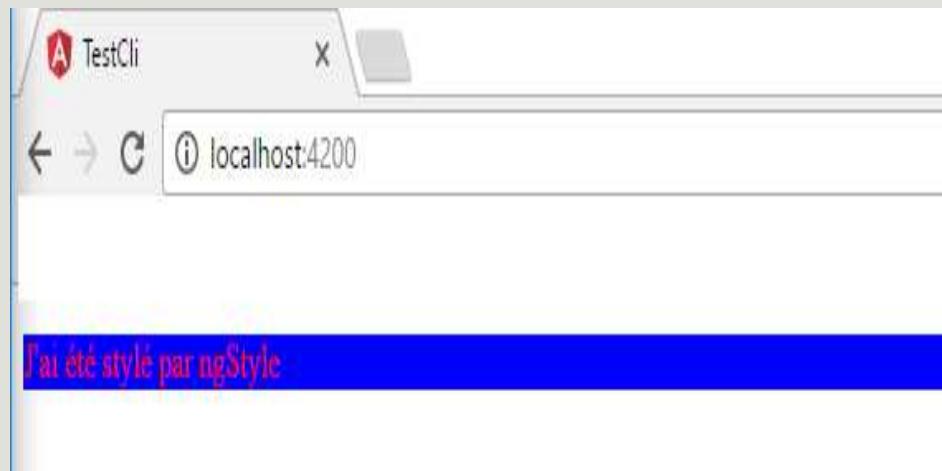
Les directives d'attribut (ngStyle)

```
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)

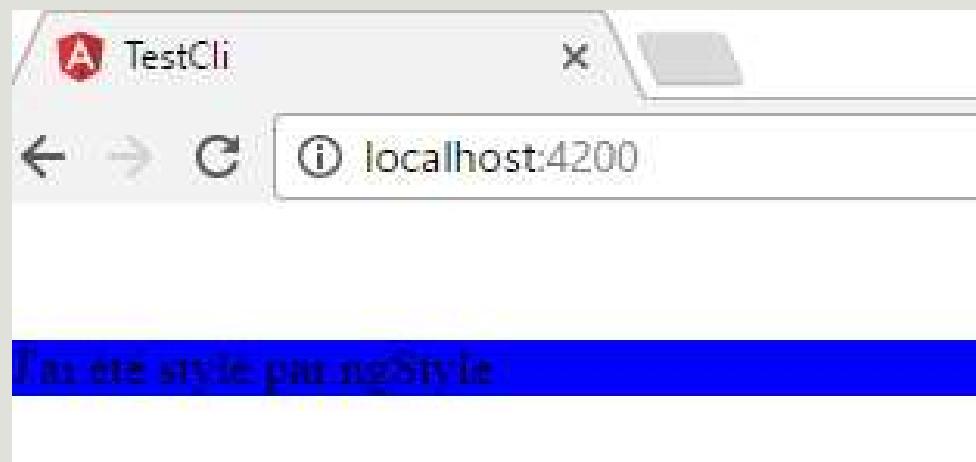


Les directives d'attribut (ngStyle)

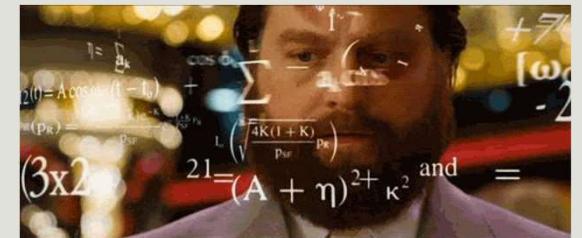
```
@Component({
  selector: 'app-root',
  template: `
    <direct-direct [myColor]="gray">J'ai
été stylé par ngStyle</direct-direct>
  `,
  styles: [
    h1 { font-weight: normal; }
    p{color:yellow;background-color: red}
  ],
})
export class AppComponent { }
```

```
import { Component, Input } from
'@angular/core';
@Component({
  selector: 'direct-direct',
  template: `
    <p [ngStyle]="{{'color':myColor,
    'font-family':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 color, un input de type number, et un select box.
- Faites en sorte que lorsqu'on écrit une couleur dans le texte input, ça 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.

Test

Les directives d'attribut (ngClass)

- Cette directive permet de modifier **l'attribut class**.
- Elle cohabite avec l'attribut **class**.
- Elle prend en paramètre
 - Une chaîne (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**.

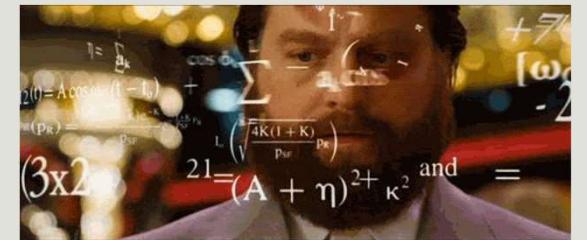
Les directives d'attribut (ngClass)

```
import {Component, Input} from '@angular/core';
@Component({
  selector: 'direct-direct',
  template: `
    <div ngClass="colorer arrierman" class="encadrer">
      test ngClass
    </div>
  `,
  styles: [
    .encadrer{ border: inset 3px black; }
    .colorer{ color: blueviolet; }
    .arrierman{background-color: salmon; }
  ]
})
export class DirectComponent{
  private myfont:string="garamond";
  @Input() private myColor:string="red";
  private myBackground:string="blue";
  private isColoree:boolean=true;
  private isArrierman:boolean=true
}
```

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

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

Exercice

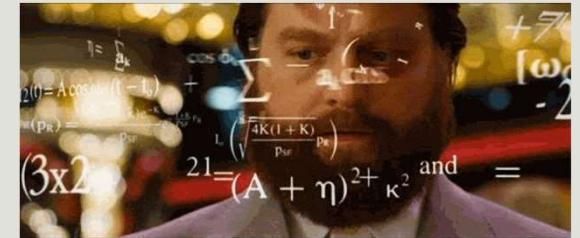


- Préparer 3 classes présentant trois thèmes différents (couleur font-size et font-police)
- Au choix du thème votre cible changera automatiquement

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 `HostListener` 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`

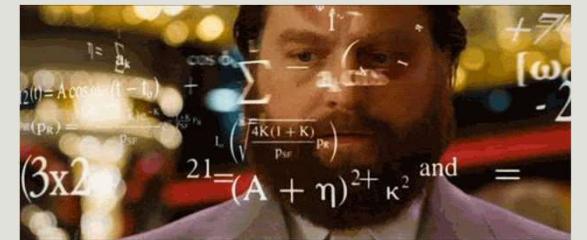
Exercice



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 keyup) 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.

Exercice



```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <div class="container">
      <app-root _ngcontent-pfp-c0 ng-version="8.2.14">
        <app-ng-style _ngcontent-pfp-c0 _ngcontent-pfp-c1>
          ... <input _ngcontent-pfp-c1 apprainbow class="form-control" style="border-color: rgb(125, 162, 230); color: rgb(125, 162, 230);"> == $0
        </app-ng-style>
      </app-root>
    </div>
  </body>
</html>
```

Styles Computed Event Listeners DOM Breakpoints »

Filter :hover .cls +

```
element.style {
  border-color: □rgb(125, 162, 230);
  color: □rgb(125, 162, 230);
}
```

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">**

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

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>
```

Les directives structurelles *ngFor

- Permet de répéter un élément plusieurs fois dans le DOM.
- Prend en paramètre les entités à reproduire.
- Fournit certaines valeurs :

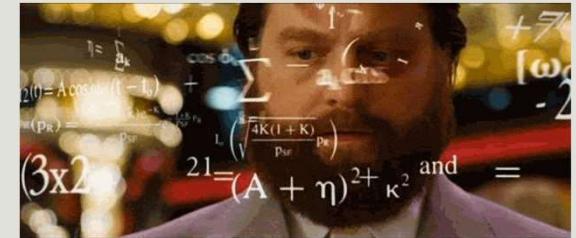
- index : position de l'élément courant
- first : vrai si premier élément
- last vrai si dernier élément
- even : vrai si l'indice est paire
- odd : vrai si l'indice est impaire

```
<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 (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.



localhost:4200

applications template html 5 res... enseignement utilities Dashboard sales ... AllDebrid: Accès au... Site de collaboration ser ram balis selection Robin des Droits - L... Fiche Google Play S...

CvTech Home Cv Color Task Manager Poc Add Students Login

 Aymen Sellaouti

 Zineddine Zidan

83

Angular

Les pipes

AYMEN SELLAOUTI

Plan du Cours

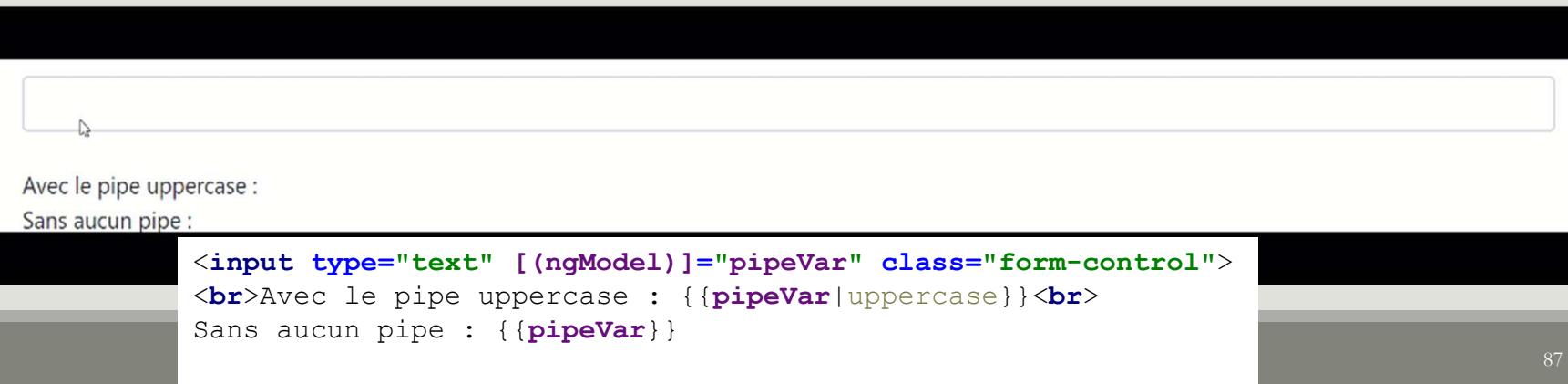
1. Introduction
 2. Les composants
 3. Les directives
- 3 Bis. Les pipes
4. Service et injection de dépendances
 5. Le routage
 6. Form
 7. HTTP

Objectifs

1. 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**.



A screenshot of a web browser window. At the top, there is a navigation bar with a logo and some text. Below it is a search bar with a magnifying glass icon. The main content area shows a text input field with the placeholder "Search". To the right of the input field, the text "Avec le pipe uppercase :" is displayed in purple. Below that, the text "Sans aucun pipe :" is also displayed in purple. At the bottom of the page, there is a dark footer bar containing some code snippets and text.

```
<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
- ...

Paramétriser un pipe

- Afin de paramétriser 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.
- Pour 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 {

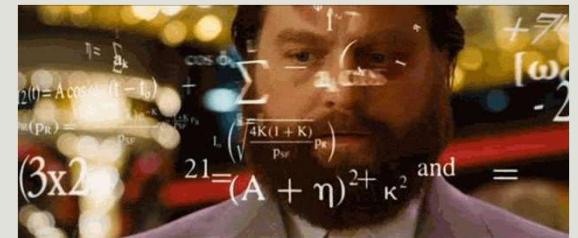
  transform(value: any, args?: any): any {
    switch (value) {
      case 'barca' : return ' blaugrana';
      case 'roma' : return ' giallorossa';
      case 'milan' : return ' rossoneri';
    }
  }
}
```

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

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

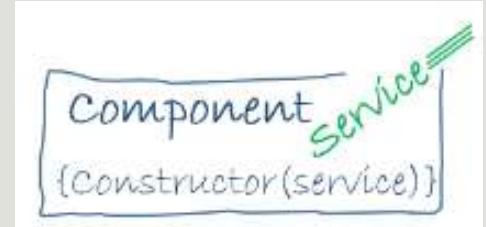
Exercice

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



Angular Service et injection de dépendances

AYMEN SELLAOUTI



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

Component 1

```
f(){};  
g(){};  
k(){};
```

Component 2

```
f(){};  
g(){};  
l(){};
```

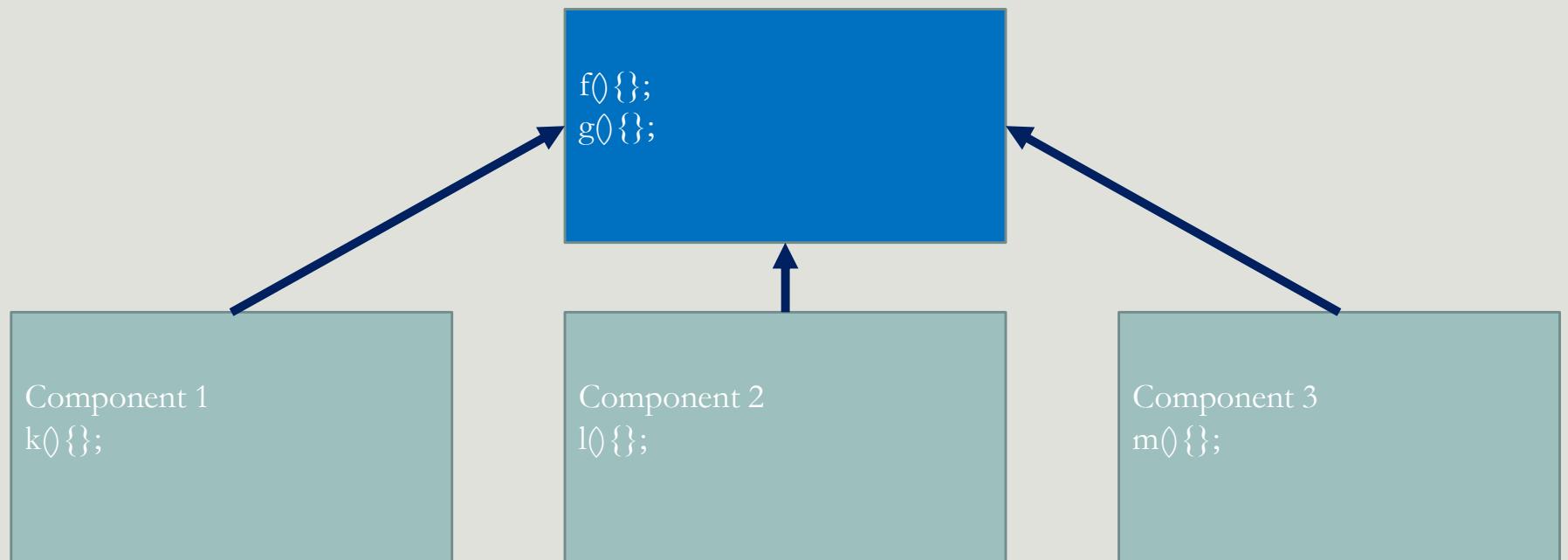
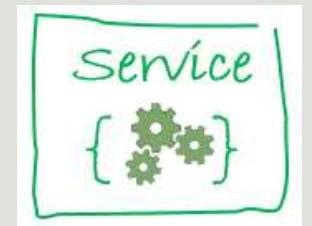
Component 3

```
f(){};  
g(){};  
m(){};
```

Redondance de code

Maintenabilité difficile

Qu'est ce qu'un service ?



Qu'est ce qu'un service ?



