



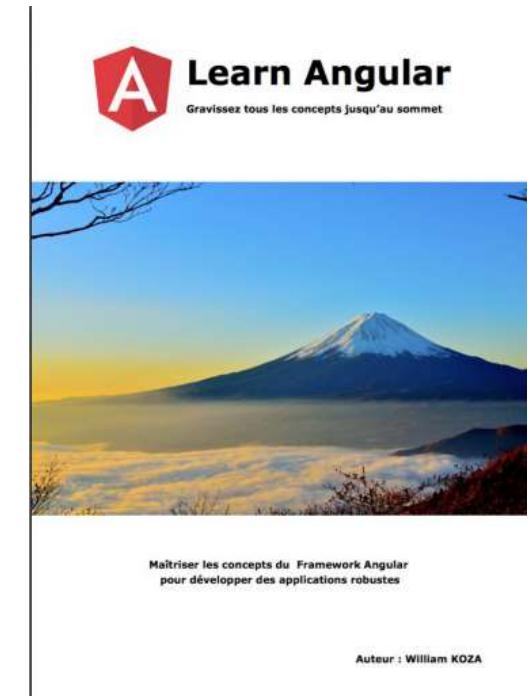
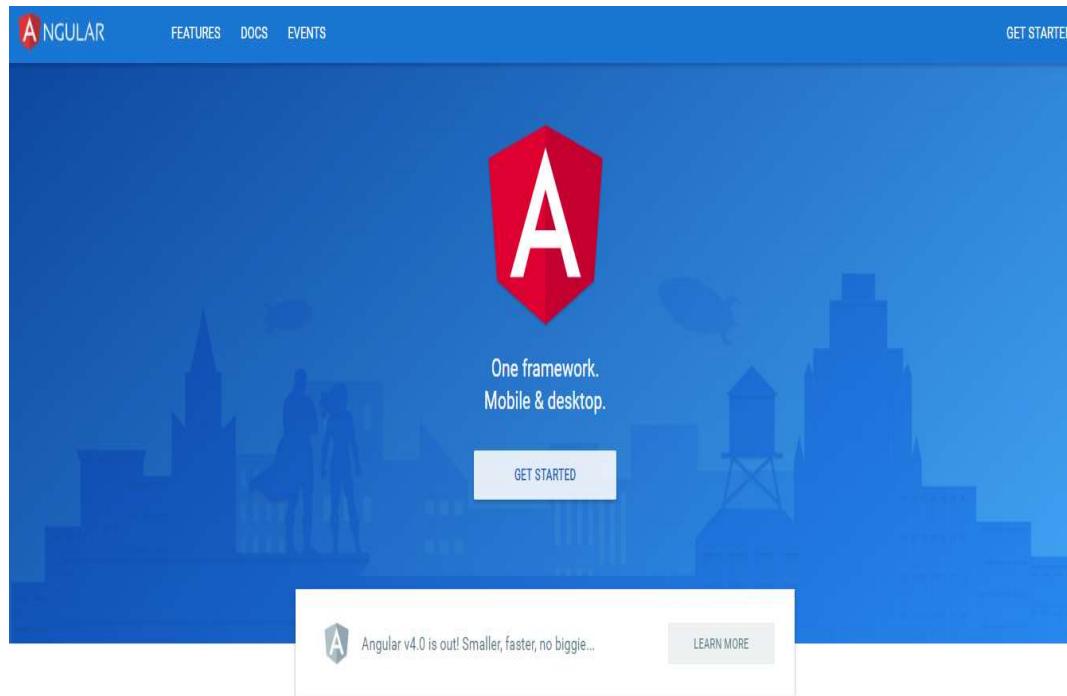
TECHNOLOGIA

Former. Performer. Transformer.

Angular Introduction

Aymen sellaouti

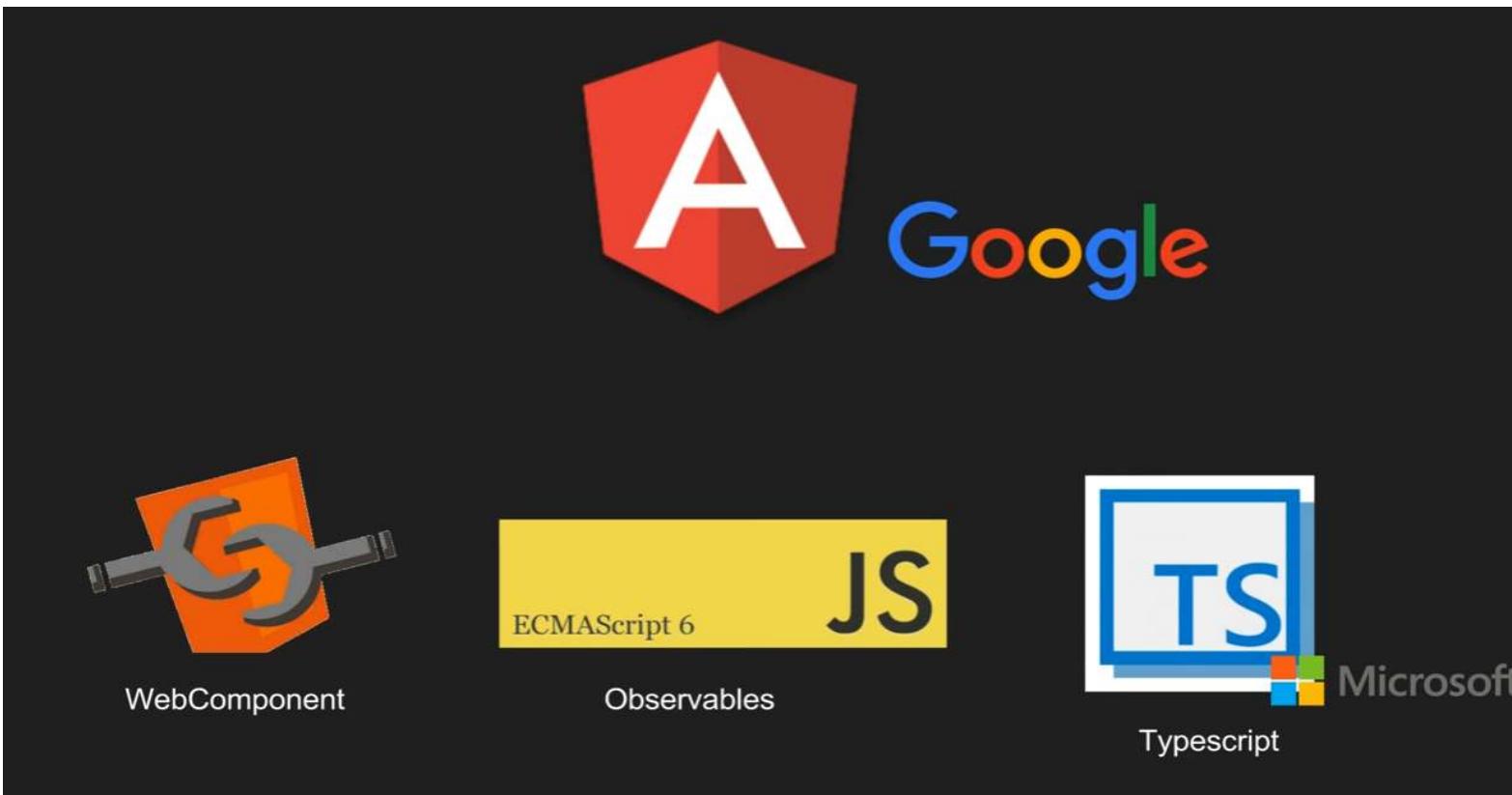
Références



Plan du Cours

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

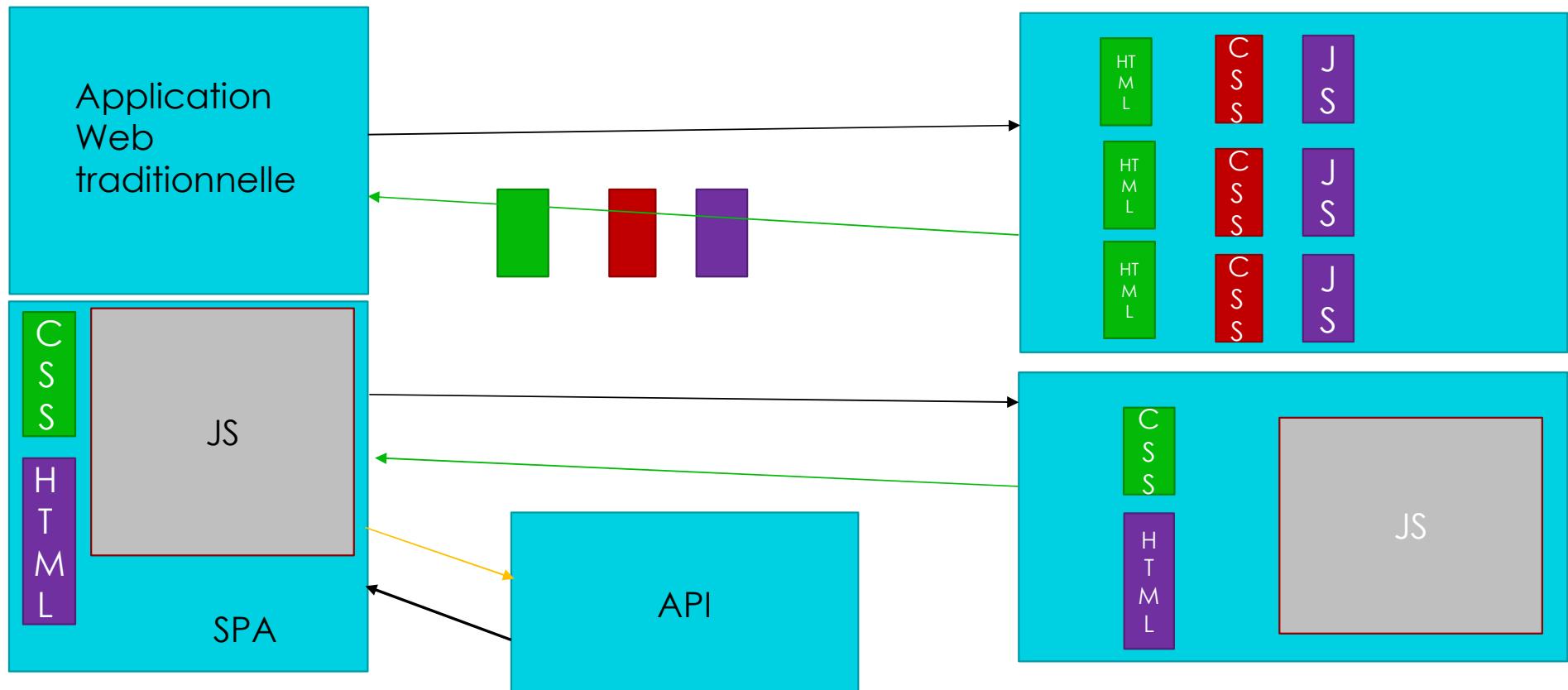
C'est quoi Angular?



C'est quoi Angular?

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

SPA

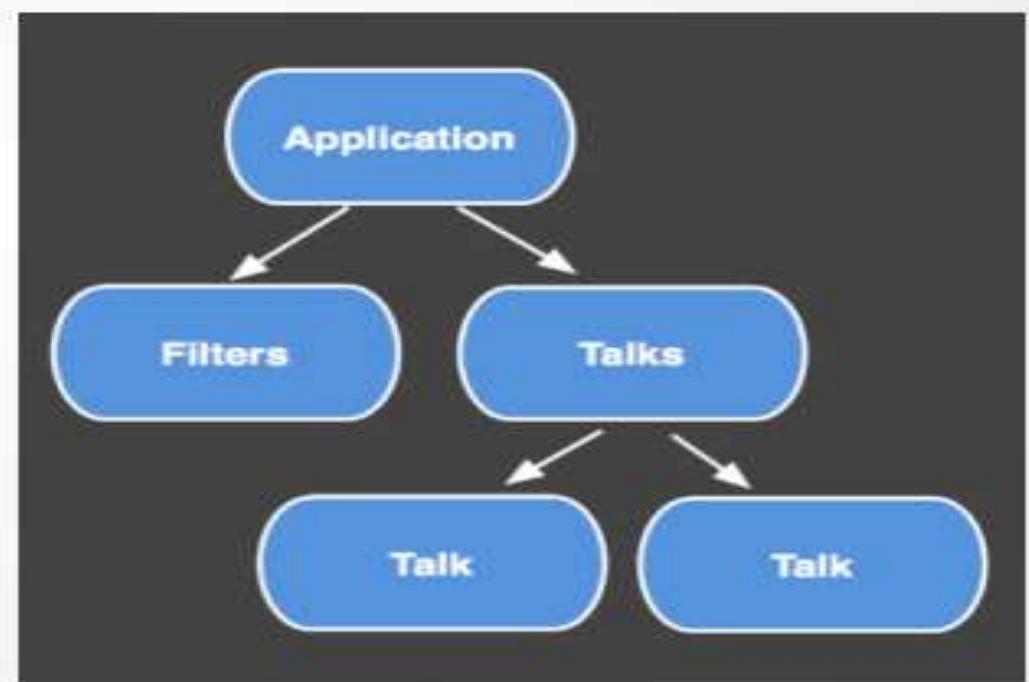


Angular : Arbre de composants

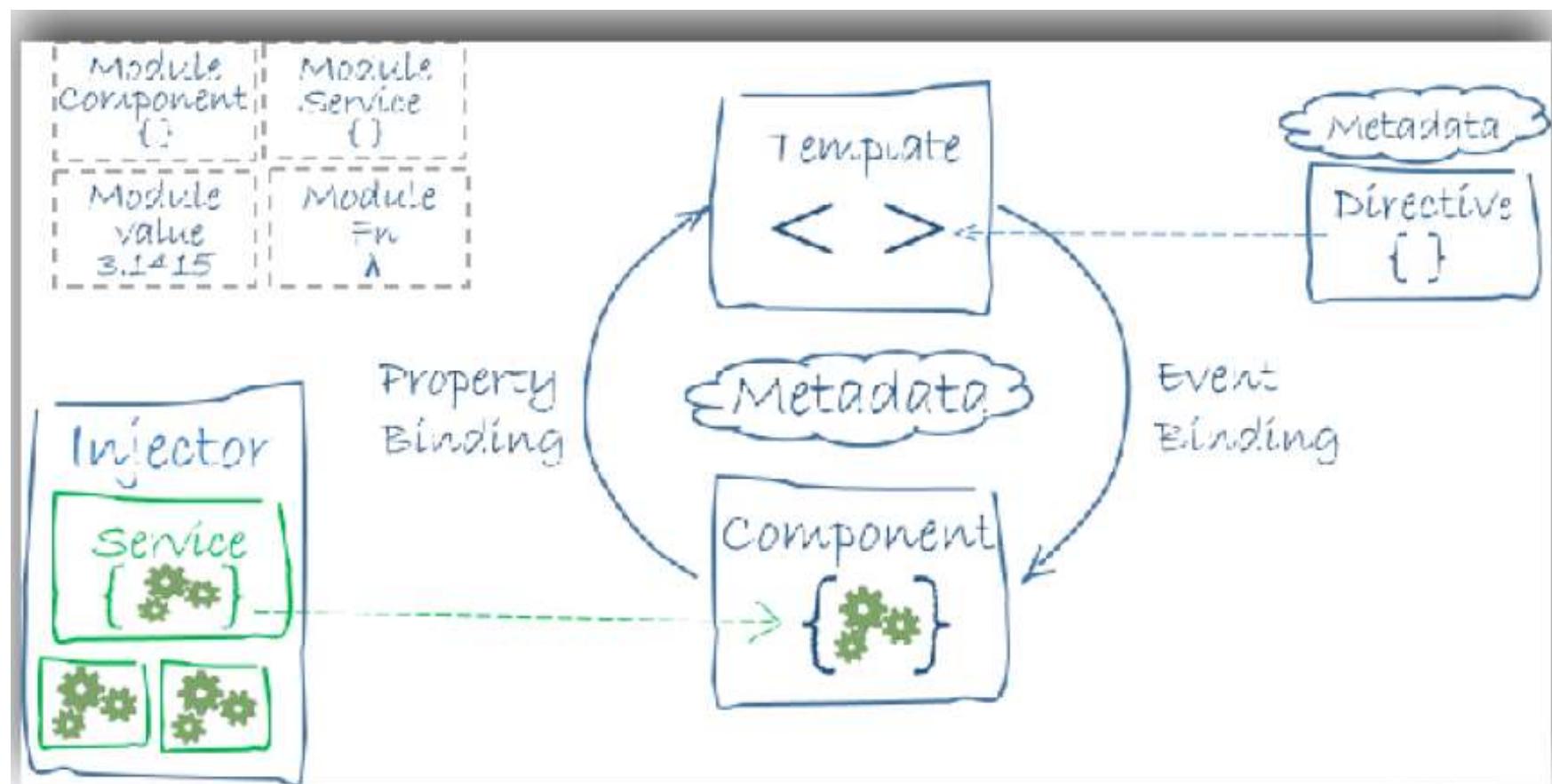
Speaker
Rich Hickey

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



Principaux concepts et notions

Component

Template

DataBinding

Méta données

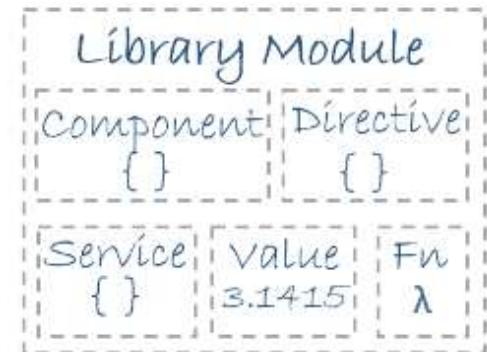
Service

Route

Injection de
dépendance

Les librairies d'Angular

- Ensemble de modules JS
- Des librairies qui contiennent un ensemble de fonctionnalités.
- Toutes les librairies d'Angular sont préfixées par `@angular`
- Récupérable à travers un import JavaScript.
- Exemple pour récupérer l'annotation component : `import { Component } from '@angular/core';`



Les composants

- Le **composant** est la partie principale d'Angular.
- Un composant s'occupe d'une partie de la vue.
- L'interaction entre le composant et la vue se fait à travers une API.

Template

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

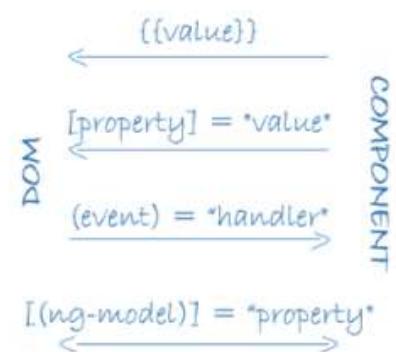
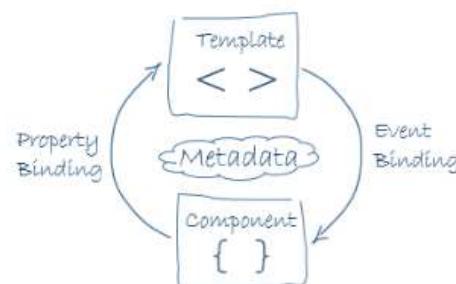
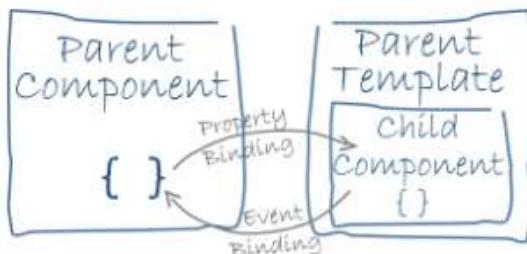
Les métadata



- Appelé aussi « decorator », ce sont des informations permettant de décrire les classes.
- `@Component` permet d'identifier la classe comme étant un composant angular.

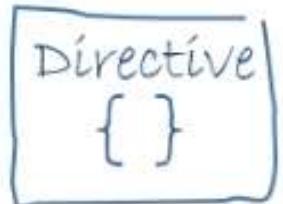
Le Data Binding

- Le data binding est le mécanisme qui permet de mapper des éléments du DOM avec des propriétés et des méthodes du composant.
- Le Data Binding permettra aussi de faire communiquer les composants.



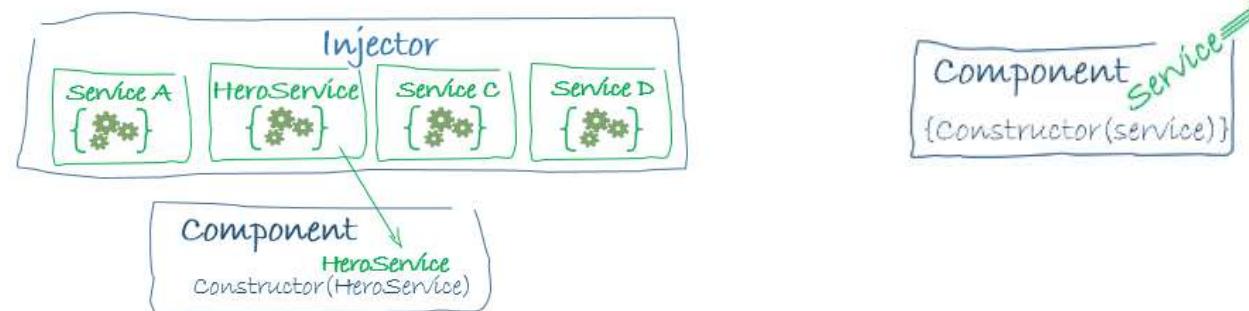
Les directives

- Les directives Angular sont des classes avec la métadata `@Directive`.
- Elles permettent de modifier le DOM et de rendre les Template dynamiques.
- Apparaissent dans des éléments HTML comme les attributs.
- Un composant est une directive à laquelle Angular a associé un Template.
- Il existe deux autres types de directives :
 - Directives structurelles
 - Directive d'attributs



Les services

- Classes permettant d'encapsuler des traitements métiers.
- Doivent être légers.
- Associées aux composants et autres classes par injection de dépendances.



