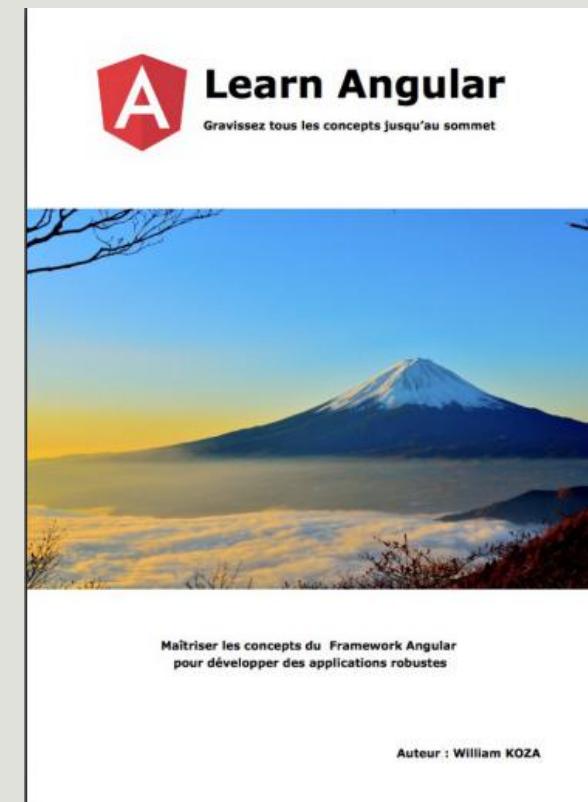
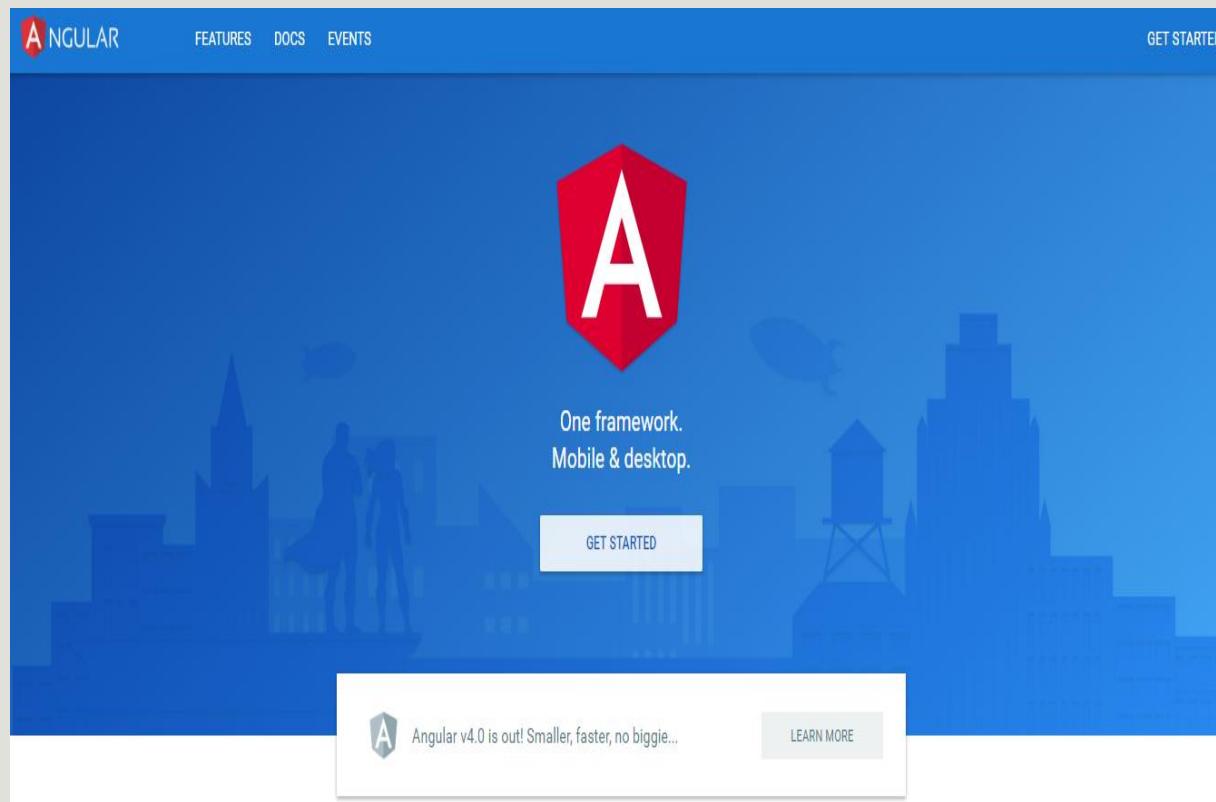


Angular Introduction

AYMEN SELLAOUTI

Références



Plan du Cours

- 1. Introduction
- 2. Les composants
- 3. Les composants Standalone
- 4. Le Binding
- 5. Le nouveau control Flow
- 6. Interaction entre composants
- 7. Directives
- 8. Pipes
- 9. Injection de dépendances
- 10. LinkedSignal
- 11. Routing
- 12. Template Driven Form
- 13. HTTP
- 14. RxJs Deep Dive
- 15. Resource
- 16. Contexte de réactivité et DestroyRef
- 17. Reactive Form
- 18. Lazy Loading
- 19. Change Detection et Patterns d'optimisation

PRÉ-REQUIS

- ▶ HTML 5 / CSS 3 / Bootstrap



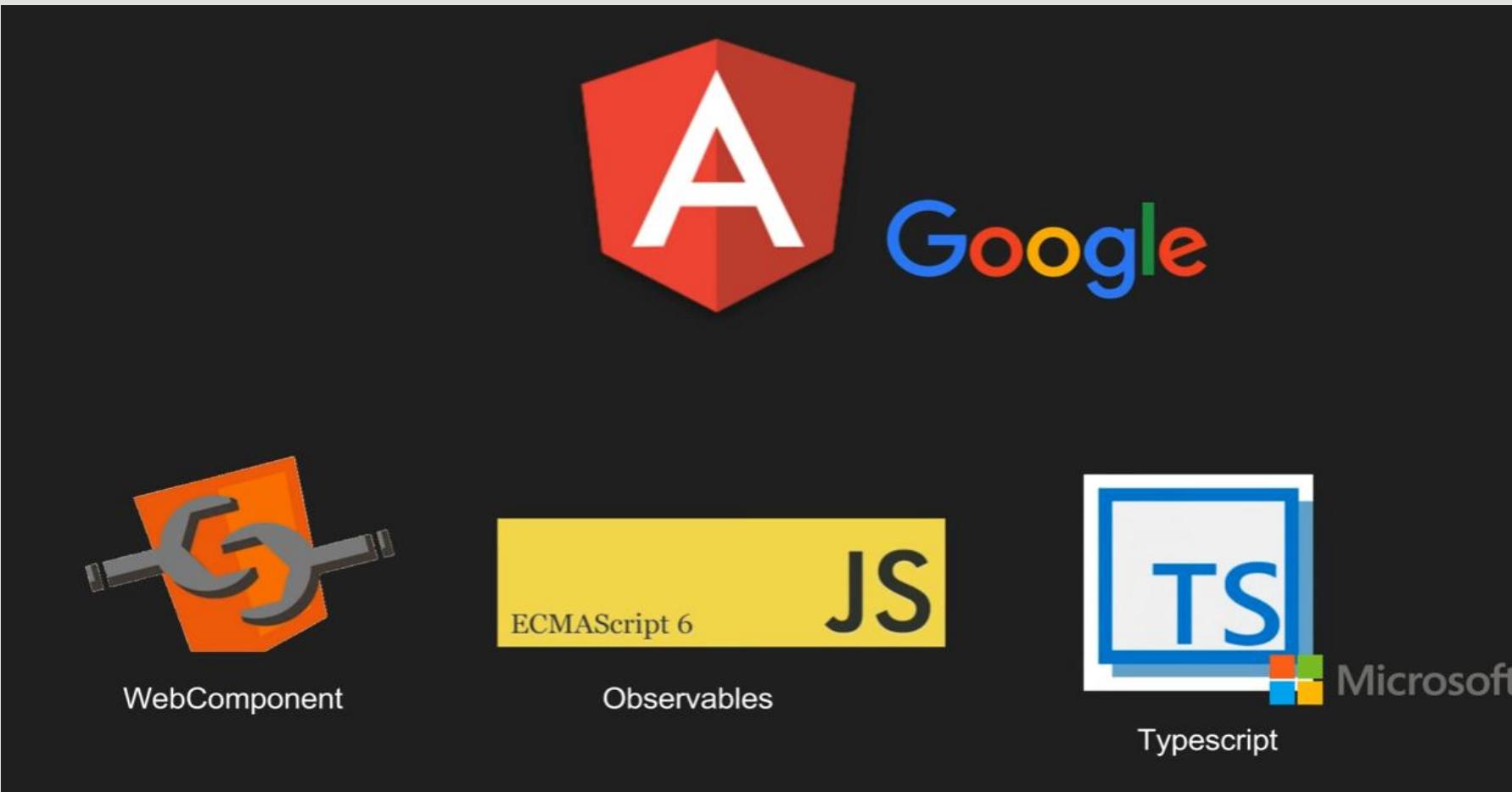
- ▶ JavaScript



- ▶ Programmation Orientée Objet

POO

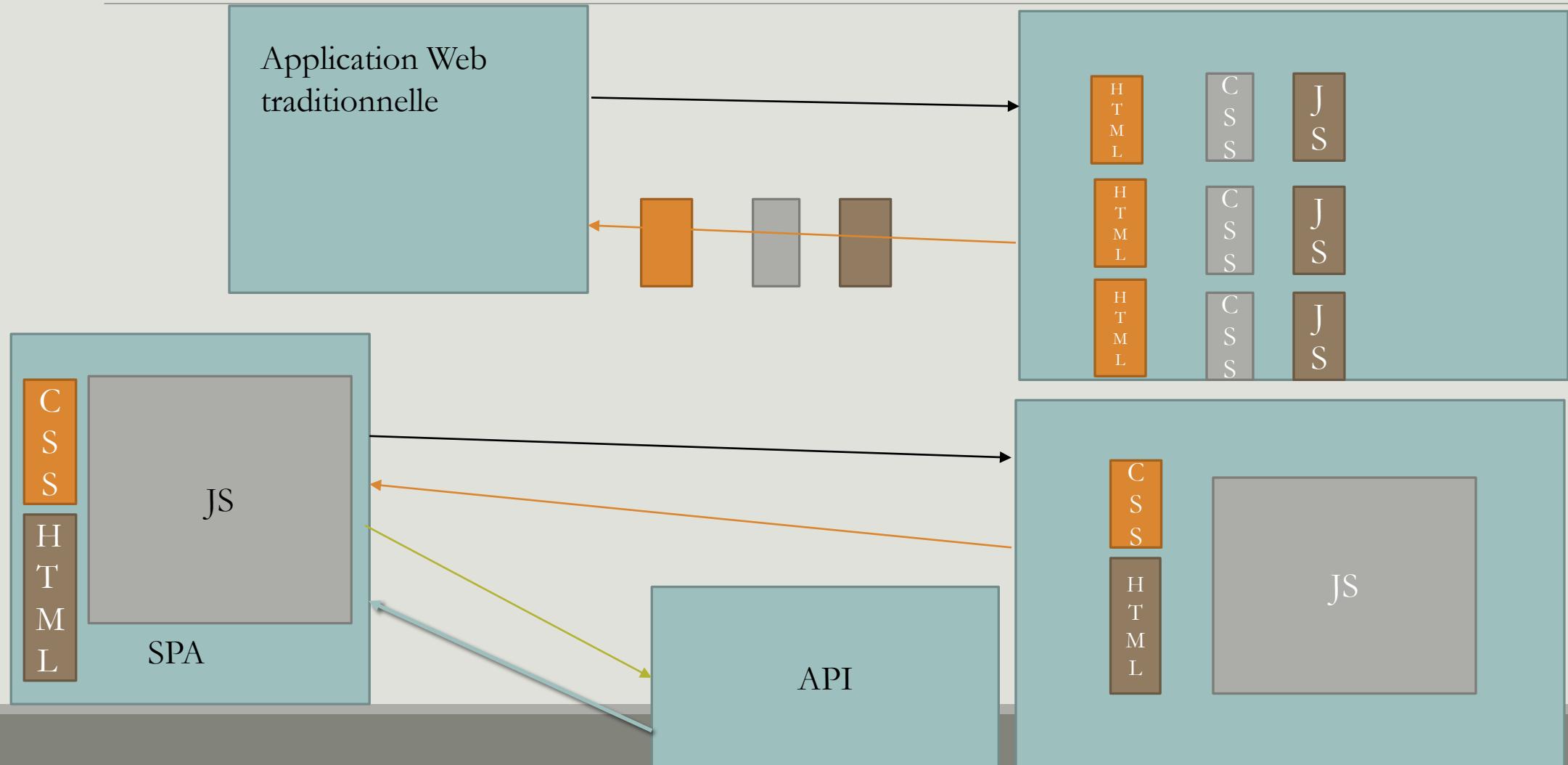
C'est quoi Angular?



C'est quoi Angular?

- Framework JS
- SPA
- Supporte plusieurs langages : ES5, EC6, TypeScript, Dart
- Rapide
- Orienté Composant

SPA



Angular : Arbre de composants

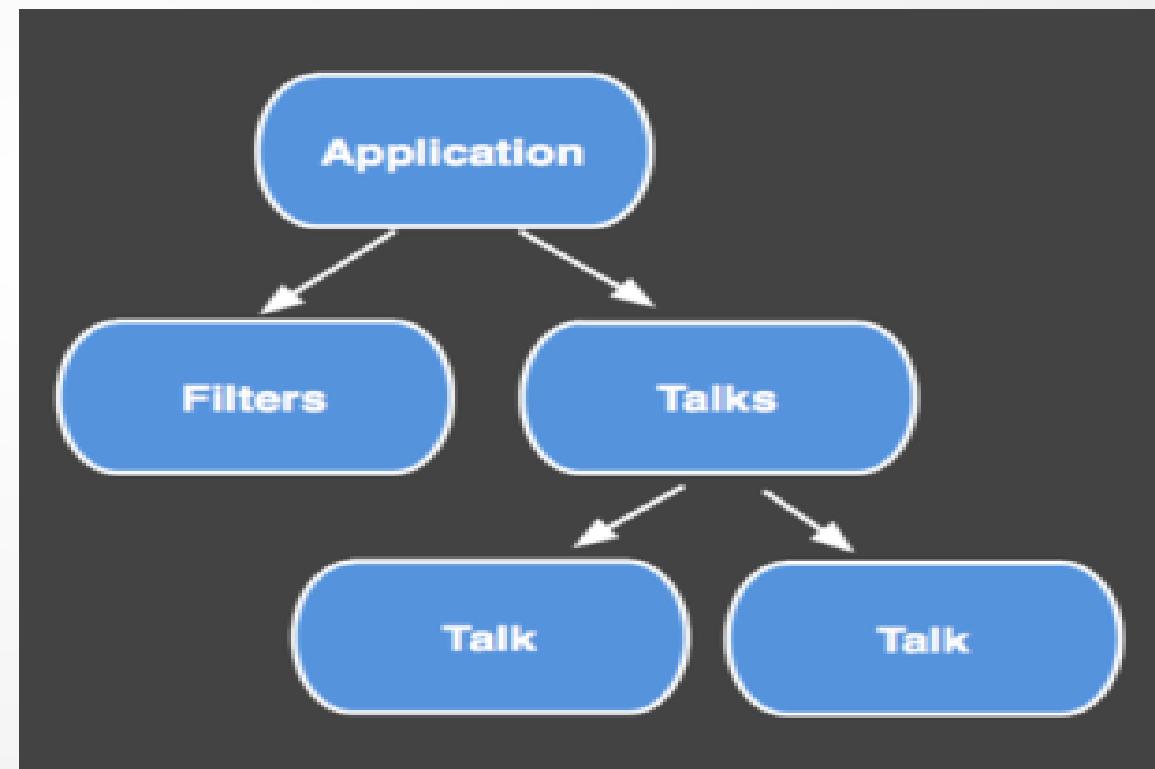
Speaker
Rich Hickey

FILTER

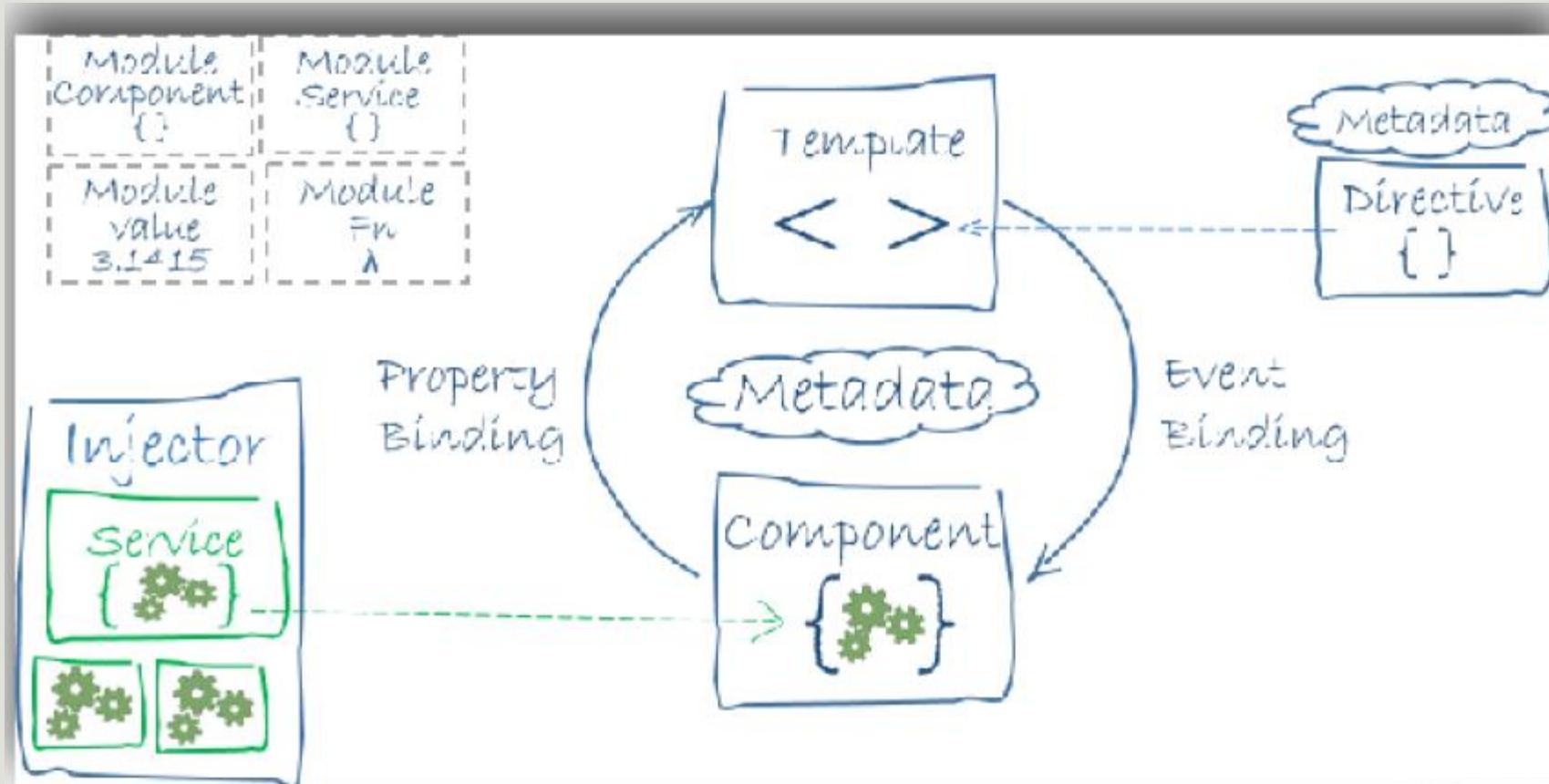
Rating 9.1 Are We There Yet?
Rich Hickey
WATCH RATE

Rating 8.5 The Value of Values
Rich Hickey
WATCH RATE

Rating 8.2 Simple Made Easy
Rich Hickey
WATCH RATE

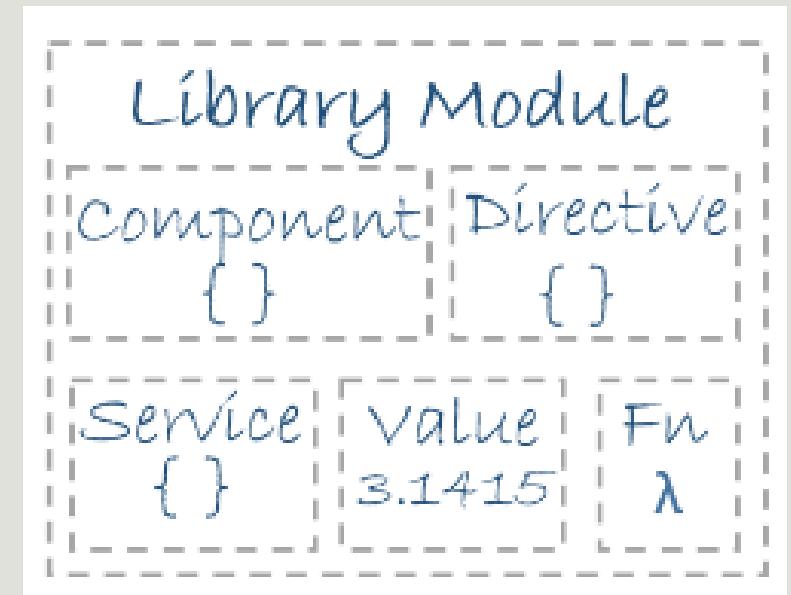


Architecture Angular



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.

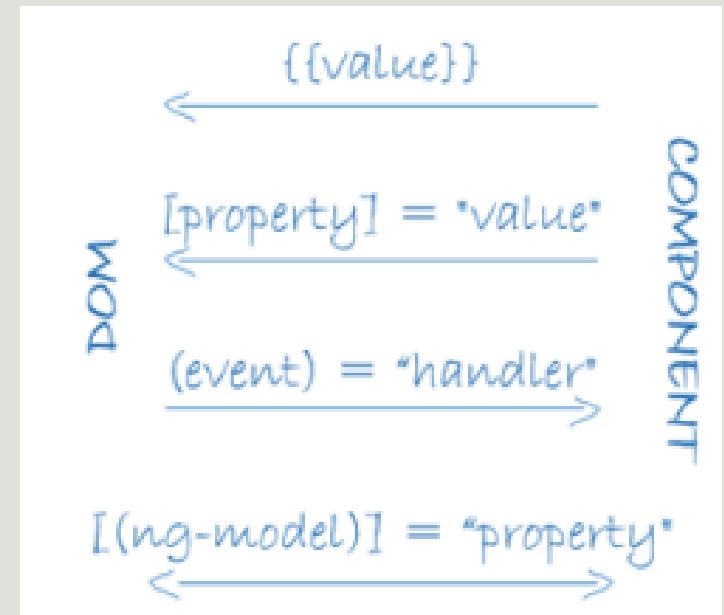
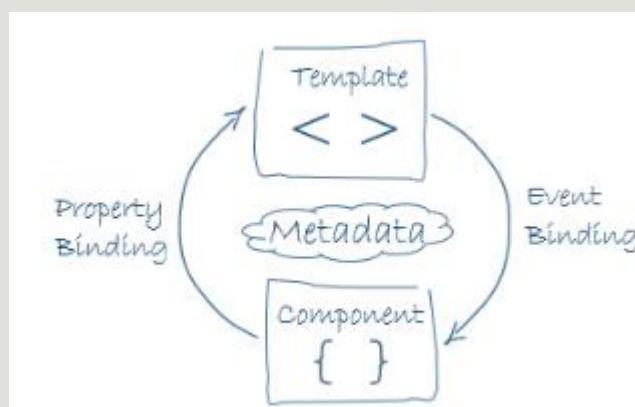
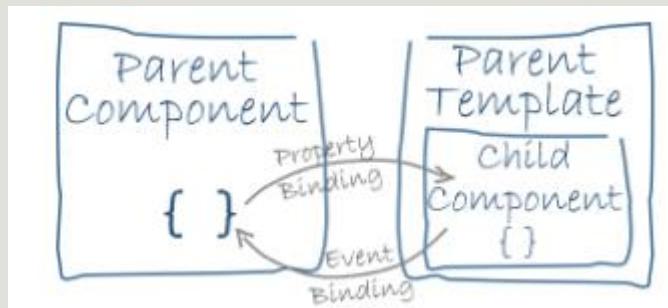


Les métas data

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

Le Data Binding

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



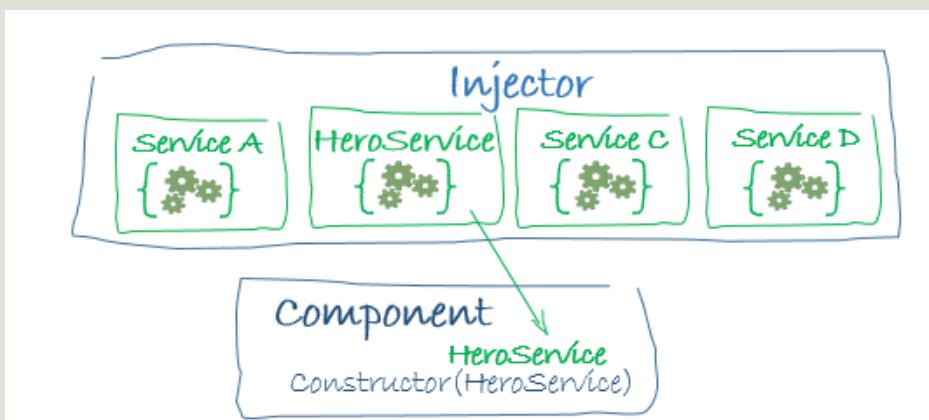
Les directives

- Les directives Angular sont des classes avec la métadata `@Directive`. Elle permettent de modifier le DOM et de rendre les Template dynamiques.
- Apparaissent dans des éléments HTML comme les attributs.
- Un composant est une directive à laquelle Angular a associé un Template.
- 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 :

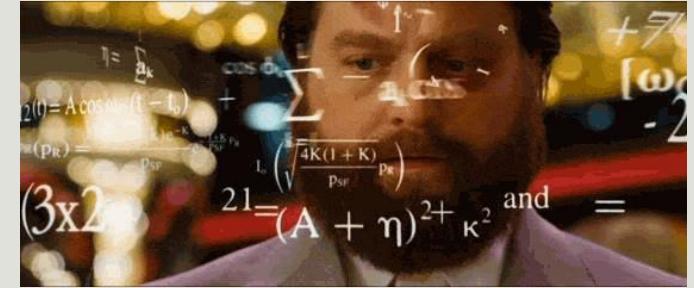
`git clone https://github.com/angular/quickstart.git quickstart`

- 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@19.2.15` installe la version [19.2.15](#)
 - [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@19.2.15 new projectName`](#)
- Afin d'avoir du help pour le cli tapez [`ng help`](#)
- Lancer le projet en utilisant la commande [`ng serve`](#)

Repo Github du Projet

- Pour notre formation, cloner ce projet dans lequel vous allez trouver les exemples et corrections des exercices:

```
git clone https://github.com/aymensellaouti/ngGL42526.git
```

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 "./../node_modules/bootstrap/dist/css/bootstrap.css";
```

Pour la majorité des bibliothèques, vous avez un fichier **package.json** avec les différents paths, vous permettant de ne pas spécifier le path complet, c'est le cas de bootstrap.

```
@import 'bootstrap';
```

```
"main": "dist/js/bootstrap.js",
"module": "dist/js/bootstrap.esm.js",
"sass": "scss/bootstrap.scss",
"style": "dist/css/bootstrap.css",
```

Essayer la même chose avec font-awesome.

Angular

Les composants

AYMEN SELLAOUTI

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

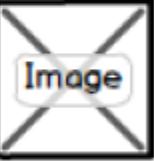
The screenshot displays a user interface for an 'Inventory Management App'. The top navigation bar includes standard icons for back, forward, close, and search, along with a title 'Inventory Management App'. Below this is a horizontal menu bar with three items: 'Home' (white background), 'Products' (blue background, currently selected), and 'Help'. A secondary navigation bar below the menu shows the current path: 'Products > Products List'. The main content area is a 'ProductList Component' containing three products, each represented by a card with an 'Image' placeholder icon:

- 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**
Adoeds Active Hat
Men > Accessories > Hats
\$238.99

Annotations on the right side identify the components: 'Navigation Component' points to the top bar, 'Breadcrumbs Component' points to the path bar, and 'ProductList Component' points to the main content area.

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



Premier Composant

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

[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>

Angular Les standalones composants

AYMEN SELLAOUTI

Pourquoi les NgModules

- La principale raison de l'introduction initiale de NgModules était pragmatique : **regrouper des blocs de construction qui sont utilisés ensemble.**
- Ceci permet non seulement d'augmenter la commodité pour les développeurs, mais également pour **le compilateur Angular**.
- En effet le NgModule permet de **définir le contexte de compilation**. A partir de ce contexte, le compilateur apprend où le programme est autorisé à appeler quels composants.
- Les deux parties **imports et declarations** permettent de définir ce contexte de compilation.

Pourquoi les NgModules

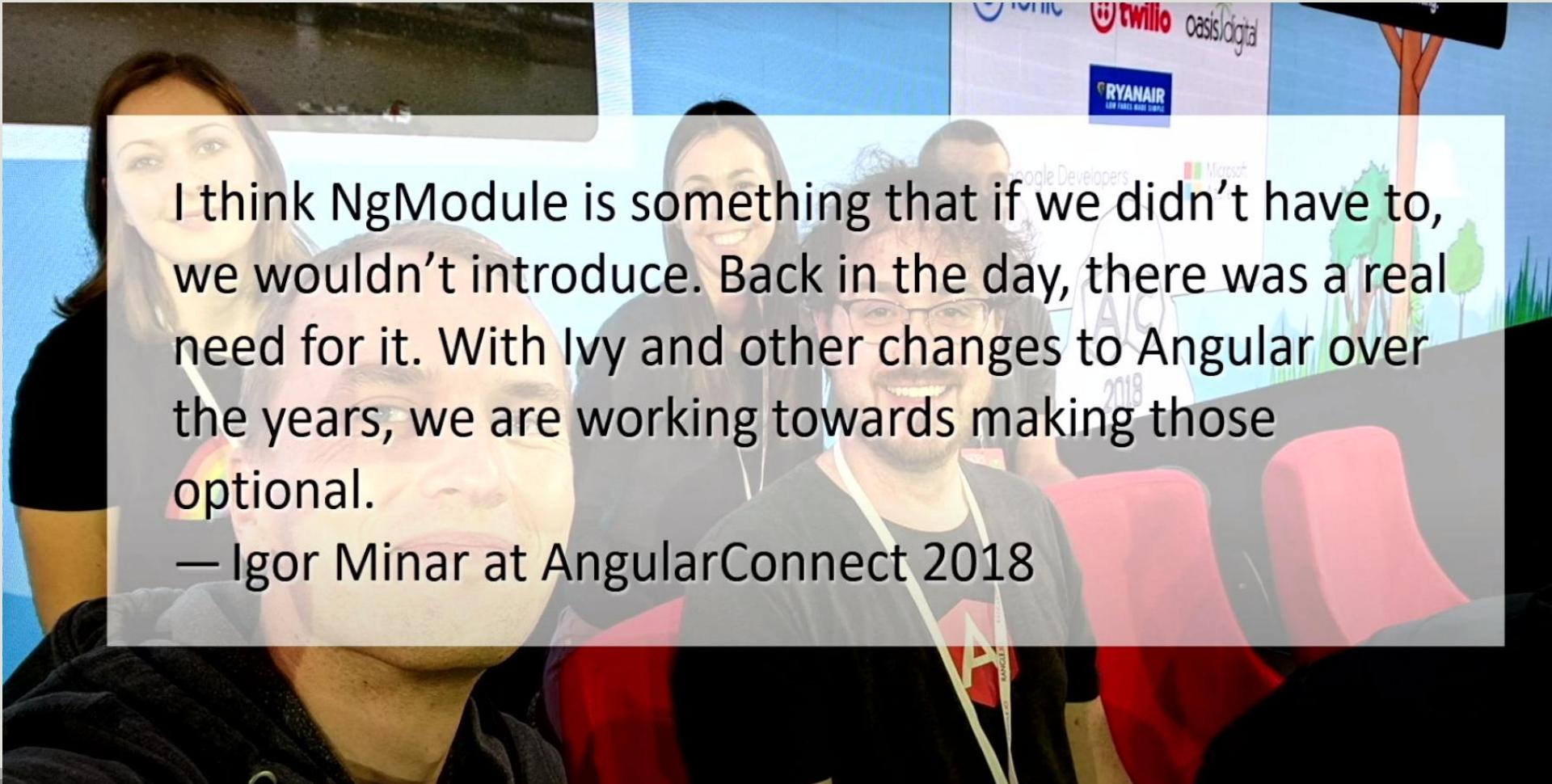
```
@NgModule({
  declarations: [
    AppComponent,
    ColorComponent,
    RainbowDirective,
    Btc2usdPipe,    ],
  imports: [
    BrowserModule,
    FormsModule,
  ],
  providers: [
    AuthInterceptorProvider,
    UUID_PROVIDER,
  ],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Contexte de compilation

Pourquoi les standalone Components ?

- Le premier problème qui s'est posé est **l'ambiguïté** que pose ce **deuxième système de module en parallèle à celui de EcmaScript**.
- Ceci a **compliqué l'apprentissage** pour les nouveaux apprenants.
- Ceci a poussé la Team Angular à faire en sorte qu'avec Ivy, le nouveau compilateur d'Angular, **on ait un contexte de compilation par composant**.
- C'est ce qui a permis la naissance des standalone component.

Pourquoi les standalone Components ?



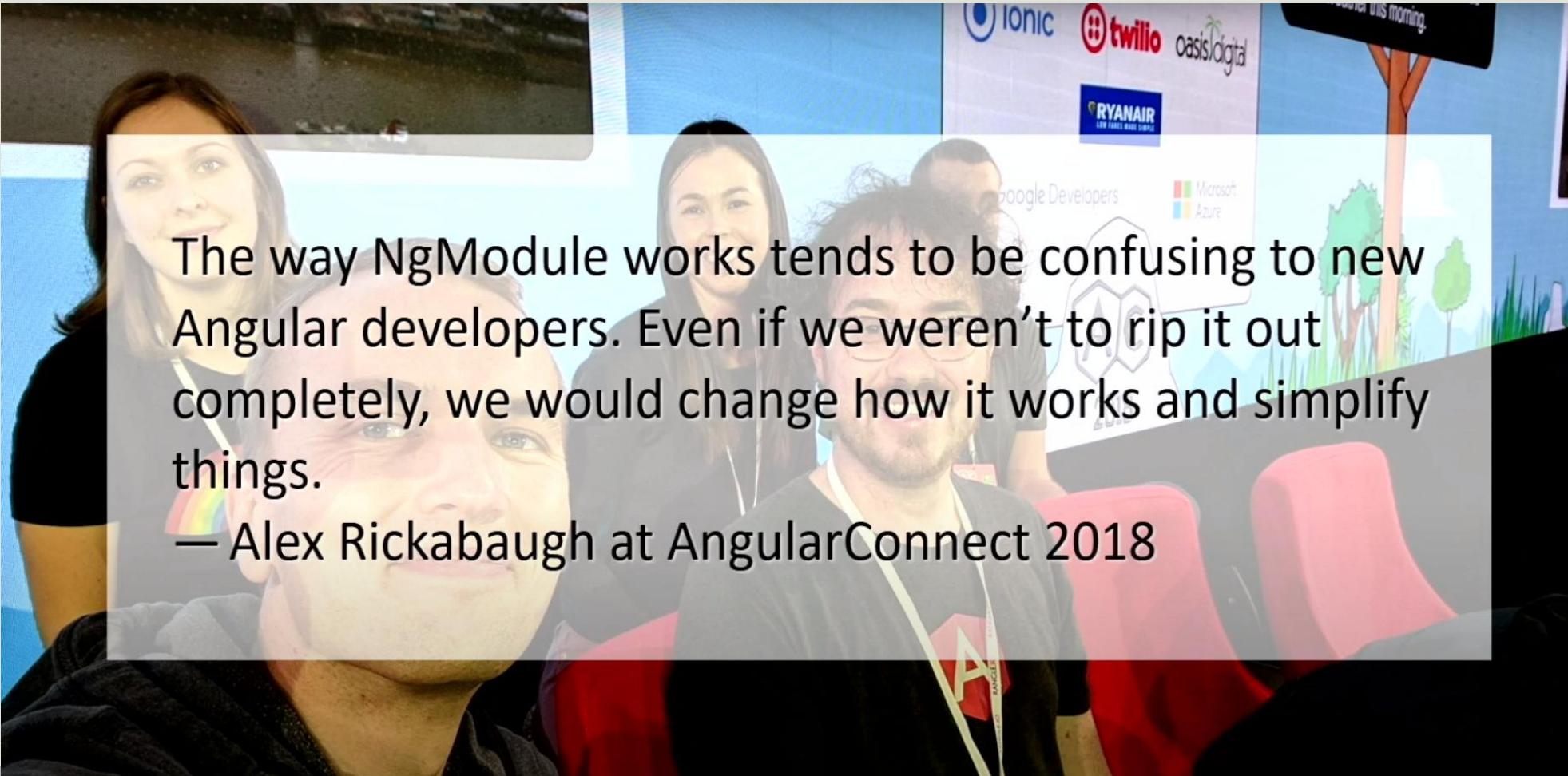
I think NgModule is something that if we didn't have to, we wouldn't introduce. Back in the day, there was a real need for it. With Ivy and other changes to Angular over the years, we are working towards making those optional.

— Igor Minar at AngularConnect 2018

Pourquoi les standalone Components ?

Single Component Angular Module

SCAM



Pourquoi les standalone Components ?

Single Component Angular Module

SCAM

- La deuxième problématique est la **granularité de lazy loading**.
- La question que beaucoup d'équipes se sont posée est : Pourquoi devons nous faire du **lazy loading pour tout un module quand nous avons besoin de juste un composant** ?
- C'est là où le pattern **SCAM (Single Component Angular Module)** a été introduit, c'est l'ancêtre du Standalone Component

Standalone Component

Les composants autonomes

- Un **standalone component** est un **composant indépendant d'un module**.
- Les composants standalone offrent un **moyen simplifié** de créer des applications Angular.
- Les applications existantes peuvent éventuellement et progressivement adopter le nouveau style autonome sans aucune modification majeure.
- Ceci est aussi **applicable** aux **directives** et aux **pipes**.
- L'utilisation d'une telle vision **simplifie l'apprentissage d'Angular**
- Elle **facilite aussi plusieurs aspects** comme le **lazy loading**

Standalone Component

Les composants autonomes

- Une **nouvelle propriété a été introduite** au niveau de l'objet de configuration du composant, c'est la propriété **standalone**.
- Si elle est à **true**, l'élément est **considéré standalone**.
- **A partir d'angular 19, la valeur par défaut est true, donc plus besoin de la mettre**
- Ces éléments **ne doivent plus être associés** à un **tableau de déclaration** d'un NgModule.
- Une nouvelle clé **import** permet aussi **d'importer les dépendances nécessaires**, que ce soit d'autres standalone components ou d'autres NgModule.
- Pour faire simple un composant Standalone est un **composant autonome**, avant c'est sa **famille** (Le NgModule auquel il appartient) qui été **en charge de ses dépendances**(imports et declarations). **Maintenant, c'est à lui de le faire**

Standalone Component

Les composants autonomes

- Si vous avez travaillé avec une **version antérieure à Angular18**. Afin de créer un *standalone* component, procéder comme vous le faites d'habitude avec un composant et ajouter l'option **-standalone** (**à partir d'Angular 18 c'est devenu le comportement par défaut**).

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

Standalone Component

Les composants autonomes

- Les **imports définissent le contexte de compilation du composant** : tous les autres blocs de construction que les composants autonomes sont autorisés à utiliser.
- Elles permettent donc **d'importer d'autres composants autonomes, mais aussi des NgModules existants.**
- Ceci rend le **composant autonome et augmente ainsi sa réutilisabilité.**
- Elle nous oblige également à réfléchir aux dépendances du composant.
- **Cette tâche s'avère extrêmement monotone et chronophage.** Ce qu'on faisant une fois dans le module on le fera pour chaque composant.
- Les **ides essayent de remédier à ça via les autocomplete**

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

Standalone Component

Le modèle mental

- Même si **ce n'est pas géré de la même manière par le compilateur d'Angular**, vous pouvez voir un **standalone component comme un NgModule avec un seul composant**

```
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css'],  
  imports: [  
    RouterOutlet,  
    HighlightDirective,  
    CardComponent  
,  
    standalone: true,  
  ]  
})  
export class AppComponent {
```



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

```
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css'],  
})  
export class AppComponent {}
```

Standalone Component

Les composants autonomes

Utilisation dans les NgModule

- Si vous voulez utiliser un **Standalone Component dans un NgModule**, **importer le comme** vous importer **un module**.
- Avec cette manière de faire, un **composant standalone peut être importé dans plusieurs modules**, alors qu'avant **un composant ne pouvait être déclaré que dans un seul module**.

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

Standalone Component

Bootstraper une application avec un Standalone Component

- Afin de **bootstraper** votre application avec un **standalone component**, utilisez la fonction **bootstrapApplication** dans **main.ts**, à la place de **platformBrowserDynamic().bootstrapModule**, en lui passant le **Standalone Component que vous voulez déclencher au lancement de l'application.**
- Dans **index.html** appelez **ce Standalone Component.**

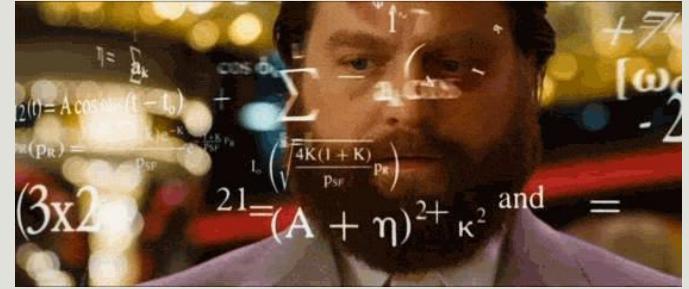
```
bootstrapApplication(StandaloneComponent);
```

main.ts

```
<body>
<div class="container">
  <app-standalone></app-standalone>
</div>
</body>
</html>
```

index.html

Migration vers les standalones



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

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

Même si cet outil est très efficace, il vous reste quelques modules et factorisation à gérer manuellement

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

Que contient le composant ?

- Le composant dispose de deux parties :
 - La partie HTML qui représente la Vue
 - La partie TS décrivant l'état et le comportement du composant

L'état et le comportement d'un composant

```
export class FirstComponent {  
  //Propriétés : état State  
  name = 'First Component';  
  isHidden = false;  
  message = '';  
  //Méthodes : comportement Behaviour  
  showHide() {  
    this.isHidden = !this.isHidden;  
  }  
  changeMessage(newMessage: string) {  
    this.message = newMessage;  
  }  
}
```

Classe

```
<!-- attributs: l'état de la balise -->  
<p hidden> First Component works!</p>  
<div class="alert alert-info">  
  Je test le binding  
</div>  
<p>Le contenu de l'input est : </p>  
<input  
  type="text"  
  class="form-control"  
>  

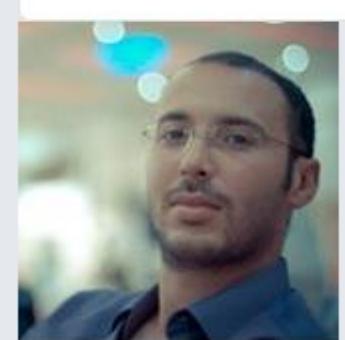
```

Template

First Component works!

Je test le binding

Le contenu de l'input est :



- Et si on voulait afficher quelque chose dans notre template ?
- Et si on voulait contrôler un attribut de notre template ?
- Et si on voulait réagir à un événement qui survient dans notre template ?

Afficher des propriétés dans le Template Interpolation

- L'interpolation permet de projeter des valeurs de propriétés dans votre template.
- Angular utilise la syntaxe "double accolades {{ }} " pour l'interpolation

```
export class FirstComponent {  
    //Propriétés : état State  
  
    name = 'First Component';  
    isHidden = false;  
    message = '';  
    //Méthodes : comportement Behaviour  
    showHide() {  
        this.isHidden = !this.isHidden;  
    }  
    changeMessage(newMessage: string) {  
        this.message = newMessage;  
    }  
}
```

```
<!-- attributs: l'état de la balise -->  
<p hidden> First Component works!</p>  
<p hidden> {{name}} works!</p>  
<div class="alert alert-info">  
    Je test le binding  
</div>  
<p>Le contenu de l'input est : </p>  
<input  
    type="text"  
    class="form-control"  
>  

```



Contrôler un attribut de notre template Property Binding

- Afin de **contrôler un attribut d'une des balises de notre Template**, angular nous fournit le concept de **Binding de propriété (Property Binding)**.
- C'est un Binding unidirectionnel.
- La propriété liée au composant est interprétée avant d'être ajoutée au Template.
- syntaxe: **[propriété] = "varOuCte"**

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

Contrôler un attribut de notre template Property Binding

```
export class FirstComponent {  
    //Propriétés : état State  
    name = 'First Component';  
    isHidden = false; ←  
    message = '';  
    imagePath = 'assets/images/as.jpg'; ←  
    //Méthodes : comportement Behaviour  
    showHide() {  
        this.isHidden = !this.isHidden;  
    }  
    changeMessage(newMessage: string) {  
        this.message = newMessage;  
    }  
}
```

```
<!-- attributs: l'état de la balise -->  
<p hidden>{{name}} works!</p>  
<p [hidden]="isHidden"> {{name}} works!</p>  
  
<div class="alert alert-info">  
    Je test le binding  
</div>  
<p>Le contenu de l'input est : {{ message }}</p>  
<input  
    type="text"  
    class="form-control"  
    >  
  
<img [src]="imagePath" alt="aymen">
```

First Component works!

Je test le binding

Le contenu de l'input est :



Réagir à un événement qui survient dans notre template Event Binding

- Afin d'écouter et de réagir à un événement déclenché dans notre Template, angular nous fournit le concept de Binding d'événement (Event Binding).
- C'est un binding unidirectionnel, Il permet d'interagir du DOM vers le composant. L'interaction se fait à travers les événements.
- Syntaxe : (evenement)="methodeAExecuter()">>

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

Réagir à un événement qui survient dans notre template

Event Binding

```
export class FirstComponent {  
  //Propriétés : état State  
  name = 'First Component';  
  isHidden = false;  
  message = '';  
  //Méthodes : comportement Behaviour  
  showHide() {  
    this.isHidden = !this.isHidden;  
  }  
  changeMessage(newMessage: string) {  
    this.message = newMessage;  
  }  
}
```

```
<!-- attributs: l'état de la balise -->  
<p [hidden]="isHidden"> {{name}} works!</p>  
<!-- Au click appelle la fonction showHide -->  
<div (click)="showHide()" class="alert alert-info">  
  Je test le binding  
</div>  
<p>Le contenu de l'input est : {{ message }}</p>  
<input  
  type="text"  
  class="form-control"  
>
```

TEMPLATE REFERENCE

- Dans certains cas d'utilisation, vous avez besoin de récupérer **la référence d'un objet du DOM dans votre template**. Par exemple la référence d'un input pour accéder à sa valeur.
- Pour ce faire, on utilise le symbole # pour la création d'une variable de référence.

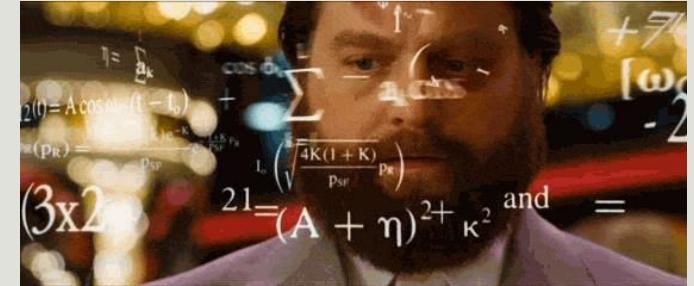
```
export class FirstComponent {  
    //Propriétés : état State  
    name = 'First Component';  
    isHidden = false;  
    message = '';  
    //Méthodes : comportement Behaviour  
    showHide() {  
        this.isHidden = !this.isHidden;  
    }  
    changeMessage(newMessage: string) {  
        this.message = newMessage;  
    }  
}
```

```
<!-- attributs: l'état de la balise -->  
<p hidden> First Component works!</p>  
<div class="alert alert-info">  
    Je test le binding  
</div>  
<p>Le contenu de l'input est : </p>  
<input  
    #myInput ←  
    (change)="changeMessage(myInput.value)"  
    type="text"  
    class="form-control"  
    >  

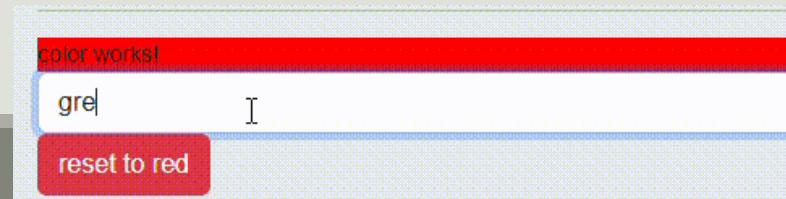
```



Exercice



- Créer un nouveau composant. Ajouter y un Div et un input de type texte.
- Fait 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]**



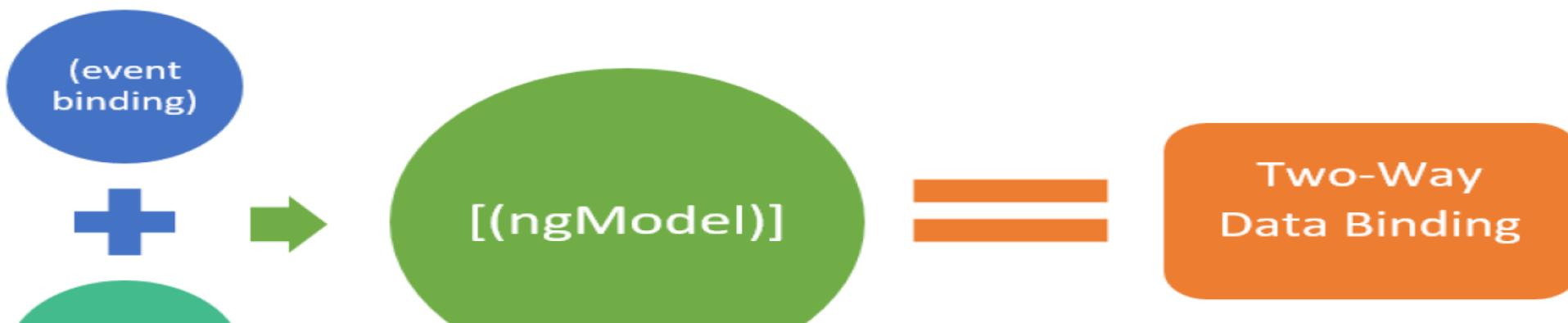
Two way Binding

- 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

Two way Binding

PROPERTY BINDING + EVENT BINDING



```
<hr>
Change me <input [(ngModel)]="nom">
<br>My new name is {{nom}}
```

Template

Property Binding et Event Binding

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

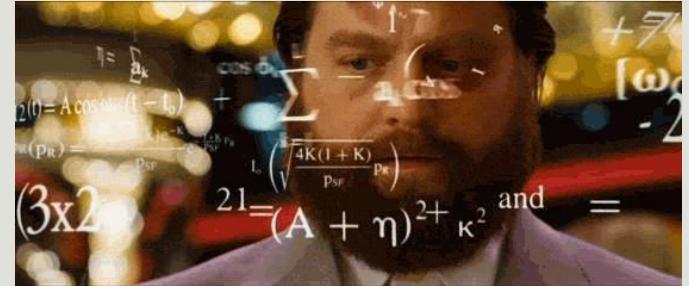
@Component({
  selector: 'app-two-way',
  templateUrl: './two-way.component.html',
  styleUrls: ['./two-way.component.css']
})
export class TwoWayComponent {
  two:string="myInitValue";
}
```

Component

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

Template

Exercice



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

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

Exercice



Aymen Sellaouti
trainer

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

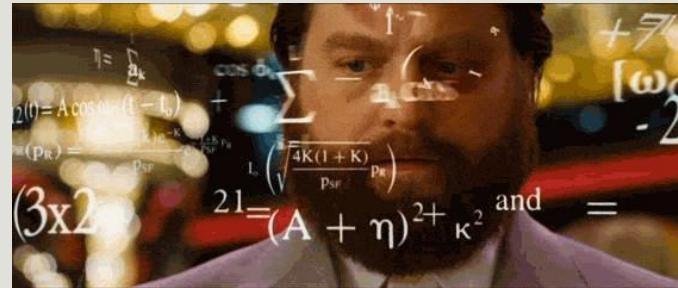
Auto Rotation

name :

firstname :

job :

path :


$$I(t) = A \cos(\omega t - I_0) + \sum_{k=1}^{\infty} I_k \cos(k\omega t - \phi_k)$$
$$(P_R) = \frac{I_0^2}{P_{SF}} \rightarrow \frac{I_0^2 R_S}{P_{SF}}$$
$$I_0 = \left(\sqrt{\frac{4K(1+K)}{P_{SF}}} P_R \right)^{1/2}$$
$$(3x2) \quad 21 = (A + \eta)^2 + \kappa^2 \text{ and } =$$

Exercice

Two Way Binding



Sellaouti Aymen
Enseignant
tant qu'il y a de la vie il y a de l'espoir

Nom : Sellaouti

Prénom : aymen

Job : Enseignant

image : as.jpg

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

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

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

Auto Rotation

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 : Sellaouti

Prénom : aymen

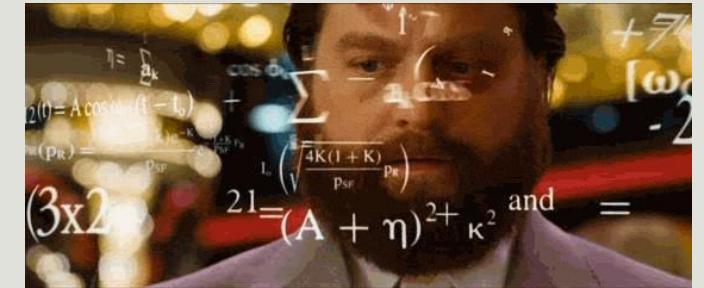
Job : Enseignant

image : as.jpg

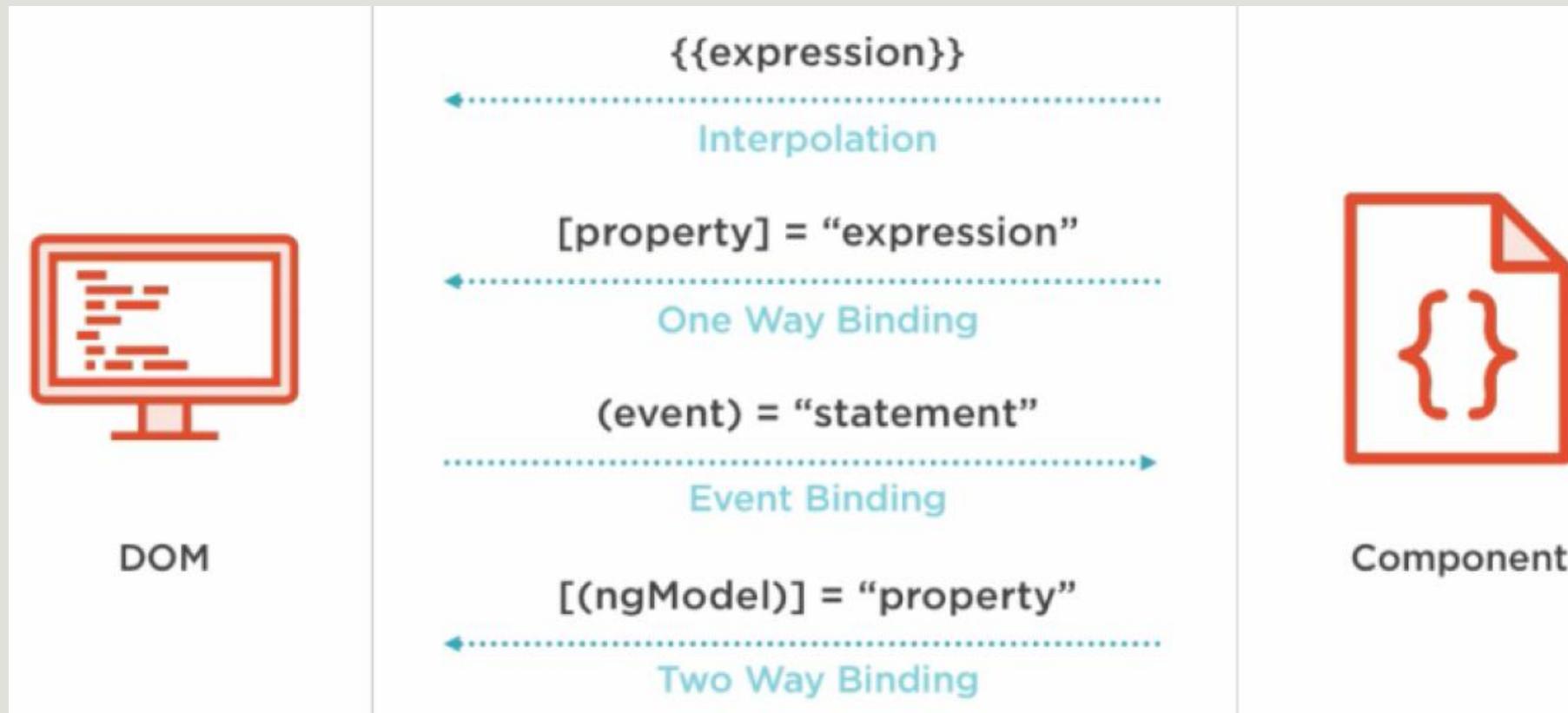
Citation Favorite : tant qu'il y a de la vie il y a de l'espoir

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

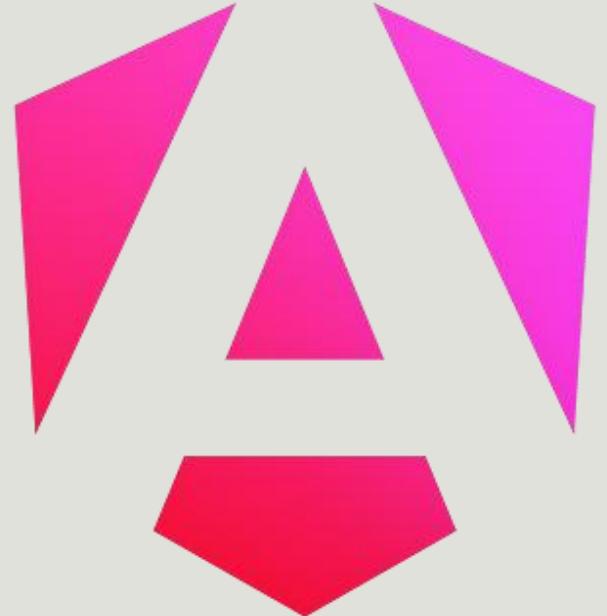
Mots clé de votre travail : HTML CSS JS PHP Symfony Angular



Résumé : Property Binding



Les Signaux



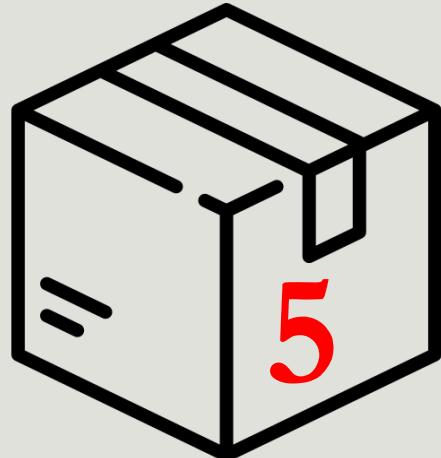
AYMEN SELLAOUTI



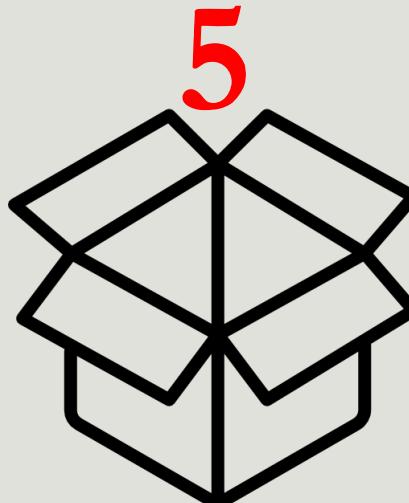
Qu'est ce qu'un Signal ?



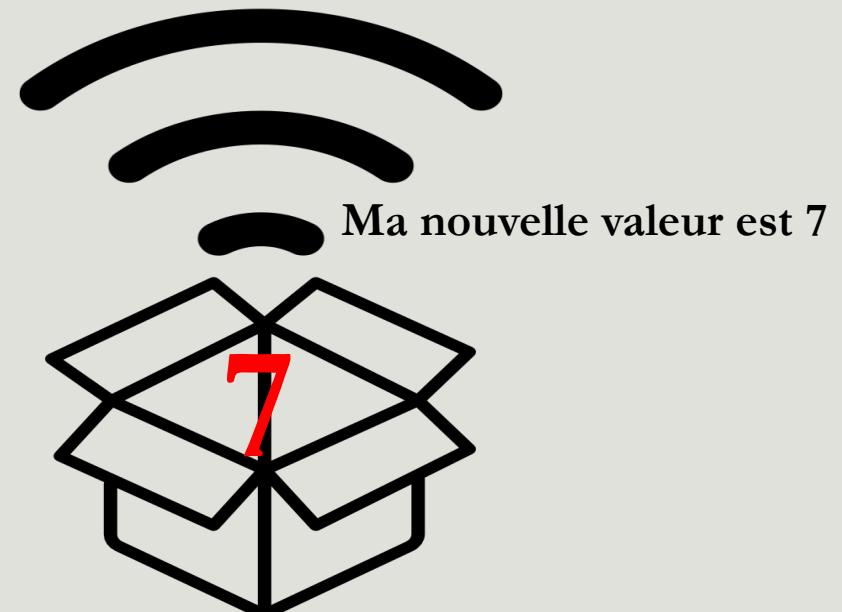
- Un signal agit comme une **enveloppe (wrapper) autour d'une valeur**, ayant la capacité **d'avertir les consommateurs lorsque la valeur change**.
- On peut modéliser ça en disant qu'un signal c'est une donnée qui a le pouvoir de notifier ses changements.



`value = signal(5)`



`value()`



`value.set(7)`

Qu'est ce qu'un Signal ?



- Un signal est une **primitive réactive** qui représente une valeur et qui nous permet de
 - **suivre ses changements** au fil du temps.
 - **modifier** cette même valeur en **notifiant tous ceux qui en dépendent**.
- Elle permet de définir l'état réactif de votre application et d'avoir une **identification précise de quels composants sont impactés par un changement**.

Qu'est ce qu'un Signal ?



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

Pourquoi intégrer les signaux



Reactivity Everywhere

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



Precision Updates

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

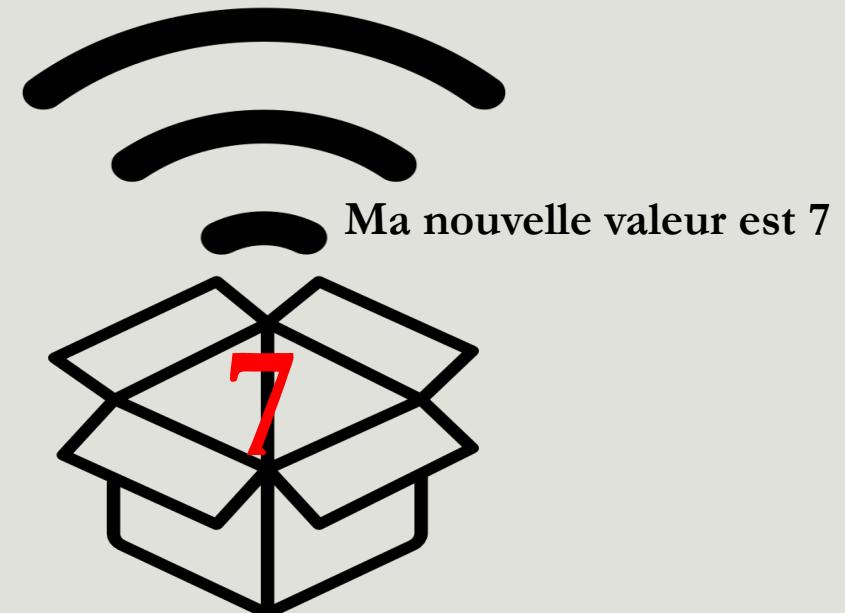
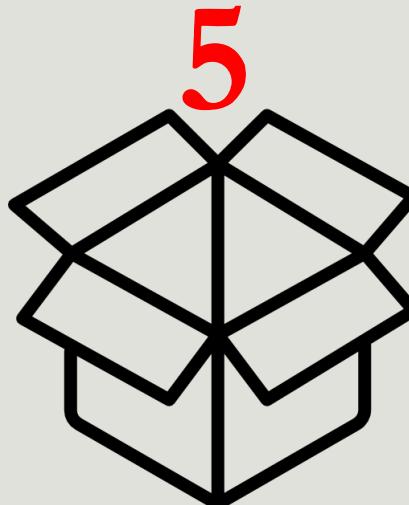
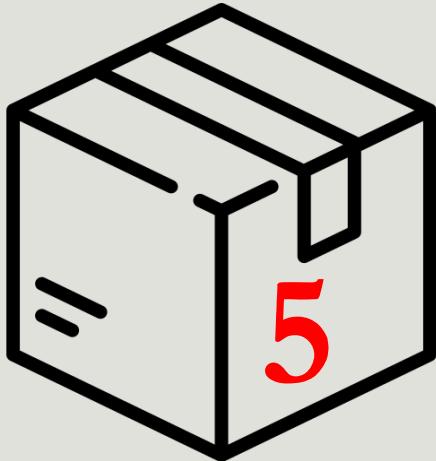


Lightweight Dependencies

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

API

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



Créer un signal via l'api signal()

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

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

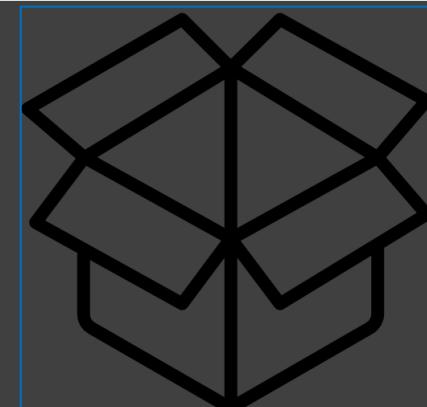


Récupérer la valeur d'un signal

- ▶ 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');
}
```

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**.

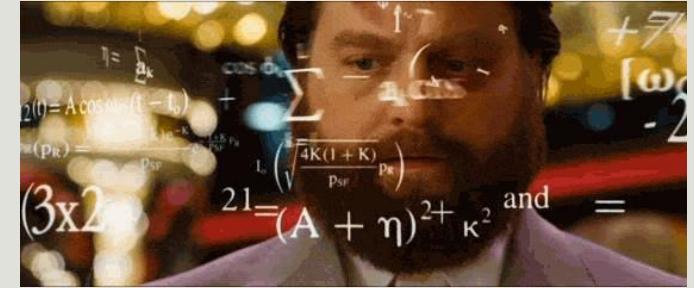
```
@Component({
  selector: 'app-signal-api',
  standalone: true,
  imports: [],
  template: `
    <h1>Hello {{ lastname() }}</h1>
    <input type="number" #input
      (change)="setCounter(+input.value)"
    />
    <h2 (click)="increment()">
      Click here. You clicked {{ counter() }} times
    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

Quand utiliser un signal et non une variable standard ?

- Utilisez les signals
 - pour chaque **variable d'état de votre composant dont la valeur change**.
 - lorsque vous voulez **suivre les changements** de votre variable
 - quand la **valeur de votre variable** doit **changer quand d'autres variables changent**
- N'utilisez pas les signals
 - quand la **valeur ne change pas**
 - Quand c'est une **variable locale** que vous **n'utilisez pas dans votre ui** ou dans dans **calculs réactifs**
 - pour les **événements**
 - pour les **opérations asynchrones**

Exercice

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



Cas d'utilisation

- Soit le composant SumComponent. Que se passe t il si on incrément x ou y ?

```
export class SumComponent {  
  x = 3;  
  y = 5;  
  z = this.x + this.y;  
}
```

```
<input type="number" [(ngModel)]="x"  
      class="form-control">  
<input type="number" [(ngModel)]="y"  
      class="form-control">  
{{ x }} + {{ y }} = {{ z }}
```

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**.
- Vous **ne pouvez pas modifier un computed**, il n'est pas Writable

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

computed()

Modifier un signal dans un computed

- computed() s'attend à ce que le calcul **soit léger et rapide**, et **n'entraîne aucun effet secondaire**.
- Vous **ne pouvez pas modifier un signal dans un computed**, ceci provoquera une **erreur**.
- Ne **modifiez aucune variable** dans les computed ; pas seulement les signaux, mais aussi les variables extérieures à la fonction.
- Ne **modifiez pas le DOM** et **n'exécutez pas de tâches asynchrones**.

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

- ▶ 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

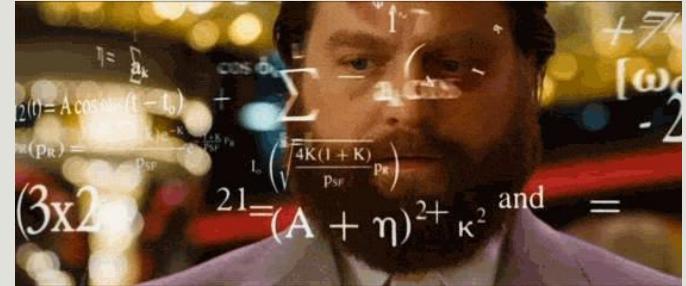
```
lastname = signal('sellaouti');
firstname = signal('aymen');
fullname = computed(() => `${this.firstname()} ${untracked(this.lastname)}`);
```

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) à 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

L'API effect()

- Dans certains cas d'utilisation, vous avez besoin d'être notifié par le changement d'un signal pour **effectuer un effet de bord**, donc faire un **traitement sans pour autant créer un nouveau signal ou en modifier d'autres**.
- Pensez à un log, update du localStorage,...
- L'effect va être réexécuté si l'un des signaux qu'il utilise émet une nouvelle valeur.

L'API effect()

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

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

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

L'API effect()

- Moment d'exécution : Un effect s'exécute pendant le cycle de détection des changements, après que les signaux suivis ont été mis à jour, mais avant que le DOM ne soit complètement rendu.
- Depuis Angular 19, les effets de composant (component effects) sont intégrés au processus de détection des changements, ce qui garantit un timing plus prévisible.
- Les effets seront exécutés le **nombre minimum de fois**. Si un effet **dépend** de **plusieurs signaux** et que plusieurs d'entre eux **changent simultanément**, une seule exécution de l'effect sera programmée.
- **Limitation** : Ne **doit pas être utilisé pour des manipulations directes du DOM**, car **le DOM peut ne pas être encore mis à jour**.

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

L'API afterRenderEffect()

- La primitive **afterRenderEffect** ([introduite dans Angular 19](#)) est une **extension de effect** qui **s'exécute après que le rendu du DOM** est terminé. Elle est conçue pour les cas où vous **devez interagir avec le DOM** ou **effectuer des opérations qui dépendent de son état final**.