- Un service peut :
- 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({
  providedIn: 'root'
})
export class FirstService {

  constructor() { }

}
```

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.

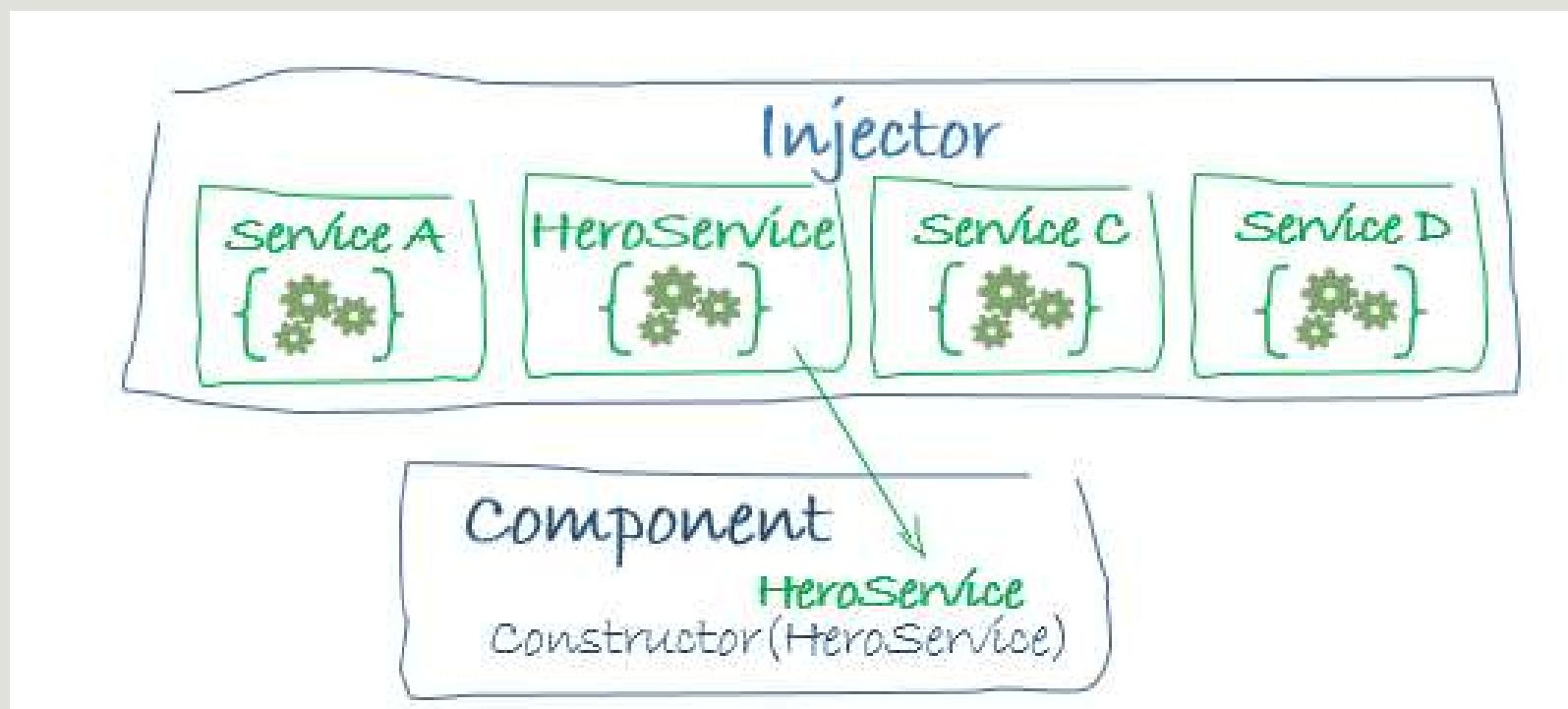
```
Classe A1{  
    Constructor(B b, C c)  
    ...  
}
```

```
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 où est-il visible ?

Injection de dépendance (DI)

- L'injection de dépendance utilise les étapes suivantes :
 - Déclarer le service via l'annotation `@Injectable`, 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'importez 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

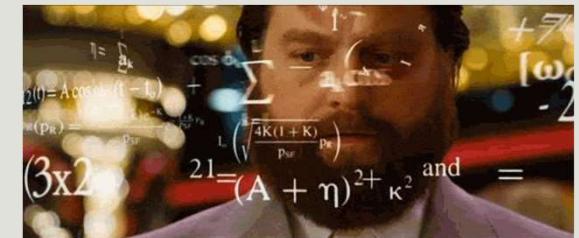
- Lazy loading : Ne charger le code des services qu'à la première injection
- Permettre le **Tree-Shaking** des services non utilisés : Si le service n'est jamais utilisé, son code ne sera entièrement retiré du build final.

@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'allez injecter aucun service dans votre service, cette annotation n'est plus nécessaire.
- **Remarque** : Angular conseille de toujours mettre cette annotation.

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 :
 - Logger les todos
 - Ajouter un Todo
 - Récupérer la liste des Todos
 - Supprimer un Todo



Name :
I

Content :

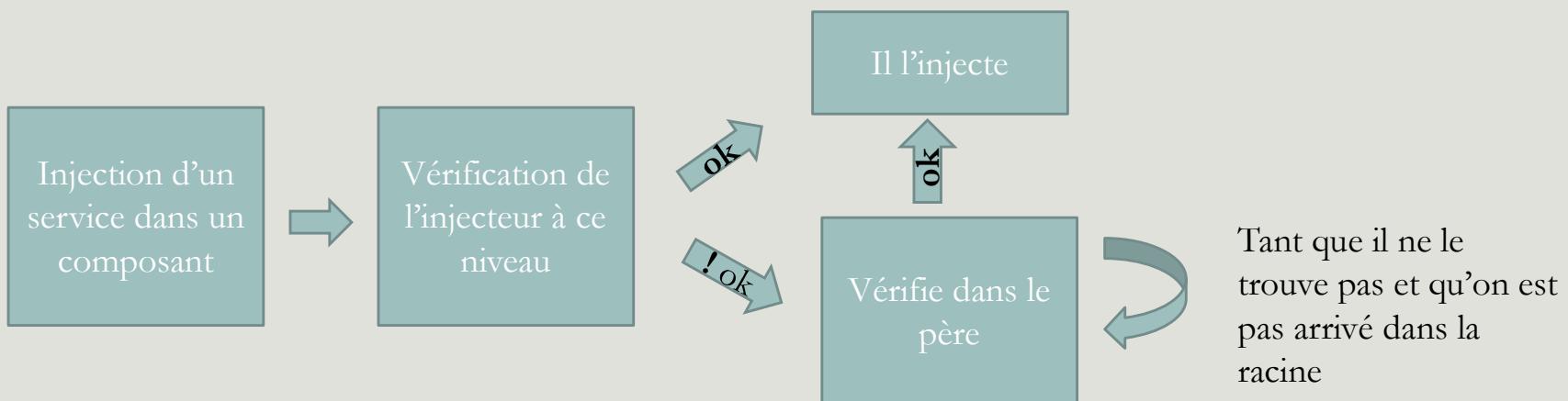
Add Todo

Exemple

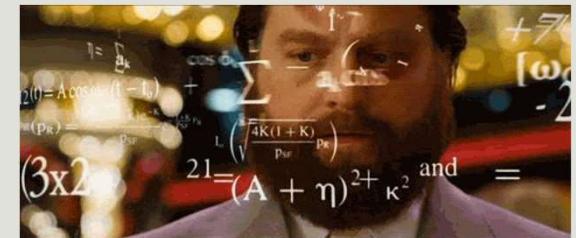
```
import { Injectable } from '@angular/core';
@Injectable()
export class LoggerService {
  constructor() { }
  Logs: string[]=[];
  log(message:string) {
    this.Logs.push(message); console.log(message);
  }
  info(message:string) {
    this.Logs.push(message); console.info(message);
  }
  debug(message:string) {
    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 créé. Cet arbre est // à l'arbre de composant.
- L'algorithme suivant permet la détection de l'injecteur adéquat :

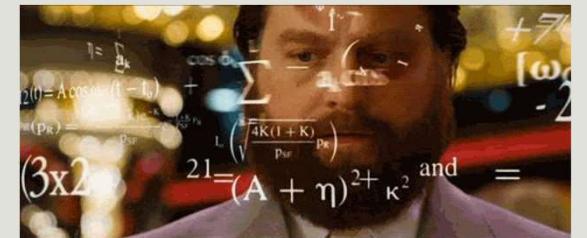


Exercice



- 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

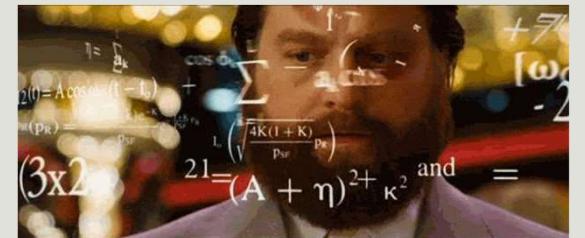
Embaucher

Liste des cvs sélectionnés pour embauche

aymen sellaouti



Exercice



applications template html 5 res... enseignement utilities Dashboard sales ... AllDebrid: Accès au... décès Site de collaboration serram bals sélection Robin des Droits - L... Fiche Google Play S...

aymen sellaoui

skander sellaoui

cherche travail worker

Angular Routing

AYMEN SELLAOUTI

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 inexistantes

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 rafraîchir 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

```
<!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>
```

1

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

2

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

```
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);
```

App.routing.ts

2

3

4

```
import { BrowserModule, } from  
'@angular/platform-browser';  
import {CUSTOM_ELEMENTS_SCHEMA, NgModule} from  
'@angular/core';  
import { FormsModule } from '@angular/forms';  
import { HttpClientModule } from '@angular/http';  
import { AppComponent } from './app.component';  
import {routing} from "./app.routing";  
  
@NgModule({  
  declarations: [  
    AppComponent,  
  ],  
  imports: [  
    BrowserModule,  
    FormsModule,  
    HttpClientModule,  
    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 où 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 où 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>
```

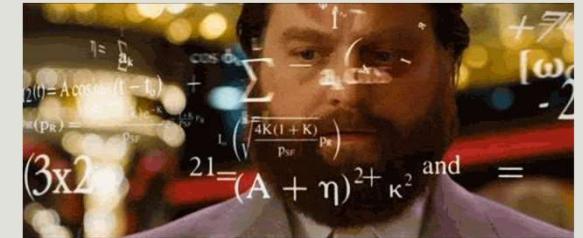
Syntaxe minimaliste d'une route

- Une route est un objet.
- Les deux propriétés essentielles sont `path` et `component`.
- `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 ça 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 :
- ```
<a [routerLink]="'todo'" routerLinkActive="active">Gérer les
cvs
```

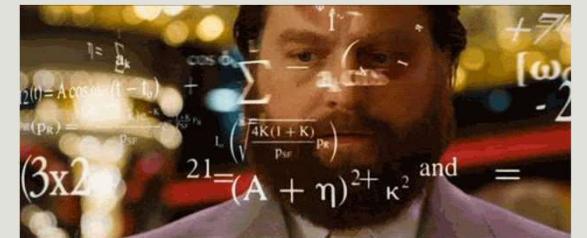
# Déclencher une route routerLink

---

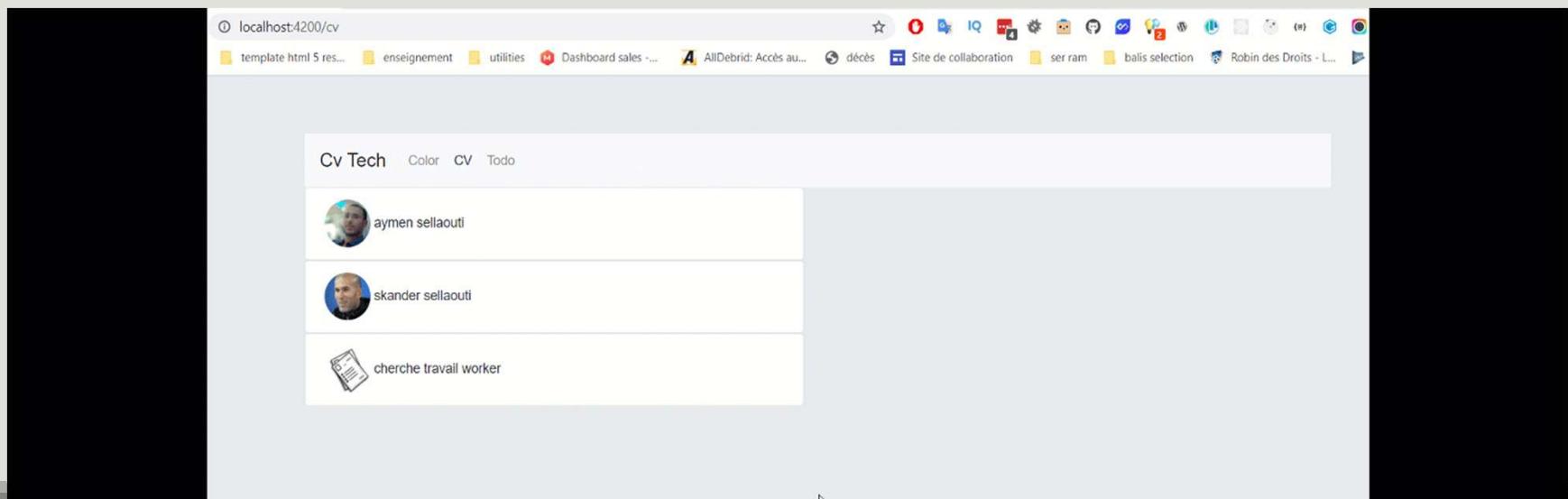
- **routerLinkActive="active"** va associer la classe active à l'uri cible ainsi qu'à tous ces ses ancêtres.
- Par exemple si on a l'uri 'cv/liste' la classe active sera ajouté à cet uri ainsi qu'à l'uri 'cv' et 'list'.
- Pour identifier uniquement l'uri cible, ajouter la directive suivante :

[routerLinkActiveOptions] = "{exact: true}"

# Exercice



- Faites en sorte d'avoir un composant Header 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 le service **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) { }
 onNavigate() {
 this.router.navigate(['/about/10']);
 }
}
```

# 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. Utilisez l'objet **snapshot**

# Récupérer les paramètres d'une route ActivatedRoute / snapshot

---

- La propriété **snapshot** de l'objet **ActivatedRoute** contient un instantané de l'état de la route actuelle.
- Elle contient des **informations** telles que **l'URL actuelle, les paramètres de route actuels,...**
- Elle est généralement utilisée pour **accéder aux informations de route** dans un composant **lors de son initialisation**.
- Il est important de noter que **l'instantané de la route ne change pas lorsque la route change**. Il représente un **état figé** de la route lors de son instantiation.

# Récupérer les paramètres d'une route

## ActivatedRoute / snapshot

---

- Voici quelques propriétés courantes de l'API snapshot :
  - **url**: Retourne l'URL de la route actuelle sous forme de tableau de segments d'URL.
  - **params**: Retourne un objet qui contient les paramètres de route actuels.
  - **queryParams**: Retourne un objet qui contient les paramètres de requête actuels.
  - **fragment**: Retourne la partie de l'URL après le symbole "#".
  - **data**: Retourne les données de route associées à la route actuelle.
  - **component**: Retourne le composant de route actuel.
  - **routeConfig**: Retourne la configuration de la route actuelle.