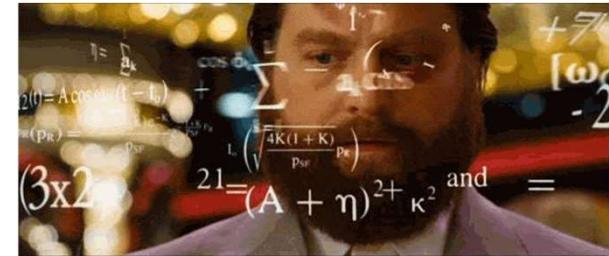
Installation d'Angular

- Deux méthodes pour installer un projet Angular.
 - Cloner ou télécharger le QuickStart seed proposé par Angular.
 - Utiliser le Angular-cli afin d'installer un nouveau projet (conseillé).
 - Remarque : L'installation de NodeJs est obligatoire afin de pouvoir utiliser son npm (Node Package Manager).

Installation d'Angular QuickStart

- Deux méthodes
 - Télécharger directement le projet du dépôt Git
 - <https://github.com/angular/quickstart>
 - Ou bien le cloner à l'aide de la commande suivante :
`git clone https://github.com/angular/quickstart.git quickstart`
 - Se positionner sur le projet
 - Installer les dépendances à l'aide de npm : `npm install`
 - lancer le projet à l'aide de npm : `npm start`

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@17.3.11` installe la version **17.3.11**
 - [`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@17.3.11 new projectName`
- Afin d'avoir du help pour le cli tapez [`ng help`](#)
- Lancer le projet en utilisant la commande [`ng serve`](#)

<https://cli.angular.io/>

Angular dépendances

Angular CLI, Angular, Node.js, TypeScript, and RxJS version compatibility matrix. Officially part of the Angular documentation as of 2023-04-19
<https://angular.io/guide/versions>

[Raw](#)

Angular CLI version	Angular version	Node.js version	TypeScript version	RxJS version
~16.0.0	~16.0.0	^16.13.0 ^18.10.0	>=4.9.5 <5.1.0	^6.5.5 ^7.4.0
~15.2.0	~15.2.0	^14.20.0 ^16.13.0 ^18.10.0	>=4.8.4 <5.0.0	^6.5.5 ^7.4.0
~15.1.0	~15.1.0	^14.20.0 ^16.13.0 ^18.10.0	>=4.8.4 <5.0.0	^6.5.5 ^7.4.0
~15.0.5	~15.0.4	^14.20.0 ^16.13.0 ^18.10.0	~4.8.4	^6.5.5 ^7.4.0
~14.3.0	~14.3.0	^14.15.0 ^16.10.0	>=4.6.4 <4.9.0	^6.5.5 ^7.4.0
~14.2.0	~14.2.0	^14.15.0 ^16.10.0	>=4.6.4 <4.9.0	^6.5.5 ^7.4.0
~14.1.3	~14.1.3	^14.15.0 ^16.10.0	>=4.6.4 <4.8.0	^6.5.5 ^7.4.0
~14.0.7	~14.0.7	^14.15.0 ^16.10.0	>=4.6.4 <4.8.0	^6.5.5 ^7.4.0
~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
~13.2.6	~13.2.7	^12.20.2 ^14.15.0 ^16.10.0	>=4.4.4 <4.6.0	^6.5.5 ^7.4.0
~13.1.4	~13.1.3	^12.20.2 ^14.15.0 ^16.10.0	>=4.4.4 <4.6.0	^6.5.5 ^7.4.0
~13.0.4	~13.0.3	^12.20.2 ^14.15.0 ^16.10.0	~4.4.4	^6.5.5 ^7.4.0
~12.2.18	~12.2.17	^12.14.1 ^14.15.0	>=4.2.4 <4.4.0	^6.5.5 ^7.0.1
~12.1.4	~12.1.5	^12.14.1 ^14.15.0	>=4.2.4 <4.4.0	^6.5.5
~12.0.5	~12.0.5	^12.14.1 ^14.15.0	~4.2.4	^6.5.5
~11.2.19	~11.2.14	^10.13.0 ^12.11.1	>=4.0.8 <4.2.0	^6.5.5
~11.1.4	~11.1.2	^10.13.0 ^12.11.1	>=4.0.8 <4.2.0	^6.5.5

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

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";*
- Essayer la même chose avec font-awesome.

Angular Les composants



Objectifs

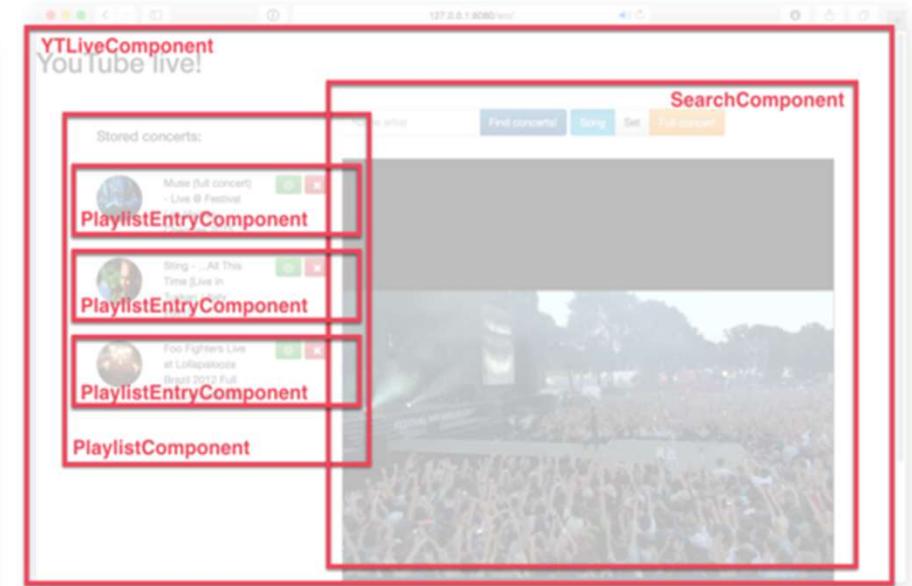
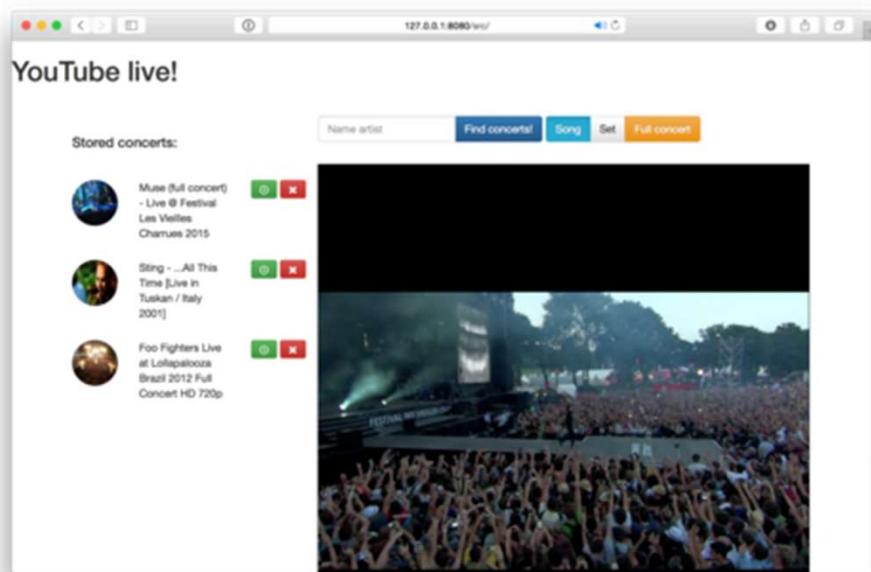
1. Comprendre la définition du composant
2. Assimiler et pratiquer la notion de Binding
3. Gérer les interactions entre composants.

Qu'est-ce qu'un composant (Component)

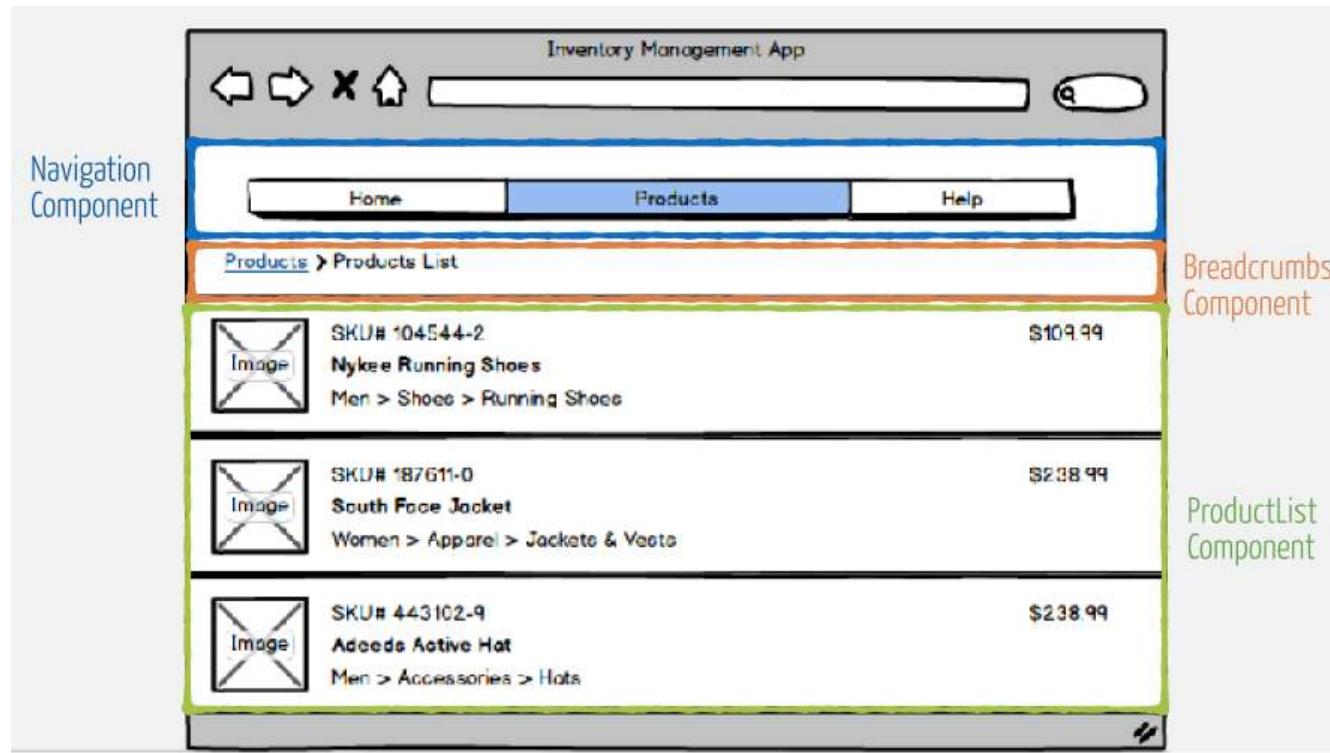
- Un **composant** est une **classe** qui permet de **gérer une vue**. Il se charge **uniquement** de cette vue là.
Plus simplement, un composant est un fragment HTML géré par une classe JS (component en angular et controller en angularJS)
- Une application Angular est un **arbre de composants**
- La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



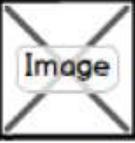
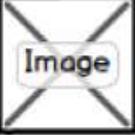
The screenshot shows a web application window titled "Inventory Management App". The top bar includes standard browser controls (back, forward, search, etc.) and a title bar. Below the title bar is a navigation bar with three items: "Home", "Products" (which is highlighted in blue), and "Help". A search bar is positioned to the right of the navigation bar. The main content area has a header "Products > Products List". The content is divided into three horizontal sections, each representing a product item:

- Product 1:** SKU# 104544-2, Nykee Running Shoes, \$109.99. Category: Men > Shoes > Running Shoes. Includes an "Image" placeholder icon.
- Product 2:** SKU# 187611-0, South Face Jacket, \$238.99. Category: Women > Apparel > Jackets & Vests. Includes an "Image" placeholder icon.
- Product 3:** SKU# 443102-9, Adeeds Active Hat, \$238.99. Category: Men > Accessories > Hats. Includes an "Image" placeholder icon.

Annotations on the left side identify the "Navigation Component" (covering the top bar and part of the content), the "Breadcrumbs Component" (covering the breadcrumb header), and the "ProductList Component" (covering the main content area). The "Image" placeholder icons are labeled "Image" in a small box.

Quelques exemples

Product Row Component

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

Quelques exemples



```
@Component({  
  selector: 'app-standalone',  
  standalone: true,  
  imports: [CommonModule],  
  templateUrl: './standalone.component.html',  
  styleUrls: ['./standalone.component.css']  
})  
export class StandaloneComponent {}
```

Premier Composant

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

Import: permet d'importer toutes les dépendances du 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
 - **Si l'application est modulaire**, 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**

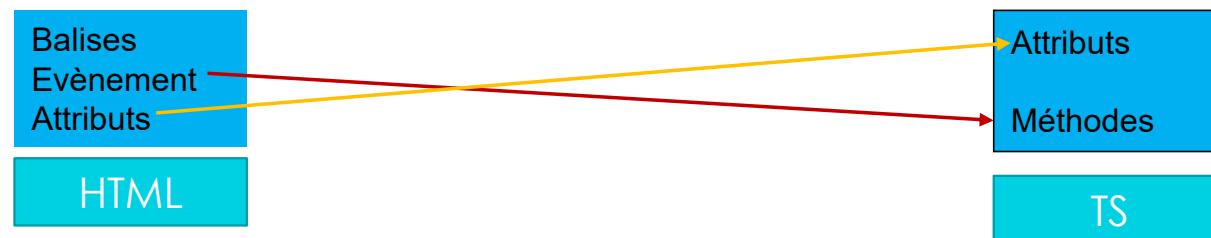
Création d'un composant

- 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>

<https://angular.io/cli/generate>

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.
- Deux possibilités pour la syntaxe:
 - [propriété]="**varOuCte**"
 - **bind**-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.
- Deux possibilités pour la syntaxe :
 - <(evenement)="fct()">
 - **on-evenement**

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

```

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

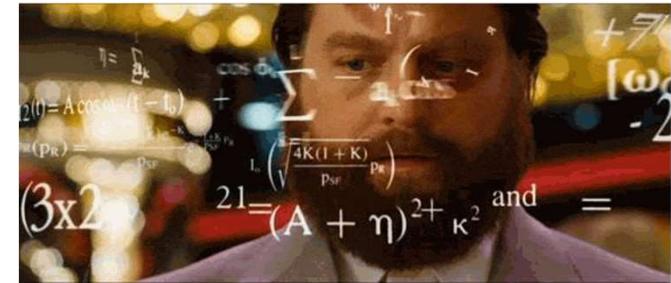
```

<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.
- Faites en sorte que lorsqu'on écrit une couleur dans l'input, ça 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]**

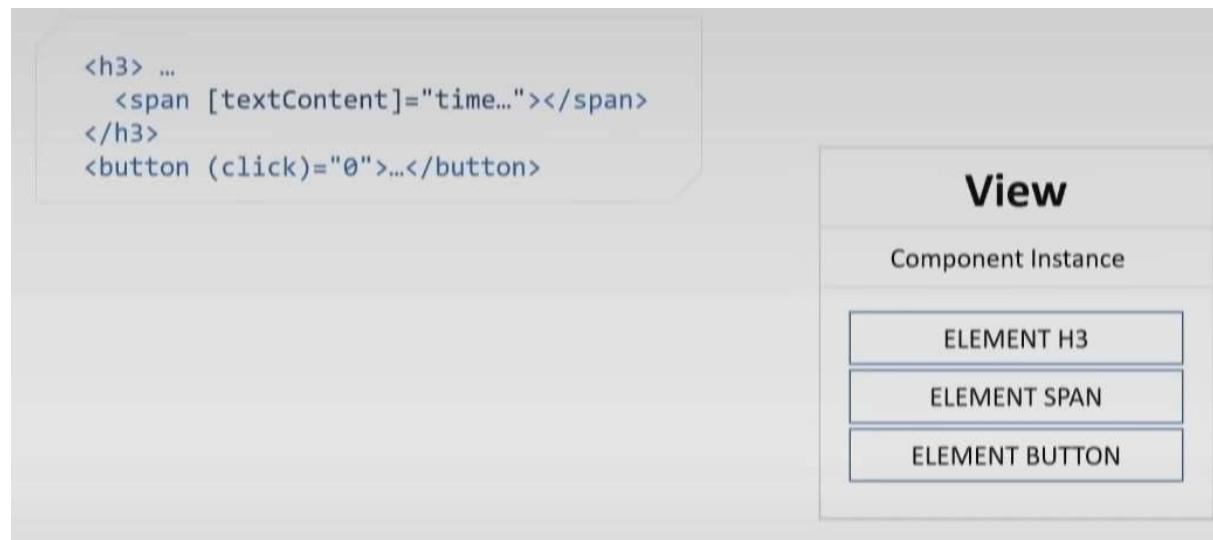
Change Detection

C'est quoi ?

- **Change Detection** est l'un des mécanismes les plus importants dans Angular.
- Il permet de **suivre les changements d'état** de votre application et **d'afficher les modifications** dans votre vue.
- Il **garantit que l'interface utilisateur suit toujours** d'une façon synchrone **l'état interne de votre application**.

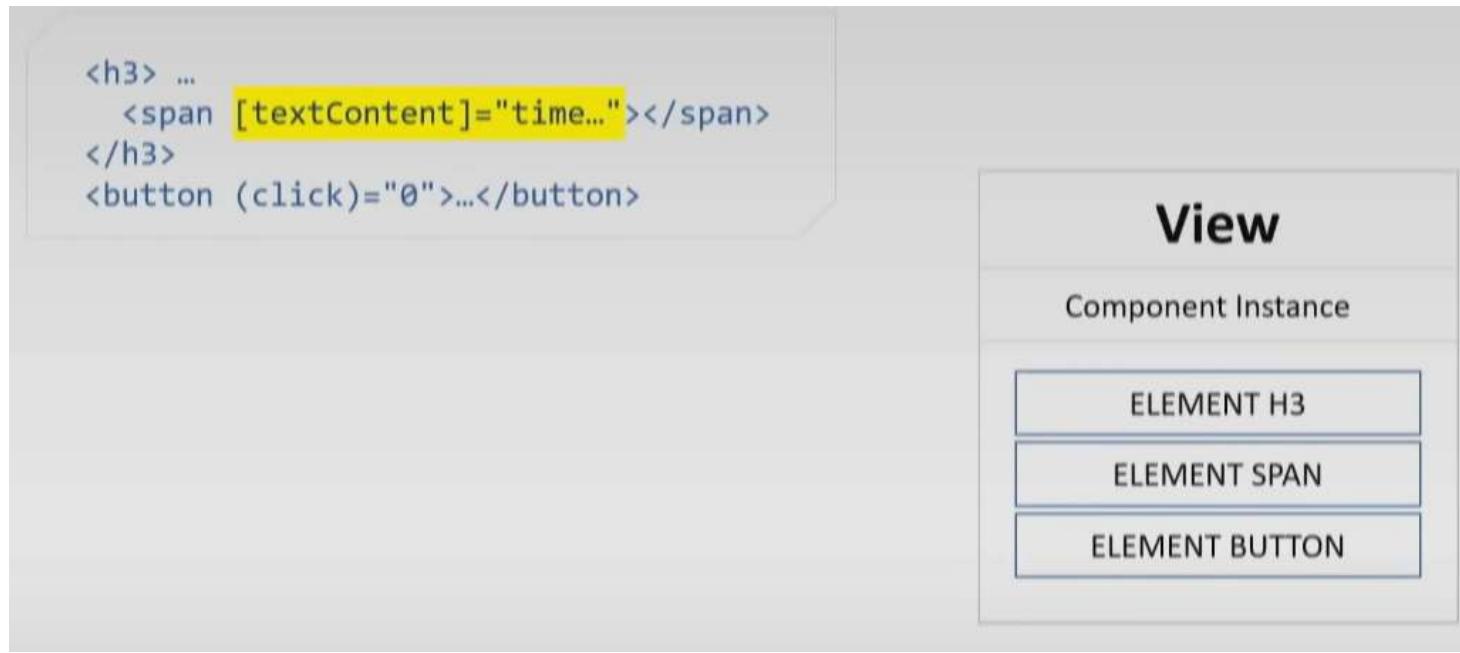
Change Detection

- **Chaque composant** dans Angular est représenté par une structure appelée **View**. Elle contient entre autres **l'instance** de la classe Composant appelée **componentInstance** ainsi que la **liste** des **éléments du DOM** représentant le Template.