```
@ViewChild('myDiv') myDiv!: ElementRef<HTMLDivElement>;
constructor() {
  effect(() => {
    console.log(`myDiv width effect: ${this.myDiv.nativeElement.offsetWidth}px`);
    console.log(`myDiv height effect: ${this.myDiv.nativeElement.offsetHeight}px`);
  });
  afterRenderEffect(() => {
    console.log(`myDiv width afterRenderEffect:${this.myDiv.nativeElement.offsetWidth}px`);
    console.log(`myDiv height afterRenderEffect: ${this.myDiv.nativeElement.offsetHeight}px`);
  });
}
```

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']);  
currenciesSignal = this.currencies.asReadOnly();
```

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>
```

@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>
```

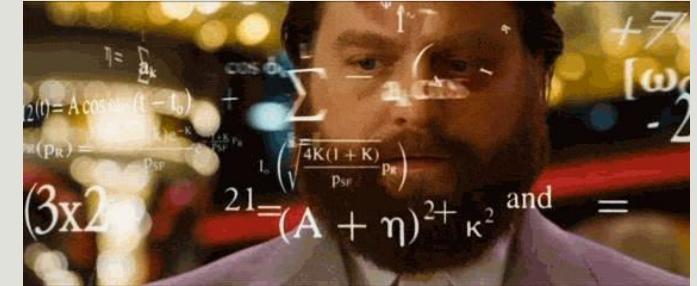
Control flow

@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, soit suivie d'une icône de votre choix

sellaouti
lewandowski ❤
dupont

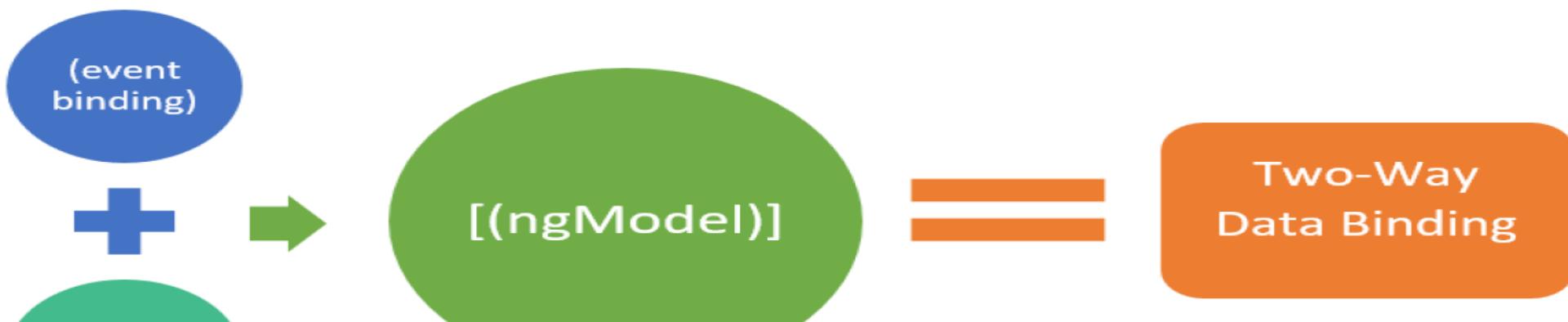
Two way Binding

- 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

Two way Binding

PROPERTY BINDING + EVENT BINDING



```
<hr>
Change me <input [(ngModel)]="nom">
<br>My new name is {{nom}}
```

Template

Property Binding et Event Binding

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

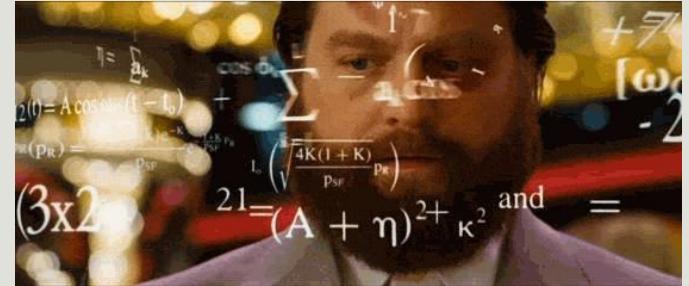
@Component({
  selector: 'app-two-way',
  templateUrl: './two-
way.component.html',
  styleUrls: ['./two-way.component.css']
})
export class TwoWayComponent {
  two:any="myInitValue";
}
```

Component

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

Template

Exercice



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

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

Exercice



Aymen Sellaouti
trainer

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

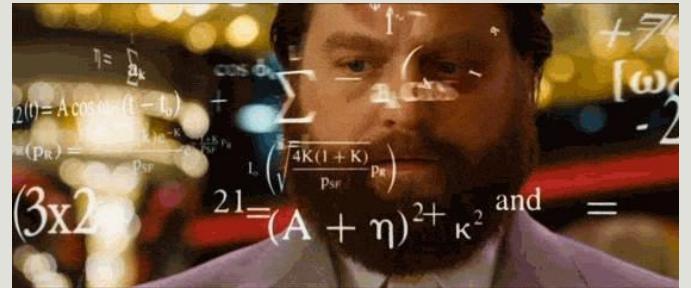
Auto Rotation

name :

firstname :

job :

path :


$$I \approx \sum_{k=1}^{\infty} I_k$$
$$I_2(t) = A \cos(\omega t - I_0)$$
$$P_R = \frac{I_0}{D_{SF}} e^{-\frac{I_0}{D_{SF}} r_B}$$
$$(3x2) = \frac{21}{(A + \eta)^2 + \kappa^2} \text{ and } =$$

Exercice

Two Way Binding



Sellaouti Aymen
Enseignant
tant qu'il y a de la vie il y a de l'espoir

Nom : Sellaouti

Prénom : aymen

Job : Enseignant

image : as.jpg

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

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

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

Auto Rotation

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 : Sellaouti

Prénom : aymen

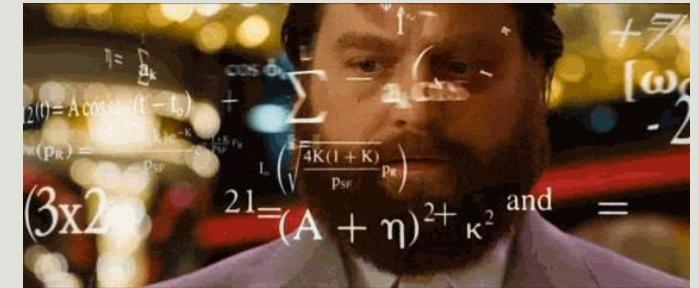
Job : Enseignant

image : as.jpg

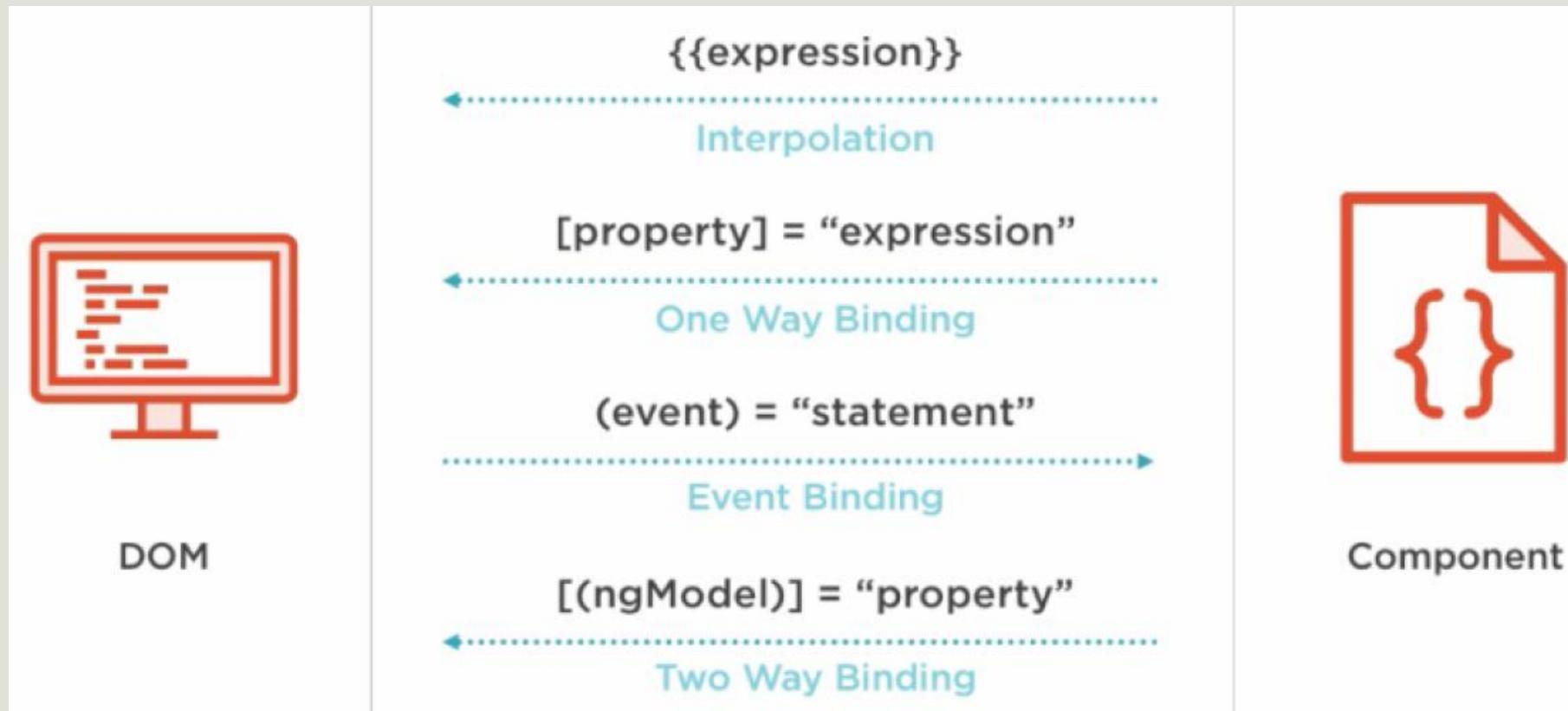
Citation Favorite : tant qu'il y a de la vie il y a de l'espoir

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

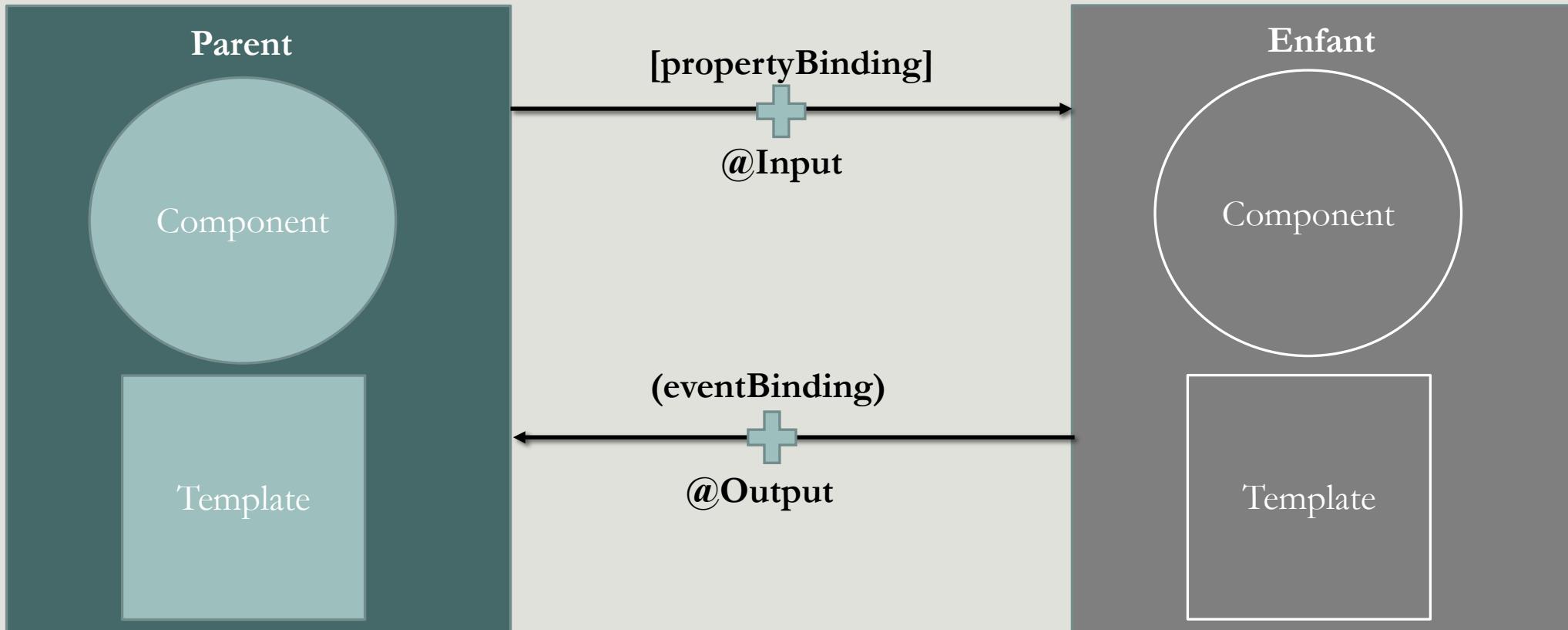
Mots clé de votre travail : HTML CSS JS PHP Symfony Angular



Résumé : Property Binding

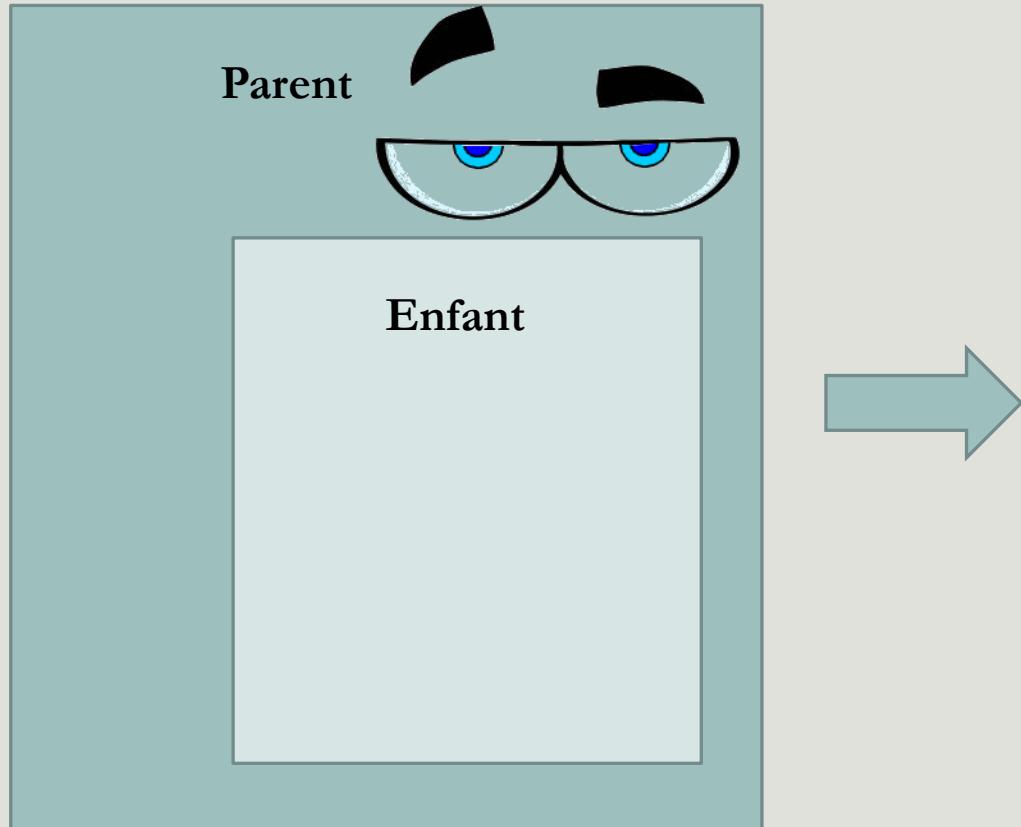


Interaction entre composants



Pourquoi ?

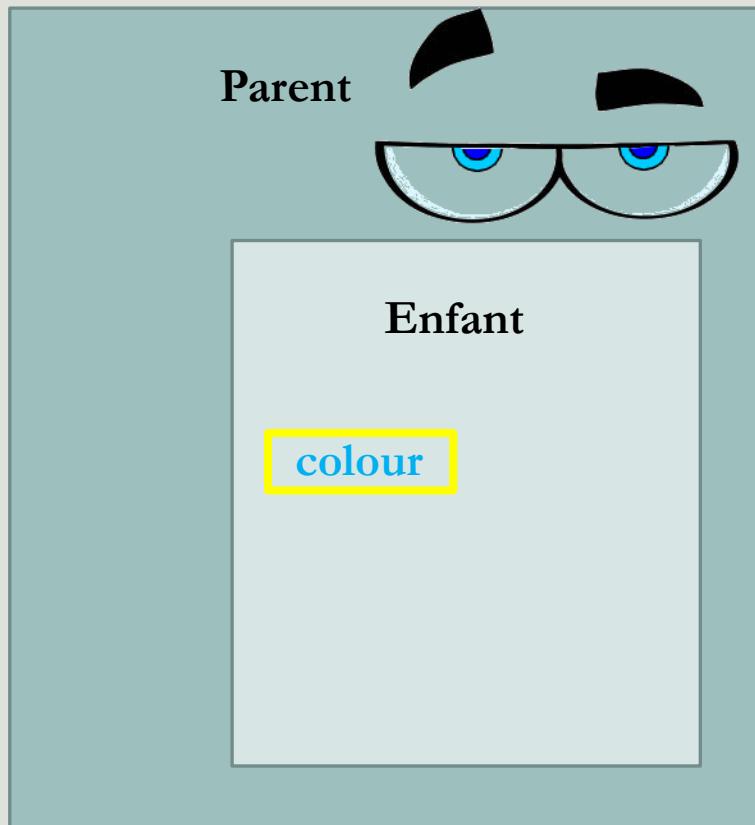
Le père voit le fils, le fils ne voit pas le père



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

@Component({
  selector: 'app-pere',
  template: `
    <p>Je suis le composant père </p>
    <forma-fils></forma-fils>
  `,
  styles: []
})
export class PereComponent {
  color = 'green';
}
```

Interaction du père vers le fils



- Le parent **voit l'enfant** mais **ne peut pas voir ses propriétés**.
- Solution : Faire du **property binding avec @input**, qui peut prendre un objet en paramètre (pour spécifier que l'envoie d'une valeur est required par exemple).

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

@Component({
  selector: 'app-child-first',
  templateUrl: './child-first.component.html',
  styleUrls: ['./child-first.component.css']
})
export class ChildFirstComponent implements OnInit {
  colour:string;
}
```

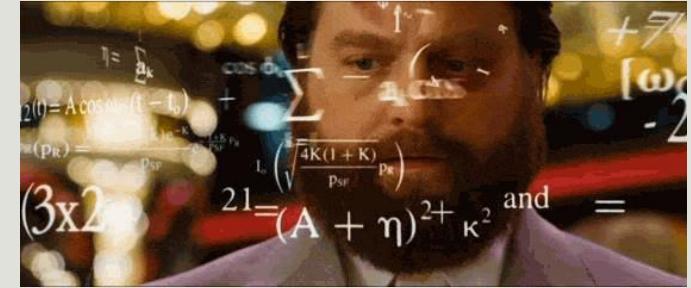
```
<app-fils [colour]="color"/>
```

Template du pere

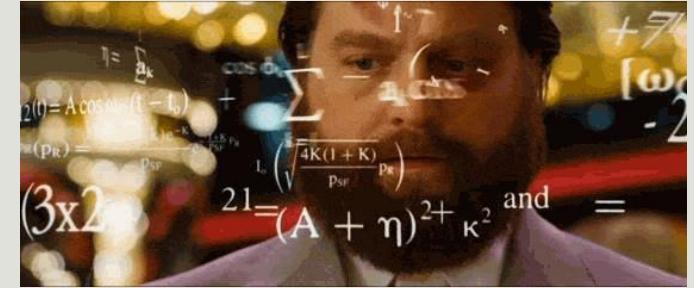
Ts du fils

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 paire ou impaire.
- Crée 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`

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

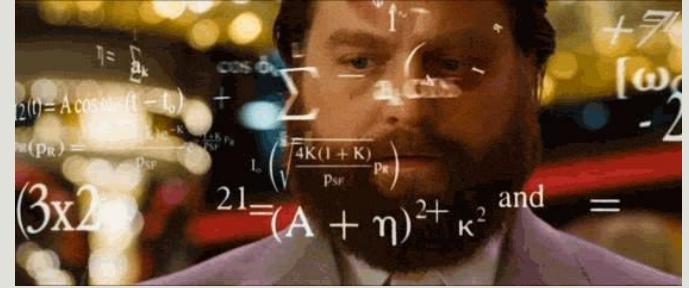
```
counter = input.required({  
  alias: 'counter',  
  transform: (value: number) => value * 100,  
});
```

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

```
// avant  
@Input({required: true})  
Counter!:number;  
// après  
counter = input.required<number>();
```

Signal Input

- Reprenez cet exemple en utilisant les Signal Input



0 is even

Signal Input Migration

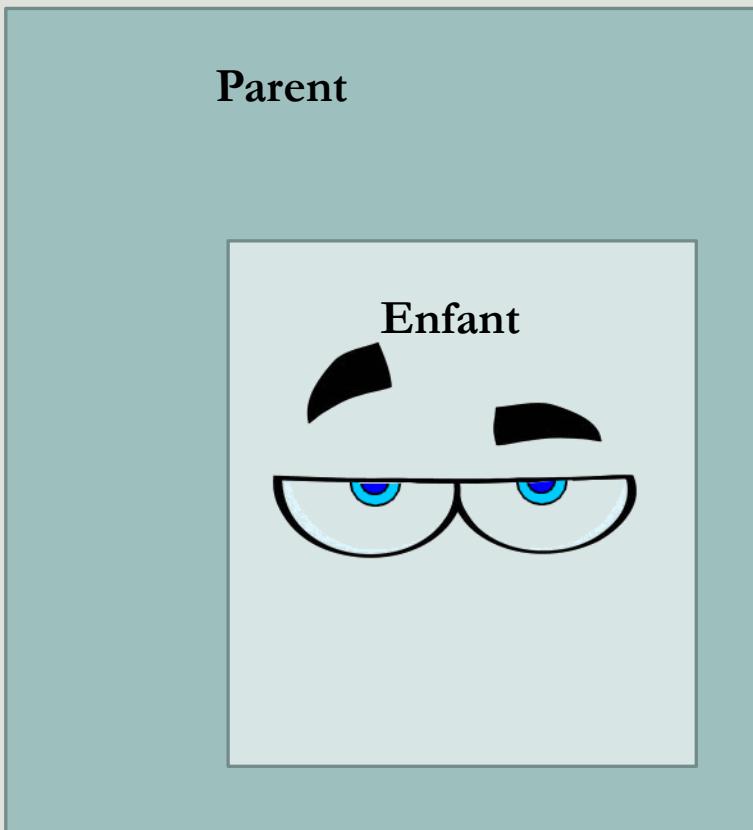
- Vous pouvez migrer toute votre application vers les signals input via cette commande :

```
ng generate @angular/core:signal-input-migration
```

- Vous avez plusieurs options de migrations :
 - **--best-effort-mode** : Par défaut, la **migration ignore les entrées qui ne peuvent pas être migrées en toute sécurité**. Elle tente de refactoriser le code de la manière la plus sûre possible.
 - Lorsque l'option **--best-effort-mode est activée**, la migration **s'efforce de migrer autant que possible, même si cela risque d'endommager votre build**.
 - **--insert-todos** Lorsque cette option est activée, la **migration ajoute des tâches à effectuer aux entrées qui n'ont pas pu être migrées**. Ces tâches incluent les raisons pour lesquelles les entrées ont été ignorées.
 - **--analysis-dir** : Dans les projets volumineux, vous pouvez utiliser cette option pour **réduire le nombre de fichiers analysés**.

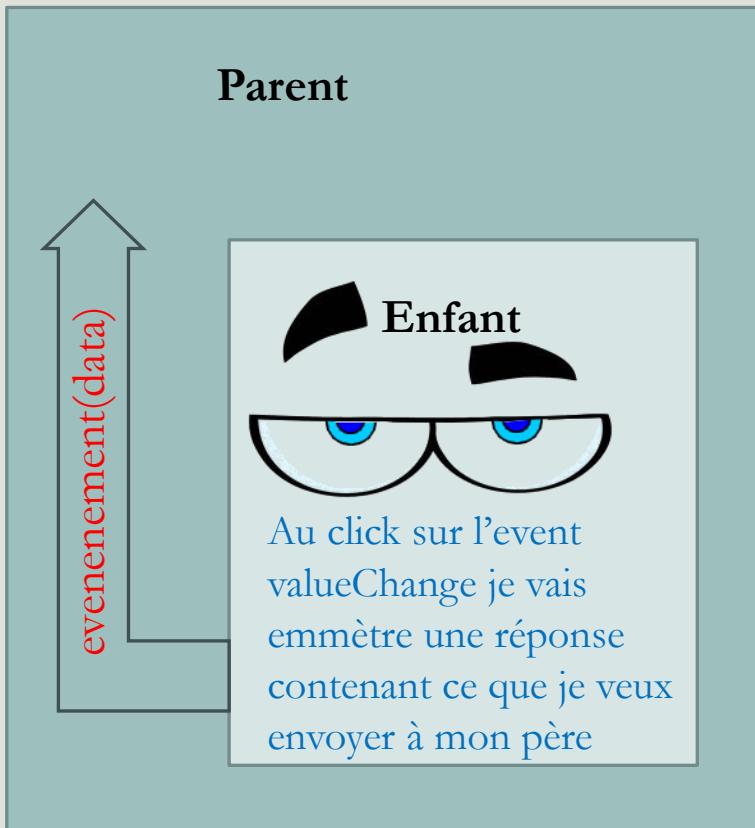
Interaction du fils vers le père

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



- L'enfant ne voit pas le parent car il ne sait simplement pas par quel composant il a été appelé.
- Solution : 1. Faire du event binding avec @output

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

@Component({
  selector: 'app-child-first',
  templateUrl: './child-first.component.html',
  styleUrls: ['./child-first.component.css']
})
export class ChildFirstComponent implements OnInit {
  @Output() sendRequest = new EventEmitter();
}
```

```
<app-fils [colour]="color"/>
```

Template du pere

Template du fils

Interaction du fils vers le père

2. Configurer l'événement

```
sendEvent() {
    this.sendRequest.emit('Accuse la réception de la couleur ' + this.color);
}
```

3. Récupérer l'événement dans le composant parent

```
<app-child-first (sendRequest)="ReceivedEvent($event)"></app-child-first>
```

4. Traiter le message reçu de la part du composant enfant

```
ReceivedEvent(msg : any)
{
    alert(msg);
}
```

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 RxJs 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

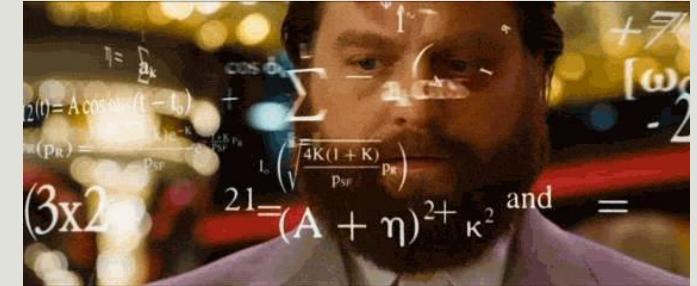
Signal Output Migration

- Vous pouvez migrer toute votre application vers les signals input via cette commande :

```
ng generate @angular/core:signal-output-migration
```

- Comme pour Input, Vous avez plusieurs options de migrations

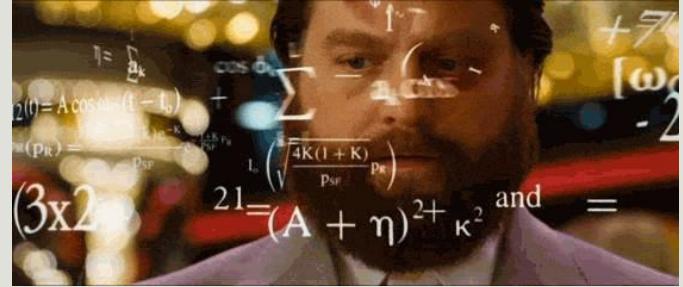
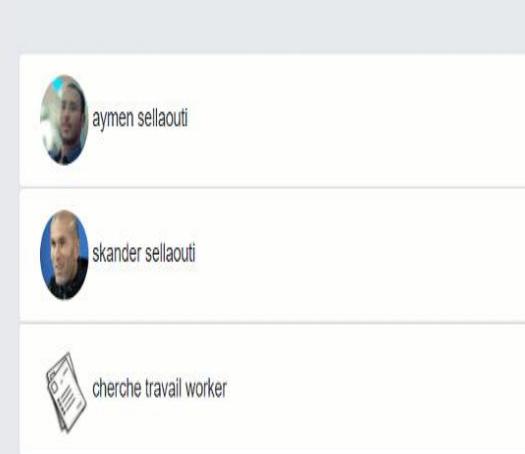
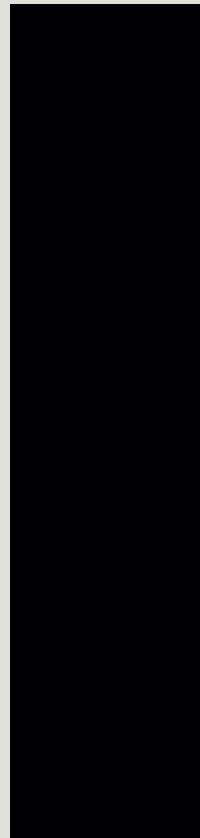
Exercice



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

Exercice

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



Exercice

- Voici la décomposition de l'interface en composant

Un cv est caractérisé par :

id

name

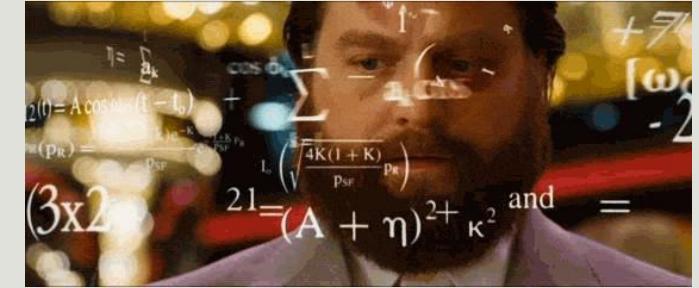
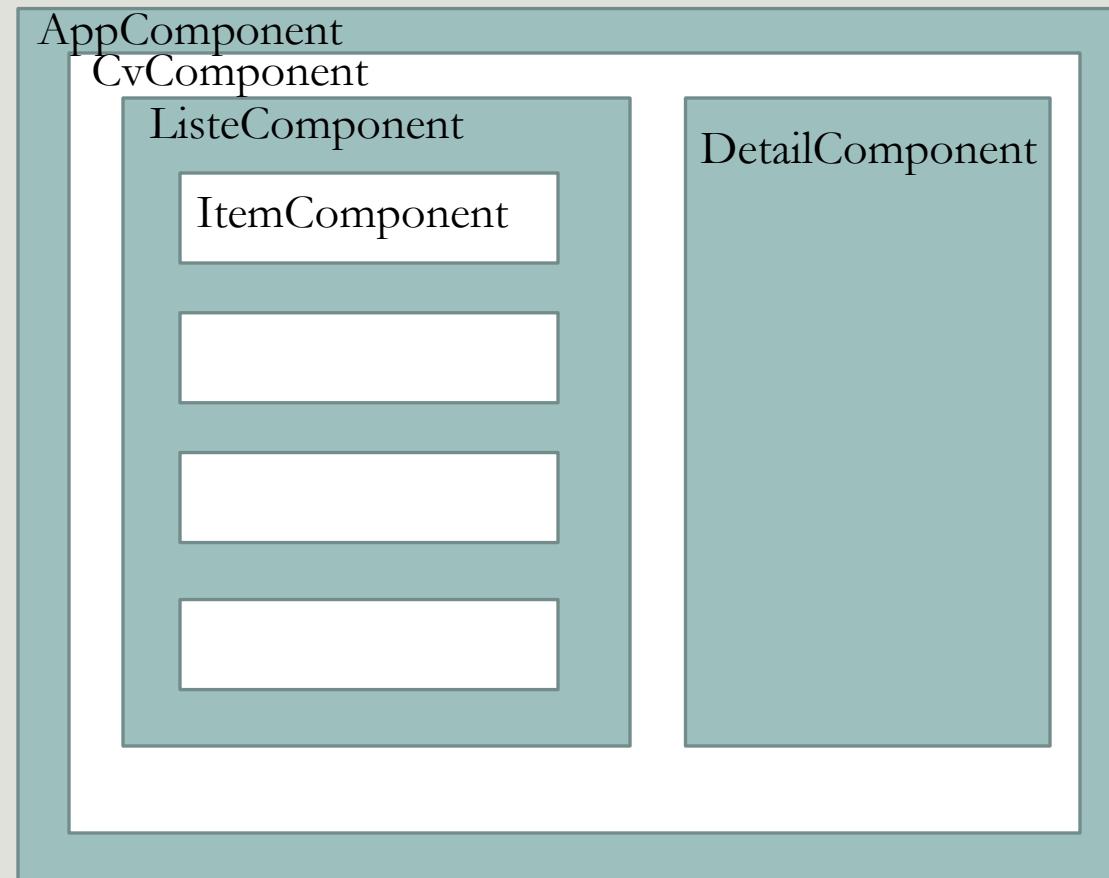
firstname

Age

Cin

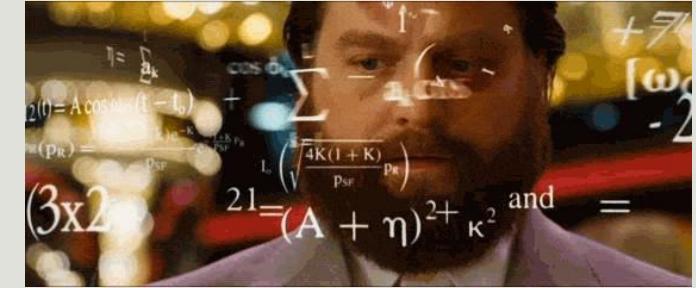
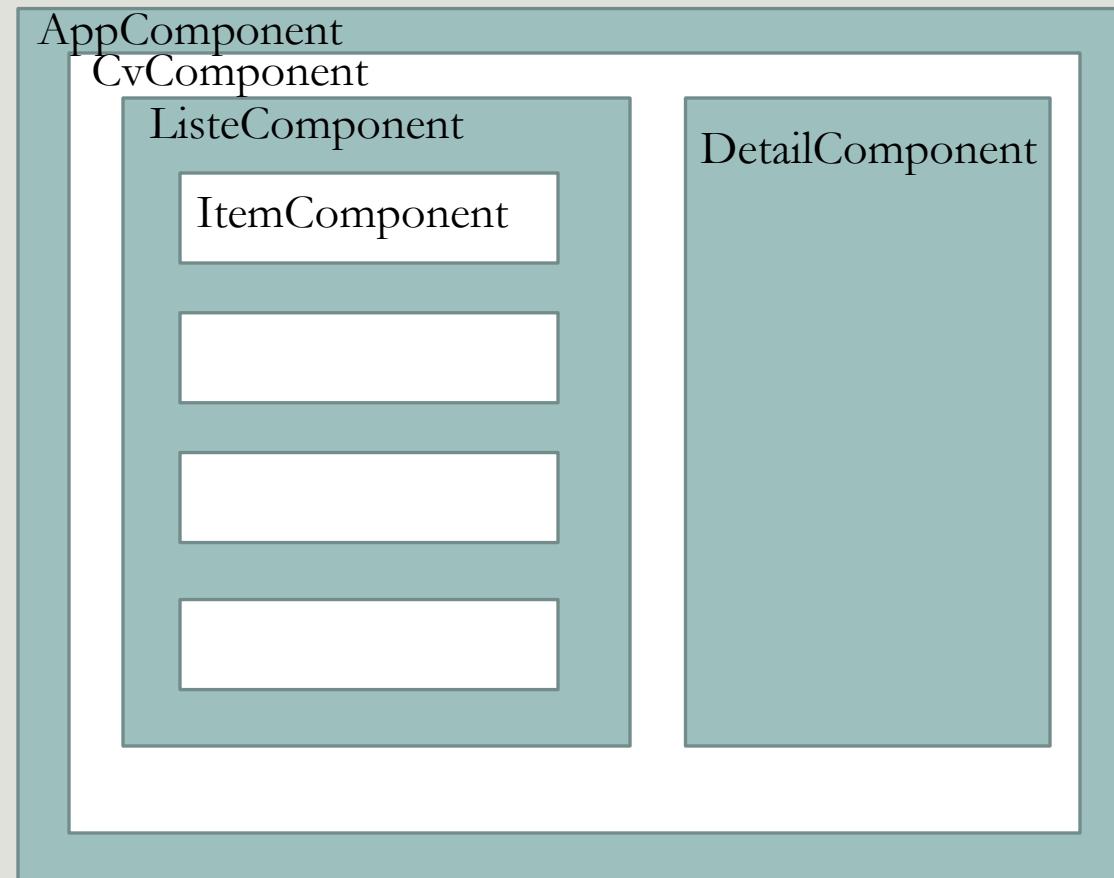
Job

path



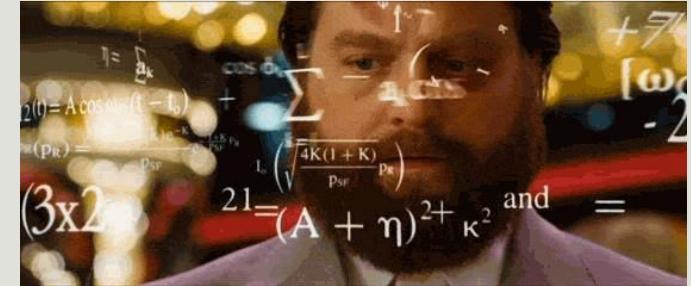
Exercice

- 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 Cv's 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

Un cv est caractérisé par :

id

name

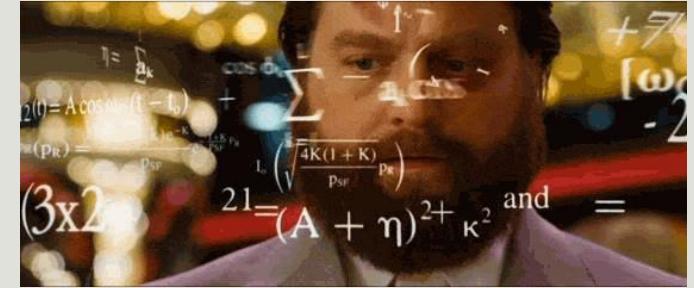
firstname

Age

Cin

Job

path



sellaouti aymen

sellaouti skander



Aymen Sellaouti
Teacher

36

Au click sur le Cv les détails sont affichés



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 annotée avec le décorateur `@Directive()`** qui permet à Angular **d'attacher un comportement spécifique à un élément HTML**.
- Les directives permettent :
 - **Réutilisabilité du code** : en centralisant un comportement ou un style **réutilisable** dans toute l'application.
 - **Séparation des préoccupations (Separation of concerns)** : La logique **métier est séparée du design** ou de l'affichage.
 - **Clarté et lisibilité** : Plutôt qu'un code lourd dans le composant, la directive gère une fonctionnalité spécifique (exemple : gestion de permissions d'un bouton).
 - **Personnalisation facile** : Tu peux créer des directives sur mesure pour enrichir le comportement de tes composants

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**.
- Il existe des **directives fournies par Angular** mais vous pouvez **créer vos propres directives**

Les directives structurelles

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

Les directives structurelles *ngIf

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

Exemple

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

Les directives structurelles *ngFor

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

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

```
<ul>
  <li *ngFor="let episode of episodes; let i = index;>
    Episode {{i+1}} {{episode.title}}
  </li>
</ul>
```

Les directives d'attribut (ngStyle) soft depreciation à partir d'NG19

- Cette directive permet de modifier **l'apparence** de **l'élément cible**.
- Elle est placée entre [] **[ngStyle]** qu'on ajoute dans la balise cible.
- Elle prend en paramètre un **attribut** représentant un **objet** décrivant le **style** à appliquer.
- Dans cet objet, la **clé** va représenter **l'attribut de style que vous voulez gérer**, e.g. color, backgroundColor, fontSize, fontFamily, border.
- La **valeur** représente **la valeur associée** à cette propriété de style et qui peut être une constante, dans ce cas elle ne change pas ou une **variable** et donc elle **suivra toujours la valeur de cette variable**.
- Dans cet exemple, l'attribut de style **color** prend sa valeur de la variable color, il sera donc 'lightblue'
- Pour l'attribut de style **fontFamily** on lui a associé une constante qui est 'garamond'.

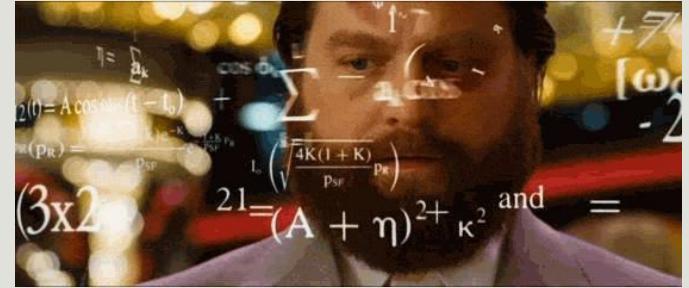
```
@Component({  
  selector: 'app-ngstyle-exemple',  
  templateUrl: './ngstyle-exemple.component.html',  
  styleUrls: ['./ngstyle-exemple.component.css']  
})  
export class NgstyleExempleComponent {  
  color = 'lightblue';  
}
```

```
<p  
[ngStyle]="{  
  backgroundColor: 'red',  
  color: color,  
  fontFamily: 'garamond'  
}"  
>ngstyle-exemple works!</p>
```

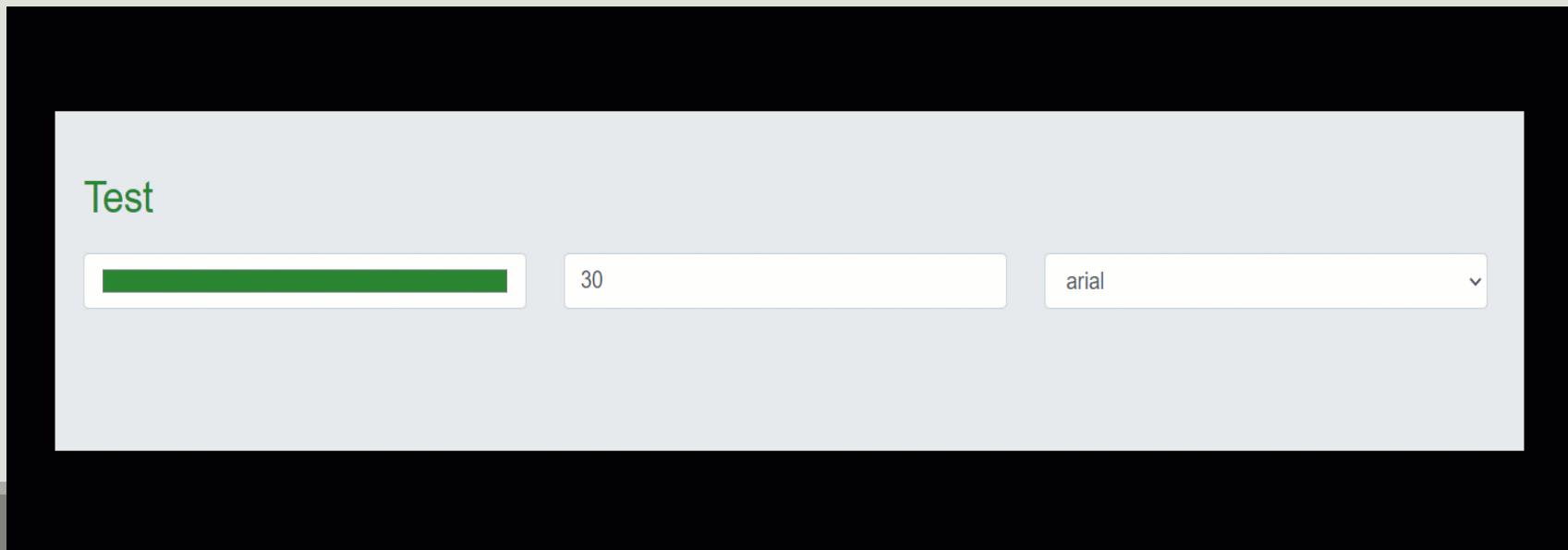
Les directives d'attribut (ngStyle) soft depreciation à partir d'NG19



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 change la couleur du color input, ça devienne la couleur du paragraphe. Et que lorsque on change le nombre dans le number input la taille de l'écriture.
- Finalement ajouter une liste et mettez-y un ensemble de police. Lorsque le user sélectionne une police dans la liste, la police dans le paragraphe change.



Les directives d'attribut (ngClass) soft depreciation à partir d'NG19

- 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**.
- Dans cet exemple la classe CSS off sera appliquée à la Div

```
@Component({  
  selector: 'app-ngclass-exemple',  
  templateUrl: './ngclass-exemple.component.html',  
  styleUrls: ['./ngclass-exemple.component.css'],  
}  
export class NgclassExempleComponent {  
  isOn = false;  
}
```

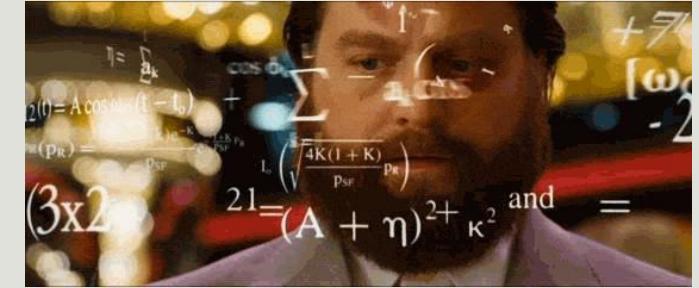
```
<div  
[ngClass]="{  
  on: isOn,  
  off: !isOn  
}"  
class="ampoule">lampe</div>
```

Les directives d'attribut (ngClass) soft depreciation à partir d'NG19

The slide displays four examples of ngClass usage, each with a green checkmark indicating it is valid:

- class expression**: `<div [class]=(true) ? 'my-class' : 'blank'>`
- class string**: `<div [class]="'my-class'">`
- class array**: `<div [class]=["user", "phone", "email"]>`
- class object**:
 - `<div [class]={
 'user': getStatus() === 'active',
 'phone': true,
 'email': false,
}>`
 - OR**
 - `<div
 [class.user]="getStatus() === 'active'"
 [class.phone]="true"
 [class.email]="false"
>`

Exercice



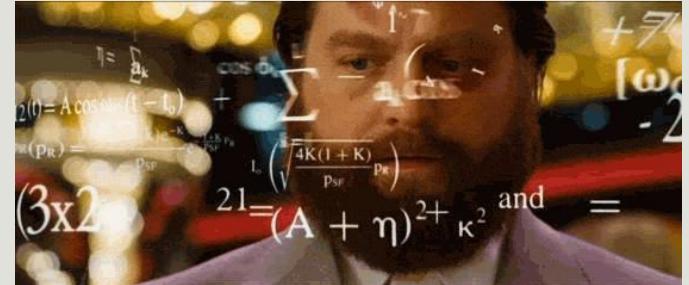
- Afin d'améliorer l'affichage de la liste des cvs dans votre CvTech, reprenez le composant cvList et faites-en sortes d'avoir les éléments successifs colorés d'une manière permutée entre deux couleurs.



Customiser une directive d'attribut

- 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";`
- Dans cet exemple on lie (bind) l'attribut **bg** de la directive à l'attribut **style.backgroundColor** de l'élément hôte (celui qui est tagué) de la directive. A chaque fois qu'on change **bg** le **style.backgroundColor** de l'élément hôte change.
- Si on veut associer un **événement** à notre directive on utilise un `HostListener` qu'on associe à un **événement** déclenchant une **méthode**.
 - Exemple : `@HostListener('mouseenter') mouseover() {`
`this.bg =this.highlightColor;`
`}`
- Afin d'utiliser le HostBinding et le HostListner il faut les importer du `core d'angular`

Exercice



Un truc plus 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.
- Préparer une méthode qui permet de générer aléatoirement une couleur dans votre directive.
- Faite en sorte qu'en appliquant votre directive à un input, à chaque écriture d'une lettre (event keyup) la couleur change en prenant aléatoirement l'une des couleurs de votre tableau. Pensez à utiliser Math.random() qui vous retourne une valeur entre 0 et 1.

The screenshot shows a browser's developer tools with the element inspector open. The selected element is an input field with the class "form-control". The element's style is defined as:

```
border-color: #rgb(125, 162, 230);  
color: #rgb(125, 162, 230);
```

The surrounding DOM structure includes a container div, an app-root component, and an app-ng-style component.

Customiser une attribut directive

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

HostBinding et HostListener

- La solution la plus appropriée est de ne plus utiliser les décorateur `@HostBinding` et `@HostListener` et de plutôt **passer par l'attribut host de votre décorateur `@Directive` ou `@Componet`**.
- Cet attribut prend en paramètre un objet dans lequel vous pouvez utiliser le binding standard

```
@Directive({
  selector: '[appHighlight]',
  standalone: true,
  host: {
    '[style.backgroundColor]': 'this.color()',
    '(mouseenter)': 'this.onMouseEnter()',
    '(mouseleave)': 'this.onMouseLeave()',
  }
})
export class HighlightDirective { //...}
```

Angular

Les pipes

AYMEN SELLAOUTI

Objectifs

1. Définir les pipes et l'intérêt de les utiliser
2. Vue globale des pipes prédéfinies
3. Créer un pipe personnalisé

Qu'est ce qu'un pipe

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

A screenshot of a web browser window. At the top, there is a black header bar. Below it, a white input field contains the text "Hello world". Underneath the input field, there are two paragraphs of text:

Avec le pipe uppercase :
Sans aucun pipe :

The bottom portion of the screenshot shows the browser's developer tools, specifically the Elements tab, displaying the HTML code for the page:

```
<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`

Exemple de pipe

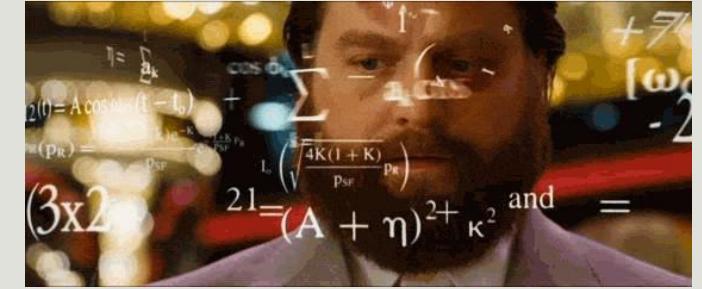
```
import { Pipe, PipeTransform } from  
'@angular/core';  
  
@Pipe({  
  name: 'team',  
})  
export class TeamPipe implements PipeTransform  
{  
  transform(value: any, args?: any): any {  
    switch (value) {  
      case 'barca':  
        return 'blaugrana';  
      case 'roma':  
        return 'giallorossa';  
      case 'milan':  
        return 'rossoneri';  
    }  
  }  
}
```

```
<ol>  
  @for ( team of teams() ; track team ) {  
    <li>  
      {{team | team}}  
    </li>  
  }  
</ol>
```

```
@Component({  
  selector: 'app-root',  
  imports: [RouterOutlet, TeamPipe],  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css'  
})  
export class AppComponent {  
  teams = signal(['milan', 'barca', 'roma']);  
}
```

1. rossoneri
2. blaugrana
3. giallorossa

Exercice



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

Angular Service et injection de dépendances



AYMEN SELLAOUTI

Objectifs

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

Comment fonctionne un restaurant ?



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

Comment fonctionne un restaurant (système d'injection de dépendances) ?

Providers

Injector

IOC

Injection



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 :
 - Provider les dépendances (**Préparer notre menu, définir quels plats nous pouvons fournir**) via **l'annotation @Injectable et son attribut providedIn** (ici les services), dans le **provider du module ou du composant** (**C'est la préparation de votre menu**).
 - Injecter le service (**commander le plat souhaité**) en le passant comme **paramètre du constructeur** de la **classe** (composant ou service) qui en a besoin ou à partir de la version 14 utiliser la fonction **inject** (**c'est la bonne pratique**).

Injection de dépendance (DI)

Provider les dépendances et les injecter (Préparer notre menu, et commander)

```
@NgModule({  
    providers: [CvService],  
    bootstrap: [AppComponent],  
})  
export class AppModule {}
```

```
@Component({  
    selector: 'app-cv',  
    templateUrl: './cv.component.html',  
    styleUrls: ['./cv.component.css'],  
    providers: [CvService]  
})  
export class CvComponent {
```

```
export const appConfig: ApplicationConfig = {  
    providers: [ CvService ],  
};
```

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

Provider

Injecter

```
export class CvComponent {  
    constructor(  
        private cvService: CvService  
    ) {}
```

ancienne

```
export class CvComponent {  
    private cvService = inject(CvService);  
}
```

nouvelle

Les providers personnalisés la fonction inject (après Angular 14)

- La fonction inject vous permet d'injecter un injectable.
- Avant Angular 14 et à partir d'Angular 9, la fonction inject pouvait être utilisé uniquement dans la factory de l'InjectionToken ou dans le factory du @Injectable.
- A partir d'Angular 14, vous pouvez l'utiliser dans tout le contexte d'injection de dépendances comme vos composants, directives et pipes.
- Le premier intérêt est le type safety avec le décorateur @Inject.
- Il facilite aussi l'héritage en externalisant la dépendance du composant.
- Faites attention quand vous utilisez inject elle doit être dans un contexte d'injection

```
export class CvComponent {  
    // Donne moi le sayHelloService  
    sayHelloService = inject(SayHelloService);  
    cvService = inject(CvService);  
}
```

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**. Ceci permet **d'optimiser votre code** via :
- **Lazy loading** : N'instancie un service que s'il est injecté au moins une fois
- Permettre le **Tree-Shaking** des services non utilisés : Si le service n'est jamais utilisé, son **code sera entièrement retiré du build final**.

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

@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.

Injection de dépendance (DI)

Avantages

- L'intérêt de l'injection de dépendance est donc :
 - **Couplage lâche**
 - Facilement **remplacer une implémentation** d'une dépendance à des fins de test
 - Prendre en charge **plusieurs environnements** d'exécution
 - Fournir de **nouvelles versions d'un service** à un tiers qui utilise votre service dans sa base de code, etc.

Standalone Component

Les composants autonomes

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 **plusieurs fonctions spécifiques** pour les **modules** les **plus utilisés** comme le **http** avec sa fonction **provideHttpClient()**, ou **provideRouter(APP_ROUTES)**.

Standalone Component

Les composants autonomes

Configurez l'injection de dépendance

```
export const appConfig: ApplicationConfig = {  
  providers: [  
    provideRouter(  
      routes,  
      withComponentInputBinding()  
    ),  
    provideHttpClient(  
      withInterceptorsFromDi()  
    ),  
    importProvidersFrom(  
      TranslateModule.forRoot({  
        loader: {  
          provide: TranslateLoader, useFactory: HttpLoaderFactory, deps: [HttpClient],  
          },  
        })  
      ),  
    ],  
  };
```

Standalone Component

Les composants autonomes

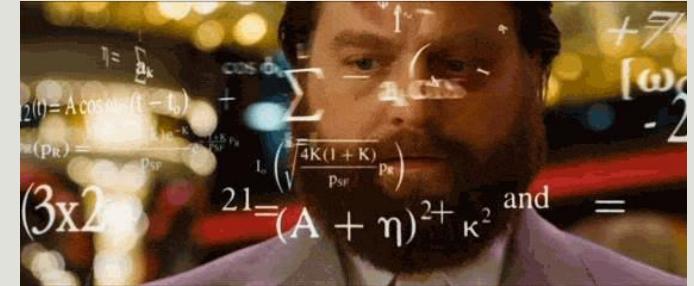
Créer un service

- Afin de créer un service, vous pouvez passer par la commande
ng g s nomService
- Créer manuellement une classe et de la fournir
 - en ajoutant `@Injectable({‘providedIn’: ‘root’})`
 - ou en la providant via un provider

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

A screenshot of a web-based application for managing todos. The interface includes a 'Name:' input field containing 'I', a 'Content:' input field, and a blue 'Add Todo' button. Below these fields is a large, empty rectangular area where todo items would be listed.

DI Hiérarchique

- Dans Angular vous disposez de plusieurs endroits où vous pouvez définir les fournisseurs pour vos dépendances :
 - Module
 - Composant
 - Directive !

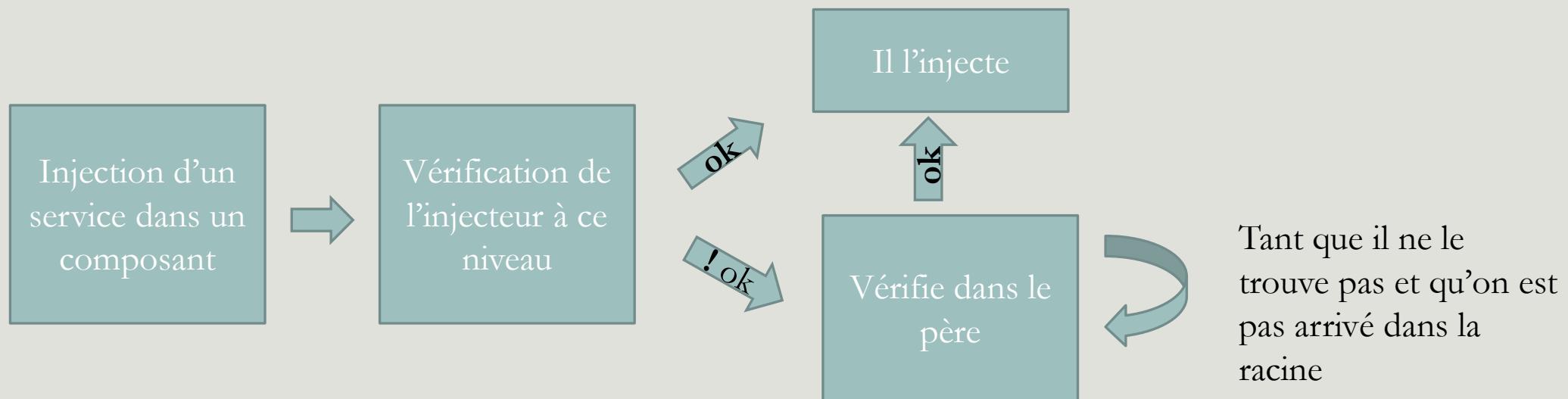
DI Hiérarchique

- Le système d'injection de dépendance d'Angular est **hiérarchique**
- Il possède **deux hiérarchies d'injecteur**
 - Une **hiérarchie d'injecteur niveau composant (element Injector Hierarchy)**
 - Une hiérarchie d'injecteur **niveau module.**
- La **hiérarchie composant** (appelé aussi **Node Injector Hierarchy**) est la plus prioritaire

DI Hiérarchique

Hiérarchie d'injecteur niveau composant

- 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 :

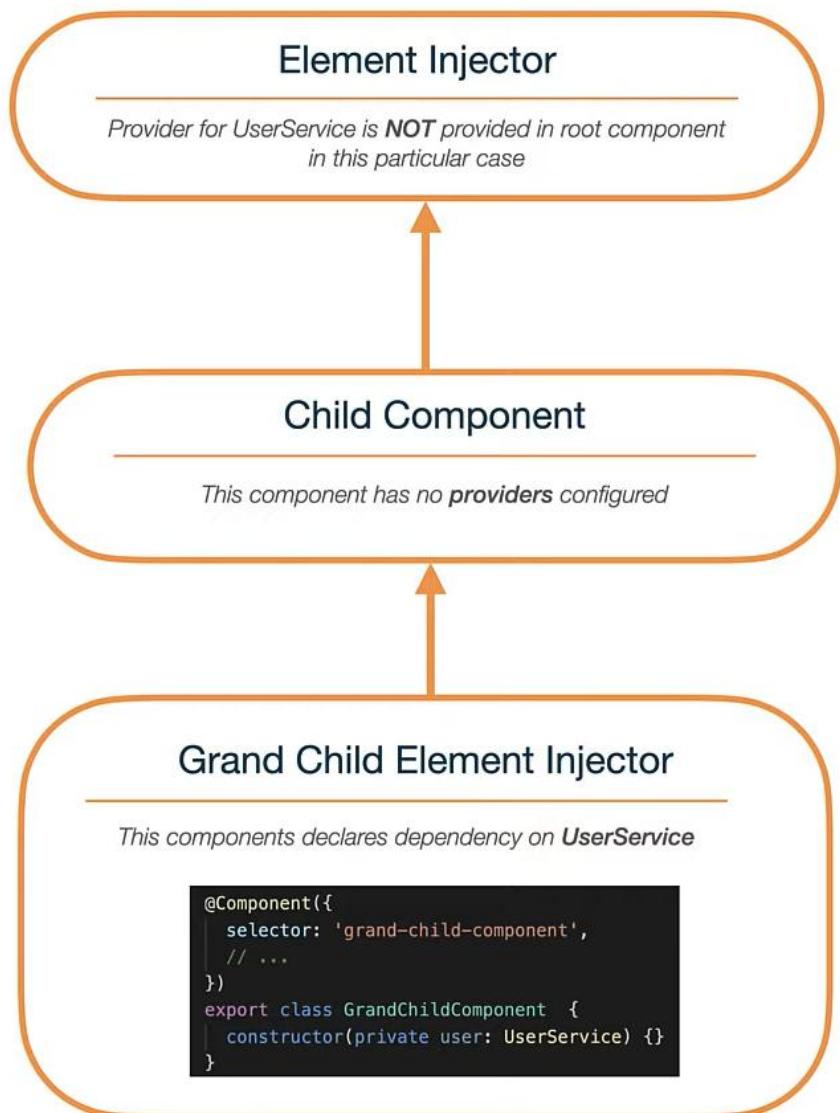


DI Hiérarchique

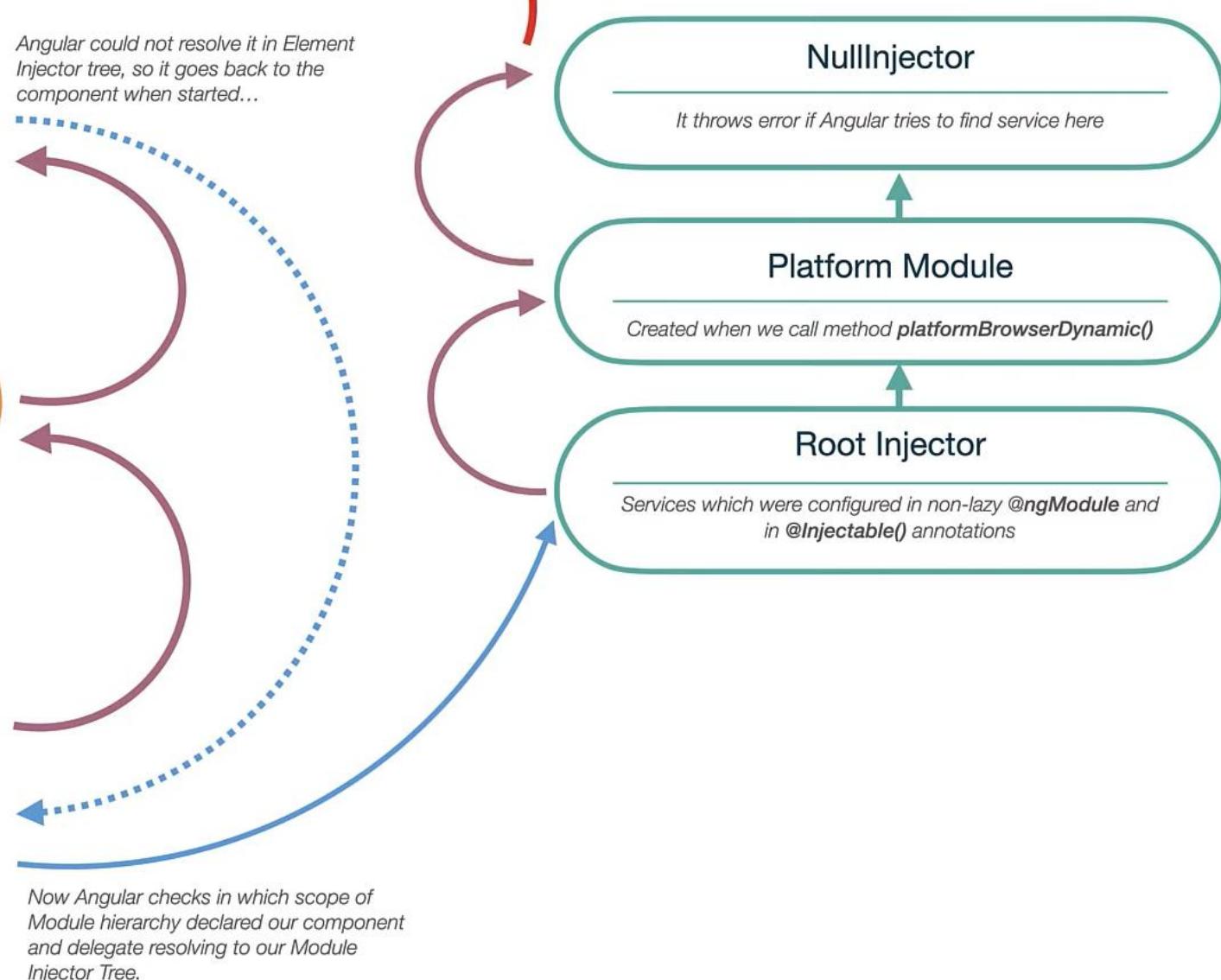
Hiérarchie d'injecteur niveau Module

- Si Angular **ne trouve pas de provider** au niveau de la **hiérarchie des composants**, il va aller voir dans la **hiérarchie de module**.
- Le ModuleInjector peut être configuré de deux manières en utilisant :
 - La propriété `@Injectable` pour référencer un **NgModule**, ou '**root**'
 - Le tableau des providers dans `@NgModule()`
- Le moduleInjector **identifie les providers disponibles** en effectuant un **aplatissement de tous les tableaux de fournisseurs** qui peuvent être atteints en suivant **les NgModule.imports de manière récursive**.

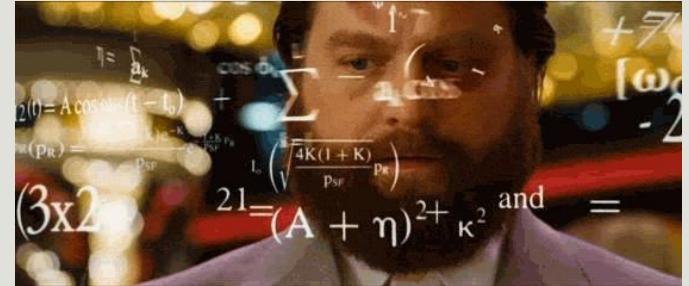
Element Injector Hierarchy



Module Injector Hierarchy

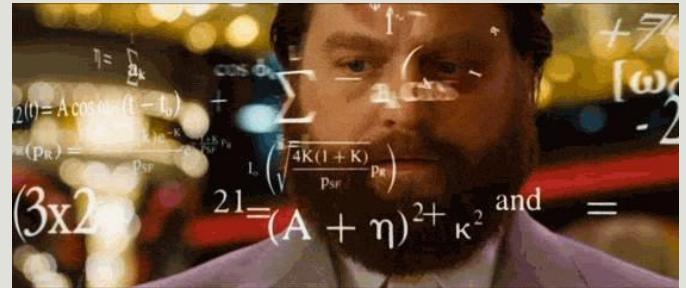


Exercice



- 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



CvTech Home Cv Add Cv Todo Mini word Color Rxjs Logout [Francais](#) [English](#)



Mandi Chaabane



Mandi Iah



Aymen Bellacouti

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

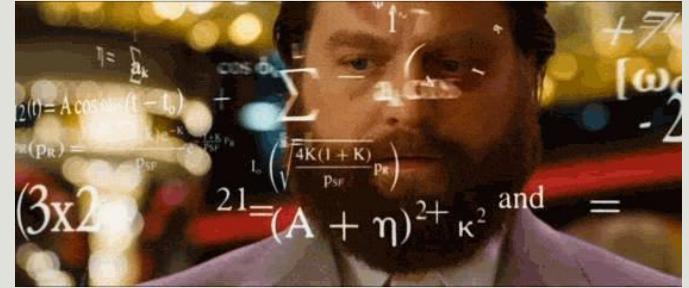
Job Description
Web design, Adobe Photoshop, HTML5, CSS3, Corel and many others...

235 Followers 114 Following 35 Projects

[embaucher](#) [détails](#)

Footer

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

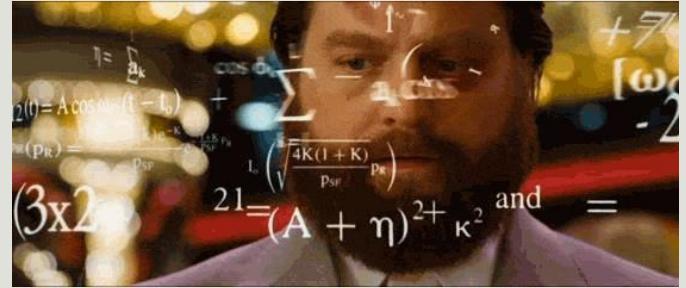
Embaucher

Liste des cvs sélectionnés pour embauche

aymen sellaouti



Exercice



A screenshot of a web browser window. The address bar shows 'www.google.com'. The search results page displays three items: 1. 'aymen sellaouti' with a small profile picture. 2. 'skander sellaouti' with a small profile picture. 3. 'cherche travail worker' with a small icon of a document. The rest of the page is mostly blank or obscured by a black redaction box.

Injection de dépendance (DI)

Deep Dive



Injection de dépendance (DI)

Que peut on injecter

Que peut on injecter ?

- Toute dépendance de votre classe, à savoir :
 - Des instances de classes
 - Des constantes

Injection de dépendance (DI)

Aller plus loin : Comment ça fonctionne

- Reprenons l'exemple de notre restaurant. Pour préparer son menu, **le chef de la cuisine a une recette associé à chacune des plats du menu**. Mais imaginons qu'un client ait envie d'un plat qui n'est pas dans le menu. Que doit-il faire ?
- **Fournir sa propre recette au chef.**
- **C'est ce qu'on appelle une providerFactory**



Injection de dépendance (DI)

Les providersFactory

- Une **providerFactory** est la **recette** permettant à votre système d'inversion de control de créer les dépendances
- Ceci implique que **pour toute dépendance** de votre application quelque soit son type, il **existe une Provider Factory** qui **sait comment la créer** et qui le fait.
- Le moyen permettant de spécifier au système d'injection de dépendance d'Angular comment créer la dépendance est la fonction **Provider factory**.
- *Une Provider factory* est simplement une **fonction** simple qu'**Angular** peut **appeler** afin de **créer une dépendance**.
- Cette fonction peut être **créeée implicitement par Angular** en utilisant quelques conventions simples (c'est une recette du menu connues par le chef, c'est le cas le plus répondu) ou **par vous-même (votre propre recette)**.

Injection de dépendance (DI)

Exemple d'un provider Factory

```
function todoServiceProviderFactory(): TodoService {
    return new TodoService();
}

function todoServiceProviderFactory(http:HttpClient): TodoService {
    return new TodoService(http);
}
```

Injection de dépendance (DI)

Associer votre Factory Provider à Angular

Token

- Maintenant il reste à dire à Angular **quand utiliser ce provider**.
- Donc, nous devons répondre à cette interrogation : **Comment Angular sait-il quoi injecter**, et donc **quelle Provider factory appeler pour créer quelle dépendance (Donc quelle recette suivre ?)**?
- Pour ce faire, vous pouvez utiliser des **Tokens (C'est les noms des plats de votre menu, ils identifient un plat)**.
- Un Token peut avoir plusieurs formes et il a pour **rôle d'identifier une Provider Factory**.

Injection de dépendance (DI)

Associer votre Factory Provider à Angular

Angular injection token

- La première forme de Token est l'**Angular injection token**
- C'est une instance de la class **InjectionToken**
- Son rôle est **d'identifier le service** dans le système d'injection de dépendance

```
export const TODOS_SERVICE_TOKEN =  
  new InjectionToken<TodoService>("TODO_SERVICE_TOKEN");
```

Injection de dépendance (DI)

Associer votre Factory Provider à Angular

Configurer le Provider

- Maintenant que nous avons notre Provider Factory et notre Token, nous devons **configurer** Angular pour qu'ils les prennent en considération.
- Ceci sera fait à travers le **Provider** qui n'est qu'un **objet de configuration**.
- Il peut prendre en paramètres 3 clés (pas que):
 - **provide**: qui est notre Token
 - **useFactory**: qui est notre Factory
 - **deps**: un tableau des dépendances de votre factory

Injection de dépendance (DI)

Associer votre Factory Provider à Angular

Configurer le Provider

```
export const TODOS_SERVICE_TOKEN =
  new InjectionToken<TodoService>("TODO_SERVICE_TOKEN");
function todoServiceProviderFactory(http:HttpClient): TodoService {
  return new TodoService(http);
}
```

```
providers: [
  {
    provide: TODOS_SERVICE_TOKEN,
    useFactory: todoServiceProviderFactory,
    deps: [HttpClient]
  }
]
```

app.config.ts

Injection de dépendance (DI)

Associer votre Factory Provider à Angular

Injecter la factory

- Il reste une dernière étape, à savoir **comment injecter notre dépendance dans notre classe.**
- On utilise le décorateur **@Inject** au niveau du **constructeur** et on lui passe le **Token du Provider Factory** que nous voulons injecter.

```
constructor(  
    @Inject(TODOS_SERVICE_TOKEN) private todoService: TodoService  
){}
```

Les providers personnalisés

Les autres formes de Tokens

- Comme nous l'avons présenté, le Token peut avoir plusieurs formes. Parmi elles, le **nom de la classe**.
- Le token peut être aussi une **chaine de caractères**, mais ceci est déconseillé afin d'éviter les collisions de noms.
- Le **TOKEN** doit être **unique pour éviter toute collision**, les **Provider factory** sont **stockés dans une map**, et si le provider est simple et qu'il a le même nom, la map ne contiendra que **le dernier provider** défini.

Les providers personnalisés

Les autres formes de Tokens

```
providers: [
  {
    provide: TodoService,
    useFactory: todoServiceProviderFactory,
    deps: [HttpClient]
  }
],
```

```
constructor(
  @Inject(TodoService) private todoService: TodoService
) {}
```

Les providers personnalisés

useClass

- Une autre option s'offre à vous, et au lieu de spécifier la Fonction du Provider Factory avec `useFactory`, vous pouvez utiliser la clé **useClass**.
- En utilisant **useClass**, Angular saura que la valeur que nous transmettons est un **constructeur valide**, qu'Angular peut simplement **appeler en utilisant l'opérateur new**.

Les providers personnalisés

useClass

```
providers: [
  {
    provide: TodoService,
    useClass: TodoService,
    deps: [HttpClient]
  }
],
```

```
constructor(
  @Inject(TodoService) private todoService: TodoService
) {}
```

Les providers personnalisés

useClass

- Une autre fonctionnalité très pratique de **useClass** est que pour ce type de dépendances, Angular **essaiera de déduire le Token d'injection au moment de l'exécution** en fonction de la **valeur des annotations de type Typescript**.
- Cela signifie qu'avec les dépendances useClass, nous n'avons même **plus besoin du décorateur Inject**, ce qui explique pourquoi vous le voyez rarement.

Les providers personnalisés

useClass

```
providers: [  
  {  
    provide: TodoService,  
    useClass: TodoService,  
    deps: [HttpClient]  
  }  
,
```

Le Token est déterminé par
Angular en utilisant le Type
TodoService

```
constructor(private todoService: TodoService) {}
```

Les providers personnalisés

useClass

- En utilisant le décorateur `@Injectable`, votre Provider devient encore plus simple puisqu'**on n'a plus à spécifier les dépendances qui seront directement déterminé au niveau du constructeur** par le Système d'Injection de dépendance d'Angular

```
providers: [
{
  provide: TodoService,
  useClass: TodoService,
  deps: [HttpClient]
},
],
```

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

```
providers: [
{
  TodoService,
}
],
```

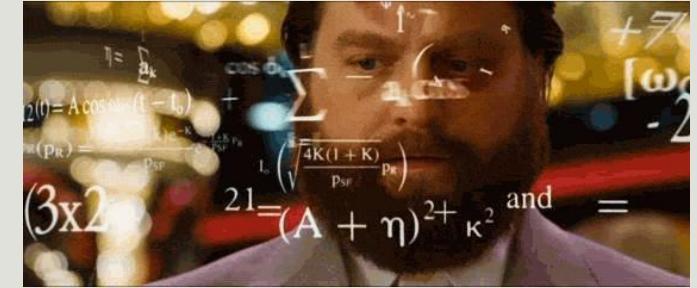
Les providers personnalisés

useClass

- La syntaxe **useClass** est aussi utile pour injecter dynamiquement une classe.
- Imaginez que le service de log dépend de l'environnement de développement.

```
@Module({
  providers: [
    {
      provide: LoggerService,
      useClass:
        config.env === 'development'
          ? DevelopmentLoggerService
          : ProductionLoggerService,
    }
  ],
}) export class AppModule {}
```

Exercice



- Nous supposons que votre API est encore en phase de teste ou est en maintenance. Vous voulez continuer à travailler avec votre CvComponent.
- Créer un service FakeCvService
- Faites en sorte d'avoir une fonction getCvs qui retourne un tableau de cvs
- Dans votre cvComponent **provider** ce **fake service** à la place du vrai **en vous basant sur une configuration**.
- Vérifier le bon fonctionnement
- Remarque: pour créer un observable, vous pouvez utiliser l'opérateur de création of.

Autres providers

- Dans certains cas d'utilisation, l'utilisation standard des providers ne convient pas, imaginer l'un des cas suivants :
 - Vous souhaitez créer une instance personnalisée au lieu de laisser le container le faire pour vous.
 - Vous voulez **injecter une bibliothèque externe**
 - Vous voulez **mockez une classe pour le teste**
 - Vous voulez injecter des **instances différentes** selon **le contexte** ...
- Angular nous permet de définir des providers particuliers selon votre besoin.

Les providers personnalisés

useValue

- La syntaxe **useValue** est utile pour injecter
- Une valeur constante,
- Une bibliothèque externe
- Remplacer une implémentation réelle par un objet fictif.