# Récupérer les paramètres d'une route ActivatedRoute / snapshot

```
▼ snapshot: ActivatedRouteSnapshot
 ► component: class DetailsCvComponent
 ► data: {cv: ...}
 fragment: null
 outlet: "primary"
 ► params: {id: '27'}
 ► queryParams: {}
 ▼ routeConfig:
 ► component: class DetailsCvComponent
 path: ":id"
 ► resolve: {cv: f}
 ► [[Prototype]]: Object
 ► url: [UrlSegment]
 _lastPathIndex: 1
 ► _paramMap: ParamsAsMap {params: ...}
 ► _resolve: {cv: f}
 ► _resolvedData: {cv: ...}
 ► _routerState: RouterStateSnapshot {_root: TreeNode, url: '/cv/2'}
 ► _urlSegment: UrlSegmentGroup {segments: Array(2), children: ...}
 ► children: Array(0)
 firstChild: null
 ▼ paramMap: ParamsAsMap
 ► params: {id: '27'}
 keys: ...
 ► [[Prototype]]: Object
 parent: (...)
```

# Récupérer les paramètres d'une route

## ActivatedRoute / snapshot

---

- Donc pour accéder à votre propriété, passez par l'objet `snapshot`
- Avec `snapshot`, vous avez deux méthodes pour récupérer les paramètres:
- Via la **propriété `params`** qui retourne un tableau d'objet des paramètres
- Via la propriété **`paramMap`**
  - Appeler sa méthode `get`
  - Passez lui le nom de la propriété souhaitée.

```
this.activatedRoute.snapshot.params['id']
```

```
this.activatedRoute.snapshot.paramMap.get('id')
```

# 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;
 onNavigate() {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
```

# Récupérer Les queryParameters

---

- Les **queryParameters** sont récupérable de la même façon que les paramètres. Soit d'une façon statique avec **snapshot via la propriété queryParams** ou sa propriété **queryParamMap** et sa méthode **get**.
- Soit dynamiquement via l'observable **queryParams**

```
this.activatedRoute.snapshot.queryParams['page']
```

```
this.activatedRoute.snapshot.queryParamMap.get('page')
```

# La route joker

---

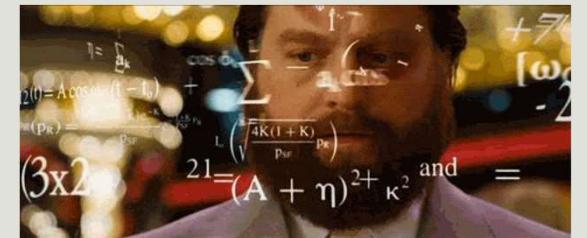
- Il existe une route **joker** qui **matche n'importe quelle autre route**.  
C'est la route **'\*\*'**.

# Exemple

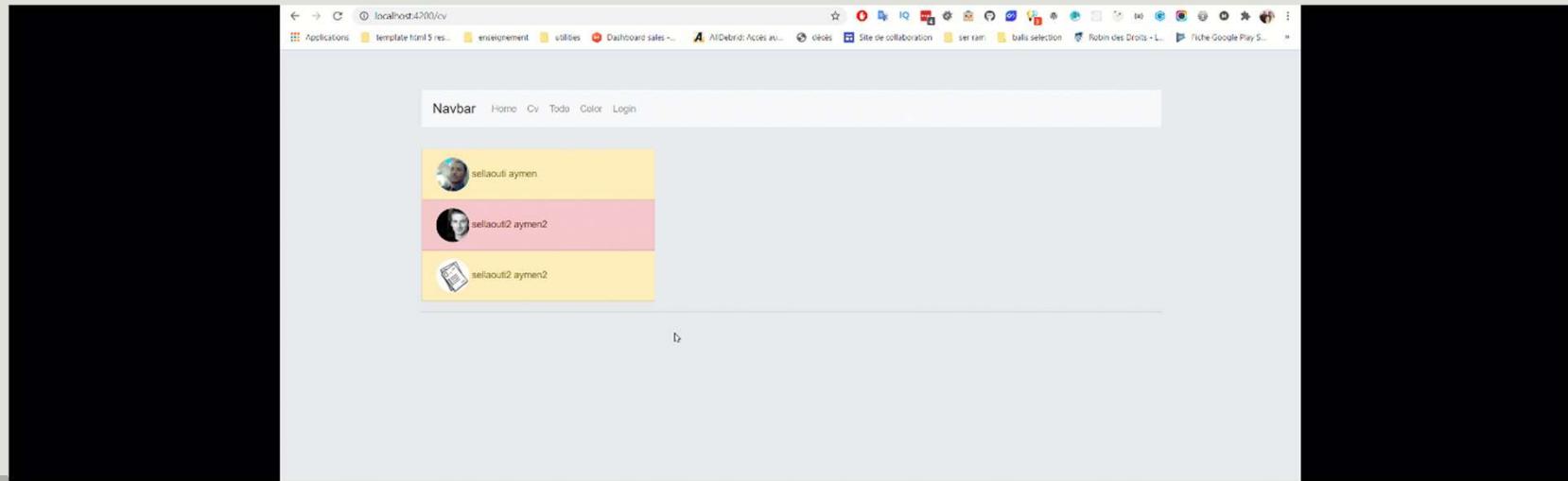
---

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

# Exercice



- Ajouter les fonctionnalités suivantes 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

# 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(formulaire)" #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.

# Approche basée Template NgForm

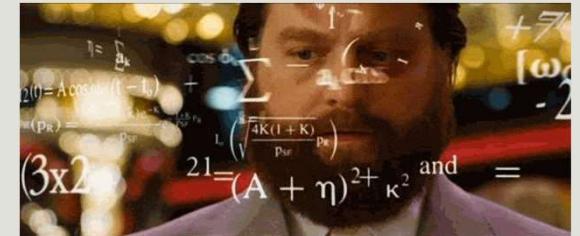
---

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 / invalid** : informe si le formulaire est valide ou non
- **Untouched / touched** : informe si le formulaire est touché ou non
- **pristine** : le formulaire n'a pas été touché, c'est l'opposé du dirty

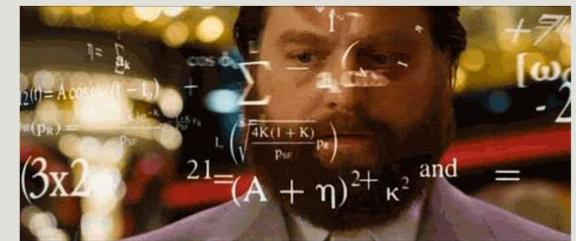
# Exercice

---



- Créer un formulaire d'authentification contenant les champs suivants :
  - Email
  - Password
  - Envoyer
- Si un champ est invalide alors il devra avoir une bordure rouge.
- Les deux champs sont obligatoires
- Le password doit avoir au moins 4 caractères.
- 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*.

# Exercice



localhost:4200/login

Navbar Home Cv Todo Color Login

Email :

password :

Login

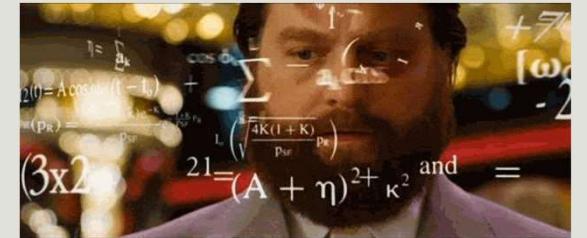
# Approche basée Template

## 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 **ngModel**  
`#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.
- Ajouter un champ d'erreur pour signaler l'erreur à l'utilisateur.

A screenshot of a web browser window showing a login form. The URL bar indicates the page is "localhost:4200/login". The form contains fields for "email" and "password", both with placeholder text. Below the fields is a green "Login" button. The entire form area is flanked by two large black rectangular redaction boxes, one on the left and one on the right, obscuring the content of the browser's address bar and some of the page's header.

# Approche basée Template

## Associer des valeurs par défaut aux champs

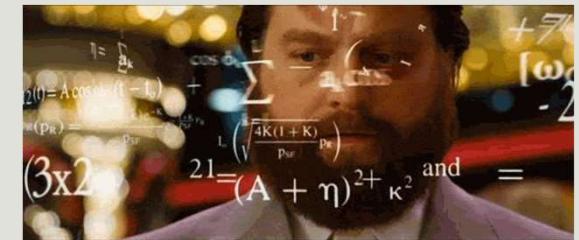
---

- Pour associer des valeurs par défaut aux 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 associée au contrôle d'un élément on ajoute le **property binding** avec **[ngModel]**

# Exercice

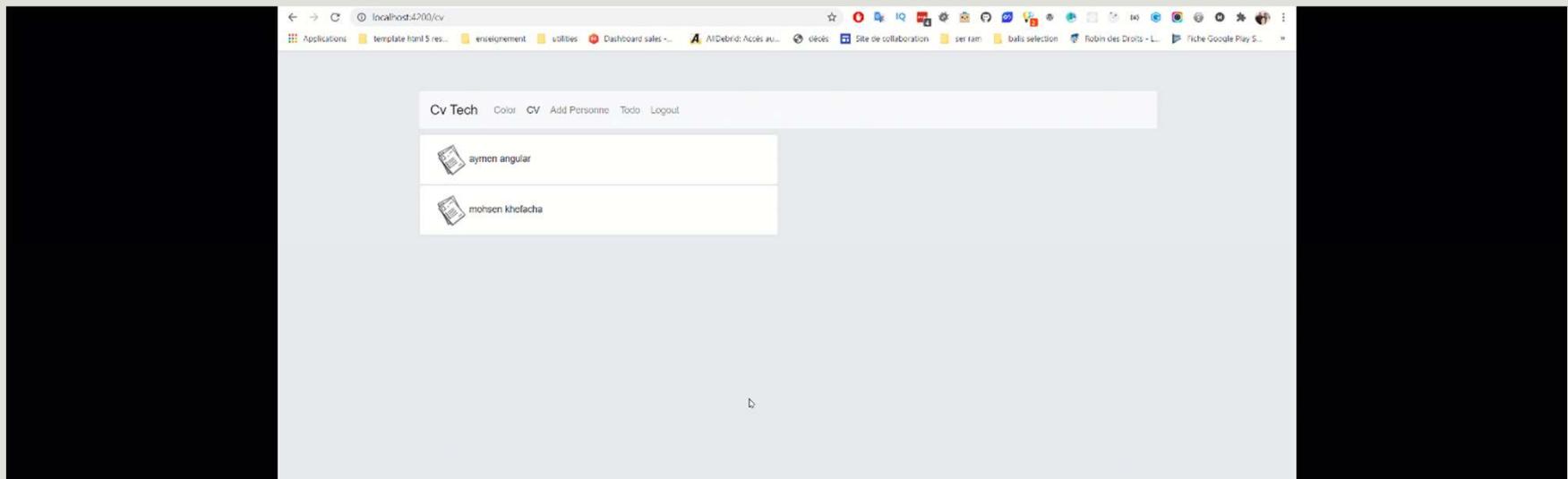
---

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



# Exercice

- Ajouter dans votre cvTech un composant contenant un formulaire. Ce formulaire devra vous permettre d'ajouter un utilisateur.
- Après l'ajout forwarder le user vers la liste des cvs.



# Angular HTTP et Déploiement

---

AYMEN SELLAOUTI

# Objectifs

---

1. Comprendre le design pattern Observateur (Observer) 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 Module HttpClient

# Programmation Asynchrone

---

Programmation non bloquante.

# Les promesses

---

- Ce sont des objets qui représentent une complétion ou l'échec d'une opération asynchrone.

([https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Utiliser\\_les\\_promesses](https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Utiliser_les_promesses))

- Le fonctionnement des promesses est le suivant :
  - On crée une promesse.
  - La promesse va toujours retourner deux résultats :
    - resolve en cas de succès
    - reject en cas d'erreur
  - Vous devrez donc gérer les deux cas afin de créer votre traitement

# Promesse

---

```
var promise2 = new Promise((resolve, reject) => {
 setTimeout(() => {
 resolve(3);
 }, 5000);
});