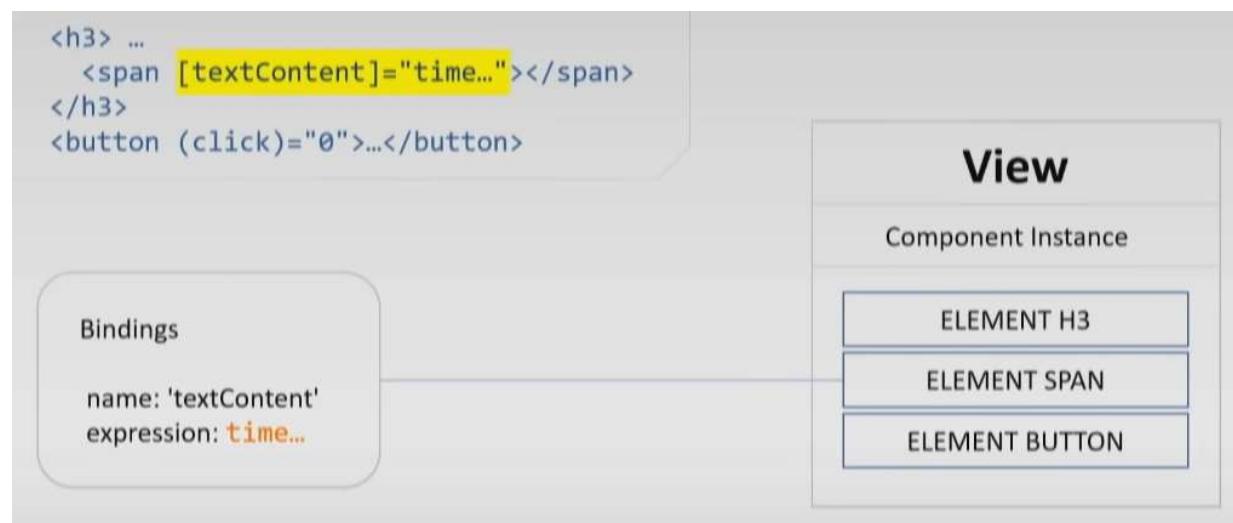
Change Detection

- Ensuite, quand le compilateur traite le composant, il **identifie les éléments qui nécessite un changement** lors du changement d'état.



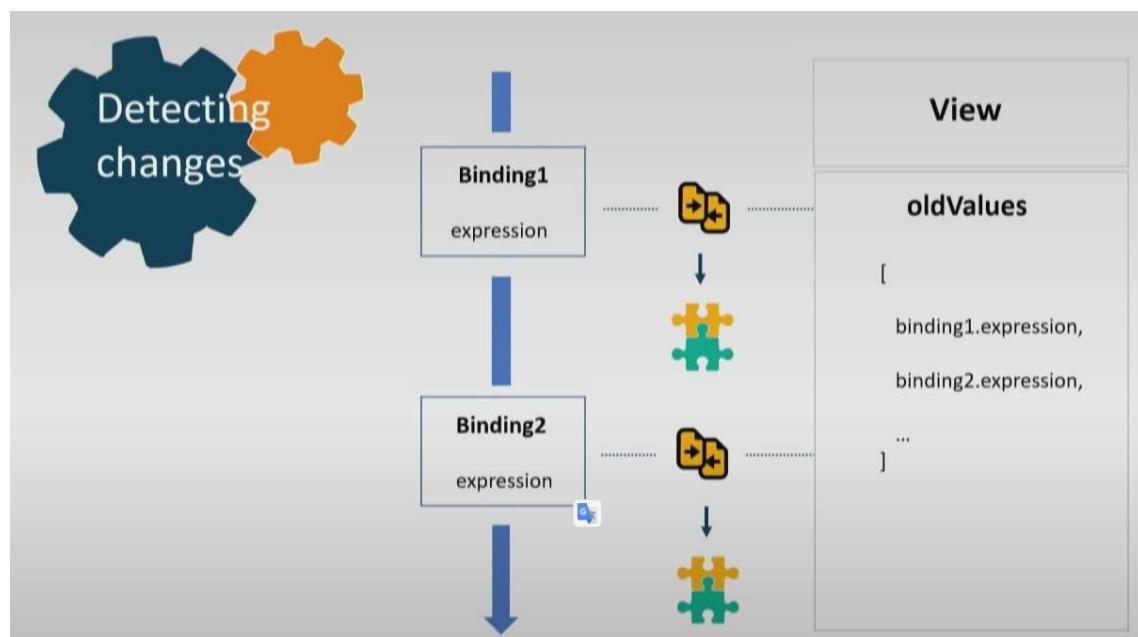
Change Detection

- Pour chacun de ces éléments, il crée des objets Bindings. C'est une structure de données qui informe sur deux choses :
 - Que voulons nous mettre à jour dans le Dom
 - Ou récupérer la nouvelle valeur



Change Detection

- Ensuite, dès qu'un **change Detection** est déclenché, Angular va parcourir l'ensemble des Views (Component) et évaluer la nouvelle expression du Binding et la **comparer à la précédente**.
- Si la valeur est modifiée, elle met à jour le DOM.



Change Detection

Quand déclencher un Change Detection

- Un Change Detection est déclenché dans ces cas d'utilisation
1. **Initialisation des composants.** Par exemple, lors du lancement d'une application angular, Angular charge le composant principal et déclenche **ApplicationRef.tick()** pour appeler la détection de changement et le rendu de la vue.
 2. Les **event listener** du DOM peuvent mettre à jour les données dans un composant Angular et déclencher le Change Detection.
 3. **Les requêtes HTTP.**

Change Detection

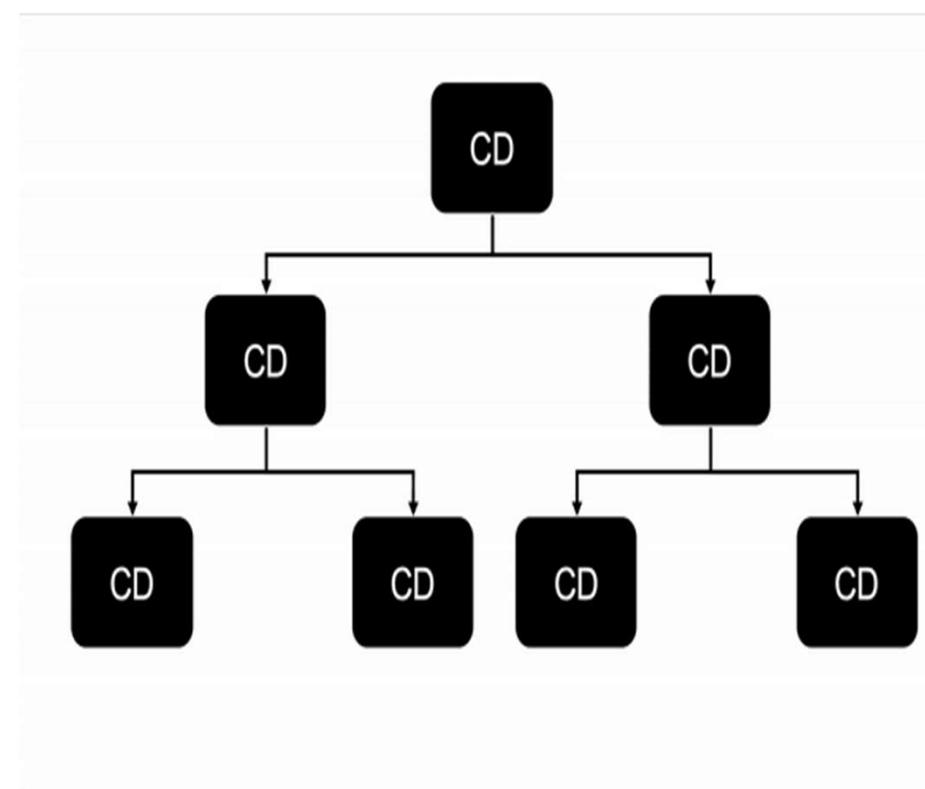
Quand déclencher un Change Detection

4. Les **MacroTasks**, tells que `setTimeout()` ou `setInterval()`.
En effet vous pouvez mettre à jour les données dans la callback function d'une macroTask comme `setTimeout()`.
5. Les **MicroTasks**, comme `Promise.then()` dont les callback peuvent mettre à jour les données.
6. **D'autres opérations asynchrones** qui peuvent mettre à jour vos données telles que `WebSocket.onmessage()` et `Canvas.toBlob()`.

Change Detection

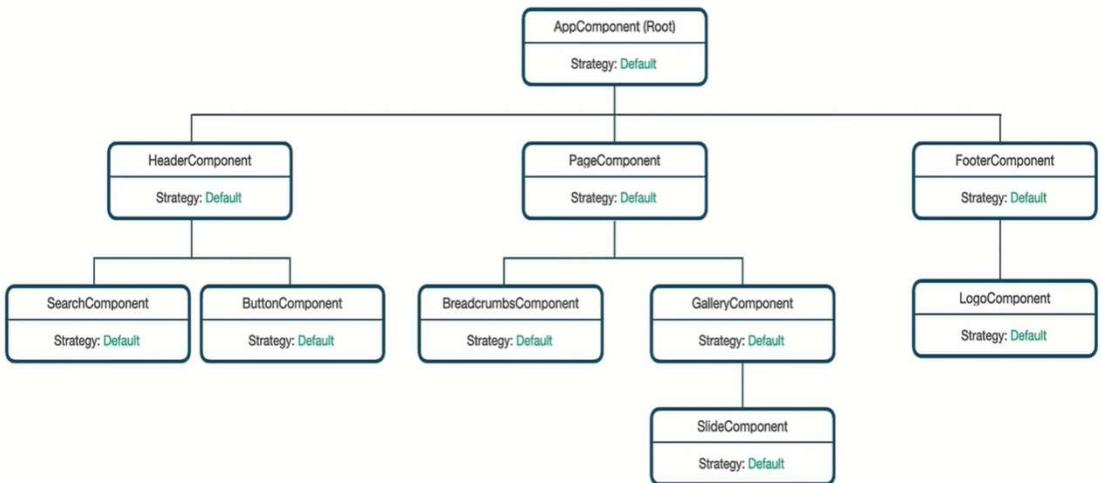
La stratégie par défaut

- Ici, dans **tous les composants** de l'arbre de composant, le **change detector alloué à chaque composant**, compare la valeur courante et la valeur précédente des propriétés.
- Si la **valeur change**, il va marquer une propriété **isChanged à true**.



Change Detection

Scenario 1



Change Detection Problème

- Le problème majeur d'Angular était **son incapacité à détecter le changement et où il se produit exactement.**
- Ceci est la cause majeure du parcours de **l'intégralité** de l'arbre afin **d'identifier l'endroit exact où le changement s'est effectué.**
- Ceci permet à angular de **minimiser la mise à jour du DOM** qui est une opération **très couteuse.**

Change Detection

- Quand un événement se déclenche dans votre application, Angular **parcourt tout votre arbre de composants** pour **chercher où les modifications doivent être effectuées**.
- Ce parcours, Angular le fait de manière très rapide mais le processus pourrait être encore plus rapide.
- D'où l'intérêt d'utiliser les **signaux**, qui vont assister Angular **en lui indiquant où exactement il doit checker les changements**.
- Les changements peuvent être opérés par Angular dans une petite partie d'un template (un bloc **@if** ou **@for**).

Performances

- Les **signaux** permettent de **réduire le nombre de calculs effectués lors de la détection des changements** dans une application Angular. Cela se traduit par de meilleures performances d'exécution.
- Avec les **signaux**, il **sera** possible de **vérifier les changements uniquement dans les composants concernés**.
- Les **signaux vont permettre** de **rendre Zone.js facultatif** dans les versions futures d'Angular. **Zone.js** est une bibliothèque utilisée par Angular pour détecter les changements et exécuter les tâches asynchrones

Qu'est ce qu'un Signal ?



- Un signal agit comme une **enveloppe (wrapper) autour d'une valeur**, mais avec la capacité supplémentaire 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 est 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 **certain 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.

Signal 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**.

```
@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('sellaouti');
}
```

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('sellaouti');
}
```

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

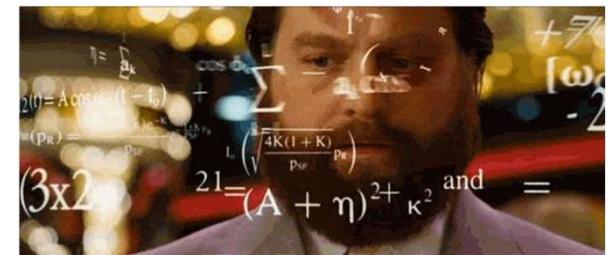


- Pour modifier la valeur d'un **signal**, on peut passer par :
- La Méthode **set**, qui permet **d'affecter une nouvelle valeur au signal**.
- La méthode **update**, qui permet de **calculer une nouvelle valeur** d'un signal **en fonction de sa valeur précédente**.

```
@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 créée 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('sellouti');
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())}`;
});
```

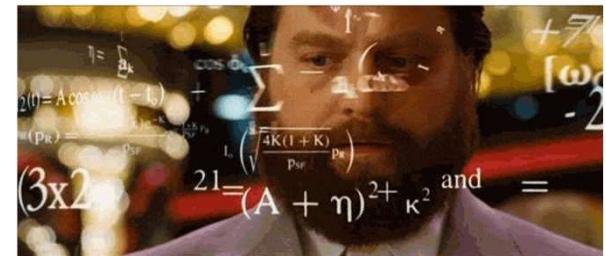
✖ ➔ ERROR Error: NG0600: Writing to [VM1270:1](#)
signals is not allowed in a `computed` or
an `effect` by default. Use
`allowSignalWrites` in the
`CreateEffectOptions` to enable this inside
effects.

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

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		

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.as_READONLY();
```

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';

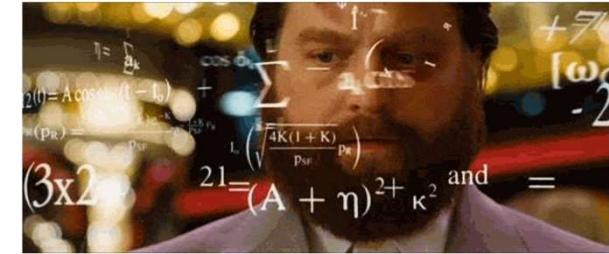
@Component({
  selector: 'app-two-way',
  templateUrl: './two-way.component.html',
  styleUrls: ['./two-way.component.css']
})
export class TwoWayComponent {
  two:any="myInitValue";
}
```

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 côté les données à insérer dans une carte visite. De l'autre côté et instantanément les données de la carte seront mises 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

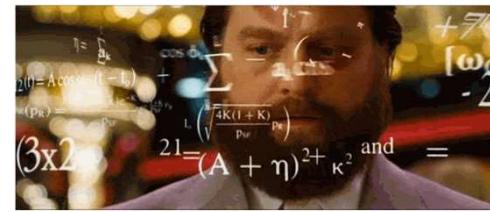
A digital card for Aymen Sellaouti, a trainer. The card features a circular profile picture of Aymen, his name, title, a quote, and an "Auto Rotation" button.

Aymen Sellaouti
trainer

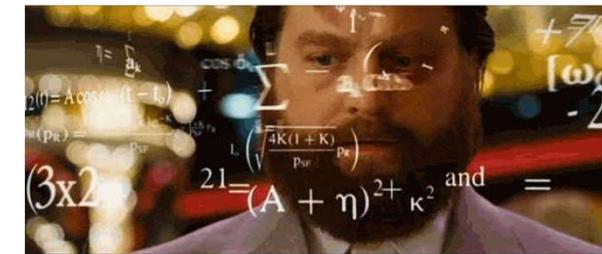
"I'm the new Sinatra, and since I made it here I can make it anywhere, yeah,
they love me everywhere"

Auto Rotation

name : sellaouti
firstname : aymen
job : trainer
path : rotating_card_profile3.png



Exercice



Two Way Binding



Sellaouti Aymen
Enseignant

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

Auto Rotation

Nom :

Prénom :

Job :

image :

Citation Favorite :

Décrivez nous votre travail :

Mots clé de votre travail :

Two Way Binding

"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 :

Prénom :

Job :

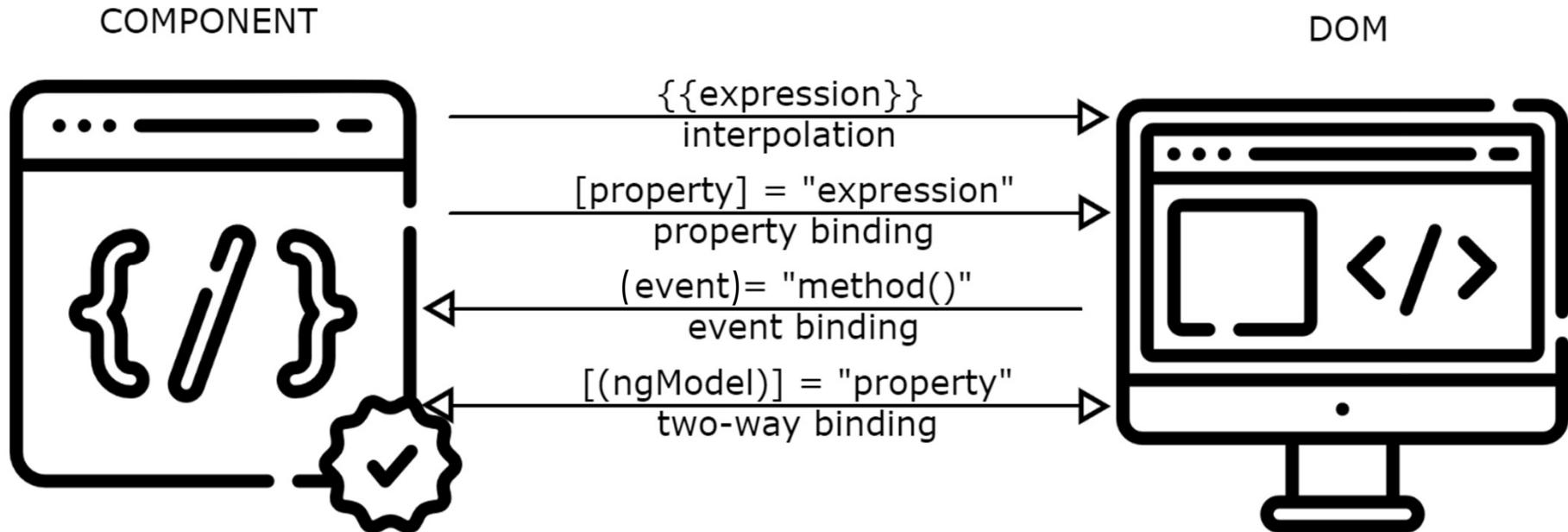
image :

Citation Favorite :

Décrivez nous votre travail :

Mots clé de votre travail :

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

Angular 17

Le nouveau flux de control

- Afin de continuer sur la logique de simplification de l'apprentissage d'Angular, l'équipe Angular a proposé de **nouveaux flux de contrôle**.
- Les flux suivants ont été proposés :
 - @if
 - @for
 - @switch

Control flow

@if

- **@if** est associé à une expression booléenne. Si cette expression est fausse (false) alors l'élément et son contenu sont retirés du DOM, ou jamais ajoutés).
- **@if(condition)**
 - Si le booléen est true alors l'élément host est visible.
 - Si le booléen est false alors l'élément host est caché.

```
@if (authService.isAuthenticated()) {  
  <button  
    (click)="deleteCv(cv)"  
    class="btn btn-danger">  
    Delete  
  </button>  
}
```

Control flow

@if, @else if et @else

- @if peut également être utilisé avec @else if et/ou @else selon le besoin.
- La **syntaxe est très intuitive**, demandé vous comment vous aurez fait en Javascript et **ajoutez un @ :D**.

```
@if (connectedUser) {  
    Hello {{ connectedUser.name }}  
} @else {  
    <div>Merci de vous connectez</div>  
}
```

Control flow

@if, @else if et @else

```
@if (!connectedUser) {  
    <div class="alert alert-danger">Merci de vous connectez</div>  
} @else if (!connectedUser.activated) {  
    <div class="alert alert-warning">  
        Hello {{ connectedUser.name }}, merci d'activer votre compte  
    </div>  
} @else {  
    <div class="alert alert-success">Hello {{ connectedUser.name }}</div>  
}
```

Control flow

@for

- La directive structurelle **@for** permet de boucler sur un itérable et d'injecter les éléments dans le DOM.

```
<ul>
  @for (episode of episodes; track episode.id) {
    <li>{{ episode.title }}</li>
  }
</ul>
```



Control flow

@for

➤ **@for** fournit certaines informations sur la boucle en cours :

- **\$index**: position de l'élément.
- **\$odd**: true si l'élément est à une position impaire.
- **\$even**: true si l'élément est à une position paire.
- **\$first**: true si l'élément est à la première position.
- **\$last**: true si l'élément est à la dernière position.

```
<ul>
  @for (episode of episodes; track episode.id) {
    <li>
      Episode {{ $index + 1 }} : {{ episode.title }}
    </li>
  }
</ul>
```

Control flow

@for track

- Avec @For la **fonction de tracking est devenue obligatoire**
- La fonction de suivi créée via l'instruction **track** est **utilisée pour permettre au mécanisme de détection des changements d'Angular de savoir exactement quels éléments mettre à jour dans le DOM** après les modifications de l'itérable d'entrée.
- La fonction de suivi **indique à Angular comment identifier de manière unique un élément de la liste.**

```
@for (episode of episodes; track episode.id) {  
  <li>{{ episode.title }}</li>  
}
```

Control flow

@for track

- En principe, il devrait toujours y avoir quelque chose d'unique dans les éléments sur lesquels vous itérer.
- Dans le pire des cas, s'il n'y a rien d'unique dans les éléments du tableau, vous pouvez utiliser \$index de l'élément, c'est-à-dire la position de l'élément dans le tableau.