```
providers: [
  {
    useValue: [{ lundi: 'Angular' }, { mardi: 'Still Angular' }],
    provide: 'TODOS_LIST',
  },
  TodoService,
],
```

Les providers personnalisés useValue

- Si vous injecter une classe, l'utilisation de l'injection via le constructeur reste d'actualité.
- Sinon, pour injecter ce provider utiliser la syntaxe **@Inject, qui prend en paramètre le Token ou la fonction inject en dehors du constructeur**

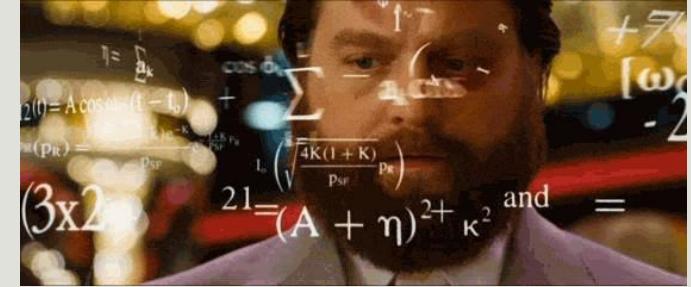
```
providers: [
  {
    provide: 'TODOS_LIST',
    useValue: [{ lundi: 'nestJs' }, { mardi: 'Still NestJs' }],
  },
  TodoService,
],
```

```
constructor(
  @Inject('TODOS_LIST') todoList,
) {
  console.log('Fake Todo List', todoList);
}
```

```
@Controller('todo')
export class TodoController {
  todoList = inject('TODOS_LIST');
```

Exercice

- Provider la fonction uuid
- Faite en sorte de l'utiliser dans le TodoService



Contexte d'injection

- Le **contexte d'injection** est **l'environnement dans lequel Angular peut résoudre les dépendances via son système d'injection** (injecteurs).
- Cela inclut les **constructeurs** de classes (e.g. composants, services, directives), les **initialisations de propriétés**, et certaines fonctions spécifiques (e.g. resolver, guard, intercepteur).
- Le système DI d'Angular repose sur un **injecteur (instance de Injector)**, qui **maintient un registre des fournisseurs (providers) pour résoudre les dépendances**.
- Sachant que Le DI d'Angular est hiérarchique : chaque composant, module, ou environnement a son propre injecteur, et la résolution des dépendances dépend de la hiérarchie (élément → parent → module → racine → plateforme → null injector).

Inject et le Contexte d'injection

- La fonction **inject** doit être exécutée dans un contexte d'injection, car elle a besoin d'accéder à l'injecteur actif pour résoudre le token demandé. Sans ce contexte, Angular ne sait pas quel injecteur utiliser, ce qui entraîne une erreur (NG0203: inject() must be called from an injection context).
- Les contextes valides incluent :
 - **Constructeur** d'une classe gérée par le DI (composants, services, directives, pipes).
 - **Initialiseur de champ** dans ces classes.
 - **Fonction d'usine** pour un Provider ou un InjectionToken (via useFactory).
 - **Fonctions exécutées via runInInjectionContext** (pour créer un contexte manuellement).
 - **API spécifique**, comme les gardes de route ou les résolveurs, qui s'exécutent dans un contexte d'injection.

Les apis actuels computed et effect permettent-ils de tout gérer ?

- Les **computed Signals** sont en **lecture seule** : Les Signals dérivés via `computed()` permettent de **calculer des valeurs basées sur d'autres Signals**, mais ils **ne sont pas modifiables**. Cela posait **problème pour des cas où un état devait être à la fois dérivé d'un autre Signal et modifiable par l'utilisateur ou une logique applicative**.
- Les **effect ne sont pas idéaux pour modifier des états** : Bien que les effects **permettent de réagir aux changements de Signals**, leur utilisation pour mettre à jour d'autres Signals peut entraîner des problèmes comme des erreurs **ExpressionChangedAfterItHasBeenChecked** ou des **cycles de détection de changements inutiles**.
- **Complexité dans les cas d'états dépendants** : Dans des **scénarios où un état doit être réinitialisé ou synchronisé avec un autre Signal tout en restant modifiable**, les développeurs devaient souvent recourir à des **solutions verbeuses ou à des contournements complexes**, ce qui rendait le **code moins clair et plus sujet aux erreurs**.

linkedSignals

- Etudions ce code ensemble. Ici, nous souhaitons récupérer deux inputs permettant à l'utilisateur de spécifier la couleur d'entrée et de sortie d'un élément.
- En même temps nous souhaitons initialiser le signal bgc avec cette valeur.
- Ici l'utilisation d'un computed ne permet pas de modifier bgc ultérieurement. On se trouve donc avec deux solutions verbeuses ou à des contournements complexes.
- Ici les effects sont utilisés à contre nature, on y modifie un autre signal. Et on ne veut plus des hooks du cycle de vie. C'est dans ce cadre que les linkedSignal on été introduite.

```
export class HighlightDirective implements  
OnInit{  
  ngOnInit(): void {  
    this.bgc.set(this.out());  
  }  
  private element = inject(ElementRef);  
  in = input('yellow');  
  out = input('red');  
  bgc = signal('');
```

```
export class HighlightDirective {  
  in = input('yellow');  
  out = input('red');  
  bgc = signal('');  
  initiateBgcEffect = effect(() => {  
    untracked(() => {  
      this.bgc.set(this.out());  
    })  
  })
```

linkedSignals

- Un **linkedSignal** est un **writable signal** qui **se met à jour à chaque fois que les signals dont il dépend change**.
- **Un linked Signal** permet de simplifier la gestion des états dépendants dans des scénarios où un **Signal doit être à la fois réactif et modifiable**, rendant le code plus clair, plus maintenable et plus performant.
- Un linkedSignal est donc un signal :
 - **Réactif** : Il **se met à jour automatiquement en fonction d'un ou plusieurs Signals sources**.
 - **Modifiable** : Il **peut être modifié** directement via des méthodes comme **.set()** ou **.update()**.
 - **Capable de préserver un contexte** : Il permet **d'accéder à la valeur précédente du Signal et de sa source** pour des mises à jour intelligentes.

linkedSignals

- Un **linkedSignal** est une fonction peut être modélisée de deux manière en prenant en paramètre:
 - Une fonction
 - Un objet
- Pour les **cas simples ou la fonction références tous les signaux dont dépend votre linkedSignal préférer l'utilisation d'une fonction**

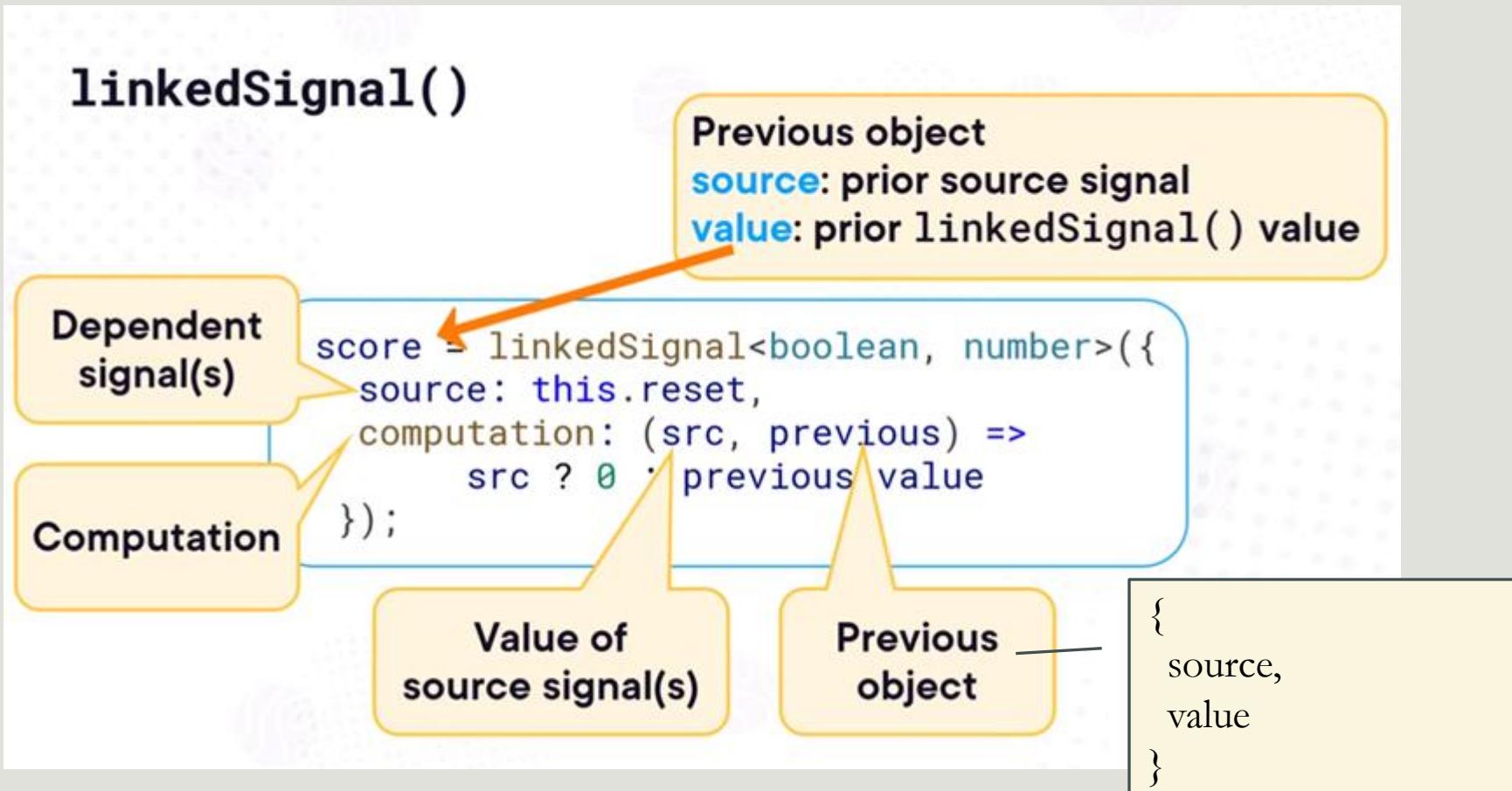
```
bgc = linkedSignal(() => this.out());
```

- Dans cet exemple, bgc est un linkedSignal qui aura comme valeur initiale this.out et qui pourra être modifié.

linkedSignals

- Pour les cas où
 - Le calcul **dépend de l'ancienne valeur du linkedSignal ou du signal source dont il dépend**
 - **La fonction ne référence pas le signal dont elle dépend**, par exemple elle retourne une constante que l'élément sélectionné change.
- Préférer l'utilisation d'un objet en paramètre de votre linkedSignal. Il prend en paramètre :
 - **source** : Le Signal (**ou une fonction retournant un Signal ou un objet des signaux dont vous dépendez**) qui déclenche la mise à jour du linkedSignal. **Cela peut être tout type de signal.**
 - **computation** : Une fonction qui définit comment la valeur du linkedSignal est calculée à partir de la source. Elle reçoit :
 - La nouvelle valeur de la source.(Optionnel)
 - Un objet **previous** contenant :
 - **previous.source** : L'ancienne valeur de la source.
 - **previous.value** : L'ancienne valeur du linkedSignal.
- Remarque : Si vous utilisez previous, il **faut spécifier les types génériques explicitement (linkedSignal<S, D>)**.
- **equal** (optionnel) : Une **fonction d'égalité personnalisée pour déterminer si la nouvelle valeur calculée est différente de l'ancienne, afin d'éviter des mises à jour inutiles**. Par défaut, Angular utilise une **comparaison stricte (====)**.

linkedSignals



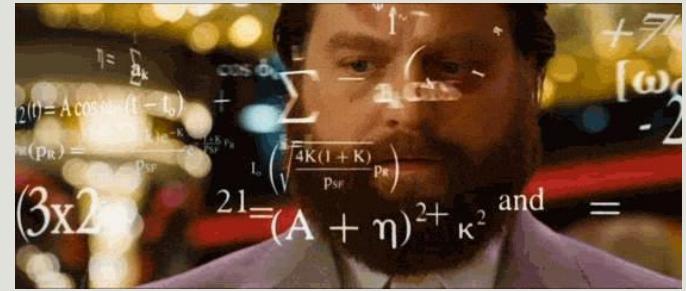
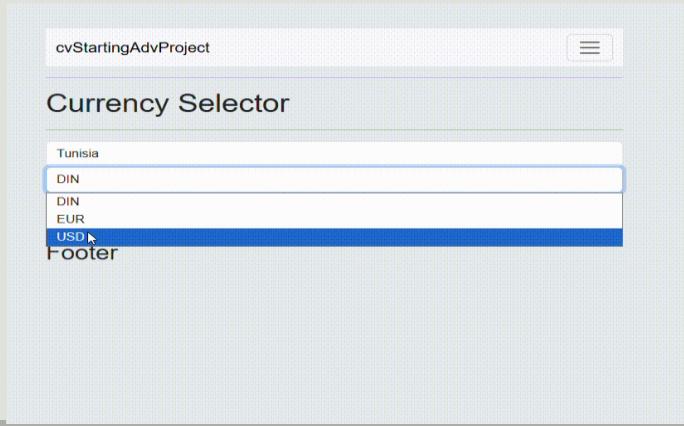
linkedSignals

Quand utilisez un computed et quand utiliser un linkedSignal

- Utilisez un **computed** quand
 - vous créer un **signal à partir d'un ou plusieurs autres signaux**
 - la valeur est **calculée automatiquement quand ces éléments changent et pas autrement**
 - la valeur est en **lecture seule**
- Préférer l'utilisation d'un **linkedSignal**. Il prend en paramètre :
 - vous créer un **signal à partir d'un ou plusieurs autres signaux**
 - Le calcul **dépend de l'ancienne valeur**
 - La valeur est en **lecture et écriture**

linkedSignal

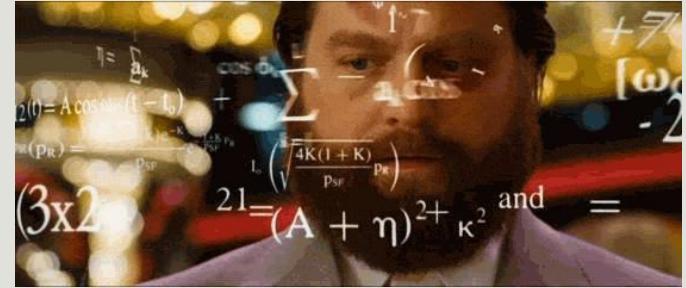
- Nous avons un composant CurrencySelector
- Nous souhaitons qu'à chaque sélection d'un pays, la liste des monnaies associés s'affiche dans le second select.
- Il faudra que l'élément affiché dans la liste des monnaies soit :
 - La même monnaie sélectionnée auparavant si elle existe dans le nouveau pays sélectionné
 - La première monnaie si l'ancienne valeur n'existe pas



```
export const countries: Country[] = [  
  {  
    name: 'Tunisia',  
    defaultCurrency: 'DIN',  
    acceptedCurrencies: ['DIN', 'EUR', 'USD'] },  
  {  
    name: 'France',  
    defaultCurrency: 'EUR',  
    acceptedCurrencies: ['EUR', 'USD'],  
  },  
  {  
    name: 'UK',  
    defaultCurrency: 'GBP',  
    acceptedCurrencies: ['GBP', 'USD'] },  
  {  
    name: 'UAE',  
    defaultCurrency: 'AED',  
    acceptedCurrencies: ['AED', 'EUR', 'USD'],  
  },  
  {  
    name: 'USA',  
    defaultCurrency: 'USD',  
    acceptedCurrencies: ['USD'] },  
];
```

linkedSignal

- Transformer le composant ProductFilterComponent en utilisant les apis des signaux



products-filter LinkedSignal Example

Search products

All

Reset

- Angular T-shirt (Clothing)
- Signal Sticker (Accessories)
- RxJS Mug (Accessories)
- Zone-Free Hoodie (Clothing)
- Nglf Coffee Cup (Accessories)
- TypeScript Socks (Clothing)
- CDK Tote Bag (Accessories)
- Material Mousepad (Accessories)
- Zoneless Snapback (Clothing)
- Angular Laptop Sleeve (Accessories)
- Component Anatomy Poster (Decor)
- Directive Decal Pack (Accessories)
- CLI Enamel Pin (Accessories)
- Angular Water Bottle (Accessories)
- Zone-Free Mug (Accessories)

Angular Routing

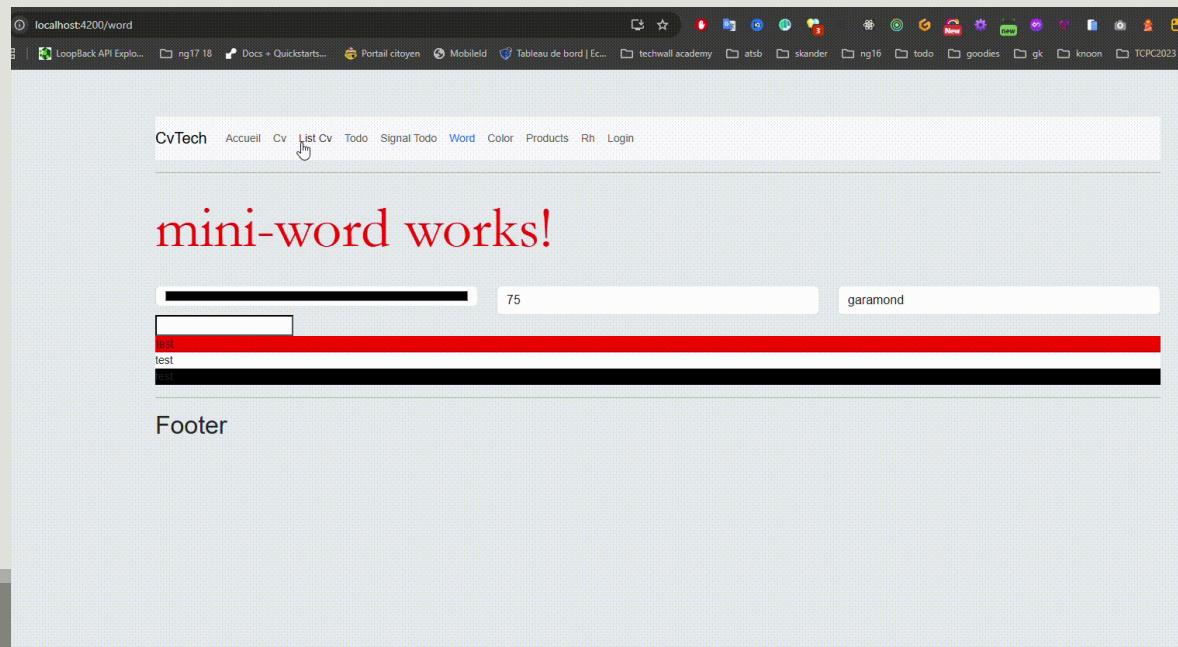
AYMEN SELLAOUTI

Objectifs

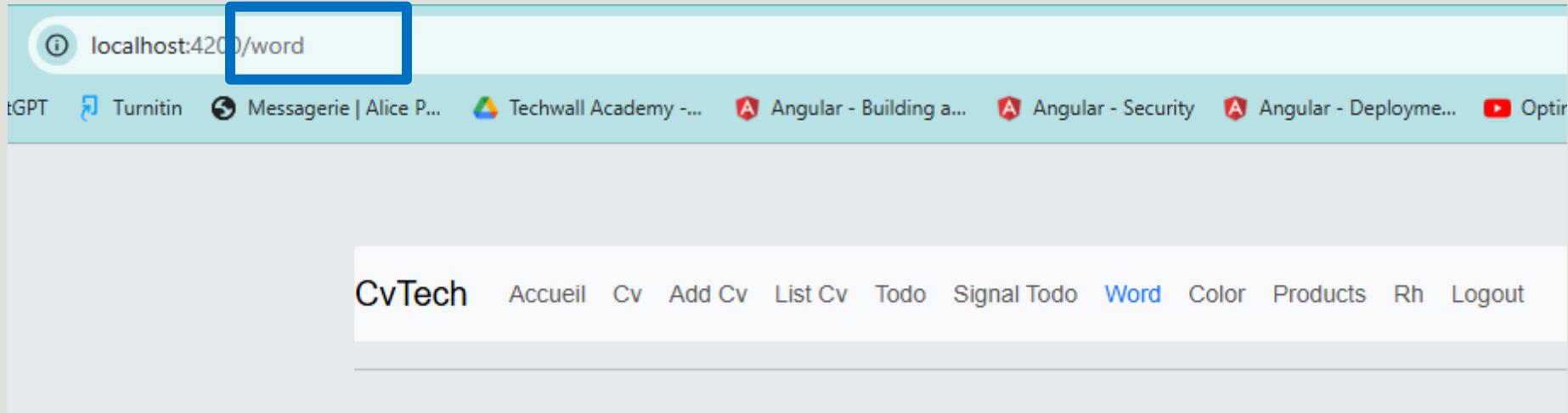
1. Définir le routeur d'Angular
2. Définir une route
3. Déclencher une route à partir d'un composant
4. Ajouter des paramètres à une route
5. Récupérer les paramètres d'une route à partir du composant.
6. Préfixer un ensemble de routes
7. Gérer les routes inexistantes

Comment fonctionne le système de routing d'Angular et quels sont ses composantes

- Quand vous **naviguez dans un site web**, c'est votre URI qui vous permet de demander **quelle page (ressource) afficher**.
- A chaque fois que vous changer d'URI, Angular détecte ce changement et essaye de déterminer à quoi correspond l'URI demandée. Pour ce faire nous avons besoin de deux choses :
 - Un **responsable** de cette tâche : c'est le **RouterModule**
 - De définir cette **relation URI, Composant** pour que le RouterModule sache quoi faire : c'est votre **route**.



Création d'un système de Routing Application modulaire



```
@NgModule({
  imports: [
    RouterModule.forRoot(routes),
  ],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

```
const routes: Routes = [
  { path: 'word', component: FirstComponent },
  { path: 'word', component: MiniWordComponent },
  { path: 'color', component: ColorComponent },
];
```

Syntaxe minimaliste d'une route

- Une route est un objet.
- Les deux propriétés essentielles sont `path` et `component`.
- `Path` représente l'URI
- `component` permet de spécifier le composant à exécuter.

```
const routes: Routes = [
  { path: '', component: FirstComponent },
  { path: 'word', component: MiniWordComponent },
  { path: 'color', component: ColorComponent },
];
```

Création d'un système de Routing Application Standalone

- Afin de créer un système de routing, il suffit de provider vos routes avec la fonction **provideRouter** au niveau de la fonction **bootstrapApplication**.
- Vous ne devez plus importez le **RouterModule**, c'est la fonction **provideRouter** qui s'en charge.

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

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

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

Création d'un système de Routing

Application Standalone

provideRouter

- La fonction **provideRouter** ne prend pas uniquement en charge les **routes** mais elle permet aussi l'implémentation des fonctionnalités additionnelles du routeur et qui étais définies auparavant dans le **RouterModule** via la méthode **forRoot**.
- Les options introduites dans les fonctions **provideQuelqueChose** suivent aussi un pattern de nommage : **withQuelqueChose**
- **withDebugTracing()**, permet par exemple de déclencher le **débogage des évènements du routeur**.
- Un autre avantage de ce pattern est de **faciliter le tree shaking** et donc **l'élagage du code mort** qui est plus simple avec les fonctions

Création d'un système de Routing

Application Standalone

provideRouter

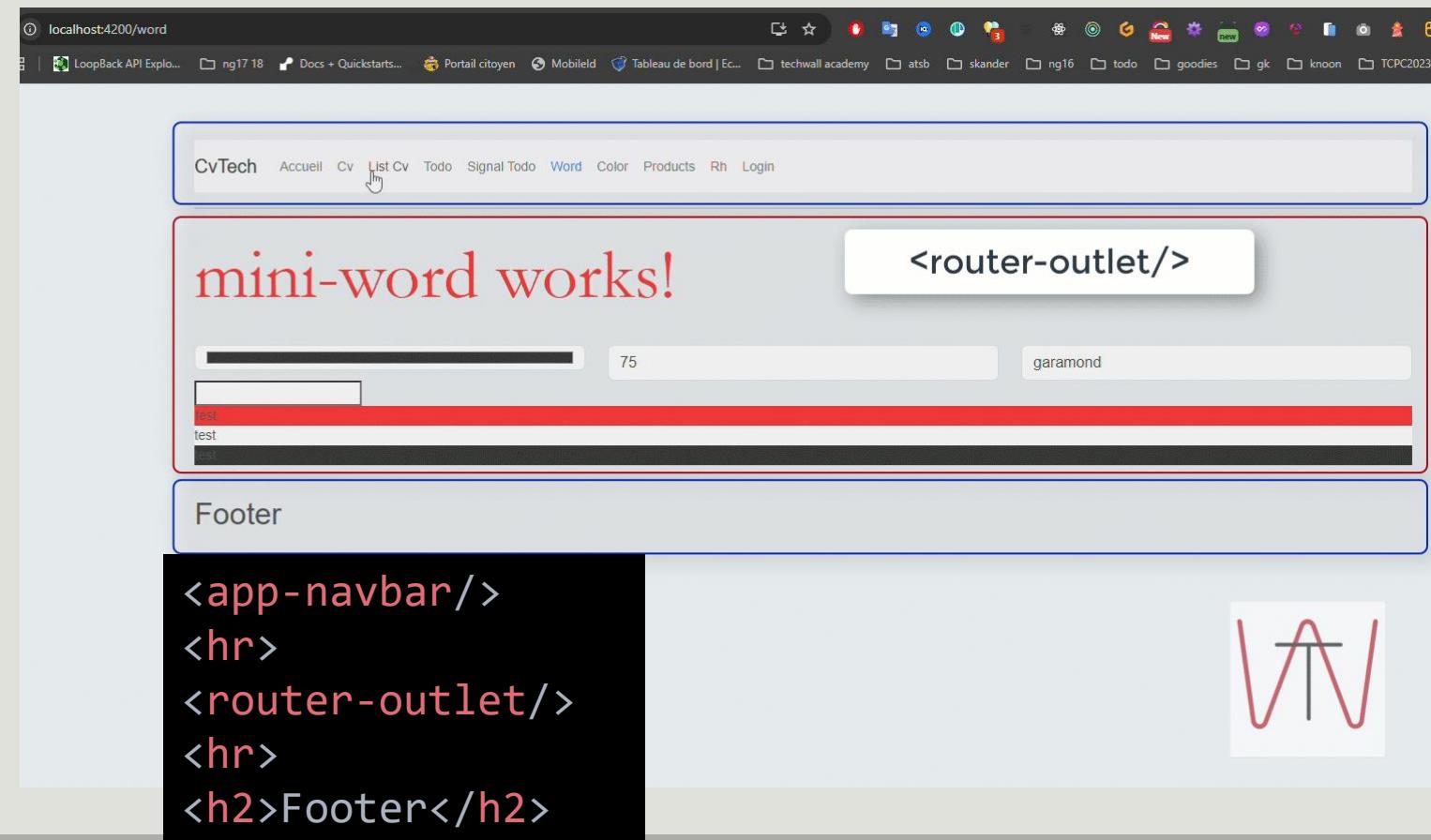
Enables logging of all internal navigation events to the console. Extra logging might be useful for debugging purposes to inspect Router event sequence.

```
}, @usageNotes
{
  Basic example of how you can enable debug tracing:

  const appRoutes: Routes = [];
  bootstrapApplication(AppComponent,
    {
      providers: [
        provideRouter(appRoutes, withDebugTracing())
          ],
        withDebugTracing()
      ),
    
```

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

- Généralement dans vos sites ou applications Web vous avez un template que vous suivez.
- Ce template contient une **partie statique** et une **partie variable**.
- La partie variable sera celle où Angular devra afficher le composant associé à la route.
- Pour indiquer cette partie variable vous devez utiliser la directive `<router-outlet>`.



Création d'un système de Routing Application Standalone

- Maintenant et dans les **différents composants** qui vont utiliser les **éléments du module de routing**, vous avez deux choix.
- **Importer le RouterModule**
- Ceci fonctionne **vous pouvez utiliser le router-outlet**, mais **ce n'est pas la meilleure façon de faire**, ca n'explique pas de quoi dépend exactement votre AppComponent.

```
@Component({  
  selector: 'app-root',  
  template:  
    `<router-outlet></router-outlet>  
    `,  
  styleUrls: ['./app.component.css'],  
  imports: [RouterModule],  
  standalone: true,  
})  
export class AppComponent {
```

Création d'un système de Routing Application Standalone

- Importer uniquement ce dont vous dépendez, ce que vous avez besoin d'utiliser, n'oubliez pas il n'y a pas que les Standalone Components, on parle aussi de **Standalone Directive et Pipe** et c'est le même modèle mental.

```
@Component({
  selector: 'app-root',
  template: `
    <router-outlet></router-outlet>
  `,
  styleUrls: ['./app.component.css'],
  imports: [RouterOutlet],
  standalone: true,
})
export class AppComponent {
```

Déclencher une route routerLink

routerLink : DIRECTIVE VS PROPERTY

➤ **RouterLink** a deux variantes : une directive et une property de la directive.

1. On opte pour la directive quand la route cible est statique (sans paramètres).
2. On fait du property binding quand la route cible est dynamique.

```
@Component({
  selector: 'app-root',
  template: `
    <a [routerLink]="['cv', cv.id]">route dynamique</a>
    <a routerLink="cv">route statique</a>
    <router-outlet></router-outlet>`,
  styleUrls: ['./app.component.css'],
  imports: [RouterOutlet, RouterLink],
  standalone: true,
})
export class AppComponent {
```

Créer des slices de routes

- Si vous utilisez les standalones components et afin **d'organisez vos routes par fonctionnalité (feature)**, vous pouvez créer des **slices de routes** et les **appeler**.

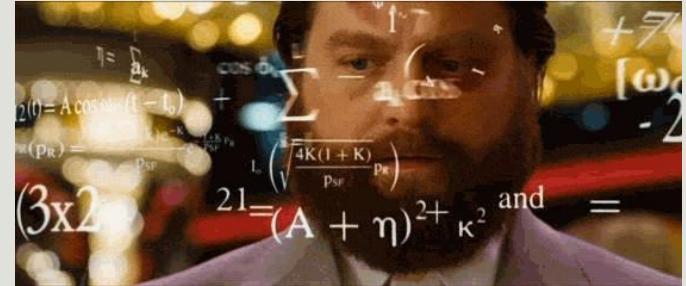
```
export const FEATURE_ROUTES = [  
  { path: 'featurehome', component: FeatureHomeComponent},  
  { path: 'otherfeaturehome', component: OtherFeatureHomeComponent },  
]
```

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

Exercice

slices de routes

- Faites ceci pour les routes de votre feature Cv

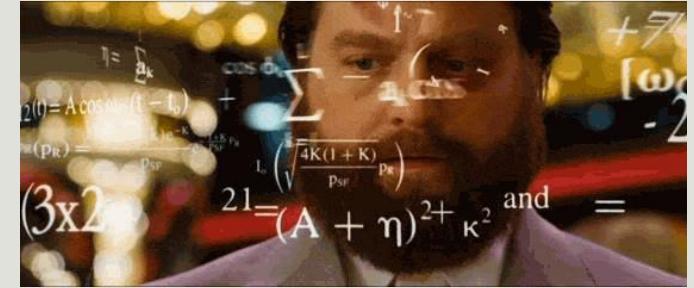


Déclencher une route routerLink

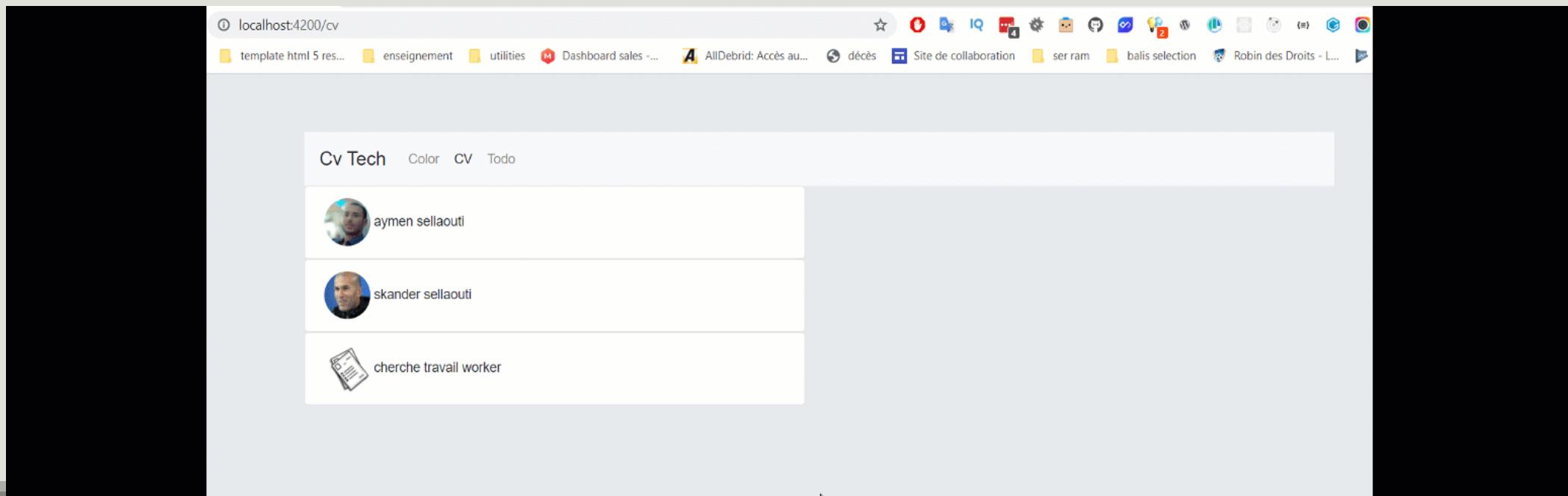
- Afin de **mettre en avant la route sélectionnée par l'utilisateur** dans votre menu, vous pouvez utiliser la directive **routerLinkActive**
- **routerLinkActive** prend en paramètre la **liste des classes CSS** que vous voulez **appliquez** à la **route sélectionnée ainsi qu'à tous ces ses ancêtres**.
- Par exemple si on a l'URI **/cv/liste** les classes de **routerLinkActive** seront ajoutées à cet URI ainsi qu'à l'URI **/cv** et **/**.
- Pour identifier uniquement l'uri cible, ajouter la directive **routerLinkActive** avec la valeur **{exact: true}**

```
<a class="nav-item nav-link"
  routerLinkActive="active text-primary"
  [routerLinkActiveOptions]="{exact: true}"
  [routerLink]=[']">Accueil</a>
```

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.



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: ...
```

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.

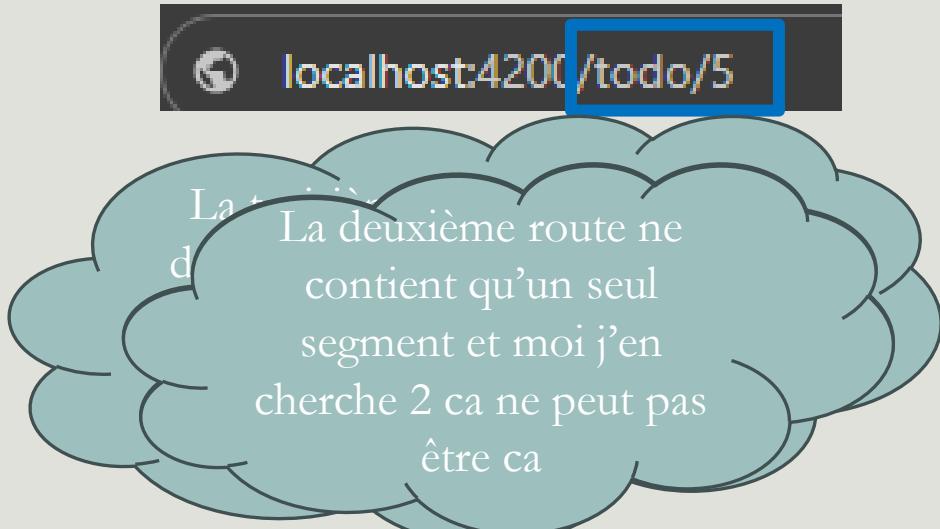
```
import { ActivatedRoute, Router } from "@angular/router";
router = inject(Router);
onLoadCv(id: number) {
  // méthode conseillée
  this.router.navigate(['cv', id]);
  // alternative
  this.router.navigate(['cv/${id}']);
}
```

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**.

```
const routes: Routes = [
  { path: '', component: FirstComponent },
  { path: 'todo/:id', component: TodoDetailsComponent },
  { path: 'color', component: ColorComponent },
];
```

Création d'un système de Routing



```
@NgModule({  
  imports: [  
    RouterModule.forRoot(routes),  
  ],  
  exports: [RouterModule],  
})  
export class AppRoutingModule {}
```

Je cherche une route avec deux segments

- 1- Le premier segment si c'est une constante il doit être todo sinon si c'est une variable peut contenir n'importe quoi
- 2- Le deuxième segment si c'est une constante il doit être 5, sinon si c'est une variable, il peut contenir n'importe quoi

```
const routes: Routes = [  
  { path: 'todo/5' , component: FirstComponent },  
  { path: 'color' , component: ColorComponent },  
  { path: 'todo/:id' , component: TodoDetailsComponent },  
];
```

Récupérer les paramètres d'une route

- Afin de récupérer les paramètres d'une root au niveau d'un composant on doit procéder comme suit :
 1. Importer **ActivatedRoute** qui nous permettra de récupérer les paramètres de la root de "**@angular/router**".
 2. Injecter **ActivatedRoute** au niveau du composant.
 3. Utilisez l'objet **snapshot**

```
import { ActivatedRoute } from "@angular/router";

export class DetailsCvComponent {
  acr = inject(ActivatedRoute);
  constructor() {
    this.acr.snapshot.params;
  }
}
```

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.

```
▼ 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.

▼ `snapshot.params:`
`id: "662"`

```
import { ActivatedRoute } from "@angular/router";

export class DetailsCvComponent {
  acr = inject(ActivatedRoute);
  constructor() {
    console.log({ 'snapshot.params': this.acr.snapshot.params });
  }
}
```

Passer le paramètre à travers le tableau de routerLink

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

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

Déclencher une route routerLink

Route relative Vs Route absolue

- Quand vous définissez la route vers laquelle vous voulez naviguer, si on **préfixe la valeur passé au routerLink** avec :
 - '/', la route sera **absolue**.
 - **Rien ou './'**, le Router **regardera dans les enfants du ActivatedRoute**.
 - '../', le Router regardera dans le niveau au-dessus dans l'arbre de routes.
- Quand on désire **passer un path relativement à la route dans laquelle on est**, il est possible de passer à la **méthode navigate**, en argument, **un objet dont la clé relativeTo** avec comme **valeur** est une **instance de ActivatedRoute**, informe sur le fait **qu'on veut une route relative à la route active**.

```
this.router.navigate(['edit'], {relativeTo: this.activatedRoute});
```

Les queryParameters

- Les **queryParameters** sont les paramètres envoyés 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')
```

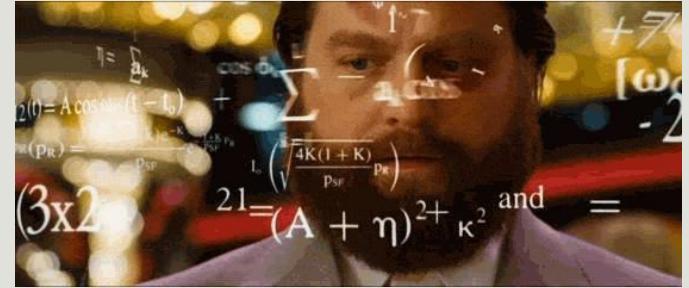
La route joker

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

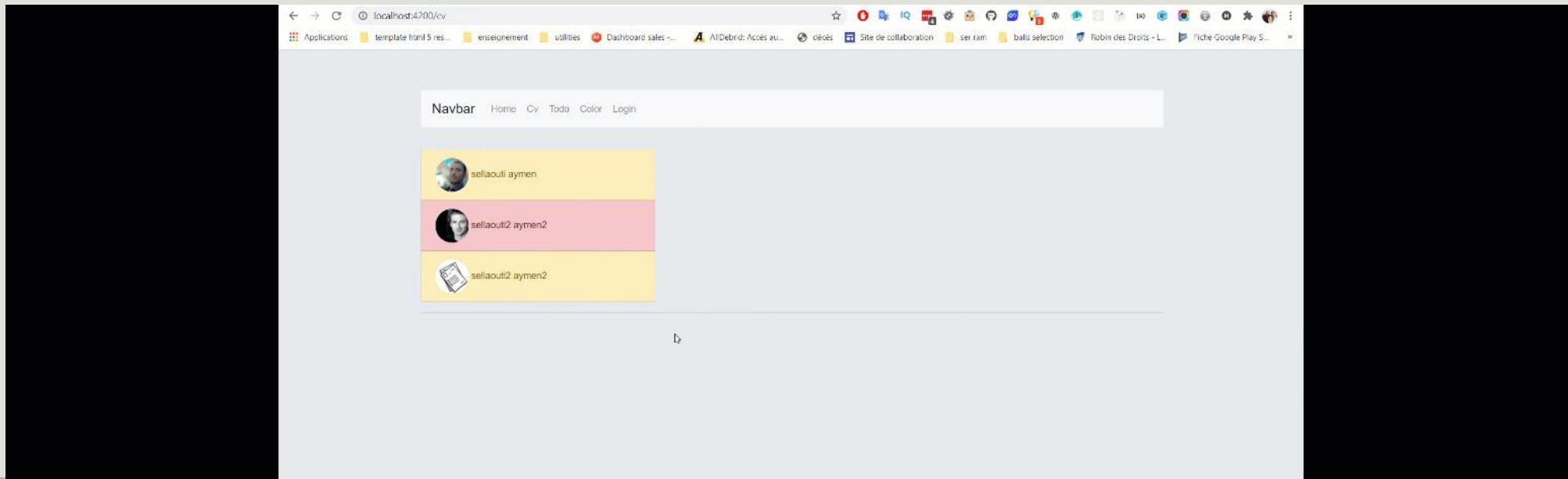
Exemple

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

Exercice



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



Route Fils

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

Route Fils

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

Route Fils

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

Route fils / définition dans un parent

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

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

Route Fils

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

Route Fils

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

Angular Form

AYMEN SELLAOUTI

Approche de gestion de FORM

1. Approche basée Template
2. Approche réactive

Objetctifs

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

Approche basée Template / Template Driven Approach

- 1 Importer le module **FormsModule** dans **app.module.ts** si vous travaillez avec un **application modulaire**, sinon **dans votre composant** si vous travaillez avec les **standalone component**
- 2 Angular détecte automatiquement un objet form à l'aide de la balise FORM. Cependant, il ne détecte aucun des éléments (inputs).
- 3 Spécifier à Angular quel sont les éléments (contrôles) à gérer.
 - Pour chaque élément ajouter la directive angular **ngModel**.
 - Identifier l'élément avec un nom permettant de le détecter et de l'identifier dans le composant.
- 4 Associer l'objet représentant le formulaire à une variable et la passer à votre fonction en utilisant le référencement interne # et la directive **ngForm**

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

Approche basée Template / Template Driven Approach

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

Template

```
export class  
TmeplateDrivenComponent {  
  onSubmit(formulaire:  
NgForm) {  
  
    console.log(formulaire);  
  }  
}
```

Component.ts

Approche basée Template Validation

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

- required
- email

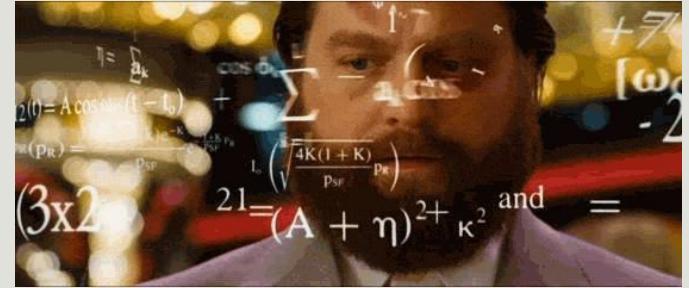
La propriété valid de ngForm permet de vérifier si le formulaire est valid ou non en se basant sur les validateurs qu'ils contient.

Approche basée Template NgForm

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

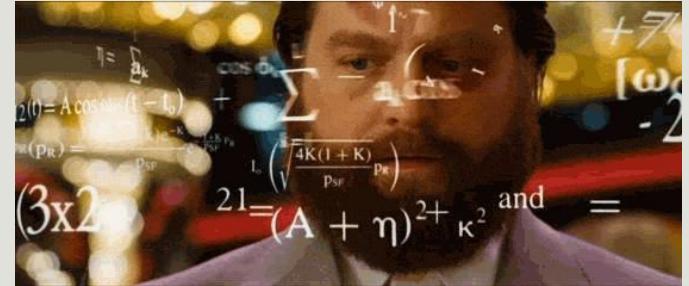
- **dirty** : informe sur le fait que l'une des propriétés du formulaire a été modifié ou non
- **Valid / invalid** : informe si le formulaire est valide ou non
- **Untouched / touched** : informe si le formulaire est touché ou non
- **pristine** : le formulaire n'a pas été touché, c'est l'opposé du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

Exercice



- Créer un formulaire d'authentification contenant les champs suivants :
 - Email
 - Password
 - Envoyer
- Si un champ est invalide alors il devra avoir une bordure rouge.
- Les deux champs sont obligatoires
- Le password doit avoir au moins 4 caractères.
- Un champ vide et non encore modifié ne peut avoir de bordure rouge que s'il a été touché.
- Le bouton « envoyer » ne doit être cliquable que si le formulaire est valide.
- Utiliser le binding sur la propriété *disabled*.

Exercice



localhost:4200/login

Applications enseignement utilities Dashboard sales ... AllDebrid: Accès au... décès Site de collaboration ser ram balls selection Robin des Droits - L... Fiche Google Play S...

Navbar Home Cv Todo Color Login

Email :

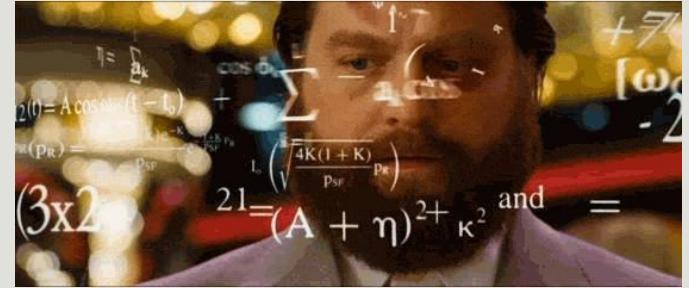
password :

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.

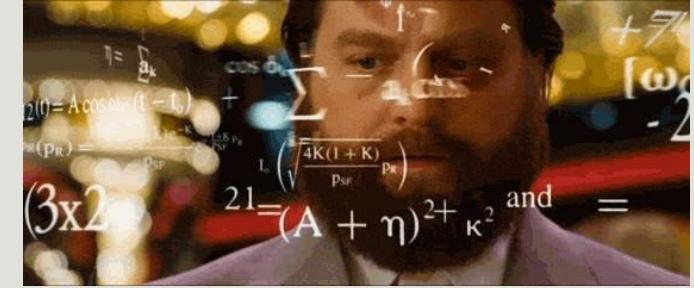
Approche basée Template

Associer des valeurs par défaut aux champs

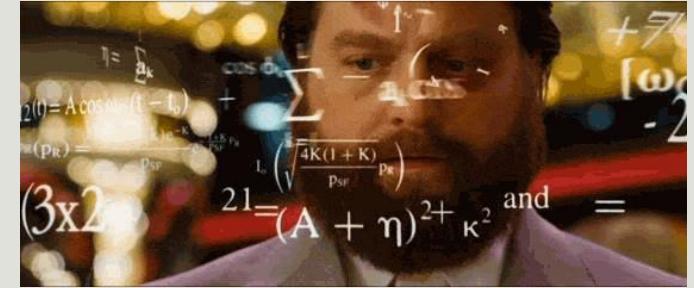
- Pour associer des valeurs par défaut aux champs d'un formulaire associé à Angular il faut le faire à partir du composant.
- Afin de gérer les valeur du formulaire à partir du composant il faut du binding.
- Au lieu d'avoir juste la primitive ngModel associée au contrôle d'un élément on ajoute le **property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

Exercice

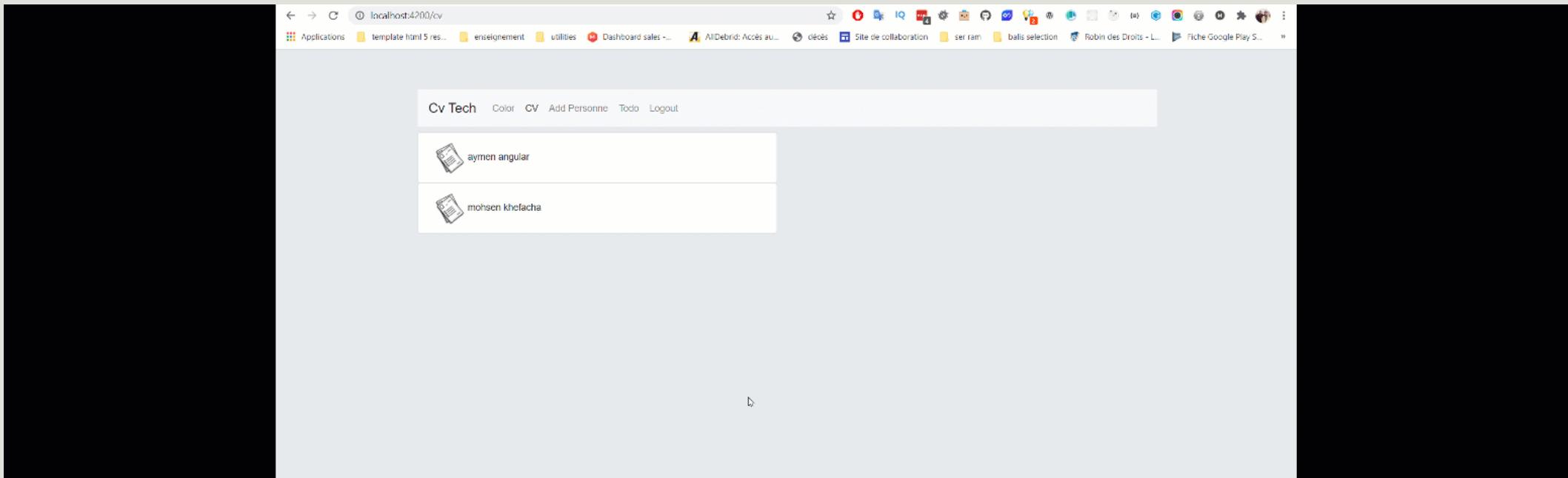
- Ajouter la valeur par défaut « myUserName » au champ username.



Exercice



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



Angular HTTP et Déploiement

AYMEN SELLAOUTI

Objectifs

1. Comprendre le design pattern 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 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)

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

Promesse

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve(3);
  }, 5000);
});
promise.then(function (x) {
  console.log('resolved with value :', x);
});
```

Qu'est ce que la programmation réactive

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

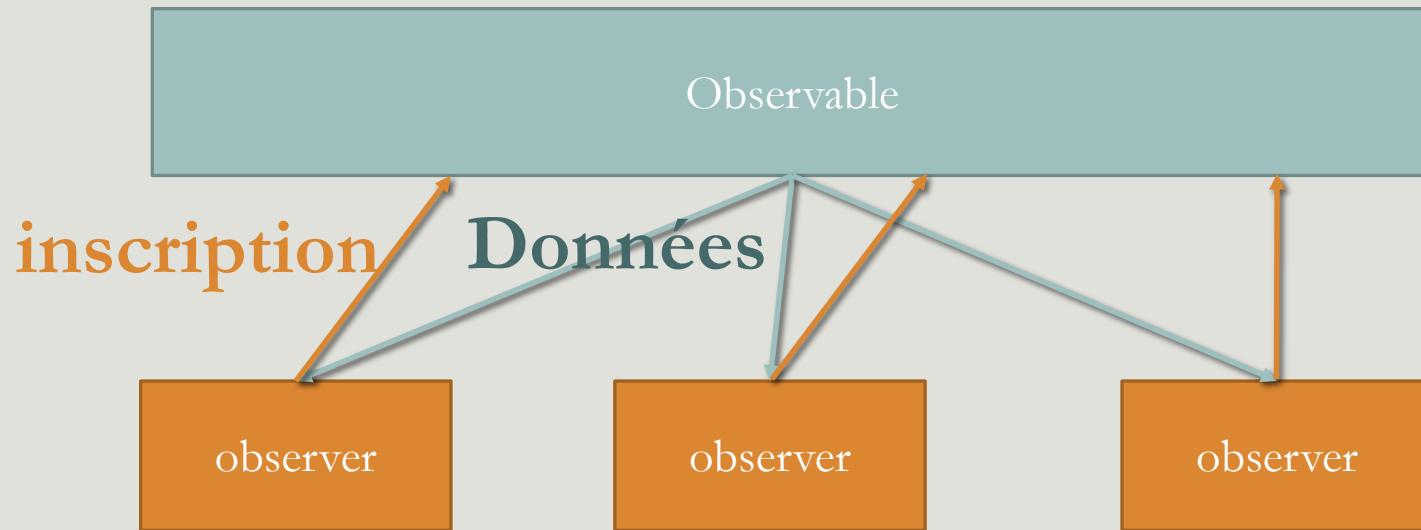
Programmation reactive =

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

Le pattern « Observer »

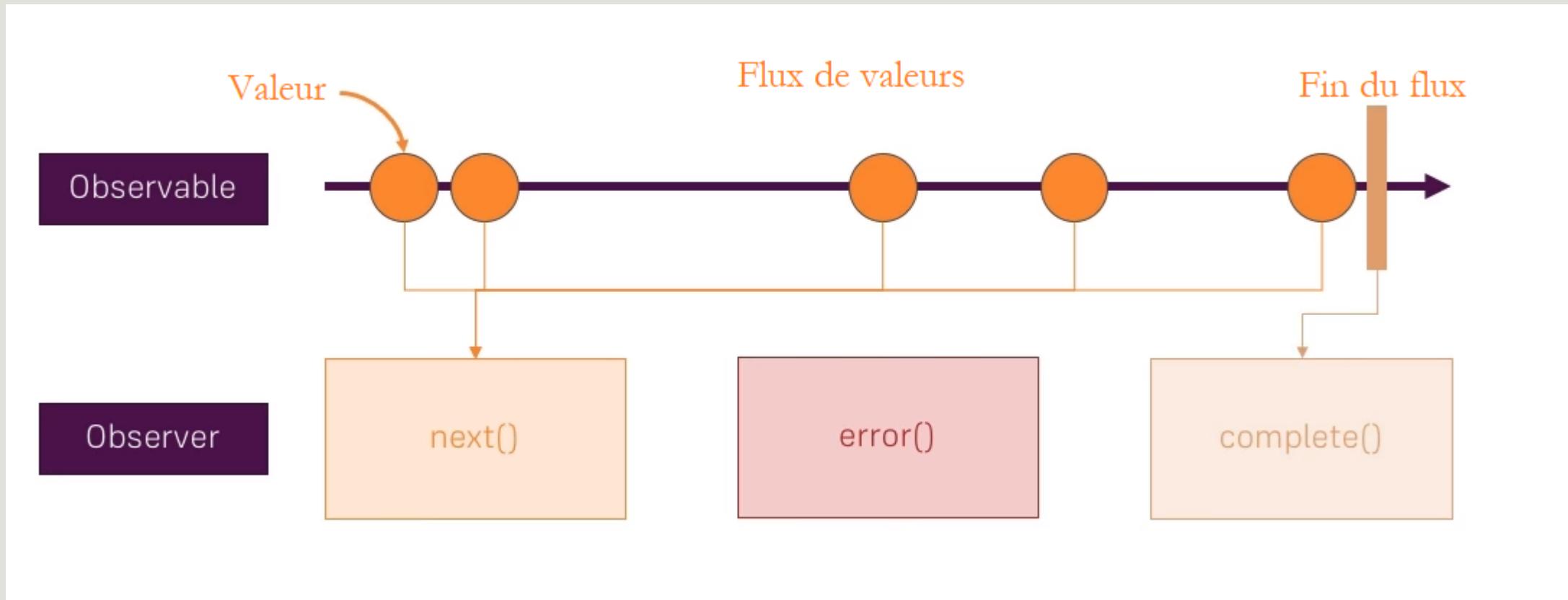
- Le patron de conception **Observateur** permet à un objet de garder la trace d'autres objets, intéressés par l'état de ce dernier.
- Il définit une relation entre objets de type un-à-plusieurs.
- Lorsque l'état de cet objet **change**, il **notifie** ces **observateurs**.

Observables, Observers et subscriptions



traitement

Fonctionnement



Promesse Vs Observable

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

S'inscrire à un observable

- Afin de **montrer votre intérêt à un flux Observable**, vous pouvez utiliser la méthode **subscribe** qu'il vous offre.
- En vous inscrivant à cet observable, vous pouvez **passez à cette méthode** un **objet qui a 3 fonctions** :
 - **next**, qui sera **appelé à chaque fois qu'une nouvelle valeur arrive dans le flux**. Elle prend en **paramètre cette valeur** et vous permet d'implémenter ce que vous voulez faire avec.
 - **Error, déclenché une fois en cas d'erreur** qui vous permet d'implémenter le comportement en réaction à cette erreur
 - **Complete, déclenché dès la fin du flux** qui vous permet d'implémenter le comportement en réaction à la fin du flux.

S'inscrire à un observable

```
myObservable$.subscribe({
    next: (data) => {
        //TODO: implementez le comportement quand vous recevez les données
    },
    error: (error) => {
        //TODO: implementez le comportement quand vous avez une erreur
    },
    complete: () => {
        //TODO: implementez le comportement à la fin du flux
    },
});
```

Exemple d'un Observable

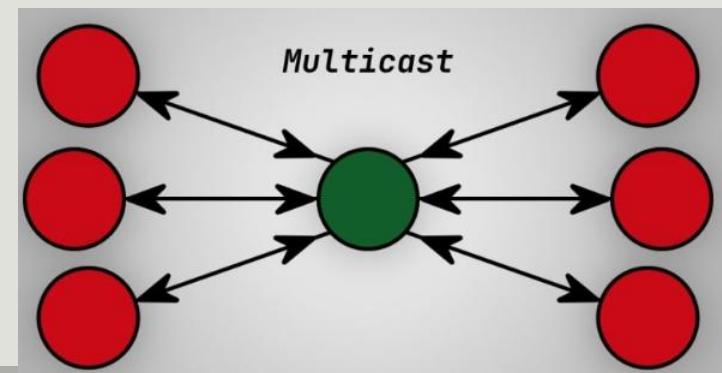
Création et inscription

```
export class TestObservableComponent {  
  myObservable$: Observable<number>;  
  constructor() {  
    this.myObservable$ = new Observable((observer) => {  
      let i = 5;  
      const intervalIndex = setInterval(() => {  
        if (!i) {  
          observer.complete();  
          clearInterval(intervalIndex);  
        }  
        observer.next(i--);  
      }, 1000);  
    });  
    this.myObservable$.subscribe({  
      next: (val) => {  
        console.log(val);  
      },  
    });  
  }  
}
```



Hot Vs Cold Observable

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



asyncPipe

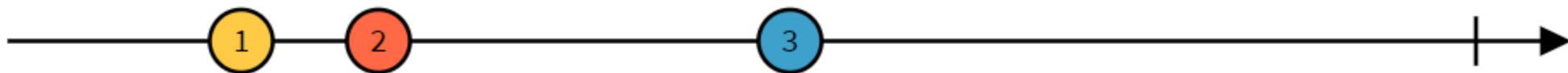
- asyncPipe est un pipe qui permet d'afficher directement un observable.
- `{ { valeurSourceAsynchrone | async } }`
- L'asyncPipe s'inscrit automatiquement à l'observable et affiche le dernier résultat envoyé.
- Quand le composant est détruit l'asyncPipe se désinscrit automatiquement de l'observable.

Les opérateurs de l'observable

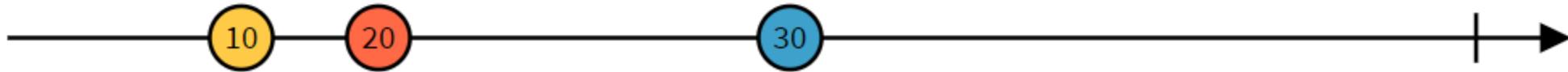
- Les opérateurs sont des fonctions. Il y a deux types d'opérateurs :
- **Un opérateur pipeable** est une fonction qui **prend un observable** comme entrée et **renvoie un autre observable**. C'est une opération pure : **le précédent Observable reste inchangé**.
 - Syntaxe : `monObservable.pipe(opertaeur1(),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

Quelques opérateurs utiles de l'Observable

map



`map(x => 10 * x)`



Quelques opérateurs utiles de l'Observable

filter



```
filter(x => x > 10)
```



Quelques opérateurs utiles de l'Observable

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

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

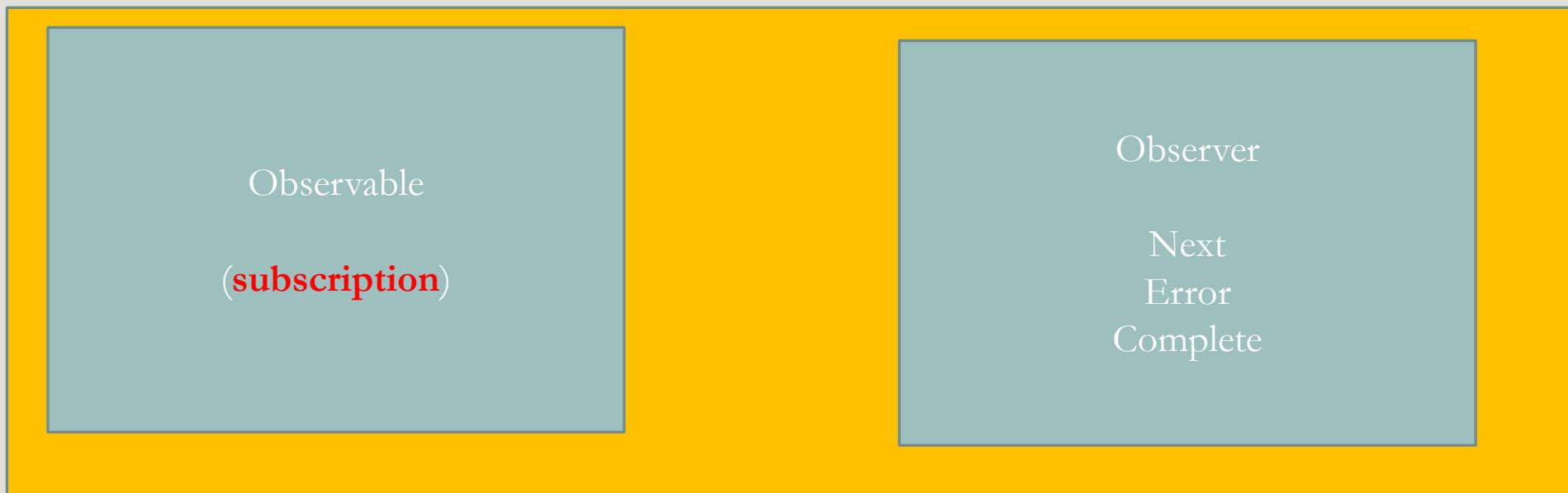
<http://rxmarbles.com/>

Les subjects

- Afin de créer des **flux chauds**, RxJs nous fournit une structure de données appelées **Subject**.
- Pensez au Subject comme à **un flux que vous créez au fur et à mesure** que vous recevez les données.
- C'est une **création en temps réel**, vous **n'avez aucune information au préalable** sur le contenu du flux.
- C'est comme la **transmission d'un match à la télé en direct**. Votre chaîne ne sait pas quelles images elle va recevoir, elle ne les a pas en mémoire sous forme d'une vidéo, à chaque fois qu'elle reçoit une image de la source, elle vous la transmet.

Les subjects

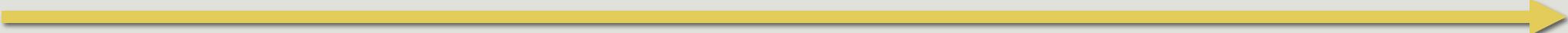
- Un **subject** est un type particulier d'observable. En effet Un **subject** est en même temps un **observable** et un **observer**, il possède donc les méthodes next, error et complete.
- Pour **broadcaster** une nouvelle valeur, il suffit d'appeler la méthode **next**, et elle sera diffusé aux Observateurs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



Les subjects

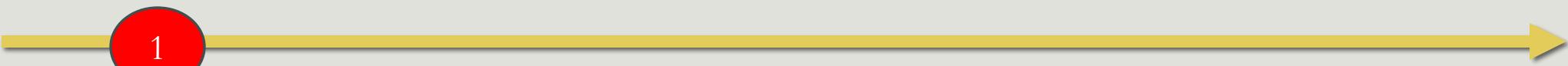
- Afin de créer un flux chaud commencer par **instancier un Subject**. Ceci vous permet d'avoir un flux vide auquel on peut s'inscrire.

```
nbUser$ = new Subject<Number>();
```

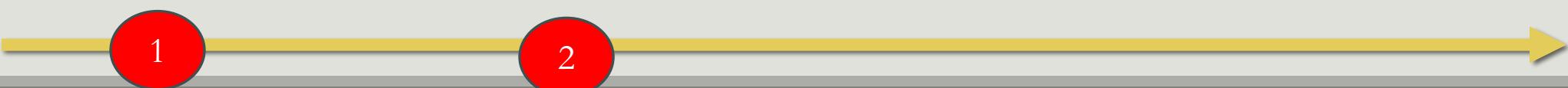


- Pour ajouter une valeur dans le flux, il suffit d'appeler la méthode next.

```
this.nbUser$.next(1);
```



```
this.nbUser$.next(2);
```

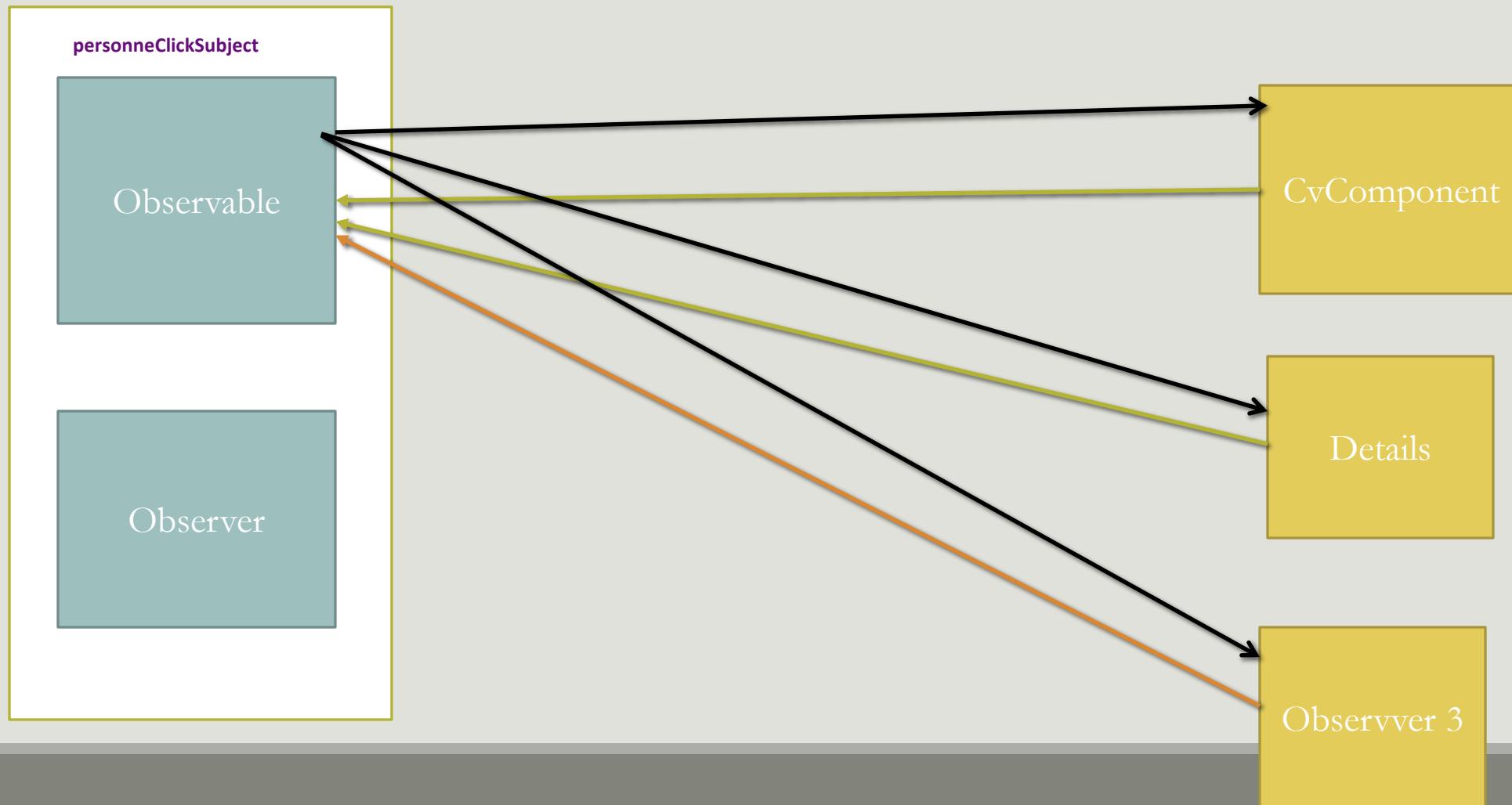


Les subjects

- Si vous êtes intéressé par ce flux, c'est un flux comme un autre, il vous suffit donc de vous inscrire.

```
nbUser$.subscribe({  
    next:(nb) => {},  
    error:(e) => {},  
    complete:() => {},  
})
```

Les subjects



Installation de HTTP

- A partir de la version **18, le HttpClientModule est devenu deprecated.**
- Afin d'utiliser le service HttpClient, vous devez le configurer en utilisant la méthode **provideHttpClient**, que la plupart des applications incluent dans la clé providers de votre app.config.ts.

```
export const appConfig: ApplicationConfig = {  
  providers: [  
    provideHttpClient(),  
  ],  
};
```

Installation de HTTP

- Une fois configuré, vous pouvez injecter votre service là où vous en avez besoin.

```
http = inject(HttpClient);
```

Interagir avec une API Get Request

- Afin d'exécuter une requête **get** le module http nous offre une méthode **get**.
- Cette méthode retourne un **Observale**.
- Inscrivez vous à cet observable et définissez le comportement en cas de succès, erreur ou complétion

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

Interagir avec une API POST Request

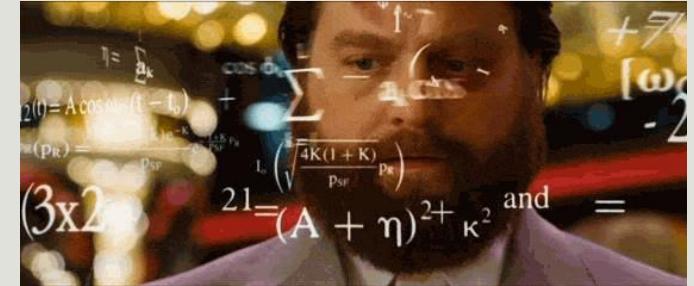
- Afin d'exécuter une requête POST le module http nous offre une méthode post.
- Cette méthode retourne un Observale.
- Diffère de la méthode get avec un attribut supplémentaire : body

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

Documentation

<https://angular.io/guide/http>

Exercice



- Accéder au site <https://jsonplaceholder.typicode.com/>
- Utiliser l'API des posts pour afficher la liste des posts. En attendant le chargement des données afficher un message « loading... ».
- Ajouter un input. A chaque fois que vous écrivez un élément dans cet input il sera ajouté dans la liste.

Les headers

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

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

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

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

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

Les paramètres

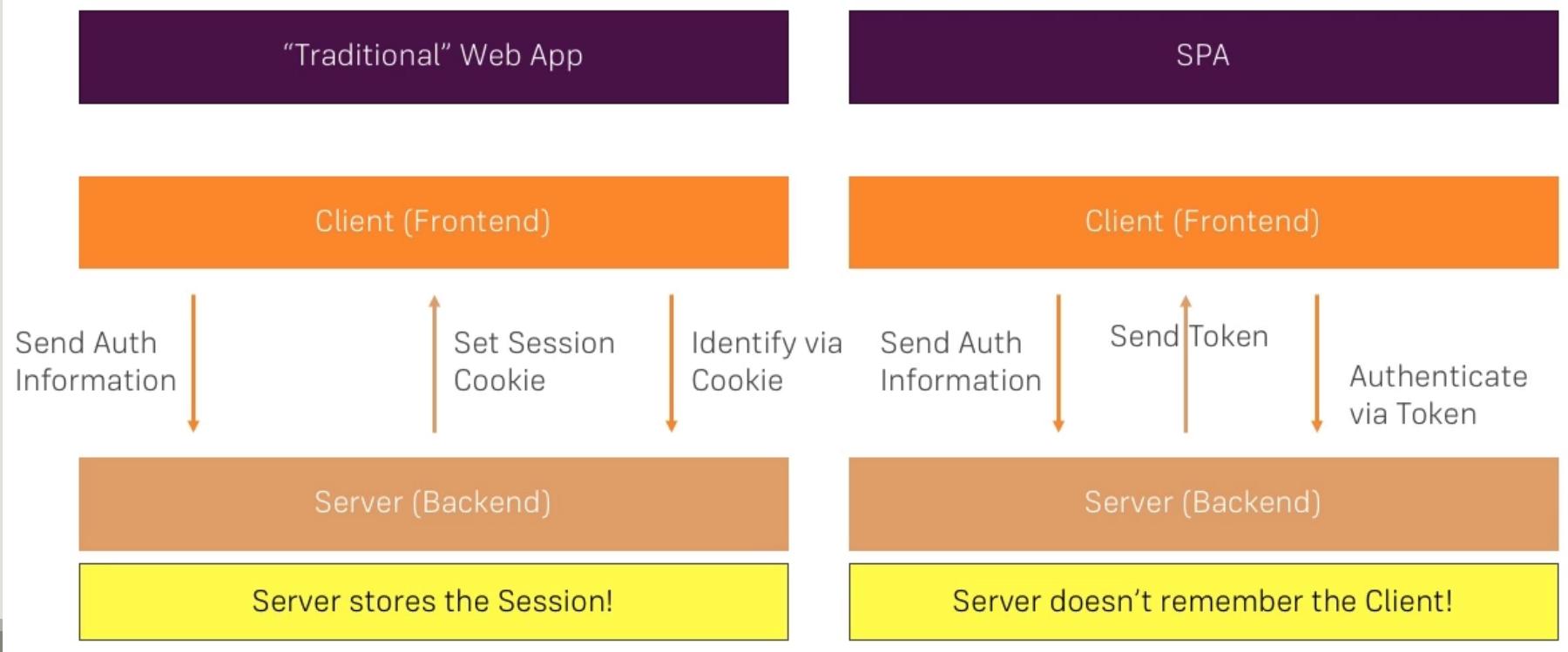
- Afin d'ajouter des paramètres à vos requêtes, le HttpClient vous offre la classe HttpParams.
- Cette classe est une classe immutable (read Only).
- Elle propose une panoplie de méthode helpers permettant de la manipuler.
- `set(clé,valeur)` permet d'ajouter des headers. Elle écrase les anciennes valeurs.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

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

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

Authentification

How does Authentication work?



Ajouter le token dans la requête

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

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

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

Ajouter le token dans la requête

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

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

Sécuriser vos routes

- Dans vos applications, certaines routes ne doivent être accessibles que si vous êtes authentifié. Ce cas d'utilisation se répète souvent et c'est un sous cas de la sécurisation de vos routes.
- Angular a pris en considération ce cas en fournissant un mécanisme via l'utilisation des **Guard**.