promise2.then(
 function (x) {
 console.log('resolved with value :', x);
 }
)
```

# 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).

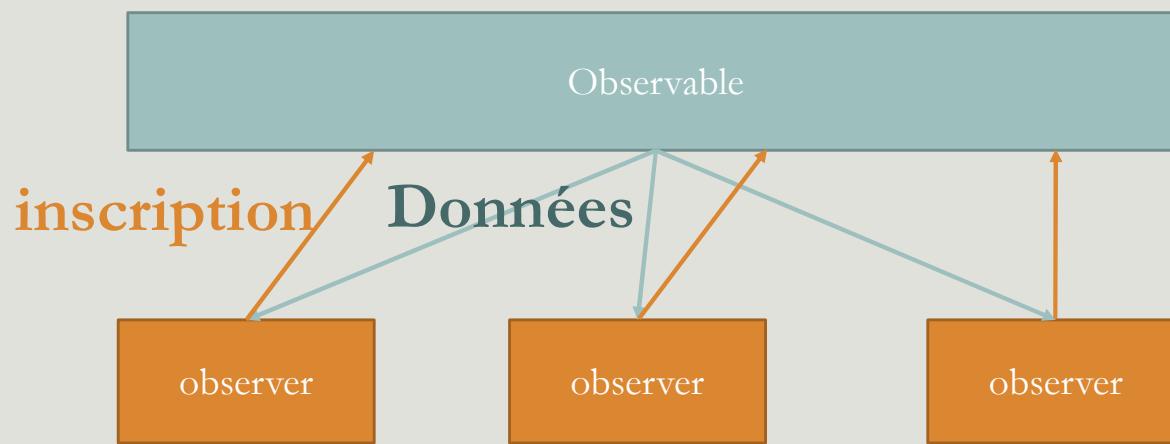
# Le pattern « Observer »

---

- Le patron de conception **Observateur** 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**.

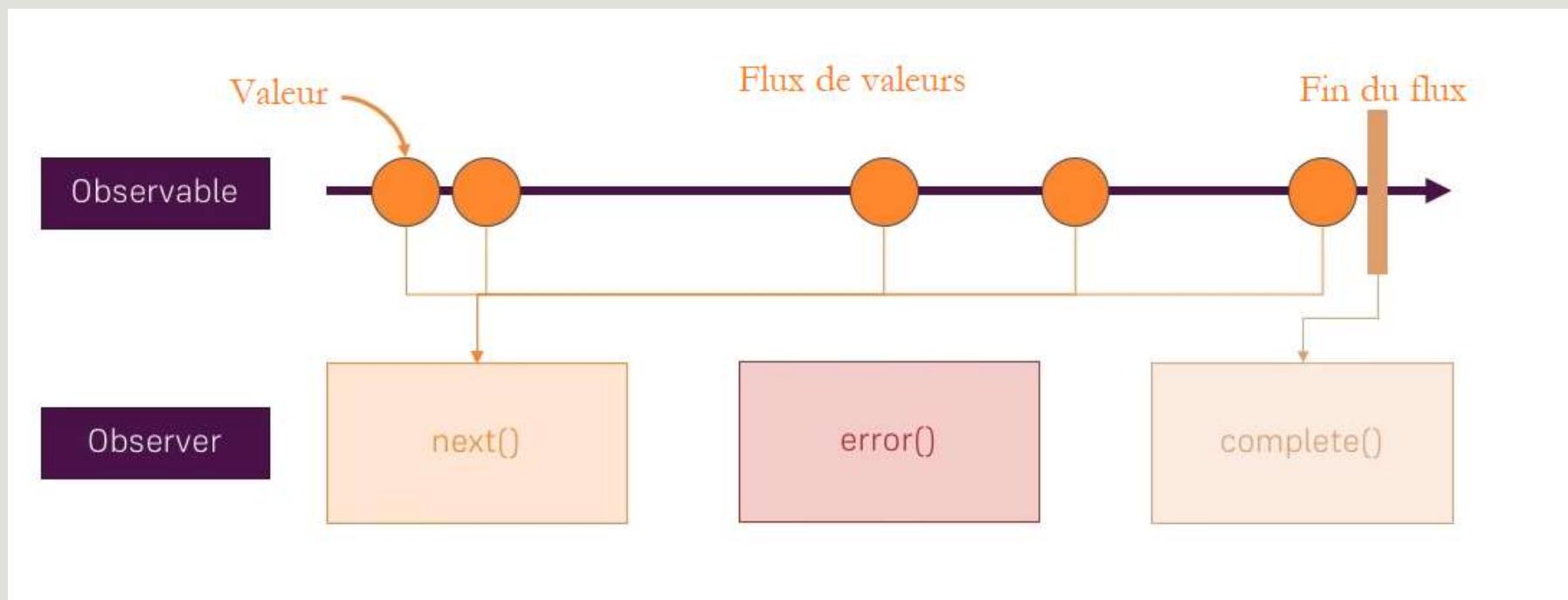
# Observables, Observers et subscriptions

---



**traitement**

# Fonctionnement



# Promesse Vs Observable

Promesse	Observable
Un promesse gère un seul événement	Un observable gère un « flux » d'événements.
Non annulable.	Annulable.
Traitement immédiat.	Lazy : le traitement n'est déclenché qu'à la première utilisation du résultat.
Deux méthodes uniquement (then/catch).	Une centaine d'opérateurs de transformation natifs (map, reduce, merge, filter, ...).
	Opérateurs tels que retry, replay

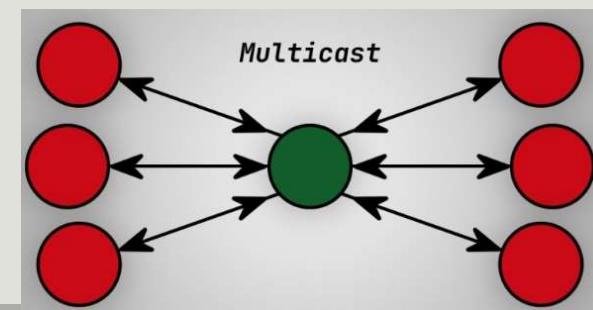
# Observable

---

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

# Hot Vs Cold Observable

- Les **Cold Observables** commencent à émettre des valeurs uniquement quand on s'y inscrit. Les **Hot observables**, par contre émettent toujours.
- Les **Cold Observables** diffusent un flux par inscrit, ils sont **unicast**. Chaque nouvelle inscription crée un **nouveau contexte d'exécution**.
- Les **Hot observables**, sont **multicast**, le **même flux est partagé par tous les inscrits**.
- Dans les **Cold Observables**, la **source de données est à l'intérieur** de l'observable.
- Dans les **HotObservables**, la **source de données est à l'extérieur** de l'observable.



# 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.

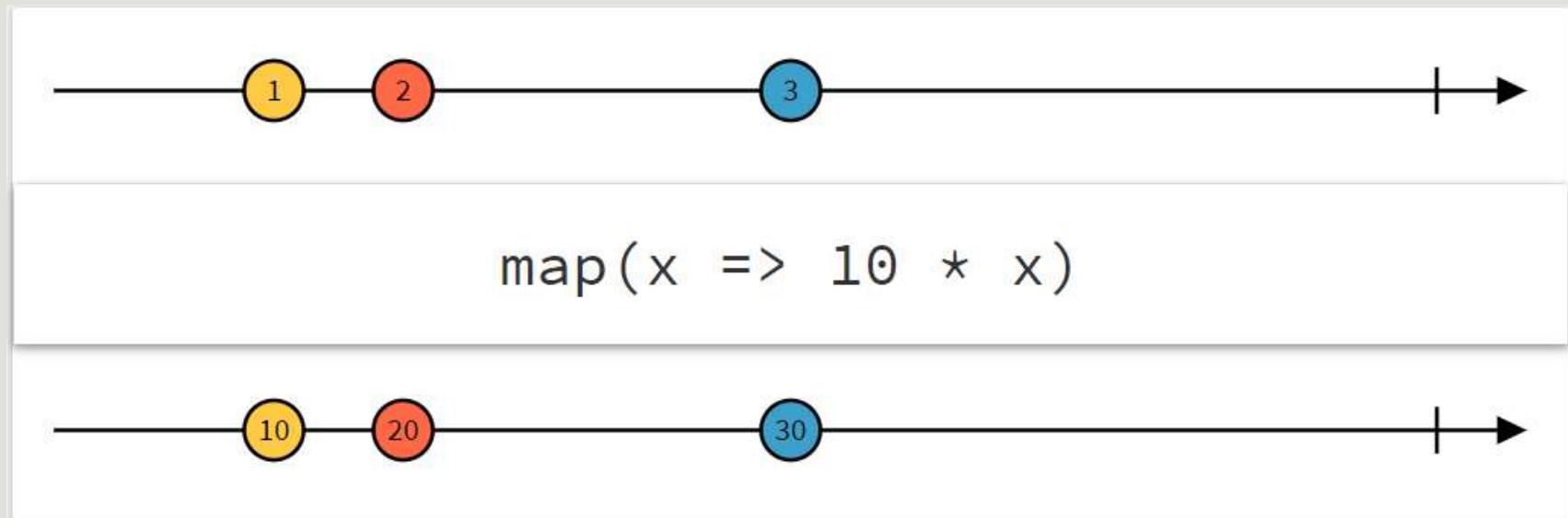
# Les opérateurs de l'observable

---

- 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.

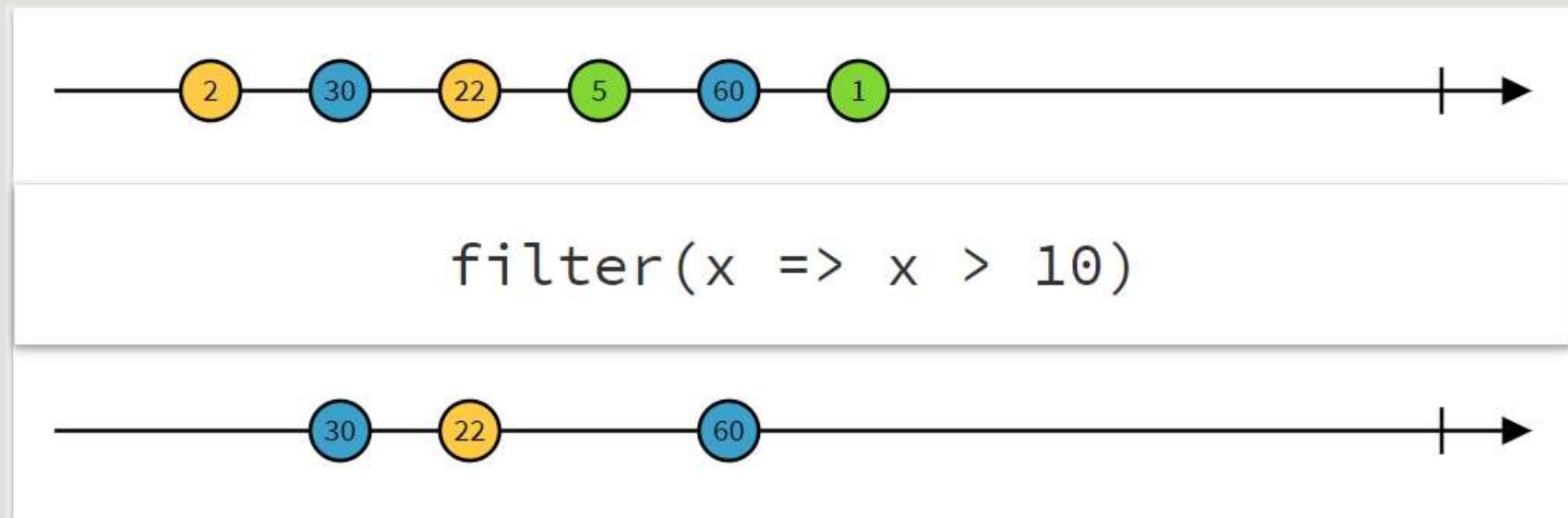
# Quelques opérateurs utiles de l'Observable

## map

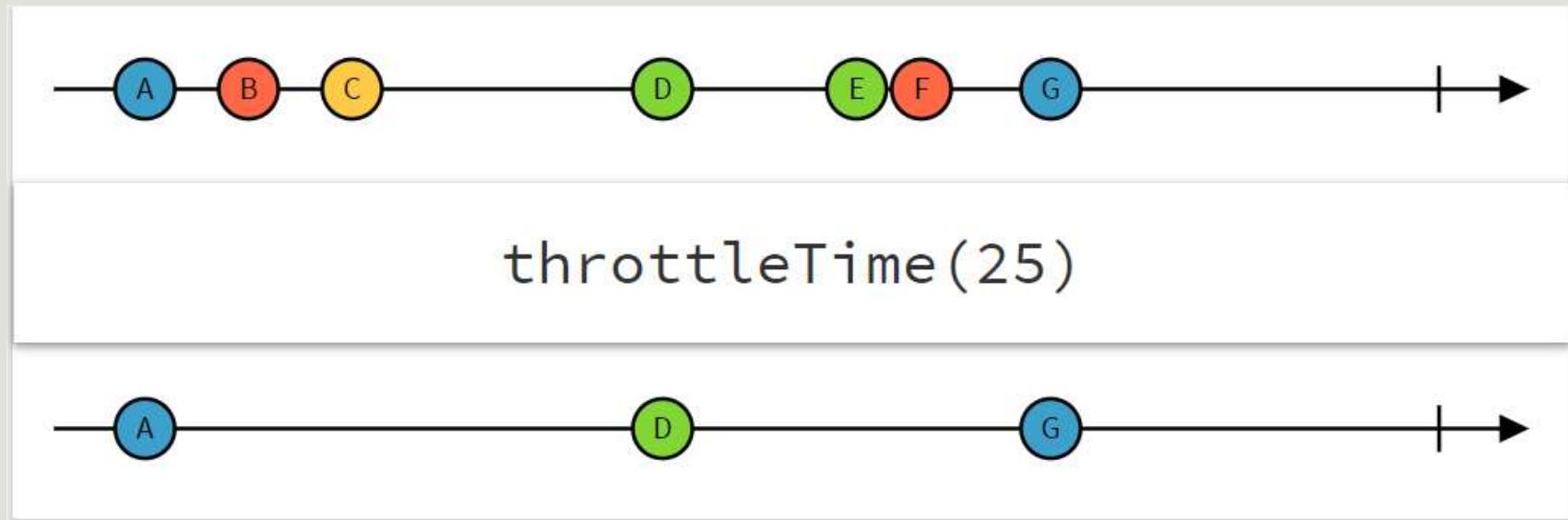


# Quelques opérateurs utiles de l'Observable

## filter



# Quelques opérateurs utiles de l'Observable



# Quelques opérateurs utiles de l'Observable

---

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

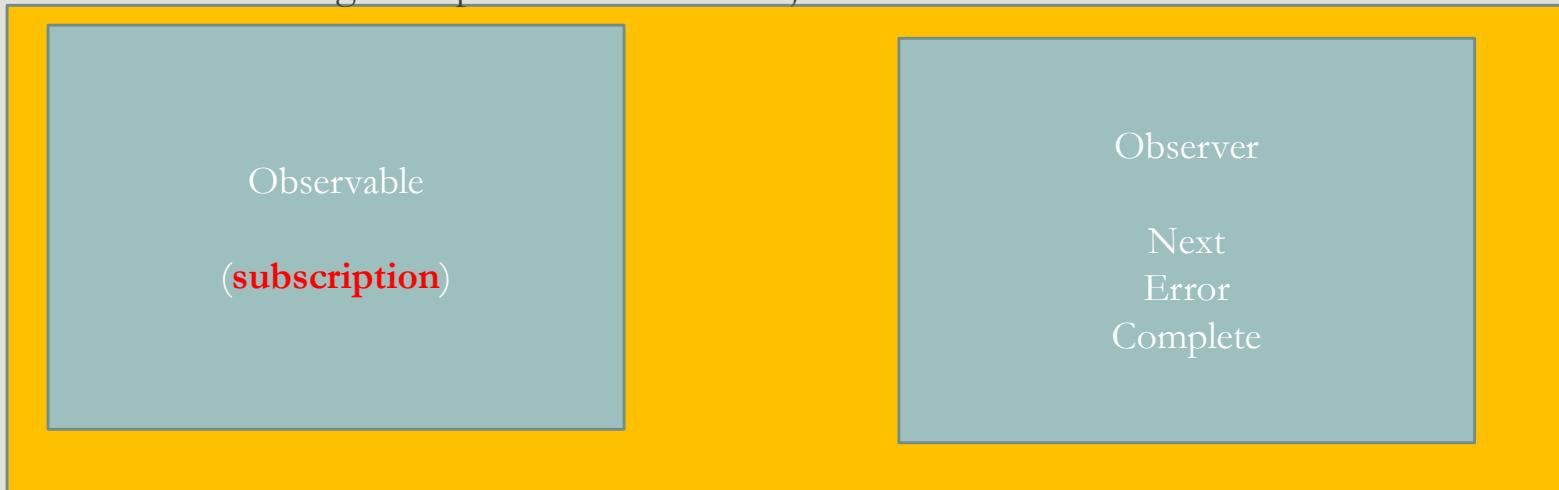
<http://reactivex.io/rxjs/manual/overview.html#operators>

<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.

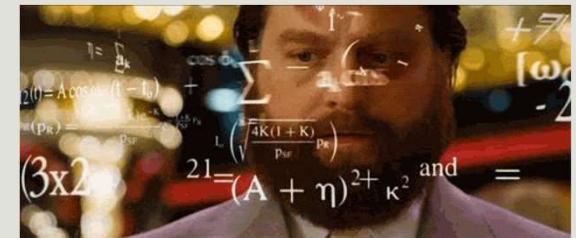


# Les subjects



# 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**.
- Cet 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');
 }
);
```

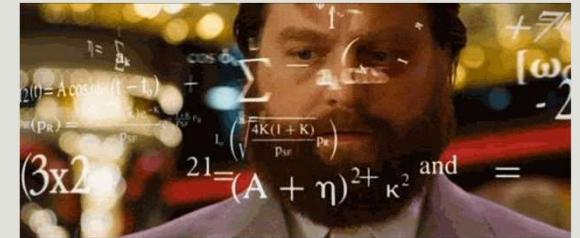
# Documentation

---

<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

---

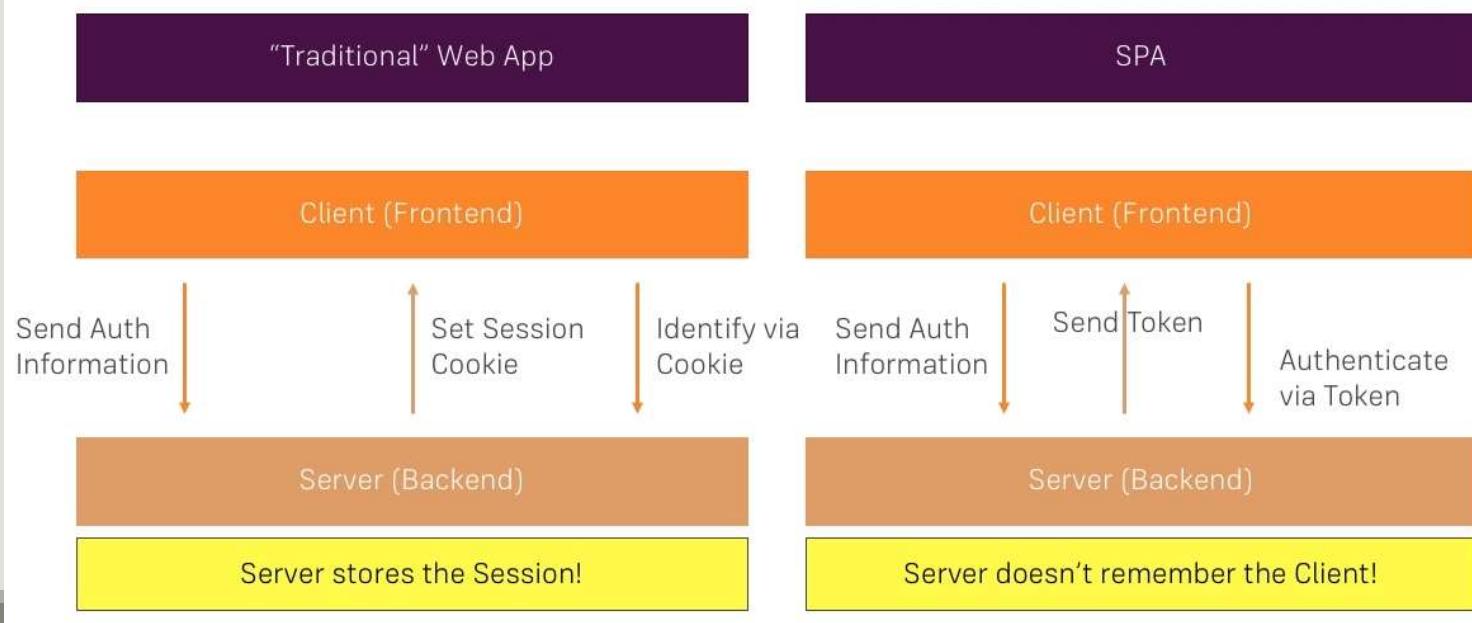
- Afin d'ajouter des paramètres à vos requêtes, le HttpClient vous offre la classe HttpParams.
- Cette classe est une classe immutable (read Only).
- Elle propose une panoplie de méthode helpers permettant de la manipuler.
- `set(clé,valeur)` permet d'ajouter des headers. Elle écrase les anciennes valeurs.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