```
@for (episode of episodes; track $index) {  
    <li>{{ episode.title }}</li>  
}
```

Control flow

@For, la gestion d'un itérable vide

- Afin de gérer le cas où votre itérable est vide, nous avons le block @empty.
- Ce bloque n'est activé que si l'itérable sur lequel vous bouclez est vide.

```
<ol class="list-group">
  @for (player of players; track player.id) {
    <li class="list-group-item">{{player.name}}</li>
  }
  @empty {
    <li class="list-group-item list-group-item-danger">La liste ne nous est
      pas encore parvenue</li>
  }
</ol>
```

Control flow **@switch**

- Avec **@switch**, vous pouvez créer des switch très simplement.
- Vous avez les trois opérateurs :
@switch, @case, @default
 - **@switch** : définir l'élément sur lequel switcher
 - **@case** : pour identifier le cas
 - **@default** : pour les valeurs par défaut

```
@switch(streamingService) {  
    @case ('Disney+') {  
        <div>'Mandalorian'</div>  
    } @case ('AppleTV') {  
        <div>'Ted Lasso'</div>  
    } @default {  
        <div>'Peaky Blinders'</div>  
    }  
}
```

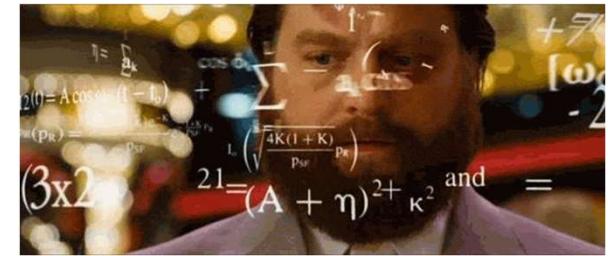
Exercice

- Créez un nouveau composant names.
- Définissez un tableau de noms
- Affichez la liste des noms et faites en sorte que les noms dont la taille dépasse 10 caractères soit suivie d'une icône de votre choix.

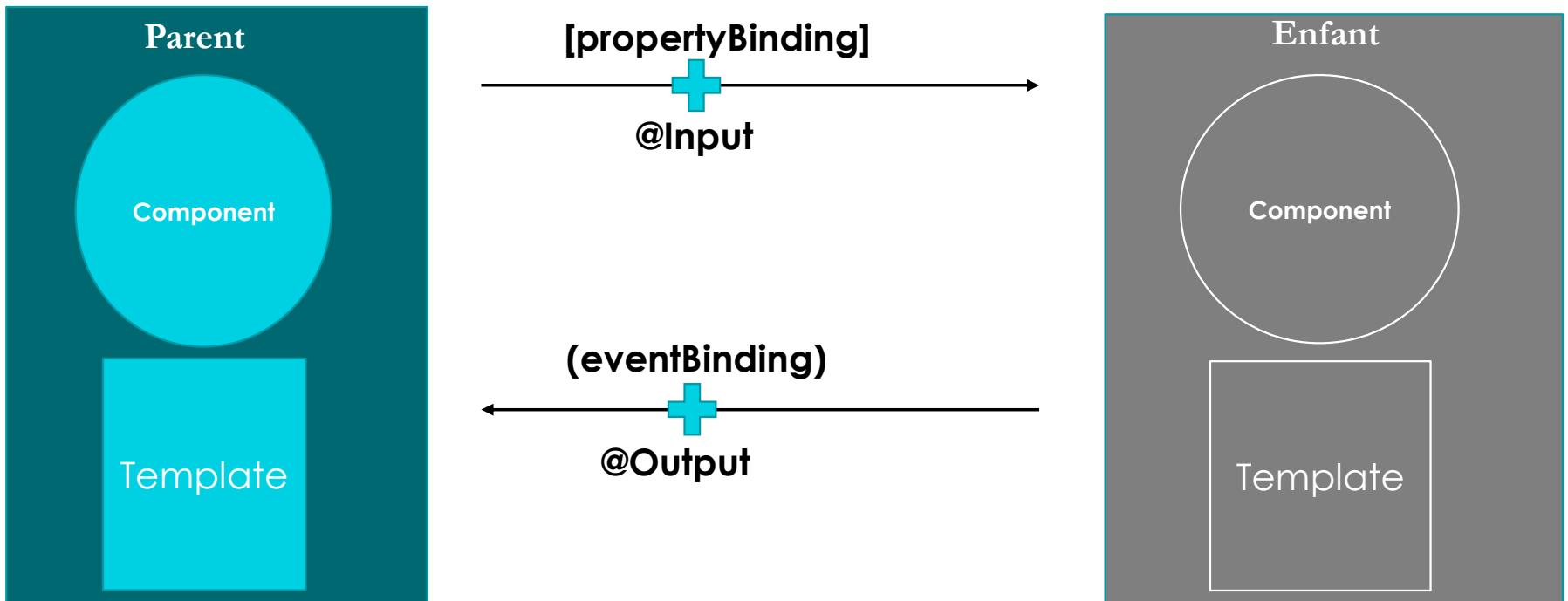
sellaouti

lewandowski 

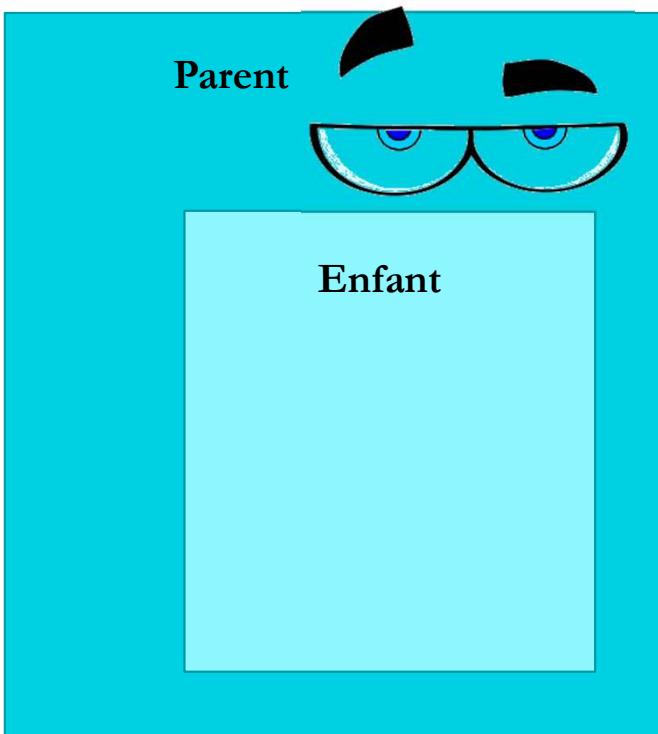
dupont



Interaction entre composants



Pourquoi ?

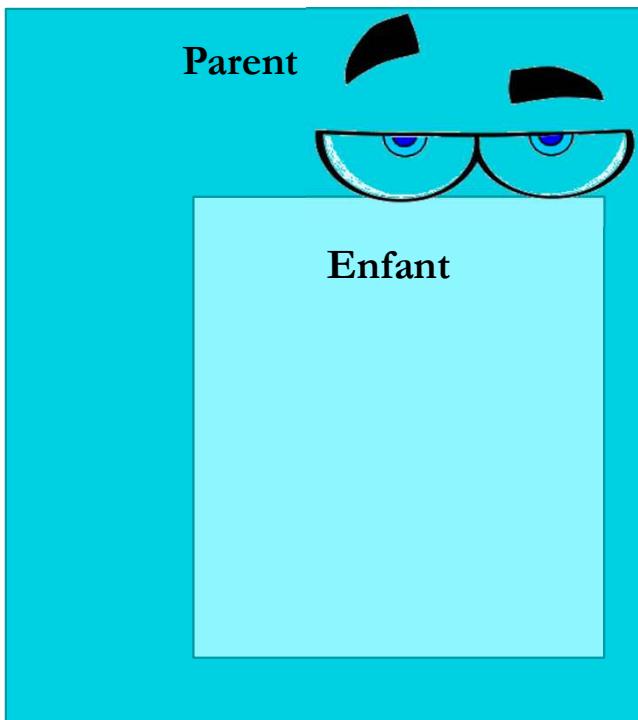


```
import { Component } from '@angular/core';

@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

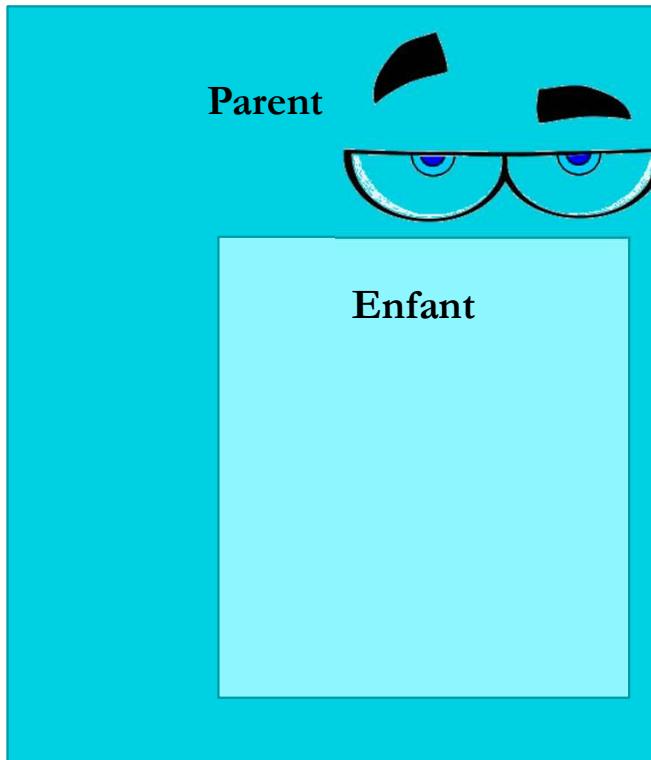
Le père peut directement envoyer au fils des données par Property Binding



```
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

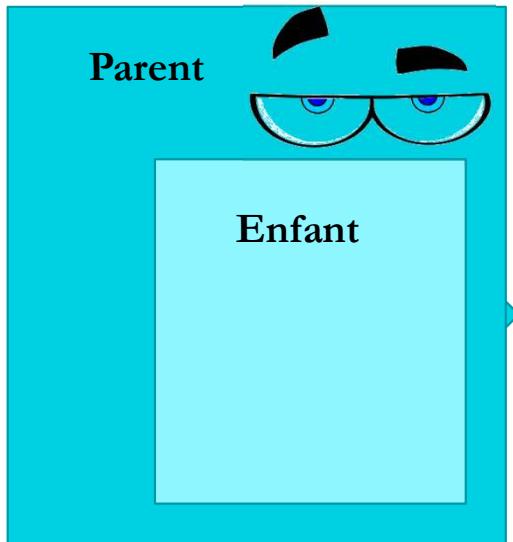
Problème : Le père voit le fils mais pas ces propriétés !!! Solution : les rendre visible avec Input



```
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';

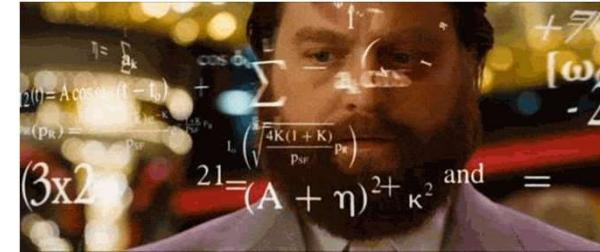
@Component({
  selector: 'app-root',
  template: `
    <p>Je suis le composant père</p>
    <forma-fils [external]="title">
    </forma-fils>
  `,
  styles: []
})
export class AppComponent {
  title = 'app works !';
}
```

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

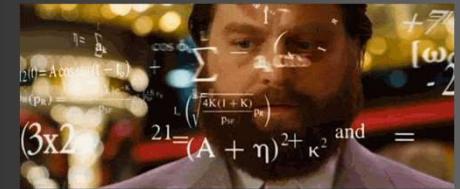
@Component ({
  selector: 'app-input',
  templateUrl:
  './input.component.html',
  styleUrls:
  ['./input.component.css']
})
export class
InputComponent {
  @Input()
  external:string;
}
```

Exercice

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



Signal Input



- Commençons cette partie avec un petit exercice. Créer un composant isEven qui récupère un entier en input et qui affiche s'il est pair ou impair.
- Créer un autre composant qui contient un champ de texte de type number.
- Ce composant doit appeler le composant isEven et lui passer en paramètre la valeur de l'input.



Signal Input

- Pour que l'exemple précédent fonctionne, on avait deux méthodes :
 - Utiliser un setter
 - Utiliser le OnChange hook
- La version 17.1 a vu Angular introduire le concept de *Signal Input* pour améliorer l'interaction entre les composants.
- Les Signal Input permettent de lier les valeurs des composants parents. Ces valeurs sont exposées à l'aide d'un signal et peuvent changer au cours du cycle de vie de votre composant.

```
import { Input, input } from '@angular/core';
isEven!: boolean;
// Sans Signal Input
@Input() isEven: number;
// Avec les signal Input
counter = input<number>();
```

Signal Input

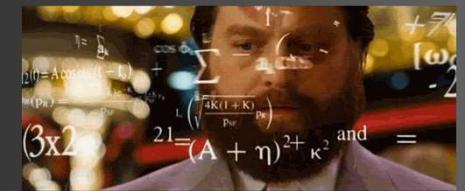
- Comme les **@Input**, le Signal Input peut être optionnel ou **required**
- Il peut avoir une **valeur par défaut**
- Il peut avoir une **Alias**
- Et vous pouvez le **transformer**

```
// optional
counter = input<number>();
// Valeur par défaut = 0
counter = input<number>(0);
// Required
counter = input.required<number>();
```

```
counter = input(0, {
  alias: 'counter',
  transform: (value: number) => value * 100,
});

counter = input.required({
  alias: 'counter',
  transform: (value: number) => value * 100,
});
```

Signal Input

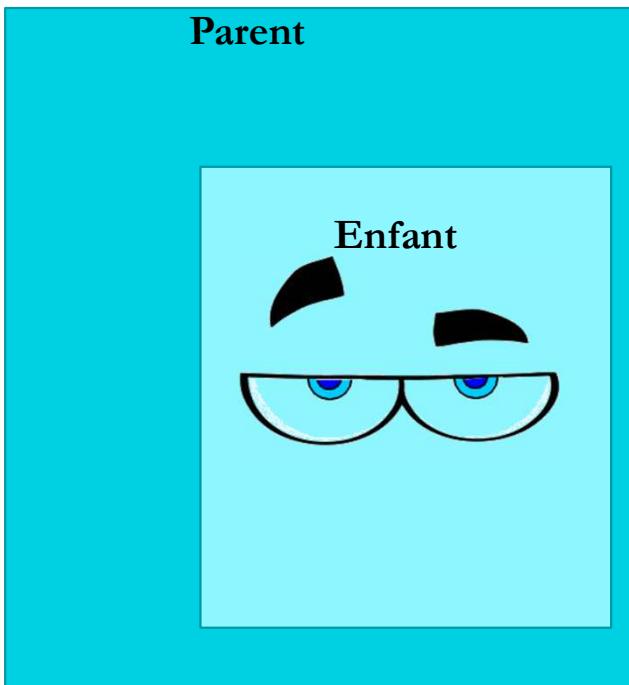


- Reprenez cet exemple en utilisant les Signal Input



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 transiter les informations.

Parent



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

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

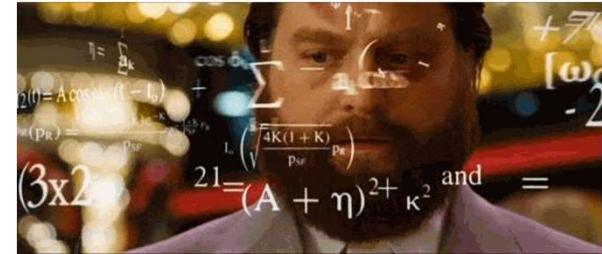
Enfant

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

Parent

Exercice

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



Signal Output

- L'API output() remplace directement le décorateur @Output() traditionnel.
- Le décorateur @Output n'est pas obsolète
- Angular a donc ajouté output() comme nouvelle façon de définir les sorties des composants dans Angular, d'une manière plus sûre et mieux intégrée à RxJ que l'approche traditionnelle @Output et EventEmitter.
- La syntaxe a été simplifiée.

```
<app-item (selectCv)="onSelectCv($event)" [cv]="cv" />
```

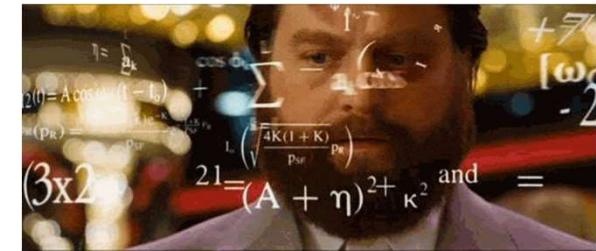
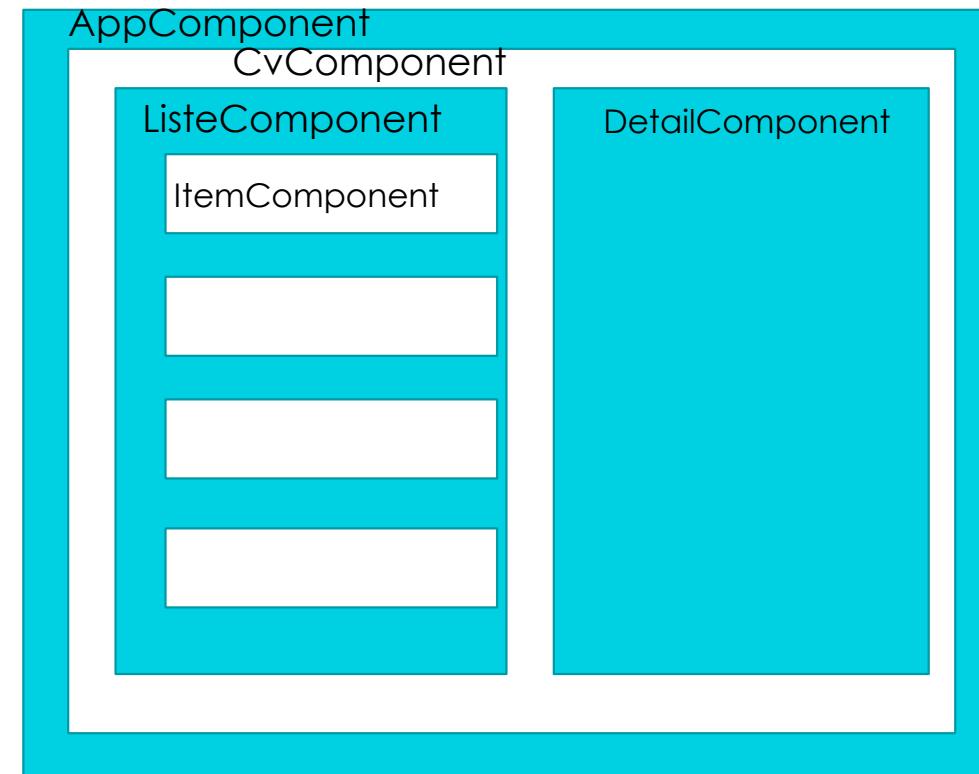
Pere

```
/* Sans les signal Output */
@Output() oldSelectCv = new EventEmitter<Cv>();
/* Avec les signal Output */
selectCv = output<Cv>();
onClick() {
  if (this.cv) {
    /* Cette partie du code reste la même */
    this.selectCv.emit(this.cv);
  }
}
```

Fils

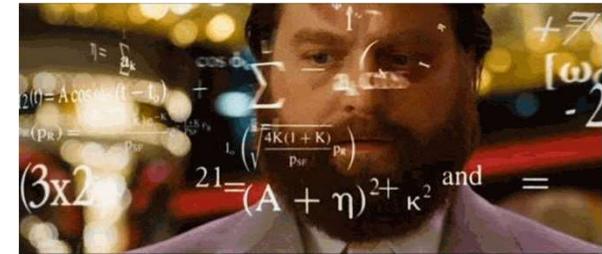
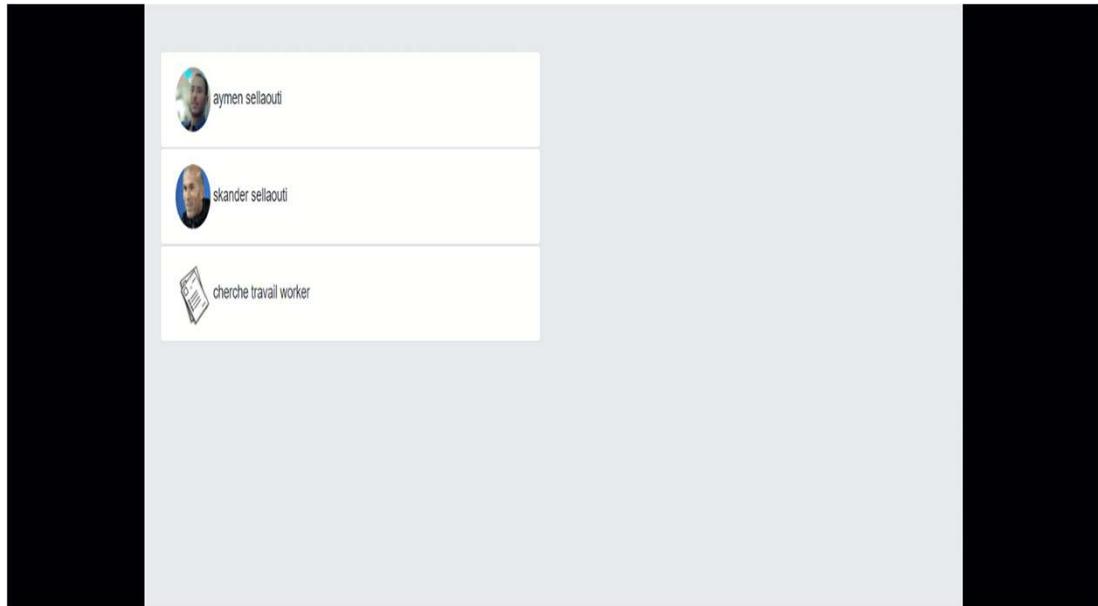
Exercice

- 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 apparaîtra 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.