Guard

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

Guard / canActivate

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

Guard / canActivate

1

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

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor() {}

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

Guard

2

- A partir d'Angular 14 et la possibilité d'utiliser la fonction inject dans tous les contextes d'injection, Angular préconise les fonctionnels Guards.
- Commencez par créer votre Guard et ajoutez-y votre logique.
- Si vous retournez true ou une promise de true ou un Observable de true la route sera activée.
- Si c'est false la navigation vers la route sera annulée

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

Guard / canActivate

2

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

App.module.ts

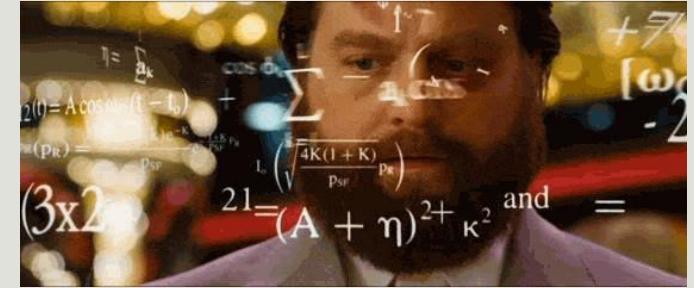
Guard / canActivate

3

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

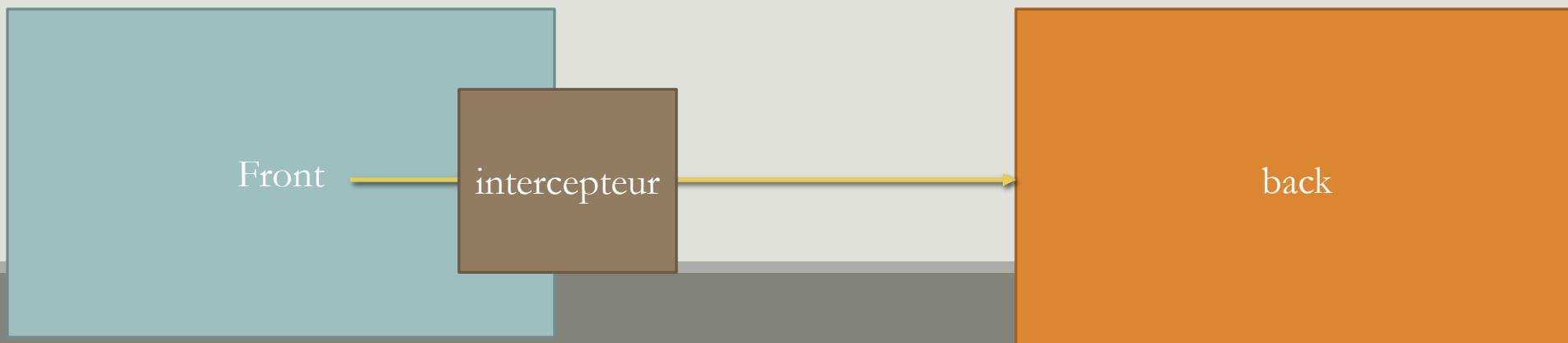
Exercice

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

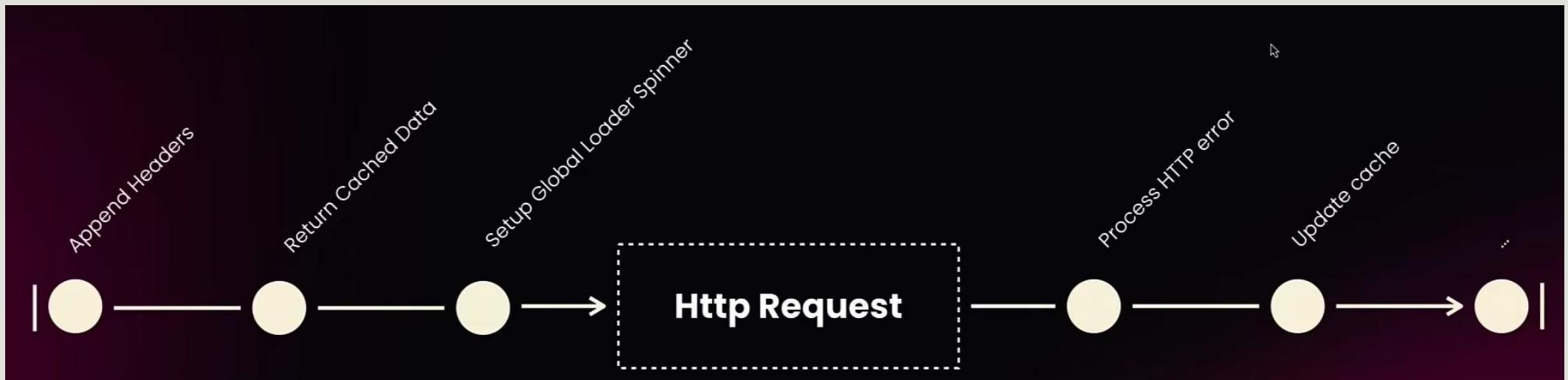


Les intercepteurs

- 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 soit une fonction (angular 17) ou une classe qui **implémente l'interface HttpInterceptor.**
- En implémentant cette interface, chaque intercepteur va devoir **implémenter** la méthode **intercept**.



Les intercepteurs



Les intercepteurs

- Dans les anciennes versions d'angular on devait passer par une class qui implémente l'interface HttpInterceptor.

```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  intercept( request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    return next.handle(request);
  }
}
```

Les intercepteurs

- Dans les nouvelles versions d'angular nous passons par une **fonction**.
- Dans cette fonction vous récupérer comme premier paramètre l'objet **request** qui représente la **requête HTTP**.
- En deuxième paramètre, vous avez la fonction next qui vous permet de remettre la requête de continuer son parcours.

```
import { HttpInterceptorFn } from "@angular/common/http";
export const newInterceptor: HttpInterceptorFn = (req, next) => {
  return next(req);
};
```

Les intercepteurs : changer la requête

- Par défaut la requête est immutable, on ne peut pas la changer.
- Solution : la cloner via sa méthode clone, changer les headers du clone et le renvoyer.

```
const cloneReq = request.clone({
  setHeaders: {
    'Authorization': token
  }
});
// Chainer la nouvelle requete avec next.handle
return next.handle(newReq);
```

Les intercepteurs

Appliquer l'intercepteur

- Afin d'appliquer l'intercepteur, vous devez le fournir au niveau du provider.
- Si vous utilisez un **intercepteur fonctionnel**, vous utilisez la fonction **withInterceptors** qui est une des options de **provideHttpClient**.
- Elle prend en paramètre un tableau d'intercepteur

```
bootstrapApplication(AppComponent, {providers: [
  provideHttpClient(
    withInterceptors([firstInterceptor, secondInterceptor]),
  )
]});
```

Declarative Mindset

A declarative mindset is not easy to develop

A declarative mindset makes it easier to develop

Les opérateurs de l'observable

- Les opérateurs sont des fonctions. Il y a deux types d'opérateurs :
- **Les opérateurs de création**, elles permettent de créer un Observable. Par exemple : `of(1, 2, 3)` crée un observable qui va émettre 1, 2, et 3, l'un après l'autre.
- **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

- Les opérateurs de création sont utilisés pour créer de nouveaux observables.
- Ils sont divisés en **opérateurs de création** et en **opérateurs de création de jointure**.
- La principale différence entre eux réside dans le fait que les opérateurs de création de **jointure créent des observables à partir d'autres observables**, alors que les **opérateurs de création** créent des observables **à partir d'objets qui diffèrent des observables**.

Les opérateurs de création from

- **from** est utilisé pour convertir des types d'objets JavaScript comme un **tableau**, une **promesse** ou un **objet itérable** en une séquence **observable** de valeurs.
- L'opérateur émet également une **chaîne** sous forme de **séquence de caractères**.

```
from([1, 2, 3]).subscribe({  
    next: (data) => console.log('[from]', data),  
    complete: () => console.log('[from] complete'),  
});
```

[from] 1
[from] 2
[from] 3
[from] complete

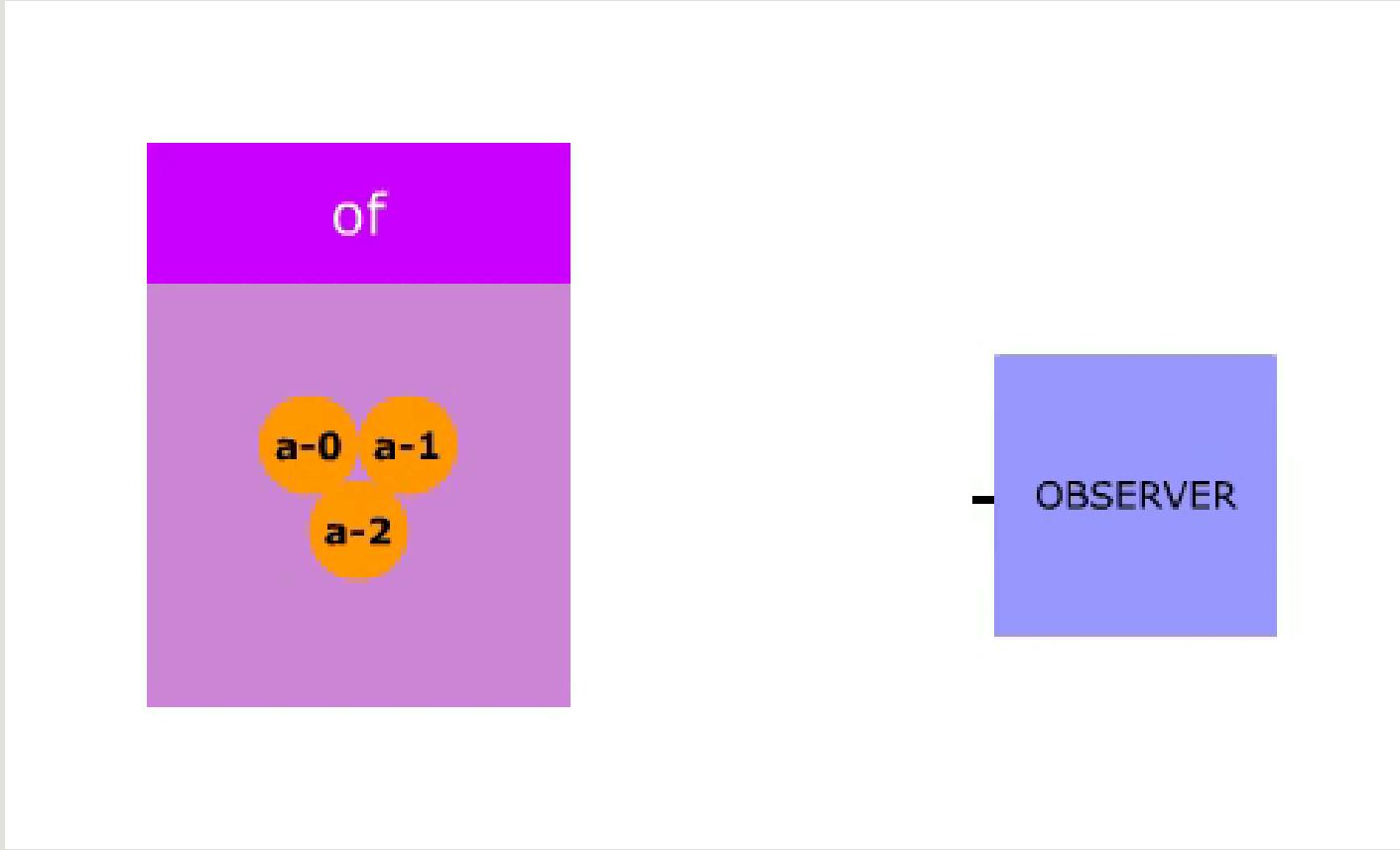
Les opérateurs de création of

- l'opérateur **of** est utilisé afin de **convertir un argument en un observable**
- il ne fait **aucun aplatissement ou conversion** et **émet** chaque argument sous le **même type qu'il reçoit**
- **of** est couramment utilisé lorsque vous avez simplement **besoin de renvoyer une valeur là où un observable est attendu** ou de **démarrer une chaîne observable**.

```
of([1, 2, 3], new Date(), {  
    name: 'sellaouti',  
    firstname: 'aymen',  
}).subscribe({  
    next: (data) => console.log('[of]', data),  
})
```

```
[of] ▶ (3) [1, 2, 3]  
[of] Tue Jan 03 2023 13:46:43 GMT+0100 (West Africa Standard Time)  
[of] ▶ {name: 'sellaouti', firstname: 'aymen'}  
[of] complete
```

Les opérateurs de création of



Les opérateurs de création timer

- L'opérateur **timer** est un opérateur de création utilisé pour créer un observable qui commence à émettre les valeurs après un délai d'attente, et la valeur continuera d'augmenter après chaque appel.
- Il prend en **entrée** en **premier paramètre quand déclencher** l'évent.
- En **second paramètre** en cas de volonté de répétition chaque **combien de millisecondes reprendre l'émission**.

```
timer(1000).subscribe((val) => console.log('[timer] : ' + val));
```

```
[timer] : 0
```

```
timer(1000, 1000).subscribe((val) => console.log('[timer] : ' + val));
```

```
[timer] : 0  
[timer] : 1  
[timer] : 2  
[timer] : 3  
[timer] : 4
```

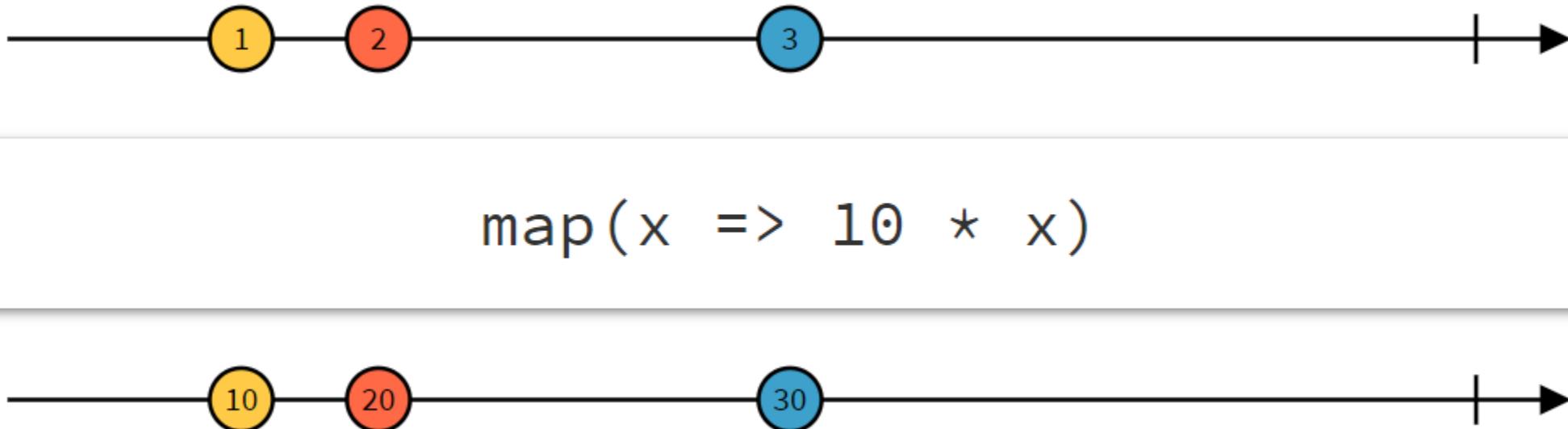
Les opérateurs de création fromEvent

- L'opérateur **fromEvent** est utilisé pour émettre un observable en se basant sur un événement.
- Il prend en paramètre l'élément cible puis l'événement à écouter.

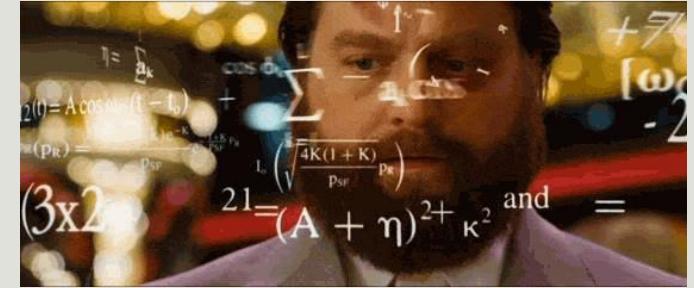
```
export class FromEventComponent implements AfterViewInit {  
  @ViewChild('btn') button!: ElementRef;  
  onClickButton$: Observable<Event>;  
  ngAfterViewInit() {  
    this.onClickButton$ = fromEvent(this.button.nativeElement, 'click');  
  }  
}
```

Quelques opérateurs utiles de l'Observable opérateur pipable

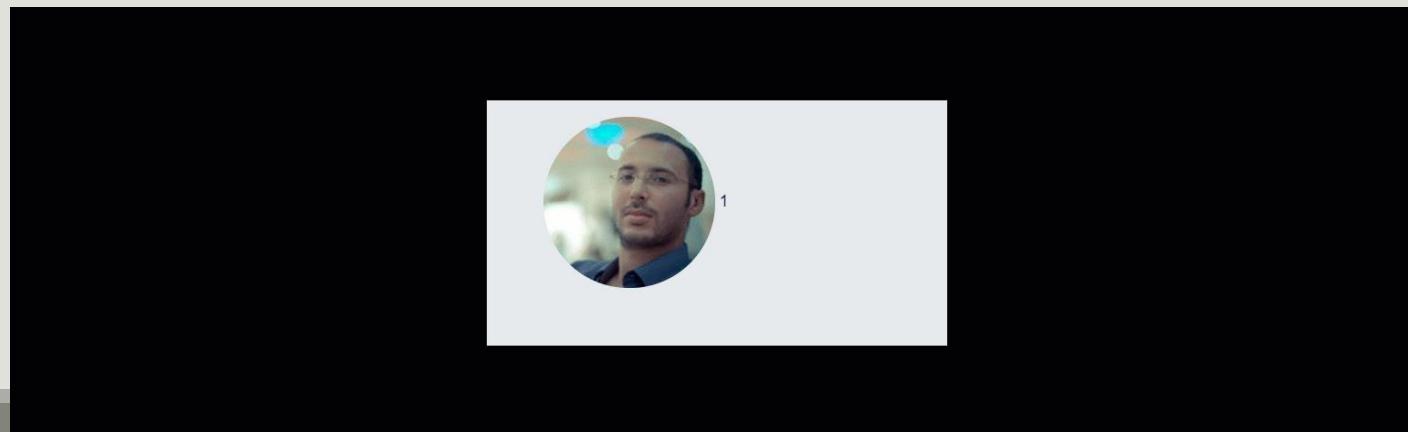
map



Exercice



- Ecrire un composant qui affiche une suite d'images non stop.
- Utiliser un observable comme source d'images.
- Vous devez uniquement utiliser des opérateurs pour faire le travail.
- La taille de l'image, la liste des images et le temps entre chaque image doit être paramétrable.



Quelques opérateurs utiles de l'Observable opérateur pipable

filter



```
filter(x => x > 10)
```



Quelques opérateurs utiles de l'Observable

opérateur pipable

take

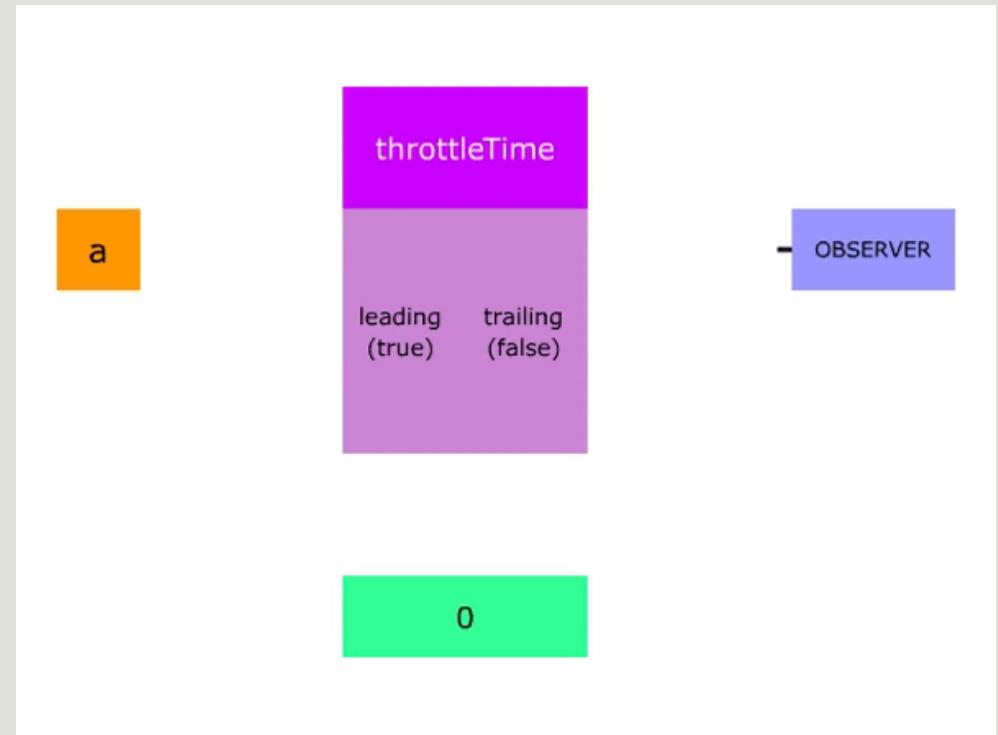
- L'opérateur **take** prend des valeurs de la source observable à l'observateur (mise en miroir) **jusqu'à ce qu'il atteigne le seuil de valeurs défini** pour l'opérateur.
- Sur chaque valeur, **take** compare le nombre de valeurs qu'il a mises en miroir avec le seuil défini. Si le **seuil est atteint**, il termine le flux en se **désabonnant** de la source et en transmettant la notification **complète** à l'observateur.



Quelques opérateurs utiles de l'Observable

opérateur pipable throttleTime

- **throttleTime** prend en paramètres un nombre x représentant le nombre de millisecondes.
- Au **départ** le **timer est désactivé**.
- Dès que la **première valeur est émise**, elle la laisse passer et lance un timer pendant x ms.
- Pendant la durée du timer **rien ne passe et même si une nouvelle valeur arrive le timer reste inchangé**.
- Une fois le timer fini, **throttleTime refait la même chose** et attend la première émission pour reprendre le même processus.
- **Cas d'utilisation :** Permettre à l'utilisateur de **déclencher une fois un évènement** dans un **intervalle** de temps donné.
- **Appeler** une **fonction une seule fois** dans un **intervalle** de temps particulier au **survol de la souris**.



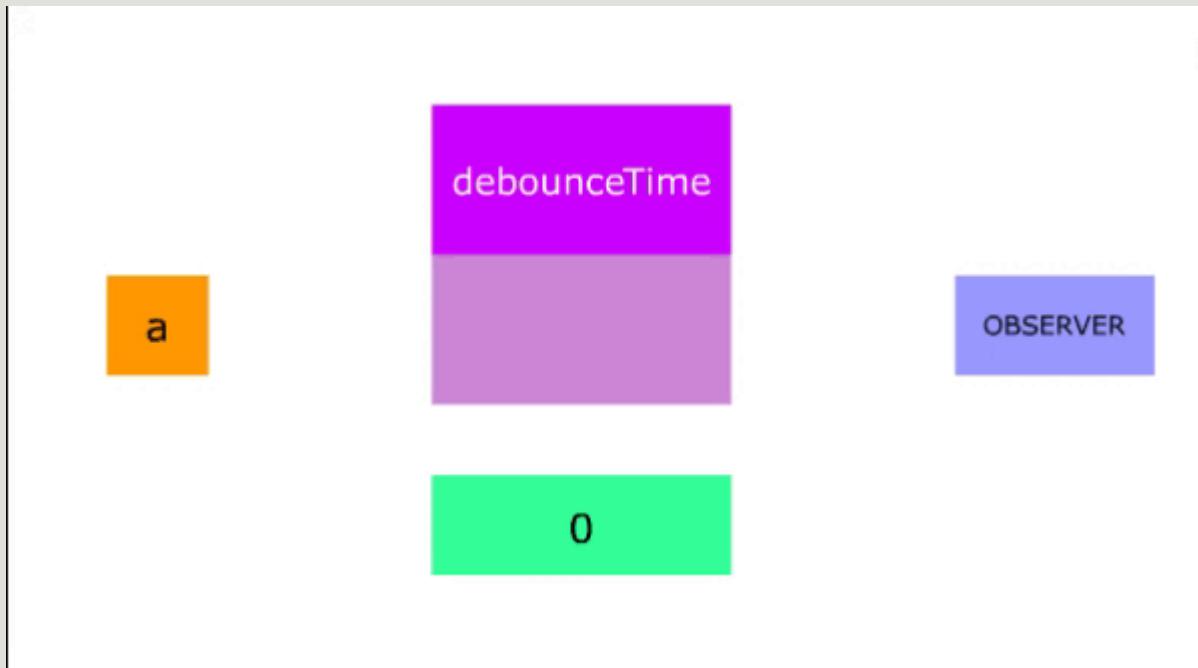
Quelques opérateurs utiles de l'Observable

opérateur pipable debounceTime

➤ **debounceTime** retarde les valeurs émises par une source pour le **temps d'échéance donné**. Si dans ce délai une **nouvelle valeur arrive**, la **valeur en attente précédente est supprimée** et **l'intervalle est réinitialisé**.

➤ De cette façon, **debounceTime garde** une trace de la **valeur la plus récente** et émet cette valeur la plus récente lorsque l'heure d'échéance donnée est dépassée.

➤ **L'autocomplete** est le **cas classique** du **debounceTime**



Les opérateurs d'aplatissement/ flattening operators

- Une erreur courante commise dans les applications Angular consiste à **imbriquer les abonnements observables**.
- Cette syntaxe **n'est pas recommandée** car elle est **difficile à lire et peut entraîner des bogues des effets secondaires inattendus**.
- Par exemple, cette syntaxe rend **difficile la désinscription** correcte de tous ces observables.
- De plus, si observable1 émet plus d'une fois dans un court laps de temps, **nous pourrions vouloir annuler l'abonnement précédent à observable2** et en démarrer un nouveau basé sur les nouvelles données reçues d'observable1.

```
this.activatedRoute.params.subscribe(  
  (params) => {  
    this.cvService.findPersonneById(params.id)  
      .subscribe(  
        (personne) => {  
          this.personne = personne;  
        },  
        (erreur) => {  
          if (!this.personne) {  
            this.router.navigate(['cv']);  
          }  
        }  
      );  
  }  
);
```

Les opérateurs d'aplatissement/ flattening operators

- L'opérateur **d'aplatissement** sont des opérateur qui permettent d'émettre un flux à partir d'un autre.
- Ils permettent **d'éviter les inscriptions imbriquées avec subscribe**.
- Il en existe plusieurs variantes et qui permettent de spécifier comment gérer les flux récupérés :
 - Est-ce qu'on est encore intéressé par l'inscription précédente ?
 - Est-ce que l'ordre des inscriptions est important ?

Les opérateurs d'aplatissement/ flattening operators

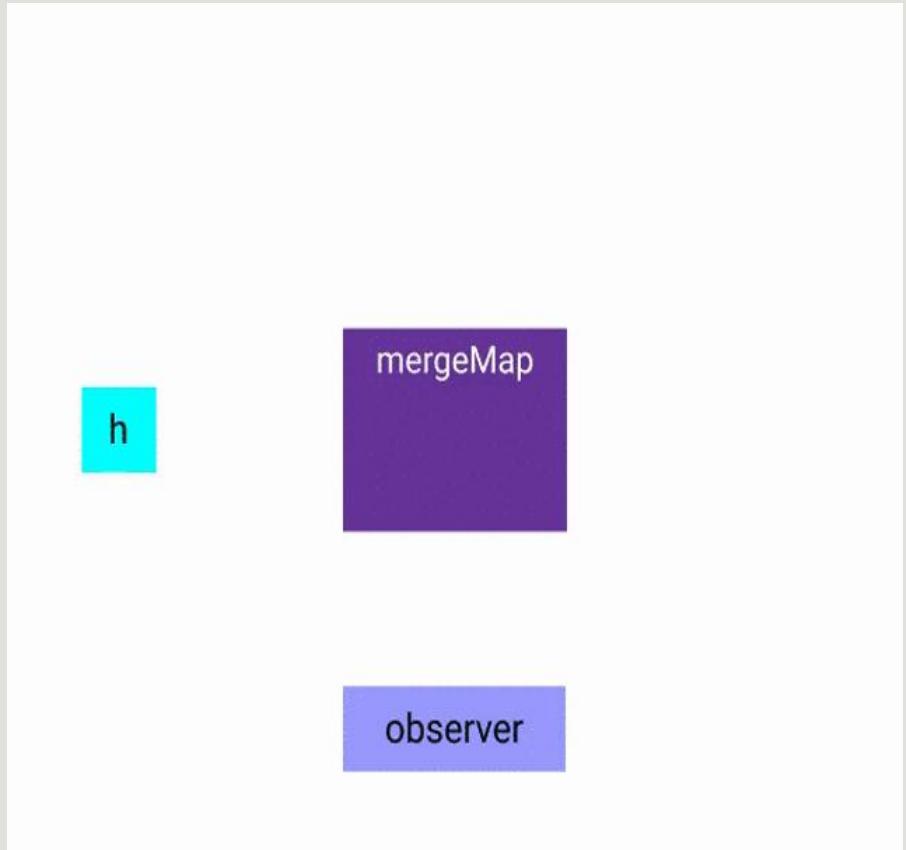
MergeMap

- L'opérateur **mergeMap** est essentiellement une **combinaison** de deux opérateurs - **merge et map**.
- La partie **map** vous permet de **mapper** une valeur d'une source observable à un **flux observable**. Ces flux sont souvent appelés flux internes.
- La partie **merge combine** tous les flux internes observables renvoyés par la map et **émet simultanément toutes les valeurs de chaque flux d'entrée**.

Les opérateurs d'applatissement/ flattening operators

MergeMap

- Au fur et à mesure que les valeurs de toute séquence combinée sont produites, ces valeurs sont émises dans le cadre de la séquence résultante.
- Utilisez cet opérateur **si vous n'êtes pas concerné par l'ordre des émissions** et que vous êtes simplement **intéressé par toutes les valeurs provenant de plusieurs flux combinés** comme si elles étaient produites par un seul flux.



Les opérateurs d'applatissement/ flattening operators MergeMap

```
timerObs(timer: number, name: string, iteration = 4) {
  return new Observable((observer: Observer<string>) => {
    let i = 0;
    const x = setInterval(() => {
      if (i >= iteration) {
        observer.complete();
        clearInterval(x);
      }
      observer.next(`observable ${name} ${++i}`);
    }, timer);
  });
}
```

```
params = [
  {name: 'obs1', timer: 1000, iteration: 4 },
  {name: 'obs2', timer: 1500, iteration: 4 },
];
constructor(private rxjsService: RxjsService) {
  from(this.params).pipe(
    mergeMap((param) => thisrxjsService.timerObs(param.timer,
param.name))
  ).subscribe(
    (data) => console.log(data)
  )
}
```

observable obs1 1
observable obs2 1
observable obs1 2
observable obs1 3
observable obs2 2
observable obs1 4
observable obs2 3
observable obs2 4

Les opérateurs d'applatissement/ flattening operators

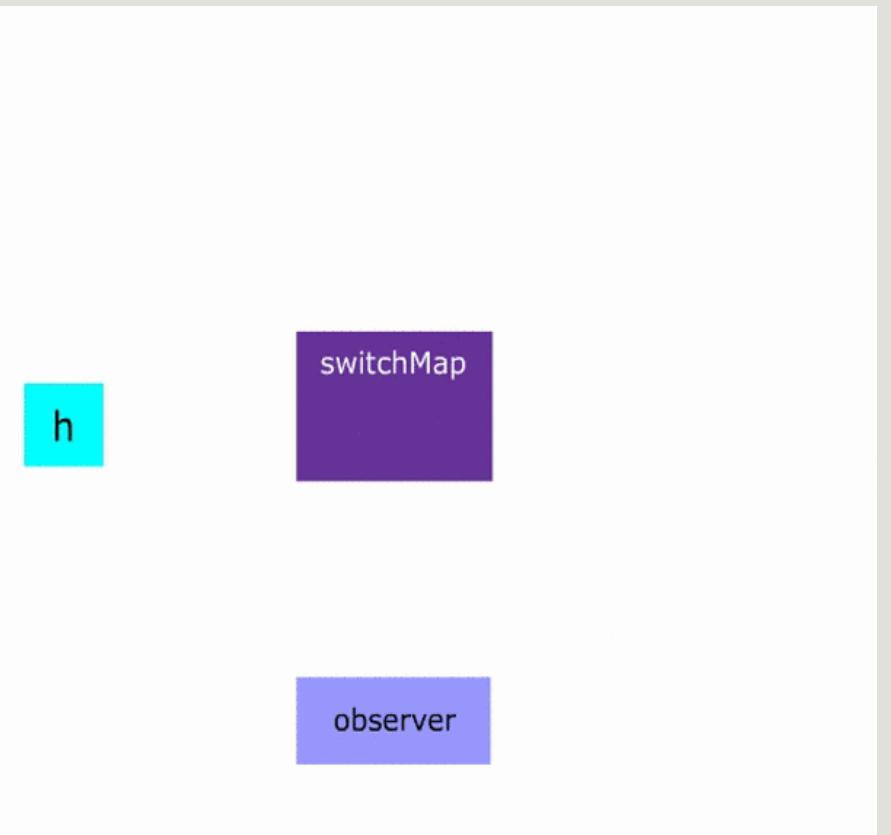
SwitchMap

- L'opérateur **switchMap** est essentiellement une combinaison de deux opérateurs - **switchAll et map**.
- La partie de **map** vous permet de mapper une valeur d'une source observable d'ordre supérieur à un flux observable interne.
- La partie **switch** s'abonne à l'observable interne fourni le plus récemment émis par un observable d'ordre supérieur, en **se désabonnant de tout observable interne précédemment souscrit**.
- L'opérateur switchMap effectue toutes les actions suivantes :
 - **Annule et se désabonne automatiquement** du second observable lorsque le premier émet une nouvelle valeur.
 - Se **désabonne automatiquement du second observable** si nous nous **désinscrivons du premier**.
 - S'assure que les deux observables se **produisent en séquence**, l'un après l'autre.

Les opérateurs d'applatissement/ flattening operators

SwitchMap

- **switchMap** n'a qu'un seul abonnement actif à la fois à partir duquel les valeurs sont transmises à un observateur.
- Une fois que l'observable d'ordre supérieur émet une nouvelle valeur, **switchMap** exécute la fonction pour obtenir un nouveau flux observable interne et commute les flux. Il se désabonne du flux actuel et s'abonne au nouvel observable interne.



Les opérateurs d'applatissement/ flattening operators SwitchMap

- Une erreur courante commise dans les applications Angular consiste à **imbriquer les abonnements observables**.
- Cette syntaxe **n'est pas recommandée** car elle est **difficile à lire et peut entraîner des bogues des effets secondaires inattendus**.
- Par exemple, cette syntaxe rend **difficile la désinscription** correcte de tous ces observables.
- De plus, si observable1 émet plus d'une fois dans un court laps de temps, **nous pourrions vouloir annuler l'abonnement précédent à observable2** et en démarrer un nouveau basé sur les nouvelles données reçues d'observable1.

```
this.activatedRoute.params.subscribe(  
  (params) => {  
    this.cvService.findPersonneById(params.id)  
      .subscribe(  
        (personne) => {  
          this.personne = personne;  
        },  
        (erreur) => {  
          if (!this.personne) {  
            this.router.navigate(['cv']);  
          }  
        }  
      );  
  }  
);
```

Les opérateurs d'applatissement/ flattening operators SwitchMap

```
timerObs(timer: number, name: string, iteration = 4) {  
  return new Observable((observer: Observer<string>) => {  
    let i = 0;  
    const x = setInterval(() => {  
      if (i >= iteration) {  
        observer.complete();  
        clearInterval(x);  
      }  
      observer.next(`observable ${name} ${++i}`);  
    }, timer);  
  });  
}
```

```
params = [  
  {name: 'obs1', timer: 1000, iteration: 4 },  
  {name: 'obs2', timer: 1500, iteration: 4 },  
];  
constructor(private rxjsService: RxjsService) {  
  from(this.params).pipe(  
    switchMap((param) => thisrxjsService.timerObs(param.timer,  
param.name))  
  ).subscribe(  
    (data) => console.log(data)  
  )  
}
```

observable	obs2	1
observable	obs2	2
observable	obs2	3
observable	obs2	4

Les opérateurs d'applatissement/ flattening operators

SwitchMap

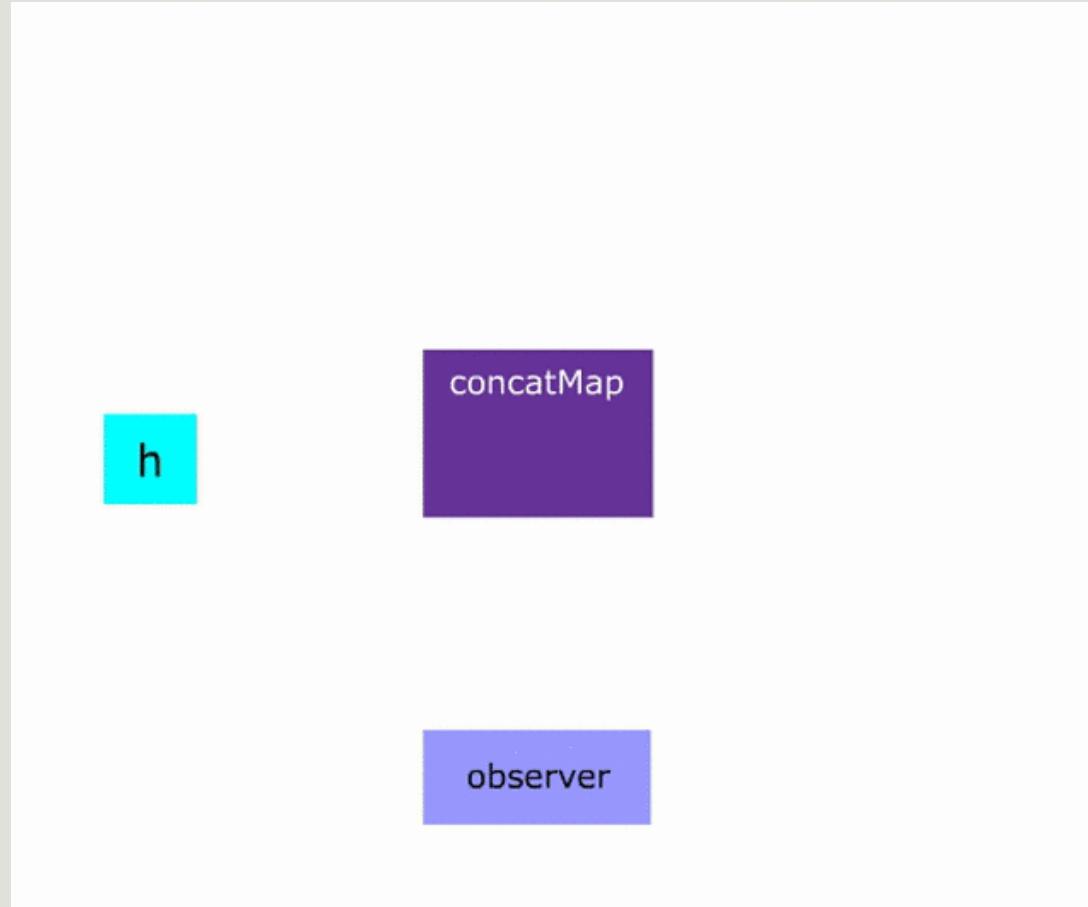
```
export class SwitchMapComponent implements AfterViewInit {
  search: string = '';
  @ViewChild('searchInput') searchInput!: ElementRef;
  constructor(private rxjsService: RxjsService) {}
  ngAfterViewInit (): void {
    const change$ = fromEvent<KeyboardEvent>(this.searchInput.nativeElement, 'keyup');
    change$.pipe(
      map((event: KeyboardEvent) => this.search),
      debounceTime(300),
      switchMap((search) => {
        return this.rxsService.getProducts(search)
      })
    )
    .subscribe(
      (input) => {//...Do what you want with your products}
    )
  }
}
```

Les opérateurs d'applatissement/ flattening operators

ConcatMap

- L'opérateur **concatMap** est essentiellement une combinaison de deux opérateurs - **concat et map**.
- La partie map vous permet de mapper une valeur d'une source observable à un flux observable. Ces flux sont souvent appelés flux internes.
- La partie **concat combine** tous les flux internes observables renvoyés par la map et **émet séquentiellement toutes les valeurs** de chaque flux d'entrée.
- Utilisez cet opérateur **si l'ordre des émissions est important** et que vous souhaitez d'abord voir les valeurs émises par les flux qui passent par l'opérateur en premier.

Les opérateurs d'applatissement/ flattening operators ConcatMap



Les opérateurs d'applatissement/ flattening operators ConcatMap

```
timerObs(timer: number, name: string, iteration = 4) {
  return new Observable((observer: Observer<string>) => {
    let i = 0;
    const x = setInterval(() => {
      if (i >= iteration) {
        observer.complete();
        clearInterval(x);
      }
      observer.next(`observable ${name} ${++i}`);
    }, timer);
  });
}
```

observable obs1 1
observable obs1 2
observable obs1 3
observable obs1 4
observable obs2 1
observable obs2 2
observable obs2 3
observable obs2 4

```
params = [
  {name: 'obs1', timer: 1000, iteration: 4 },
  {name: 'obs2', timer: 1500, iteration: 4 },
];
constructor(private rxjsService: RxjsService) {
  from(this.params).pipe(
    concatMap((param) => this.rxsService.timerObs(param.timer,
param.name))
  ).subscribe(
    (data) => console.log(data)
  )
}
```

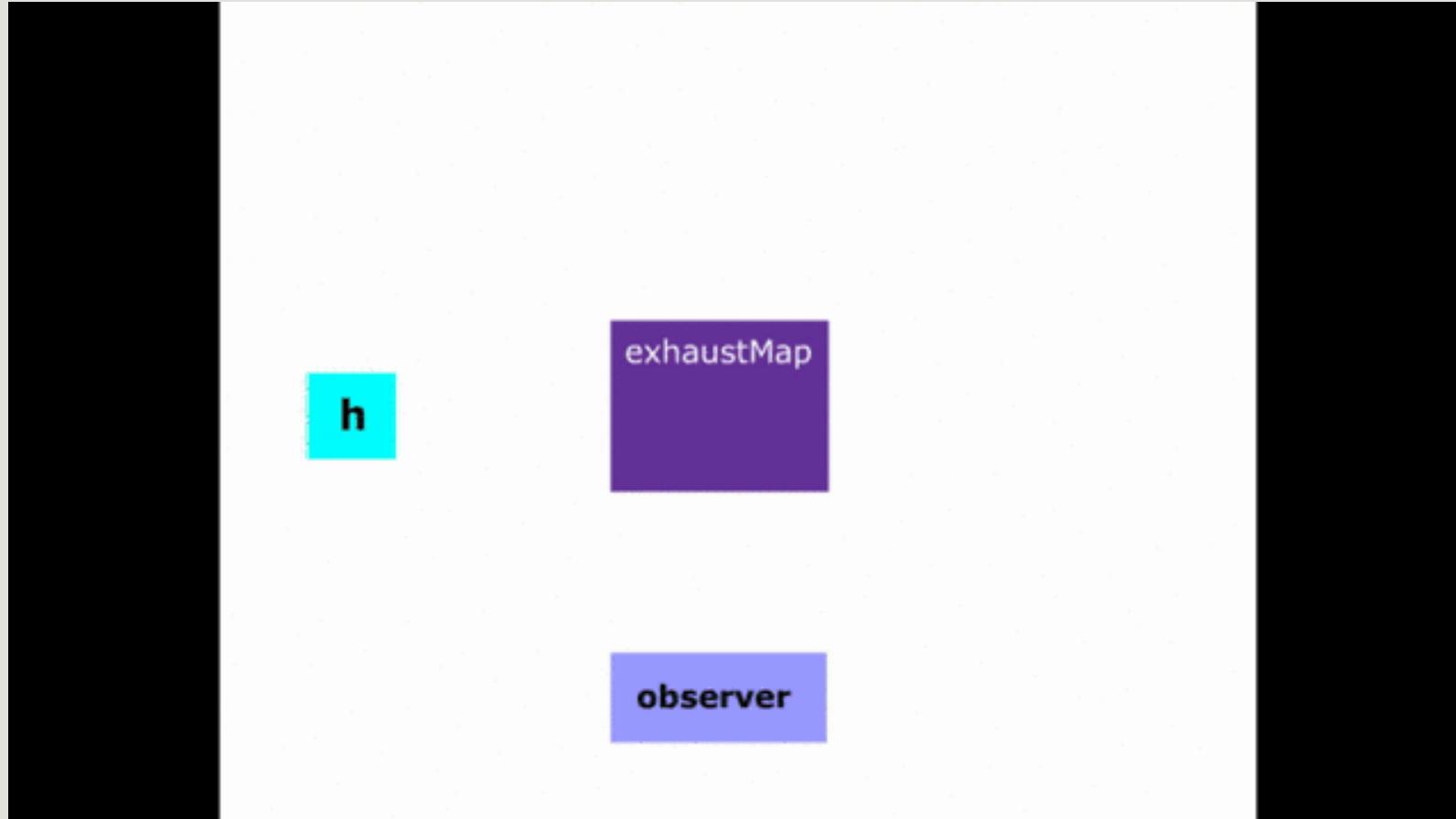
Les opérateurs d'applatissement/ flattening operators

ExhaustMap

- L'opérateur **exhaustMap** est essentiellement une combinaison de deux opérateurs - **exhaust** et **map**.
- La partie **map** vous permet de mapper une valeur d'une source observable d'ordre supérieur à un flux observable interne.
- La partie **exhaust** s'abonne ensuite à un observable interne et transmet des valeurs à un observateur s'il n'y a pas déjà d'abonnement actif, sinon il ignore simplement les nouveaux observables internes.
- **exhaustMap** n'a **qu'un seul abonnement actif à la fois** à partir duquel les valeurs sont transmises à un observateur. Lorsque l'observable d'ordre supérieur émet un nouveau flux observable interne, **si le flux actuel n'est pas terminé, ce nouvel observable interne est abandonné**.
- Une fois le flux actif en cours terminé, l'opérateur attend qu'un autre observable interne s'abonne en ignorant les observables internes précédents.

Les opérateurs d'applatissement/ flattening operators

ExhaustMap



Les opérateurs d'applatissement/ flattening operators

ExhaustMap

```
timerObs(timer: number, name: string, iteration = 4) {  
  return new Observable((observer: Observer<string>) => {  
    let i = 0;  
    const x = setInterval(() => {  
      if (i >= iteration) {  
        observer.complete();  
        clearInterval(x);  
      }  
      observer.next(`observable ${name} ${++i}`);  
    }, timer);  
  });  
}
```

```
params = [  
  {name: 'obs1', timer: 1000, iteration: 4 },  
  {name: 'obs2', timer: 1500, iteration: 4 },  
];  
constructor(private rxjsService: RxjsService) {  
  from(this.params).pipe(  
    exhaustMap((param) => thisrxjsService.timerObs(param.timer,  
param.name))  
  ).subscribe(  
    (data) => console.log(data)  
  )  
}
```

observable	obs1	1
observable	obs1	2
observable	obs1	3
observable	obs1	4

Les opérateurs d'applatissement/ flattening operators combineLatest

- **combineLatest** permet de fusionner plusieurs flux en prenant la valeur la plus récente de chaque entrée observable et en émettant ces valeurs à l'observateur sous forme de sortie combinée (généralement sous forme de tableau).
- Les opérateurs **mettent en cache la dernière valeur pour chaque observable d'entrée** et seulement une fois que **tous les observables d'entrée ont produit au moins une valeur**, il **émet les valeurs mises en cache combinées à l'observateur**.
- Le flux résultant **se termine lorsque tous les flux internes sont terminés** et génère une **erreur si l'un des flux internes génère une erreur**.
- D'autre part, si **un flux n'émet pas de valeur mais se termine**, le **flux résultant se terminera au même moment sans rien émettre**.
- De plus, si un flux d'entrée n'émet aucune valeur et ne se termine jamais, combineLatest n'émettra jamais et ne se terminera jamais.

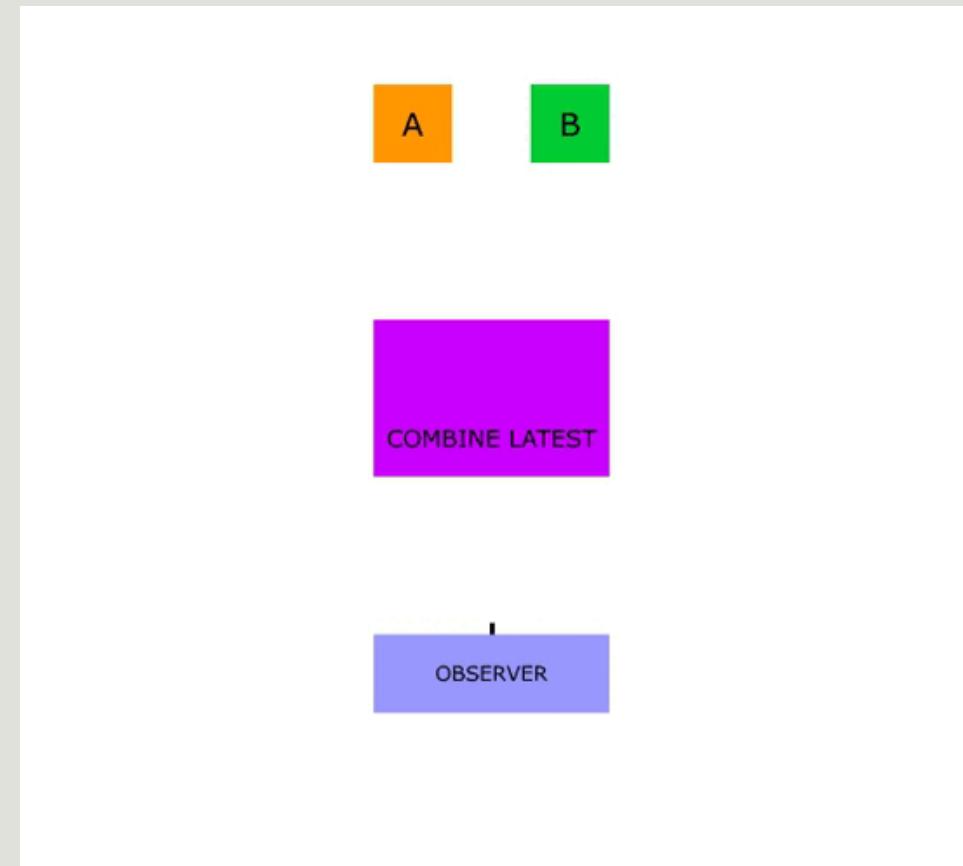
Les opérateurs d'applatissement/ flattening operators combineLatest

```
const cv$ = this.cvService.loadCvById(cvId)
  .pipe(startWith(null));

const skills$ = this.cvService.loadAllCvSkills(cvId);

this.data$ = combineLatest([cv$, skills$])
  .pipe(
    map(([cv, skills]) => {
      return {
        cv,
        skills
      }
    })
  );

```



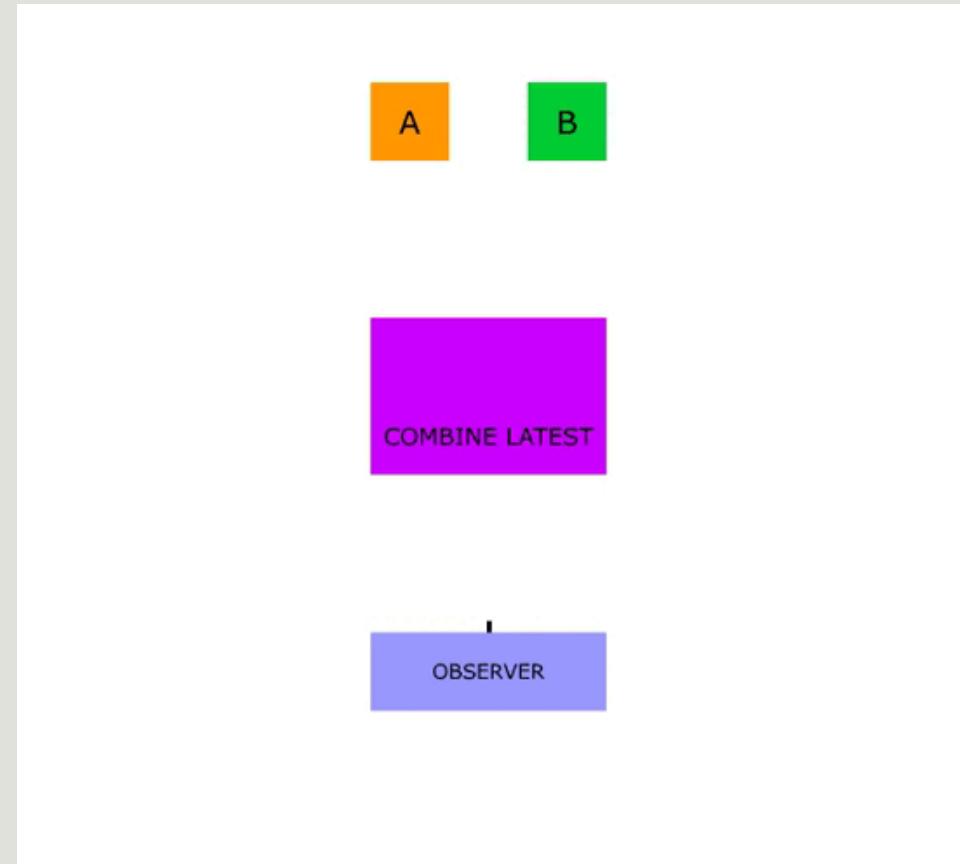
Les opérateurs d'applatissement/ flattening operators combineLatest

```
const cv$ = this.cvService.loadCvById(cvId)
    .pipe(startWith(null));

const skills$ = this.cvService.loadAllCvSkills(cvId)
    .pipe(startWith([]));

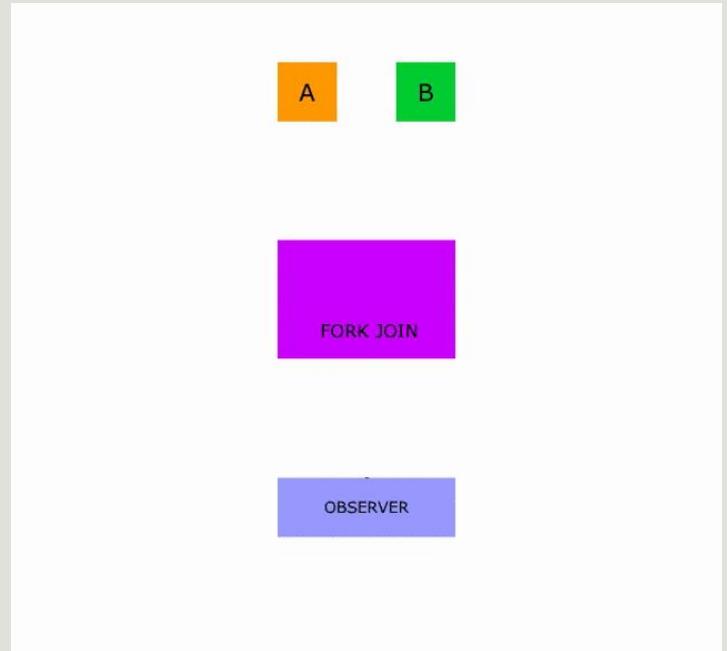
this.data$ = combineLatest([cv$, skills$])
    .pipe(
        map(([cv, skills]) => {
            return {
                cv,
                skills
            }
        })
    );

```



Les opérateurs d'applatissement/ flattening operators forkJoin

- **forkJoin** peut être appliqué à n'importe quel nombre d'Observables. Lorsque **tous ces Observables sont terminés**, **forkJoin** émet la **dernière valeur de chacun d'eux**.
- Un cas d'utilisation courant pour **forkJoin** est lorsque nous devons **déclencher plusieurs requêtes HTTP en parallèle** avec le **HttpClient** puis **recevoir toutes les réponses en même temps dans un tableau de réponses**.



Les opérateurs d'applatissement/ flattening operators forkJoin

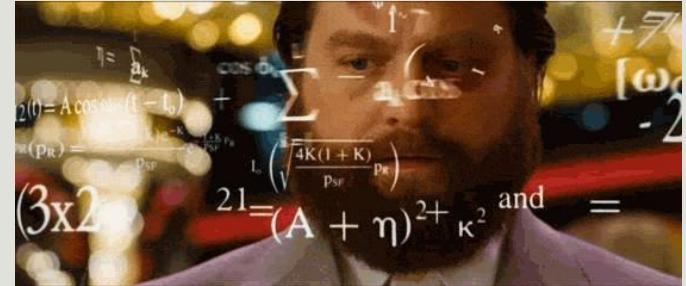
```
forkJoin({
  users: http.get<User[]>('https://jsonplaceholder.typicode.com/users'),
  posts: http.get<Post[]>('https://jsonplaceholder.typicode.com/posts'),
}).subscribe((usersAndPosts) => {
  this.posts = usersAndPosts.posts;
  this.users = usersAndPosts.users;
});
```

```
▼ Object ⓘ
  ► posts: (100) [...], ...
  ► users: (10) [...], ...
```

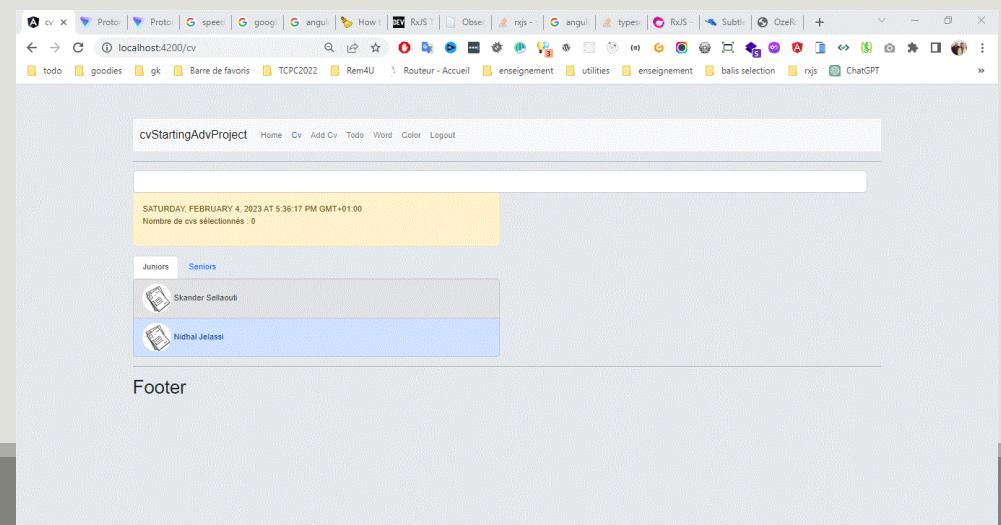
```
forkJoin([
  http.get<User[]>('https://jsonplaceholder.typicode.com/users'),
  http.get<Post[]>('https://jsonplaceholder.typicode.com/posts'),
]).subscribe(([users, posts]) => {
  this.posts = posts;
  this.users = users;
});
```

```
► (10) [...], ..., ..., ..., ...
(100) [...], ..., ..., ..., ..., ...
```

Exercice



- Sachant que pour sélectionner une personne dont le nom contient une chaîne donnée, loopback utilise la syntaxe suivante :
`{ "where": { "name": { "like": "%$name%" } } }`
- Ceci doit être fourni dans les paramètres de votre requête avec la clé `filter`. Tester le sur votre swagger.
- Ajouter un champ `input` dans la page Cv. A chaque caractère saisi, la liste des choix doit automatiquement changer et n'afficher que les cvs qui contiennent la chaîne saisie. En sélectionnant un des choix, rediriger l'utilisateur vers les détails du cv sélectionné.
- Faites en sorte d'avoir le code le plus récative possible.



Les opérateurs RxJs

Gestion des erreur

catchError

- L'opérateur **catchError** de RxJs permet de **capturer les erreurs qui se produisent dans un observable** et de les traiter de manière appropriée.
- Il permet de **continuer à émettre des valeurs depuis un observable** même si une erreur se produit, plutôt que de stopper complètement l'émission de valeurs.
- Il prend en argument une fonction qui prend en entrée l'erreur et **retourne un observable** pour gérer cette erreur.
- Si vous voulez retourner l'erreur dans votre flux utilisez l'opérateur **throwError**.

Les opérateurs RxJs

Gestion des erreurs

catchError

```
this.activatedRoute.params.pipe(  
    switchMap((param) => this.cvService.getCvById(param['id'])),  
    catchError((e) => {  
        console.log(e);  
        this.router.navigate([APP_ROUTES.cv]);  
        return throwError(() => new Error(e));  
    })  
);
```

Les opérateurs RxJs

Gestion des erreur

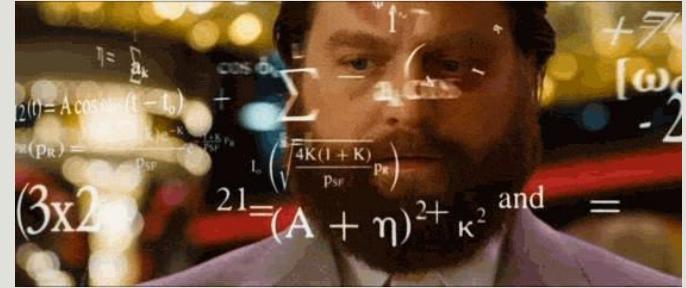
catchError

- Vous pouvez utiliser l'opérateur **EMPTY** de rxjs pour spécifier que vous **n'émettez rien** et qui émet ensuite une notification de **complétion** du flux.
- Vous pouvez aussi émettre ce que vous voulez afin de **continuer le flux**

```
catchError((e) => {  
    return EMPTY;  
}),
```

```
catchError((e) => {  
    return of(this.myService.getFakeData());  
}),
```

Exercice



- Reprenez les différentes fonctionnalités et faites en sorte que le code devienne réactif.
- Utilisez l'async pipe à chaque fois que vous le pouvez.
- Utilisez catchError pour gérer les erreurs.

Quelques opérateurs utiles de l'Observable

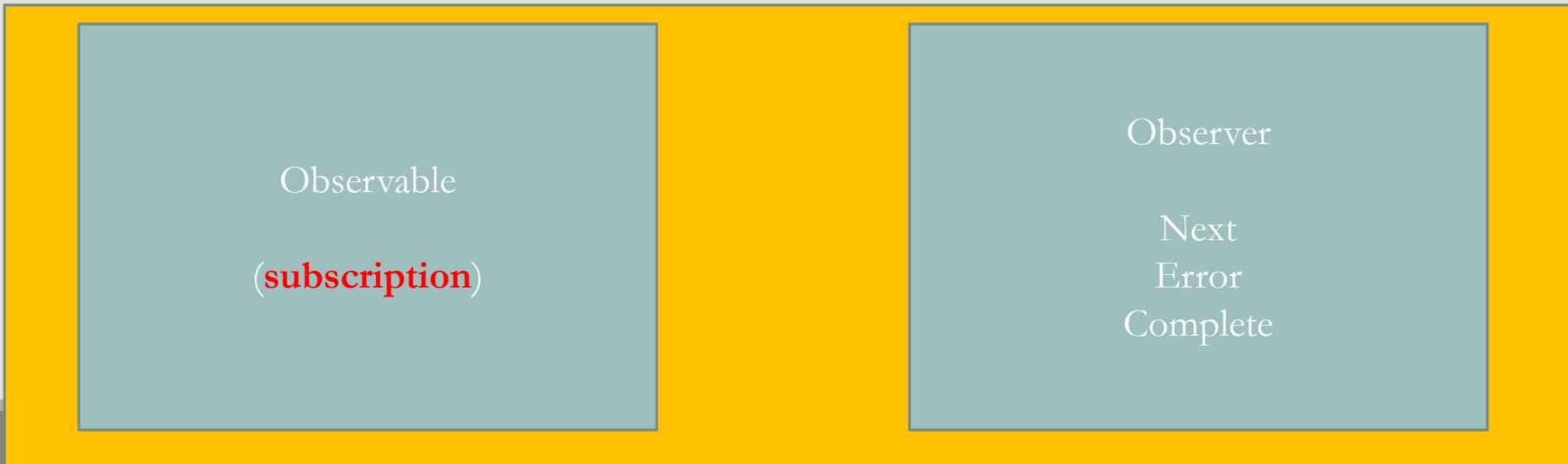
<https://angular.io/guide/rx-library>

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

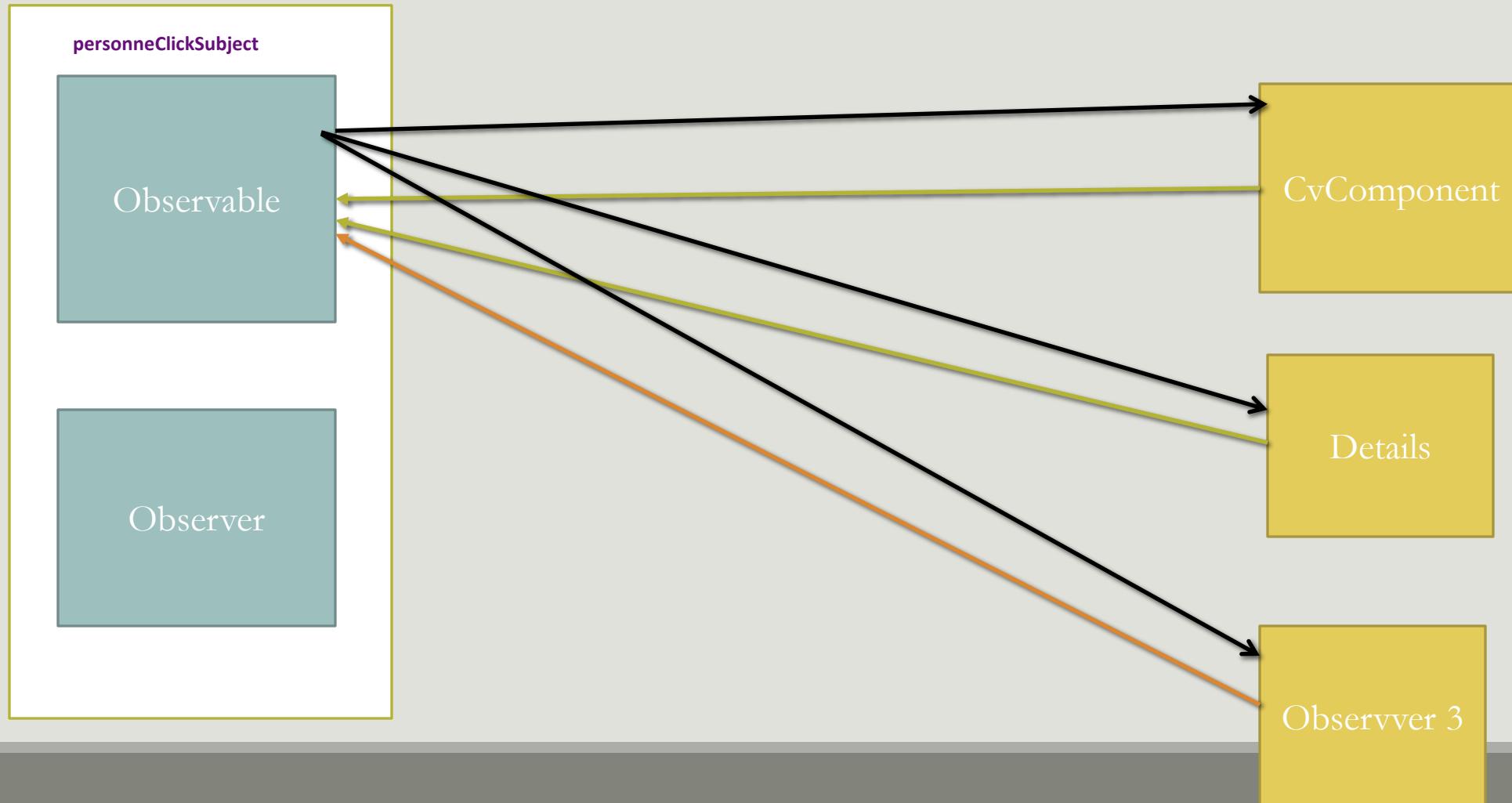
<http://rxmarbles.com/>

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



Les behaviourSubject

- Les **BehaviourSubject** sont une variante du **Subject**.
- Le **BehaviourSubject** a la particularité de **stocker la valeur "actuelle"**. Cela signifie que vous pouvez **toujours obtenir directement la dernière valeur émise** à partir du **BehaviourSubject**.
- Il existe deux façons d'obtenir cette dernière valeur émise. Vous pouvez soit **obtenir la valeur en accédant à la propriété value** sur le **BehaviourSubject**,
- Soit **vous y abonner**. Si vous vous y abonnez, le **BehaviourSubject émettra directement** la valeur actuelle à l'abonné. **Même si l'abonné s'abonne beaucoup plus tard que la valeur a été stockée**.
- Vous pouvez **créer un BehaviourSubject avec une valeur initiale** qui sera donc émise directement pour la première inscription.

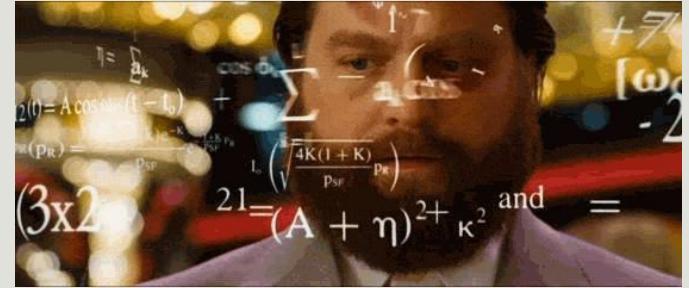
Les ReplaySubject

- Le **ReplaySubject** est comparable au BehaviorSubject dans la mesure où il peut envoyer les "anciennes" valeurs aux nouveaux abonnés.
- Il a cependant la particularité supplémentaire de pouvoir **enregistrer une partie de l'exécution de l'observable** et donc de **stocker plusieurs anciennes valeurs** et de les "rejouer" aux nouveaux abonnés.
- Lors de la création du ReplaySubject, vous pouvez spécifier **la quantité de valeurs que vous souhaitez stocker** et **pendant combien de temps** vous souhaitez les stocker.

Les AsyncSubject

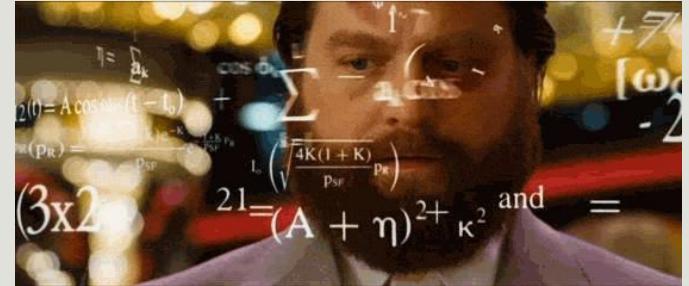
- Alors que BehaviorSubject et ReplaySubject stockent tous deux des valeurs, **AsyncSubject** fonctionne un peu différemment. L'AsyncSubject est une variante de l'objet où **seule la dernière valeur** de l'exécution Observable est **envoyée à ses abonnés**, et **uniquement lorsque l'exécution est terminée**.

Exercice



- Actuellement, dans notre application, nous utilisant une fonction isAuthenticated qui retourne true ou false si le user est authentifié ou non.
- D'un autre coté nous ne sauvegardons aucune information sur le user connecté.
- Nous voulons créer un store pour la gestion des utilisateurs nous permettant de garder la trace du user authentifié ainsi que de gérer l'état de l'utilisateur, à savoir qu'il est authentifié ou non.
- Choisissez la bonne structure

Exercice



- Faites en sorte d'avoir deux tabs dans la liste des cvs.
- Le premier tab affichera les cvs des juniors (age < 40)
- Le second tab affichera les cvs des seniors (age ≥ 40)

The screenshot shows a web application titled "cvStartingAdvProject" running on a local host. The browser's address bar shows "localhost:4200/cv". The main content area displays a list of cvs for two users: Skander Sellaouti and Nidhal Jelassi. The application has a navigation bar with links to Home, Cv, Add Cv, Todo, Word, Color, and Logout. Below the navigation bar, there is a message: "SATURDAY, FEBRUARY 4, 2023 AT 3:04:55 PM GMT+01:00" and "Nombre de cvs sélectionnés : 0". At the bottom of the page, there is a footer section.

Opérateur de Multicasting share

- L'opérateur **share** va **multicatser** les valeurs émises par une source Observable pour les abonnés.
- Les multicatser signifie que **les données sont envoyées à plusieurs destinations**.
- En tant que tel, le partage vous permet **d'éviter plusieurs exécutions de la source Observable** lorsqu'il existe plusieurs abonnements.
- **share** est particulièrement utile si vous avez besoin d'**empêcher des appels d'API répétés** ou des **opérations coûteuses exécutées par Observables**.

Opérateur de Multicasting share

```
const sharedSource$ = interval(1000).pipe(  
    tap((ind) => console.log('Processing: ', ind)),  
    map(() => Math.round(Math.random() * 100)),  
    take(2),  
    share()  
);  
sharedSource$.subscribe((x) => console.log('subscription 1: ', x));  
sharedSource$.subscribe((x) => console.log('subscription 2: ', x));  
setTimeout(  
    () => sharedSource$.subscribe((x) => console.log('subscription 3: ', x)),  
    1500  
);
```

Opérateur de Multicasting share

- Lorsque vous vous abonnez à un Observable partagé, vous vous abonnez en fait à un **Subject** exposé par l'opérateur de partage.
- L'opérateur de partage gère également **un abonnement interne à la source Observable**.
- Le **Subject** interne **est la raison pour laquelle plusieurs abonnés reçoivent la même valeur partagée**, car ils reçoivent des valeurs du sujet exposé par l'opérateur de partage.
- **share** tient un **compte des abonnés**.
- Une fois que le **nombre d'abonnés atteint 0**, **share** se désabonnera de l'Observable source et réinitialisera son Observable interne (le **Subject**).
- L'abonné (**en retard**) suivant **déclenchera un nouvel abonnement** à l'Observable source, ou en d'autres termes, une nouvelle exécution de l'Observable source.

Opérateur de Multicasting share

```
const sharedSource$ = interval(1000).pipe(  
    tap((ind) => console.log('Processing: ', ind)),  
    map(() => Math.round(Math.random() * 100)),  
    take(2),  
    share()  
);  
sharedSource$.subscribe((x) => console.log('subscription 1: ', x));  
sharedSource$.subscribe((x) => console.log('subscription 2: ', x));  
setTimeout(  
    () => sharedSource$.subscribe((x) => console.log('subscription 3: ', x)),  
    4500  
);
```

Opérateur de Multicasting shareReplay

- Dans certains cas, ce dont vous avez vraiment besoin, c'est d'un partage capable de se comporter comme le ferait un **ReplaySubject**.
- Nous voulons donc **qu'il partage à la fois la source Observable et rejoue les dernières émissions** pour les abonnés retardataires.
- Donc, il ne conserve pas le nombre d'abonnés par défaut, mais vous pouvez utiliser l'option **refCount** avec une valeur true pour activer ce comportement.
- Contrairement à share, shareReplay expose un **ReplaySubject** aux abonnés. **ReplaySubject()**.

Opérateur de Multicasting shareReplay

```
const sharedSource$ = interval(1000).pipe(  
    tap((ind) => console.log('Processing: ', ind)),  
    map(() => Math.round(Math.random() * 100)),  
    take(2),  
    shareReplay()  
);  
sharedSource$.subscribe((x) => console.log('subscription 1: ', x));  
sharedSource$.subscribe((x) => console.log('subscription 2: ', x));  
setTimeout(  
    () => sharedSource$.subscribe((x) => console.log('subscription 3: ', x)),  
    4500  
);
```

Opérateur de Multicasting shareReplay

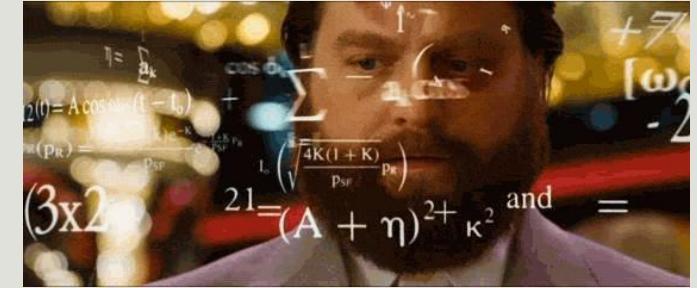
- Étant donné que **shareReplay** ne garde pas la trace d'un nombre d'abonnés par défaut, il **n'est pas en mesure de se désabonner de la source Observable**.
- Ceci est faisable si vous utilisez l'option **refCount**.
- Afin d'utiliser **shareReplay** tout en vous débarrassant des problèmes de **fuite de mémoire**, vous pouvez utiliser les options **bufferSize** et **refCount : shareReplay({ bufferSize : 1, refCount : true })**.
- **shareReplay ne réinitialise jamais son ReplaySubject interne** lorsque **refCount atteint 0**, mais **se désabonne** de la source Observable.
- Les abonnés en retard ne déclencheront pas une nouvelle exécution de la source Observable et recevront jusqu'à **N (bufferSize)** émissions.