## How does Authentication work?



# 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 c'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
- Vous pouvez utiliser la méthode : `ng g g nomGuard`

# 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 le futur état du routeur de l'application qui devra passer la validation du guard
 // https://vsavkin.com/routeur-angular-comprendre-1%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

---

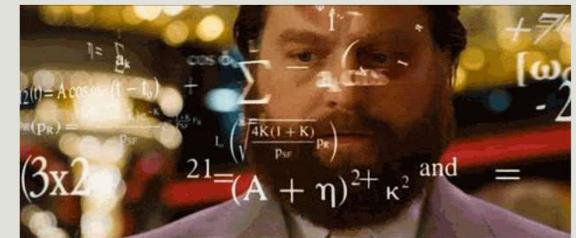
3

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

# Exercice

---

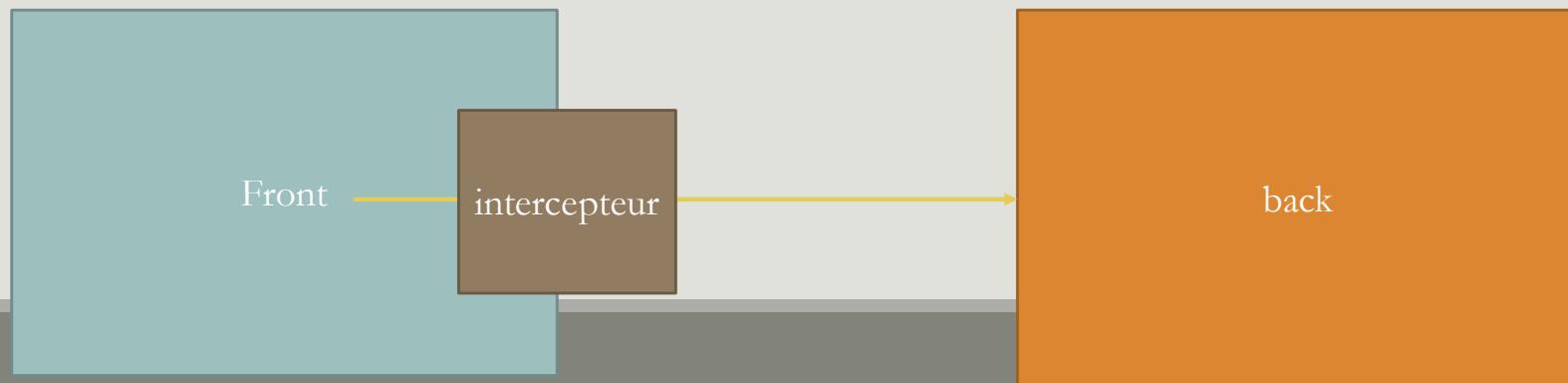
- Ajouter les guards nécessaires afin de sécuriser vos routes.
- Une personne déjà connectée ne peut pas accéder au composant de login.



# 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 là 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

---

```
export class AuthentificationInterceptor implements HttpInterceptor {
 intercept(req: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
 console.log('intercepted', req);
 return next.handle(req);
}
}
```

# Les intercepteurs

---

- Un intercepteur est injecté au niveau du provider. Si vous voulez intercepter toutes les requêtes, vous devez le provider au niveau du module principal.
- L'inscription au niveau du provider se fait de la façon suivante :

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

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

# 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);
```

# Déploiement

---

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

`ng build`

Un dossier dist sera créé 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`

# Angular Internationalisation ngx translate

---

AYMEN SELLAOUTI



# Ngx-Translate

---

- Ngx-Translate est une bibliothèque de traduction pour angular.
- Vous permet de définir la traduction de votre contenu en plusieurs langues et de switcher entre eux facilement.

<https://github.com/ngx-translate>

# Ngx-Translate Installation

---

- Afin d'installer ngx-translate vous devez d'abord installer le core de cette bibliothèque.

**npm install @ngx-translate/core**

- Une fois le core installé, il vous faut un loader qui vous permet de récupérer les fichiers de traductions.

**@ngx-translate/http-loader**

# Ngx-Translate Installation

- Initialiser le module de traduction, **TranslationModule**
- Le **HttpLoaderFactory** est requis pour la compilation AOT de votre projet.

```
TranslateModule.forRoot({
 defaultLanguage: 'fr',
 loader: {
 provide: TranslateLoader,
 useFactory: HttpLoaderFactory,
 deps: [HttpClient],
 },
}),
```

```
// AoT requires an exported function for factories
export function HttpLoaderFactory(http: HttpClient) {
 return new TranslateHttpLoader(http);
}
```

➤ L'instanciation du **TranslateHttpLoader** prend en paramètre le service permettant de récupérer les fichiers de traduction, le chemin vers les fichiers de traduction et leur extension.

➤ Par défaut les valeurs sont :  
`'./assets/i18n/','.json'`

# Ngx-Translate

## Utilisation

---

- Dans le composant qui va se charger de la traduction et du changement de la langue, injecter votre service.

```
constructor(public translateService: TranslateService) {
}
```

- Ce service offre plusieurs fonctions vous permettant de gérer la langue.
  - **setDefaultLang** qui prend en paramètre la langue par défaut.
  - **addLang** qui prend en paramètre un tableau de langue de votre application.
  - **getBrowserLang** qui retourne la liste des langues définies.
  - **use** qui prend en paramètre la langue que vous voulez instaurer dans le site.

# Ngx-Translate

## Utilisation

---

- Ce service offre plusieurs fonctions vous permettant de gérer la langue.
  - **setDefaultLang** qui prend en paramètre la langue par défaut.
  - **addLang** qui prend en paramètre un tableau de langue de votre application.
  - **getBrowserLang** qui retourne la liste des langues définies.
  - **use** qui prend en paramètre la langue que vous voulez instaurer dans le site.

```
constructor(public translateService: TranslateService) {
 translateService.setDefaultLang('fr');
 translateService.addLangs(['en', 'fr']);
 const browserLang = translateService.getBrowserLang();
 translateService.use(browserLang.match(/en|fr/) ? browserLang : 'fr');
}
```

# Ngx-Translate

## Les fichiers de traductions

---

- Les fichiers de traduction sont par défaut sous le dossier ‘assets/i18n/’.
- Ils contiennent simplement un objet JSON de paires clé-valeur, où la clé décrit le texte qui est traduit et la valeur est le texte réel dans la langue spécifiée par le fichier.
- La valeur peut également être un autre objet, ce qui vous permet de regrouper vos traductions comme vous le souhaitez.
- Dans le texte de votre valeur de traduction, vous pouvez également inclure des doubles accolades autour d'un nom de variable, ce qui vous permettra plus tard d'interpoler des chaînes de manière dynamique dans vos traductions.

# Ngx-Translate

## Les fichiers de traductions

```
{
 "APP.TITLE": "Translation Example",
 "HOME": {
 "welcomeMessage": "Thanks for joining, {{ firstName }}! It's great to have you!",
 "login": {
 "username": "Enter your user name",
 "password": "Password here"
 }
 }
}
```

en.js

```
{
 "APP.TITLE": "Exemple de traduction",
 "HOME": {
 "welcomeMessage": "Nous vous remercions pour votre présence, {{ firstName }}! C'est un plaisir de vous avoir avec nous!",
 "login": {
 "username": "Veuillez saisir votre nom d'utilisateur",
 "password": "Votre mot de passe"
 }
 }
}
```

fr.js

- Les parties variables de vos messages doivent être interpolé et passé en paramètres lorsque vous aller utiliser la traduction.

# Utiliser la traduction

---

- Maintenant que tout est en place, il faudra voir comment utiliser cette traduction.
- Afin de spécifier les éléments traductibles, vous avez deux méthodes. Les pipes, et les directives.

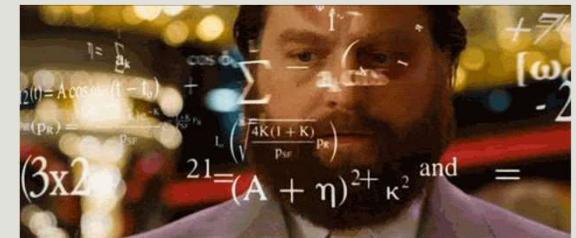
```
<h1 [translate]=""APP.TITLE""></h1>
<h2 translate="">HOME.welcomeMessage</h2>
<input type="text" placeholder="{{ 'HOME.login.username' | translate }}" />
```

- Voici comment passer des paramètres avec les deux méthodes.

```
<h2 translate [translateParams]="{ firstname: 'aymen' }">HOME.welcomeMessage</h2>
<h2 [translate]=""HOME.welcomeMessage"" [translateParams]="{ firstname: 'aymen' }"></h2>
<h2>{{ "HOME.welcomeMessage" | translate: { firstname: 'aymen' } }}</h2>
```

# Exercice

- Testez l'intégration de ngx-translate



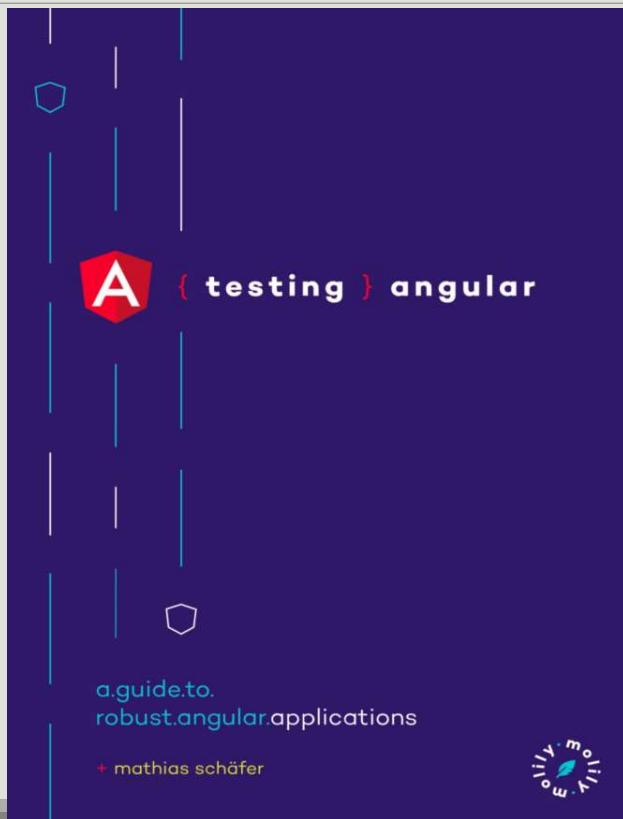
# Angular Tests Unitaires et E2E

---

AYMEN SELLAOUTI



# Références



# Tests unitaires : Introduction

---

- La **plus petite unité de test possible**
- Couvre une **petite fonctionnalité** et ne s'occupe pas de comment les différents unités testées travaillent ensemble.
- Il est **isolé** et ne doit **pas dépendre d'autres tests**.
- Rapide, fiable et pointe directement sur le bug en question.

# Tests unitaires : Pourquoi

---

- Rend votre **code plus robuste**, le bug sera identifié plus tôt
- Vous permet de **formaliser et de documenter** vos besoins
- Un bon test décrit clairement comment le code d'implémentation doit se comporter
- Un bon test doit couvrir **les scénarios les plus importants**
- Les tests rendent le changement sûr en **empêchant les régressions**

# Tests unitaires : Partout ?

---

- Alors devriez-vous écrire des tests automatisés pour tous les cas possibles afin de garantir l'exactitude ?
- Non, disent les principes de l'ISTQB : "**Les tests exhaustifs sont impossibles**".
- Il n'est **ni techniquement faisable ni utile** d'écrire des tests pour toutes les entrées et conditions possibles.
- Au lieu de cela, vous devez **évaluer les risques** d'un certain cas et rédiger d'abord des **tests pour les cas à haut risque**.
- Même s'il était viable de couvrir tous les cas, cela vous donnerait un **faux sentiment de sécurité**.

# Angular et les tests unitaires

---

- Angular avec sa structuration et ses différentes couches se marie parfaitement avec les tests unitaires.
- **Chaque couche ayant un rôle unique** et une tache bien spécifique, les **tests unitaires seront donc propres à chaque partie**, composant, pipe, service, directive, ...
- L'utilisation de **l'injection de dépendance** implique le **couplage faible** et donc des tests isolés.
- Les **providers** qui permettent de fournir des **classes fictives facilitent** aussi cette notion **d'isolation**.

# Jasmin et Karma

---

➤ **Jasmin** est un framework permettant de faciliter la création de tests. Il contient un ensemble de fonctionnalités permettant d'écrire plusieurs types de test.



**Jasmine**



**karma** est un task runner pour vos tests. Il utilise un fichier de configuration afin de gérer le process de test en identifiant les fichiers de chargement, le framework de test, le navigateur à lancer...

# Lancement d'un test

---

- Afin de lancer un test, vous avez juste besoin d'une seule commande et Karma fait le reste. Il exécutera les tests, ouvrira le navigateur, et affichera un rapport sur l'ensemble des tests.

`ng test`

# Concepts de base de jasmin describe (suite)

---

- Pour ce qui est de Jasmine, un test se compose d'une ou plusieurs suites. Une suite est déclarée avec un bloc describe :

```
describe('Suite description', () => {
 /* ... */
});
```

- Chaque suite décrit un morceau de code, le code à tester.

# Concepts de base de jasmin Specification (**it**)

---

- Chaque **describe** se compose d'une ou plusieurs **spécifications**.
- Une **spécification** est déclarée avec un bloc **it** :

```
describe('description de la suite', () => {
 it('description de la spécification', () => {
 /* ... */
 });
});
```

- **it** est une **fonction** qui prend deux paramètres.
- Le premier paramètre est une **chaîne** avec une **description lisible**
- Le second paramètre est une **fonction** contenant **le code de votre test**

# Concepts de base de jasmin Specification (**it**)

---

- Pour écrire le titre de votre test, le **it**..., demandez vous ce que doit faire le code que vous testez.
- Pour une LampeComponent par exemple, il doit allumer et éteindre la lampe, on aura donc :

```
it('switch on the lamp', () => {
 /* ... */
});
it('switch off the lamp', () => {
 /* ... */
})
```

- Après it, un verbe suit généralement, comme switch on, une deuxième famille de testeur préfère suivre le it par **should**

# Concepts de base de jasmin Specification (**it**)

---

- À l'intérieur du bloc **it** se trouve le code de test réel.
- Indépendamment du framework de test, le code de test se compose généralement de trois phases :
  - **Arrange**
  - **Act**
  - **Assert**.
- **Arrange** est la **phase de préparation** et de mise en place. Par exemple, la classe testée est instanciée. Les dépendances sont mises en place. Des espions (spy) et des faux sont créés.
- **Act** est la **phase où l'interaction** avec le code testé. Par exemple, une méthode est appelée ou un élément HTML du DOM est cliqué.
- **Assert** est la **phase où le comportement du code est contrôlé** et vérifié. Par exemple, la sortie réelle est comparée à la sortie attendue.

# Concepts de base de jasmin Specification (**it**)

---

- Imaginons que nous voulons tester un service qui permet d'additionner et de soustraire des entiers.
- Nous commençons par tester l'addition.
  - **Arrange**
    - Nous devons créer une instance du service et de ses dépendances s'ils existent
  - **Act**
    - Appeler la fonction add avec deux paramètres
  - **Assert.**
    - Vérifier que la fonction retourne le bon résultat

# Concepts de base de jasmin

## Attente (Expectation)

---

- Dans la phase **d'affirmation (Assert)**, le test **compare la sortie ou la valeur de retour réelle à la sortie ou à la valeur de retour attendue**. S'ils sont **identiques**, le test **réussit**. S'ils **diffèrent**, le test **échoue**.
- Afin de gérer ça, jasmine nous offre la fonction **expect**.
- Cette **fonction** est **associée** à un ensemble de **matchers** permettant de **faciliter la validation** de vos **attentes ou expectations**.