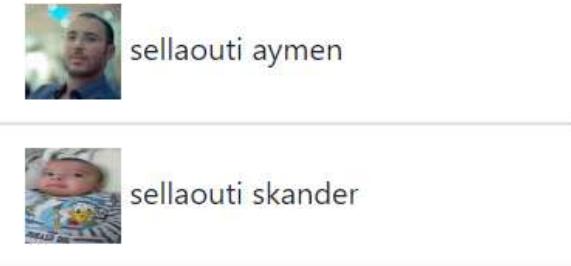
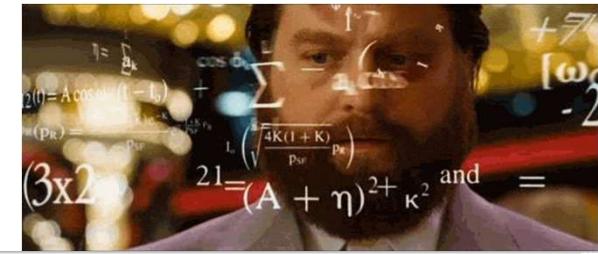
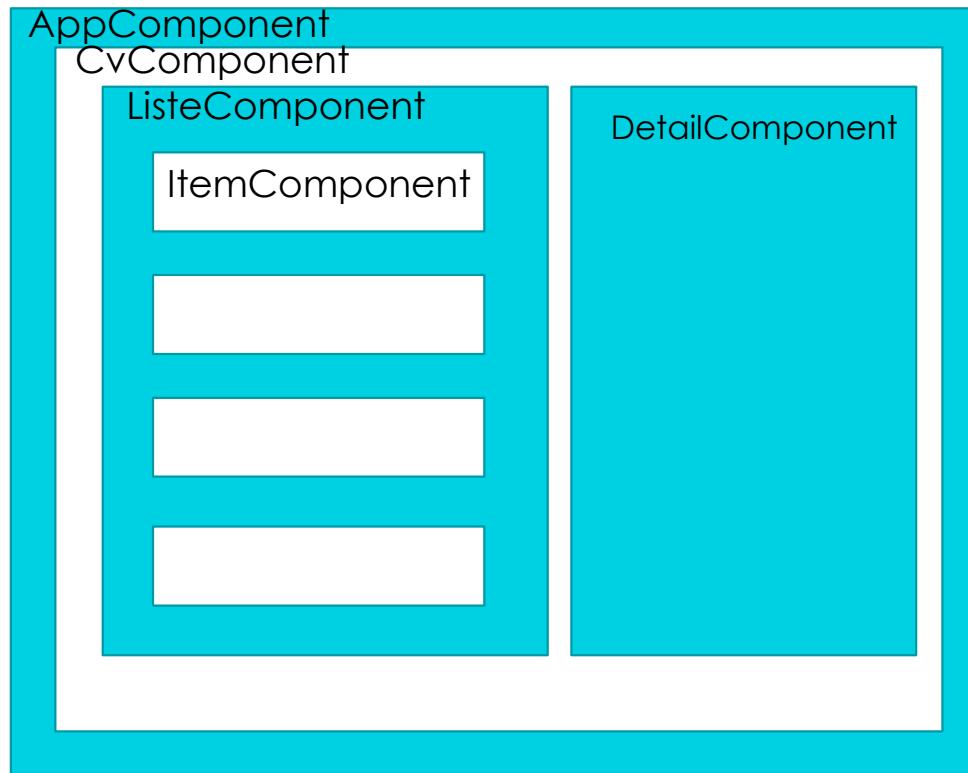


Exercice

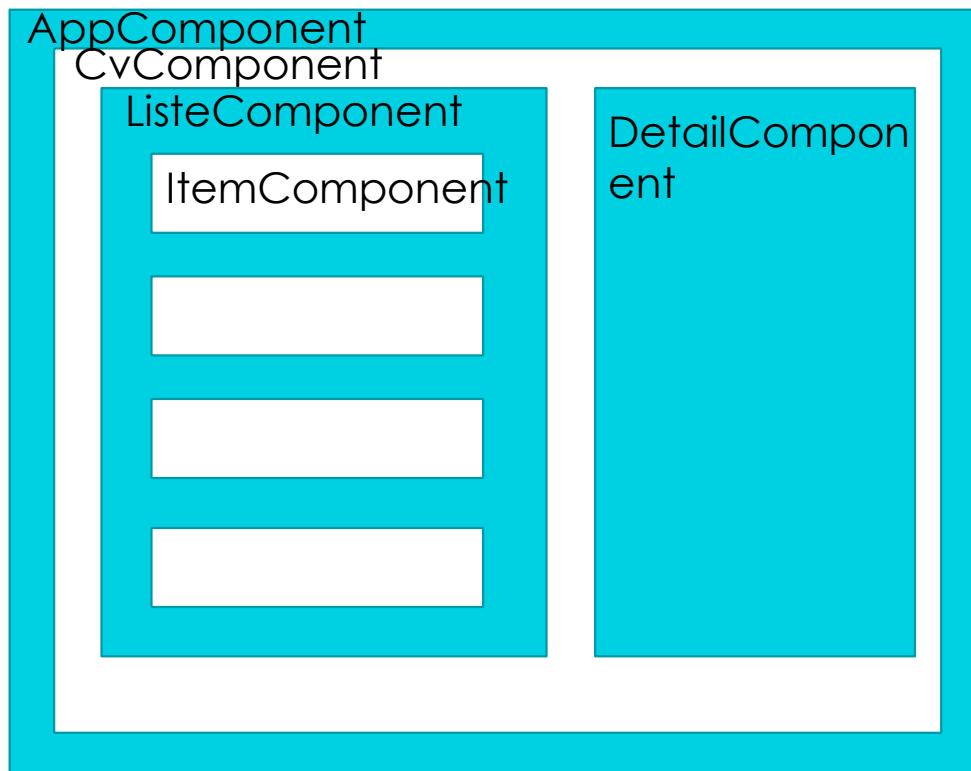
- Le but de cet exercice est de créer une mini plateforme de recrutement.
- La première étape est de créer la structure suivante avec une vue contenant deux parties :
 - Liste des Cvs inscrits
 - Détail du Cv qui apparaitra au click
- Il vous est demandé juste d'afficher un seul Cv et de lui afficher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



Exercice



Exercice




Au click sur le Cv les détails sont affichés

Auto Rotation

36

104

Exercice

Un cv est caractérisé par :

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

Aymen Sellaouti
Teacher

36

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



```
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 placée entre [] **[ngStyle]**
- Elle prend en paramètre un **attribut** représentant un **objet** décrivant le **style** à appliquer.
- Elle utilise le **property Binding**.

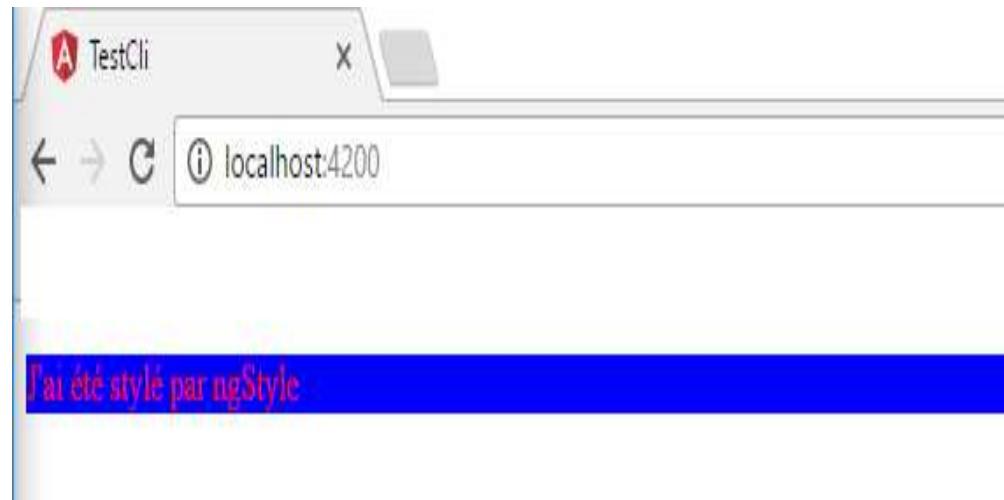
Les directives d'attribut (ngStyle)

```
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)

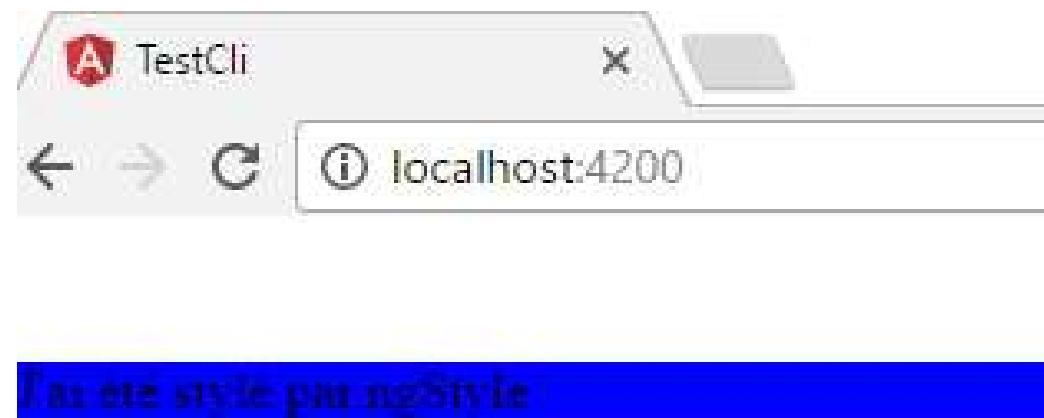


Les directives d'attribut (ngStyle)

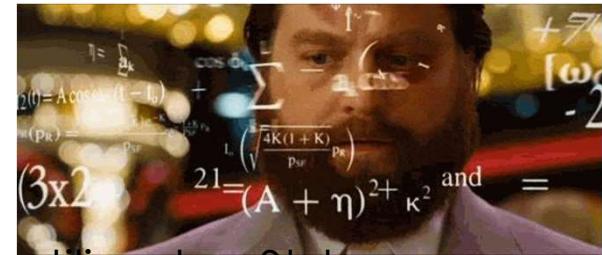
```
@Component({
  selector: 'app-root',
  template: `
    <direct-direct [myColor]="gray">J'ai
    été stylé par ngStyle</direct-direct>
  `,
  styles: [
    h1 { font-weight: normal; }
    p{color:yellow;background-color: red}
  ],
})
export class AppComponent {
```

```
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"
}
```

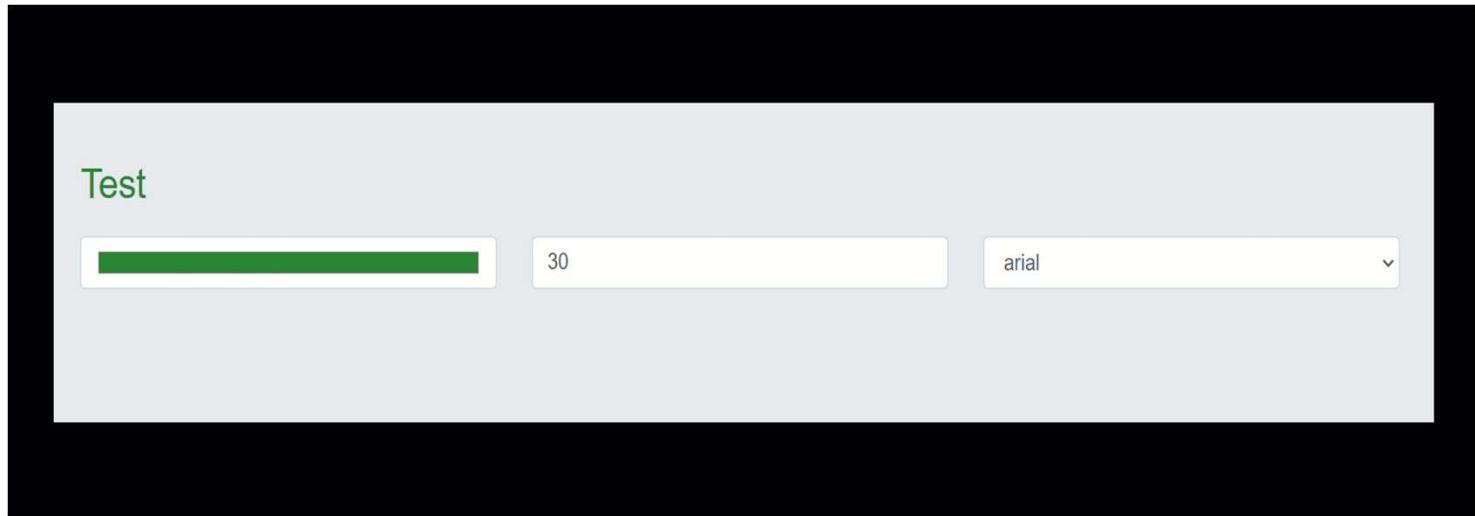
Les directives d'attribut (ngStyle)



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 mettre-y un ensemble de police. Lorsque le user sélectionne une police dans la liste, la police dans le paragraphe change.



Les directives d'attribut (ngClass)

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

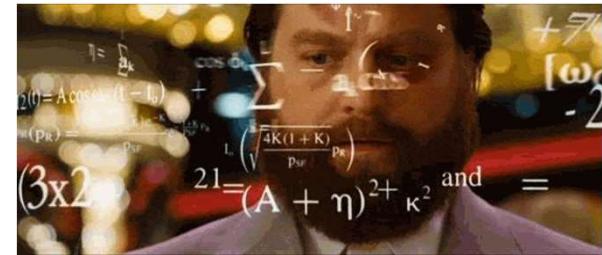
```
import {Component, Input} from '@angular/core';
@Component({
  selector: 'direct-direct',
  template: `
    <div ngClass="colorer arrierman" class="encadrer">
      test ngClass
    </div>
  `,
  styles: [
    .encadrer{ border: inset 3px black; }
    .colorer{ color: blueviolet; }
    .arrierman{background-color: 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



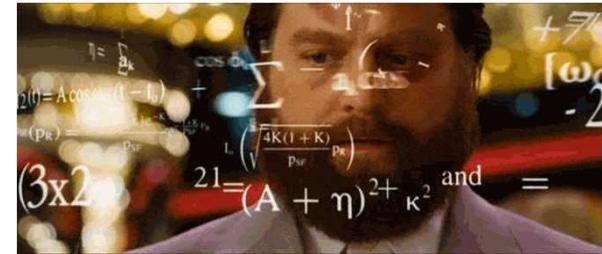
Customiser un attribut directif

- 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 `HostListener` il faut les importer du `core d'angular`

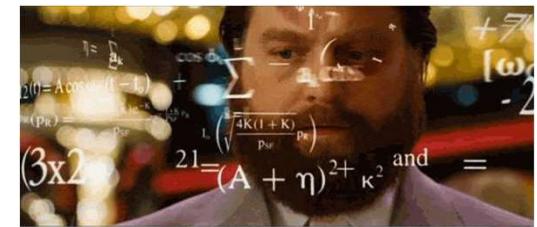
Exercice

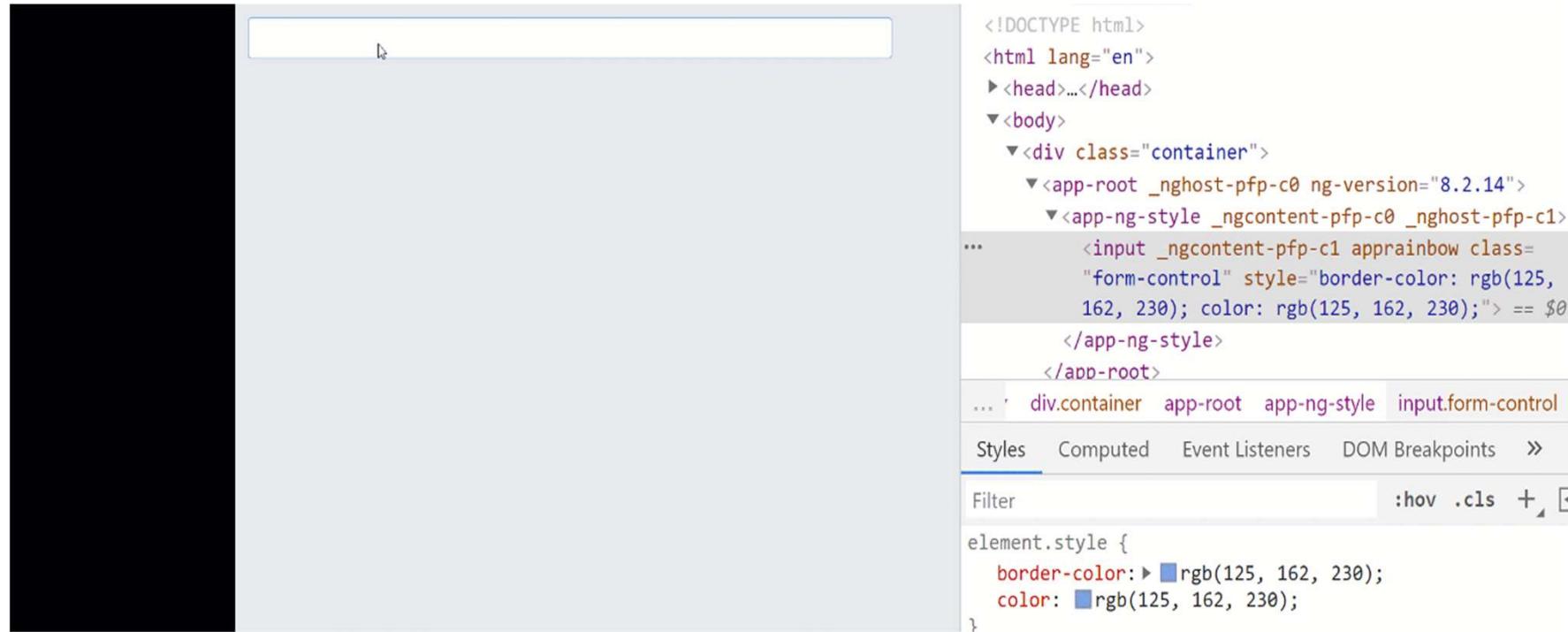
Un truc plus sympa on va créer un simulateur d'écriture arc en ciel.

- Créer une directive
- Créer un hostbinding sur la couleur et la couleur de la bordure.
- Créer un tableau de couleur dans votre directive.
- Faites 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





The screenshot shows the DOM structure of an input field within a container and app-root components. The input field has a class of "form-control". The styles tab is selected, showing the following CSS:

```

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

```

Customiser un attribut directif

- 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 **prefix ***.
- 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 pair
- odd : vrai si l'indice est impair

```
<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>
```

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 une pipe personnalisée

Qu'est-ce qu'une pipe

- Une 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 offertes par Angular et prêt à l'emploi.
- Vous pouvez créer vos propres pipes.



Avec le pipe uppercase :

Sans aucun pipe :

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

Syntaxe

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

Les pipes disponibles par défaut (Built-in pipes)

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

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

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

Paramétrer un pipe

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

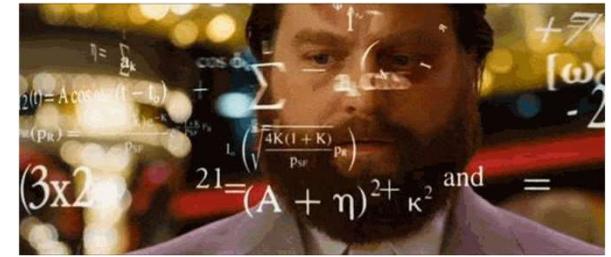
Pipe personnalisé

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

```
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'];  
}
```

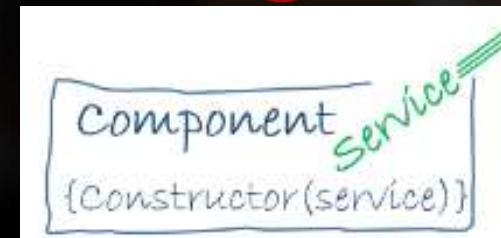
Exercice



- Créer une pipe appelée defaultImage qui retourne le nom d'une image par défaut que vous stockerez dans vos assets au cas où la valeur fournie aux pipes 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(){};
```

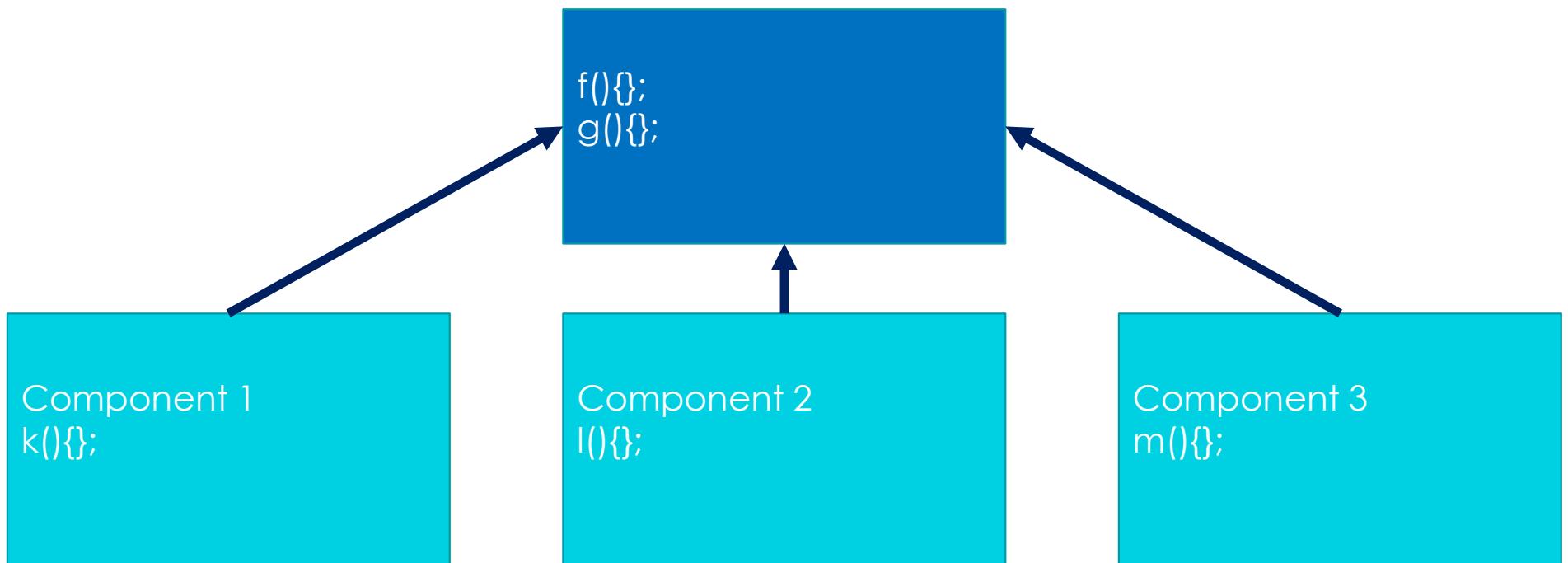
Component 3

```
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

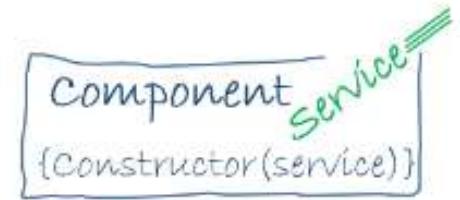
```
import { Injectable } from '@angular/core';

@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.