Opérateur de Multicasting shareReplay

```
const sharedSource$ = interval(1000).pipe(  
    tap((ind) => console.log('Processing: ', ind)),  
    map(() => Math.round(Math.random() * 100)),  
    take(2),  
    shareReplay({ bufferSize: 2, refCount: true })  
);  
sharedSource$.subscribe((x) => console.log('subscription 1: ', x));  
sharedSource$.subscribe((x) => console.log('subscription 2: ', x));  
setTimeout(  
    () => sharedSource$.subscribe((x) => console.log('subscription 3: ', x)),  
    4500  
);
```

Exercice

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



Se désinscrire

- La méthode **subscribe** permet de s'inscrire à un **observable**.

- **Problème :** Cette souscription reste valide même après la disparition de la variable ce qui sature la mémoire pour rien.

Se désinscrire complete

- Lorsqu'un observable se termine (méthode `complete`), tous les abonnements sont automatiquement désabonnés.
- Si vous savez qu'un observable se terminera, vous n'avez pas à vous soucier de nettoyer les abonnements.
- Cela est vrai pour tout observable créé à partir de plusieurs des sources observables intégrées dans Angular telles que les méthodes du `HttpClient` ou le service `ActivatedRoute`.
- Il n'y a pas de convention standard qui indique ce que les observables peuvent terminer, ou quand, il appartient donc au développeur de faire les recherches nécessaires.

Se désinscrire async pipe

- La première et la plus courante solution pour **gérer les abonnements** est **l'async pipe**.
- Plutôt que de vous abonner à des observables dans vos composants et de définir des propriétés sur votre classe pour les données de l'observable, préférez l'utilisation de **l'async pipe**.
- L'async **pipe nettoie ses propres abonnements** quand et selon les besoins, vous déchargeant de cette responsabilité. Maintenant, cela devrait être **votre premier choix et fonctionnera pour la plupart des scénarios**.

Se désinscrire async pipe

```
export class TestUnsubscribeComponent {  
  todos$: Observable<Todo[]>;  
  firstTodo$: Observable<Todo>;  
  constructor(private todoService: TodoService) {  
    this.todos$ = this.todoService.getTodos();  
    this.firstTodo$ = this.todoService.getTodo(1);  
  }  
}
```

```
<div *ngIf="firstTodo$ | async as firstTodo">  
  The first todo is :  
  {{ firstTodo.id }} | {{firstTodo.title}}  
</div>  
<ol>  
  <li *ngFor="let todo of todos$ | async">  
    <pre>{{todo | json}}</pre>  
  </li>  
</ol>
```

Se désinscrire

Et si on peut pas utiliser l'async pipe ?

- Que faire lorsque vous ne pouvez pas utiliser l'async pipe ? Il y a plusieurs options.
- La première et la plus simple consiste à utiliser un **Subscription** pour **collecter tous les abonnements** que vous créez, vous permettant de les nettoyer plus tard au moment opportun.
- L'objet **Subscription** dans RxJs n'est pas seulement représentatif d'un seul **subscription** à un flux. Il est également représentatif d'un **agrégat de subscription** qui contient d'autres abonnements. Cela lui permet d'être utilisé comme point de contrôle unique pour n'importe quel nombre d'abonnements.
- Pour se faire, utilisez sa méthode **add** qui prend en paramètre une **subscription**.

Se désinscrire

Et si on peut pas utiliser l'async pipe ?

```
export class TestUnsubscribeComponent implements OnDestroy{
  todos$: Observable<Todo[]>;
  firstTodo$: Observable<Todo>;
  subscription: Subscription;
  nb = 0;
  constructor(private todoService: TodoService, private testService: TestService ) {
    this.todos$ = this.todoService.getTodos();
    this.firstTodo$ = this.todoService.getTodo(1);
    this.subscription = this.testService.click$.subscribe(() => this.nb++);
  }

  ngOnDestroy(): void {
    this.subscription.unsubscribe();
  }
}
```

Se désinscrire

Et si on peut pas utiliser l'async pipe ?

```
export class TestUnsubscribeComponent implements OnDestroy{
  todos$: Observable<Todo[]>;
  firstTodo$: Observable<Todo>;
  subscription: Subscription = new Subscription();
  nbClick = 0;
  nbUpdates = 0;
  constructor(private todoService: TodoService, private testService: TestService ) {
    this.todos$ = this.todoService.getTodos();
    this.firstTodo$ = this.todoService.getTodo(1);
    this.subscription.add(this.testService.click$.subscribe(() => this.nbClick++));
    this.subscription.add(this.testService.update$.subscribe(() => this.nbUpdates++));
  }
  ngOnDestroy(): void {
    this.subscription.unsubscribe();
  }
}
```

Se désinscrire

Utiliser l'opérateur takeUntilDestroyed

- L'opérateur **takeUntilDestroyed** de [@angular/core/rxjs-interop](#) vous permet de gérer **l'unsubscription**.
- Cette fonctionnalité est **disponible à partir d'angular 16**

```
this.myObservable$.pipe(  
  map((element) => element * 3),  
  filter((element) => element % 2 == 0),  
  takeUntilDestroyed()  
);
```

Se désinscrire

Utiliser un signal

- Une autre façon de se désabonner, et peut-être une manière plus élégante, consiste à utiliser des **signaux**.
- Les signaux sont un moyen d'utiliser d'autres flux pour contrôler les flux auxquels vous êtes peut-être abonné.
- Les signaux sont obtenus avec l'opérateur **takeUntil** dans le pipe et tout autre observable. L'autre observable peut être un Subject ou un autre observable ; tout ce qui émet un next.

Se désinscrire

Et si on ne peut pas utiliser l'async pipe ?

```
export class TestUnsubscribeComponent implements OnDestroy{
  todos$: Observable<Todo[]>;
  signal$ = new Subject();
  nbClick = 0;
  constructor(private todoService: TodoService, private testService: TestService ) {
    this.todos$ = this.todoService.getTodos();
    this.testService.click$
      .pipe(takeUntil(this.signal$))
      .subscribe(() => this.nbClick++);
  }
  ngOnDestroy(): void {
    this.signal$.next('i am emitting stop emitting on your side :S');
    this.signal$.complete();
  }
}
```

signaux et rxJs

Interopérabilité

- ▶ Les signaux **ne sont pas là pour enlever RxJs** mais plutôt pour **limiter** son utilisation là où il est le **plus approprié** de l'utiliser.
- ▶ **RxJS** se marie mieux avec les **tâches asynchrones** alors que les **signaux** sont **synchrone**s.
- ▶ Nous **pouvons utiliser des signaux et des observables ensemble**, et nous pouvons **les convertir l'un en l'autre**.
- ▶ **Deux fonctions** pour faire cela sont disponibles dans le tout nouveau package : [**@angular/core/rxjs-interop**](#)
 - ▶ **toObservable qui convertit un signal en un observable**
 - ▶ **toSignal qui convertit un observable en un signal.** Elle prend en premier paramètre l'observable et en second paramètre un objet **ToSignalOptions** permettant de définir **la valeur initiale**.

signaux et rxJs

Interopérabilité

```
cvs = toSignal(  
    this.todoService.getTodos(),  
    { initialValue: [] }  
);
```

Quand utiliser Les signaux

- Les signaux sont excellents pour **gérer l'état**
- Les signaux **ont toujours une valeur**.
- Tout ce qui est **synchrone** peut être modélisé avec des signaux.
- Les signaux **n'émettent rien** : un **consommateur doit extraire leur valeur en cas de besoin, et une valeur sera toujours renvoyée (de manière synchrone)**.

Quand utiliser RxJs

- Vous avez besoin d'un **observable**, et non d'un **signal**, lorsque :
 - Vous vous souciez du **moment** où la valeur sera émise ;
 - Vous **ne pouvez pas renvoyer une valeur instantanément** ;
 - Vous vous souciez de **la complétion** : un **observable** peut être **complet**, un **signal** est **permanent** ;
 - vous devez **être notifié de chaque modification** ;
 - vous souhaitez **filtrer ou retarder le moment où vous émettez la valeur**.
- **Événements DOM**, requêtes **XHR**, **saisie utilisateur** : vous pouvez facilement les utiliser avec **debounceTime()**, **concatMap()**, **take(1)**, **skip(2)**, **forkJoin()**, et annuler les requêtes avec **switchMap()**.
- Si votre source de données ne peut pas répondre à la question « **Quelle est la valeur actuelle ?**» **sans délai**, elle **ne peut pas être présentée comme un signal**.
- Les signaux se distinguent par un autre domaine : **la représentation de l'état de l'application**.

signaux et rxJs

Interopérabilité

- Les signaux **ne sont pas là pour enlever RxJs** mais plutôt pour **limiter** son utilisation là où il est le **plus approprié** de l'utiliser.
- **RxJS** se marie mieux avec les **tâches asynchrones** alors que les **signaux** sont **synchrone**s.
- Nous **pouvons utiliser des signaux et des observables ensemble**, et nous pouvons **les convertir l'un en l'autre**.
- **Deux fonctions**, pour faire cela, sont disponibles dans le tout package : **@angular/core/rxjs-interop**
 - **toObservable** qui convertit un **signal** en un **observable**
 - **toSignal** qui convertit un **observable** en un **signal**. Elle prend en premier paramètre l'observable et en second paramètre un objet **ToSignalOptions** permettant de définir **la valeur initiale**.

signaux et rxJs toObservable

- **toObservable** permet de **convertir un signal en un Observable**
- Pour ce faire, Angular utilise un **effect**.
- Cet effect utilise un **ReplaySubject** et **ajoute une valeur dans le flux à chaque fois que le signal change**.
- Puisque c'est un ReplaySubject chaque **nouvelle inscription reçoit la dernière valeur**.
- Vous devez être dans un **contexte d'injection** afin **d'utiliser** le **toObservable**.
- Le **deuxième paramètre de toObservable** est un objet d'option de type **ToObservableOptions** avec la seule propriété **injector**.

signaux et rxJs toObservable

```
/**  
 * Exposes the value of an Angular `Signal` as an RxJS `Observable`.  
 *  
 * The signal's value will be propagated into the `Observable`'s subscribers using an `effect`.  
 *  
 * `toObservable` must be called in an injection context unless an injector is provided via options.  
 *  
 * @publicApi 20.0  
 */  
declare function toObservable<T>(source: Signal<T>, options?: ToObservableOptions): Observable<T>;
```

```
interface ToObservableOptions {  
    /**  
     * The `Injector` to use when creating the underlying `effect` which  
     * watches the signal.  
     */  
    injector?: Injector;  
}
```

signaux et rxJs

ToSignalOptions

- **toSignal** permet de **convertir un observable en un Signal**
- **toSignal** s'inscrit à l'observable et crée un signal qu'il **modifie à chaque nouvelle valeur reçue du flux.**
- **toSignal** gère la désinscription de l'inscription. Il a donc besoin d'injecter l'injectable `DestroyRef` et donc d'un contexte d'injection.
- Le **deuxième paramètre de `toSignal`** et un objet d'option de type **ToSignalOptions**

```
interface ToSignalOptions<T> {  
    initialValue?: unknown;  
    requireSync?: boolean | undefined;  
    injector?: Injector | undefined;  
    manualCleanup?: boolean | undefined;  
    equal?: ValueEqualityFn<T> | undefined;  
}
```

signaux et rxJs toSignal

```
interface ToSignalOptions<T> {  
    initialValue?: unknown;  
    requireSync?: boolean | undefined;  
    injector?: Injector | undefined;  
    manualCleanup?: boolean | undefined;  
    equal?: ValueEqualityFn<T> | undefined;  
}
```

- **initialValue**: unknown : Définit une valeur initiale pour le Signal avant la première émission de l'Observable.
- **requireSync**: boolean : Exige que l'Observable émette immédiatement une valeur synchrone, sinon.
- **injector**: **Injector** : Spécifie l'injecteur pour gérer le nettoyage de l'abonnement à l'Observable.
- **manualCleanup**: boolean : Désactive le nettoyage automatique de l'abonnement, laissant la responsabilité à l'utilisateur.
- **equal**: ValueEqualityFn<T> : Fournit une fonction personnalisée pour déterminer si une nouvelle valeur justifie une mise à jour du Signal.

signaux et rxJs

toSignal

```
cvs = toSignal(  
    this.todoService.getTodos(),  
    { initialValue: [] }  
);
```

Optimisation

@defer

Prefetching

3
9
1

```
<input
  type="text" #myInput
  (keyup)="myInput.value.length > 4 ? loadDeffered = true : loadDeffered = false"
  class="form-control">
@defer(on hover; prefetch when loadDeffered) {
  <app-huge/>
}
@placeholder(minimum 1s) {
  <h2>Je suis là en attendant le vrai component:</h2>
}
@loading (minimum 1s; after 500ms) {
  <h3>loading.....</h3>
}
```

Signal et les appels HTTP

- Un **signal** est une opération **Synchrone**. Quand vous faites **un appel http** ceci revient à faire une **opération Asynchrone**.
- Afin de **briser le bridge entre ces deux mondes** et récupérer les informations concernant vos requêtes http pour **récupérer des données et les transformer en signals représentant votre State**, angular a introduit **3 API qui sont encore en Experimental** cad qu'elles peuvent ne pas devenir stables ou avoir des modifications significatives. Ces API's sont

FEATURE	PACKAGE	19	20
resource()	@angular/core	⚠	⚠
rxResource()	@angular/core/rxjs-interop	⚠	⚠
httpResource()	@angular/common/http	19.2	⚠

Signal et les appels HTTP

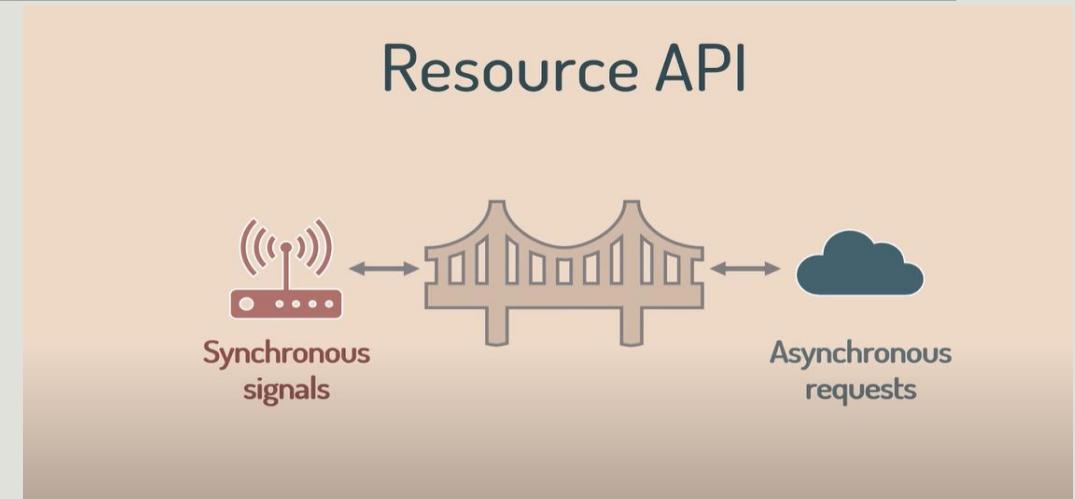
➤ Si on se pose la question suivante : De quoi pouvons nous avoir besoin lorsqu'on fait des appels HTTP ?

➤ Résultat de l'appel en cas de succès.

➤ Message d'erreur en cas d'erreur

➤ Savoir est ce que l'appel est en cours de traitement

➤ Pouvoir relancer un appel quand les paramètres dont il dépend changent



➤ Les nouveaux API Resource sont là pour faire le pont entre le monde asynchrone de RxJs et Synchrone des Signals et répondre à tous ces questions

Les resources API

Basique	resource()	Se base sur les promesses	Utilisez le si vous préférer l'api fetch de ES pour récupérer vos données
Reactive	rxResource()	Utilise les Observables RxJs	Utilisez le si vous préférez travailler avec RxJs pour vos appels http ou que vous avez besoins d'utiliser les opérateurs de RxJs
Pragmatique	La plus simple et la plus flexible.	Le dernier de la série et le plus simple à implémenter.	C'est un Wrapper sur le HttpClient, il nécessite une url et les paramètres dont il dépend. Offre une deuxième version pour les requêtes les plus complexes

resource

- L'API resource est conçue pour gérer **des données asynchrones** basées sur des **Promises**.
- Elle offre une **gestion déclarative des états asynchrones** et **annule automatiquement les requêtes précédentes si les paramètres changent** (sémantique switchMap).
- Elle **s'intègre parfaitement avec la zoneless change détection** vue que les mises à jour des signaux (**value et status**) appellent la fonction notify qui planifie le change detection.

resource

- L'API resource prend en paramètre un objet avec les paramètres suivants :

Dans ng20,
request a été
renommé en
params

Paramètre	Description	Obligatoire	Exemple
request	Fonction retournant les paramètres de la requête (ex. : ID, query).	Non	<code>() => ({ id: id() })</code>
params			
loader	Fonction retournant une Promise avec les données.	Oui	<code>({ request }) =></code> <code>fetch(...).then(res =></code> <code>res.json())</code>
defaultValue	Valeur initiale avant le chargement.	Non	<code>{ id: 0, title: '' }</code>
equal	Fonction de comparaison pour éviter les mises à jour inutiles.	Non	<code>(a, b) => a.id === b.id</code>
injector	Injecteur Angular pour résoudre les dépendances.	Non	<code>inject(Injector)</code>

resource

```
cvResource = resource<Cv | null, {id:number}>({  
    // fonction qui détermine à quel moment on va déclencher le loader  
    params: () => ({id: this.id()}),  
    // Loader va aller chercher le résultat du backend  
    loader: ({params, abortSignal, previous}) => {  
        return fetch(API.cv+ params.id).then(res => res.json());  
    },  
    defaultValue: null  
});
```

- **params (request)** prend en paramètre une **fonction** ou un **paramètre** spécifiant les **signals** dont dépend votre **resource**.
- **loader** permet de spécifier la **promesse** qui ira **chercher les données**
- **defaultValue** permet de spécifier la **valeur par défaut**

resource

Le loader

```
interface ResourceLoaderParams<R> {  
    params: NoInfer<Exclude<R, undefined>>;  
    abortSignal: AbortSignal;  
    previous: {  
        status: ResourceStatus;  
    };  
}
```

- Le loader **est déclenché dès l'initialisation de la ressource**.
- La ressource peut éventuellement disposer d'un signal qui est l'attribut request (params dans la version 20) qui définit les critères de recherche pour le loader.
- Le **loader** prend en **paramètre** un objet de type **ResourceLoaderParams**
 - **params** : Cette propriété contient les paramètres d'entrée ou les données nécessaires au calcul de la ressource. Son type est généralement déduit de la définition de la ressource, à l'exception de undefined.
 - **abortSignal** : Un objet AbortSignal permet d'annuler le chargement de la ressource en cours. Ceci est crucial pour gérer les requêtes asynchrones et prévenir les fuites de ressources lorsqu'une requête n'est plus nécessaire.
 - **previous** : Un objet contenant des informations sur l'état précédent de la ressource. Cela inclut généralement une propriété status indiquant le ResourceStatus (par exemple, « idle », « loading », « success », « error ») de la ressource avant la tentative de chargement en cours.
- **Chaque modification du signal params déclenche à nouveau le loader**.
- **ATTENTION** : L'attribut params (request) est traqué mais pas le loader. Le loader est dans un **untracked** ce qui signifie qu'un signal appelé dans le loader n'a aucun impact sur la recharge du loader.

resource

```
cvResource = resource<Cv | null, {id:number}>({  
    // fonction qui détermine à quel moment on va déclencher le loader  
    params: () => ({id: this.id()}),  
    // Loader va aller chercher le résultat du backend  
    loader: ({request, abortSignal, previous}) => {  
        return fetch(API.cv+request.id).then(res => res.json());  
    },  
    defaultValue: null  
});
```

- Afin de définir ce que retourne la params et le loader, vous pouvez le spécifier dans la partie generic de la resource : **resource<Cv | null, {id:number}>**
- Ici on l'informe que le **loader retourne un Cv ou null** et que le **params fournit un objet avec un id de type number**

Les propriétés d'une resource

- Tous les types de resources étendent l'interface Ressource :

```
interface Resource<T> {  
    /**  
     * The current value of the `Resource`, or `undefined` if there is no current value.*/  
    readonly value: Signal<T>;  
    /**  
     * The current status of the `Resource`, which describes what the resource is currently doing and  
     * what can be expected of its `value`.*/  
    readonly status: Signal<ResourceStatus>;  
    /**  
     * When in the `error` state, this returns the last known error from the `Resource`.*/  
    readonly error: Signal<unknown>;  
    /**  
     * Whether this resource is loading a new value (or reloading the existing one).*/  
    readonly isLoading: Signal<boolean>;  
    /**  
     * Whether this resource has a valid current value.  
     *  
     * This function is reactive.    */  
    hasValue(): this is Resource<Exclude<T, undefined>>;  
    /**  
     * Instructs the resource to re-load any asynchronous dependency it may have.  
     *  
     * Note that the resource will not enter its reloading state until the actual backend request is  
     * made.  
     *  
     * @returns true if a reload was initiated, false if a reload was unnecessary or unsupported  
     */  
    reload(): boolean;  
}
```

resource

- Donc **Tout objet qui étend Resource<T> encapsule une donnée asynchrone de type T** (par exemple, un objet JSON ou un tableau) et **fournit des signaux réactifs** pour suivre :
 - La **valeur** actuelle de la ressource.
 - **L'état** de la requête (en cours, réussie, échouée).
 - Les **erreurs** éventuelles.
 - Le **statut de chargement**.

resource value

- **value**: Signal<T> est un signal réactif qui contient la valeur actuelle de la ressource.
- Cette valeur est de type T (le type générique défini lors de la création de la ressource).
- Si **aucune valeur n'est disponible** (par exemple, avant le chargement ou en cas d'erreur), la valeur est **undefined**.
- Exemple d'utilisation : Dans un template Angular, on peut afficher la valeur avec `{{ resource.value() }}`. Cas d'usage : Accéder aux données chargées, comme un objet JSON d'une API.

resource

status

- status: **Signal<ResourceStatus>** est un signal réactif indiquant l'état actuel de la ressource. Dans la v19 status retourne un enum, dans la v20 elle s'est transformée en string
- Les valeurs possibles de **ResourceStatus** sont :
 - **Idle** : Aucun chargement en cours, état initial.
 - **Loading** : La ressource est en train de charger une nouvelle valeur.
 - **Resolved** : La requête a réussi, une valeur est disponible.
 - **Error** : La requête a échoué, une erreur est disponible.
 - **Reloading** : La ressource recharge une valeur existante.

```
<!-- v19 -->
@if (cvResource.status() === 2 ) {
  <div>Chargement...</div>
}

<!-- v20 -->
@if (cvResource.status() === 'Loading')
{
  <div>Chargement...</div>
}
```

resource error et isLoading

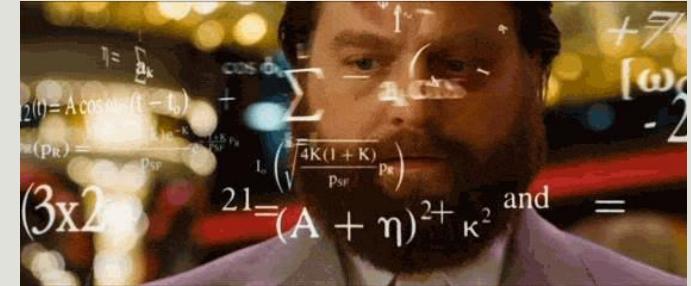
- **error** : Signal<undefined> est un signal réactif indiquant l'erreur si elle existe, undefined sinon
- **isLoading** : Signal<boolean> est un signal réactif indiquant si la resource est en cours de chargement

```
@if (cvResource.isLoading()) {  
    <div class="alert alert-info">  
        Loading ...  
    </div>  
}  
  
@if (cvResource.error()) {  
    <div class="alert alert-error">  
        Une erreur s'est produite  
    </div>  
}
```

resource reload et update

- **reload**: fonction qui permet de **relancer le loader**
- **update**: fonction qui **agit** comme le **update d'un signal** et qui **met à jour la value de la resource**
- **set**: fonction qui **agit** comme le **set d'un signal** et qui **met à jour la value de la resource**

Exercice



- Reprenez le composant Cv et faites en sorte qu'il utilise l'api resource
- Remarque avec fetch quand il y a une réponse 404 ou 500 elle ne déclenche pas une erreur. Vous devez donc la gérer manuellement.

```
loader: async ({params}) => {
  const response = await fetch(API.route);
  if (!response.ok) throw Error('Cv Not Found')
  return cv.json();
}
```

rxResource

- L'API rxResource est conçue pour gérer **des données asynchrones** basées sur **des Observables**.
- Elle offre une **gestion déclarative des états asynchrones** et **annule automatiquement les requêtes précédentes si les paramètres changent** (sémantique switchMap).
- Elle est très ressemblante à resource mais **la propriété loader** est défini via **la propriété stream**, qui prend **exactement les mêmes paramètres que loader**, car **un rxResource** est une ressource dite **de streaming émettrice** qui peut **émettre plusieurs valeurs au fil du temps**.

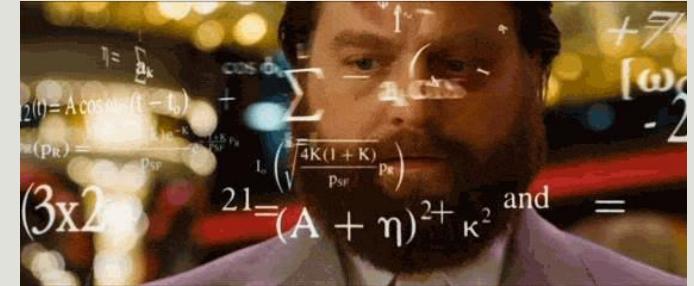
rxResource

```
cvResource = rxResource<Cv | null, {id:number}>({  
    // fonction qui détermine à quel moment on va déclencher le Loader  
    params: () => ({id: this.id()}),  
    // Loader va aller chercher le résultat du backend  
    stream: ({request, abortSignal, previous}) => {  
        return this.cvService.findById(request.id);  
    },  
    defaultValue: null  
});
```

- **params (request)** prend en paramètre une **fonction** ou un **paramètre** spécifiant les **signals** dont dépend votre **resource**.
- **stream** permet de spécifier l'Observable qui ira **chercher les données**
- **defaultValue** permet de spécifier la **valeur par défaut**

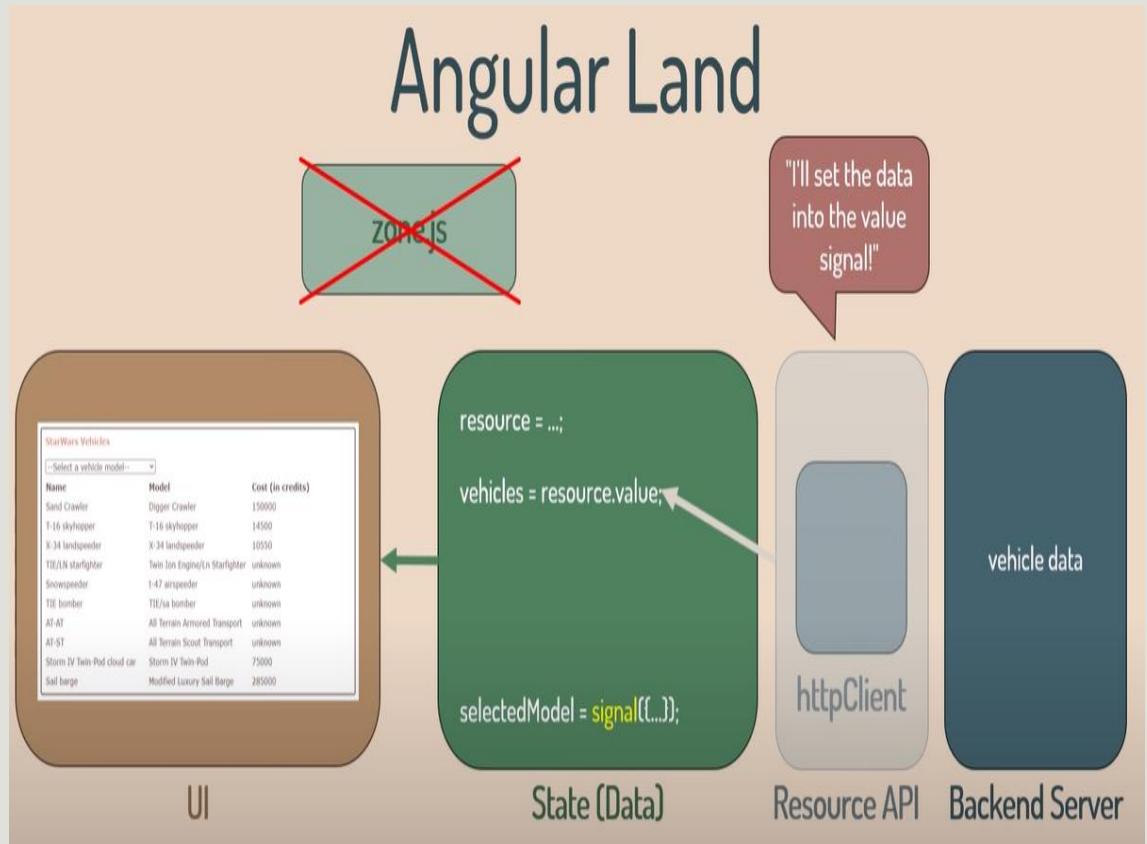
Exercice

- Reprenez le composant ProductComponent avec rxResource



Signal et les appels HTTP httpResource

- Une troisième api a été introduite dans **Angular 19.2**, c'est l'api **httpResource**.
- L'idée est de proposer un **wrapper sur le HttpClient** et qui soit plus simple et plus proche de la syntaxe du HttpClient que des apis resource et rxResource
- Comme pour les deux autres apis il retourne un objet de type **Resource, donc il expose tous ces propriétés comme value, error, status et isLoading.**
- Principalement destiné aux requêtes GET (non recommandé pour les mutations comme POST, PUT, DELETE).



Signal et les appels HTTP httpResource

- Dans **Angular 19.2** elle pouvait récupérer une **chaine, une fonction ou un objet de type HttpResourceRequest** permettant de décrire l'appel http à effectuer.
- A partir de la **version 20** on ne peut utiliser **qu'une fonction qui retourne une chaine ou un objet de type HttpResourceRequest**.
- L'introduction de httpResource rend la gestion des requêtes HTTP **plus déclarative et réactive**.
- **Au contraire du HttpClient, la httpResource déclenche son appel http dès sa création.** Ensuite elle **se recharge automatiquement** et effectue une nouvelle requête à **chaque changement d'un signal source** dont dépend la fonction.
- Dans cet exemple, à chaque fois que le **signal search change** un **nouvel appel est exécuté**

```
userHttpRessource = httpResource(() => API.users + `?name_like^${this.search()}`);
```

Signal et les appels HTTP httpResource

- Si vous avez besoin d'un **contrôle plus précis sur la requête**, vous pouvez transmettre une fonction qui renvoie un objet **HttpResourceRequest**.
- Outre la propriété **URL**, elle peut spécifier et réagir à des options telles que la **méthode**, les **paramètres** ou **les en-têtes**.

```
userHttpRessource = httpResource(() => ({  
    url: API.users,  
    params: new HttpParams().set('name_like', `^${this.search()}`),  
    method: 'GET'  
}));
```

```
interface HttpResourceRequest {  
    /**  
     * URL of the request.  
     * This URL should not include query parameters. Instead, specify query parameters through the  
     * `params` field.  
     */  
    url: string;  
    /**  
     * HTTP method of the request, which defaults to GET if not specified.  
     */  
    method?: string;  
    /**  
     * Body to send with the request, if there is one.  
     * If no Content-Type header is specified by the user, Angular will attempt to set one based on  
     * the type of `body`.  
     */  
    body?: unknown;  
    /**  
     * Dictionary of query parameters which will be appended to the request URL.  
     */  
    params?: HttpParams | Record<string, string | number | boolean | ReadonlyArray<string | number | boolean>>;  
    /**  
     * Dictionary of headers to include with the outgoing request.  
     */  
    headers?: HttpHeaders | Record<string, string | ReadonlyArray<string>>;  
    /**  
     * Context of the request stored in a dictionary of key-value pairs.  
     */  
    context?: HttpContext;  
    /**  
     * If `true`, progress events will be enabled for the request and delivered through the  
     * `HttpResource.progress` signal.  
     */  
    //...  
}
```

<https://angular.dev/api/common/http/HttpResourceRequest>

Signal et les appels HTTP

httpResource

Default value

- Par défaut la resource retourne undefined
- Si vous souhaitez définir une valeur par défaut, passez en dixième paramètre à votre fonction httpResource, un objet avec l'option defaultValue

```
userHttpRessource = httpResource<UserR[]>(  
  () => ({  
    url: API.users,  
    params: new HttpParams().set('name_like', `^${this.search()}`),  
    method: 'GET',  
  }),  
  {  
    defaultValue: []  
  }  
>;
```

Signal et les appels HTTP

httpResource

Retarder l'appel Http

- Comme mentionné au début, la **httpResource déclenche son appel http dès sa création.**
- **Si vous souhaitez retardé l'appel vous pouvez retourné undefined jusqu'au besoin de l'appel http.**

```
userHttpRessource = httpResource<UserR[]>(
  () => {
    const allow = this.allowHttp();
    return allow
      ?
      {
        url: API.users,
        params: new HttpParams().set('name_like', `^${this.search()}`),
        method: 'GET',
      }
      :
      undefined;
  }
);
```

Signal et les appels HTTP

httpResource

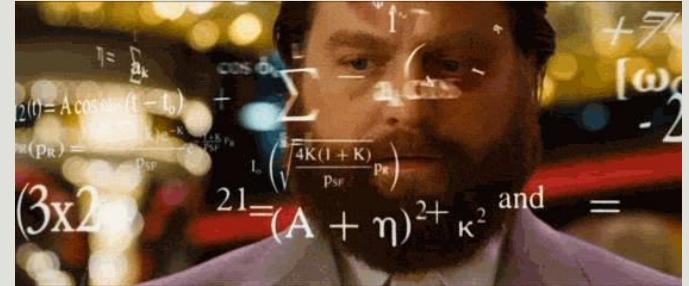
Debounce

-
- Si vous voulez appliquer un debounce ceci n'est pas natif avec httpResource.
 - **Le pattern proposé par la communauté est de déboucisé le signal déclencheur**

```
debouncedSearch = toSignal(  
    toObservable(this.search).pipe(debounceTime(500)  
, {initialValue: ''})  
userHttpRessource = httpResource<UserR[]>(  
() => {  
    return {  
        url: API.users,  
        params: new HttpParams().set('name_like', `^${this.debouncedSearch()}`),  
        method: 'GET',  
    }  
}  
);
```

Exercice

- Reprenez le composant CvComponent avec httpResource



Angular Reactive Form

AYMEN SELLAOUTI

Reactive form

- Les Reactive Form sont une deuxième méthode de gérer vos formulaires avec Angular.
- Au contraire des Template Driven Form, ses formulaires sont **générés programmatiquement** dans la partie TS.
- Ceci permet **d'alléger le template** des validateurs.
- D'autre part ca permet d'avoir un **formulaire testable**
- Finalement ceci permet de plus facilement **générer des Validateurs personnalisés.**

Reactive form

Créer un formulaire via FormGroup

- Commencer par ajouter le module **ReactiveFormsModule**.
- Afin de créer un formulaire avec l'approche réactive, vous devez créer un objet **FormGroup**.
- Un **FormGroup** prend en paramètre un objet décrivant le formulaire. Chaque champ de l'objet a comme **première propriété le nom et comme valeur un objet définissant les champs associés à ce formulaire**. Ce sont les **FormControl**.
- Chaque **FormControl** définit un **champ du formulaire**. Il prend en paramètre, **la valeur initiale**, **un Validator ou un tableau de Validator** et en troisième paramètre, **un AsyncValidator ou un tableau d'AsyncValidator**

```
FormGroup.constructor(  
  controls: {[p: string]: AbstractControl},  
  validatorOrOpts?: ValidatorFn | ValidatorFn[] | AbstractControlOptions | null,  
  asyncValidator?: AsyncValidatorFn | AsyncValidatorFn[] | null)
```

```
ngOnInit() {  
  this.form = new FormGroup({  
    name: new FormControl(null),  
    firstname: new FormControl(null),  
    age: new FormControl(null),  
  });  
}
```

Reactive form

Créer un formulaire

```
export declare interface AbstractControlOptions {  
  validators?: ValidatorFn | ValidatorFn[] | null;  
  asyncValidators?: AsyncValidatorFn | AsyncValidatorFn[] | null;  
  /**  
   * @description  
   * The event name for control to update upon.  
   */  
  updateOn?: 'change' | 'blur' | 'submit';  
}
```

```
form = new FormGroup({  
  name: new FormControl(null),  
  firstname: new FormControl("Aymen"),  
  age: new FormControl(0, {  
    asyncValidators: [],  
    validators: Validators.required,  
    updateOn: "blur",  
  }),  
});
```

Reactive form

Créer un formulaire via le service FormBuilder

- Vous pouvez aussi passer par le **service FormBuilder** et sa méthode **group**.
- Elle prend en **paramètre un objet** avec comme **clé le nom** du formControl et comme **valeur un tableau** avec comme **première propriété la valeur initiale du champ**.
- Ceci est **moins verbeux** et plus simple à gérer pour les grands formulaires.

```
this.form = this.formBuilder.group({  
  email: [null],  
  username: [null],  
  userCategory: ['employee'],  
});
```

```
constructor(  
  private formBuilder: FormBuilder  
) {}
```

Reactive form

Associer le FormGroup à votre form

- Une fois le formulaire défini, nous devons **l'associer au form de votre template**.
- Pour ce faire, vous devez ajouter la directive **formGroup** (qui se trouve dans le module **ReactiveFormsModuleModule**) au niveau de la balise form et la binder à votre objet de type **FormGroup** au niveau de votre fichier TS.



```
reactive.component.ts x reactive.component.html x
1
2
3
4
5
6
7
8
9
10
11
12

reactive.component.ts x reactive.component.html x
1 <h3 class="text-center">User Details</h3>
2 <hr>
3 <form [formGroup]="form">
4   <div>
5     <label for="email">Email</label>
6     <input class="form-control" type="email" name="email">
7   </div>
8 </form>
9 <hr>
10 <p>{{ user | json }}</p>
```

The screenshot shows a code editor with two tabs: 'reactive.component.ts' and 'reactive.component.html'. In the 'reactive.component.ts' tab, line 12 contains the code 'form: FormGroup;'. This line is highlighted with a red box. In the 'reactive.component.html' tab, line 3 contains the code '<form [formGroup]="form">'. This line is also highlighted with a red box.

Reactive form

Associer les FormControl à vos input

- Afin d'associer votre FormControl à votre champ input, utiliser la directive **formControlName** et associer le à **l'identifiant** du **FormControl**

The screenshot shows a code editor with two panes. The left pane displays TypeScript code for a `FormGroup` named `this.form`, which contains four `FormControl`s: `email`, `username`, `age`, and `password`. The `email` control is highlighted with a red box. The right pane shows the corresponding HTML template. It includes a `<div>` element containing a `<label>` for the `email` field and an `<input>` field. The `<input>` field has several attributes: `class="form-control" type="email" id="email"`, `formControlName="email"` (which is also highlighted with a red box), and `placeholder="Enter email"`. The `<input>` field is also highlighted with a red box.

```
this.form = new FormGroup( controls: {
  email: new FormControl( formState: null ),
  username: new FormControl( formState: null ),
  age: new FormControl( formState: null ),
  password: new FormControl( formState: null )
})
```

```
4   <div>
5     <label for="email">Email</label>
6     <input
7       class="form-control" type="email" id="email"
8       formControlName="email" placeholder="Enter email"/>
9   </div>
10  <div>
11    <label for="username">Username</label>
12    <input
13      class="form-control" type="text" id="username"
14      formControlName="username" placeholder="Username"/>
15  </div>
```

Reactive form

Récupérer les FormControl dans le HTML

- Afin de récupérer les FormControl dans le HTML, utiliser la méthode get de votre form group et passer lui l'identifiant du FormControl à récupérer.

```
<button  
    class="btn btn-primary"  
    [disabled]="form.get('password').valid"  
    (click)="process()">  
    Submit  
</button>
```

Reactive form

Récupérer les erreurs du FormControl dans le HTML

- Afin de récupérer les erreurs de votre control dans le HTML, utiliser l'attribut errors.

```
<div  
  *ngIf="form.get('name')?.errors && form.get('name')?.touched"  
  class="alert alert-danger"  
>
```

Reactive form

Récupérer les FormControl dans le HTML

- Une deuxième méthode pour avoir un code moins verbeux est de créer des getters pour vos champs.

```
get name(): AbstractControl {  
    return this.form.get("name");  
}
```

```
<div *ngIf="name.errors && name.touched">  
    Ce champ est obligatoire  
</div>
```

Reactive form

Soumettre le formulaire

- A l'inverse de l'approche ‘template driven’, vous n'avez pas besoin de récupérer la référence de l'objet form puisque vous l'avez déjà créée dans votre ts.
- Il suffit donc d'écouter le submit avec ngSubmit ou le clic sur un bouton pour déclencher la méthode du composant qui gérera l'envoi du formulaire.

The screenshot shows a code editor with two parts. On the left, there is a component's TypeScript code:process() {
 console.log(this.form);
}

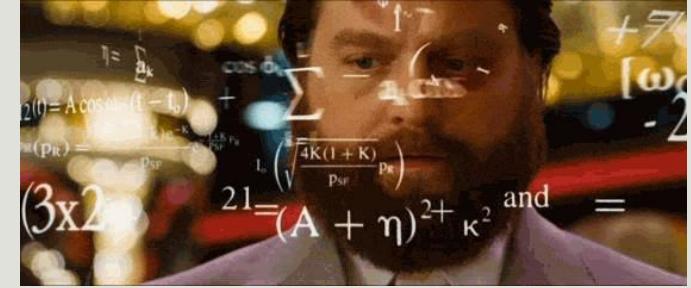
```
On the right, the corresponding HTML template is shown:
```

34 <button
35 class="btn btn-primary"
36 (click)="process()"
37 Submit
38 </button>

```
This illustrates how the component's logic (the process method) is triggered by the click event on the button.
```

Exercice

- Reprenez le formulaire de login et gérer le avec les reactives form. Ne prenez pas encore en considération les validateurs.



Reactive form

Les Validateurs

- Un **validateur** est une **fonction** qui retourne **false si un champ est valide** selon une certaine condition **sinon elle retourne une information sur l'erreur.**
- Afin de valider votre formulaire, passer en deuxième paramètre de **FormControl** une **référence à une méthode de la classe Validators** ou un **tableau de ces méthodes.**

```
FormControl(formState: null, validatorOrOpts: [Validators.1],  
  ↪email(control: AbstractControl)  
  ↪required(control: AbstractControl)  
  ▲compose(validators: null)  
  ↪nullValidator(control: AbstractControl)  
  ↪max(max: number)  
  ↪composeAsync(validators: (AsyncValidatorFn | null)[])  
  ↪maxLength(maxLength: number)  
  ↪min(min: number)  
  ↪minLength(minLength: number)  
  ↪pattern(pattern: string | RegExp)
```

Reactive form

Les Validateurs

-
- **Validateurs synchrones** : Ce sont des fonctions qui contrôle votre élément et retournent un **ensemble d'erreurs de validation ou null**. C'est ce qu'on passe en deuxième paramètre lors de l'instanciation d'un FormControl.
 - **Validateurs asynchrones** : Ce sont des fonctions asynchrones qui contrôle votre élément et retournent une **Promise ou un Observable** qui émettent un **ensemble d'erreurs de validation ou null**. C'est ce qu'on passe en troisième paramètre lors de l'instanciation d'un FormControl.
 - Pour des raisons de **performance**, Angular commence par les validateurs synchrones, s'ils passent il déclenche les validateurs asynchrones.

Reactive form

Les Validateurs offerts par Angular

- Afin d'utilisez vos validateurs, ajoutez les dans les tableaux de validateurs associés à vos champs définis lors de la création de votre formulaire.

```
new FormGroup({  
    email: new FormControl(null, [Validators.required, Validators.email]),  
    username: new FormControl(null, [Validators.minLength(3)]),  
    age: new FormControl(null, [  
        Validators.required, Validators.pattern('[0-1]?\\d{1,2}')  
    ]),  
});
```

Méthode 1

Reactive form

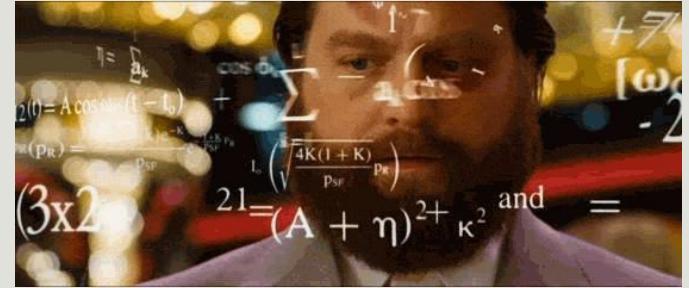
Les Validateurs offerts par Angular

- Vous pouvez utiliser des validateurs offerts par angular ou créer vos propres validateurs.
- Les validateurs de base sont les mêmes que ceux de l'approche basée Template.

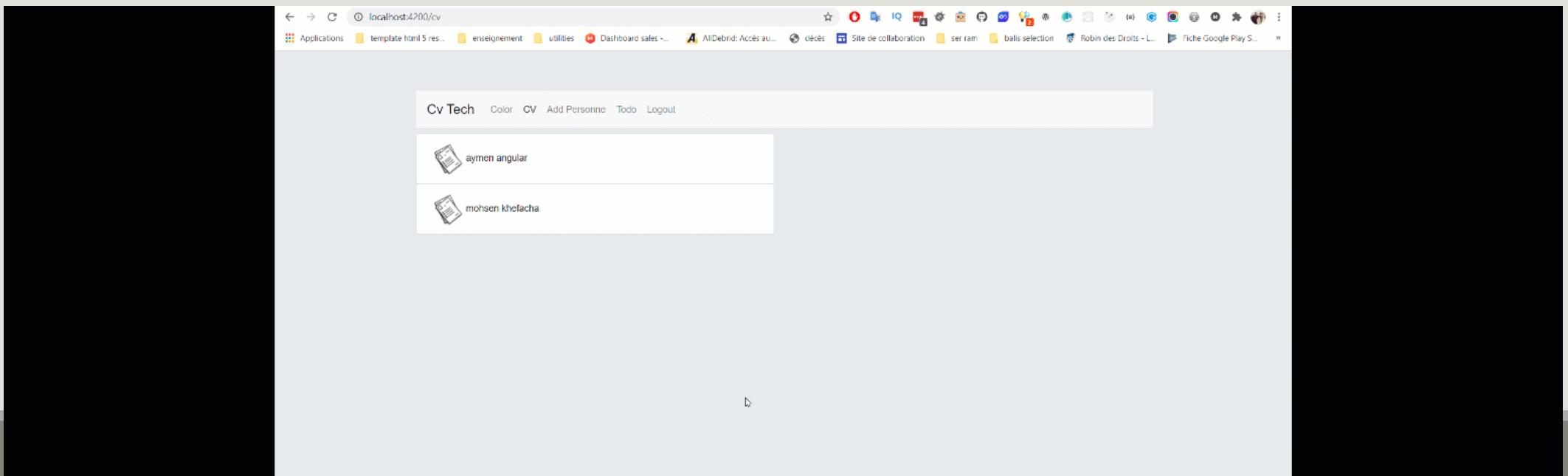
```
formGroup: FormGroup = new FormGroup({  
    name: new FormControl(null, {  
        validators: [Validators.required, Validators.minLength(3)],  
        asyncValidators: [],  
        updateOn: "change"  
    }),  
    age: new FormControl(null),  
});
```

Méthode 2

Exercice



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



Reactive form

Manipulez les valeurs de votre form

- Afin de **mettre à jour la valeur** d'un **FormControl** ou **d'un FormGroup**, vous pouvez utiliser la méthode **setValue()** qui met à jour la valeur du contrôle de formulaire et valide la structure de la valeur fournie par rapport à la structure du contrôle.
- Vous pouvez utiliser la méthode **patchValue** si vous modifiez uniquement **une partie de votre formulaire**.

```
this.form.setValue({ name: "Sellaouti", firstname: "Aymen" });
```

```
this.form.patchValue({firstname: "Aymen" });
```

Reactive form

Suivre les modifications de vos FormControls et de vos FormGroup

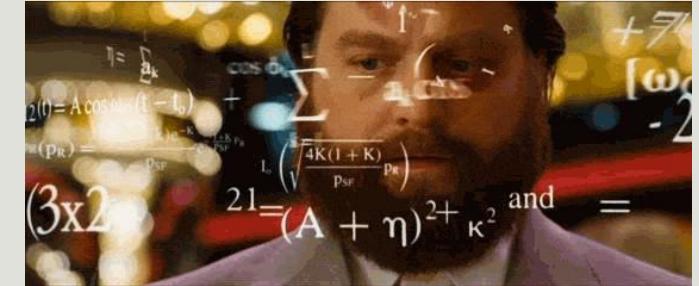
- **FormControl et FormGroup**, vous fournissent deux **Observables** permettant le **suivi des changements de valeur et de statuts**.
- **ValueChanges** est un événement déclenché par les formulaires Angular **chaque fois que la valeur de FormControl, FormGroup** change. L'observable obtient la dernière valeur du contrôle. Il nous **permet de suivre les modifications apportées à la valeur en temps réel et d'y répondre**. Par exemple, nous pouvons l'utiliser pour valider la valeur, calculer les champs calculés, ...

Reactive form

Suivre les modifications de vos FormControl et de vos FormGroup

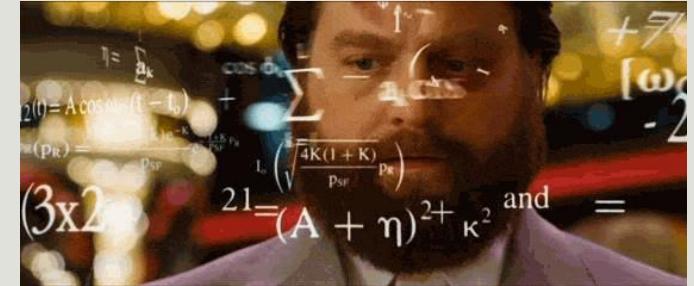
- **statusChanges** est un événement déclenché par les formulaires Angular **chaque fois que Angular calcule le statut de validation** de FormControl, FormGroup. Il renvoie un observable afin que vous puissiez vous y abonner. L'observable obtient le **dernier état du contrôle**.

Exercice



- Afin de protéger les informations personnelles des mineurs, faites en sorte que lorsque l'age de la personne possédant le Cv est inférieur à 18 ans, il ne puisse pas renseigner le path de l'image.

Exercice



- Etant donné que notre formulaire est assez volumineux et pour permettre une meilleure expérience utilisateur, nous voulons faire en sorte que si l'utilisateur saisisse un formulaire valide et qu'il sorte de la page sans l'envoyer, il puisse le retrouver rempli lorsqu'il retourne à la page d'ajout d'un cv.

Reactive form

Customiser vos validateurs

- Un custom validator est une fonction de type **ValidatorFn** qui prend en paramètre un control de type **AbstractControl** (qui contient une propriété **value** représentant la **valeur du champ à valider**) et **retourne null si la valeur est valide** ou un objet de type **ValidationErrors** s'il y a des **erreurs de validation**.

```
export declare interface ValidatorFn {  
  (control: AbstractControl): ValidationErrors | null;  
}
```

```
return !passwordValid ? {passwordStrength: true} : null;
```

Reactive form

Customiser vos validateurs

```
export function createPasswordStrengthValidator(): ValidatorFn {
  return (control: AbstractControl): ValidationErrors | null => {
    const value = control.value;
    if (!value) {
      return null;
    }
    const hasUpperCase = /[A-Z]+/.test(value);
    const hasLowerCase = /[a-z]+/.test(value);
    const hasNumeric = /[0-9]+/.test(value);
    const passwordValid = hasUpperCase && hasLowerCase && hasNumeric;
    return !passwordValid ? {passwordStrength: true} : null;
  };
}
```

Reactive form

Customiser vos validateurs

Validateurs Asynchrones

- Dans certains cas d'utilisation, la validation de votre champ ne se fait pas d'une façon **asynchrone**.
- Le cas le plus répandu est la **validation par votre backend**.
- Imaginez que vous avez un champ email et qu'il ne doit pas être redondant. La solution est d'avoir une API qui vérifie ça et qui vous retourne la réponse.
- Ce traitement étant Asynchrone, le Validateur synchrone ne fait plus affaire.
- Il faut donc créer un validateur **ASYNCHRONE**.

Reactive form

Customiser vos validateurs

Validateurs Asynchrones

- Le Validator, renvoyée par la fonction de création de validateur, doit suivre ces règles :
- Un seul argument d'entrée est attendu, qui est de type **AbstractControl**. La fonction validateur peut obtenir la valeur à valider via la propriété **control.value**
- La fonction de validation doit renvoyer une **Promise<null>**, un **Observable<null>** si aucune erreur n'a été trouvée dans la valeur du champ, ce qui signifie que la **valeur est valide**
- Si des erreurs de validation sont trouvées, la fonction doit renvoyer une **Promise ou un Observable** d'un objet de type **ValidationErrors**.

Reactive form

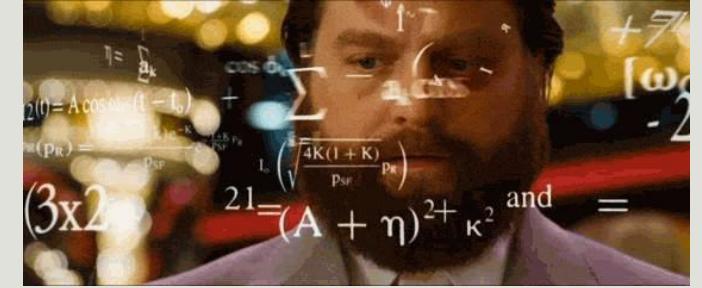
Customiser vos validateurs

Validateurs Asynchrones

```
export function userExistsValidator(authService: AuthService): AsyncValidatorFn {
  return (control: AbstractControl) => {
    return authService.findUserByEmail(control.value);
  };
}
```

Exercice

- Le champ Cin étant unique, créer un validateur asynchrone permettant de gérer cette contrainte et tester le.



Reactive form

Customiser vos validateurs

Valider un form

- Afin de valider un form, vous devez faire la même chose que pour le validateur d'un FormControl.

```
export function createDateRangeValidator(): ValidatorFn {
  return (form: AbstractControl): ValidationErrors | null => {
    const start: Date = form.get('startAt').value;
    const end: Date = form.get('endAt').value;
    if (start && end) {
      const isRangeValid = (end.getTime() - start.getTime() > 0);
      return isRangeValid ? null : {dateRange: true};
    }
    return null;
  };
}
```

Reactive form

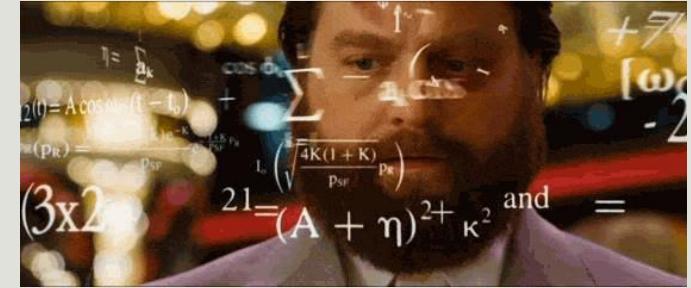
Customiser vos validateurs

Valider un form

- Maintenant et pour l'appliquer à votre FormGroup ajouter à la création de votre FormGroup un second paramètre qui est un objet options et utiliser la propriété validators

```
this.form = new FormGroup(  
  {},  
  {  
    validators: VotreValidateur  
  }  
);  
this.form = this.formBuilder.group(  
  {},  
  {  
    validators: VotreValidateur  
  }  
);
```

Exercice



- Le champ Cin étant composé de 8 caractères numériques, il présente une corrélation entre l'age de la personne et les deux premiers caractères.
- Si la personne a un age ≥ 60 ans, les deux première caractères numériques doivent être entre 00 et 19.
- Sinon ca doit être supérieur à 19.
- Créer le validateur permettant de faire ca.

Récupérer les paramètres d'une route

ActivatedRoute

- Afin de récupérer les paramètres d'une route au niveau d'un composant, Angular nous fournit un **service** qui gère la **route active**, c'est le **ActivatedRoute**
- L'objet **ActivatedRoute** dans Angular est un objet qui contient des **informations sur la route actuellement activée**.
- Il est généralement utilisé pour **accéder aux informations de route** telles que **les paramètres de route, les données de route et les paramètres de requête**.
- Il est également utilisé **pour souscrire aux changements de route**.

Récupérer les paramètres d'une route ActivatedRoute

- **params**: Retourne un observable des paramètres de route actuels.
- **queryParams**: Retourne un observable des paramètres de requête actuels.
- **data**: Retourne un observable des données de route associées à la route actuelle.
- **snapshot**: Retourne un instantané de la route actuelle.
- **url**: Retourne un observable de l'URL de la route actuelle.
- **parent**: Retourne l'instance de ActivatedRoute de la route parente.
- **firstChild**: Retourne l'instance de ActivatedRoute du premier enfant de la route actuelle.

Récupérer les paramètres d'une route ActivatedRoute

- **children**: Retourne un tableau d'instances de ActivatedRoute des enfants de la route actuelle.
- **paramMap**: Retourne un observable qui contient les paramètres de route sous forme de map.
- **queryParamMap**: Retourne un observable qui contient les paramètres de requête sous forme de map.

Récupérer les paramètres d'une route ActivatedRoute

```
activatedRoute          details-cv.component.ts:33
                      details-cv.component.ts:34
  ActivatedRoute {url: BehaviorSubject, params: BehaviorSubject, que
  ▼ ryParams: BehaviorSubject, fragment: BehaviorSubject, data: Behavi
    orSubject, ...} ⓘ
      ► component: class DetailsCvComponent
      ► data: BehaviorSubject {closed: false, currentObservers: Array(0),
      ► fragment: BehaviorSubject {closed: false, currentObservers: null,
        outlet: "primary"
      ► params: BehaviorSubject {closed: false, currentObservers: null, c
      ► queryParams: BehaviorSubject {closed: false, currentObservers: nu
      ► snapshot: ActivatedRouteSnapshot {url: Array(1), params: {...}, que
      ► title: AnonymousSubject {closed: false, currentObservers: null, c
      ► url: BehaviorSubject {closed: false, currentObservers: null, obse
      ► _futureSnapshot: ActivatedRouteSnapshot {url: Array(1), params: {
      ► _routerState: RouterState {_root: TreeNode, snapshot: RouterState
        children: (...),
        firstChild: (...),
        paramMap: (...),
        parent: (...),
        pathFromRoot: (...),
        queryParamMap: (...),
        root: (...),
        routeConfig: (...),
        [[Prototype]]: Object
      
```

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.
 - **outlet**: Retourne le nom de l'outlet de route actuel.
 - **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.paramMap.get('id')
```

Route Fils

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

Route Fils

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

Route Fils

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

Route fils / définition dans un parent

- Supposons que nous voulons avoir un Template central avec des données fixe et des parties variables dans le même template.
- En changeant les routes, le contenu principal doit rester le même et la partie variable doit changer selon la route.

Route Fils

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

Route Fils

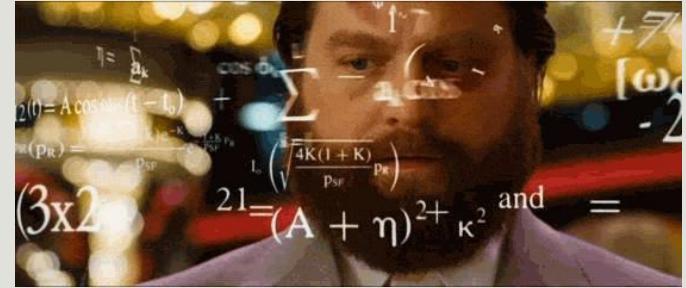
```
const CV_ROUTE: Routes = [
  {
    path: 'cv',
    component: CvComponent,
    children: [
      {path: 'detail/:id', component: DetailCvComponent },
      {path: 'addPersonne', component: FormPersonneComponent },
    ]
  }
];
```

Master Detail Implementation

- Le pattern "**master-detail**" dans Angular est un modèle d'architecture utilisé pour afficher des données dans une interface utilisateur.
- Il consiste à avoir **une vue "maître"** qui affiche une **liste d'éléments**, et une vue "**détail**" qui affiche les **détails d'un élément** sélectionné dans la vue "maître".
- Cela permet aux utilisateurs de naviguer facilement entre les différents éléments et de voir les détails correspondants sans avoir à charger une nouvelle page.
- Afin de l'implémenter, il faut utiliser les routes fils.

Exercice

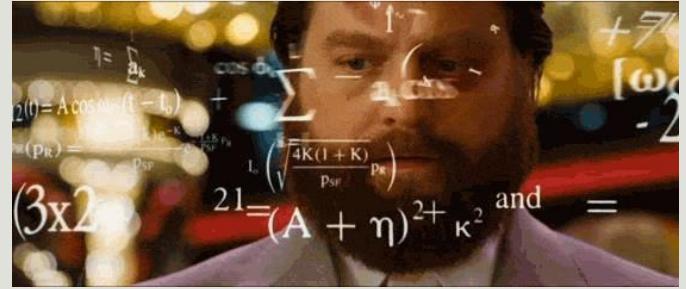
Master Detail Implementation



- Nous voulons implémenter le pattern Master Details pour notre liste de cvs.
- Créer un nouveau composant MasterDetailsCv.
- Il devra permettre l'affichage de la liste des cvs. Au click, un détail du cv sélectionné devra apparaître.

Exercice

Master Detail Implementation



A cvStartingAdvProject X +

localhost:4200/cv/list

todo goodies gk Barre de favoris TCPC2022 Rem4U Ruteur - Accueil enseignement utilities enseignement balis selection rxjs ChatGPT

cvStartingAdvProject Home Cv List Add Cv Todo Word Color Logout

container works!

resolution-modifiers works!

master-detail-cv works!

aymen sellaouti

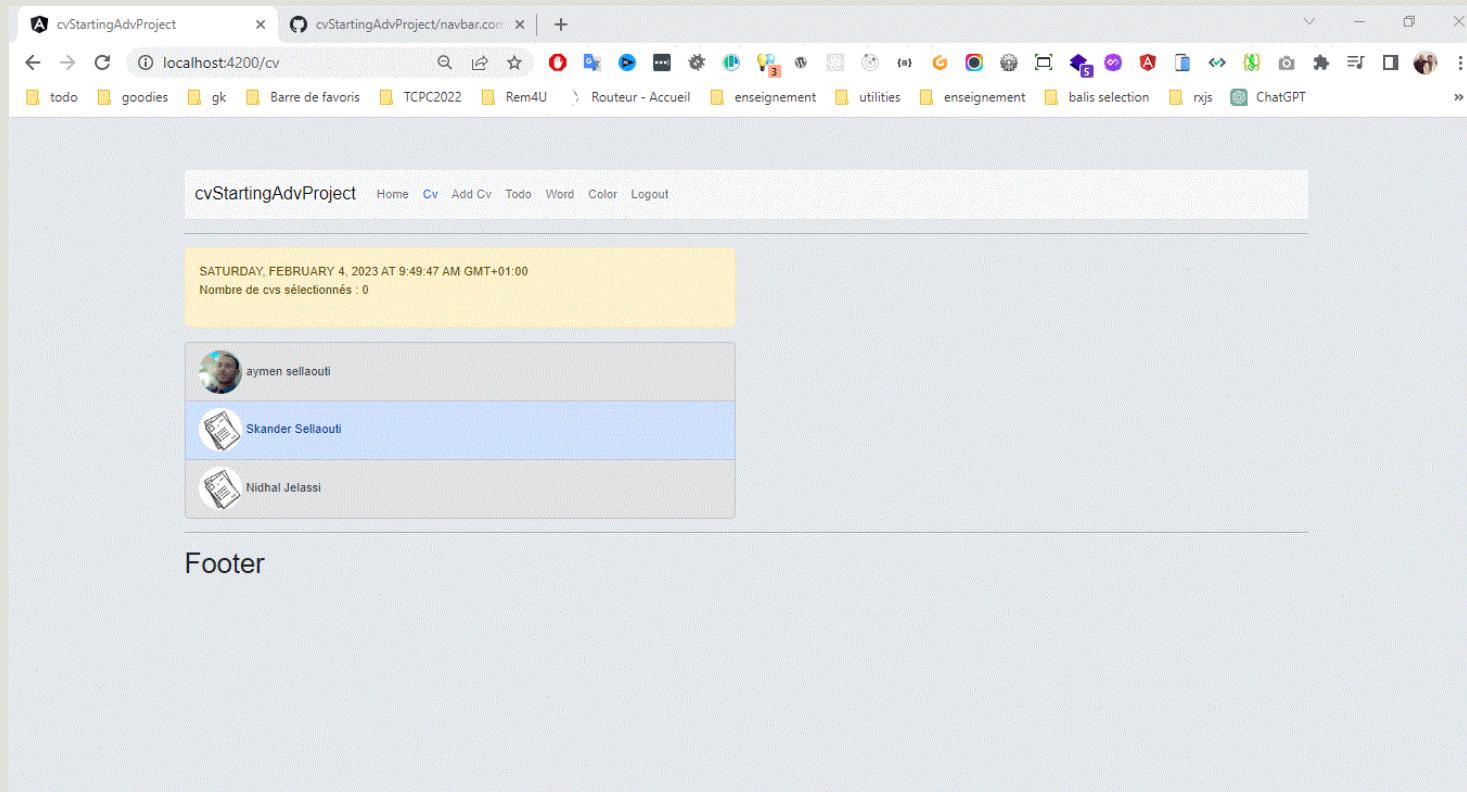
Skander Sellaouti

Nidhal Jelassi

Footer

Router Resolver

- Analysons la page DetailsCv



Router Resolver

- Le **Router Resolver** d'Angular est un mécanisme qui permet de **résoudre des données avant qu'une route ne soit chargée**.
- Il permet de **charger les données nécessaires** pour afficher une vue spécifique en **utilisant un service qui est lié à la route**.
- Il est utilisé lorsque vous avez besoin de **charger des données avant d'afficher une vue**, comme pour afficher des données d'un utilisateur avant d'afficher sa page de profil.
- Il est également utilisé pour **éviter les erreurs de chargement de vue** en **garantissant** que les données nécessaires sont **chargées avant de naviguer vers une route**.

Router Resolver

- Le **Router Resolver** est soit une fonction (à partir d'Angular 15) soit une classe qui implémente l'interface **Resolve** et qui est **générique**. Vous devez donc spécifier ce que le Resolver va fournir comme données.
- Vous devez implémenter la **fonction Resolve** qui devra **retourner** un **objet de T ou une Promise<T> ou un Observable<T>**.
- Elle prend en paramètre un objet de type **ActivatedRouteSnapshot** qui vous permet de **récupérer les paramètre de votre route** à travers le paramètre **paramMap** et sa méthode **get**.

Router Resolver

```
import { ResolveFn } from '@angular/router';

export const firstResolver: ResolveFn<boolean> = (route, state) => {
  return true;
};
```

Router Resolver

```
@Injectable({
  providedIn: 'root',
})
export class CvResolver implements Resolve<Cv> {
  resolve(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<Cv> | Cv {
    const cvId = route.paramMap.get('id');
  }
}
```

Router Resolver

- Maintenant, afin de passez **le résultat de votre resolver à votre route**, ajoutez une propriété **resolve** à votre **route** dans votre **fichier de routing** et passez lui un **object** avec comme **clé** le **nom** que vous voulez donner à votre propriété et comme **valeur** le **resolver**.

```
{  
  path: ':id',  
  component: DetailsCvComponent,  
  resolve: {  
    cv: CvResolver  
  }  
},
```

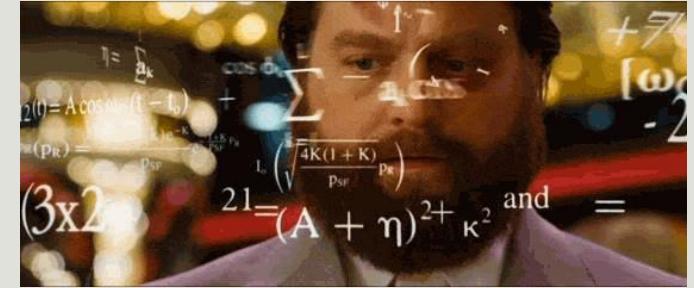
Router Resolver

- Maintenant, dans votre composant, injecter le service **ActivatedRoute**.
- Pour accéder à la valeur de retour du resolver (si c'est une Promise ou un Observable, il attendra la valeur émise), accéder à la propriété **snapshot** qui contient une **propriété data** qui **contiendra le champ**.
- Si vous voulez un accès **dynamique**, utilisez **l'Observable data de l'ActivatedRoute**.

```
this.activatedRoute.snapshot.data['cv'];
```

Exercice

- Appliquez le resolver pour le composant MasterDetailsComponent



Route Provider

A partir d'Angular 14

- A partir d'Angular 14, la clé provider a été introduite dans l'objet route.
- Ceci permettra de provider des provider pour la route et ses enfants

```
{  
  path: 'routerProvider',  
  component: RouterPoviderComponent,  
  providers: [LoggerService]  
},
```

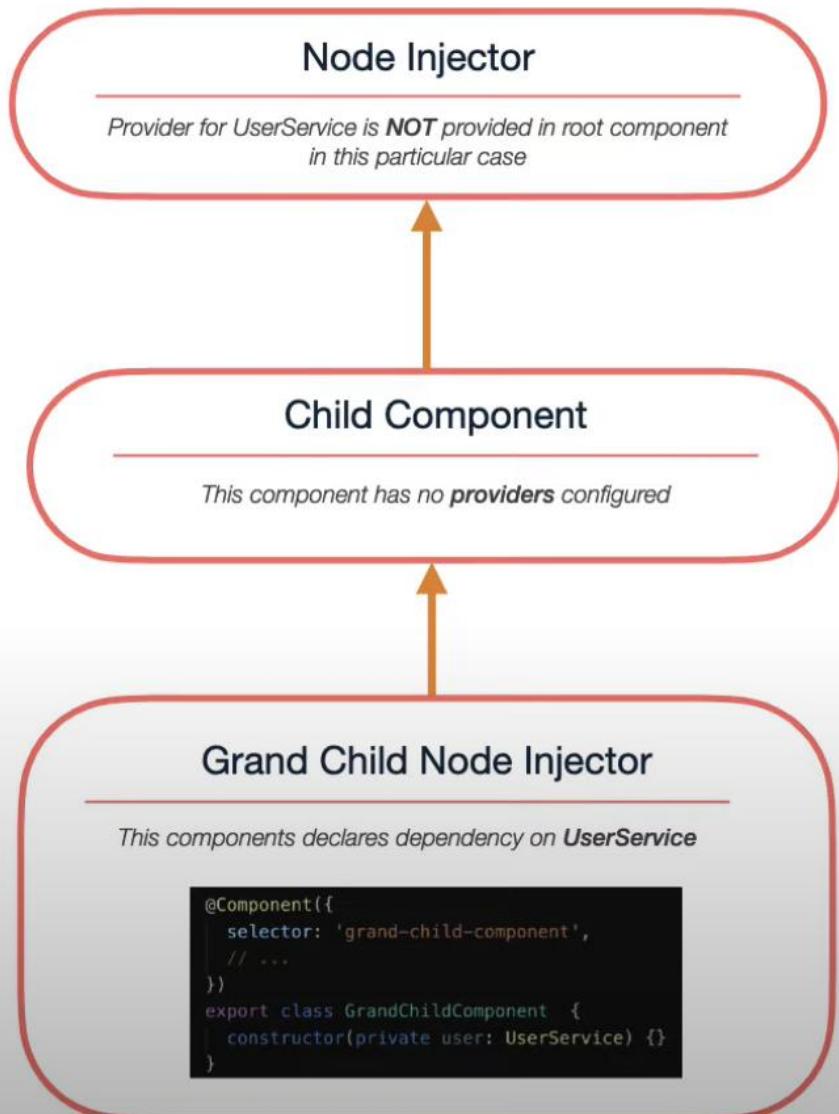
Route Provider

A partir d'Angular 14

- Ceci va créer un nouvel *Injector* qui sera appelé juste après *l'element Injector*

```
{  
  path: 'routerProvider',  
  component: RouterPoviderComponent,  
  providers: [ LoggerService ]  
},
```

Node Injector Hierarchy



Error will be thrown

Environment Injector Hierarchy

Angular could not resolve it in Element Injector tree, so it goes back to the component when started...

NullInjector

It throws error if Angular tries to find service here

Platform Injector

Created when we call method `platformBrowserDynamic()`

Root Injector

Services which were configured in non-lazy `@NgModule` and in `@Injectable()` annotations

Router Injector (/admin)

This injector is created for every route that has providers

Lazy Loading

- Par défaut, tout les modules que vous déclarer au niveau du AppModule sont chargé au lancement de l'application.
- Ceci pose un problème au niveau du Bundle généré de votre application.
- Une grande application aura une taille assez conséquente ce qui peut provoquer un problème au chargement de l'application et donc un problème d'expérience utilisateur.
- L'idée du lazyLoading et de **charge au départ le module principale** et puis de **ne charger un module que si on appelle l'une de ses routes**.
- Ceci va nous faire gagner en performance.

The screenshot shows a browser window with the address bar at `localhost:4200`. The main content area displays the text "front works!". Above the content, there is a file upload interface with a placeholder "Choisir un fichier" and a button "Upload". The browser's developer tools are open, specifically the Network tab. The Network tab shows a list of requests and a waterfall chart. The waterfall chart has time markers at 2000 ms, 4000 ms, 6000 ms, 8000 ms, and 10000 ms. Below the chart is a table of network requests:

Name	Status	Type	Initiator	Size	Time	Waterfall
localhost	304	docu...	Other	210 B	298 ...	
runtime.js	200	script	(index)	211 B	13 ms	
polyfills.js	200	script	(index)	212 B	13 ms	
styles.js	200	script	(index)	212 B	180 ...	
vendor.js	200	script	(index)	213 B	302 ...	
main.js	200	script	(index)	212 B	26 ms	
personnes	(failed)	xhr	VM3716:1	0 B	2.02 s	

Lazy Loading

➤ Afin d'implémenter l'ancienne façon d'implémenter le *Lazy Loading* des modules, on doit suivre les étapes suivantes :

1. Le **module** à charger doit **lui-même gérer sa partie routing**
2. Au niveau du routing principal (AppRoutingModule), créer une **route**, ajouter lui un path 'ça sera le **préfixe** de toutes les routes du module', et ajouter une nouvelle clé qui est **loadChildren**. Cette clé va informer angular et lui demander de ne charger le module associé que lorsque on appelle le path défini.
3. Ce **paramètre** prend ou une **chaine de caractère** qui spécifie le module à charger ou une callback function.
4. Finalement **enlever les imports des modules lazy loaded** au niveau du AppModule

```
{  
  path: "cv",  
  loadChildren: "./cv/cv.module#CvModule",  
},
```

```
{  
  path: "cv",  
  loadChildren: () => import('./cv/cv.module').then(  
    m => m.CvModule  
)  
,
```

Lazyload Standalone Component

- Maintenant qu'un composant est devenu indépendant de tout module vous pouvez le charger d'une façon fainéante.
- Vous pouvez faire du **lazy loading** pour les **standalone component** avec la clé **loadComponent**.