```
const expectedValue = 5;
const actualValue = MathService.add(2, 3);
expect(actualValue).toBe(expectedValue);
```

# Jasmin matchers

---

- Les **matchers** de Jasmine sont des **fonctions** qui permettent de **tester si une valeur donnée correspond à une condition spécifique**. Ils permettent donc de vérifier que les fonctionnalités de l'application se comportent comme prévu.
- **toBe()** : vérifie si deux valeurs sont strictement égales (utilisant l'opérateur "===")
- **toEqual()** : vérifie si deux objets ont les mêmes propriétés et les mêmes valeurs
- **toMatch()** : vérifie si une chaîne de caractères correspond à une expression régulière
- **toBeDefined()** : vérifie si une variable est définie
- **toBeUndefined()** : vérifie si une variable n'est pas définie
- **toBeNull()** : vérifie si une variable est null

<https://jasmine.github.io/api/edge/matchers.html>

# Jasmin matchers

---

- **toBeTruthy()** : vérifie si une expression est vraie
- **toBeFalsy()** : vérifie si une expression est fausse
- **toContain()** : vérifie si un tableau ou une chaîne de caractères contient un élément spécifié
- **toBeLessThan()** : vérifie si une valeur est inférieure à une autre
- **toBeGreaterThanOrEqual()** : vérifie si une valeur est supérieure à une autre
- ...

# Concepts de base de jasmin

---

- **describe (string, function)** : fonction qui prend en paramètre un titre et une ensemble de test individuel.
- **it (string, function)** : fonction représentant un test individuel qui prend en paramètre un titre et une fonction définissant un test individuel.
- **expect** : fonction qui retourne un booléen et évalue une expectation un besoin à valider par le test unitaire.

Exemple `expect(etatActuel).toBe(etatExpecté)`

- **les matchers** : sont des helpers prédéfinis permettant différentes validations.

# Concepts de base de jasmin

---

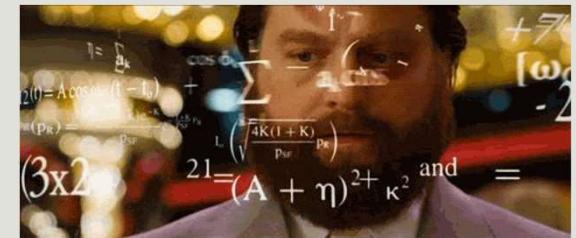
- **x**it permet d'exclure un test individuel
- **x**describe permet d'exclure tout le bloc
- **f**it permet de spécifier le test individuel à exécuter
- **f**describe permet de spécifier le bloc à exécuter.

# Exercice

- Récupérer le Répo suivant :

<https://github.com/aymensellaouti/startingTest>

- Créer les tests nécessaires pour le service MathService



# Concepts de base de jasmin

---

- Lorsque vous écrivez plusieurs spécifications dans une suite, vous réalisez rapidement que **la phase d'arrangement (Arrange) est similaire**, voire identique, dans toutes ces spécifications.
- Par exemple, lors du test du MathService, la phase Arrange consiste toujours à créer une instance de MathService.
- Afin de centraliser ces traitements réplétifs, Jasmine propose quatre fonctions : **beforeEach**, **afterEach**, **beforeAll** et **afterAll**. Ils sont **appelés à l'intérieur d'un bloc describe**.
- Ils attendent un **paramètre**, une **fonction** qui est appelée **aux étapes données**.

# Concepts de base de jasmin

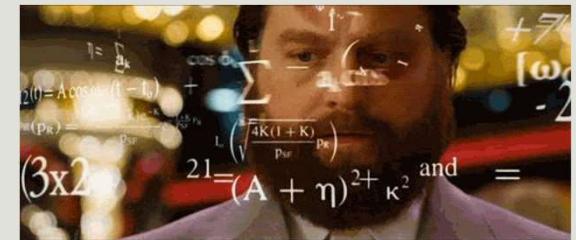
---

Jasmin offre des handlers permettant de répéter certaines fonctionnalités.

- **beforeEach** : prend en paramètre une **callback** et la **répète** avant chaque spec **it**.
- **afterEach** : prend en paramètre une **callback** et la **répète** après chaque spec **it**.
- **beforeAll** : prend en paramètre une **callback** et la **répète** avant chaque suite **describe**.
- **afterEach** : prend en paramètre une **callback** et la **répète** après chaque suite **describe**.

# Exercice

- Mettez à jour vos tests.



# Tests E2E

---

- Ces tests permettent de **SIMULER L'UTILISATION RÉELLE** de votre application.
- Certains tests ont une **vue d'ensemble de haut niveau sur l'application**.
- Ils simulent un **utilisateur interagissant avec l'application** :
  - navigation vers une adresse,
  - lecture de texte,
  - clic sur un lien ou un bouton,
  - remplissage d'un formulaire,
  - déplacement de la souris ou saisie au clavier.

# Tests E2E

---

- Ces tests font des **attentes** sur ce que **l'utilisateur voit**.
- Du point de vue de l'utilisateur, **peu importe que votre application soit implémentée dans Angular**.
- **L'expérience complète est testée => TESTS DE BOUT EN BOUT**
- Les tests de bout en bout constituent également la **partie automatisée des tests d'acceptation** puisqu'ils indiquent si l'application fonctionne pour l'utilisateur.

# Tests E2E

## Comment ça marche ?

- 
- Les tests **E2E** vont donc simuler les interactions de l'utilisateur avec votre application.
  - Vous allez donc lancer le navigateur, et le contrôler afin de simuler un scénario d'interactions.
  - Une fois le **scénario exécuté**, vous allez avoir des **attentes** (expectations), exactement comme avec les tests unitaires :
    - Est-ce que les éléments de la pages sont correct
    - Est-ce que suite au click j'ai le bon affichage
    - ...

# Tests E2E

## Cypress

---

➤ Cypress est un Framework pour les Test E2E dont les avantages sont :

1. Interface utilisateur facile à utiliser
2. Temps de développement rapide
3. Intégration parfaite avec le développement front-end
4. Exécution rapide des tests
5. Capacité à tester directement dans le navigateur
6. Possibilité de déboguer facilement les tests
7. Prise en charge native de la manipulation du DOM et de l'Ajax
8. Documentations et communauté actives
9. Tests fiables et reproductibles.

# Tests E2E

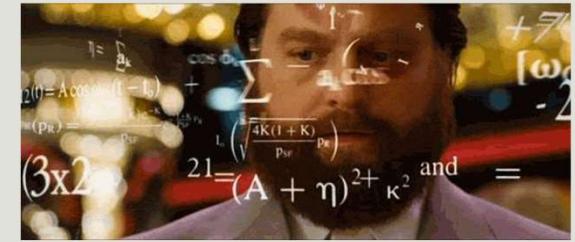
## Cypress

---

- Afin d'installer Cypress utiliser la commande **npm i cypress –save-dev**.
- Avec **angular**, utilisez la commande **ng add @cypress/schematic**
- L'utilisation de cette commande permet d'automatiser la configuration en ajoutant
  - **Cypress** et les packages npm auxiliaires à **package.json**.
  - Le fichier de configuration Cypress **cypress.config.ts**.
  - Modifiez le fichier de configuration **angular.json** afin d'ajouter des commandes d'exécution ng.
  - Créez un **sous-répertoire** nommé **cypress** avec des **templates pour vos tests**.

# Exercice

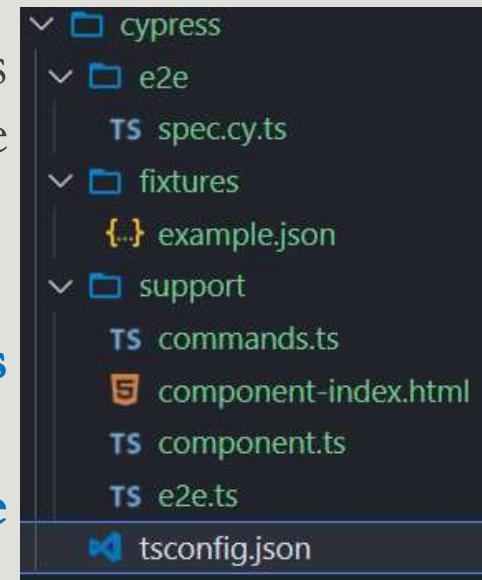
- Installez Cypress et regarder les différents fichiers ajoutés



# Tests E2E

## Le dossier cypress

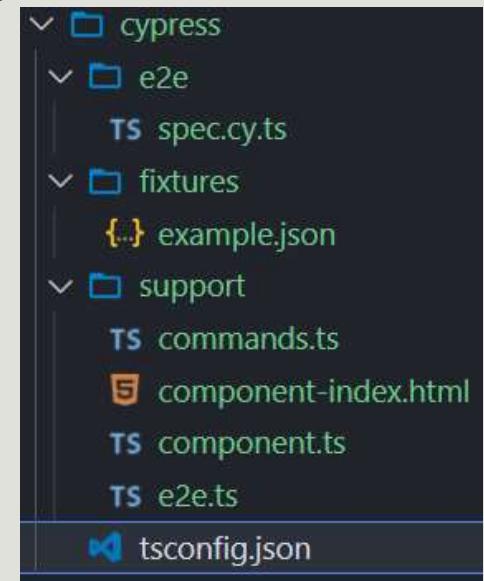
- Le dossier **cypress** généré contient :
  - Une configuration **tsconfig.json** pour tous les fichiers TypeScript spécifiquement dans ce répertoire,
  - Un répertoire **e2e** pour les **tests E2E**,
  - Un répertoire de **support** pour **les commandes personnalisées** et autres assistants de test,
  - un répertoire **fixtures** pour **les données de test**.



# Tests E2E Configuration

- Il y a aussi le fichier cypress.config.ts au niveau de la racine de votre projet.
- Ce fichier vous permet de configurer cypress

```
import { defineConfig } from 'cypress'
export default defineConfig({
 e2e: {
 'baseUrl': 'http://localhost:4200'
 },
 component: {
 devServer: {
 framework: 'angular',
 bundler: 'webpack',
 },
 specPattern: '**/*cy.ts'
 }
})
```



# Tests E2E

## Lancer les tests E2E

---

- Dans package.json on peut identifier deux commandes:
  - **cypress:open** qui exécute la commande **cypress open**
  - **cypress:run** qui exécute la commande **cypress run**

```
"cypress:open": "cypress open",
"cypress:run": "cypress run"
```

# Tests E2E

## Lancer les tests E2E

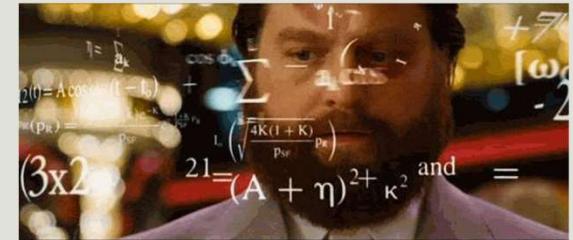
---

- **cypress open** est le **mode interactif**. Elle ouvre une fenêtre dans laquelle vous pouvez sélectionner le navigateur à utiliser et les tests à exécuter. A chaque changement, tout est mis à jour.
- **cypress run**, c'est le mode **non interactif**. Exécute les tests dans un navigateur "headless". Cela signifie que la fenêtre du navigateur n'est pas visible. Les tests sont exécutés une fois, puis le navigateur est fermé et la commande shell se termine.
- Cette commande est généralement utilisée dans un **environnement d'intégration continue**.

# Exercice

---

- Lancez cypress en mode interactif et suivez les étapes



# Tests E2E

## Ecrire des tests E2E

---

- Vous devez lancer votre **serveur dans un terminal** et **cypress dans l'autre**
- Vos **tests** doivent être dans le **dossier e2e**
- Chaque groupement de test, généralement par page sera représenté par un fichier dont **l'extension** est **.cy.ts**.
- En règle générale, un fichier contient un bloc de description **describe**.
- On peut avoir **des blocs de description imbriqués**.
- À l'intérieur, les blocs **beforeEach**, **afterEach**, **beforeAll**, **afterAll** peuvent être utilisés de la même manière que les tests Jasmine.
- À l'intérieur des blocs on peut avoir **un ou plusieurs attentes**.

# Tests E2E

## Visiter une page

- Afin d'accéder à une page vous pouvez utiliser visit
- Si vous avez défini votre baseUrl dans la config comme nous l'avons spécifié (Qui est une bonne pratique : <https://docs.cypress.io/guides/references/best-practices#Setting-a-global-baseUrl>), ajoutez l'URI vers lequel vous voulez naviguer.