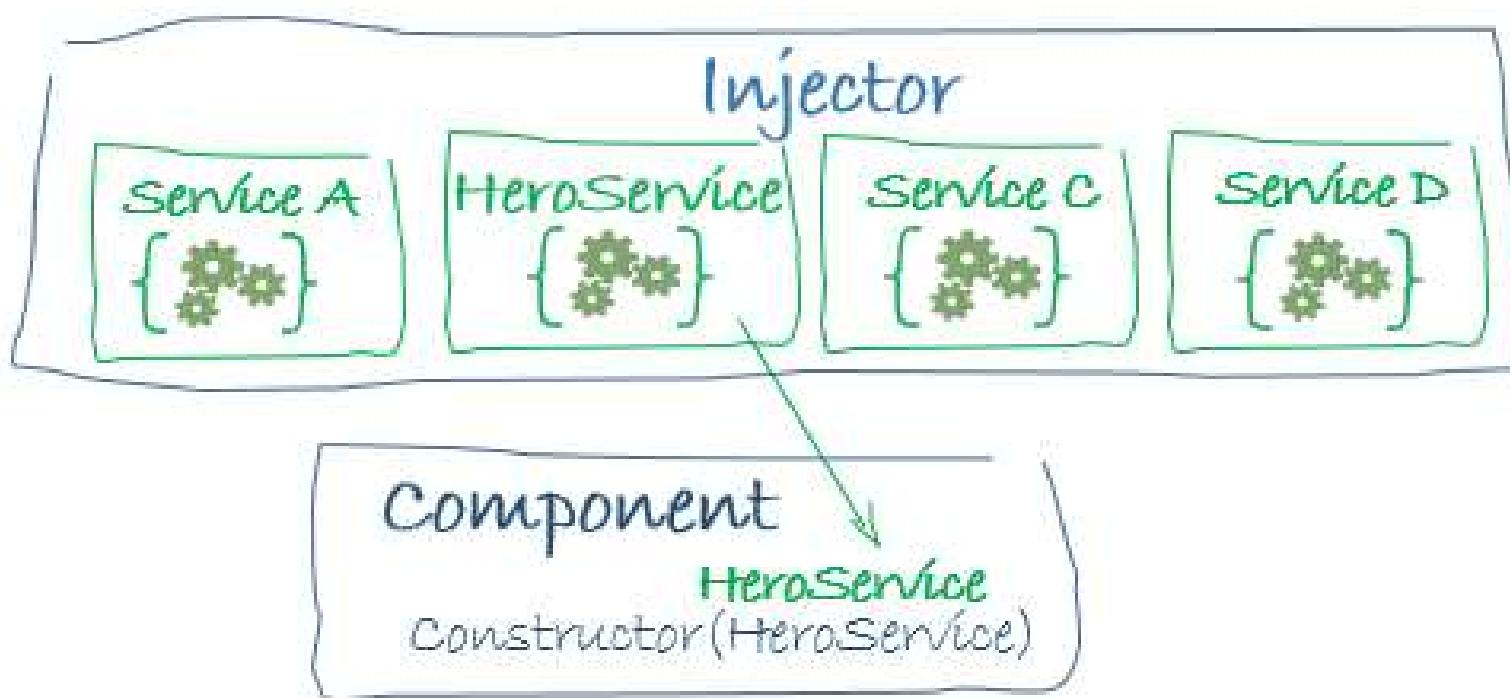
```
Classe A1{  
    Constructor(B b, C c)  
    ...  
}
```

```
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.

Moduless Application

Configurez l'injection de dépendance

- Afin de fournir un **provider** pour **toute l'application**, vous pouvez ajouter en **deuxième paramètre** de la **fonction bootstrapApplication**, un **objet d'options**.
- Cet objet contient une **clé providers** qui prend en paramètre un **tableau de providers**.
- Vous pouvez aussi **récupérer des providers offerts** par un **module** avec la fonction **importProvidersFrom(moduleCible)**.
- A partir de la **version 15**, vous avez **certaines fonctions spécifiques** pour les **modules** les **plus utilisés** comme le **http** avec sa fonction **provideHttpClient()**, ou **provideRouter(APP_ROUTES)**.

Modulless Application

Configurez l'injection de dépendance

```
bootstrapApplication(AppComponent, {  
  providers: [  
    // Importer d'un module externe et qui ne fournit pas de fonction provide  
    importProvidersFrom(  
      ToastrModule.forRoot()  
    ),  
    // Importer un service  
    TodoService,  
    // Importer en utilisant la fonction provide  
    provideAnimations(),  
  ],  
}).catch((err) => console.error(err));
```

Injection de dépendance (DI) Modular Application

```
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 {CvService} from "./cv.service";

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

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

@Injectable()
export class CvService {

  constructor() { }

}
```

Injection de dépendance (DI)

```
import { Component, OnInit } from '@angular/core';
import { Cv } from './cv';
import { CvService } from "../cv.service";

@Component({
  selector: 'app-cv',
  templateUrl: './cv.component.html',
  styleUrls: ['./cv.component.css'],
  providers: [CvService] // on peut aussi l'importer ici
})
export class CvComponent implements OnInit {
  selectedCv: Cv;

  constructor(private monPremierService: CvService) { }

  ngOnInit() {
  }
}
```

Chargement automatique du service

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

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

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

Avantage de l'utilisation du providedIn

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

`@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.

Signals et service

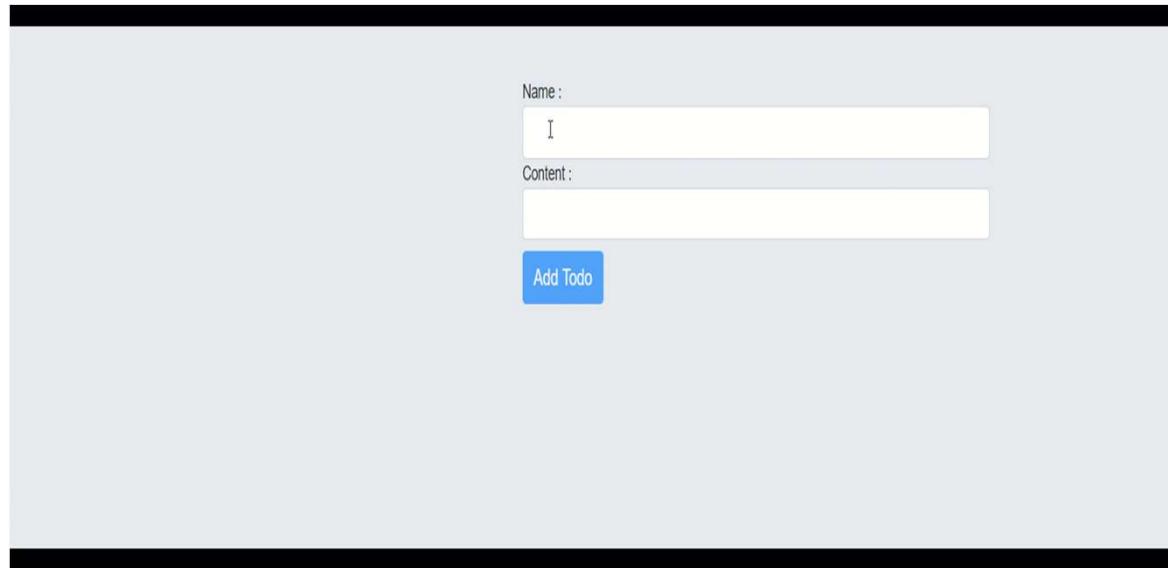
- Les signaux facilitent la réactivité dans Angular
- Nous pouvons voir un signal comme une source de vérité unique
- Il est donc logique de le partager et de permettre à toute personne intéressée d'y accéder.
- Cependant, pensez à les encapsuler pour garantir le control sur la modification.

```
private todosSignal = signal<Todo[]>([]);  
getTodosSignal(): Signal<Todo[]> {  
  return this.todosSignal.asReadOnly();  
}
```

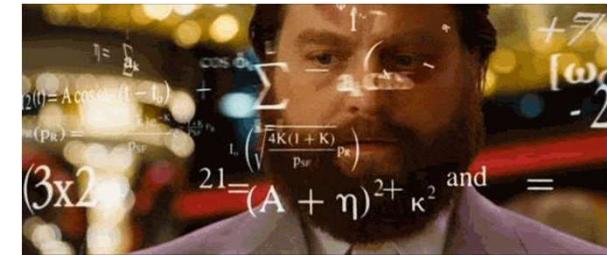
```
get todos(): Signal<Todo[]> {  
  return this.todosSignal.asReadOnly();  
}
```

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



The screenshot shows a simple web form for adding a new todo item. The form consists of two input fields: 'Name:' and 'Content:', both represented by white input boxes with placeholder text ('I' and an empty box respectively). Below the input fields is a blue button labeled 'Add Todo'. The entire form is set against a light gray background.

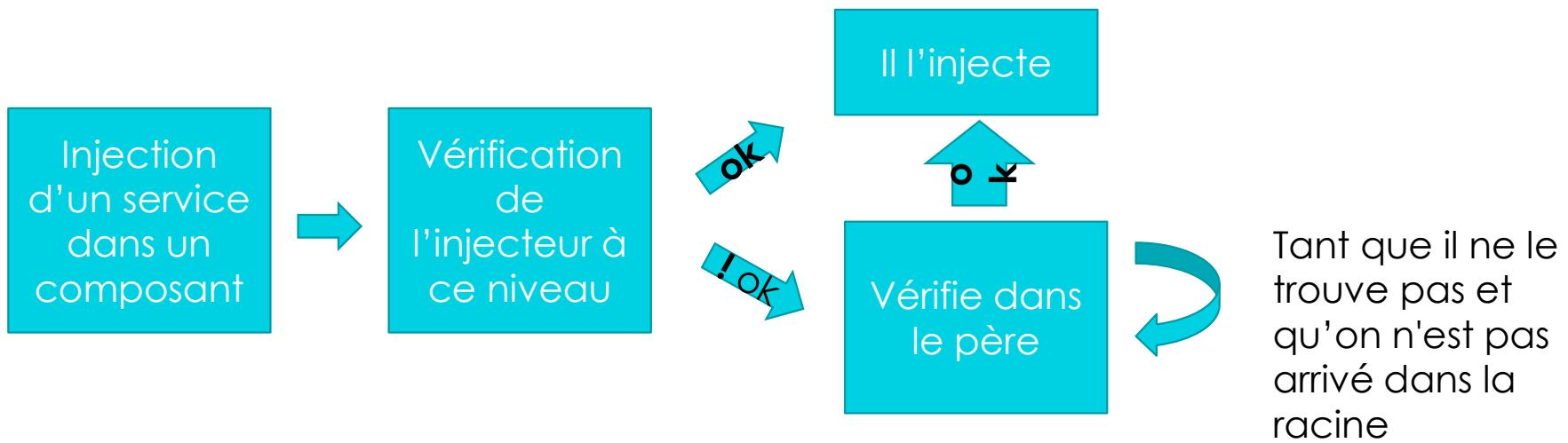


Exemple

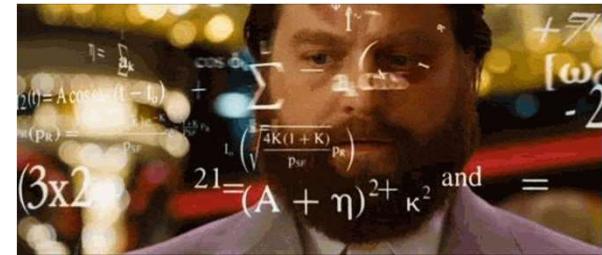
```
import { Injectable } from '@angular/core';
@Injectable()
export class LoggerService {
  constructor() { }
  Logs: string[]=[];
  log(message:string) {
    this.Logs.push(message); console.log(message);
  }
  info(message:string) {
    this.Logs.push(message); console.info(message);
  }
  debug(message:string)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

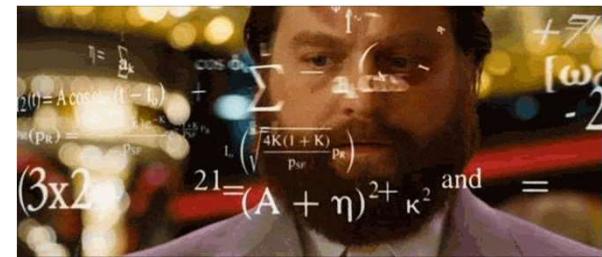


Exercice



- 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

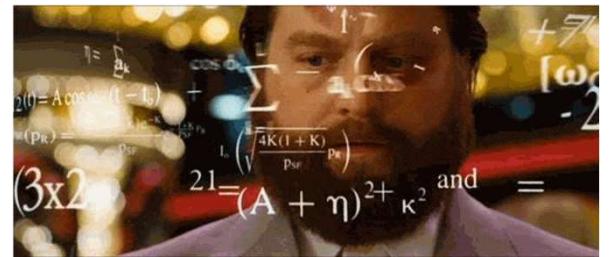
35 Projects

[Embaucher](#)

Liste des cvs sélectionnés pour embauche

aymen sellaouti

Exercice



A screenshot of a web browser interface. At the top, there is a navigation bar with various links: 'Applications', 'template html 5 res...', 'enseignement', 'utilities', 'Dashboard sales ...', 'AllDebrid: Accès au ...', 'décès', 'Site de collaboration', 'ser ram', 'balls selection', 'Robin des Droits - L...', and 'Fiche Google Play S...'. Below the navigation bar, there is a list of users:

- aymen sellaouti
- skander sellaouti
- cherche travail worker

At the bottom of the page, there is a search bar with the placeholder text 'Rechercher'.

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 ?????

Modulless Application

Création d'un système de Routing

- Si vous utilisez une application standalone, la configuration change un peu.
- Vous n'avez plus à définir un module. Il suffit de provider vos route avec la fonction **provideRouter** au niveau de la fonction **bootstrapApplication**

```
bootstrapApplication(AppComponent, appConfig)
  .catch((err) => console.error(err));
```

```
export const routes: Routes = [
  { path: '', component: HomeComponent },
]
```

```
export const appConfig: ApplicationConfig = {
  providers: [
    provideRouter(routes)
  ]
}
```

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
 1. import { **RouterModule**, **Routes** } from '@angular/router';
 2. Le **RouterModule** va permettre de configurer les routes dans votre projet
 3. 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-
BVYiISIfE1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+P
mSTsz/K68vbxEjh4u"
  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>

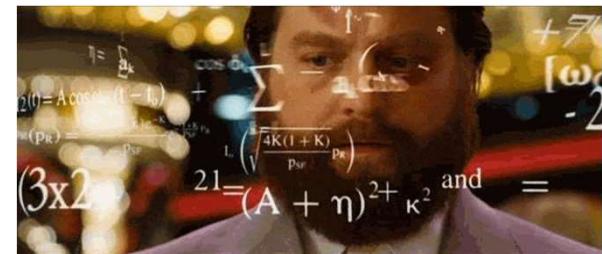
<div class="container">
  <router-outlet></router-outlet>
</div>
```

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

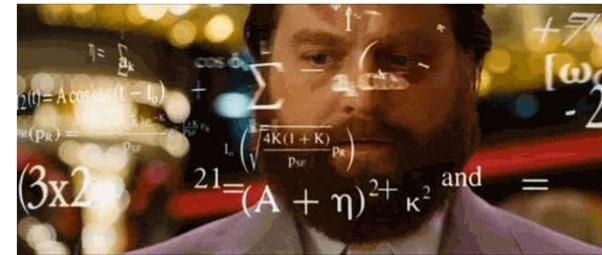
- L'idée intuitive pour déclencher une route est d'utiliser la balise **a** et son attribut **href**. Est-ce que ça risque de poser un problème ?
- L'utilisation de `<a href >` va déclencher le **chargement** de la page ce qui est inconcevable pour une **SPA**.
- La solution proposée par le router d'Angular 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

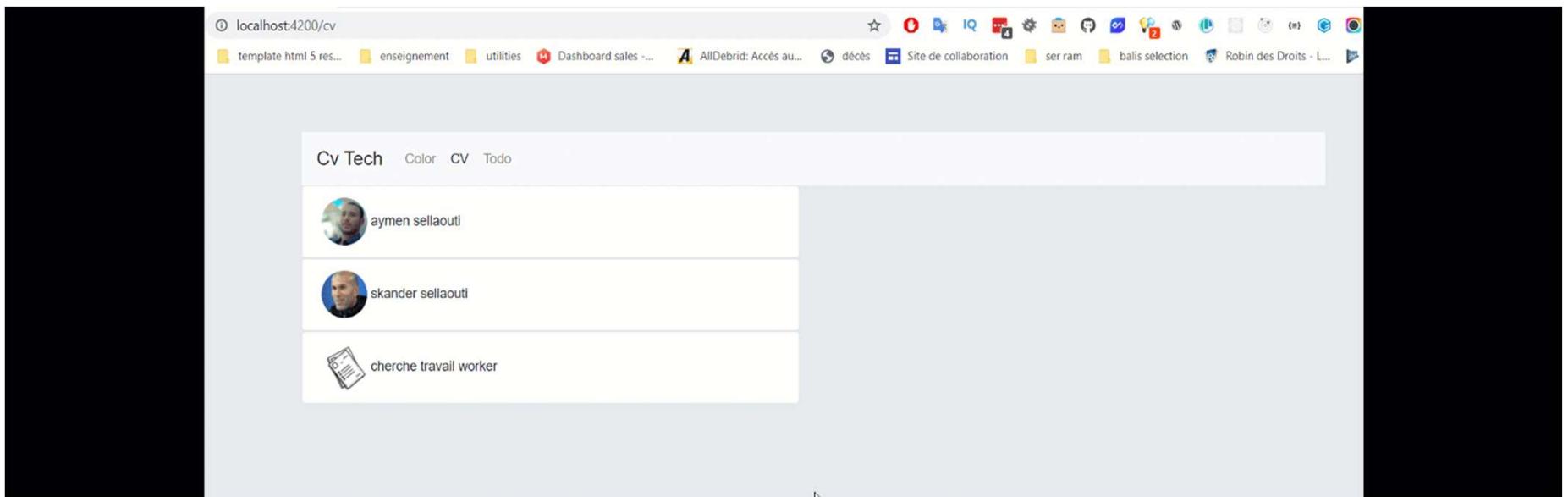
- **routerLinkActive="active"** va associer la classe active à l'uri cible ainsi qu'à tous ses ancêtres.
- Par exemple si on a l'uri 'cv/liste' la classe active sera ajouté à cet uri ainsi qu'à l'uri 'cv' et ''.
- Pour identifier uniquement l'uri cible, ajouter la directive suivante :

[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'un root au niveau d'un composant on doit procéder comme suit :
 1. Importer **ActivatedRoute** qui nous permettra de récupérer les paramètres du 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')
```