```
{  
  path: 'track',  
  loadComponent: () => import('./track-by/track-by.component')  
    .then(c => c.TrackByComponent)  
}
```

Lazyload groupement de Standalone Component

- Vous pouvez faire du **lazy loading** pour une configuration de routing (groupement de route de **standalone component**) avec la clé **loadChildren**.

```
{  
  path: 'cv',  
  loadChildren: () => import('./cv/cv.routing').then((m) => m.CV_ROUTES),  
},
```

Lazyloading et default exports

- Lors de l'utilisation de loadChildren et loadComponent, le routeur comprend et décomprime automatiquement les appels d'importation dynamiques import() avec les exportations par défaut.
- Vous pouvez en profiter pour ignorer le .then() pour de telles opérations de chargement différé.

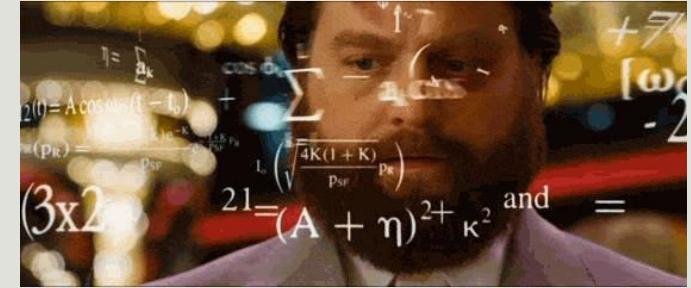
```
export default [  
  { path: '', component: CvComponent},  
]
```

```
{  
  path: 'cv',  
  loadChildren: () => import('./cv/cv.routing')  
},
```

Exercice

LazyLoading

- Faites du lazyLoading pour le composant MiniWordComponent



Preloading Lazy Loading

- Le **problème** qu'on peut identifier avec le **lazy loading** est le fait qu'en passant d'un module à un autre on aura **toujours un chargement des nouveaux modules.**
- Si vos **modules** sont **très volumineux** ou que la connexion du client est mauvaise, il y aura **plusieurs latences**. Ceci va provoquer un problème avec l'utilisateur.
- La question qui se pose est : **Y a-t-il un moyen de personnaliser les stratégies de chargement ???**

Preloading Lazy Loading

- Dans les applications modulaires, la méthode `forRoot` de votre `RouterModule` prend en second paramètre un objet vous permettant de configurer la stratégie de chargement avec la propriété `preloadingStrategy`.
- Cette propriété prend en paramètre, par défaut `NoPreloading`.
- La deuxième valeur qu'elle peut prendre est `PreloadAllModules`. Elle demande à Angular de précharger tous les `lazyLoaded Modules` une fois le Module principal chargé.

```
import { PreloadAllModules } from "@angular/router";
@NgModule({
  imports: [RouterModule.forRoot(routes, {
    preloadingStrategy: PreloadAllModules
})],
  exports: [RouterModule],
})
```

vendor.js	200	script	(index)	213 B	270 ms	
main.js	200	script	(index)	212 B	25 ms	
personnes	(failed)	xhr	VM14822:1	0 B	2.01 s	
personnes	(failed)		Other	0 B	2.00 s	
common.js	200	script	bootstrap:149	210 B	9 ms	
cv-cv-module.js	200	script	bootstrap:149	211 B	9 ms	
todo-todo-module.js	200	script	bootstrap:149	211 B	8 ms	

Preloading Lazy Loading

- Dans les applications standalone, passez la fonction withPreloading comme paramètre de votre fonction provideRouter.
- Passez lui le provider PreloadAllModules.

```
provideRouter(  
    routes,  
    withPreloading(LaStrategyDePreload)  
) ,
```

Preloading Lazy Loading

Créer votre propre stratégie

- Avec **PreloadAllModules**, tous les modules sont **préchargés**, ce qui peut en fait créer un **goulot d'étranglement** si l'application a un **grand nombre de modules à charger**.
- Une meilleure stratégie serait de **charger sélectivement les modules requis au démarrage**. Par exemple, module d'authentification, module principal, module partagé, etc.
- Pour **précharger sélectivement** un module, nous devons utiliser une **stratégie de préchargement personnalisée**.
- Créez d'abord une **classe** qui implémente l'interface **PreloadingStrategy**. La classe doit implémenter la méthode **preload()**.
- C'est cette méthode qui détermine s'il **faut précharger le module ou non**.

Preloading Lazy Loading

Créer votre propre stratégie

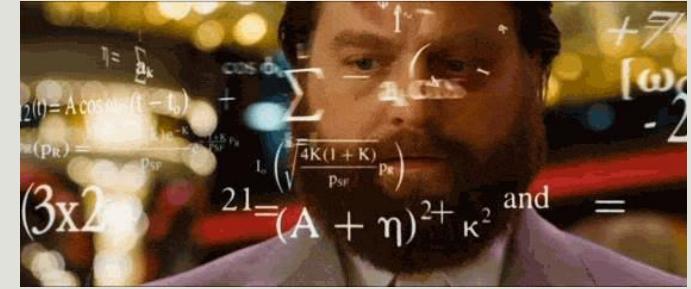
- La signature de la fonction **preload** prend en paramètre un **objet Route** représentant la route ciblée et en deuxième paramètre une fonction **load** qui retourne un Observable.
- Si vous retournez la méthode load, le module sera **préchargé**.
- Si vous **ne voulez pas le précharger** retourner un **Observable de null**.

```
@Injectable({providedIn: 'root'})
export class CustomPreloadingStrategy implements PreloadingStrategy {
  preload(route: Route, load: () => Observable<any>): Observable<any> {
    if (route.data["preload"]) {
      return load();
    }
    else {
      of(null);
    }
  }
}
```

Exercice

Preloading Stratégy

- Appliquez la Custom Preloading Strategy



Lazy Loading Migration

- Depuis Angular 18.2, il existe une migration permettant de transformer vos routes en lazy-loading components.

`ng generate @angular/core:route-lazy-loading`

- Vous pouvez aussi l'appliquer sur une partie de vos composant en passant le paramètre path

`ng generate @angular/core:route-lazy-loading --path src/app/sub-component`

Optimisation @defer

- Dans cette partie nous allons voir les éléments suivants :
- Dans quels cas d'utilisation nous aurons besoin de defer
- Comment utiliser @defer
- Quels sont les blocks offerts avec @defer : @placeholder, @loading, @error
- Qu'est ce qu'un trigger
- Quels sont les triggers offerts par Angular
- Comment créer son propre trigger

Optimisation

@defer

C'est quoi et à quoi ça sert ?

- L'une des méthodes les **plus efficaces pour améliorer les performances** d'une application Angular consiste à **différer autant que possible le chargement des ressources non critiques**.
- Les ressources **critiques** doivent être **chargées en premier**, tandis que les autres ressources **moins cruciales** doivent être chargées plus tard, uniquement si nécessaire. Ceci est fait à travers le lazy loading.
- Cependant, le lazy loading nous permet **uniquement de différer un composant complet**.
- Mais que se passe-t-il si vous disposez d'un **grand écran avec de nombreuses dépendances** et que vous souhaitez charger des **parties de la page, qui ne sont pas immédiatement visibles, au besoin**.
- Par exemple quand l'utilisateur **défile la page vers le bas** ou **clique sur un bouton**.
- Ce type de **chargement différé, très fin**, n'est tout simplement **pas possible avec le lazy loading**.
- **@defer** permet un chargement différé d'une partie de votre composant
- Il permet de dire **quand et pour quelle raison une partie sera chargée** dans le navigateur de votre client.

Optimisation @defer

- @defer est une **syntaxe des templates Angular**, qui vous permet de **charger des parties d'un modèle uniquement** lorsqu'elles sont nécessaires, **compte tenu d'une condition logique**.
- La syntaxe @defer nous permet d'implémenter des cas d'utilisation courants tels que :
 - **ne charge un composant volumineux** qu'une fois que l'utilisateur a fait défiler la **page après un certain point**
 - **charger** un composant volumineux uniquement après que l'utilisateur ait cliqué sur un bouton
 - **précharger** un composant volumineux en arrière-plan pendant que l'utilisateur lit la page, afin qu'il soit prêt au moment où l'utilisateur clique sur un bouton

Optimisation @defer

```
@defer(on interaction; prefetch on viewport) {  
  <app-huge/>  
}
```

- @defer offre deux niveaux de control:
 - Quand pré charger le bundle de la partie à différer
 - Quand la section pré chargé sera affichée à l'utilisateur
- Pour que les **dépendances** au sein d'un bloc **@defer soient différées**, elles doivent remplir deux conditions :
 - Elles **doivent être des composants autonomes (standalones)**. Les dépendances **non autonomes ne peuvent pas être différées** et seront **toujours chargées** au sein du **bundle principal, même à l'intérieur des blocs @defer**.
 - Ils ne doivent pas être **directement référencés depuis le même fichier, en dehors des blocs @defer** ; cela inclut les requêtes ViewChild.

Optimisation Hot Module Reload (HMR)

```
"scripts": {  
  "ng": "ng",  
  "start": "ng serve",  
  "build": "ng build",  
  "watch": "ng build --watch --configuration development",  
  "test": "ng test",  
  "defer:nohmr": "ng serve --no-hmr"  
},
```

- Le **HMR** permet de **recharger automatiquement uniquement les parties modifiées de votre application** Angular (comme un composant, un service ou un style) **sans avoir à recharger l'ensemble de la page ou à redémarrer le serveur de développement**.
- Cela **accélère le processus de développement** en **préservant l'état** de l'application (par exemple, les données dans un formulaire ou l'état d'une interface utilisateur) tout en appliquant les modifications en temps réel.
- Cependant, **en mode développement, il peut masquer le comportement réel de `@defer`**. En effet **tous les éléments différés sont directement chargés**, mais le **comportement lors de l'affichage reste intact**.
- Pour tester ce dernier, il faut **désactiver HMR** et recharger l'application.
- Désactiver le HMR en **ajoutant à `ng serve` l'option `--no-hmr`**

Optimisation @defer

- Afin d'appliquer `@defer` sur un bloc, il suffit de l'appeler dans votre template afin **d'englober la partie concernée**.
- Ceci permet la création **d'un bundle spécifique pour toute cette partie**
- Le nouveau bundle supplémentaire contenant le HugeComposant **ne sera chargé que lorsque le bloc @defer sera déclenché**.
- Ici, nous n'avons **spécifié aucun déclencheur** pour le bloc `@defer`, il sera donc **déclenché par défaut lorsque le navigateur est inactif (idle)**.
- Le navigateur est **considéré comme inactif** lorsque **toutes les ressources de la page ont fini de se charger**.

```
<p>defer examples!</p>
@defer {
  <app-huge/>
}
```

Lazy chunk files	Names	Raw size
chunk-5VYVMVAZM.js	signals-routes	204.20 kB
chunk-ZXPSBK30.js	change-detection-component	119.34 kB
chunk-DESNRMUU.js	cv-routing	106.30 kB
chunk-7S4X25Z5.js	flow-routes	11.92 kB
chunk-GA2RENBR.js	login-component	4.31 kB
chunk-KTHVBPQ0.js	huge-component	1.03 kB

Application bundle generation complete. [2.196 seconds]
Watch mode enabled. Watching for file changes...
→ Local: http://localhost:4200/
→ press h + enter to show help

Optimisation

@defer

Les éléments de defer

Deferrable Views (syntaxe)

@defer

Condition de chargement et
le composant à charger

@placeholder

affiché avant
déclenchement du chargement

@loading

affiché pendant le chargement

@error

affiché en cas d'erreur

```
@defer(on trigger) {  
    <my-comp />  
} @placeholder() {  
    <place-holder-comp />  
} @loading() {  
    <loading-comp />  
} @error() {  
    <err-comp />  
}
```

Optimisation

@defer

@placeholder

- Parfois, nous souhaitons simplement afficher un espace vide à l'endroit où est placé notre bloc `@defer`.
- Cependant, ceci peut avoir un comportement perturbant pour l'utilisateur, nous souhaitons donc **afficher du contenu initial à l'utilisateur, qui sera ensuite remplacé par le code chargé via le bloc `@defer`**.
- Nous pouvons le faire en encapsulant le contenu initial que nous souhaitons afficher dans un bloc **`@placeholder`**.
- Chaque composant, directive ou pipe introduite dans le bloc `@placeholder` **sera chargé dans le bundle principal**.

```
@defer {  
  <app-huge/>  
}  
@placeholder {  
  <h2>Nous chargeons le Huge Component merci de nous attendre :)</h2>  
}
```

Optimisation

@defer

@placeholder

- Comme vous l'avez remarqué, et selon l'état de la connexion, l'utilisation du **@placeholder** peut causer un effet de **vacillement (flickering)** qui est très désagréable pour l'utilisateur et provoque donc une **mauvaise expérience utilisateur**.
- Afin de gérer ça, nous pouvons utiliser le paramètre **minimum** que nous pouvons passer à **@placeholder**.
- Il permet de spécifier sa **durée minimale d'affichage** indépendamment du chargement du bloc différé.

```
@defer {  
  <app-huge/>  
}  
@placeholder(minimum 1s) {  
  <h2>Je suis là en attendant le vrai component:</h2>  
}
```

Optimisation

@defer

@loading

- Le bloc **@loading** est utilisé pour **afficher du contenu** pendant que le bloc **@defer charge toujours son bundle Javascript en arrière-plan**.
- Le contenu du block **@loading** sera rendu **uniquement pendant le chargement du bundle** app-huge. Une fois le **chargement terminé**, le message **sera supprimé de la page** et le app-huge sera affiché à sa place.
- Le bloc **@loading** accepte deux paramètres facultatifs; **minimum** et **after** :
 - **minimum** est utilisé pour spécifier la **durée minimale pendant laquelle le bloc @loading sera affiché à l'utilisateur**.
 - **after** est utilisé pour **spécifier le temps que nous devons attendre avant d'afficher l'indicateur @loading après le démarrage du processus de chargement. Si le composant se charge avant le after, ce bloque ne s'affichera pas**.

```
<p>defer works!</p>
<p>defer examples!</p>
@defer {
  <app-huge/>
}
@placeholder(minimum 1s) {
  <h2>Placeholder de AppHuge</h2>
}
@loading(minimum 1s; after 500ms){
  <h3>AppHuge is loading.....</h3>
}
```

Optimisation

@defer

@error

- Le chargement différé peut **échouer** pour une raison ou une autre (problème réseau, bug, ...)
- Afin de gérer ça vous pouvez utiliser le bloc **@error**

```
<p>defer works!</p>
<p>defer examples!</p>
@defer {
  <app-huge/>
}
@placeholder(minimum 1s) {
  <h2>Nous chargeons le Huge Component merci de nous attendre :)</h2>
}
@loading (minimum 1s; after 500ms) {
  <h3>loading.....</h3>
}
@error {
  <h2>Un problème est survenu, nous ne pouvons pas charger le Huge Component :)</h2>
}
```

Optimisation @defer

Les déclencheurs

5
9

```
@defer(on idle ; prefetch on idle) {  
  <app-huge/>  
}
```

- Comme mentionné au début, **@defer a deux niveaux de contrôle, chacun avec son déclencheur** :
 - le **déclencheur facultatif de prefetch**, qui **contrôle le moment où le bundle est chargé depuis le backend**,
 - le **déclencheur facultatif @defer**, qui contrôle le **moment où le bloc @defer est affiché à l'utilisateur**.
- Rappelez-vous, ce sont **deux événements très différents**, et nous pouvons les contrôler séparément et prendre en charge toutes sortes de cas d'utilisation avancés.
- Lorsqu'il s'agit de choisir le bon déclencheur, nous avons deux options disponibles :
 - Utiliser des **déclencheurs prédéfinis**, qui couvrent tous les cas d'utilisation les plus courants
 - Définir nos **déclencheurs personnalisés**, si nécessaire.
- Le mot-clé **on** est utilisé pour les **déclencheurs prédéfinis**, tandis que le mot clé **when** est utilisé pour les **déclencheurs personnalisés**.
- **idle est le déclencheur par défaut pour le chargement et l'affichage.**

Optimisation @defer

Les déclencheurs

declarative triggers

```
@defer(on hover; on timer(3s)) {  
  <my-comp />  
} @placeholder() {  
  <place-holder-comp />  
}
```

Are provided by Angular out of the box

- idle
- hover(target?)
- viewport(target?)
- interaction(target?)
- timer(delay)
- immediate

target = template variable (block placeholder si aucune target n'est précisée).

custom trigger

```
@defer(when isOpen()) {  
  <my-comp />  
} @placeholder() {  
  <place-holder-comp />  
}
```

A predicate or boolean

boolean | signal<boolean>

Optimisation

@defer

Les déclencheurs

-
- Angular offre 6 déclencheurs:
 - **idle** : déclenchera le chargement différé **une fois que le navigateur aura atteint un état d'inactivité** (détecté à l'aide de l'API requestIdleCallback). C'est le **comportement par défaut**.
 - **viewport** : déclenchera le bloc différé lorsque **le contenu spécifié entre dans la fenêtre** en utilisant le **IntersectionObservateur API**. Il peut s'agir du contenu @placeholder ou d'une référence d'élément.
 - **interaction** : déclenchera le blocage différé **lorsque l'utilisateur interagit avec l'élément spécifié** via des événements de **clic** ou de **keydown**.
 - **hover** : déclenchera le blocage différé **lorsque l'utilisateur va entrer dans l'élément spécifié**, les événements pris en considération ici **sont mouseenter et focusin**.
 - **immediate** : déclenche immédiatement le chargement différé, ce qui signifie qu'une fois que le client a terminé le rendu, le morceau (chunk) différé commencerait alors à être récupéré immédiatement.
 - **timer** : se déclencherait après une **durée spécifiée**. La durée est obligatoire et peut être spécifiée **en ms ou s**.

Optimisation @defer viewport

- **viewport** : déclenchera le bloc différé lorsque **le contenu spécifié entre dans la fenêtre** en utilisant le **IntersectionObservateur API**. Il peut s'agir du contenu `@placeholder` ou d'une référence d'élément.
- Vous pouvez aussi spécifiez **la référence** d'un élément de votre template qui sera le déclencheur lorsqu'on l'atteindra.

```
@defer(on viewport) {  
  <app-huge/>  
}  
@placeholder(minimum 1s) {  
  <h2>  
    Je suis là en attendant le vrai component:  
  </h2>  
}  
  
<p #deferTriggerElement>defer examples!</p>  
@defer(on viewport(deferTriggerElement)) {  
  <app-huge/>  
}  
@placeholder(minimum 1s) {  
  <h2>  
    Je suis là en attendant le vrai component:  
  </h2>  
}
```

Optimisation @defer interaction

- **interaction** : déclenchera le blocage différé lorsque l'utilisateur interagit avec l'élément spécifié via des événements de **clic** ou de **keydown**.
- Vous pouvez aussi spécifiez **la référence** d'un élément de votre template qui sera le déclencheur lorsqu'on interagie avec.

```
@defer(on interaction) {  
  <app-huge/>  
}  
@placeholder(minimum 1s) {  
  <h2>  
    Je suis là en attendant le vrai component:  
  </h2>  
}
```

```
<p #deferTriggerElement>defer trigger</p>  
@defer(on interaction(deferTriggerElement)) {  
  <app-huge/>  
}  
@placeholder(minimum 1s) {  
  <h2>  
    Je suis là en attendant le vrai component:  
  </h2>  
}
```

Optimisation @defer hover

- **hover** : déclenchera le blocage différé lorsque l'utilisateur va entrer dans l'élément spécifié, les événements pris en considération ici sont **mouseenter et focusin**.
- Vous pouvez aussi spécifiez **la référence** d'un élément de votre template qui sera le déclencheur **lorsqu'on y entre**.

```
@defer(on hover) {  
  <app-huge/>  
}  
@placeholder(minimum 1s) {  
  <h2>  
    Je suis là en attendant le vrai component:  
  </h2>  
}
```

```
<p #deferTriggerElement>defer trigger</p>  
@defer(on hover(deferTriggerElement)) {  
  <app-huge/>  
}  
@placeholder(minimum 1s) {  
  <h2>  
    Je suis là en attendant le vrai component:  
  </h2>  
}
```

Optimisation @defer

Combiner des déclencheurs

- **Plusieurs déclencheurs** d'événements peuvent être définis **simultanément**.
- Dans notre exemple,
on hover(deferTrigger); on timer(3s) signifie que le bloc de report sera déclenché si l'utilisateur entre dans l'espace réservé, **ou** après 3 secondes.
- **Plusieurs déclencheurs on activés** sont toujours des conditions **OU**.
- De même, les conditions combinées avec **when** sont également des conditions **OU**.

```
<p #deferTrigger>defer trigger</p>
@defer(on hover(deferTrigger); on timer(3s)) {
  <app-huge/>
}
```

Optimisation @defer

Créer vos triggers personnalisés

- Le principe d'un **trigger personnalisé** est simple. C'est une **expression qui retourne un booléen**.
- **Quand l'expression devient vrai, le trigger est déclenché et le placeholder est remplacé**
- Si la condition **when redevient fausse**, le bloc de placeholder **n'est pas rétabli à l'espace réservé**.

```
<input type="text" #myInput
  (keyup)="myInput.value.length > 4 ? loadDeffered = true : loadDeffered = false"
  class="form-control">
  @defer(when loadDeffered) {
    <app-huge/>
  }
  @placeholder(minimum 1s) {
    <h2>Je suis là en attendant le vrai component:</h2>
  }
```

Optimisation @defer Prefetching

```
@defer(on interaction; prefetch on viewport) {  
    <app-huge/>  
}
```

- @defer vous permet de **spécifier les conditions dans lesquelles le préchargement** des chunks des dépendances doit être déclenchée.
- Vous pouvez utiliser un mot-clé **prefetch**.
- La syntaxe de **prefetch** fonctionne de **manière similaire** aux principales conditions de **defer** et **accepte when** et/ou on pour déclarer le déclencheur.
- Comme nous l'avons mentionné avant, ceci lorsqu'on **associe when et on à defer**, nous contrôlons le **rendu**
- Lorsqu'on associe **prefetch avec on et when**, nous contrôlons **quand récupérer les ressources**.
- Cela permet des comportements plus avancés, tels que vous permettre de commencer à pré-extraire des ressources avant qu'un utilisateur n'ait réellement vu ou interagi avec un bloc différé

Angular Optimisation

AYMEN SELLAOUTI

Architecture de la couche Vue

- Lors de la construction d'une application Angular, l'une des questions les plus fréquentes auxquelles nous sommes confrontés dès le début est : **comment structurons-nous notre application ?**
- Une première réponse est : **Arbre de composant.**
- Cependant, d'autres questions se posent :
 - Est-ce que **tous les composants sont pareils?**
 - Comment les **composants doivent-ils interagir ?**
 - **Devons nous injecter des services dans n'importe quel composant ?**
 - Comment puis-je rendre **mes composants réutilisables** dans toutes les vues ?

Architecture de la couche Vue

Smart Components vs Presentational Components

- Un design pattern répond à ces questions en divisant les composants essentiellement en deux types
 - **Composants intelligents** : également appelés parfois composants applicatif ou composants conteneur.
 - **Composants de présentation** : également appelés parfois composants purs ou composants muets

Architecture de la couche Vue

Smart Components vs Presentational Components

Presentational Component

- Ce composant a uniquement pour **rôle** comme son nom l'indique, la **présentation des données** sans avoir d'informations sur la source de ces derniers.
- Ceci vous permet d'avoir des **composants indépendants et réutilisables**.
- Afin de paramétriser ce type de composant, utiliser le **@Input**
- Afin d'informer d'une action ou envoyer les données modifiées, utiliser le **@Output**.
- Essayer **d'utiliser ce type de composant autant que vous le pouvez.**

Architecture de la couche Vue

Smart Components vs Presentational Components

Smart Component

- Ces composants sont **smarts**. Ils **gèrent** donc la **partie fonctionnelle**.
- Ces composants sont chargés **d'interagir avec la couche de service** et de récupérer les données.
- Ils sont aussi **chargés de transmettre aux composants de présentation les informations nécessaires**.
- Ils doivent aussi **gérer les données et les évènements** envoyés par les **Presentational Component**.

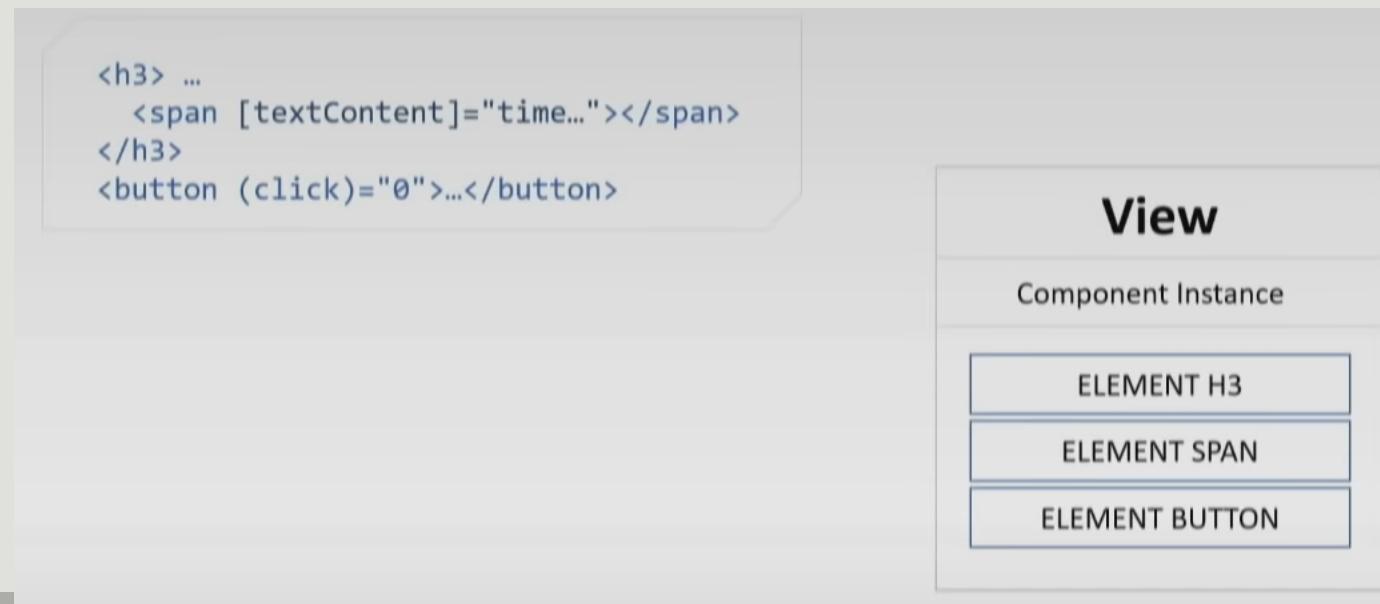
Change Detection

C'est quoi ?

- **Change Detection** est l'un des mécanisme 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**.

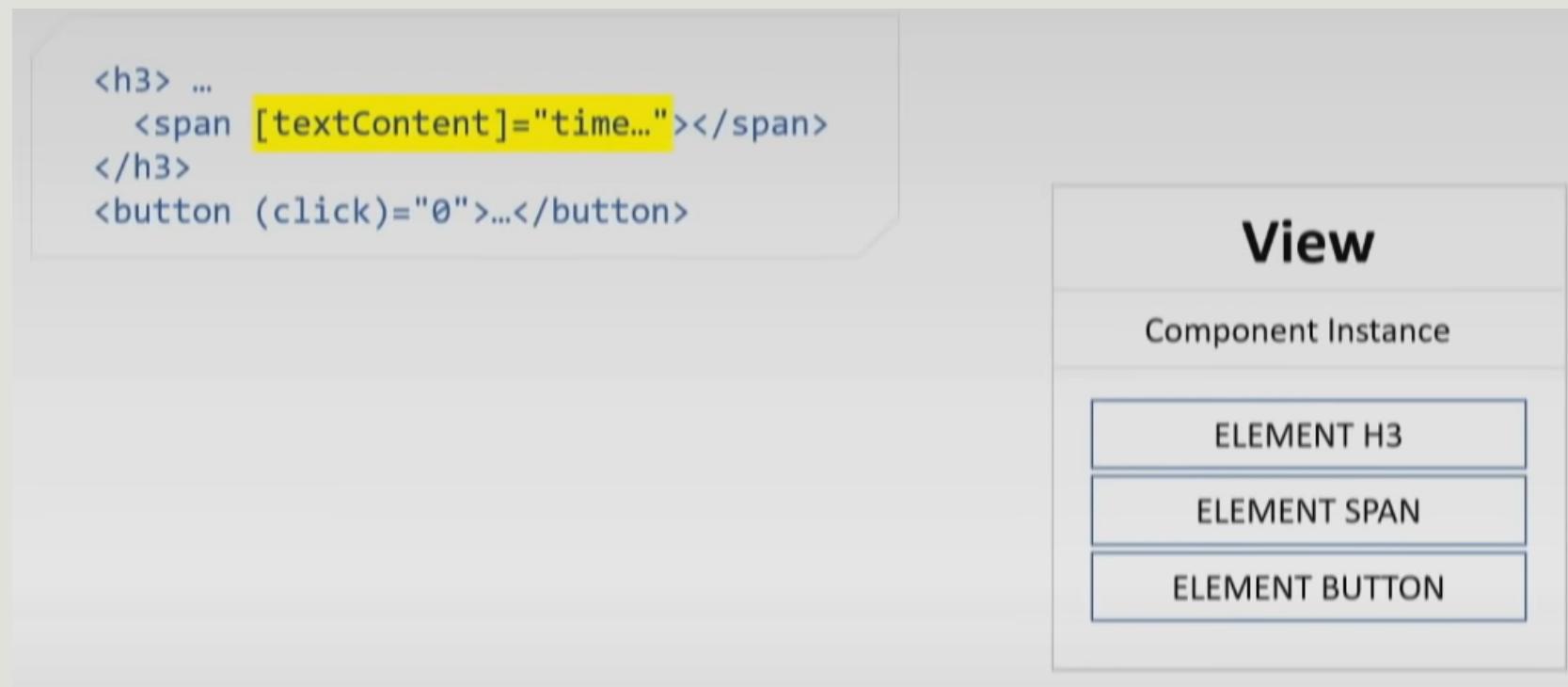
Change Detection

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



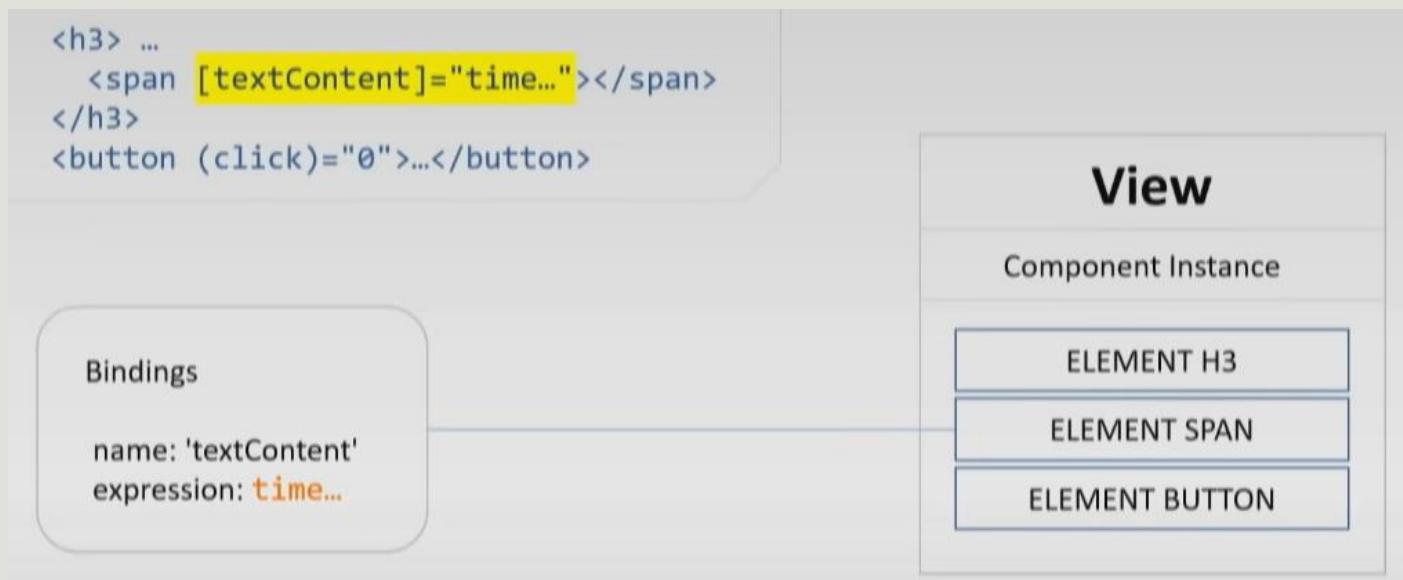
Change Detection

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



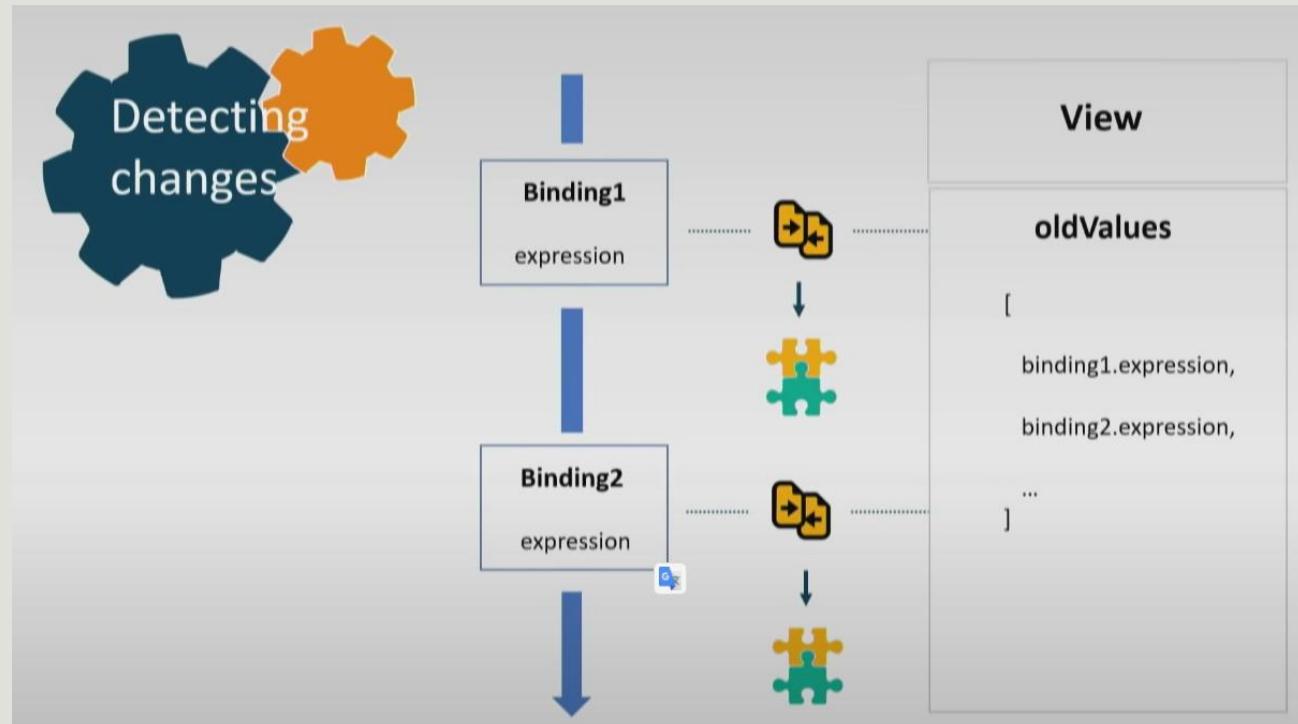
Change Detection

- 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

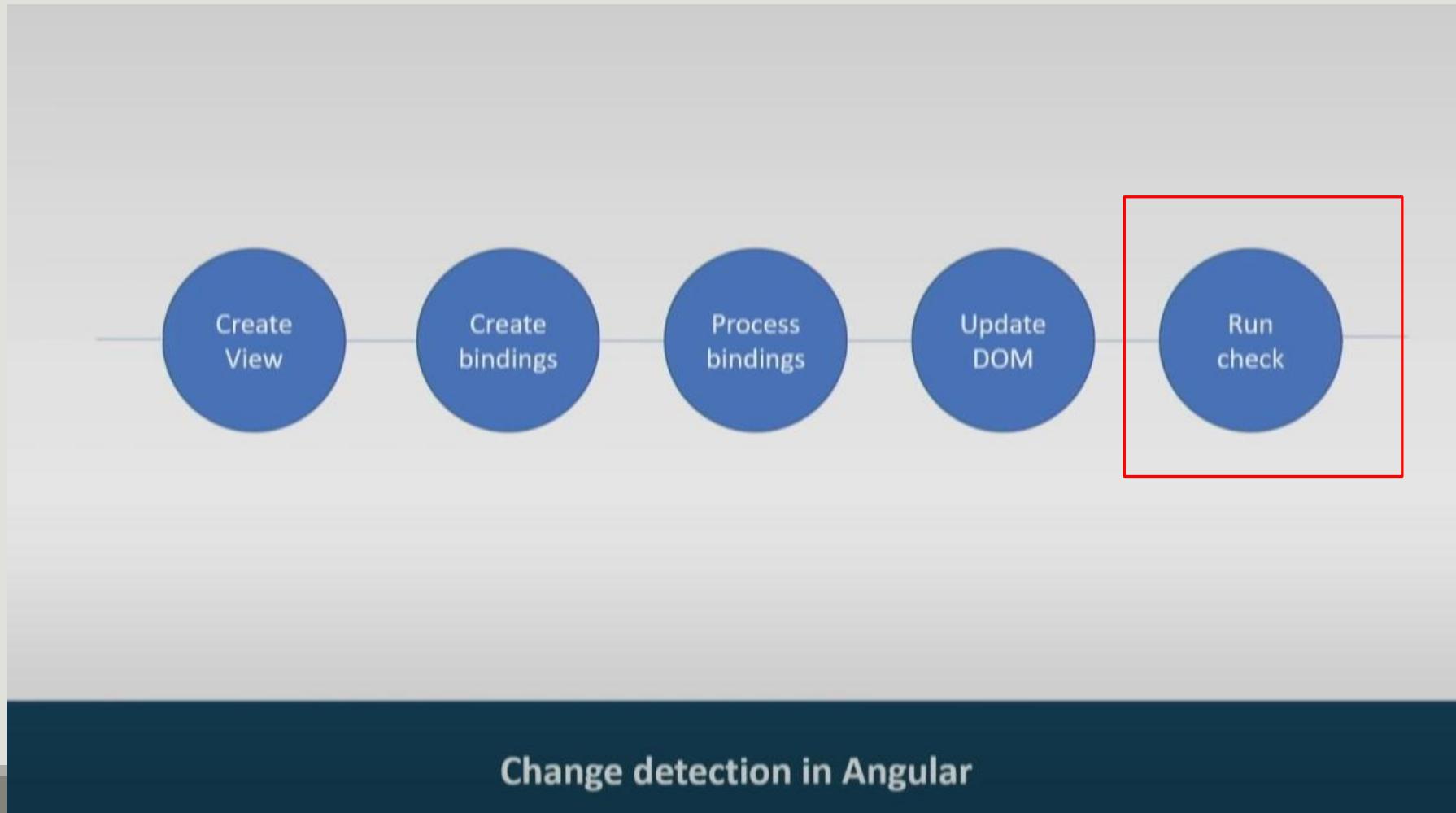


Change Detection

- Ensuite, dès qu'un **change Detectin** 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



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

Quand déclencher un Change Detection

- La liste précédente contient les scénarios **les plus courants** dans lesquels **l'application peut modifier les données**.
- Angular **exécute le Change Detection** à chaque fois qu'il **déetecte la possibilité d'un changement de données**.
- Le résultat de la détection des changements est que le **DOM est mis à jour avec de nouvelles données**.
- Angular détecte les changements de différentes manières. Pour **l'initialisation des composants**, Angular appelle explicitement la détection des changements.
- Pour les **opérations asynchrones**, Angular utilise une **zone** pour détecter les changements aux endroits où les données auraient pu muter et **exécute automatiquement la détection des changements**.

Change Detection

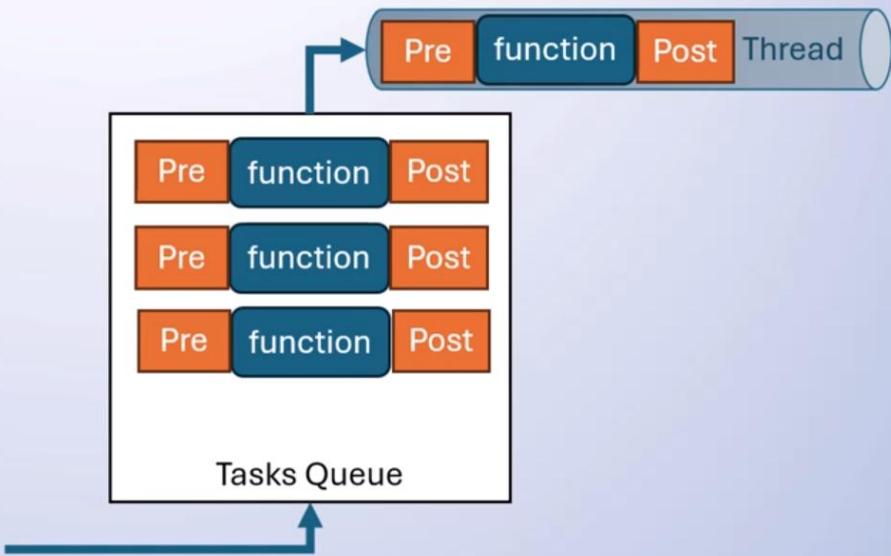
Zones et contexte d'exécution

- Afin de **déetecter les éléments susceptibles de déclencher un Change Détection**, Angular utilise **zone.js**.
- zone.js peut **suivre et intercepter** les tâches asynchrones.
- Une zone a généralement 3 phases :
 - Elle commence dans un état stable
 - Elle devient instable lorsque une tâche est déclenchée dans la zone
 - Elle redevient stable lorsque les tâches sont finalisées.
- Angular suit plusieurs API de navigateur de bas niveau au démarrage pour pouvoir détecter les changements dans l'application

Change Detection Zones

Zone - Simplified

- `setTimeout` (`function`)
- `setInterval` (`function`)
- `addEventHandler` (`function`)
- `XHR.send` (`function`)
- `Script file loaded` (`function`)
- `Promise.then` (`function`)
- `Promise.catch` (`function`)
- `queueMicrotask` (`function`)



Change Detection

Zone.js c'est quoi ?

```
//Comment savoir quand est ce que le setTimeout commence et quand ca se termine
//sans toucher le setTimeout
const oldSetTimeout = setTimeout;
setTimeout = (handler, timer) => {
  console.log('START');
  oldSetTimeout(_ => {
    handler();
    console.log('FINISH');
  }, timer);
}
//-----
setTimeout(_ => {
  console.log('some action');
}, 3000);
```

Change Detection

Zones et contexte d'exécution

- Ceci est donc **délégué à zone.js** qui suit les APIs comme EventEmitter, DOM event listeners, XMLHttpRequest, l'API fs dans Node.js et plus encore.
- Donc **si zoneJs détecte un des déclencheurs** du Change Detection, elle **notifie Angular** qui lui va déclencher le processus de Change Detection.
- Angular utilise sa **propre zone** appelée **NgZone**.
- C'est une seule zone et le Change Detection est uniquement déclenché si une **opération Asynchrone est déclenchée dans cette zone**.

Change Detection

Zones et contexte d'exécution

```
// https://github.com/angular/angular/blob/master/packages/core/src/zone/ng_zone.ts#L337
// ngzone simplified
function onEnter() {
  _nesting++;
}
function onLeave() {
  _nesting--;
  checkStable();
}
function checkStable() {
  if (zone._nesting == 0 && !zone.hasPendingMicrotasks) {
    onMicrotaskEmpty.emit(null);
  }
}
//https://github.com/angular/angular/blob/7954c8dfa3c85d12780949c75f1448c8d783a8cf/packages/core/src/application_ref.ts#L628
```

```
this._onMicrotaskEmptySubscription = this._zone.onMicrotaskEmpty.subscribe({
  next: () => {
    this._zone.run(() => {
      this.tick();
    });
  }
})
```

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

Change Detection

- Il existe **deux stratégies** de Change Detection :
 - La stratégie **par défaut**
 - La stratégie **OnPush**

Parcours de l'arbre des composants

Dans la configuration par défaut, les règles suivantes décrivent l'exécution du processus:

- le processus démarre à partir du ou des composants racine.
- L'arborescence entière des composants est vérifiée pour détecter les modifications, ce qui signifie que chaque nœud est visité.
- Le parcours s'effectue de haut en bas.
- L'ordre exact de visite des nœuds suit un algorithme de recherche en profondeur (DFS).

Change Detection

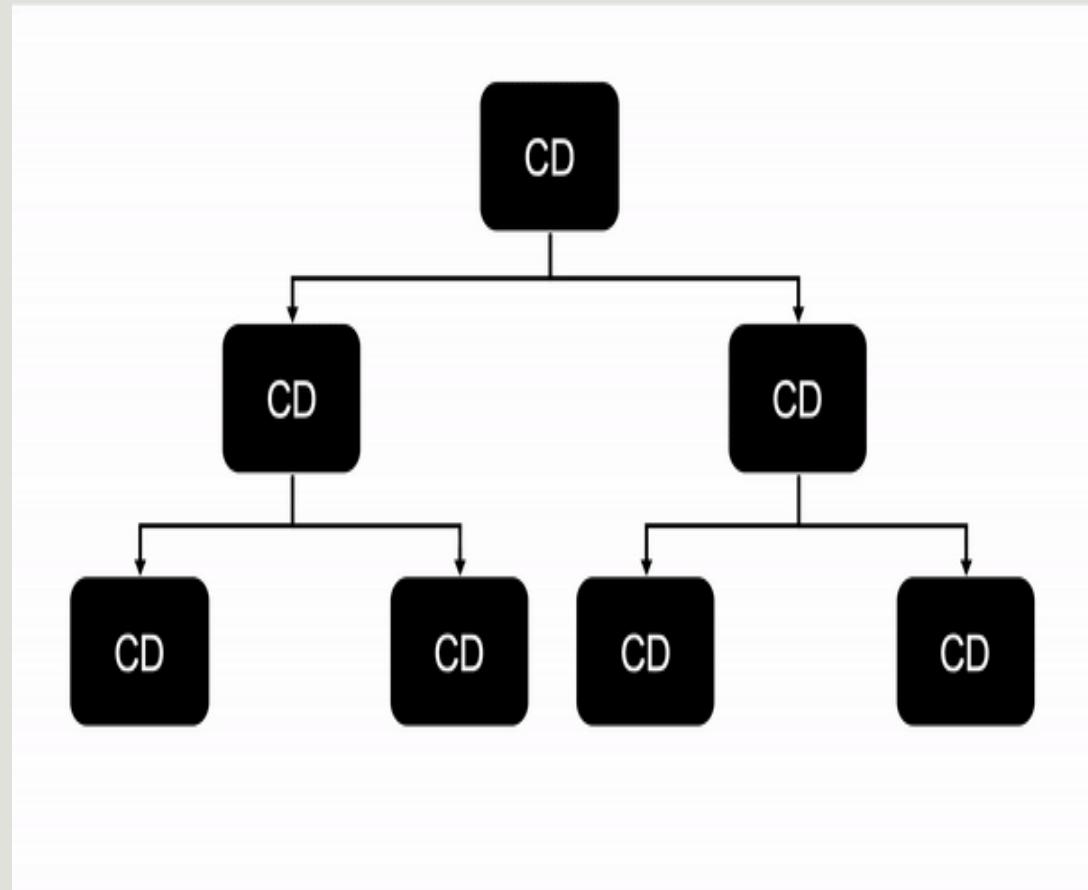
La stratégie par défaut

- Dans la stratégie par défaut, le mécanisme est le suivant :
 1. NgZone détecte une possibilité de modification.
 2. Angular est notifié afin de déclencher **le change detection**.
 3. La détection de changement **vérifie chaque composant dans l'arborescence des composants** de haut en bas **pour voir si le modèle correspondant a changé**. Ceci est appelé **dirty checking**.
 4. S'il y a une nouvelle valeur, **il mettra à jour la partie correspondante (DOM)**

Change Detection

La stratégie par défaut

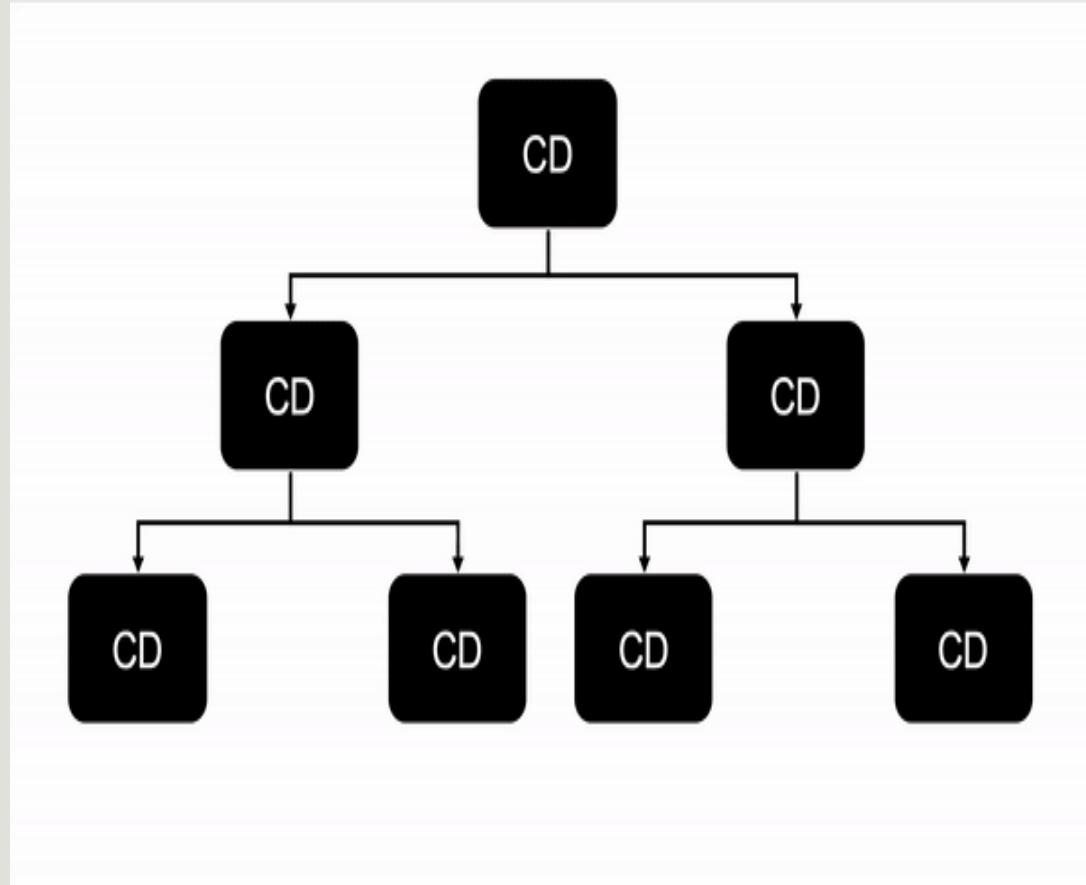
- Ici, dans **tous les composants** de l'arbre de composant, le **change detector alloué à chaque composant**, compare la valeur courante et la valeur précédente des propriétés.
- Si la **valeur change**, il va marquer une propriété **isChanged à true**.



Change Detection

La stratégie par défaut

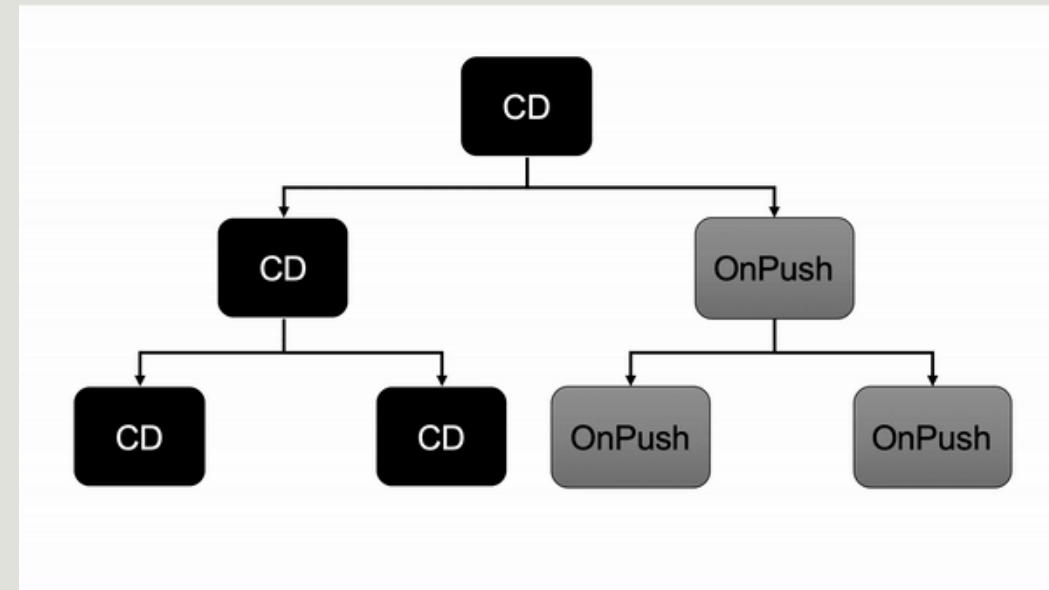
- Cette première stratégie est assez **performante pour les application petite et moyenne.**
- Ceci est du au fait qu'Angular est **très rapide pour le Change détection** pour chaque composant en utilisant la technique du **inline-caching**.
- Cependant, ceci peut ne plus suffire pour les application assez volumineuse.



Change Detection

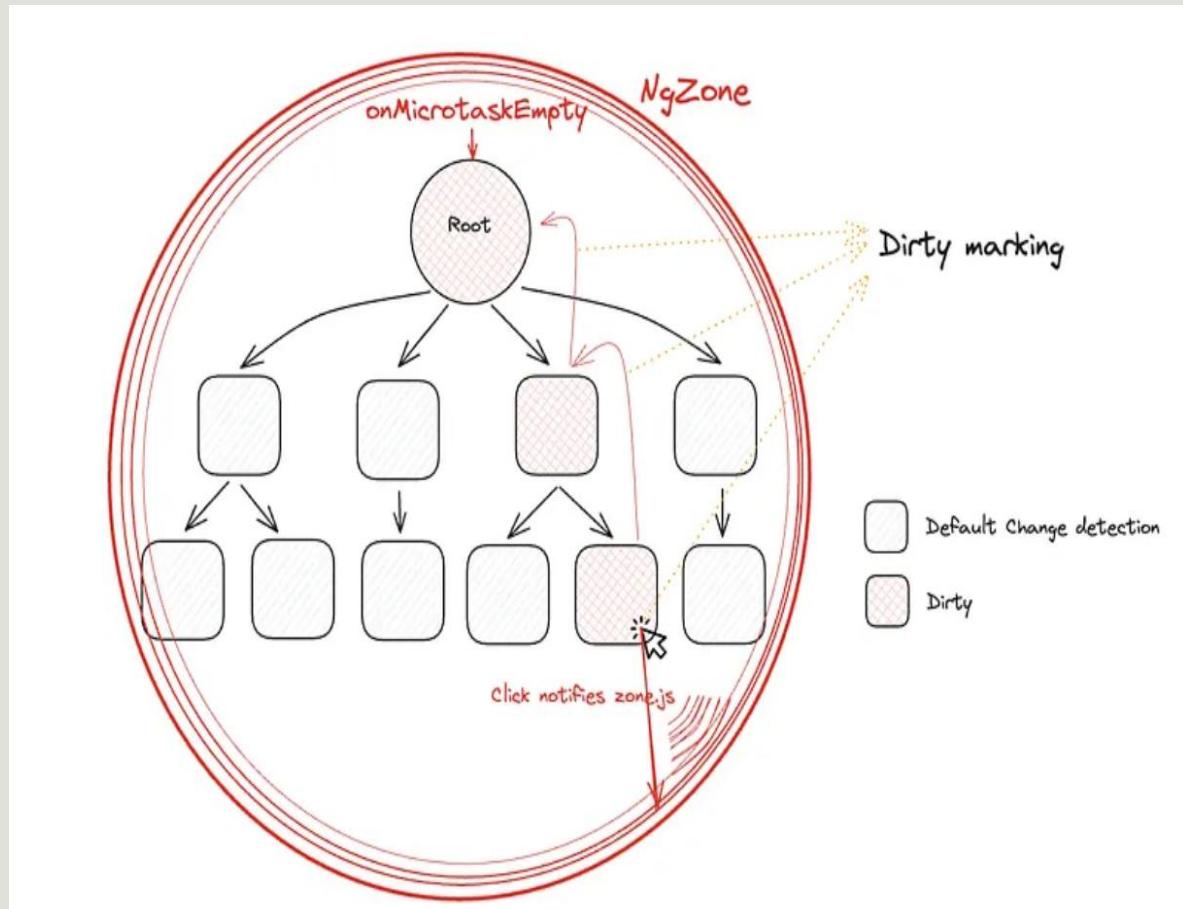
La stratégie OnPush

- Le but de cette stratégie est d'**éviter** les **tests non nécessaires** pour un **composant et sa descendance**.
- En appliquant cette stratégie, on définit une nouvelle façon pour le déclenchement du Change Detection pour ce composant



Mécanisme de Dirty Marking

- Pour comprendre cette logique, il faut comprendre le concept de Dirty Marking.
- Quand un évènement se déclenche dans un composant, **angular le marque comme dirty**.
- Etant donné que le parcours de l'arbre se fait de haut en bas, si Angular détecte qu'un composant est non dirty il n'a pas intérêt à le tester ni lui ni sa descendance.
- Pour que le OnPush fonctionne, il faut donc que toute la hiérarchie soit dirty.

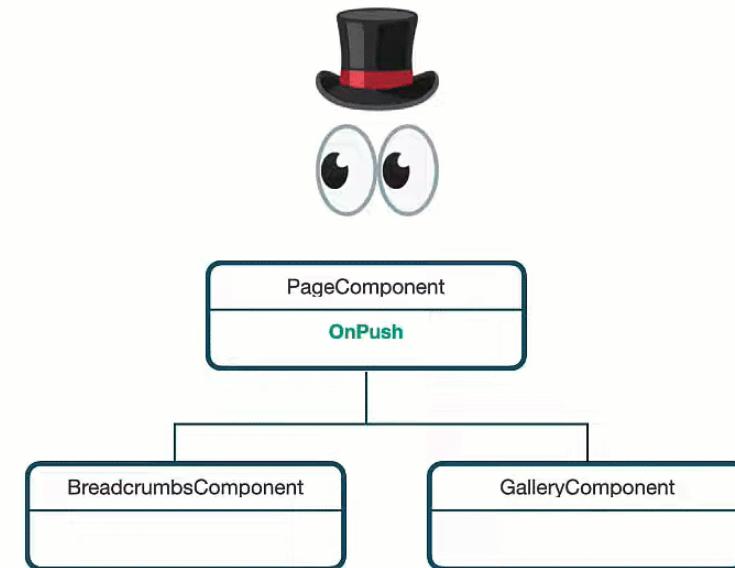


Change Detection

La stratégie OnPush

- Afin d'activer la stratégie **OnPush**, il suffit d'ajouter la propriété **changeDetection** du **@Component** object et de lui affecter la valeur **OnPush** de l'objet **ChangeDetectionstrategy**.

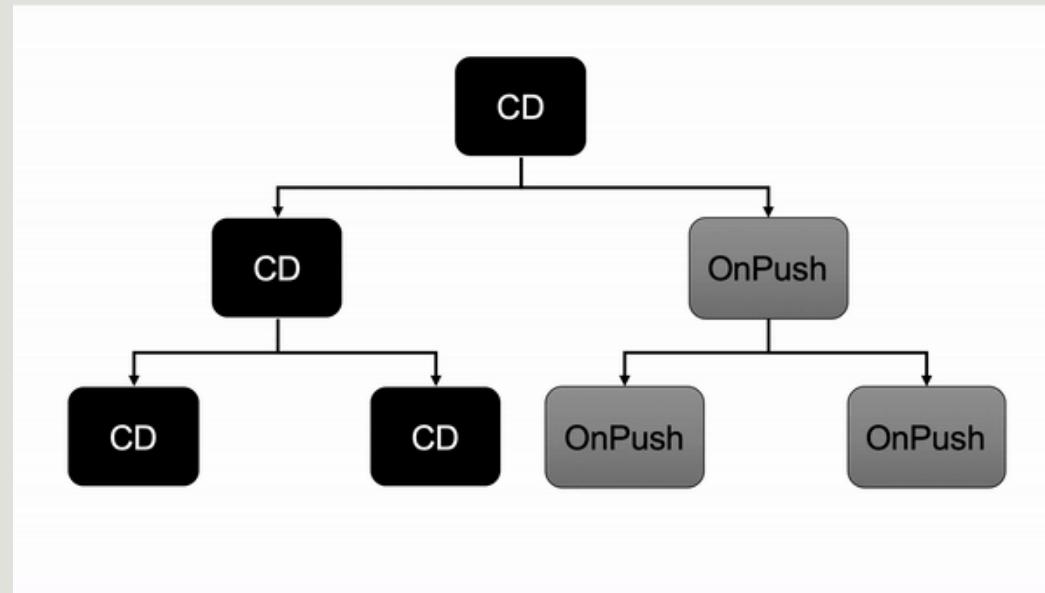
```
@Component({  
  selector: 'app-list',  
  templateUrl: './list.component.html',  
  styleUrls: ['./list.component.css'],  
  changeDetection: ChangeDetectionStrategy.OnPush  
})
```



Change Detection

La stratégie OnPush

- En utilisant cette stratégie, Angular sait que le **composant n'a besoin d'être mis à jour que si :**
 - La **référence** d'entrée d'un **Input** du composant est **modifié**
 - Le **composant ou l'un de ses enfants** déclenche un **événement**.
 - La **Change Detection** est déclenchée **manuellement**
 - Un **observable** lié au Template via le pipe **async émet une nouvelle valeur.**



Change Detection

La stratégie OnPush

- Les actions suivantes **ne déclenchent pas le Change Detection** dans ce contexte :
 - setTimeout
 - setInterval
 - Promise.resolve().then(), Promise.reject().then()
 - this.http.get('...').subscribe() (En générale, n'importe quelle inscription à un Observable RxJs)

Change Detection

La stratégie OnPush

Le changement de la référence Input

- Dans la stratégie de détection de changement par défaut, **Angular exécutera le détecteur de changement chaque fois que les données @Input() sont changées ou modifiées.**
- En utilisant la stratégie OnPush, le détecteur de changement n'est déclenché que si une nouvelle référence est transmise en tant que valeur @Input().
- Les types primitifs tels que les nombres, les chaînes, les booléens, null et indéfini sont passés **par valeur**.

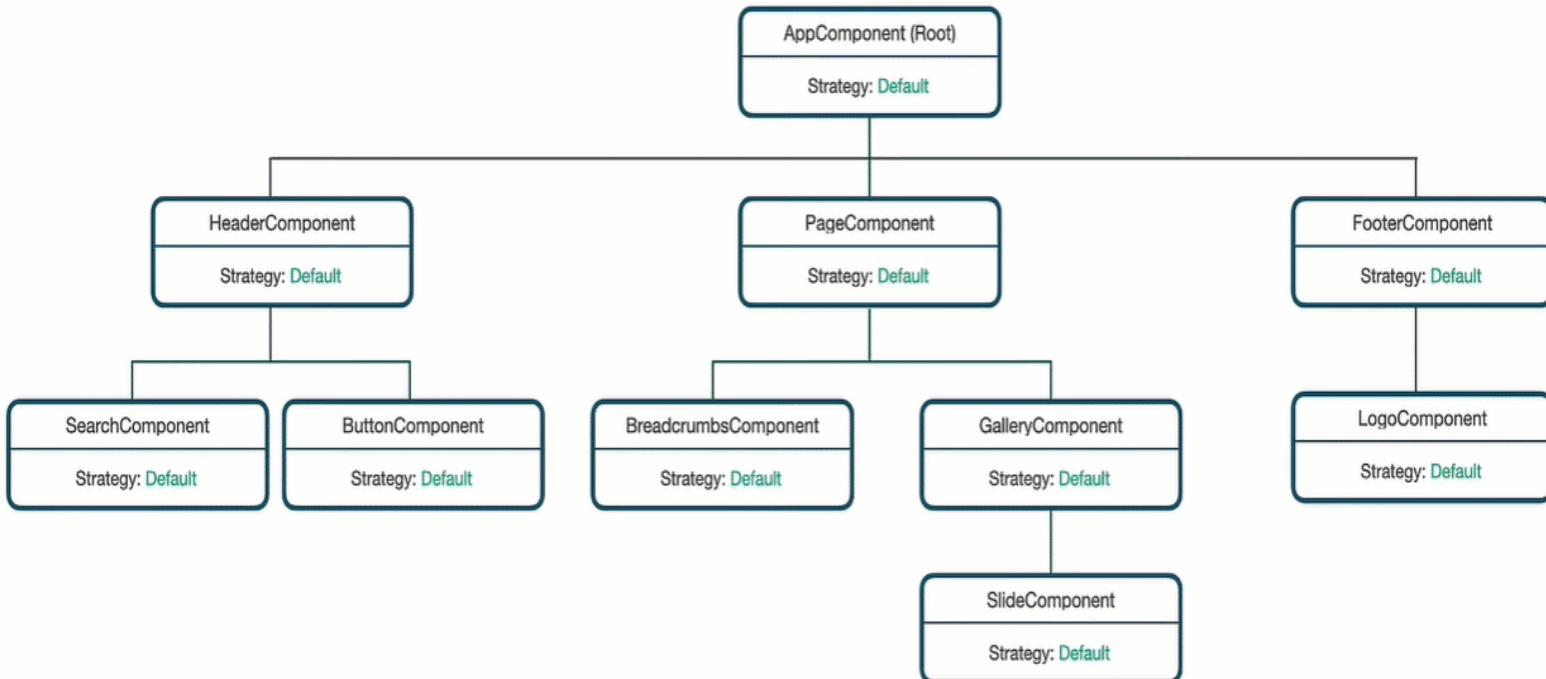
Change Detection

La stratégie OnPush

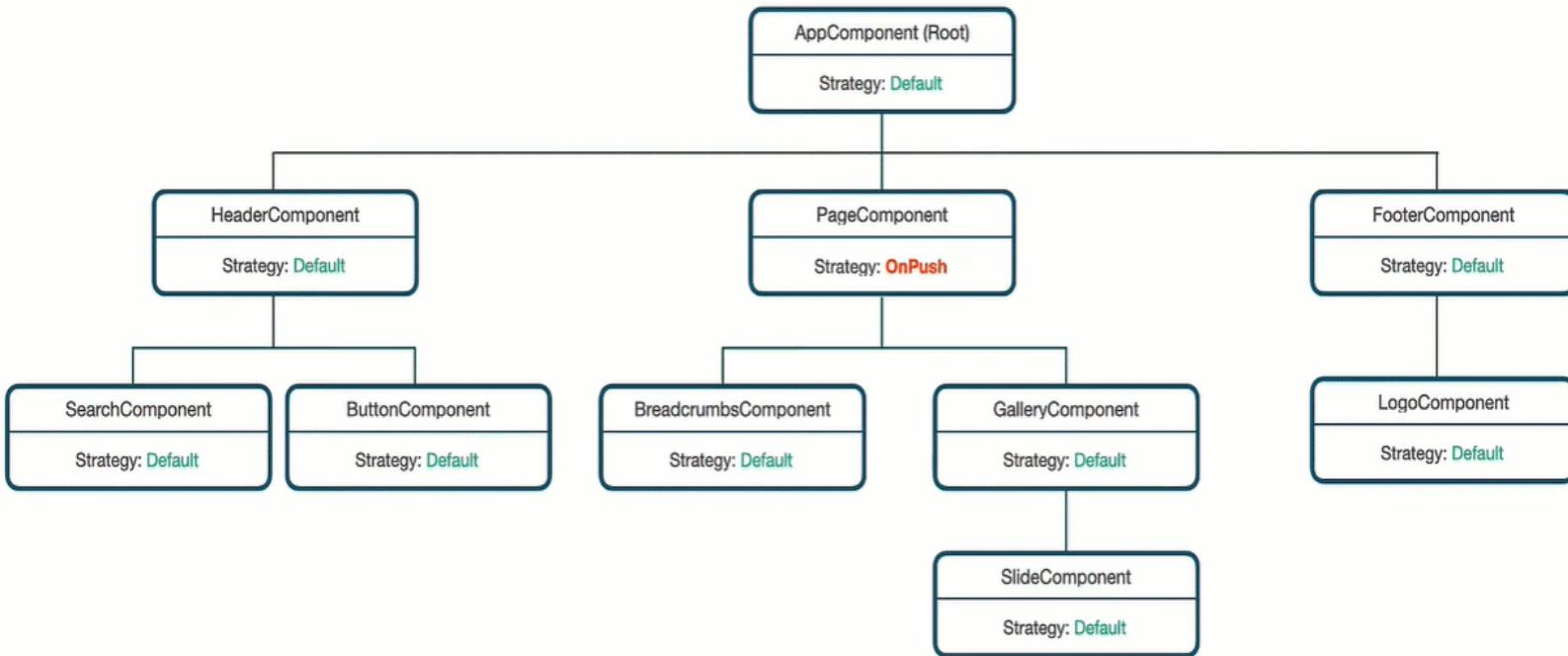
Le changement de la référence Input

- L'objet et les tableaux sont également **passés par valeur**, mais **la modification des propriétés** d'objet ou des entrées de tableau **ne crée pas de nouvelle référence et ne déclenche donc pas la détection de changement** sur un composant OnPush.
- Pour **déclencher le détecteur de changement**, vous devez passer une **nouvelle référence** d'objet ou de tableau à la place.
- Immutable.js facilite l'utilisation de l'Immutabilité.
- Il fournit des structures de données immuables persistantes pour les objets (Map) et les listes (List).

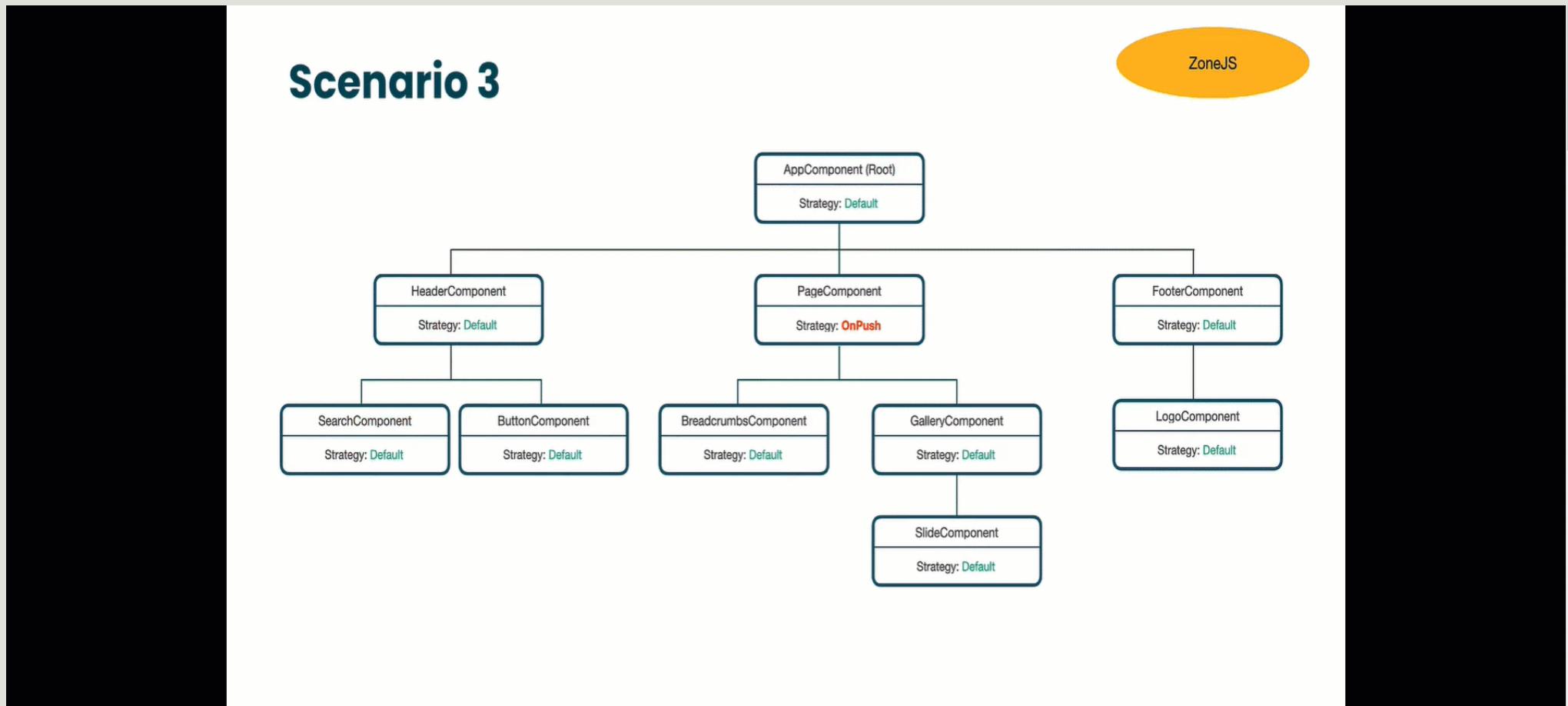
Scenario 1



Scenario 2

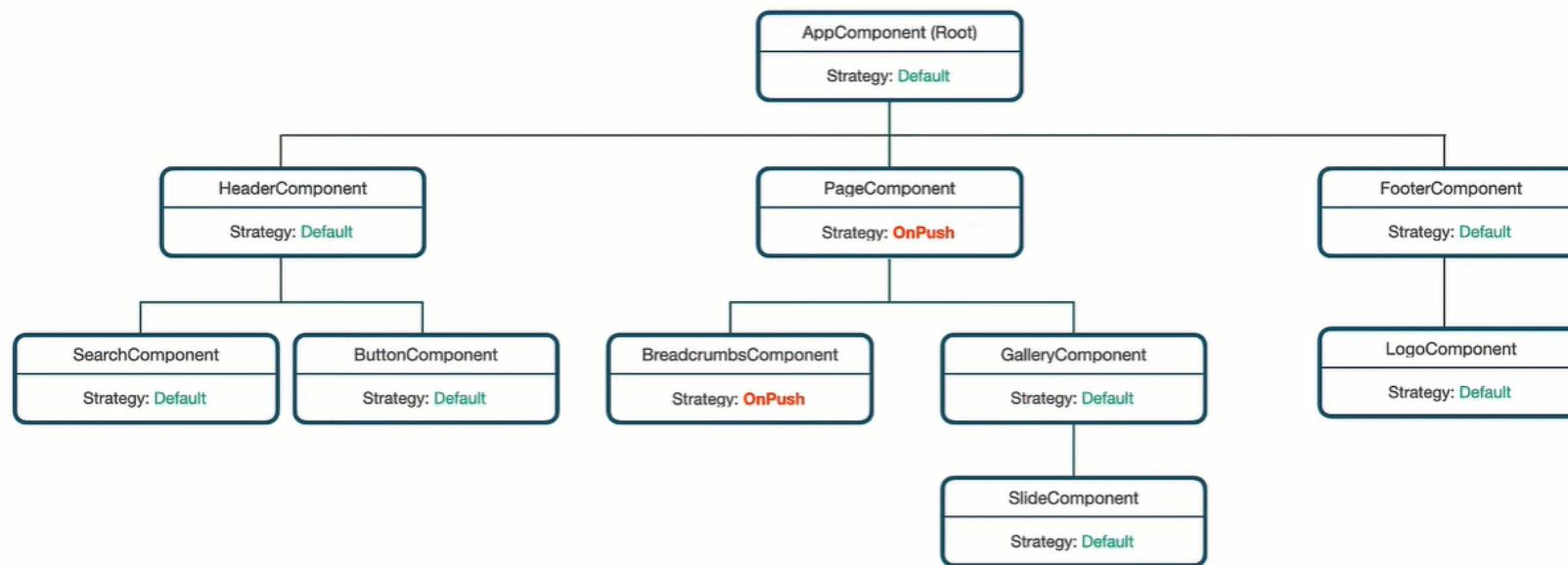


- PageComponent est OnPush
- AppComponent passe une input à Page Component



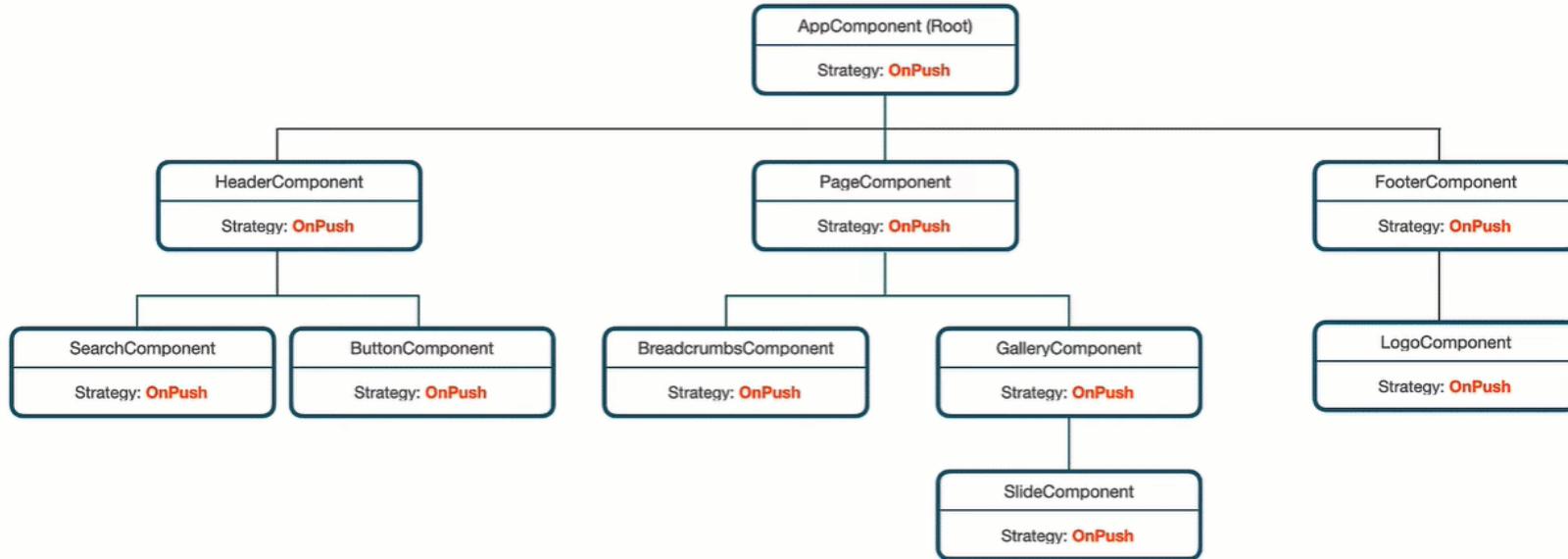
- PageComponent et BredCrumbComponent sont OnPush
- AppComponent passe une input à Page Component

Scenario 4



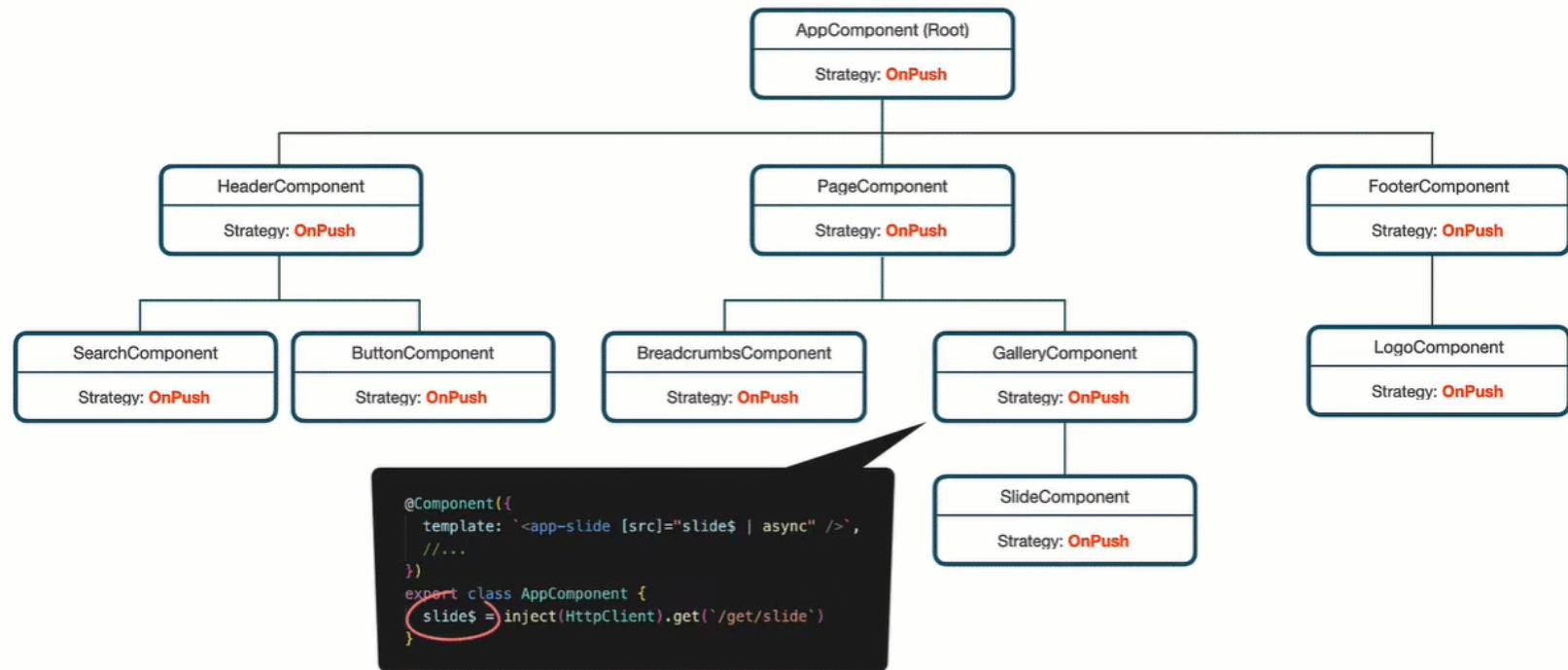
➤ Tous les composants sont OnPush

Scenario 5



- Tous les composants sont OnPush
- GalleryComponent A un pipe Async et elle passe le résultat en Input à SliderComponent

Scenario 6



Change Detection

La stratégie OnPush

-
- L'objet et les tableaux sont également **passés par valeur**, mais **la modification des propriétés** d'objet ou des entrées de tableau **ne crée pas de nouvelle référence et ne déclenche donc pas la détection de changement** sur un composant OnPush.
 - Pour **déclencher le détecteur de changement**, vous devez passer une **nouvelle référence** d'objet ou de tableau à la place.
 - Immutable.js facilite l'utilisation de l'Immutabilité.
 - Il fournit des structures de données immuables persistantes pour les objets (Map) et les listes (List).

Change Detection

Le changement déclenché manuellement

Zone Pollution Pattern

- Ce problème est identifié comme « **Zone Pollution Pattern** »
- Il se produit lorsque Angular Zone encapsule un callback qui déclenche des détections de changement redondants.
- **La solution est donc de déplacer la logique d'initialisation à l'extérieur de Angular Zone**
- **Injecter NgZone**
- Appeler la méthode **runOutsideAngular**
- Passez y une **callback** qui **effectue le traitement que vous voulez isoler.**

Change Detection

Le changement déclenché manuellement

Zone Pollution Pattern

```
constructor (ngZone: NgZone) {  
  ngZone.runOutsideAngular(() => {  
    // runs outside Angular zone, for performance-critical code  
  
    ngZone.run(() => {  
      // runs inside Angular zone, for updating view afterwards  
    });  
  });  
}
```



View and model can
get out of sync!

Change Detection

Le changement déclenché manuellement

Zone Pollution Pattern

- Vous pouvez aussi désactiver complètement la prise en main de zone.js et tout faire vous-même.