```
cy.visit('/') // visits the baseUrl
cy.visit('index.html') // visits the local file "index.html" if baseUrl is null
cy.visit('http://localhost:3000') // specify full URL if baseUrl is null or the domain is different
//the baseUrl
cy.visit({
 url: '/pages/hello.html',
 method: 'GET',
})
```

<https://docs.cypress.io/api/commands/visit>

# Tests E2E

## Sélectionner des éléments

- Certaines méthodes jouent le **double rôle de sélecteur et d'assertions** comme **get** qui vérifie que l'élément existe et qui le sélectionne
- **cy.get()**: Cette méthode permet de sélectionner un élément spécifique en utilisant un sélecteur CSS. **Exemple:** cy.get('#bouton-submit').click()
- **cy.contains()**: Cette méthode permet de sélectionner un élément en fonction du texte qu'ils contient.

**Exemple:** cy.contains('Submit').click()

- **cy.focused()**: Cette méthode permet de sélectionner l'élément qui a le focus actuellement.

**Exemple:** cy.focused().should('have.class', 'form-input-focused')

# Tests E2E

## Sélectionner des éléments

➤ **cy.first()**: Cette méthode permet de sélectionner le premier élément d'une liste d'éléments.

**Exemple:** cy.get('.liste-éléments').first()

➤ **cy.last()**: Cette méthode permet de sélectionner le dernier élément d'une liste d'éléments.

**Exemple:** cy.get('.liste-éléments').last()

➤ **cy.parent()**: Cette méthode permet de sélectionner le parent d'un élément donné.

Exemple: cy.get('.élément-enfant').parent()

# Tests E2E

## Sélectionner des éléments

- 
- **cy.root()**: Cette méthode permet de sélectionner la racine du document HTML. **Exemple:** cy.root().should('have.class', 'racine')
  - **cy.children()**: Cette méthode permet de sélectionner les enfants d'un élément donné. **Exemple:** cy.get('.élément-parent').children().should('have.length', '3')
  - **cy.next()**: Cette méthode permet de sélectionner l'élément suivant d'un élément donné.  
**Exemple:** cy.get('.élément-précédent').next()
  - **cy.prev()**: Cette méthode permet de sélectionner l'élément précédent d'un élément donné. **Exemple:** `cy.get('.élément-suivant').prev()`.

# Tests E2E

## Sélectionner des éléments

### Les bonnes pratiques

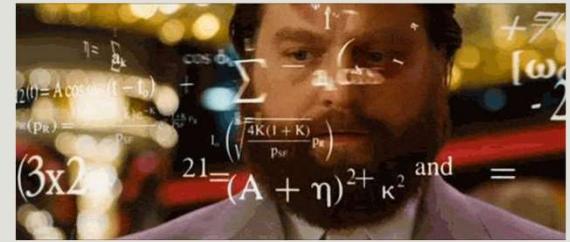
---

- Cypress **déconseille d'utiliser les sélecteurs susceptibles d'être modifiés fréquemment** comme les **classes** ou les **ids** c'est **un Anti-Pattern**.
- **La bonne pratique** est d'utiliser des **attributs** avec ce **pattern data-\*** permettant de donner un contexte à vos sélecteurs et de les **isoler des changements css et js**.
- De plus en utilisant cette technique le **selector Playground de cypress va préférer ces sélecteurs et les mettre en avant** :
  - data-cy
  - data-test
  - data-testid

# Exercice

---

- Dans la page Cv, vérifiez l'existence de la liste des cvs.
- Vérifiez qu'il n'existe pas de cvCard au départ (utiliser l'assertion `.should('not.exist')`).



# Tests E2E

## Test des requêtes HTTP

---

- Lorsque vous testez des apis, vous avez deux stratégies:
  - **Utilisez la réponse du serveur** : la stratégie de requêtes réelles consiste à **utiliser les API externes pour effectuer des tests**. Cela signifie que les tests sont **plus proches de la réalité**, mais **peuvent être plus lents et plus instables** en raison de la dépendance aux API. Cependant, cette approche **garantit une meilleure couverture des cas d'utilisation et une meilleure qualité de test** en général.
  - **Utilisez des fixtures** : La **stratégie de requêtes mockées** consiste à **remplacer les réponses API réelles par des réponses prédéfinies et contrôlées par le développeur**. Cela signifie que les tests **ne dépendent pas de la disponibilité ou de la rapidité des API**, ce qui peut accélérer les tests et les rendre plus fiables. Cependant, cette approche n'est **pas toujours réaliste et peut ne pas couvrir tous les cas d'utilisation possibles**.

# Tests E2E

## Test des requêtes HTTP

### API Réel

---

#### ➤ **Avantages**

- Plus susceptible de travailler en production
- Tester la couverture de vos endpoints
- Idéal pour le rendu HTML traditionnel côté serveur (en cas de réponse HTML et non JSON)

#### ➤ **Inconvénients**

- Nécessite de seeder des données (Base de données de test à préparer pour les différents cas)
- Beaucoup plus lent
- Plus difficile à tester les cas extrêmes

#### ➤ **Utilisation suggérée**

- Utiliser avec parcimonie
- Idéal pour les chemins critiques de votre application

# Tests E2E

## Test des requêtes HTTP

### API Mockés

---

#### ➤ Avantages

- Contrôle des corps de réponse, de l'état et des en-têtes
- Peut forcer les réponses à prendre plus de temps pour simuler le retard du réseau
- Temps de réponse rapides, < 20 ms

#### ➤ Inconvénients

- Aucune garantie que vos réponses tronquées correspondent aux données réelles envoyées par le serveur
- Aucun test couverture sur certains points de terminaison de serveur
- Pas aussi utile si vous utilisez le rendu HTML traditionnel côté serveur

#### ➤ Utilisation suggérée

- Utilisez pour la grande majorité des tests
- Mélangez et faites correspondre, ayez généralement un vrai test de bout en bout, puis remplacez le reste
- Parfait pour JSON APIs

<https://docs.cypress.io/guides/guides/network-requests>

# Tests E2E

## Test des requêtes HTTP API Mockés

---

- Cypress vous permet de **remplacer une réponse** et de **contrôler le corps, l'état, les en-têtes ou même le délai.**
- **cy.intercept()** est utilisé pour contrôler le comportement des requêtes HTTP. Vous pouvez **définir de manière statique le corps**, le **status** HTTP, les **en-têtes** et d'autres caractéristiques de réponse.
- Elle peut **prend en paramètre un grand nombre de combinaison selon votre cas d'utilisation.**

# Tests E2E

## Test des requêtes HTTP

### API Mockés

```
// spying
cy.intercept('/users/**')
cy.intercept('GET', '/users*')
cy.intercept({
 method: 'GET',
 url: '/users*',
 hostname: 'localhost',
})
// spying and response stubbing
cy.intercept('POST', '/users*', {
 statusCode: 201,
 body: {
 name: 'Peter Pan',
 },
})
// spying, dynamic stubbing, request modification, etc.
cy.intercept('/users*', { hostname: 'localhost' }), (req) => {
/* do something with request and/or response */
})
```

<https://docs.cypress.io/api/commands/intercept>

# Tests E2E

## Test des requêtes HTTP

### API Mockés / intercept, mockez une réponse

---

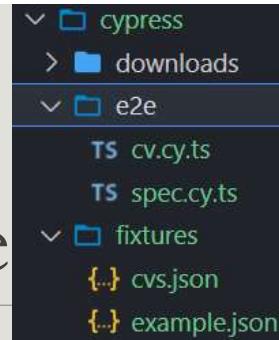
➤ Lorsque vous utilisez intercept suivez les étapes suivantes:

1. Préparer l'interception
2. Lancer l'opération souhaité
3. Lancez vos Assertions

# Tests E2E

## Test des requêtes HTTP

### API Mockés / intercept, mockez une réponse



- Vous pouvez moquer la réponse de votre api avec des fixtures.
- Les fixtures peuvent êtres de plusieurs types et vous avez le dossier fixtures pour les stocker.

```
// requests to '/update' will be fulfilled
// with a body of "success"
cy.intercept('/update', 'success')
// requests to '/users.json' will be fulfilled
// with the contents of the "users.json" fixture
cy.intercept('/users.json', { fixture: 'users.json' })
cy.intercept('/projects', {
 body: [{ projectId: '1' }, { projectId: '2' }],
})
```

```
cy.intercept('/not-found', {
 statusCode: 404,
 body: '404 Not Found!',
 headers: {
 'x-not-found': 'true',
 },
})
```

```
cy.intercept(
{
 method: 'GET',
 url: API.cv,
},
{
 fixture: 'cvs',
}
)
```

# Tests E2E

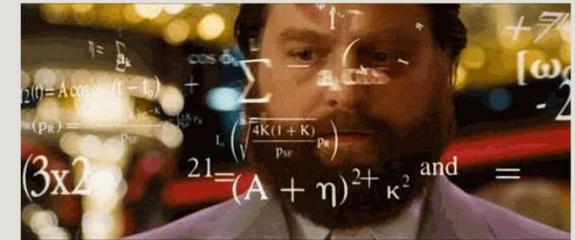
## Test des requêtes HTTP

### API Mockés / intercept, affecter un alias

- Afin de pouvoir manipuler **l'intercept**, comme par exemple l'attendre avec un **wait**, vous pouvez lui affecter un **alias**.

```
cy.intercept('http://example.com/settings').as('getSettings')
cy.wait('@getSettings')
cy.intercept({
 url: 'http://example.com/search*',
 query: { q: 'expected terms' },
}).as('search')
cy.wait('@search')
```

# Exercice



- Faite en sorte d'avoir des fixtures pour la liste des cvs permettant de tester cette liste.
- Vérifier que l'affichage utilise vos fixtures
- Ajouter des fixture pour la sélection d'un cv par son id.

# Tests E2E

## Les assertions

---

- Cypress intègre plusieurs assertions de diverses bibliothèques d'assertions JS telles que Chai, jQuery, etc.
- Nous pouvons globalement classer toutes ces assertions en deux segments en fonction du sujet sur lequel nous pouvons les invoquer :
  - Les assertions implicites
  - Les assertions explicites

# Tests E2E

## Les assertions implicites

- Lorsque **l'assertion s'applique à l'objet fourni par la commande chaînée parente**, elle s'appelle une assertion **implicite**.
- Cette catégorie d'assertions inclut généralement des commandes telles que **".should()" et ".and()**.
- Comme ces commandes **ne sont pas indépendantes** et dépendent toujours de la commande parente précédemment chaînée, elles **héritent et agissent automatiquement sur l'objet généré par la commande précédente**.
- Généralement, nous utilisons des assertions implicites lorsque nous voulons :
  - Affirmer plusieurs validations sur le même sujet.
  - Changez de sujet avant de faire des affirmations sur le sujet.

# Tests E2E

## Les assertions

```
cy.get('.assertion-table')
 .find('tbody tr:last')
 .should('have.class', 'success')
 .find('td')
 .first()
 // valider le contenu d'un élément
 .should('have.text', 'Column content')
 .should('contain', 'Column content')
 .should('have.html', 'Column content')
 .should('match', 'td')
```

```
<table class="table table-bordered assertion-table">
 <thead>
 <tr><th>#</th><th>Column heading</th><th>Column heading</th></tr>
 </thead>
 <tbody>
 <tr><th scope="row">1</th><td>Column content</td><td>Column content</td></tr>
 <tr><th scope="row">2</th><td>Column content</td><td>Column content</td></tr>
 <tr class="success"><th scope="row">3</th><td>Column content</td><td>Column content</td></tr>
 </tbody>
</table>
```

#	Column heading	Column heading
1	Column content	Column content
2	Column content	Column content
3	Column content	Column content

// Pour vérifier qu'un texte valide une expression régulière,  
// préférer l'utilisation de contains

```
cy.get('.assertion-table')
 .find('tbody tr:last')
 // finds first element with text content matching regular
 // expression
 .contains('td', /column content/i)
 .should('be.visible')
```

<https://docs.cypress.io/api/commands/should>

# Tests E2E

## Les assertions

### have

---

- **exist** : pour vérifier l'existence d'un élément
- **be.visible** : pour vérifier la visibilité d'un élément
- **be.enabled** : pour vérifier l'état activé/désactivé d'un élément
- **be.checked** : pour vérifier l'état coché/décoché d'un élément
- **have.value** : pour vérifier la valeur d'un élément
- **have.text** : pour vérifier le texte d'un élément
- **have.css** : pour vérifier la présence d'un attribut de style sur un élément
- **have.class** : pour vérifier la présence d'une classe sur un élément

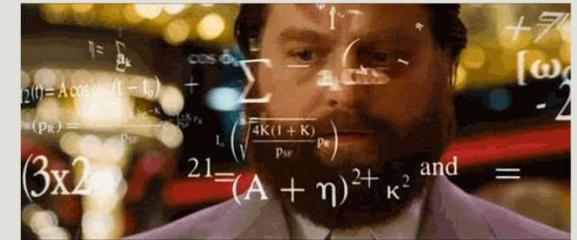
<https://docs.cypress.io/api/commands/should>

# Tests E2E

## Les assertions

- 
- **have.length** : pour vérifier la longueur d'un objet.
  - **be.true / be.false** : pour vérifier si une expression est vraie ou fausse
  - **eq / equal / eql** : pour vérifier l'égalité de deux valeurs
  - **contain** : pour vérifier la présence d'une valeur dans un tableau ou une chaîne de caractères
  - **match** : pour vérifier si une chaîne de caractères correspond à une expression régulière
  - **be.greaterThan / be.lessThan** : pour vérifier si une valeur est supérieure ou inférieure à une autre valeur.

# Exercice



- Dans la page des cvs vous avez deux onglets, un pour les seniors et un pour les juniors.
- Vérifiez que vous avez les deux onglets
- Vérifiez que le premiers élément correspond pour les deux listes
- Vérifiez que La taille des deux listes est correcte.
- Vérifiez que le premier onglet et visible et que le second ne l'est pas

# Tests E2E

## Location

- Afin d'avoir des information sur la localisation actuelle, donc l'url actif, vous pouvez utiliser la commande location.
- Avec l'assertion should, vous pouvez lui passer une callback qui prend en paramètre la location et appelle les expectations que vous voulez valider.

```
cy.location().should((location) => {
 expect(location.pathname).to.equal('/cv/1');
});
```

```
Location : ▾ Object {
 auth: "",
 authObj: undefined,
 hash: "",
 host: "localhost:4200",
 hostname: "localhost",
 href: "http://localhost:4200/cv/1",
 origin: "http://localhost:4200",
 pathname: "/cv/1",
 port: "4200",
 protocol: "http:",
 search: "",
 superDomain: "localhost",
 superDomainOrigin: "http://localhost:4200",
 ▶ toString: f wrapper()
 ▶ [[Prototype]]: Object
```

# Tests E2E

## Déclencher des actions

---

- Cypress vous permet de simuler des fonctions.
- Pour **écrire dans un élément DOM**, utilisez la commande **.type()**.
- Vous pouvez effacer le champ avant de taper avec **clear()**

```
it('Visits the initial project page', () => {
 cy.visit('/');
 cy.contains('Faurecia');
 cy.get('[data-cy=email-input]')
 .type('aymen@email.com')
 .should('have.value', 'aymen@email.com')
});
```

# Tests E2E

## Déclencher des actions

- Pour avoir le **focus sur un élément du DOM**, utilisez la commande **focus()**
- Pour **perdre le focus sur un élément du DOM**, utilisez la commande **blur()**
- Pour **soumettre un formulaire**, utilisez la commande **cy.submit()**
- Pour **cliquer** sur un **élément du DOM**, utilisez la **commande click()**. Si l'élément **n'est pas visible** ou **n'est pas enabled** ajouter en paramètre {force:true} `.click({ force: true });`
- Pour **cocher une case ou une radio**, utilisez la commande **check()**.
- Pour **sélectionner une option dans un select**, utilisez la commande **select()**.

# Tests E2E

## Déclencher des actions

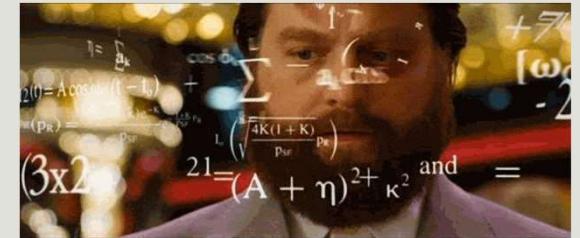
- Afin de simuler le click sur un caractère spécial, entre, insert, delete, utilisez la syntaxe suivante:

```
// Special characters:
cy.get('input').type('{enter}')
cy.get('input').type('{backspace}')
cy.get('input').type('{del}')
cy.get('input').type('{esc}')
cy.get('input').type('{end}')
cy.get('input').type('{home}')
cy.get('input').type('{insert}')
cy.get('input').type('{moveToEnd}') // Move cursor to the end of
// typeable element
cy.get('input').type('{moveToStart}') // Move cursor to the start of
// typeable element
cy.get('input').type('{pageDown}') // Scroll down
cy.get('input').type('{pageUp}') // Scroll up
cy.get('input').type('{selectAll}') // Select the entire input value
```