Récupérer les paramètres d'une route via `@Input` et `input`

- A partir d'angular 16 nous pouvons directement mapper les paramètres de votre route via `@Input` et la fonction `input`
- Il faut d'abord spécifier à Angular que vous voulez le faire en allant dans votre `appConfig` et en ajoutant l'option **`withComponentInputBinding`**

```
provideRouter(  
    routes,  
    // This allow us to get all router data, param, resolver from Input  
    // without injecting acr  
    withComponentInputBinding()  
),
```

Récupérer les paramètres d'une route via `@Input` et `input`

- Ensuite récupérer le paramètre en spécifiant le nom de la propriété (ou de l'alias) avec le nom du paramètre

```
{  
  path: 'inputParam/:id',  
  component: RouteParamInputComponent,  
},
```

```
id = input.required();
```

```
userId = input.required({alias:'id'});
```

```
@Input() id = '';
```

```
@Input('id') userId = '';
```

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

```
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</a>
```

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')
```

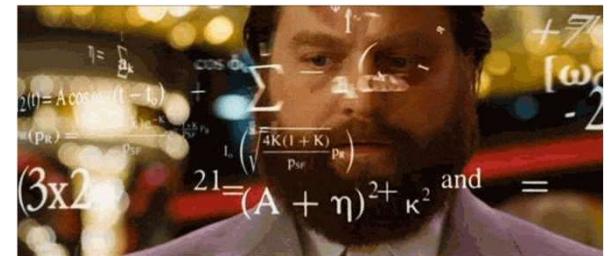
La route joker

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

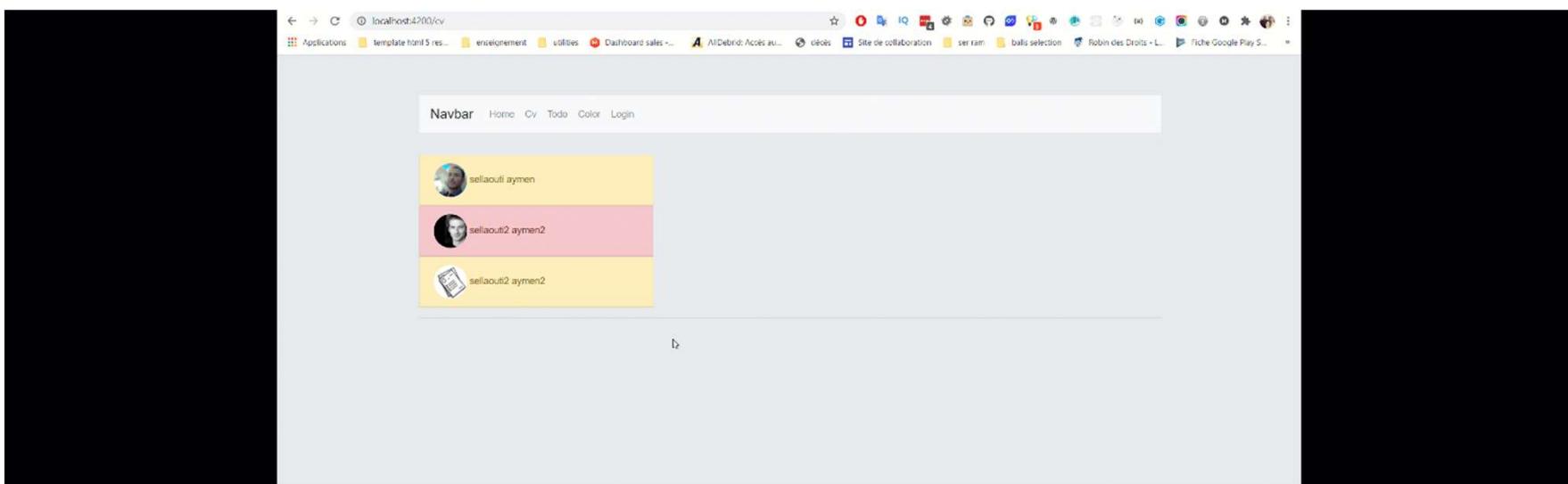
Exemple

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



Angular Form

Aymen sellaouti



Approche de gestion de FORM

1. Approche basée Template
2. Approche réactive

Objectifs

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 contrôles du formulaire

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

Approche basée Template NgForm

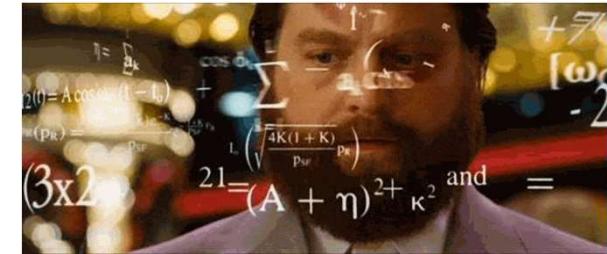
En détectant le formulaire, Angular décore les différents éléments du formulaire avec des classes qui informe sur leur état :

- **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

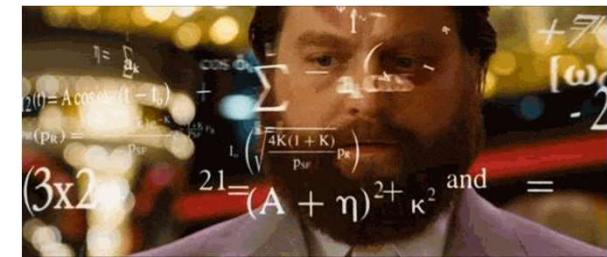
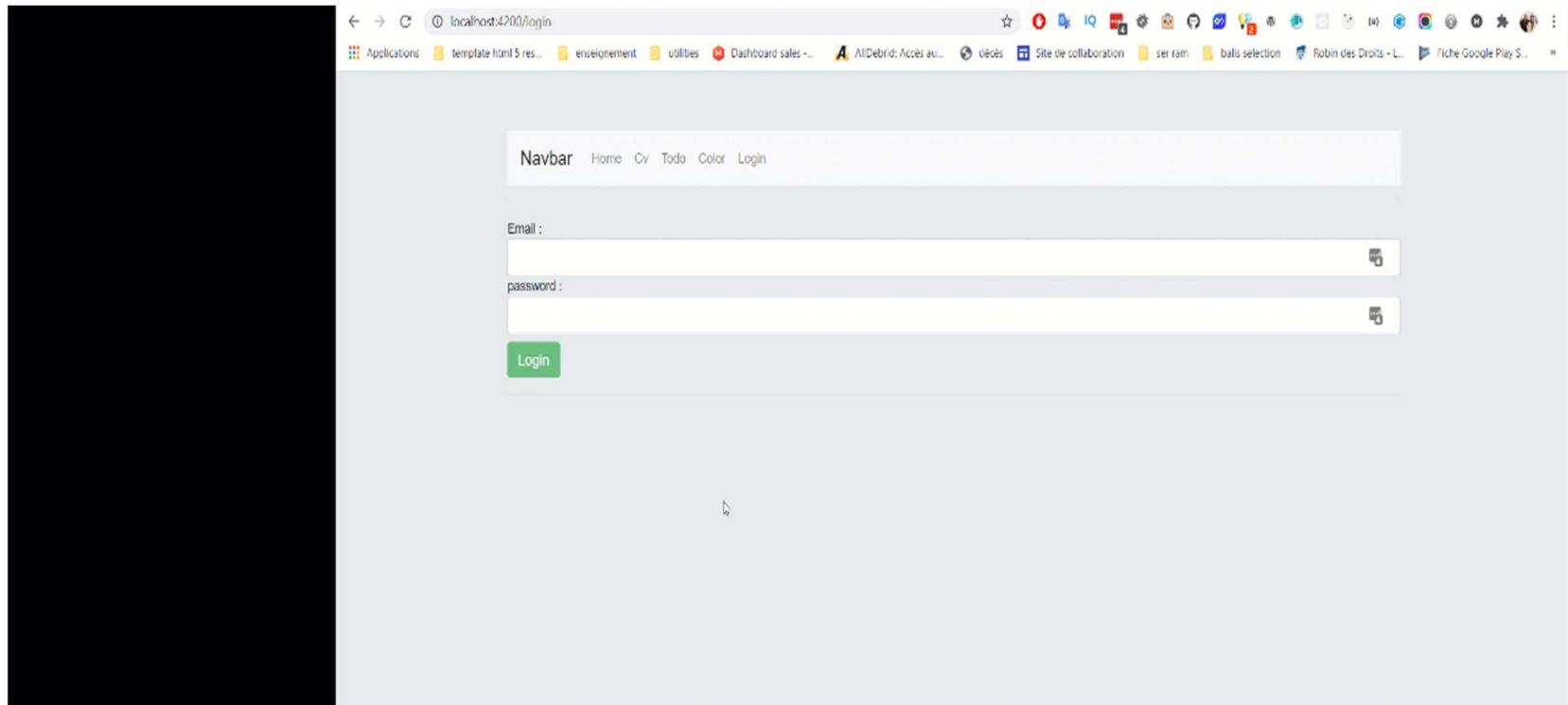


Exercice

- 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

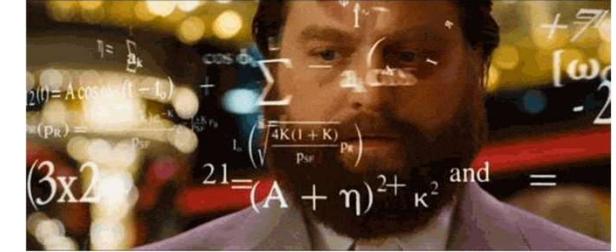


Approche basée Template

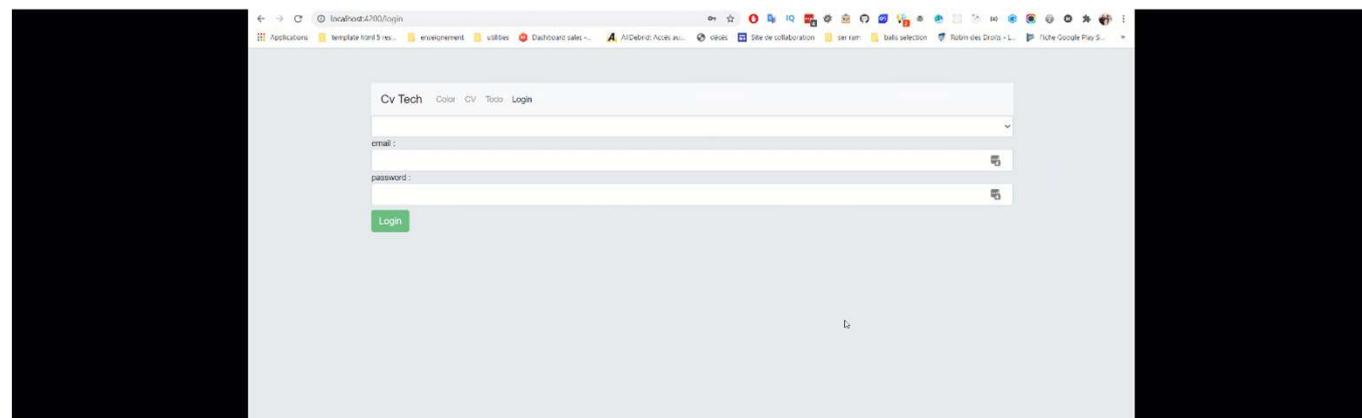
Accéder aux propriétés d'un champ (contrôle) du formulaire

- 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 apparaître que si l'utilisateur accède ou modifie le champ email.
- Ajouter un champ d'erreur pour signaler l'erreur à l'utilisateur.

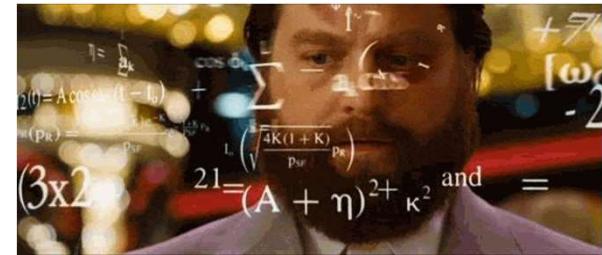


Approche basée Template Associer des valeurs par défaut aux champs

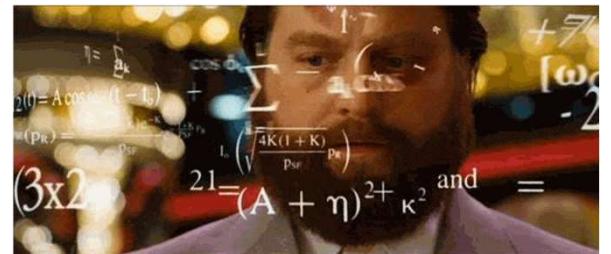
- Pour associer des valeurs par défaut aux champs d'un formulaire associé à Angular il faut le faire à partir du composant.
- Afin de gérer les valeurs du formulaire à partir du composant il faut du binding.
- Au lieu d'avoir juste la primitive ngModel associée au contrôle d'un élément on ajoute le **property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

Exercice

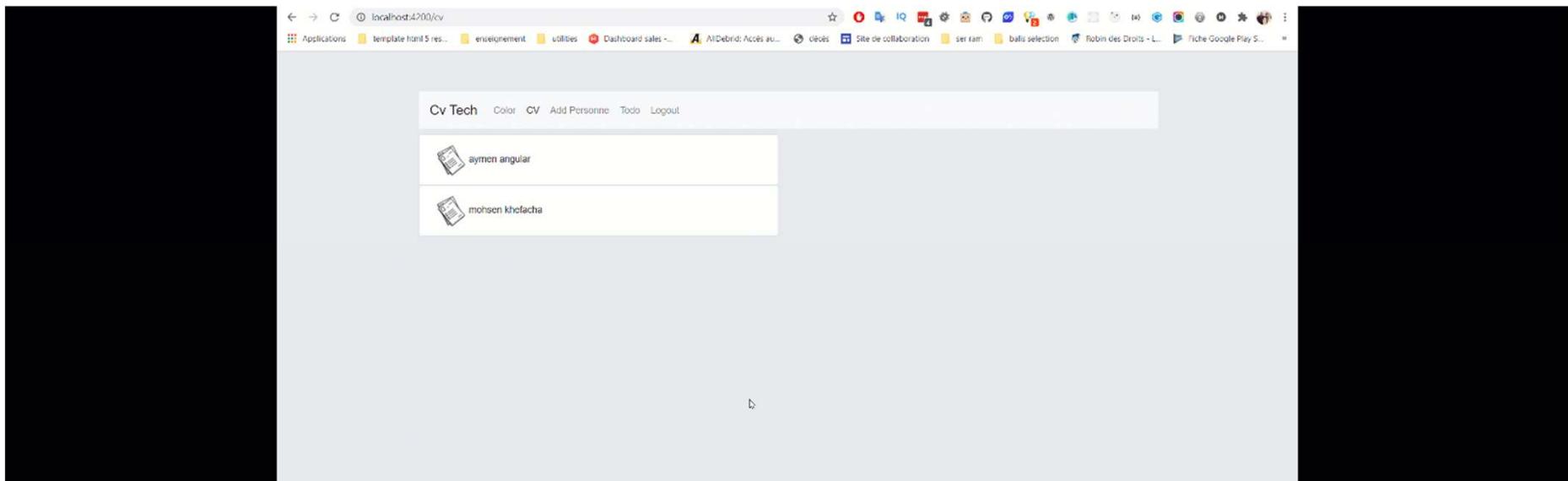
- Ajouter la valeur par défaut « myUserName » au champ username.



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



HTTP

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

Le 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 :
 - résolve en cas de succès
 - rejet 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

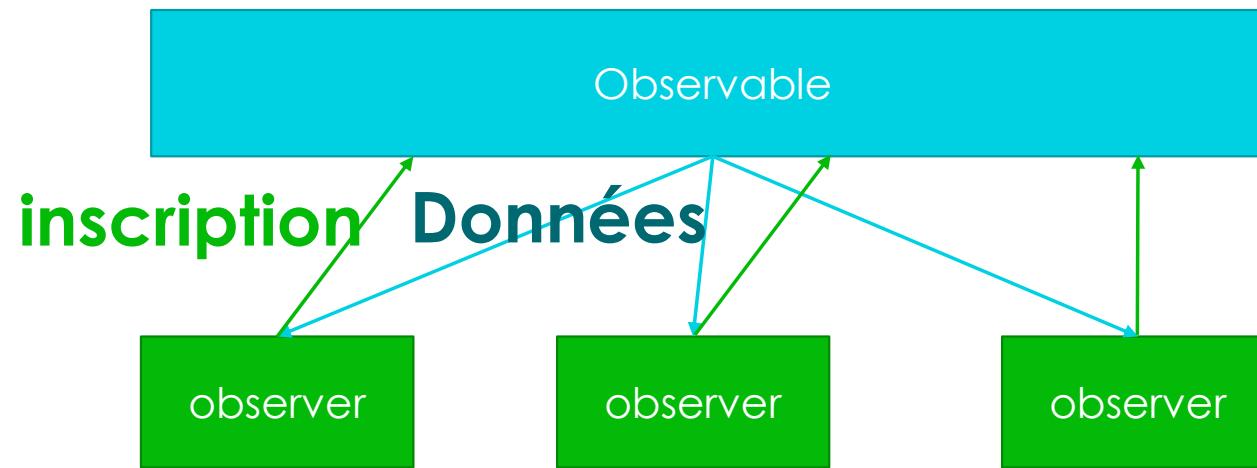
1. Nouvelle manière d'appréhender les appels asynchrones
2. Programmation avec des flux de données asynchrones

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

Le pattern « Observer »

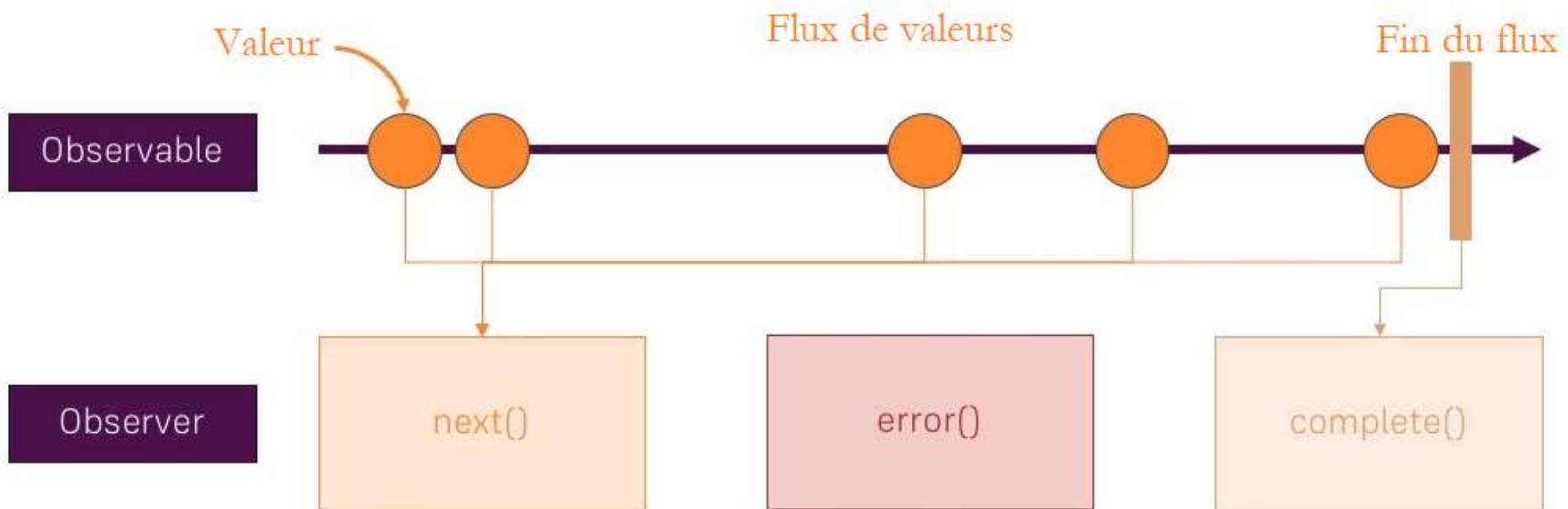
- Le patron de conception **Observable** permet à un objet de garder la trace d'autres objets, intéressés par l'état de ce dernier.
- Il définit une relation entre objets de type un-à-plusieurs.
- Lorsque l'état de cet objet **change**, il **notifie** ces **observateurs**.

Observables, Observers et subscriptions



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);
});
```

asyncPipe

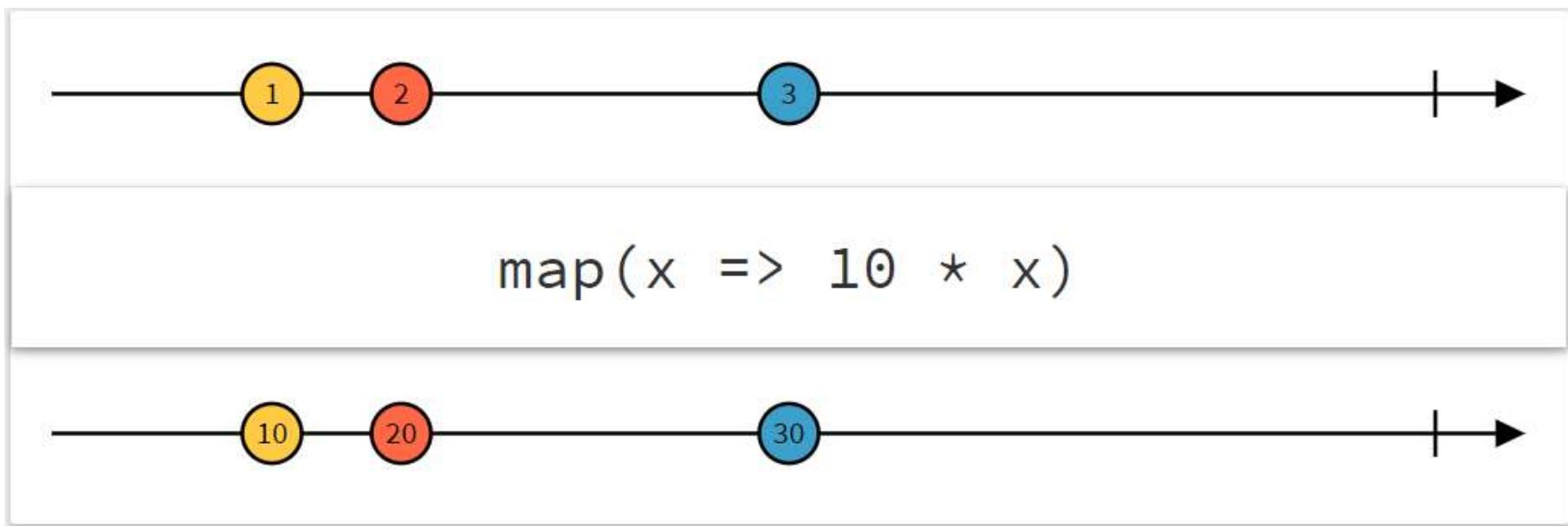
- asyncPipe est un pipe qui permet d'afficher directement un observable.
- {{ valeurSourceAsynchrone | **async** }}
- L'asyncPipe s'inscrit automatiquement à l'observable et affiche le dernier résultat envoyé.
- Quand le composant est détruit l'asyncPipe se désinscrit automatiquement de l'observable.