```
platformBrowserDynamic().bootstrapModule(AppModule , [  
    {ngZone: 'noop'}  
])
```

```
constructor(private applicationRef: ApplicationRef) {  
    applicationRef.tick();  
}
```

Optimiser une application

Change Detection - La stratégie OnPush

Out of Bound Change Detection

- Ce problème se produit lorsqu'une **action qui modifie uniquement l'état local d'un composant déclenche des changes detection dans des parties qui n'ont aucun rapport avec cette action.**
- Solution : Utiliser OnPush et considérer du refactoring de votre composant.

Problem

Local state change triggers out of bounds change detection

Identification

Change detection performed in components outside the scope of the change

Resolution

Use OnPush and consider refactoring

Change Detection

La stratégie OnPush

Le changement de la référence Input

Optimisation

- Pensez à externaliser les parties où vous avez des évènements et le component recevant le **@Input**.
- Ceci va faire en sorte que le composant avec le **@Input ne sera réaffiché que lorsqu'il aura une nouvelle référence.**

Optimiser une application

Recalculation of referentially transparent expressions

- Cette problématique est soulevée lorsque vous avez une **expression** dans votre Template qui doit être **recalculé même lorsque ses paramètres ne changent pas**.

Problem

Redundant calculations

Identification

Detection for changes takes longer
than expected given the state changes

Resolution

Use pure pipes or memoization

Change Detection

Recalculation of referentially transparent expressions

Optimisation

- Pensez à utiliser des **pipes** pour vos calculs.
 - Il faut avoir deux conditions :
 - **Pas d'effets de bords** (side effect), vous nappelez pas d'api, pas de console,
...
 - Se sont des **opérations pures**, si l'entrée ne change pas, le résultat ne change pas il reste le même
- Pourquoi recalculer alors => utiliser les pipes, si l'entrée ne change pas, il ne recalculera pas l'opération.

Change Detection

Recalculation of referentially transparent expressions

Optimisation

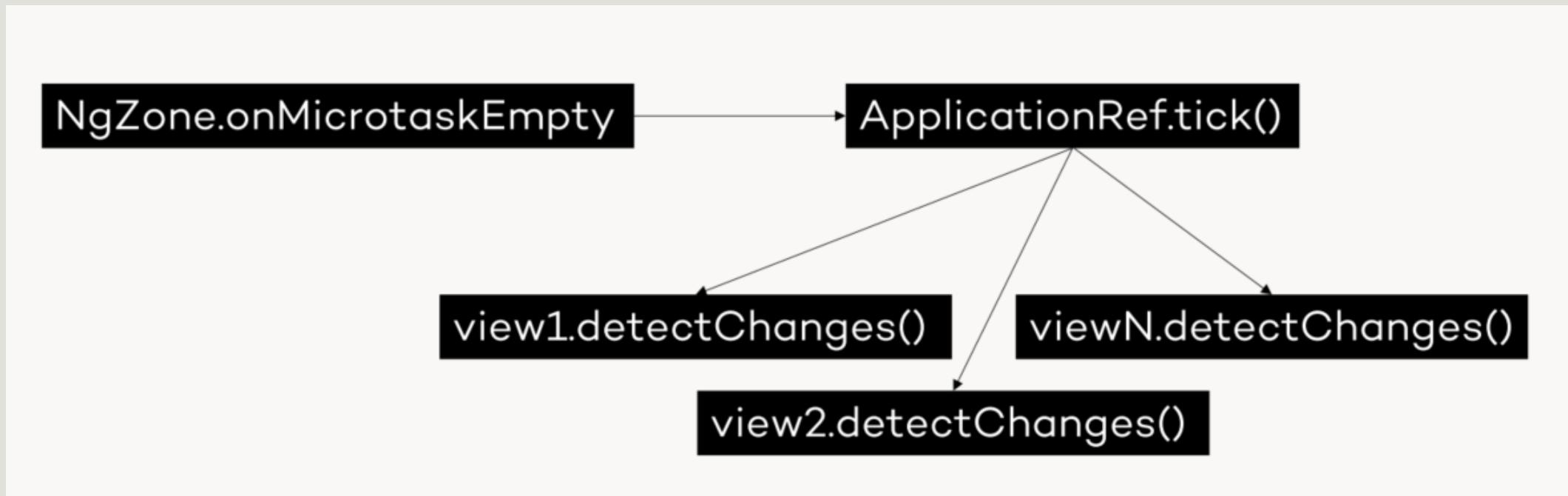
- Vous pouvez alors aller encore plus dans l'optimisation, puisque ce sont des **fonctions pures**. Si une fonction a été **déjà calculée, pourquoi le refaire** même si ce n'est pas la même entrée.
- Pensez à utiliser le concept de cache avec le **memo-decorator**.
- importer le décorateur memo de la bibliothèque **memo-decorator** et appliquer le à votre fonction transform.

npm i memo-decorator

Change Detection

Le changement déclenché manuellement

- Par défaut c'est zone.js à travers NgZone qui gère la partie déclenchement du change detection.



Change Detection

Le changement déclenché manuellement

```
@Injectable()
export class ApplicationRef {
  // ...
  constructor /* ... */ {
    this._zone.onMicrotaskEmpty
      .subscribe( {next: () => { this._zone.run(() => { this.tick(); }); }});
  }
  // ...
  tick(): void {
    // ...
    for (let view of this._views) {
      view.detectChanges();
    }
    // ...
  }
  // ...
}
```

Change Detection

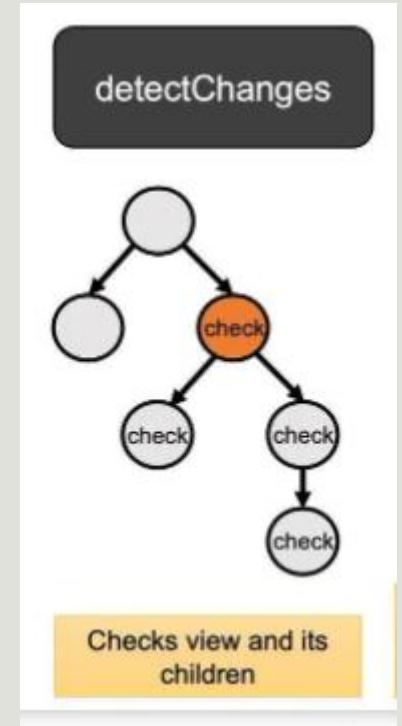
Le changement déclenché manuellement

- Il existe trois méthodes pour déclencher manuellement les détections de changement :
- **La méthode `tick()` de `ApplicationRef`** qui déclenche la détection de changement pour **l'ensemble de l'application** en **respectant la stratégie de détection de changement d'un composant**

Change Detection

Le changement déclenché manuellement

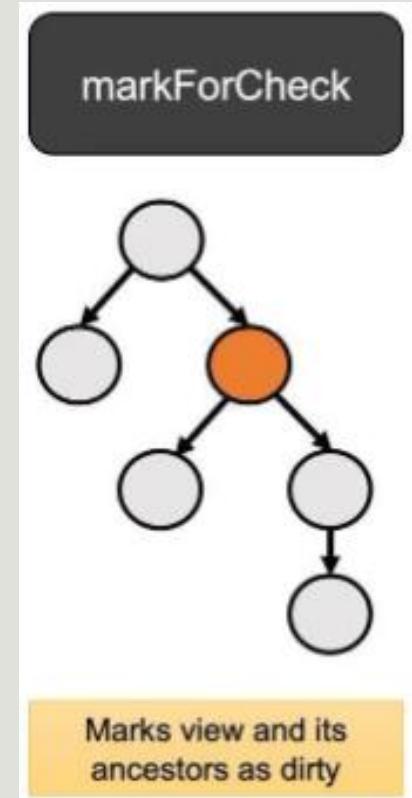
- Il existe trois méthodes pour déclencher manuellement les détections de changement :
- `detectChanges()` du **ChangeDetectorRef** qui **exécute la détection de changement sur cette vue et ses enfants** en gardant à l'esprit la stratégie de détection de changement.
- Il peut être utilisé en combinaison avec `detach()` pour implémenter des contrôles de détection de changement locaux.



Change Detection

Le changement déclenché manuellement

- Il existe trois méthodes pour déclencher manuellement les détections de changement :
- `markForCheck()` de `ChangeDetectorRef` qui **ne déclenche pas la détection de changement** mais **marque tous les ancêtres OnPush** comme devant être vérifiés une fois, soit dans le cadre du cycle actuel ou du cycle de détection de changement suivant. Il exécutera la Change Detection sur les composants **marqués même s'ils utilisent la stratégie OnPush**.

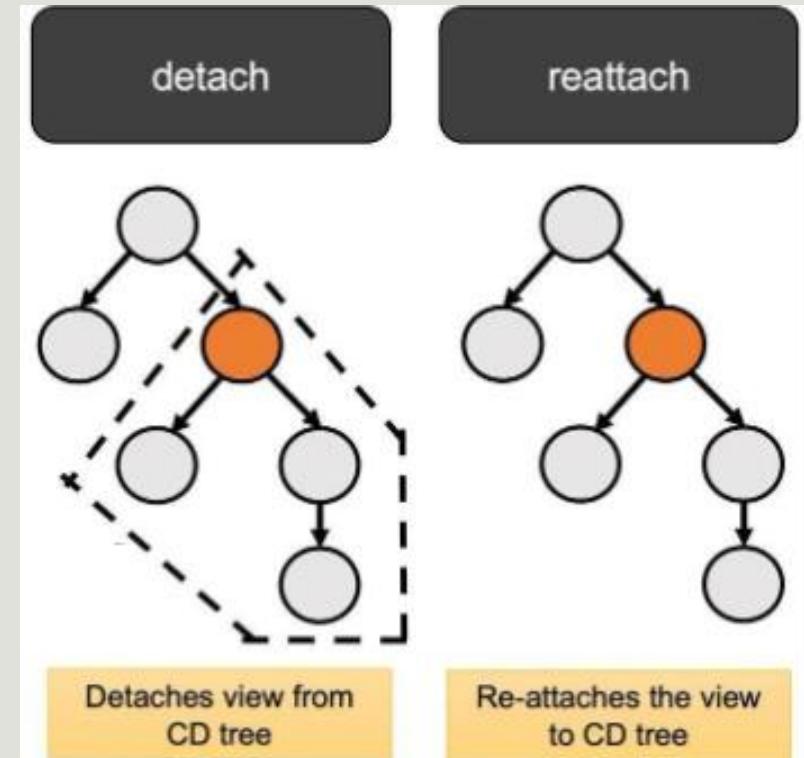


Change Detection

Détacher un composant du change detection

- Vous pouvez **détacher un composant complètement du Change Detection**.
- Ceci peut être fait pour les **composants qui n'ont pas d'état** et donc pas besoin que le ChangeDetector agisse.
- Vous avez **beaucoup de calcul lourd** et vous voulez **gérer seul le déclenchement du Change Detection** localement.
- Vous pouvez rattacher le composant quand c'est nécessaire.

```
constructor(cdRef: ChangeDetectorRef) {  
    cdRef.detach(); // detaches this view from the CD tree  
    // cdRef.detectChanges(); // detect this view & children  
    // cdRef.reattach();  
}
```



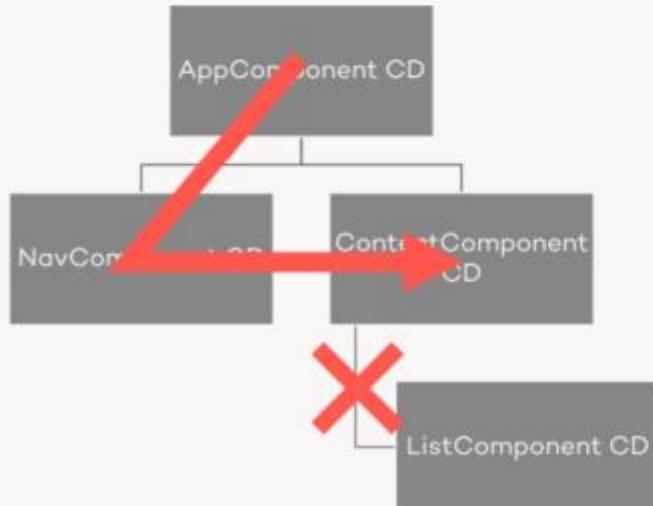
Change Detection

Détacher un composant du change detection

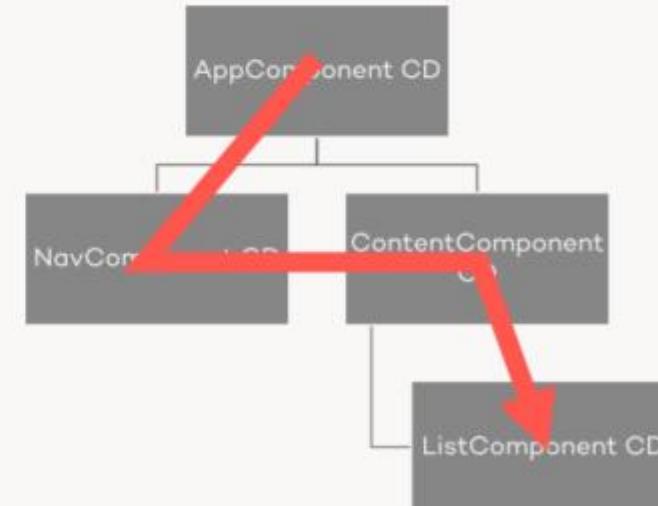
Change Detector

Detaching Components

`changeDetector.detach();`

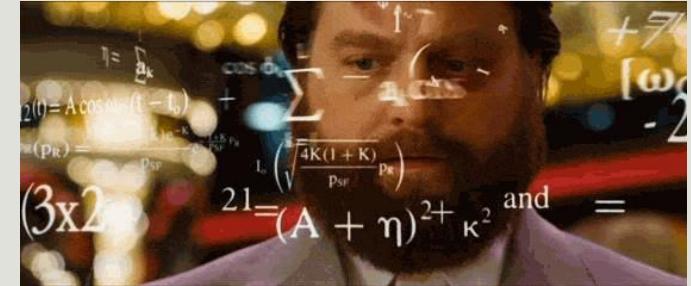


`changeDetector.reattach();`



View and model can
get out of sync!

Exercice



- Clone ce projet :
<https://github.com/aymensellaouti/ExempleNgOptimisation>
- Lancez votre projet, c'est le RhComponent qui est exécuté.
- Analysez ses problèmes et essayez de les résoudre avec les techniques étudiées.

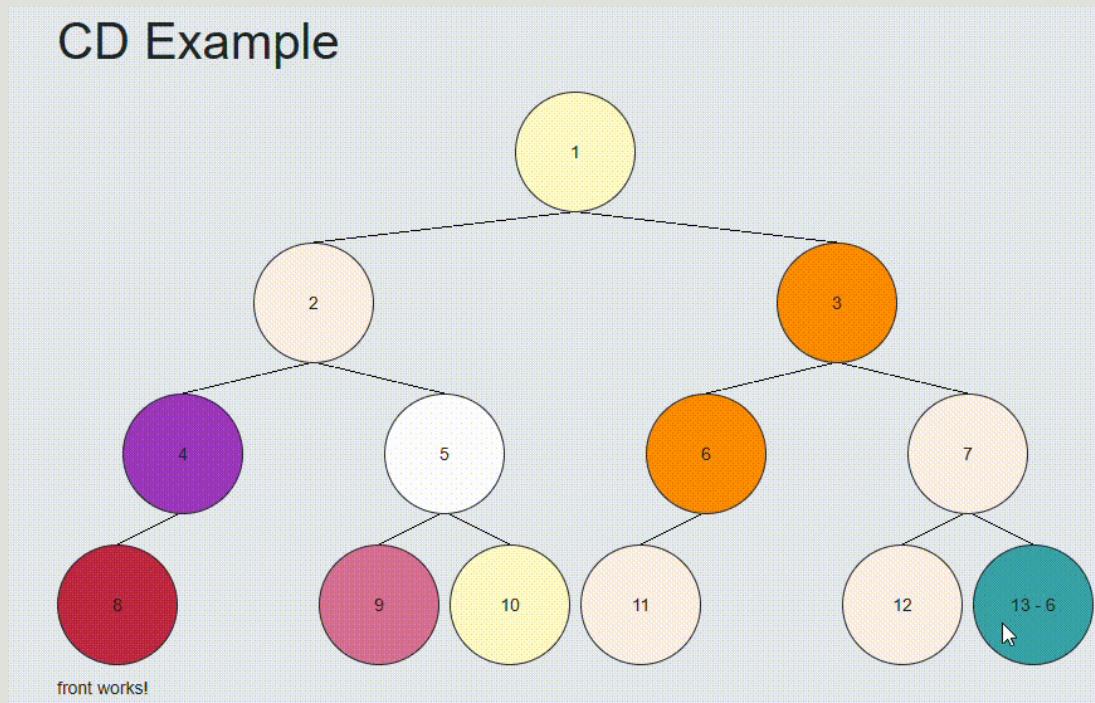
Notes

1. Zone Pollution Pattern
2. Out of Bound Change Detection
3. OnPush Stratégie
4. Recalculation of referentially transparent expressions

Change Detection

La stratégie par défaut

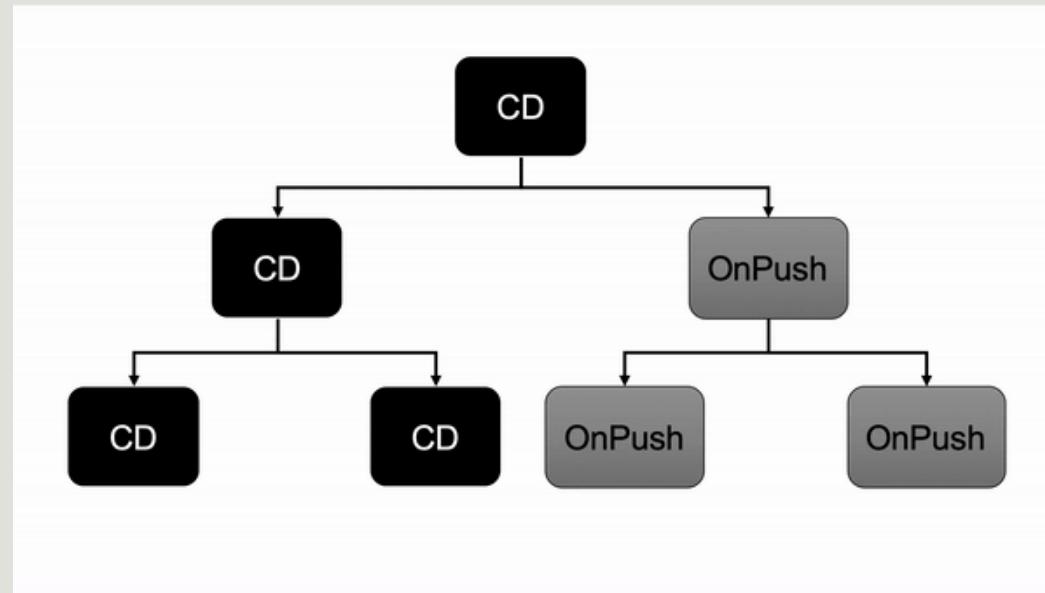
- Prenons le dossier Change Detection et testons un exemple standard avec que du Default Change Detection Startegy



Change Detection

La stratégie OnPush

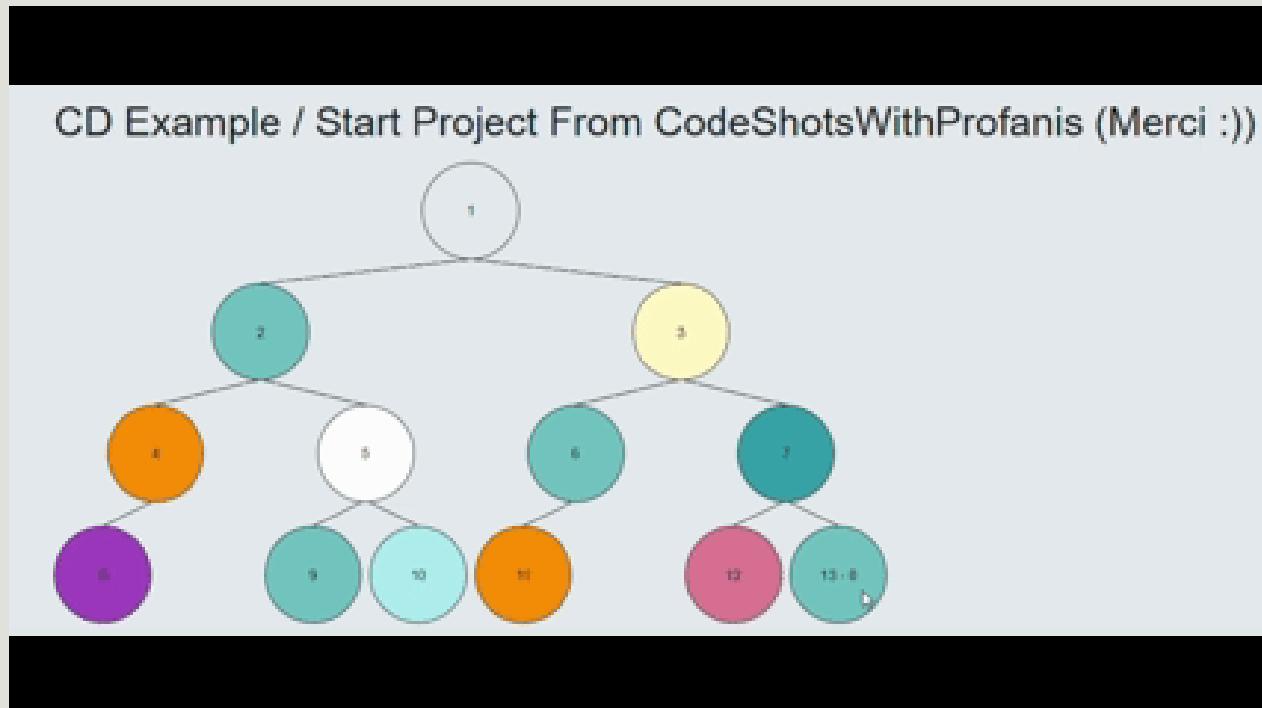
- La deuxième stratégie est la OnPush. En utilisant cette stratégie, Angular sait que le **composant n'a besoin d'être mis à jour que si :**
 - La **référence** d'entrée d'un **Input** du composant est **modifiée**
 - Le **composant ou l'un de ses enfants** déclenche un **événement**.
 - La **Change Detection** est déclenchée **manuellement**
 - Un **observable** lié au Template via le pipe **async** émet une **nouvelle valeur**.
 - **La nouveauté : Un Signal change**



Change Detection

La stratégie OnPush

- Mettons tous les composants en OnPush avec un AsyncPipe et un BehaviourSubject qui change au clic



Les signaux et le Change Detection

ng17 Local Change Detection

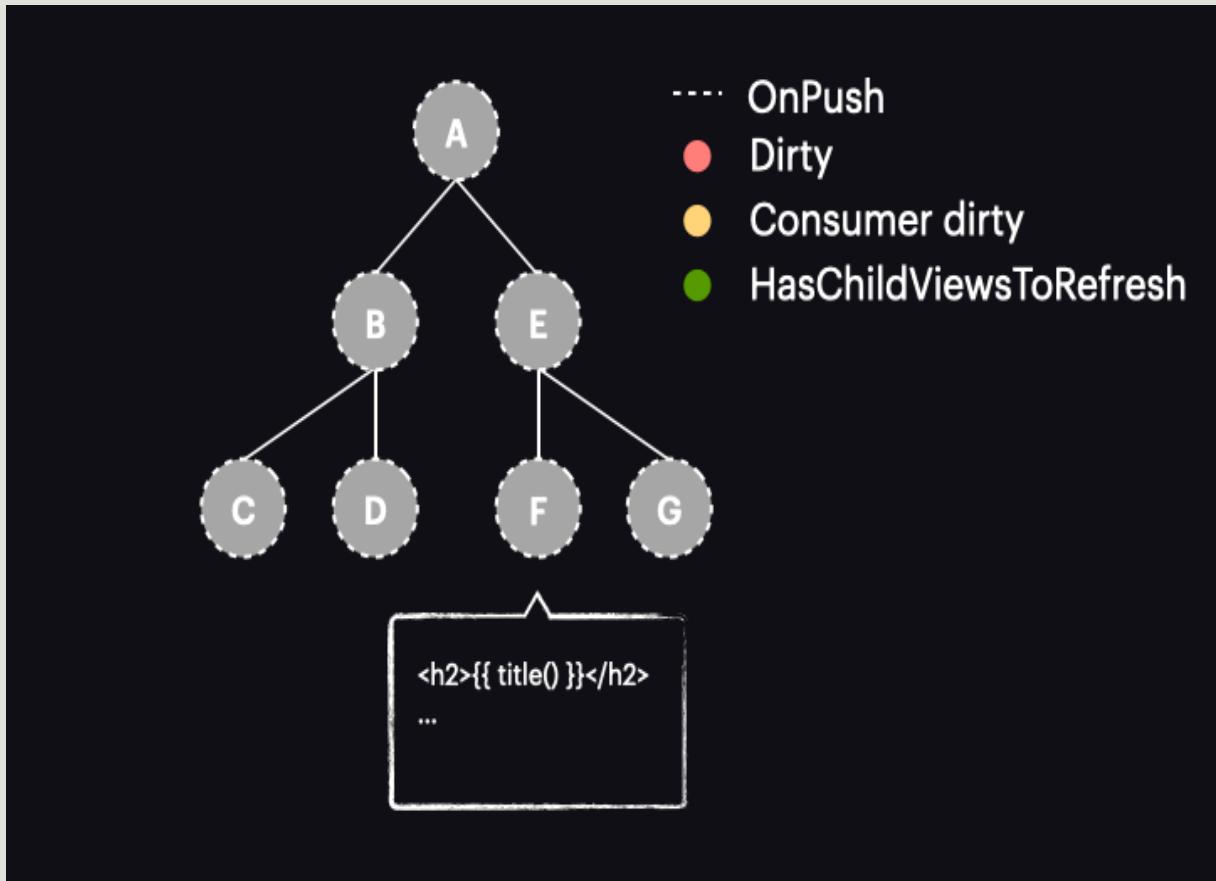
- Avec l'introduction des signaux et à partir de la version 17 d'Angular, quand un signal est déclenché, **on ne marque plus toute l'arborescence comme dirty**.
- L'idée est de pouvoir **atteindre le composant dirty, sans pour autant rafraîchir ou déclencher le CD dans toute la hiérarchie si on n'en a pas besoin**.
- La première étape consiste à **ne plus marquer le composant comme dirty** mais plutôt le marquer comme un **refreshView (consumer dirty)**.
- Un nouvel élément a vu le jour. C'est la fonction **markAncestorsForTraversal**.
- Cette fonction remonte du composant en passant par ses ancêtres jusqu'au composant racine (comme le faisait markViewDirty), mais **au lieu de les marquer comme dirty**, elle laisse le composant actuel tel quel (puisque le consommateur réactif est déjà marqué comme dirty). Ses ancêtres, en revanche, reçoivent un nouvel indicateur **HasChildViewsToRefresh**.

Les signaux et le Change Detection

ng17 Local Change Detection

OnPush + Signal

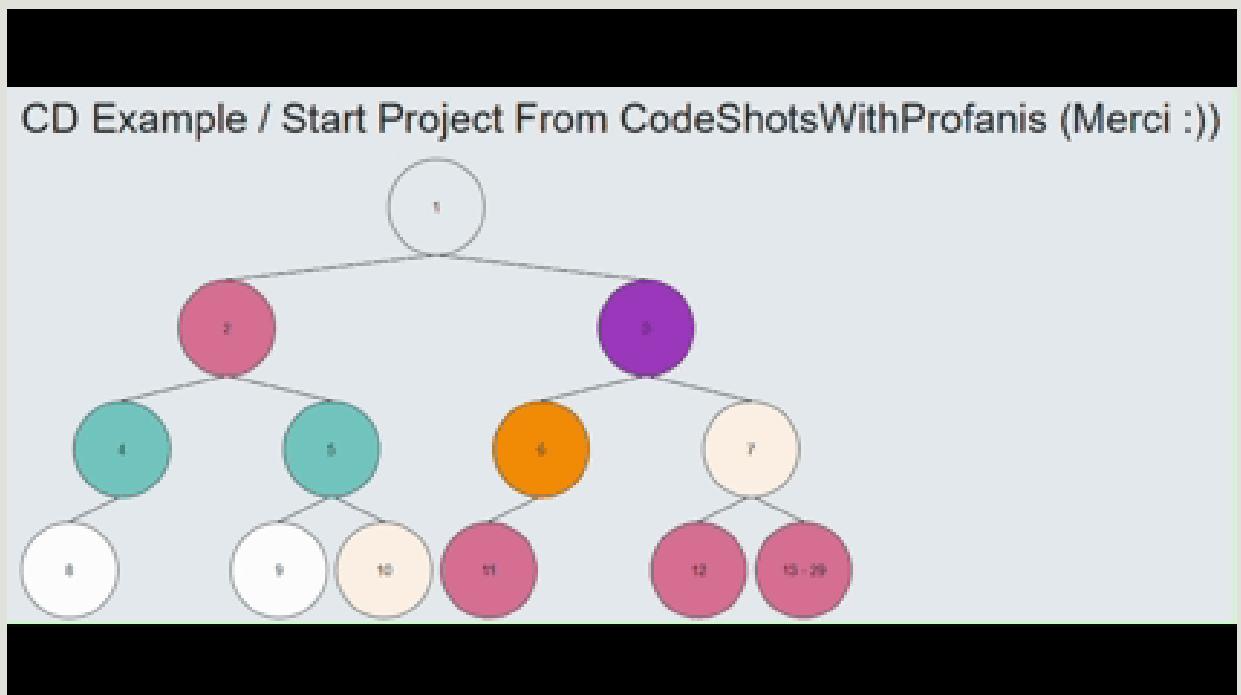
- Le mécanisme de détection des modifications a également été mis à jour.
- Désormais, lorsque le processus démarre avec l'arbre dans cet état, il **passe par les composants A et E sans effectuer de détection de modifications.**
- En effet, ils sont **OnPush**, mais **non dirty**s. Grâce au nouvel indicateur **HasChildViewsToRefresh**, **Angular continue de parcourir les nœuds marqués avec cet indicateur et de rechercher le composant nécessitant une détection de modifications** (dans notre exemple, il s'agit de celui dont le consommateur réactif est dirty).
- Lorsqu'il **atteint le composant F**, il constate que son consommateur réactif est dirty ; **ce composant est donc détecté comme modifié – et c'est le seul !**



Change Detection

Local Change Detection

- Mettons tous les composants en OnPush
- Dans le composant 13 créons un signal et ensuite un setInterval qui incrémente ce signal.



Les signaux et le Change Detection

ng17 Local Change Detection

Attention

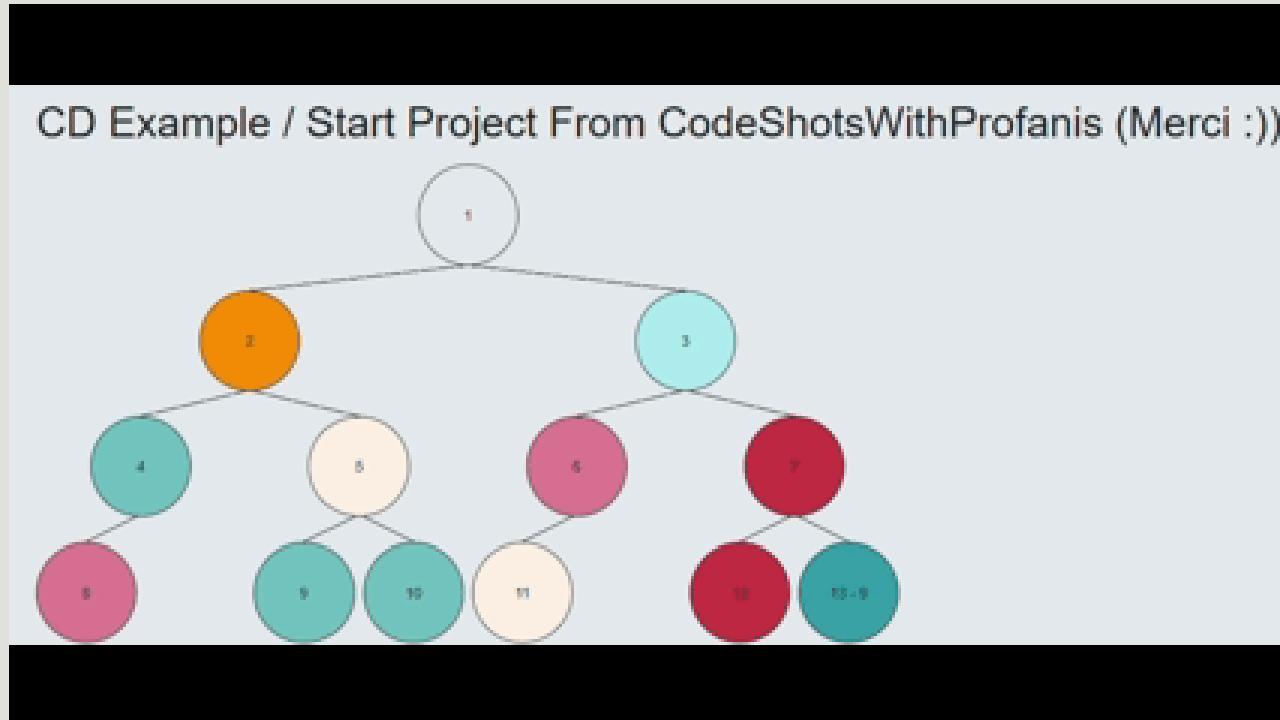
- **Les anciennes règles de détection des changements sont toujours en vigueur.** Donc si vous utilisez l'une des règles qui marque le composant OnPush à Dirty ceci impliquera aussi que ses ancêtres seront marqués comme Dirty. Ceci s'explique par le fait que **markAncestorsForTraversal est appelé, mais markViewDirty l'est également.**
- Cela nous amène à la **conclusion que la source du changement de la valeur du signal est importante. Si elle provient d'un élément qui marque le composant comme dirty, elle n'offre aucun avantage par rapport à la stratégie OnPush standard utilisée**, par exemple, avec le pipe asynchrone.
- **Pour bénéficier de la détection des changements semi-locale, le déclencheur doit provenir d'une action qui ne marque pas le composant comme dirty**, mais qui planifie néanmoins la détection des changements pour nous.
- Exemples : setInterval, setTimeout Observable.subscribe ...

Change Detection

Local Change Detection

Attention

- Mettons tous les composants en OnPush
- Dans le composant 13 créons un signal et suite à un click on incrémenté le signal.

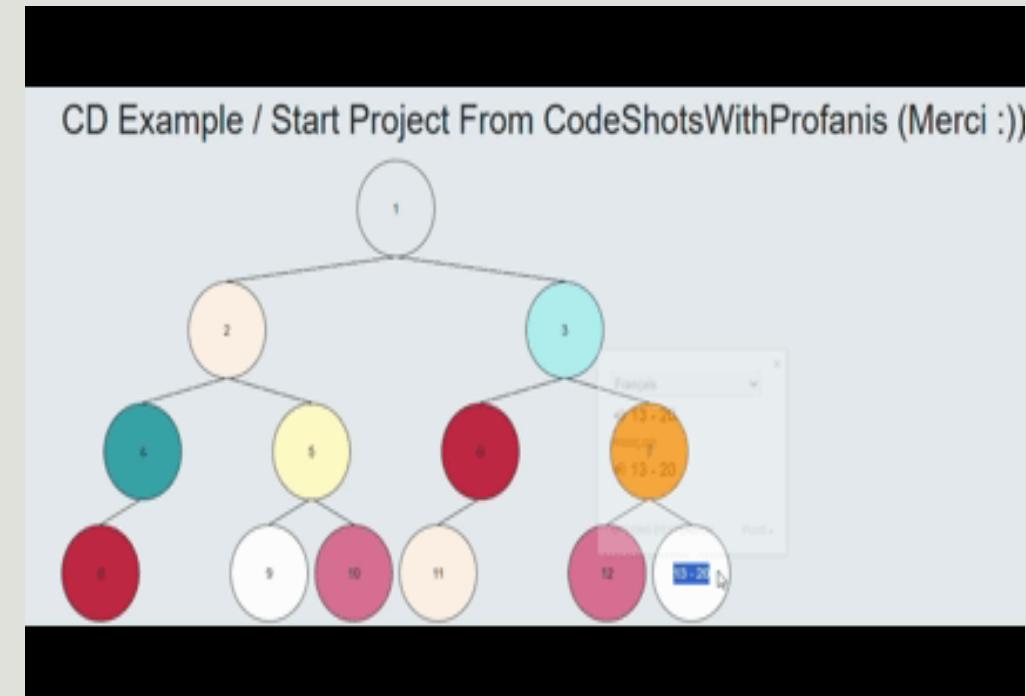


Change Detection

Local Change Detection

Attention / Solution

- Mettons tous les composants en OnPush
- Dans le composant 13 créons un signal et suite à un click on incrémente le signal.
- Afin d'éviter que ça se passe on peut **déclencher un event sans passer par un event binding** qui est pris en charge par le OnPush Change Détection.
- Vous pouvez utiliser l'opérateur **fromEvent** de RxJs.

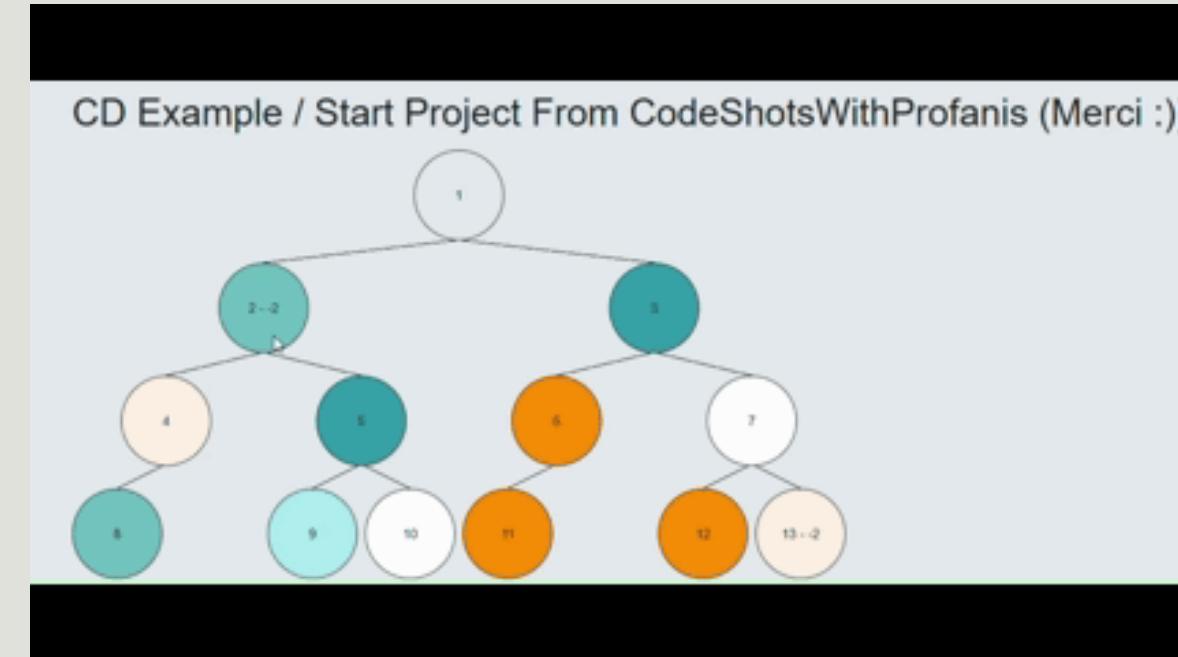


Change Detection

Local Change Detection

Partage d'état (RxJs) dans un service

- Mettons tous les composants en OnPush
- Partageons un Behaviour Subject dans un service et utilisons le dans Comp 2 et 13.

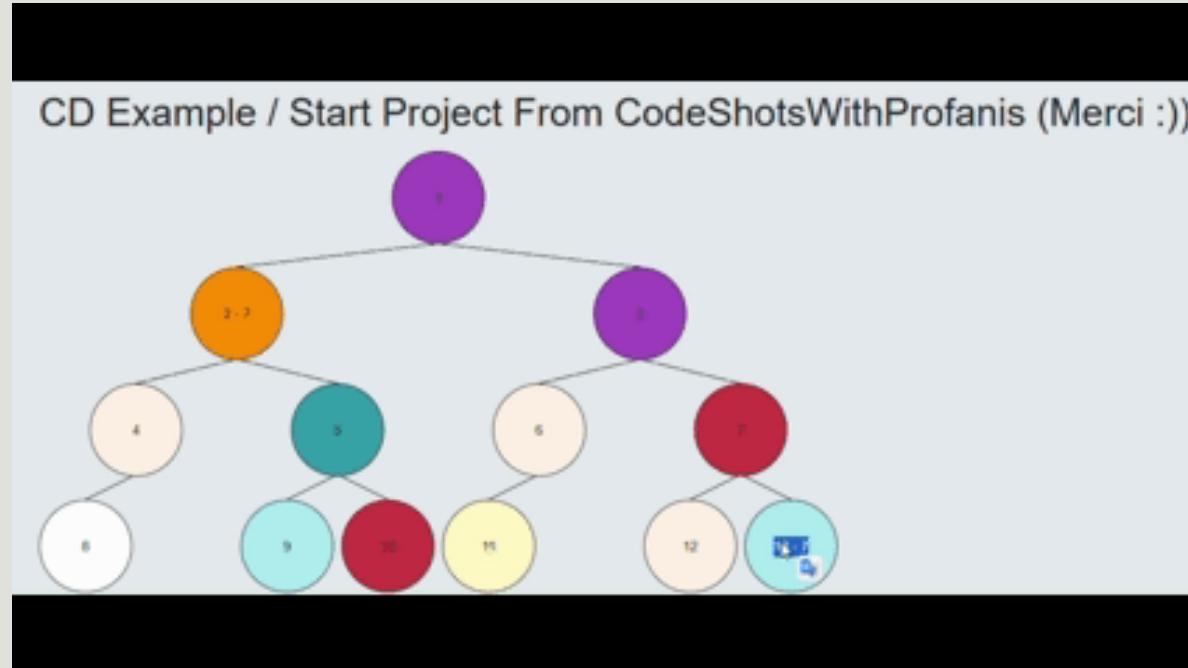


Change Detection

Local Change Detection

Partage d'état (Signal) dans un service

- Mettons tous les composants en OnPush
- Partageons un signal maintenant dans un service et utilisons le dans Comp 2 et 13.



Les signaux et le Change Detection ng18 Zoneless CD

- A partir d'Angular 18, nous commençons à parler de **Zoneless CD**.
- Elle est devenue stable dans la version 20.2
- L'idée est la suivante, on doit trouver un moyen de nous **débarrasser de ZoneJs**.
- Il faut ensuite **lui trouver un remplaçant**, sans lui qui **va déclencher le CD**, c'est là où intervient le **ChangeDetectionScheduler**
- Même si ZoneJs présente beaucoup d'avantages, il présente certaines lacunes :
 - La taille du bundle final c'est **10KB à 30KB qui seront gagné**
 - **ZoneJs vous dit simplement qu'il y a la possibilité que l'état ait changé**
 - **Difficile à débugger**
 - Ne plus avoir à **exécuter ZoneJs lors du chargement de l'application**

Les signaux et le Change Detection ng18 Zoneless CD

- La nouvelle logique a donc changé, plutôt que de déclencher la détection de changement « **lorsqu'une opération vient de se produire et que quelque chose a pu changer** », le framework la déclenche désormais « **lorsqu'il reçoit une notification indiquant que les données ont changées** ».
- Pour ce faire, un nouveau **Schedular a été introduit**.
- Une nouvelle méthode **notify** est aussi introduite permettant de spécifier **que l'état a changé**.
- La fonction **notify()** du **ChangeDetectionScheduler** est déclenchée explicitement par des actions spécifiques qui signalent **qu'un changement dans l'état de l'application nécessite une mise à jour du DOM**.
- Elle repose sur des **déclencheurs explicites**.

Les signaux et le Change Detection ng18 Zoneless CD

- La fonction `notify` est déclenchée quand :
 - Un **signal lu dans le template reçoit une nouvelle valeur**
 - Lorsqu'un **composant** est marqué comme **Dirty**
 - Via la fonction **markViewDirty**.
 - Une nouvelle valeur reçue par un **AsyncPipe**
 - Un **binding sur un évènement** dans le template
 - Un appel à **ComponentRef.setInput**
 - Un appel explicite à **ChangeDetectorRef.markForCheck**, entre autres.

Les signaux et le Change Detection

ng18 Zoneless CD

Activer le mode Zoneless

- Vous pouvez activer le mode Zoneless de deux méthodes selon si vous êtes dans une application modulaire ou standalone.
- N'oubliez pas d'enlevez ZoneJS de votre application.

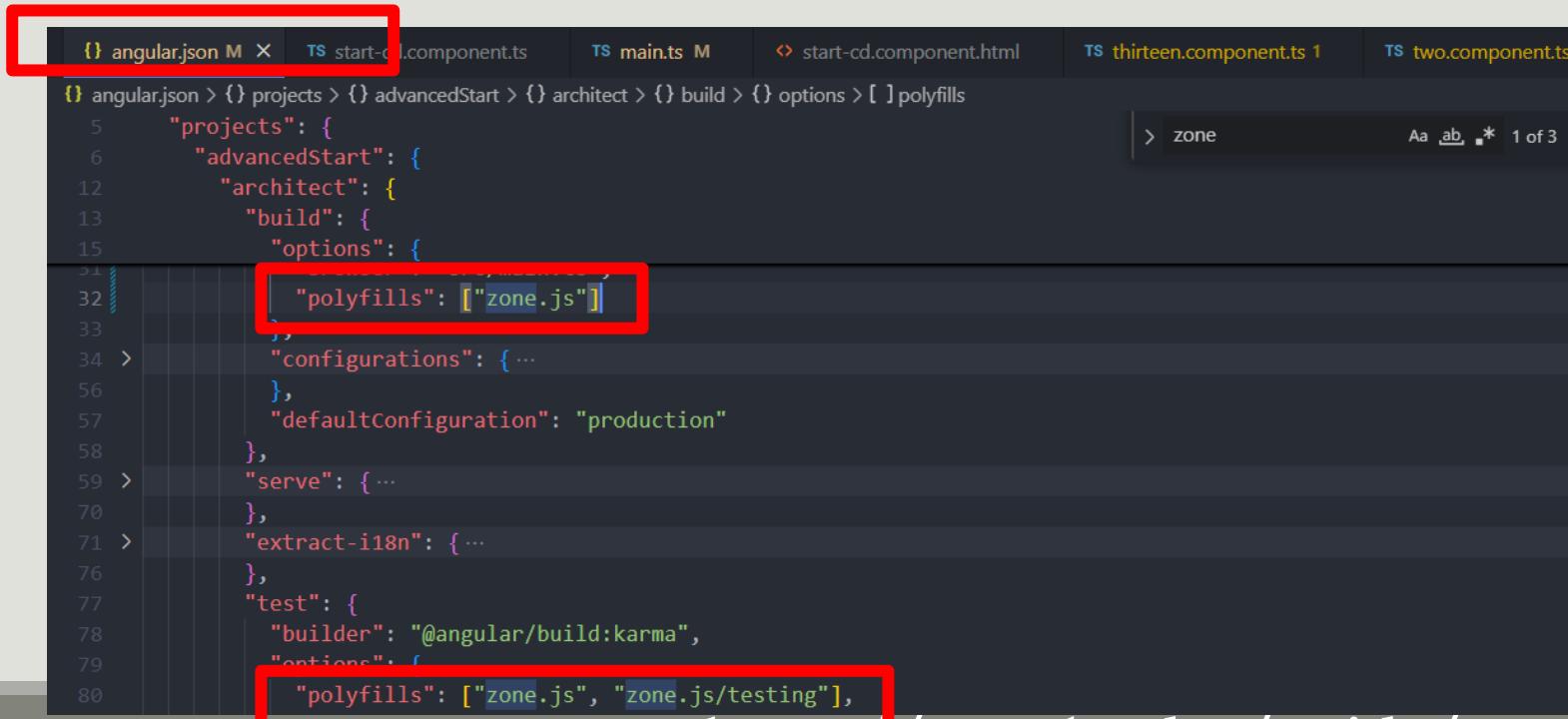
```
// standalone bootstrap
bootstrapApplication(MyApp, {providers: [
  provideZonelessChangeDetection(),
]});
// NgModule bootstrap
platformBrowser().bootstrapModule(AppModule);
@NgModule({
  providers: [provideZonelessChangeDetection()]
})
export class AppModule {}
```

Les signaux et le Change Detection

ng18 Zoneless CD

Activer le mode Zoneless / Désinstaller ZoneJs

- Désinstallez-le de vos dépendances et de votre fichier angular.json
- Supprimer les appels des polyfills dans angular.json



```
angular.json
{
  "projects": {
    "advancedStart": {
      "architect": {
        "build": {
          "options": {
            "polyfills": ["zone.js"]
          }
        }
      }
    }
  }
}
```

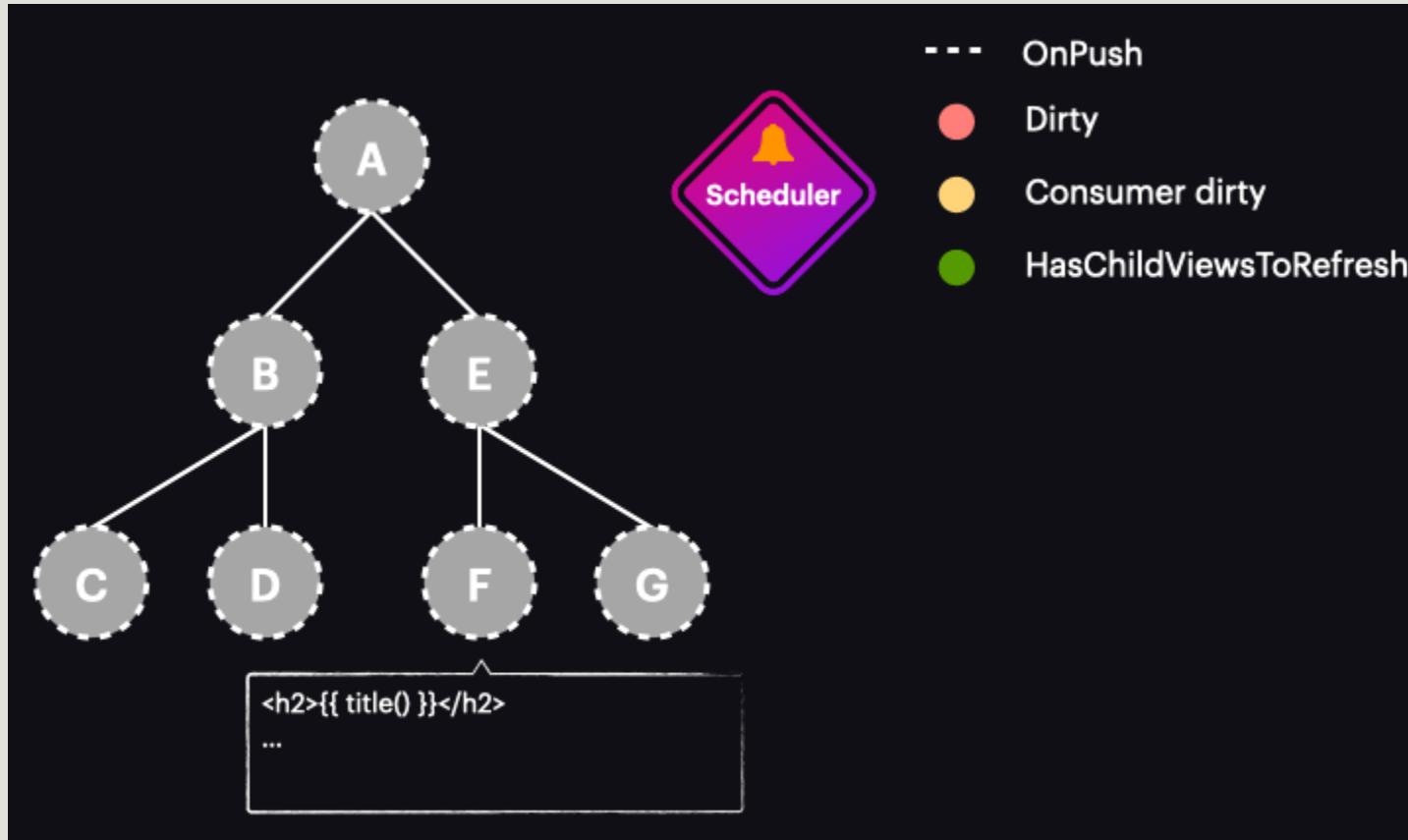
Avec ZoneJs

```
< class ZoneImpl {
  static __symbol__ = __symbol__;
  static assertZonePatched() {
    if (global["Promise"] !==
patches["ZoneAwarePromise"]) {
      throw new Error("Zone.js has detected that ZoneA...
    }
  }
}
```

Sans ZoneJs

```
Angular is running in development mode.
Attempting initialization Fri Jul 25 2025 15:18:50 GMT+0100 (UTC+01:00)
> Zone
✖ Uncaught ReferenceError: Zone is not defined at <anonymous>:1:1
>
```

Les signaux et le Change Detection ng18 Zoneless CD



Les signaux et le Change Detection

ng18 Zoneless CD

Quand appelle-t-on notify ?

- **Mise à jour d'un signal** : Lorsqu'un **signal** est **modifié via signal.set()**, **signal.update()**, Angular appelle automatiquement **notify()** pour planifier un cycle de change detection.

```
counter = signal(0);
increment() {
  this.counter.set(this.counter() + 1); // Déclenche notify()
}
```

Les signaux et le Change Detection

ng18 Zoneless CD

Quand appelle-t-on notify ?

- **Événements DOM dans les templates** : Les événements liés dans les templates, comme (click), (input), ou autres **host listeners**, déclenchent notify() lorsque l'événement est déclenché.
- **Utilisation de l'AsyncPipe** : Lorsqu'un observable ou une promesse émet une nouvelle valeur via un **AsyncPipe** dans le template, notify() est déclenché pour refléter la mise à jour.

```
<button (click)="handleClick()">Click</button>
```

```
<div>{{ data$ | async }}</div>
```

Les signaux et le Change Detection

ng18 Zoneless CD

Quand appelle-t-on notify ?

- **Appel explicite à `ChangeDetectorRef.markForCheck()`** : Lorsqu'un composant utilise `ChangeDetectionStrategy.OnPush` ou dans des cas où **un changement asynchrone n'est pas couvert par les signaux**, appeler `markForCheck()` déclenche `notify()` pour marquer le composant pour vérification.
- **Mise à jour des inputs via `ComponentRef.setInput`** : Lorsqu'un input d'un composant dynamique est mis à jour avec `ComponentRef.setInput`, `notify()` est appelé pour planifier le change detection.

```
setTimeout(() => {
  this.data = 'Updated';
  this.cdr.markForCheck(); // Déclenche notify()
}, 1000);
```

```
componentRef.setInput('message', 'Nouveau message'); // Déclenche notify()
```

Les signaux et le Change Detection

ng18 Zoneless CD

Quand appelle-t-on notify ?

- **Attachement ou détachement de vues** : Des opérations comme la création, l'attachement, ou le détachement de vues (par exemple, via `ViewContainerRef.createComponent` ou `ViewContainerRef.detach`) peuvent déclencher `notify()` pour s'assurer que le DOM est synchronisé.

```
this.vcr.createComponent(MyComponent); // Déclenche notify()
```

Les signaux et le Change Detection

ng18 Zoneless CD

Cas où notify() n'est PAS déclenché

- Dans la zoneless change detection, les opérations **asynchrones comme setTimeout, setInterval, ou les requêtes HTTP** (fetch, XMLHttpRequest) **ne déclenchent pas automatiquement notify()**, contrairement à Zone.js. Vous **devez explicitement** signaler les changements via :
 - Une mise à jour de signal.
 - Un appel à markForCheck()
 - Une interaction via un événement DOM ou AsyncPipe.

```
setTimeout(() => {  
  this.data = 'Updated'; // Ne déclenche PAS notify()  
}, 1000);
```

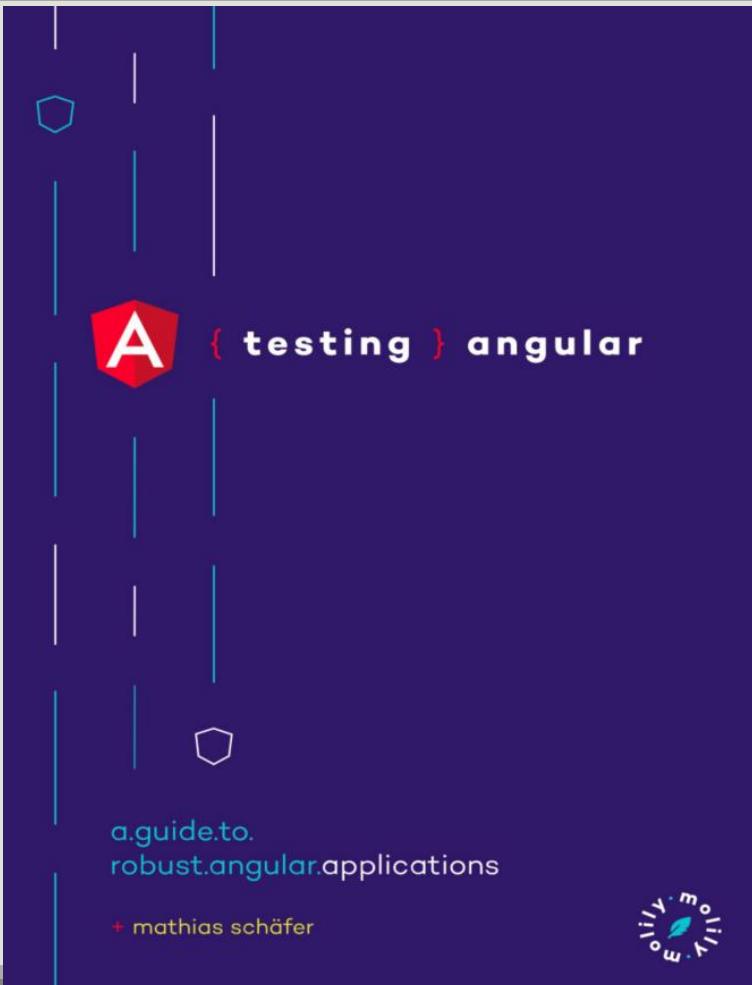
```
setTimeout(() => {  
  this.data = 'Updated';  
  this.cdr.markForCheck(); // Déclenche notify()  
}, 1000);
```

Angular Tests Unitaires et E2E

AYMEN SELLAOUTI



Références



Tests unitaires : Introduction

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

Tests unitaires : Pourquoi

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

Tests unitaires : Partout ?

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

Angular et les tests unitaires

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

Jasmin et Karma

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



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

Lancement d'un test

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

ng test

```
Karma v 6.4.2 - connected; test: complete;
Chrome 134.0.0.0 (Windows 10) is idle
Jasmine 4.6.0
Ran 4 of 10 specs - run all
finished in 0.046s
Incomplete: fit() or fdescribe() was found, 4 specs, 0 failures, randomized with seed 40279

ColorUtComponent
  • should create
MathService
  • should work
  • SPEC HAS NO EXPECTATIONS should add two numbers
Cv Service
  • should return all cvs
AuthService
  • should work
CounterComponent
  • CounterComponent exist
AppComponent
  • should have as title 'startingTest'
  • should render title
  • should create the app
Cv
  • should create an instance
```

Concepts de base de jasmin describe (suite)

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

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

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

Concepts de base de jasmin

Specification (**it**)

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

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

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

Concepts de base de jasmin

Specification (**it**)

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

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

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

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

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

Concepts de base de jasmin

Specification (**it**)

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

```
describe("Utils Service", () => {  
  it('should count distinct car', () => {  
    //Arrange  
    const utilsS = new UtilsService();  
  
    //Act  
    const result = utilsS.countDistinctCar('aymen sellaouti');  
  
    // Assert  
    const expectedResult = 11;  
    expect(result).toEqual(expectedResult);  
  });  
});
```

Concepts de base de jasmin

Attente (Expectation)

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

```
expect(actualValue).toBe(expectedValue);
```

Jasmin matchers

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

Jasmin matchers

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

Concepts de base de jasmin

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

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

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

Concepts de base de jasmin

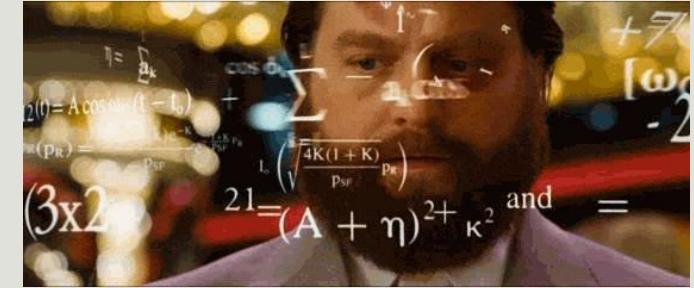
- **x**it permet d'exclure un test individuel

- **x**describe permet d'exclure tout le bloc

- **fit** permet de spécifier le test individuel à exécuter

- **f**describe permet de spécifier le bloc à exécuter.

Exercice



- Récupérer le Répo suivant :
<https://github.com/aymensellaouti/startinTest>
- Créer les tests nécessaires pour le service MathService

Concepts de base de jasmin

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

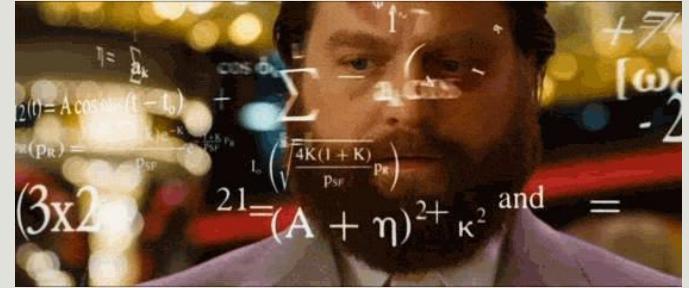
Concepts de base de jasmin

Jasmin offre des handlers permettant de répéter certaines fonctionnalités.

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

Exercice

- Mettez à jour vos tests.



Tests E2E

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

Tests E2E

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

Tests E2E

Comment ça marche ?

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

Tests E2E

Cypress

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

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

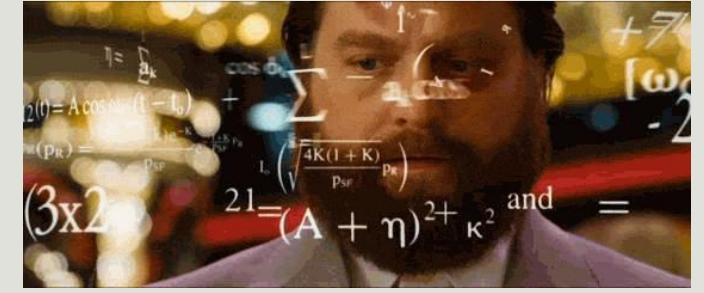
Tests E2E

Cypress

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

Exercice

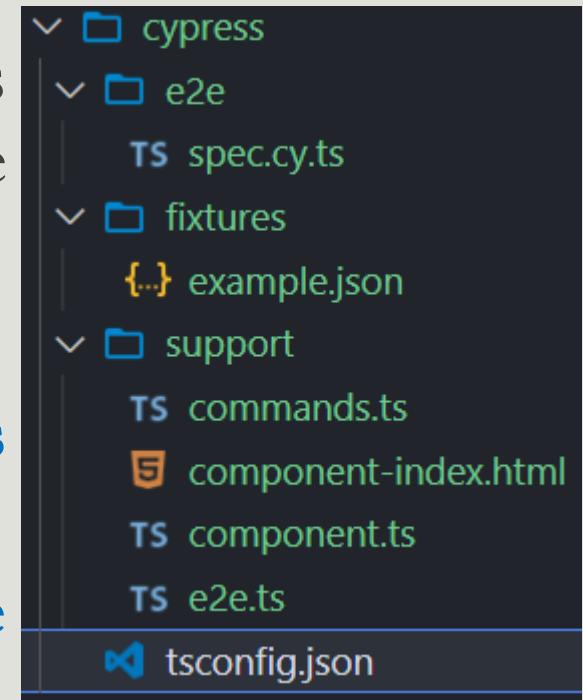
- Installez Cypress et regarder les différents fichiers ajoutés



Tests E2E

Le dossier cypress

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

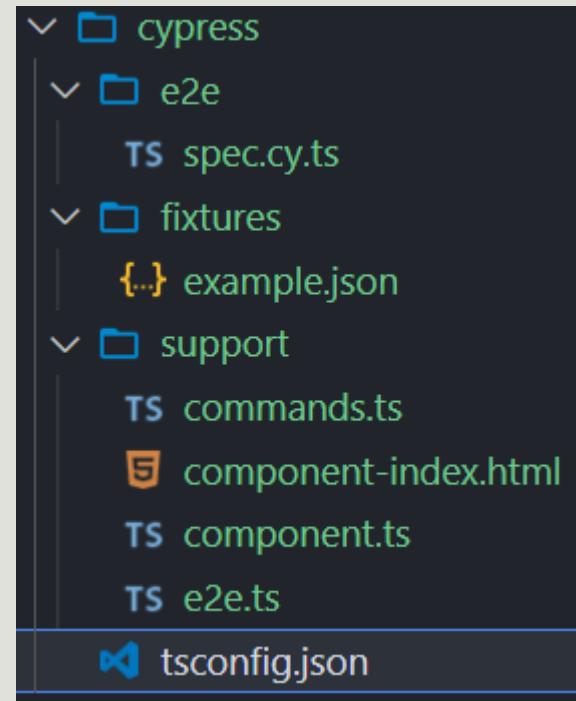


Tests E2E

Configuration

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

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



Tests E2E

Lancer les tests E2E

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

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

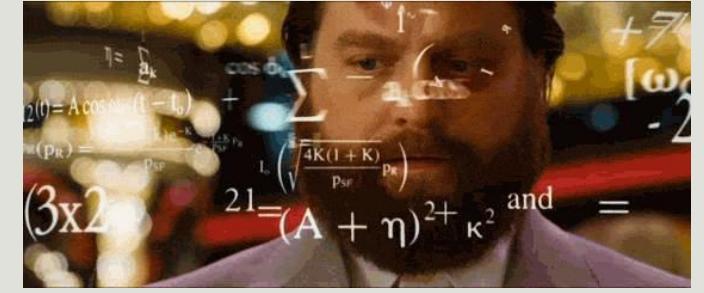
Tests E2E

Lancer les tests E2E

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

Exercice

- Lancez cypress en mode interactif et suivez les étapes



Tests E2E

Ecrire des tests E2E

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

Tests E2E

Visiter une page

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

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

Tests E2E

Sélectionner des éléments

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

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

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

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

Tests E2E

Sélectionner des éléments

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

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

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

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

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

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

Tests E2E

Sélectionner des éléments

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

Tests E2E

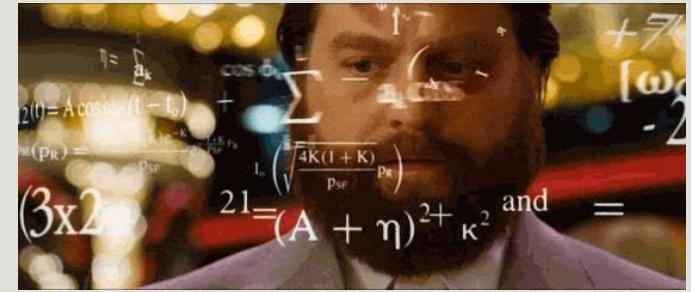
Sélectionner des éléments

Les bonnes pratiques

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

Exercice

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



Tests E2E

Test des requêtes HTTP

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

Tests E2E

Test des requêtes HTTP

API Réel

➤ **Avantages**

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

➤ **Inconvénients**

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

➤ **Utilisation suggérée**

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

Tests E2E

Test des requêtes HTTP

API Mockés

➤ Avantages

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

➤ Inconvénients

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

➤ Utilisation suggérée

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

Tests E2E

Test des requêtes HTTP API Mockés

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

Tests E2E

Test des requêtes HTTP

API Mockés

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

Tests E2E

Test des requêtes HTTP

API Mockés / intercept, mockez une réponse

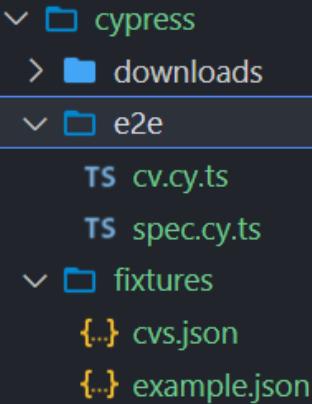
➤ Lorsque vous utilisez intercept suivez les étapes suivantes:

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

Tests E2E

Test des requêtes HTTP

API Mockés / intercept, mockez une réponse



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

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

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

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

Tests E2E

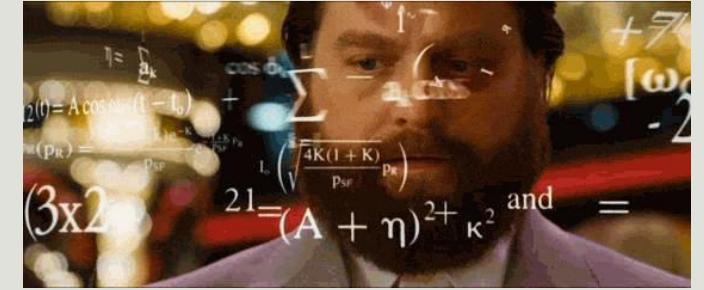
Test des requêtes HTTP

API Mockés / intercept, affecter un alias

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

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

Exercice



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

Tests E2E

Les assertions

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

Tests E2E

Les assertions implicites

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

Tests E2E

Les assertions

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

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

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

```
// Pour vérifier qu'un texte valide une expression régulière,
// préférer l'utilisation de contains
cy.get('.assertion-table')
  .find('tbody tr:last')
  // finds first element with text content matching regular
  // expression
  .contains('td', /column content/i)
  .should('be.visible')
```

Tests E2E

Les assertions have

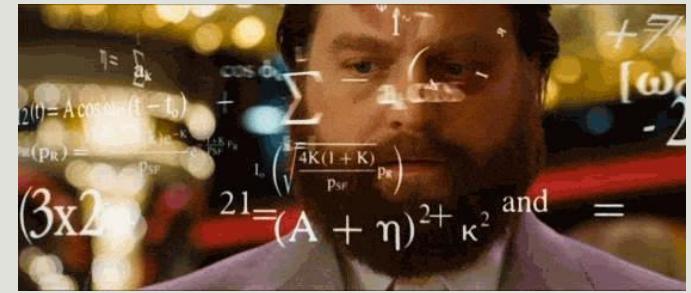
- **exist** : pour vérifier l'existence d'un élément
- **be.visible** : pour vérifier la visibilité d'un élément
- **be.enabled** : pour vérifier l'état activé/désactivé d'un élément
- **be.checked** : pour vérifier l'état coché/décoché d'un élément
- **have.value** : pour vérifier la valeur d'un élément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

Tests E2E

Les assertions

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

Exercice



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

Tests E2E

Location

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

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

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

Tests E2E

Déclencher des actions

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

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

Tests E2E

Déclencher des actions

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

Tests E2E

Déclencher des actions

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

```
// Special characters:
```

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

```
// Arrows:
```

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

```
// Modifier keys:
```

```
cy.get('input').type('{shift}')
cy.get('input').type('{ctrl}')
cy.get('input').type('{alt}')
```

Gestion des inscriptions

DestroyRef et takeUntilDestroy

- Avant Angular 16 Le processus de **nettoyage des inscriptions RxJs** était un problème courant, notamment lorsqu'il s'agissait d'implémenter un cycle de vie de composant/directive avec de longs abonnements RxJS.
- La méthode standard consistait à utiliser un **hook de cycle de vie OnDestroy** pour compléter tous les flux pertinents et exécuter d'autres actions de nettoyage via la **méthode unsubscribe d'une subscription** ou **l'opérateur takeUntil** et un **subject**. Cependant, cela entraînait du **boilerplate et du code passe-partout indésirable**.
- Une autre solution pour réaliser cette implémentation consistait à utiliser des **solutions tierces comme @ngneat/until-destroy**, mais contrairement aux solutions intégrées, elles peuvent **engendrer des problèmes de compatibilité et une intégration moins fluide**.

Gestion des inscriptions

DestroyRef et takeUntilDestroy

- A partir d'Angular 16 a introduit un **nouvel injectable** permettant **d'enregistrer les fonctions de rappel de nettoyage** qui seront exécutées à la fin du cycle de vie **sans passer par le hook OnDestroy**.
- C'est le **DestroyRef**. Il offre une méthode `onDestroy` qui prend en paramètre une méthode qui sera exécuté à la fin de vie du composant.