```
// Arrows:
cy.get('input').type('{upArrow}')
cy.get('input').type('{downArrow}')
cy.get('input').type('{leftArrow}')
cy.get('input').type('{rightArrow}')
```

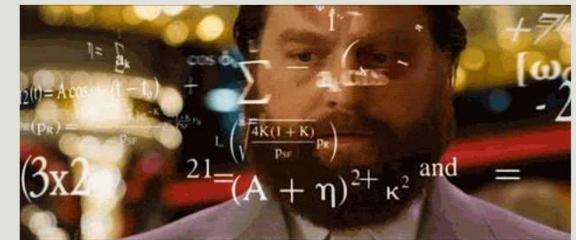
```
// Modifier keys:
cy.get('input').type('{shift}')
cy.get('input').type('{ctrl}')
cy.get('input').type('{alt}')
```

# Exercice



- Afin de migrer vers la version 18 à partir de la version 16, j'ai effectué deux migrations :  
16>17 puis 17>18 en utilisant la commande :
- `ng update @angular/core@17 @angular/cli@17` puis
- `ng update @angular/core@18 @angular/cli@18`

# Exercice



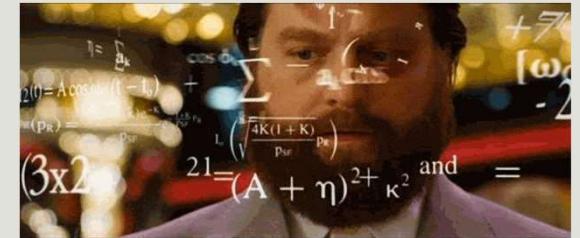
Angular offre un moyen **de migrer automatiquement une partie** de votre application Angular modulaire (15.2.0) et plus vers une application standalone.

- Vous pouvez suivre les différentes étapes du Guide qui présente des parties automatiques et d'autres manuelles.
- Exécutez la migration **dans l'ordre indiqué** ci-dessous, **en vérifiant que votre code est généré et exécuté entre chaque étape** :
  1. Exécutez `ng g @angular/core:standalone` et sélectionnez "Convert all components, directives and pipes to standalone"
  2. Exécutez `ng g @angular/core:standalone` et sélectionnez "Remove unnecessary NgModule classes", **cette étape risque de garder plusieurs modules.**
  3. Exécutez `ng g @angular/core:standalone` et sélectionnez "Bootstrap the project using standalone APIs".

<https://angular.io/guide/standalone-migration>

# Exercice

---



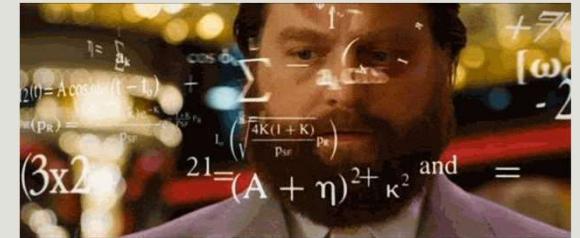
- Si vous voulez migrer votre ancien code et utilisez le nouveau work flow, angular vous fournit une commande qui le fait pour vous :
  - **ng g @angular/core:control-flow**
- Cette fonctionnalité est encore en developer Preview pour Angular 17.2

<https://angular.dev/reference/migrations/control-flow>

# Exercice

---

- Afin d'utiliser la nouvelle fonction inject, vous pouvez utiliser la commande : **ng generate @angular/core:inject**



<https://angular.dev/reference/migrations/inject-function>

# Les Signaux



---

AYMEN SELLAOUTI





# Qu'est ce qu'un Signal ?

- Un signal agit comme une **enveloppe (wrapper)** autour d'une valeur, ayant la capacité **d'avertir les consommateurs lorsque la valeur change**.
- Un signal est une **primitive réactive** qui représente une valeur et qui nous permet de
  - **suivre ses changements** au fil du temps.
  - **modifier** cette même valeur en **notifiant tous ceux qui en dépendent**.
- Elle permet de définir l'état réactif de votre application et d'avoir une **identification précise de quels composants sont impactés par un changement**.



# Qu'est ce qu'un Signal ?

- Les signaux sont un concept utilisé dans plusieurs frameworks (Qwik, SolidJs, Vue, KnockoutJs)
- Le concept du **Signal** dans Angular est une fonctionnalité introduite en '**Developer Preview**' dans la version **16** de la bibliothèque `@angular/core` et **stable à partir de Angular 17**.
- Il a pour objectif **de simplifier le développement** en donnant une **alternative plus simple que RxJS** pour gérer **certaines cas de réactivité** d'un façon plus simple.

# Pourquoi intégrer les signaux



## Reactivity Everywhere

Les signaux permettent de réagir aux changements d'état (State) n'importe où dans notre code et pas seulement au sein d'un composant.



## Precision Updates

Les signaux boostent les performances de votre application en réduisant le travail que fait Angular pour garder le DOM à jour avec les valeurs des données



## Lightweight Dependencies

Les signaux pèsent 2KB, n'ont pas besoin de charger des dépendances tierces et ne représentent aucun coût de démarrage lorsque votre application se charge.

# API

---

- ▶ Angular propose trois principales primitives pour utiliser les signaux :
  - ▶ signal
  - ▶ computed
  - ▶ effect

# Créer un signal via l'api signal()

---

- La fonction **signal** permet d'initialiser une variable et d'informer Angular et son contexte à chaque fois que sa valeur change.
- Elle retourne un objet de type **WritableSignal**.
- **Un signal doit avoir une valeur initiale.**

```
@Component({
 selector: 'app-signal-api',
 standalone: true,
 imports: [],
 styleUrls: ['./signal-api.component.css'],
 template: ` <h1>Hello World</h1> `,
})
export class SignalApiComponent {
 lastname: WritableSignal<string> = signal('sellouti');
}
```

# Récupérer la valeur d'un signal

---

- ▶ Afin de **récupérer la valeur d'un signal** il suffit de l'appeler comme une fonction.

```
@Component({
 selector: 'app-signal-api',
 standalone: true,
 imports: [],
 styleUrls: './signal-api.component.css',
 template: ` <h1>Hello {{ lastname() }}</h1> `,
})
export class SignalApiComponent {
 lastname: WritableSignal<string> = signal('selliaouti');
}
```

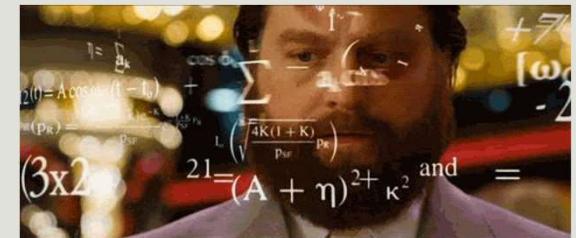
# Les méthodes de modifications de signal : set() et update()

- Pour modifier la valeur d'un **signal**, on peut passer par :
- La Méthode **set**, qui permet **d'affecter une nouvelle valeur au signal**.
- La méthode **update**, qui permet de **calculer une nouvelle valeur** d'un signal **en fonction de sa valeur précédente**.

```
@Component({
 selector: 'app-signal-api',
 standalone: true,
 imports: [],
 template:
 `<h1>Hello {{ lastname() }}</h1>
 <input type="number" #input
 (change)="setCounter(+input.value)"
 />
 <h2 (click)="increment()">
 Click here. You clicked {{ counter() }} times
 </h2>
 `,
})
export class SignalApiComponent {
 lastname = signal('aymen');
 counter = signal(0);
 increment() {
 this.counter.update((currentValue) => currentValue + 1);
 }
 setCounter(val: number) {
 this.counter.set(val);
 }
}
```

# Exercice

- Reprenez L'exercice ‘ColorComponent’ en utilisant les signaux.



# computed()

---

- ▶ L'api **computed()** permet de créer un **nouveau signal** dont la valeur **dépend d'autres signaux**.
- ▶ Lorsqu'un **signal est mis à jour, tous ses signaux dépendants seront alors automatiquement mis à jour**.
- ▶ On note que **computed()** retourne un **objet de type Signal** et non **WritableSignal**.

```
lastname = signal('aymen');
firstname = signal('sellaouti');
fullname = computed(() => `${this.firstname()} ${this.lastname()}`)
```

# computed()

- ▶ Pour identifier un signal qui a changé, et donc si on doit exécuter un computed, Angular utilise **Object.is**.
- ▶ `Object.is()` permet de déterminer si deux valeurs sont identiques. Deux valeurs sont considérées identiques si :
  - ▶ elles sont toutes les deux `undefined`
  - ▶ elles sont toutes les deux `null`
  - ▶ elles sont toutes les deux `true` ou toutes les deux `false`
  - ▶ elles sont des chaînes de caractères de la même longueur et avec les mêmes caractères (dans le même ordre)
  - ▶ elles sont toutes les deux le même objet (même référence)
  - ▶ elles sont des nombres et
    - ▶ sont toutes les deux égales à `+0`
    - ▶ sont toutes les deux égales à `-0`
    - ▶ sont toutes les deux égales à `NaN`

# computed()

## Comment ça marche ?

- ▶ L'arbre de dépendance est créé dynamiquement à chaque appel du computed.
- ▶ Il faut donc faire très attention dans la définition de vos computed lorsqu'il y a des traitements conditionnels.

```
@Component({
 template: `
 <h3>Counter value {{ counter() }}</h3>
 <h3>Derived counter: {{ derivedCounter() }}</h3>
 <button (click)="increment()">Increment</button>
 <button (click)="multiplier = 10">Set multiplier to 10</button>
 `,
})
export class ComputedProblemComponent {
 counter = signal(0);
 multiplier: number = 0;
 derivedCounter = computed(() => {
 if (this.multiplier < 10) {
 return 0;
 } else {
 return this.counter() * this.multiplier;
 }
 });
 increment() {
 console.log(`Updating counter...`);
 this.counter.set(this.counter() + 1);
 }
}
```



## computed()

### Exclure un signal de l'arbre de dépendance

---

- ▶ Si pour une raison ou une autre vous voulez lire une valeur d'un signal dans un computed mais sans l'intégrer dans l'arbre de dépendance, vous pouvez utiliser l'Api **untracked**.
- ▶ Dans cet exemple le computed fullname ne sera mis à jour que si le signal firstname change

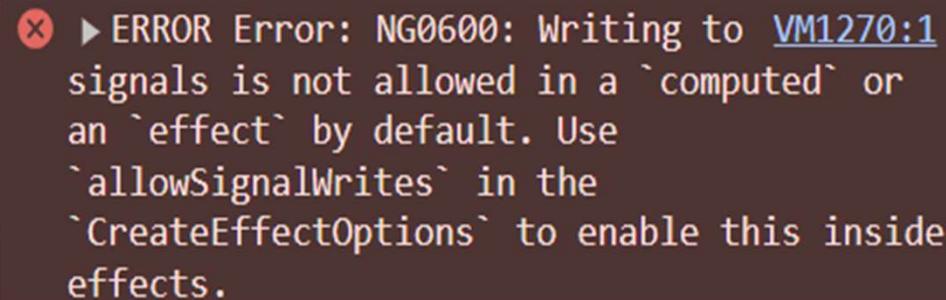
```
lastname = signal('sellaouti');
firstname = signal('aymen');
fullname = computed(() => `${this.firstname()} ${untracked(this.lastname)}`);
```

# computed()

## Modifier un signal dans un computed

- La fonction computed doit être **sans effets de bord**, ce qui signifie qu'elle ne doit accéder qu'aux valeurs des signaux dépendants (ou à d'autres valeurs impliquées dans le calcul) et éviter toute mise à jour.
- Vous **ne pouvez pas modifier un signal dans un computed**, ceci provoquera une **erreur**.

```
fullname = computed(() => {
 console.log('i am computing....');
 this.firstname.set('ccc');
 return `${this.firstname()} ${untracked(this.lastname)}`;
});
```



✖ ▶ ERROR Error: NG0600: Writing to [VM1270:1](#) signals is not allowed in a `computed` or an `effect` by default. Use `allowSignalWrites` in the `CreateEffectOptions` to enable this inside effects.

computed()

## Les computed, autres propriétés

---

- Les computed sont paresseux (**lazy**), ce qui signifie que computed **n'est invoquée que lorsque quelqu'un s'intéresse (lit) sa valeur.** Cela permet **d'optimiser les performances** en évitant les calculs inutiles.
- Les computed sont **automatiquement supprimés** lorsque la référence du signal calculée devient hors de portée.
- Cela garantit que les ressources **sont libérées et qu'aucune opération de nettoyage explicite n'est requise.**
- Ceci est due à l'utilisation des weakReference avec les signaux

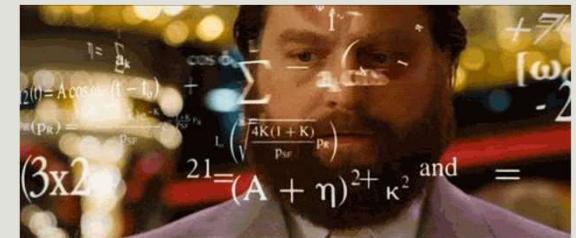
# Writablesignals et signals

---

- ▶ Par défaut, tous les **signaux** créés par la fonction Signal sont de type **WritableSignal**. Ainsi, n'importe quelle entité peut modifier sa valeur (via les méthodes set() et update()).
- ▶ Si on désire **interdire toute modification de la valeur d'un signal**, Angular nous propose la méthode **asReadOnly()**.
- ▶ Les **signaux créés par computed** sont, **par défaut, read only**.

```
private currencies = signal(['USD', 'EUR', 'GBP']);
currencies$ = this.currencies.asReadOnly();
```

# Exercice



- Créer un composant TTC qui permet de calculer le prix TTC d'un produit selon le nombre de pièces et la TVA. Sachant que l'utilisateur peut changer toutes les valeurs et que par défaut la quantité est de 1 le prix est de 0 et la tva est de 18.
- Si le nombre de pièces est entre 10 et 15 une remise de 20% est appliquée.
- Si le nombre de pièces est supérieur à 15 une remise de 30% est appliquée.

## TTC Calculator

Prix HT 100	Quantité 10	TVA 18
Prix unitaire TTC : \$118.00		Prix total TTC : \$1,180.00
Discount : \$0.00		

# L'API effect()

---

- ▶ Dans certains cas d'utilisation, vous avez besoin d'être notifié par le changement d'un signal pour effectuer un effet de bord, donc faire un traitement sans pour autant créer un nouveau signal ou en modifier d'autres.
- ▶ Pensez à un log, à un calcul d'un nombre de click, faire un appel à une API pour enregistrer une valeur,...
- ▶ L'API effect est là afin de vous donner cette possibilité
- ▶ L'effect va être réexécuté si l'un des signaux qu'il utilise émet une nouvelle valeur.

# L'API effect()

- Vous pouvez **créer un effet** via la fonction **effect**.
- Les effets s'exécutent toujours au moins une fois.
- Lorsqu'un effet est exécuté, il **suit tous les signaux qu'il contient**. Chaque fois que l'une de ces valeurs de **signal change**, l'effet se **reproduit**.
- L'effet est **semblable aux computed**, les effets gardent une **trace de leurs dépendances de manière dynamique** et ne suivent que les **signaux** qui ont été **lus lors de l'exécution la plus récente**.

```
constructor() {
 effect(() => {
 console.log(`count:${this.counter()}`);
 });
}
```

```
private logEffect = effect(() => {
 console.log(`
 The current count is:${this.counter()}
 `);
});
```

# L'API effect()

---

- Les effets s'exécutent toujours de manière **asynchrone**, pendant le processus de **change detection**.
- Les effets seront exécutés le **nombre minimum de fois**. Si un effet **dépend** de **plusieurs signaux** et que plusieurs d'entre eux **changent simultanément**, **une seule exécution de l'effet sera programmée**.
- Remarque : l'API effect() est toujours en aperçu développeur (17.3).

```
export class EffectComponent {
 counter = signal(0);
 constructor() {
 this.counter.set(1);
 this.counter.set(2);
 }
 private logEffect = effect(() => {
 console.log(`
 The current count is:
 ${this.counter()}`);
 });
}
```

# L'API effect()

---

- Un **effect** doit être défini dans un contexte d'injection.
- Ceci est faisable dans un component, un pipe, une directive ou le constructeur d'un service.
- Vous pouvez aussi **injecter l'Injector** et le **passer à l'effect en deuxième paramètre** qui représente un objet d'options.

```
private logEffectWithInjector = effect(() => {
 console.log(
 `The current count is: ${this.counter()}`)
);
}, {
 injector: this.injector
});
```

## effect()

# Exclure un signal de l'arbre de dépendence

---

- ▶ Si pour une raison ou une autre, vous voulez lire une valeur d'un signal dans un effect, mais sans l'intégrer dans l'arbre de dépendance, vous pouvez utiliser l'Api **untracked**.
- ▶ **Untracked** peut prendre en paramètre une fonction

```
/**
 * Execute an arbitrary function in a non-reactive (non-tracking) context. The
 * executed function can, optionally, return a value.
 */
export declare function untracked<T>(nonReactiveReadsFn: () => T): T;
```

---

aymen.sellaouti@gmail.com