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.

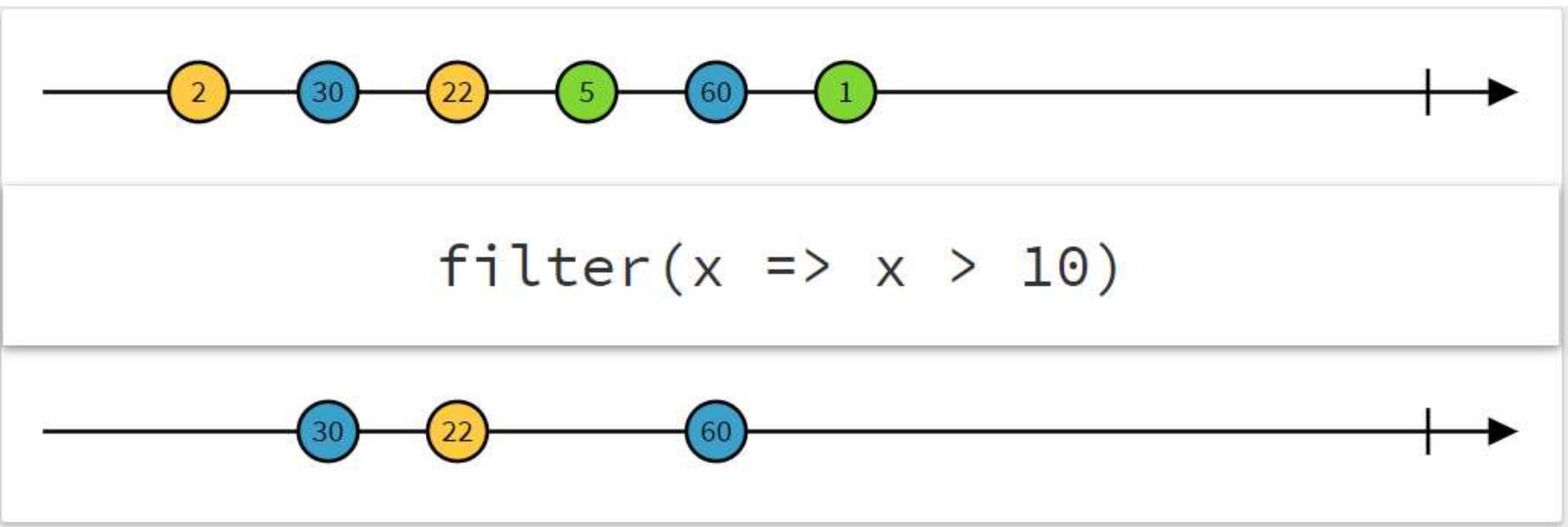
Quelques opérateurs utiles de l'Observable

- **map**

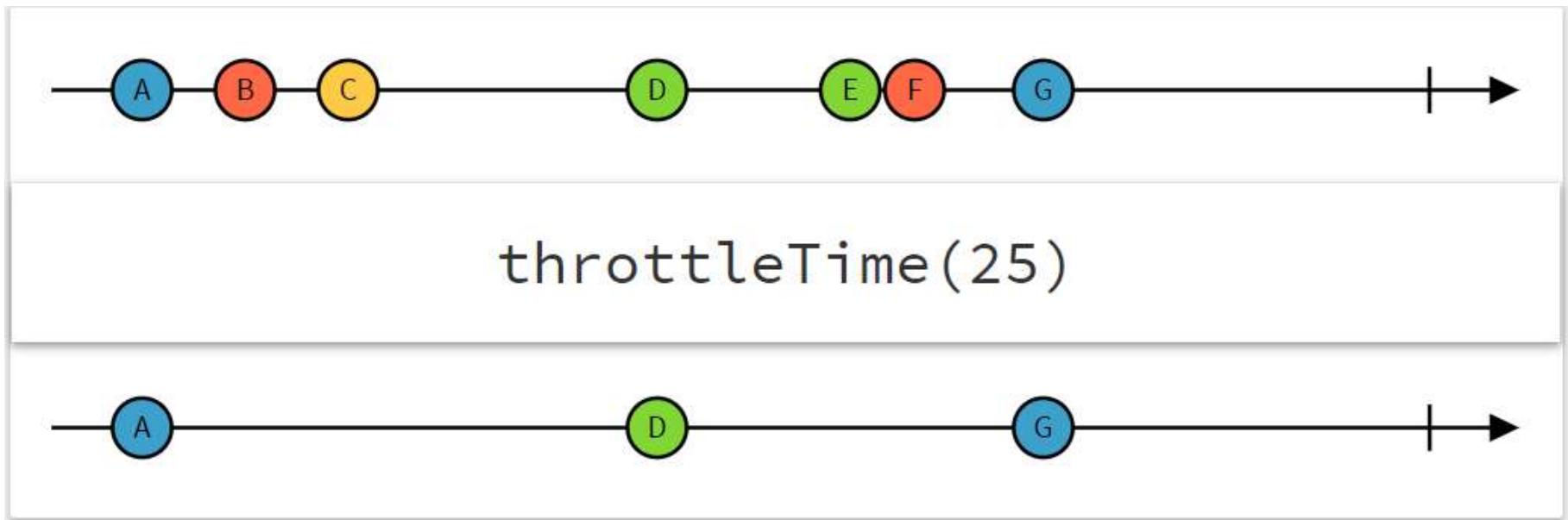


Quelques opérateurs utiles de l'Observable

- **filter**



Quelques opérateurs utiles de l'Observable

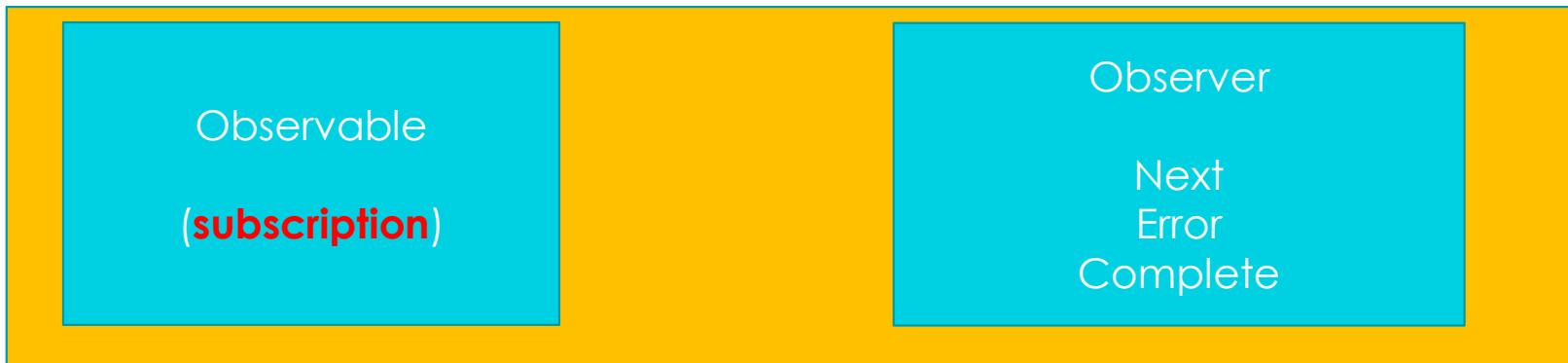


Quelques opérateurs utiles de l'Observable

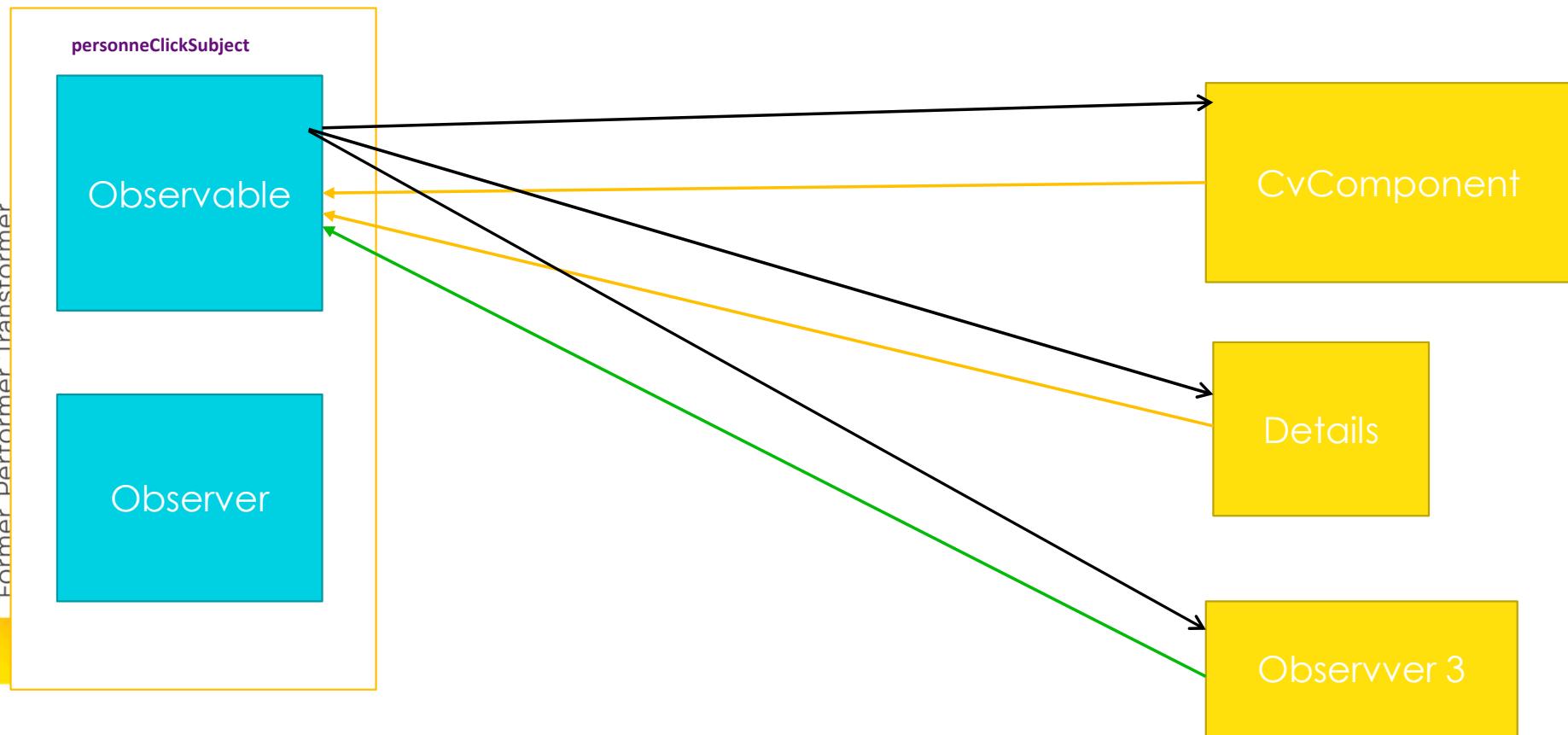
- <https://angular.io/guide/rx-library>
- <http://reactivex.io/rxjs/manual/overview.html#operators>
 - <http://rxmarbles.com/>

Les subjects

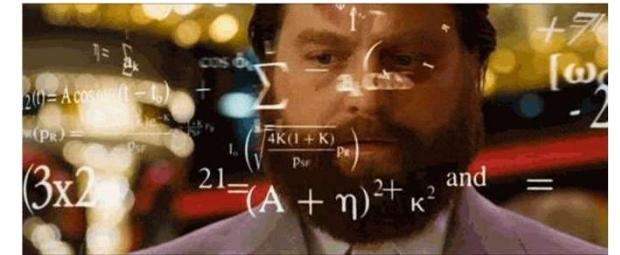
- Un **subject** est un type particulier d'observable. En effet Un **subject** est en même temps un **observable** et un **observer**, il possède donc les méthodes next, error et complete.
- Pour **broadcaster** une nouvelle valeur, il suffit d'appeler la méthode **next**, et elle sera diffusé aux Observateurs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

Moduless Application Installation de HTTP

- Le module permettant la consommation d'API externe s'appelle le HTTP MODULE.
- Afin de provider les services qu'il vous offre, utiliser la méthode **provideHttpClient()**, dans le tableau de provider de votre application.

```
bootstrapApplication(AppComponent, {  
  providers: [  
    provideHttpClient(),  
  ]).catch((err) => console.error(err));
```

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 **Observable**.
- 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.

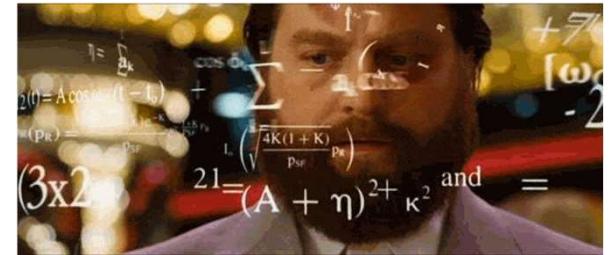
Interagir avec une API POST Request

```
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

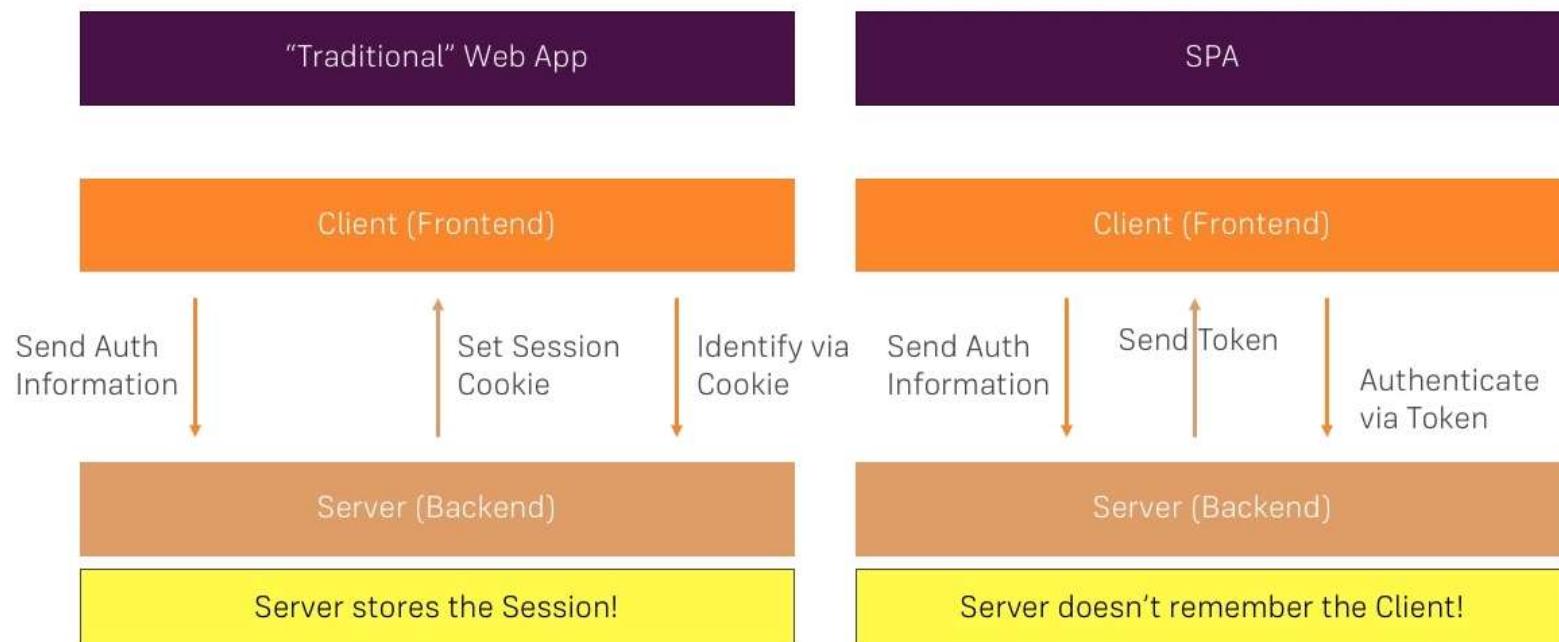
- Afin d'ajouter des paramètres à vos requêtes, le HttpClient vous offre la classe HttpParams.
- Cette classe est une classe immutable (read Only).
- Elle propose une panoplie de méthode helpers permettant de la manipuler.
- `set(clé,valeur)` permet d'ajouter des headers. Elle écrase les anciennes valeurs.
- `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 s'est un sous cas de la sécurisation de vos routes.
- Angular a pris en considération ce cas en fournissant un mécanisme via l'utilisation des **Guard**.

Guard

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

Guard / canActivate

- Afin d'utiliser le guard `canActivate` (de même pour les autres), vous devez créer un classe qui implémente l'interface `CanActivate` et donc qui doit implémenter la méthode `canActivate` de sorte qu'elle retourne un booléen permettant ainsi l'accès ou non à la route cible.
 - 1
- Vous devez ensuite ajouter cette classe dans le provider.
 - 2
- Finalement pour l'appliquer à une route, ajouter la dans la propriété `canActivate`. Cette propriété prend un tableau de guard. Elle ne laissera l'accès à la route qu'esi la totalité des guard retourne true.
 - 3
- **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

1

- A partir d'Angular 14 et la possibilité d'utiliser la fonction inject dans tous les contextes d'injection, Angular préconise les fonctionnels Guards.

```
export const myGuard: CanActivateFn = (route, state) => {  
  return true;  
};
```



Guard / canActivate

2

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

App.module.ts

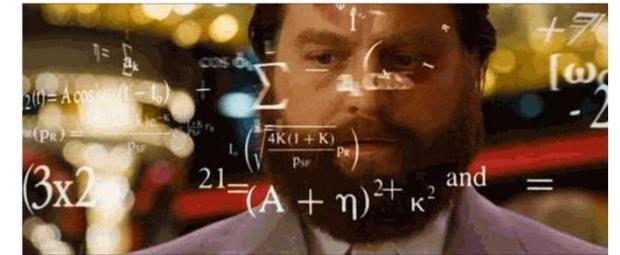


Guard / canActivate

3

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

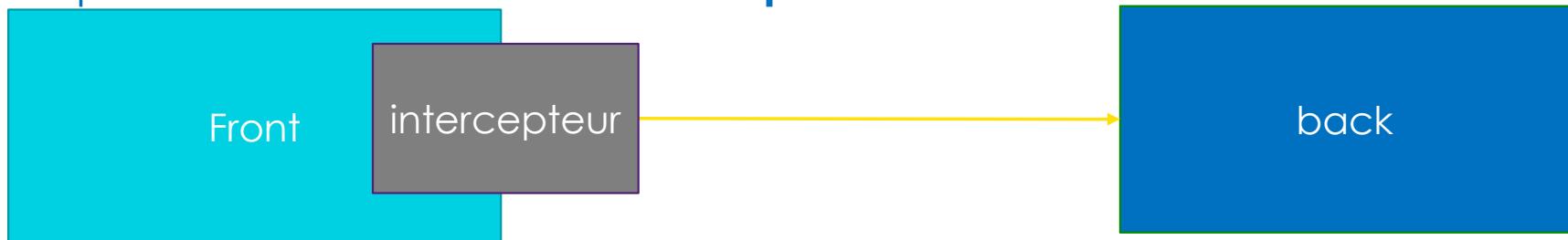
Exercice



- 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.

```
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 créer contenant votre projet
- Pour tester localement votre projet, télécharger un serveur HTTP virtuel avec la commande suivante :

`Npm install http-server -g`

- Lancer maintenant votre projet à l'aide de cette commande :
`http-server dist/NomDeVotreProjet`

Merci pour votre attention