```
constructor() {  
  const subscription = interval(1000).subscribe(console.log);  
  this.destroyRef.onDestroy(() => subscription.unsubscribe());  
}
```

Gestion des inscriptions

DestroyRef et takeUntilDestroy

- L'introduction du **DestroyRef** a permis d'introduire un nouvel opérateur **takeUntilDestroy** RxJs qu'il utilise pour **désinscrire le flux automatiquement** à la mort du composant sans avoir à gérer les inscriptions ou passer par OnDestroy.

```
observable.pipe(takeUntilDestroyed())
```

Gestion des inscriptions

DestroyRef et takeUntilDestroy

```
constructor() {  
  const subscription = interval(1000).subscribe(console.log);  
  this.destroyRef.onDestroy(() => subscription.unsubscribe());  
}
```

- Supposons que nous voulons factoriser ce code de sorte de l'utiliser à chaque besoin afin de lancer des compteurs à la demande tout en gérant la désinscription

```
import { DestroyRef, inject } from "@angular/core";  
import { interval } from "rxjs";  
  
export const startCounting = () => {  
  const destroyRef = inject(DestroyRef);  
  const subscription = interval(1000).subscribe(console.log);  
  destroyRef.onDestroy(() => subscription.unsubscribe());  
}
```

```
constructor() {  
  startCounting();  
}
```

Gestion des inscriptions

DestroyRef et takeUntilDestroy

- Supposons que nous voulons déclencher le compteur au click sur un bouton.
- Est-ce que ce code fonctionne ?

```
count() {  
  startCounting();  
}
```

```
<button class="btn btn-success" (click)="count()"> Count </button>
```

Gestion des inscriptions

DestroyRef et takeUntilDestroy

- Cette fonction **ne marchera que si elle est dans un contexte d'injection**.
- **En dehors** la fonction **inject** utilisé dans startCounting **ne pourra pas injecter le DestroyRef** vu qu'elle n'a **pas accès à l'objet Injector**.

```
import { DestroyRef, inject } from "@angular/core";
import { interval } from "rxjs";

export const startCounting = () => {
  const destroyRef = inject(DestroyRef);
  const subscription = interval(1000).subscribe(console.log);
  destroyRef.onDestroy(() => subscription.unsubscribe());
}
```

Gestion des inscriptions

DestroyRef et takeUntilDestroy

- Le premier pattern pour résoudre ce problème est de **passer le DestroyRef comme paramètre pour la fonction** tout en le **conservant comme valeur par défaut.**

```
export const startCounting = (destroyRef:DestroyRef =
  inject(DestroyRef)) => {
  const subscription = interval(1000).subscribe(console.log);
  destroyRef.onDestroy(() => subscription.unsubscribe());
};
```

```
count() {
  // Dans un contexte d'injection
  startCounting();
}
```

```
count() {
  // En dehors d'un contexte d'injection
  startCounting(this.destroyRef);
}
```

Gestion des inscriptions

DestroyRef et takeUntilDestroy

- La deuxième solution est **d'appeler la fonction en spécifiant qu'elle est dans un contexte d'injection** via la fonction `runInInjectionContext` à qui en passer **en premier paramètre l'injecteur** qu'on récupère via le **token Injector** et en **deuxième paramètre une méthode dans laquelle on met ce qu'on veut appeler dans le contexte d'injection**, ici la méthode `startCounting`.

```
injector = inject(Injector);
count() {
    //V1
    //startCounting(this.destroyRef);
    //V2
    runInInjectionContext(this.injector, () => {
        startCounting();
    })
}
```

Gestion des inscriptions

DestroyRef et takeUntilDestroy

- Reprenons ensemble notre composant compteur et au lieu de juste loguer faisant en sorte qu'avec un setInterval on incrémente un signal.
- Ajouter ensuite un effect qui logue la valeur du compteur
- Tester le comportement en affichant puis en cachant le composant. Que pouvons nous conclure ?

Gestion des inscriptions

DestroyRef et takeUntilDestroy

- Faites en sorte maintenant de lancer l'effect à la demande suite à un clique sur un bouton.
- Est-ce que ca change quelque chose ?

L'API effect()

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

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

L'API effect()

- Vous pouvez aussi stopper l'effect manuellement en conservant la valeur de retour de effect qui est de type EffectRef et appeler sa méthode destroy.

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

N_X

AYMEN SELLAOUTI

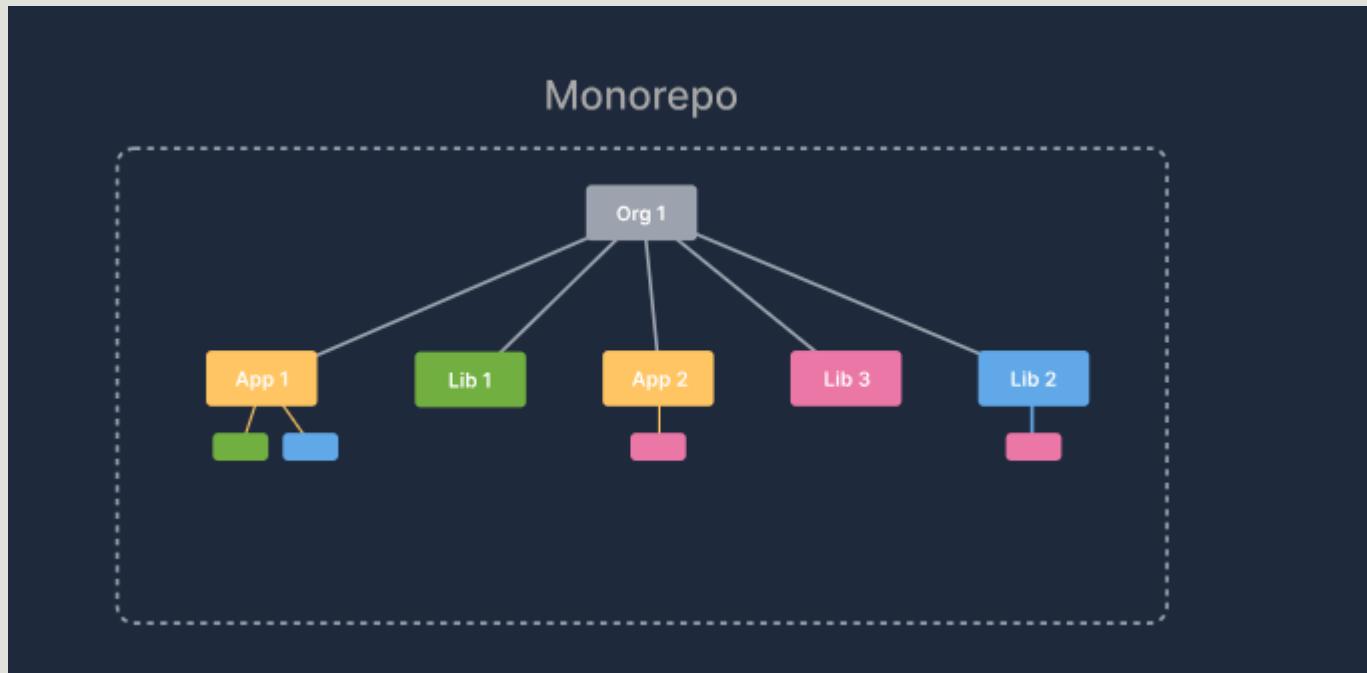


Objectifs

1. Définition d'un monorepo
2. Monorepo Vs Polyrepo
3. Introduction au cli Nx
4. Création d'un workspace
5. Architecture applicative

Définition d'un Monorepo

- Un monorepo est un **référentiel unique** contenant plusieurs projets distincts, **avec des relations bien définies**.

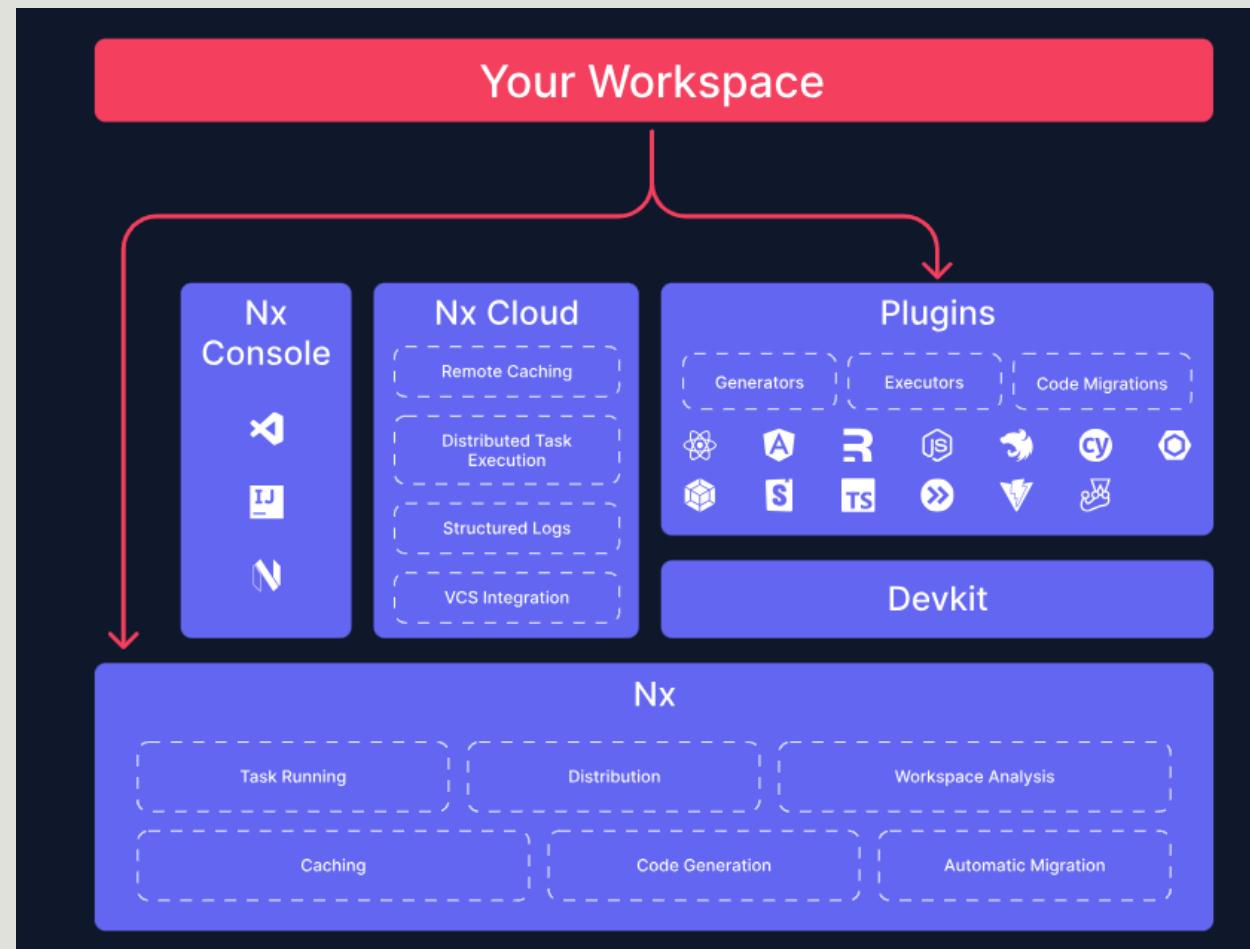


Monorepo Vs Polyrepo

- **Monorepo** et **polyrepo** sont deux options permettant de stocker du code dans une application de contrôle de source telle que GitHub.
- Si vous créez un **nouveau repository pour chaque projet** ou application, on parle alors de structure **polyrepo** ou **multi-repository**.
- En revanche, lorsque **toutes les applications se trouvent dans un seul grand repository**, on parle alors de structure **monorepo**.
- Pour les grandes entreprises, un monorepo peut facilement contenir des centaines d'applications (GOOGLE).

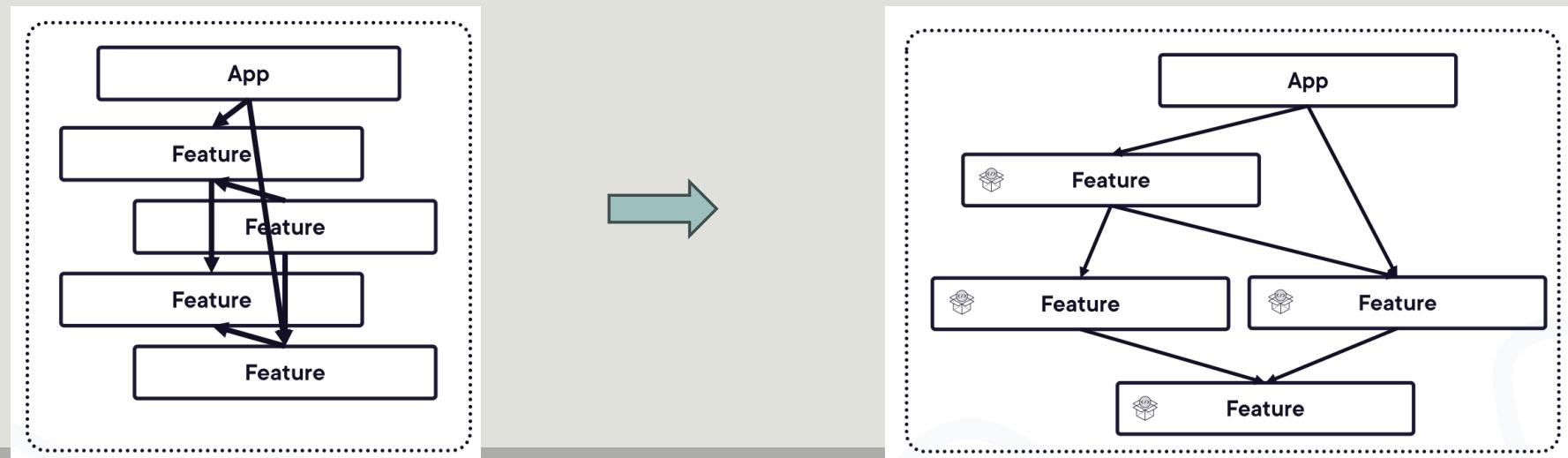
Nx

- Nx est un système de build open source qui introduit des outils et des techniques qui améliore la productivité du développeur.



Pourquoi Nx

- Lorsque votre application grandit, **la gestion des dépendances devient de plus en plus complexe.**
- Nx vous permet de **décomposer votre application en un ensemble de bibliothèque** que vous pouvez **partager avec vos différentes applications.**
- Nx permet aussi, en gardant un **graphe de dépendance**, d'optimiser le **build le test le lint** de vos applications.
- En utilisant un **système de cache**, Nx **peut identifier si un élément a été modifié ou non** et donc **cibler quelle partie tester par exemple.**



Pourquoi Nx

- Vous pouvez utiliser Nx pour :
 - **accélérez les builds et les tests de votre projet existant** en utilisant des techniques de **cache et de parallélisation**
 - **Accélérer votre développement** en vous offrant une **cli** qui permet de **générer des projets, des applications, des librairies tout en se chargeant de la configuration et de la mise en place**
 - **Garantir la cohérence et la qualité du code** avec des générateurs personnalisés et des règles de lint.
 - **Mettre à jour vos frameworks et vos outils**

Introduction au cli Nx

- Afin d'utiliser Nx, vous pouvez passer par le cli offert par Nx.
- Vous pouvez l'installer globalement via : `npm i -g nx@latest`
- Deux options s'offrent à vous
 - Créer un projet Nx
 - Migrer une application existante vers un projet Nx.
- Pour créer le projet Nx utiliser la commande suivante :

`npx create-nx-workspace@latest nomDeVotreProjet`

`npm create nx-workspace@latest nomDeVotreProjet`
- Une fois lancée, vous aller avoir un ensemble de questions

Introduction au cli Nx

Créer un Workspace

- Pour créer le projet Nx, utiliser la commande suivante :

npx create-nx-workspace@latest nomDeVotreProjet

- Une fois lancée, vous allez avoir un ensemble de questions
- Ici vous pouvez choisir la technologie sur laquelle vous travaillez.

```
DELL@AymenSellouati MINGW64 /d/projects/nx projects
$ npx create-nx-workspace@latest myFirstNxProject

NX Let's create a new workspace [https://nx.dev/getting-started/intro]

? Which stack do you want to use? ... []
None: Configures a TypeScript/JavaScript monorepo.
React: Configures a React application with your framework of choice.
Vue: Configures a Vue application with your framework of choice.
Angular: Configures a Angular application with modern tooling.
Node: Configures a Node API application with your framework of choice.
```

Introduction au cli Nx

Créer un Workspace

- Si vous choisissez Angular, vous aurez deux choix :
 - **Integrated Monorepo** qui permet **d'avoir un Monorepo** avec une application par défaut **Angular** et un **Workspace prêt à l'emploi**
 - **Standalone (non Monorepo)** avec une **application Angular** qui sera gérée par Nx

```
DELL@AymenSellalouti MINGW64 /d/projects/nx projects
$ npx create-nx-workspace@latest myFirstNxProject

NX Let's create a new workspace [https://nx.dev/getting-started/intro]

✓ Which stack do you want to use? · angular
? Integrated monorepo, or standalone project? ... □
Integrated Monorepo: Nx creates a monorepo that contains multiple projects.
Standalone: Nx creates a single project and makes it fast.
```

Créer un Workspace Standalone

➤ Si vous choisissez **Angular Standalone**, vous aurez un projet Angular standard géré par Nx.

➤ Vous pouvez aussi dès le départ spécifier le choix avec l'option preset :

```
npx create-nx-workspace@latest nomApp --preset=angular-standalone
```

➤ Un ensemble de questions vous seront posé concernant Angular puis Nx.

➤ Le terme **Standalone** représente **un seul projet et pas un mono répo** et qui n'a rien à avoir avec la notion de Standalone d'Angular mais **c'est le vocabulaire de Nx**.

➤ Même si ce choix ne vous donne pas toute la puissance de Nx, il vous offre plusieurs avantages tels que :

➤ La **gestion du build, test et lint en parallèle et d'une manière optimisée via le cache**

➤ Facilite la **génération de bibliothèque** interne permettant un code modulaire

➤ Permet une **architecture plus robuste** avec des **frontières entre les différentes couches** avec des règles standards évitant les dépendances circulaires et des **règles personnalisées que vous pouvez définir**.

Créer un Workspace Standalone

```
DELL@AymenSellouati MINGW64 /d/projects/nx projects
$ npx create-nx-workspace@latest firstNxAngularStandaloneApplication --preset=angular-standalone

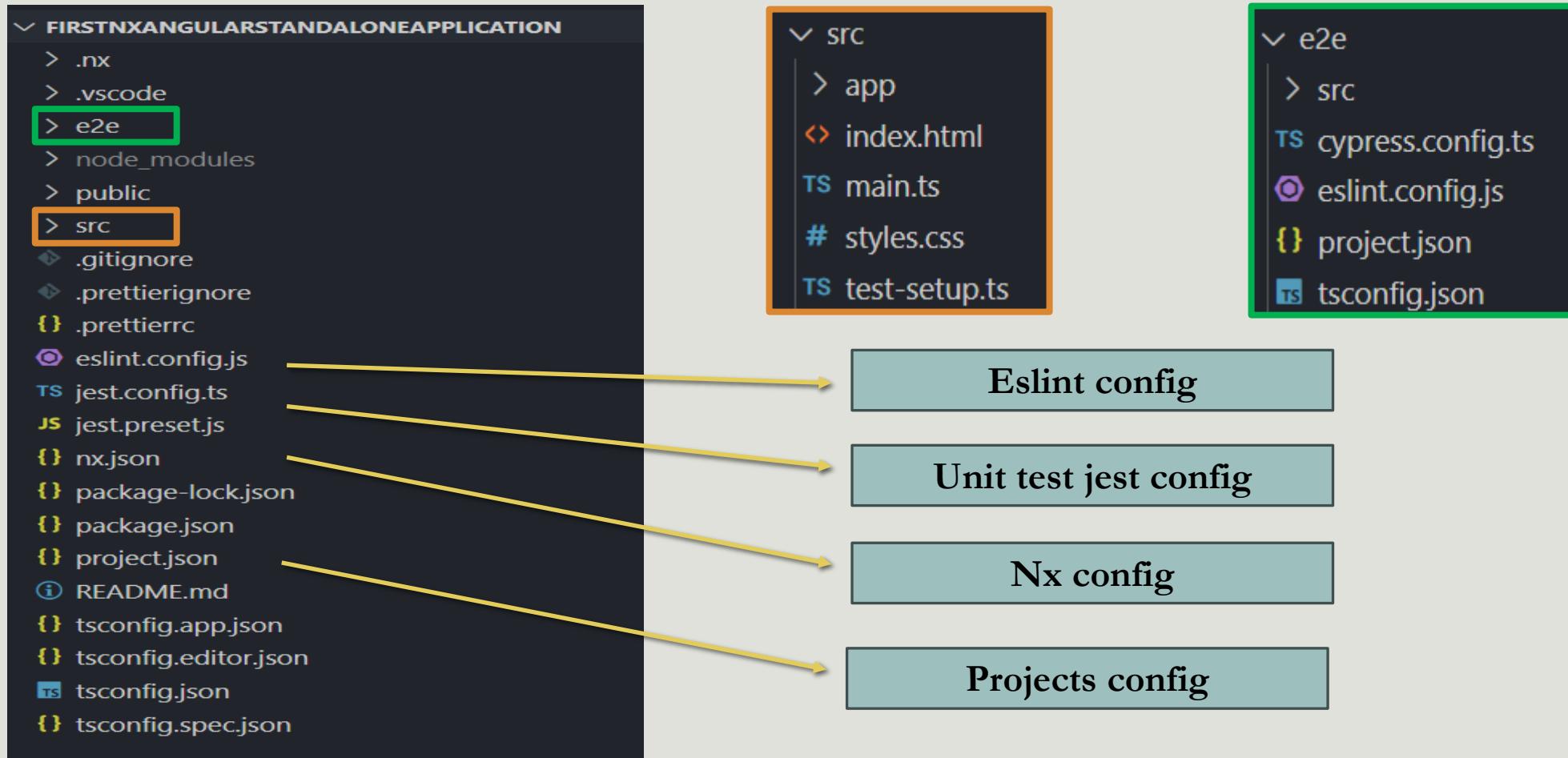
NX Let's create a new workspace [https://nx.dev/getting-started/intro]

✓ Which bundler would you like to use? · esbuild
✓ Default stylesheet format · css
✓ Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? · No
✓ Test runner to use for end to end (E2E) tests · cypress
✓ Which CI provider would you like to use? · skip
✓ Would you like remote caching to make your build faster? · yes
✓ Will you be using GitHub as your git hosting provider? · Yes

NX Creating your v20.1.0 workspace.

✓ Installing dependencies with npm
✓ Successfully created the workspace: firstNxAngularStandaloneApplication.
✓ Nx Cloud has been set up successfully
```

Créer un Workspace Standalone



Créer un Workspace Monorepo

- Si vous **choisissez Integrated-monorepo**, vous aurez un **monorepo avec un projet Angular géré par Nx**.
- Vous pouvez aussi dès le départ spécifier le choix avec l'option preset :

```
npx create-nx-workspace@latest nomApp --preset=angular-monorepo
```

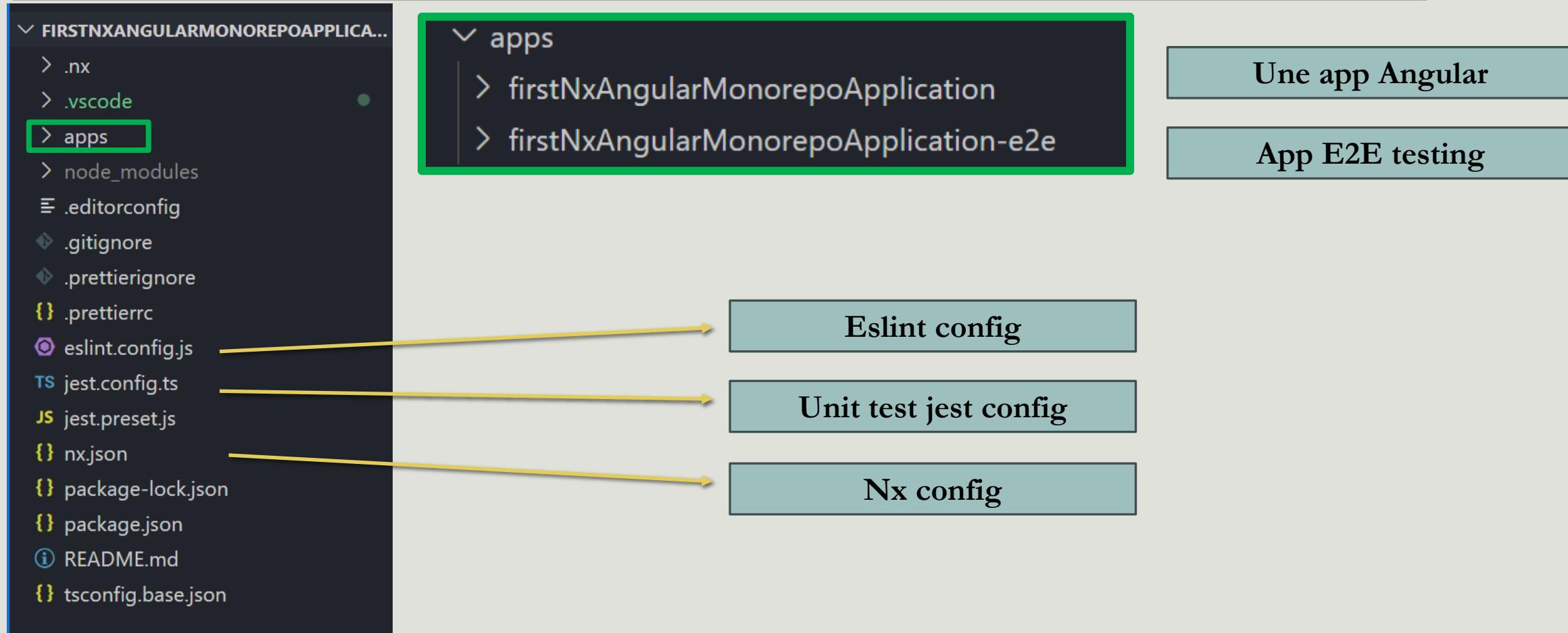
- Un ensemble de questions vous seront posé concernant Nx uniquement cette fois ci.

```
DELL@AymenSellouati MINGW64 /d/projects/nx projects
$ npx create-nx-workspace@latest firstNxAngularMonorepoApplication --preset=angular-monorepo
✓ Which CI provider would you like to use? · skip
✓ Would you like remote caching to make your build faster? · yes
✓ Will you be using GitHub as your git hosting provider? · Yes

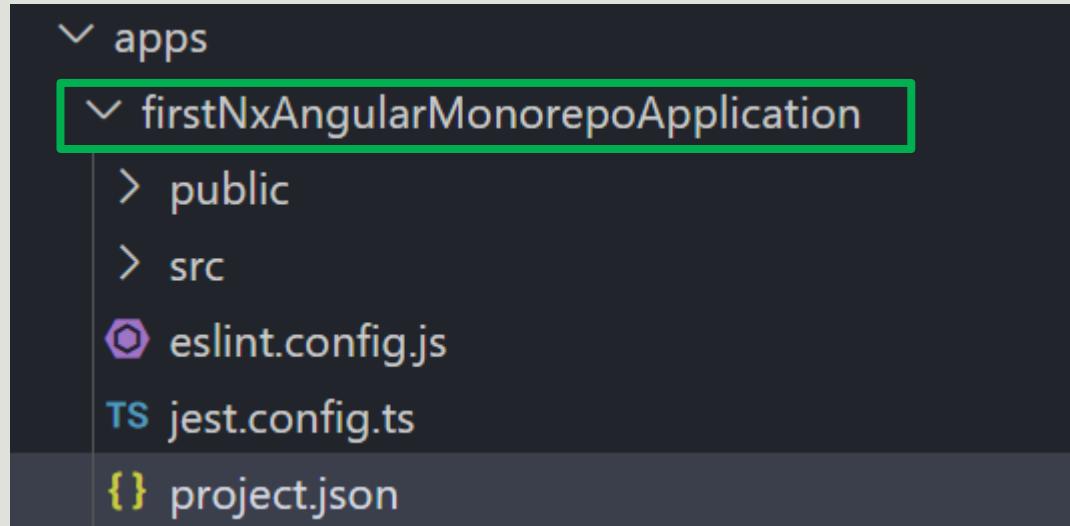
NX  Creating your v20.1.0 workspace.

✓ Installing dependencies with npm
✓ Successfully created the workspace: firstNxAngularMonorepoApplication.
✓ Nx Cloud has been set up successfully
```

Créer un Workspace Monorepo



Créer un Workspace Monorepo



A screenshot of a code editor showing the contents of the 'project.json' file. The file defines a single application target named 'firstNxAngularMonorepoApplication'. It includes configurations for linting, building, serving, extracting i18n, testing, and serving static files. Hovering over the 'name' field shows a tooltip: 'Nx Targets: lint, build, serve, extract-i18n, test, serve-s...'. The 'targets' section lists various targets with their descriptions:

```
{  
  "name": "firstNxAngularMonorepoApplication",  
  "$schema": ".../node_modules/nx/schemas/project-schema.json",  
  "projectType": "application",  
  "prefix": "app",  
  "sourceRoot": "apps/firstNxAngularMonorepoApplication/src",  
  "tags": [],  
  "targets": {  
    "build": { ... },  
    "serve": { ... },  
    "extract-i18n": { ... },  
    "lint": { ... },  
    "test": { ... },  
    "serve-static": { ... }  
  }  
}
```

Installer Nx dans un projet existant

- Si vous voulez installer nx sur un projet existant, exécuter la commande suivante :
- **nx init (par défaut vous aurez une standalone Nx Application)**
- Une fois la commande lancée, vous aurez un ensemble de questions concernant nx
 - La première concernant le cache, à savoir sur quelles commandes voulez-vous appliquer le cache
 - La deuxième concerne le remote caching offert par nx.

```
NX 🐛 Checking versions compatibility

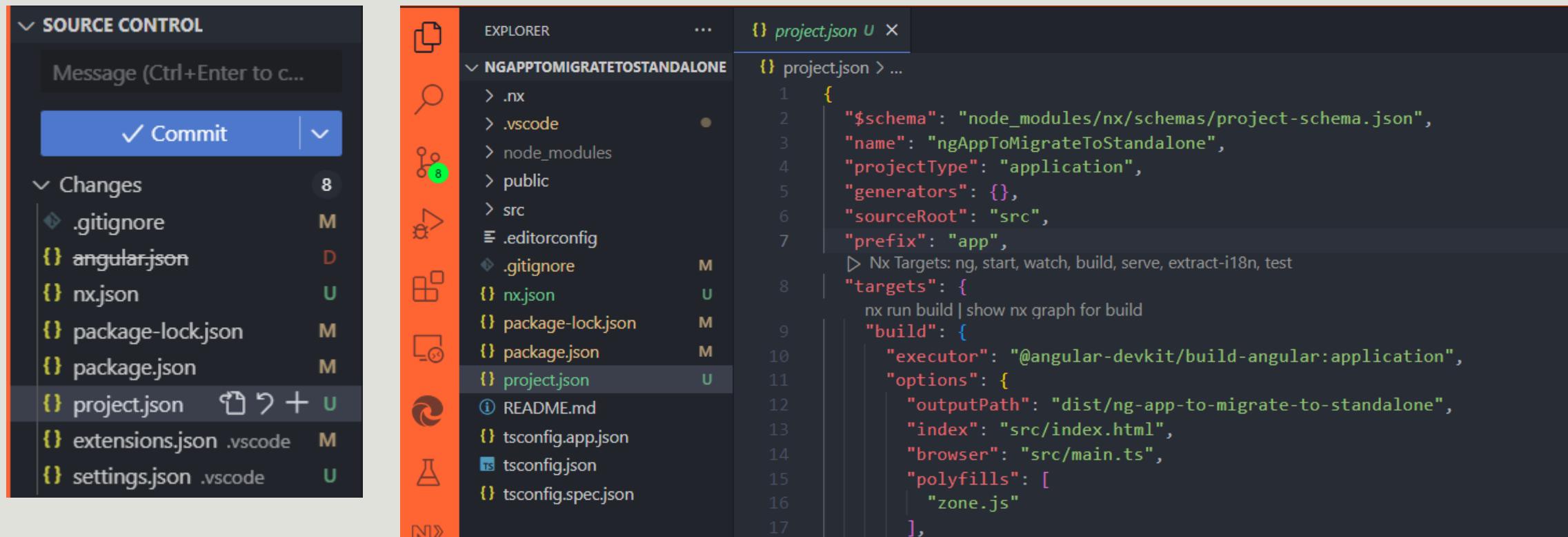
NX ✅ The Angular version is compatible with the latest version of Nx!

NX 🎯 Nx initialization

NX 🧑 Please answer the following questions about the targets found in your angular.json in order to generate task runner configuration
✓ Which of the following targets are cacheable? (Produce the same output given the same input, e.g. build, test and lint usually are, serve and start are not) · build, test
✓ Would you like remote caching to make your build faster? · yes

NX 📦 Installing dependencies
```

Installer Nx dans un projet existant Standalone



Installer Nx dans un projet existant

- Si vous ajoutez **l'option `-integrated`**, c'est un **monorepo** intégrant votre projet qui sera créé.
- Le dossier **src sera supprimé et remplacé par le dossier apps** contenant vos applications
- Un dossier **libs sera ajouté**
- Chaque application aura son project.json qui remplacera le angular.json habituel

Installer Nx dans un projet existant Standalone

The screenshot shows the VS Code interface with the Explorer, Editor, and Problems panes visible. The Explorer pane shows a monorepo structure with projects like 'ngAppToMigrateToMonorepo' and 'apps\ngAppToMigrateToMonorepo'. The Editor pane displays the contents of the 'project.json' file for the 'ngAppToMigrateToMonorepo' project. The file defines an application target with configurations for 'build' and 'production', and lists assets, styles, and scripts. The Problems pane on the right shows 38 errors, primarily related to missing files like 'tsconfig.base.json' and 'tsconfig.tools.json'.

```
1  {
2    "name": "ngAppToMigrateToMonorepo",
3    "$schema": "../../node_modules/nx/schemas/project-schema.json",
4    "projectType": "application",
5    "generators": {},
6    "sourceRoot": "apps/ngAppToMigrateToMonorepo/src",
7    "prefix": "app",
8    "targets": {
9      "nx run build | show nx graph for build"
10     "build": {
11       "executor": "@angular-devkit/build-angular:application",
12       "options": {
13         "outputPath": "dist/apps/ngAppToMigrateToMonorepo",
14         "index": "apps/ngAppToMigrateToMonorepo/src/index.html",
15         "browser": "apps/ngAppToMigrateToMonorepo/src/main.ts",
16         "polyfills": ["zone.js"],
17         "tsConfig": "apps/ngAppToMigrateToMonorepo/tsconfig.app.json",
18         "assets": [
19           {
20             "glob": "**/*",
21             "input": "apps/ngAppToMigrateToMonorepo/public"
22           }
23         ],
24         "styles": ["apps/ngAppToMigrateToMonorepo/src/styles.css"],
25         "scripts": []
26       },
27       "configurations": {
28         "nx run build:production"
29       },
30       "production": {
31         "budgets": [
32           {
33             "type": "initial",
34             "maximumWarning": "500kB",
35             "maximumError": "1MB"
36           }
37         ]
38       }
39     }
40   }
41 }
```

<https://nx.dev/recipes/angular/migration/angular>

Migrer d'un projet standalone à un projet monorepo

- Nx vous facilite aussi **la migration d'un projet standalone vers un projet monorepo**
- Vérifier que vous avez le plugin **@nx/workspace**, sinon installer le via la commande **nx add @nx/workspace**
- Lancer ensuite la commande **nx g convert-to-monorepo**
- Vérifier que tout est ok, des fois il y a certains fichiers qui ne sont pas bien gérés

Exécuter des tâches

- Nx **facilite** votre travail pour **l'exécution des tâches**
- Il va **parser les fichiers project.json** de **chaque projet** afin d'identifier le **nom** du projet et les **commandes** qu'il permet d'exécuter
- Vous pouvez aussi définir le projet par défaut dans le fichier nx.json avec la propriété defaultProject

```
{} nx.json > {} targetDefaults > {} @nx/eslint:lint > [ ] inputs
1  {
2    "$schema": "./node_modules/nx/schemas/nx-schema.json",
3    "defaultBase": "master",
4    "namedInputs": ...
5  ,
6    "defaultProject": "ngNxCvTechForma",
```

```
apps > cvViewer > {} project.json > ..
1  {
2    "name": "cvViewer",
3    "$schema": "../../node_modules/nx/schemas/project-schema.json",
4    "projectType": "application",
5    "prefix": "app",
6    "sourceRoot": "apps/cvViewer/src",
7    "tags": [],
8    "targets": {
9      "build": {
10        "nx run build | show nx graph for build
11      },
12      "serve": {
13        "nx run serve | show nx graph for serve
14      },
15      "extract-i18n": {
16        "nx run extract-i18n | show nx graph for extract-i18n
17      },
18      "lint": {
19        "nx run lint | show nx graph for lint
20      },
21      "test": {
22        "nx run test | show nx graph for test
23      },
24      "serve-static": {
25        "nx run serve-static | show nx graph for serve-static
26      }
27    }
28  }
```

Exécuter des tâches

- En se basant sur cette logique, vous pouvez exécuter la tâche que vous voulez avec le pattern de commande suivant :

`nx <targetName> < projectName > <options>`

- Exemple : nx **build cvViewer --watch**

- Si vous voulez exécuter une tâche sur le projet par défaut plus la peine de le spécifier

```
$ nx build cvViewer --watch

> nx run cvViewer:build:production --watch

> Building...
✓ Building...
Initial chunk files | Names | Raw size | Estimated transfer size
main-OTU53U2Q.js | main | 230.40 kB | 61.38 kB
polyfills-FFHMD2TL.js | polyfills | 34.52 kB | 11.28 kB
styles-5INURTSO.css | styles | 0 bytes | 0 bytes
| Initial total | 264.92 kB | 72.66 kB

Application bundle generation complete. [4.611 seconds]

Watch mode enabled. Watching for file changes...
Output location: D:\projects\nx projects\ngNx CvTechForma\dist\apps\cvViewer
```

```
$ nx build --watch

> nx run ngNx CvTechForma:build:production --watch

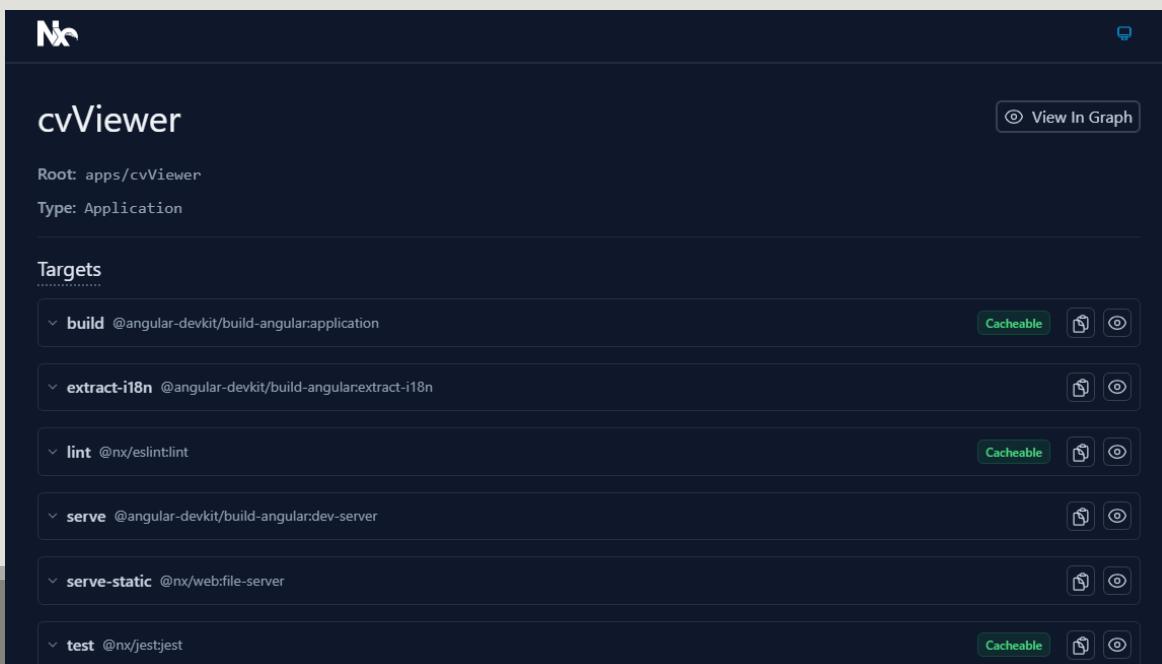
> Building...
✓ Building...
Initial chunk files | Names | Raw size | Estimated transfer size
chunk-ZBF6570U.js | - | 251.66 kB | 67.69 kB
styles-JQ6IFEZI.css | styles | 237.42 kB | 23.77 kB
polyfills-FFHMD2TL.js | polyfills | 34.52 kB | 11.28 kB
main-YW02JM20.js | main | 24.13 kB | 5.92 kB
| Initial total | 547.74 kB | 108.67 kB

Lazy chunk files | Names | Raw size | Estimated transfer size
chunk-Y2EDWBSB.js | index | 50.36 kB | 11.68 kB
```

Exécuter des tâches

- Vous pouvez aussi utiliser la syntaxe suivante :
- `nx run projectName:task:configuration options`
- `nx run ngNx CvTechForma:build:production --watch`
- Vous pouvez aussi **lister vos task en mode visuel** avec la commande

`nx show project projectName`



Exécuter des tâches

- Vous pouvez aussi **exécuter plusieurs tâches en parallèles** avec la syntaxe suivante :
- **nx run-many -t test lint e2e**
- Cette commande lancera les trois tâches, en parallèle, pour tous les projets, exécutant ainsi deux lint un test et le e2e.
- Si vous relancez une seconde fois la même commande vous aurez un message vous informant que le résultat est envoyé depuis le cache :

```
npx nx run-many -t test lint e2e

✓ nx run e2e:lint [existing outputs match the cache, left as is]

✓ nx run firstNxAngularStandaloneApplication:lint [existing outputs match the cache, left as is]
✓ nx run firstNxAngularStandaloneApplication:test [existing outputs match the cache, left as is]
✓ nx run e2e:e2e [existing outputs match the cache, left as is]
```

Exécuter des tâches

- Vous pouvez aussi **spécifier les projets que vous voulez cibler** en ajoutant l'option **--projects (-p)** suivie de la liste des projets.

```
nx run-many --target=test --watch  
  
nx run-many --target=test --projects=project1,project2
```

```
$ nx run-many -t build lint test -p cvViewer  
  
✓ nx run cvViewer:test [local cache]  
✓ nx run cvViewer:build:production [local cache]  
✓ nx run cvViewer:lint [existing outputs match the cache, left as is]
```

NX Successfully ran targets **build**, **lint**, **test** for project cvViewer (107ms)

Nx **read** the output from the cache instead of running the command for 3 out of 3 tasks.

Générer des éléments de votre application Angular

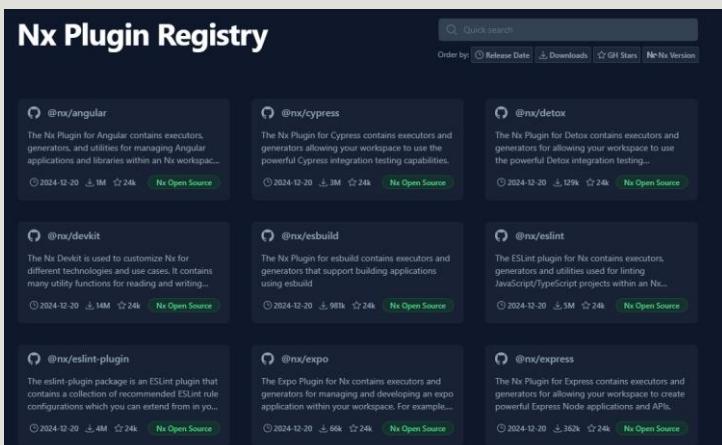
- Nx vient vous aider dans votre application Angular, il **offre donc les mêmes fonctionnalités que le cli d'Angular.**
- L'équivalent de vos schematics Angular sont appelés **Generator** dans Nx et ils existent pour plusieurs Framework et non seulement Angular.
- Le **générateur d'angular** est le [**@nx/angular**](#)
- Ces générateurs vont vous permettre de faire pleins de choses tels que la **création des éléments du Framework (composant, directives, pipes, ...)** ainsi que la **configuration du projet.**
- Pour **lister les générateurs** tapez la commande : **nx list**
- Pour **lister les fonctionnalités de votre générateur** tapez la commande :
nx list @nx/angular

Générer des éléments de votre application Angular

- Pour **lister les plugins nx** tapez la commande :

nx list

- Vous pouvez voir la liste des plugins aussi dans la page allouée <https://nx.dev/plugin-registry>



```
$ nx list
NX  Installed plugins:

@angular-devkit/build-angular (executors)
@angular/animations
@angular/cli
@angular/common
@angular/compiler
@angular/compiler-cli
@angular/core (generators)
@angular/forms
@angular/language-service
@angular/platform-browser
@angular/platform-browser-dynamic
@angular/router
@nx/angular (executors,generators)
@nx/cypress (executors,generators)
@nx/eslint (executors,generators)
@nx/eslint-plugin
@nx/jest (executors,generators)
@nx/js (executors,generators)
@nx/web (executors,generators)
@nx/workspace (executors,generators)
@schematics/angular (generators)
angular-eslint
ngx-toastr
nx (executors,generators)

NX  Also available:

@nx/detox (executors,generators)
@nx/esbuild (executors,generators)
@nx/expo (executors,generators)
@nx/express (generators)
@nx/gradle (graph)
@nx/nest (generators)
```

<https://nx.dev/plugin-registry>

Générer des éléments de votre application Angular

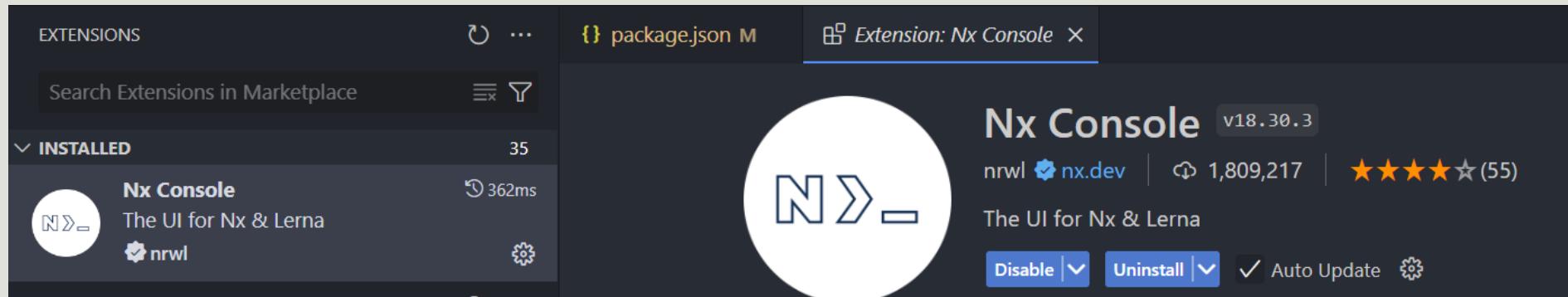
- Les générateurs font partie des plugins Nx et peuvent être appelés à l'aide de la commande `nx generate` (ou `nx g`) en utilisant la syntaxe suivante :

```
nx g <plugin-name>:<generator-name> [options]
```

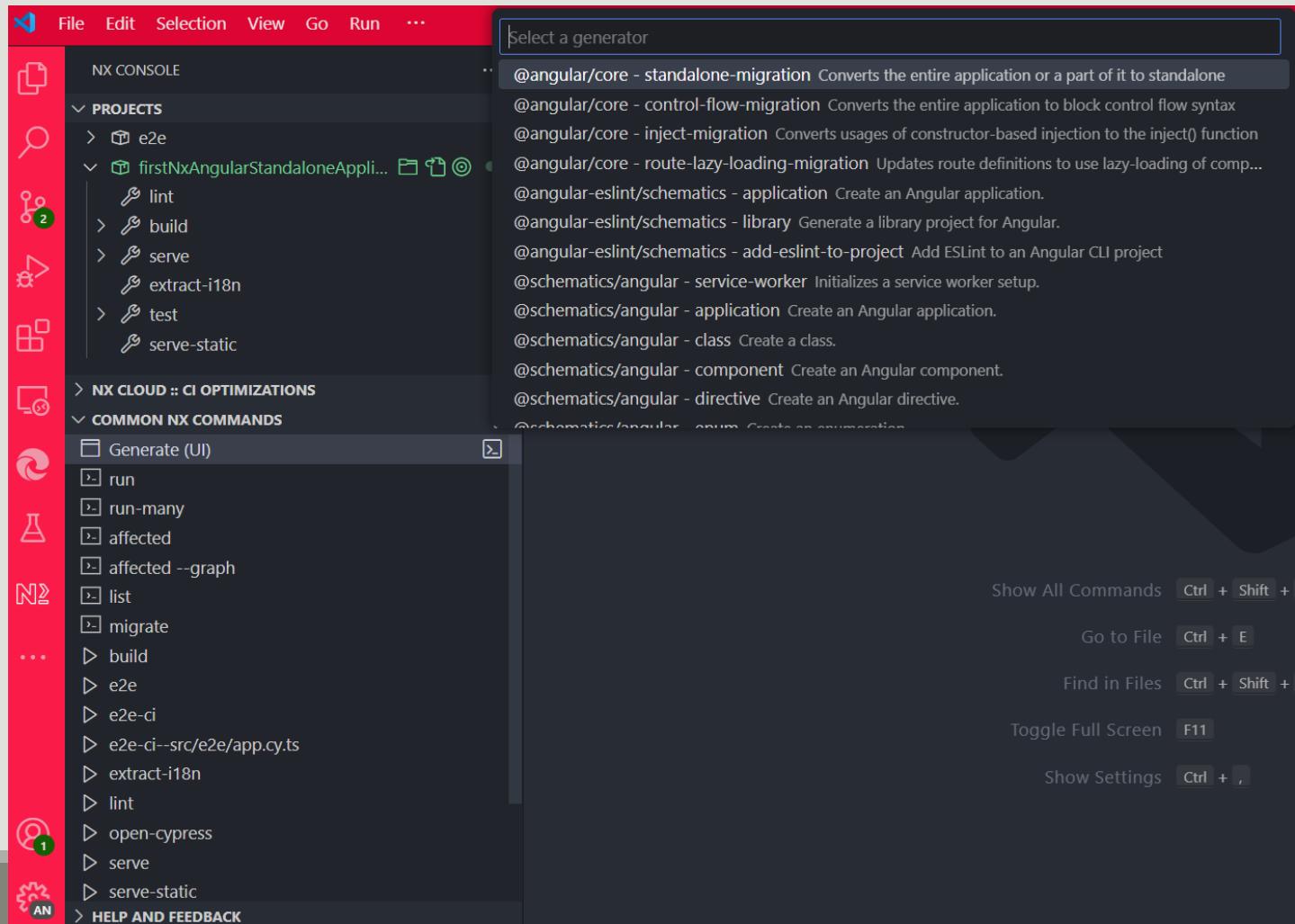
- Chaque generator a ses propres options
- par exemple pour générer un composant on doit utiliser le `@nx/angular:component` generator et lui spécifier le path :
`nx g @nx/angular:component --path=libs/shared/UI-COMMON/src/lib/hello/hello`
- Pour générer une librairie on utilise le `@nx/angular:library` generator et on lui passe le path et le name
`nx generate @nx/angular:library --directory=libs/shared/types --name=sahred-types`

Générer des éléments de votre application Angular

- Au lieu de passer pas la console, vous pouvez utiliser **un plugin Nx** disponible pour **VsCode** et les produits **jetbrains**, c'est **Nx Console**.
- Installez le et ouvrez le



Générer des éléments de votre application Angular



Architecture Applicative

Les librairies locales

- Afin de **modulariser votre architecture** et **découpler vos modules** vous pouvez utiliser les **librairies locales**.
- Ceci vous permet **d'externaliser un ensemble de fonctionnalités (features)** ou de **librairies ui** ou des **types dans des librairies sans les déployer sur npm**.
- La **séparation des concernes est encore plus évidente** et **l'exposition d'APIs pour chaque librairie** permet d'exposer uniquement ce que vous voulez.
- Une **meilleure scalabilité dans la partie CI, chaque librairie aura son propre lint test et build** et **Nx** fera ce qu'il faut pour **optimiser** tout ca.
- Chaque **librairie est vu comme un projet à part** permettant une **meilleure gestion des équipes** qui peuvent chacune s'occuper d'une librairie.

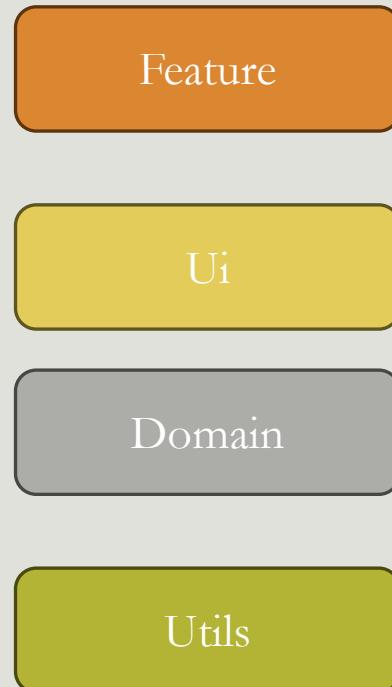
Architecture Applicative

Les différents types de librairies

- Nx catégorise les librairies en 4 types. Cependant, **ce n'est pas une catégorisation rigide**. Chaque projet peut avoir sa propre catégorisation selon le besoin propre du projet.
- **Les feature libraries** : Ce sont des librairies qui implémente une logique métier et qui sont intelligents. Ils peuvent communiquer avec la source de données et ça peut être des pages de votre application.
- **Les UI libraries** : contiennent que des **composant présentationnels (dumb)**.
- **Les domain ou data-access libraries** : c'est le code permettant l'interaction avec le backend. Vous y trouvez vos **services, models ou la partie state management si elle existe**.
- **Les utility libraries** : les fonctions réutilisables par l'application et les librairies.

Architecture Applicative

Les différents types de librairies



CvComponent

ListComponent

CvService

Fonctions utilitaires et helpers



Architecture Applicative

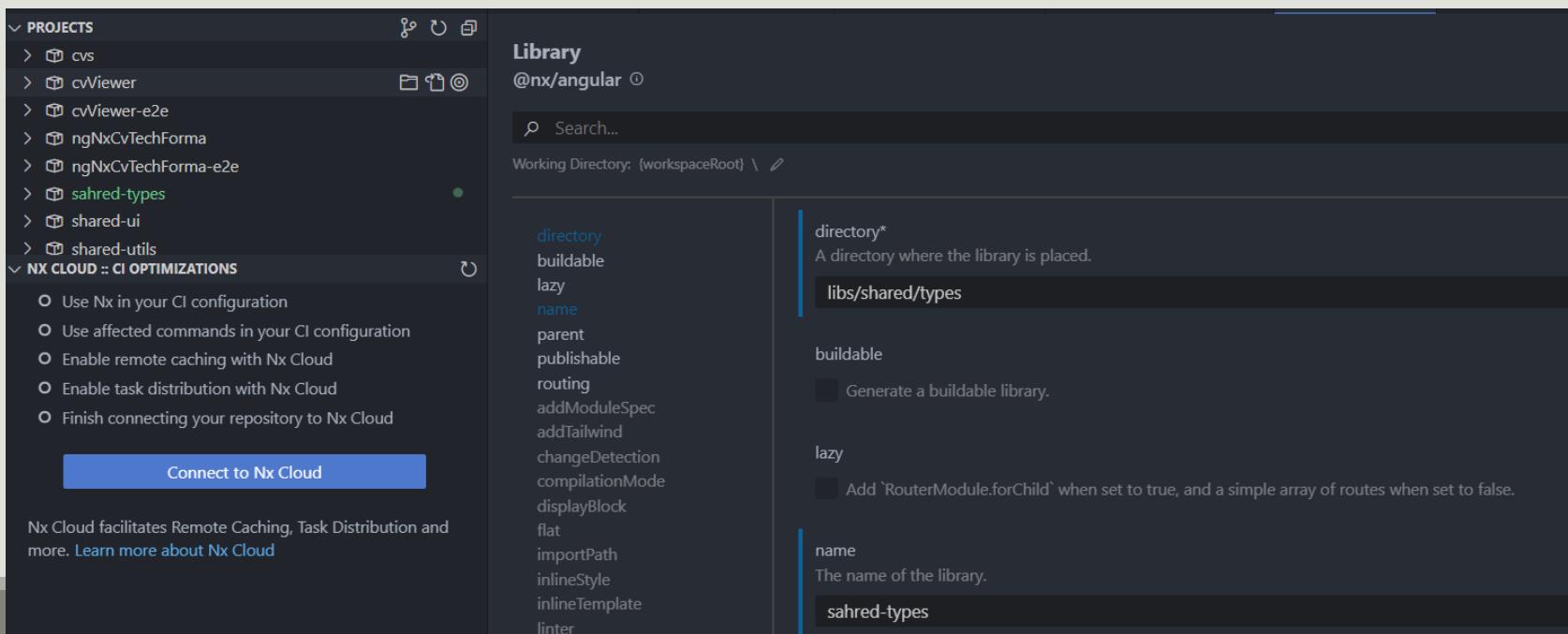
Les librairies locales

Créer une librairie via Nx

➤ Afin de créer votre librairie, vous pouvez passer par Nx Console ou la commande utilisant votre générateur.

`nx generate @nx/angular:library --directory=libs/shared/types --name=sahred-types`

Vous pouvez ajouter l'option `--dry-run` afin de visualiser les changements avant d'exécuter la commande

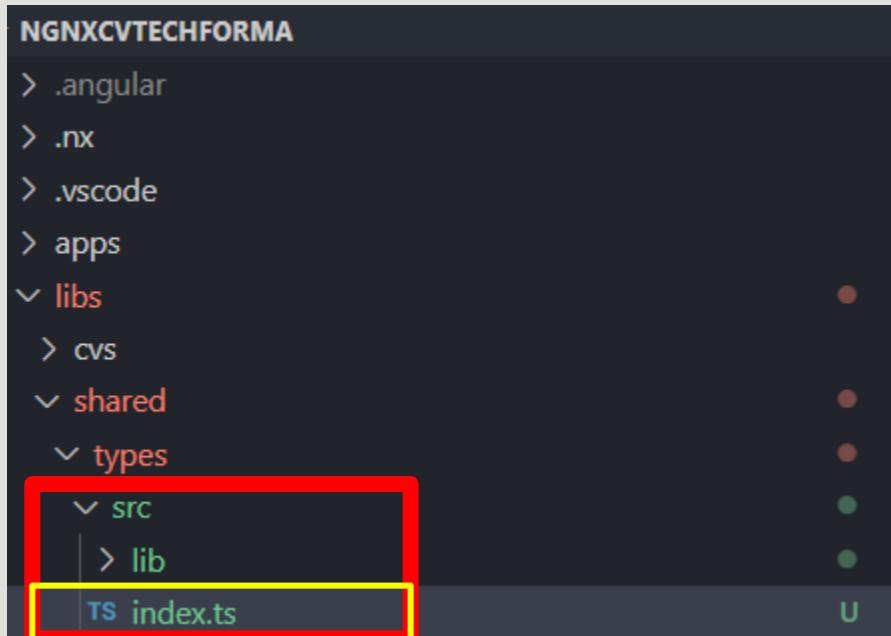


Architecture Applicative

Les librairies locales

Le contexte privé

- Tout ce qui se trouve dans la librairie est **considéré privé par défaut**.
- Tout **ce qui doit être visible** doit être **exporté via le fichier index.ts** sous le dossier source de votre librairie. C'est ce qu'on appelle **barrel**.



A screenshot of a code editor showing the contents of index.ts. The file path is libs > shared > types > src > index.ts. The code contains two lines: 1. `export * from './lib/shared-types/shared-types.component';` and 2. An empty line. The first line is highlighted with a yellow box.

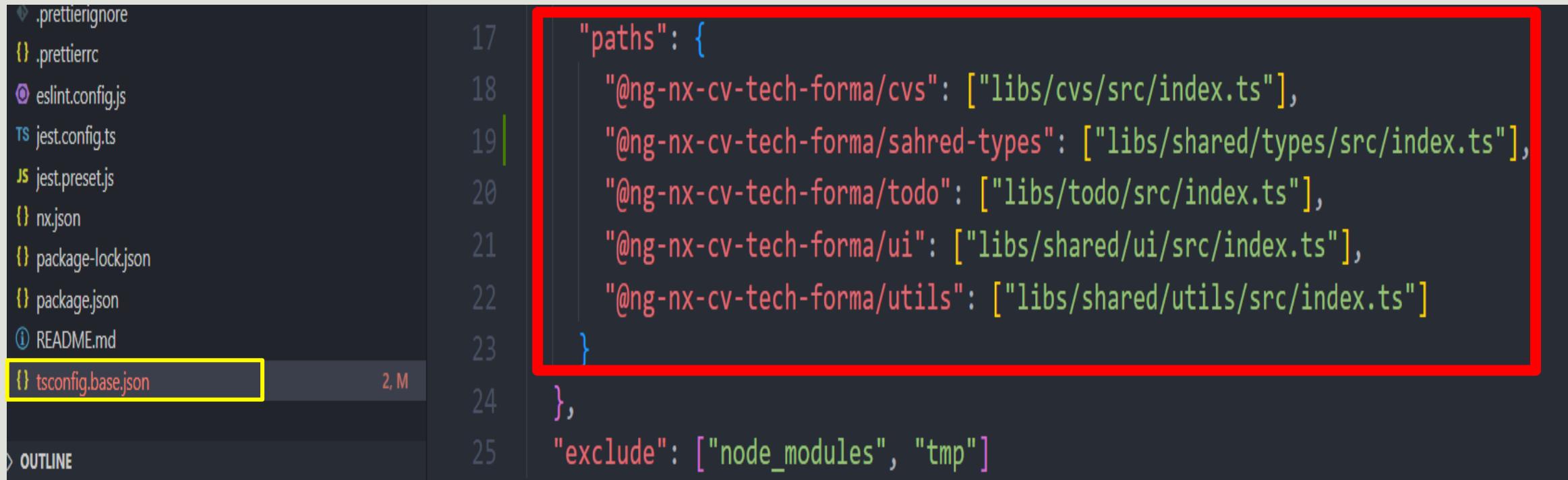
```
libs > shared > types > src > index.ts
1  export * from './lib/shared-types/shared-types.component';
2
```

Architecture Applicative

Les librairies locales

Définition de paths

- Lorsque vous créer une librairie, un **path est automatiquement définie** pour elle par nx au niveau du fichier **tsconfig.base.json** de votre workspace.



```
17 "paths": {  
18     "@ng-nx-cv-tech-forma/cvs": ["libs/cvs/src/index.ts"],  
19     "@ng-nx-cv-tech-forma/shared-types": ["libs/shared/types/src/index.ts"],  
20     "@ng-nx-cv-tech-forma/todo": ["libs/todo/src/index.ts"],  
21     "@ng-nx-cv-tech-forma/ui": ["libs/shared/ui/src/index.ts"],  
22     "@ng-nx-cv-tech-forma/utils": ["libs/shared/utils/src/index.ts"]  
23 },  
24 },  
25 "exclude": ["node_modules", "tmp"]
```

Architecture Applicative

Les librairies locales

Définition de paths

- Maintenant vous pouvez accéder à vos librairies comme si c'était des packages npm standard.

The screenshot shows a code editor with two panes. The left pane displays the file structure of an Angular project named 'NGNXCVTechForma'. It includes an 'apps' folder containing an 'ngNxCvTechForma' module, which has an 'app' folder with files like 'app.component.css', 'app.component.html', 'app.component.spec.ts', and 'app.component.ts'. Below 'ngNxCvTechForma' is an 'e2e' folder. The right pane shows the 'app.component.ts' file. A yellow box highlights the import statement for 'CvsComponent': `import { CvsComponent } from '@ng-nx-cv-tech-forma/cvs';`. Another yellow box highlights the usage of 'CvsComponent' in the component's template: `<lib-cvs/>`. The bottom pane shows the 'app.component.html' file with a tooltip for 'app-nx-welcome'.

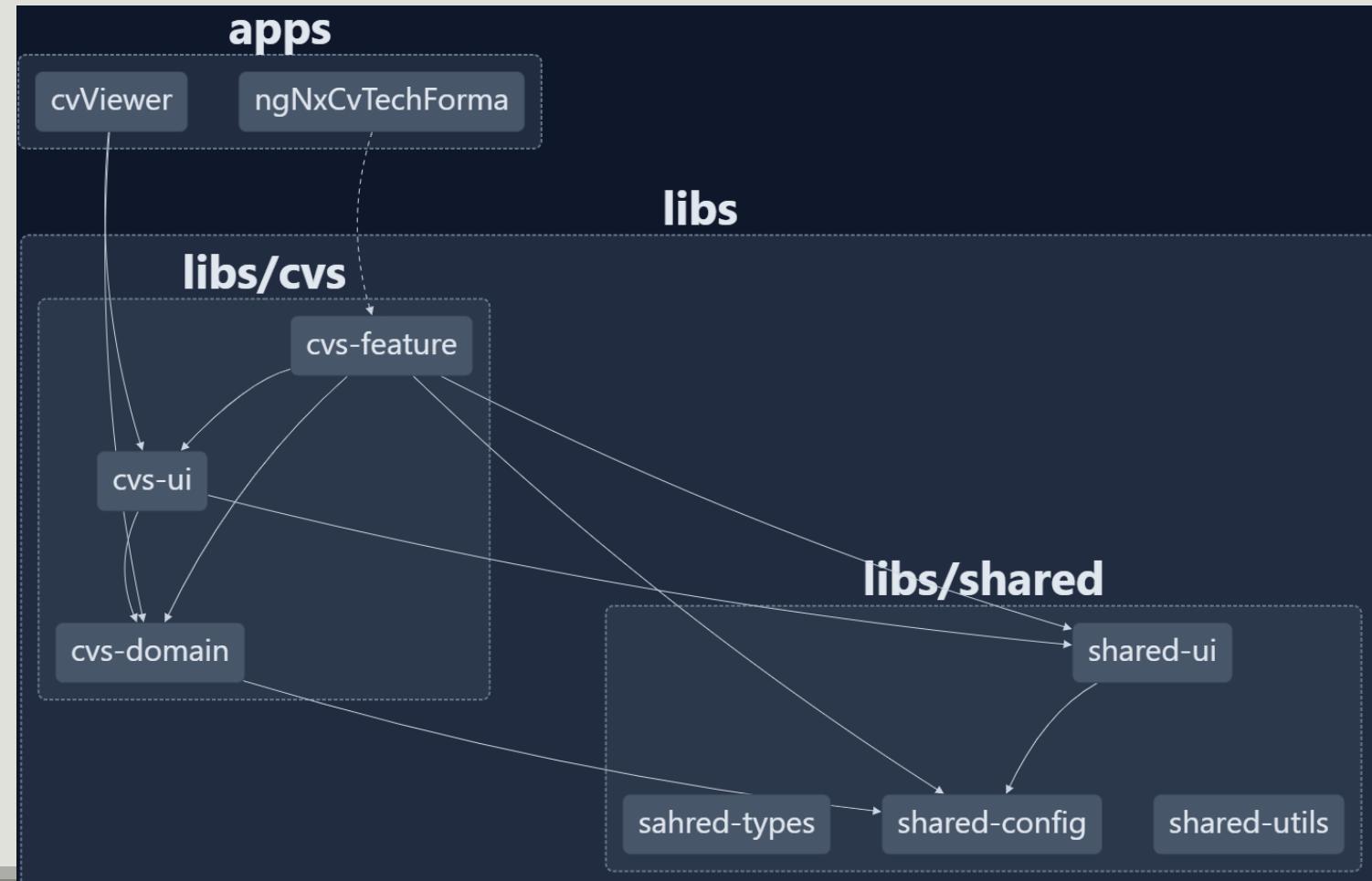
```
apps > ngNxCvTechForma > src > app > ts app.component.ts > AppComponent
1 import { Component } from '@angular/core';
2 import { RouterModule } from '@angular/router';
3 import { NxWelcomeComponent } from './nx>Welcome.component';
4 import { CvsComponent } from '@ng-nx-cv-tech-forma/cvs';

5
6 @Component({
7   standalone: true,
8   imports: [NxWelcomeComponent, RouterModule, CvsComponent],
9   selector: 'app-root',
10  templateUrl: './app.component.html',
11  styleUrls: ['./app.component.css'],
12 })
13 export class AppComponent {
14   title = 'ngNxCvTechForma';

< app.component.html M >
apps > ngNxCvTechForma > src > app > < app.component.html > app-nx>Welcome
Go to component
1 <app-nx>Welcome></app-nx>Welcome>
2 <lib-cvs/>
3 <router-outlet></router-outlet>
4
```

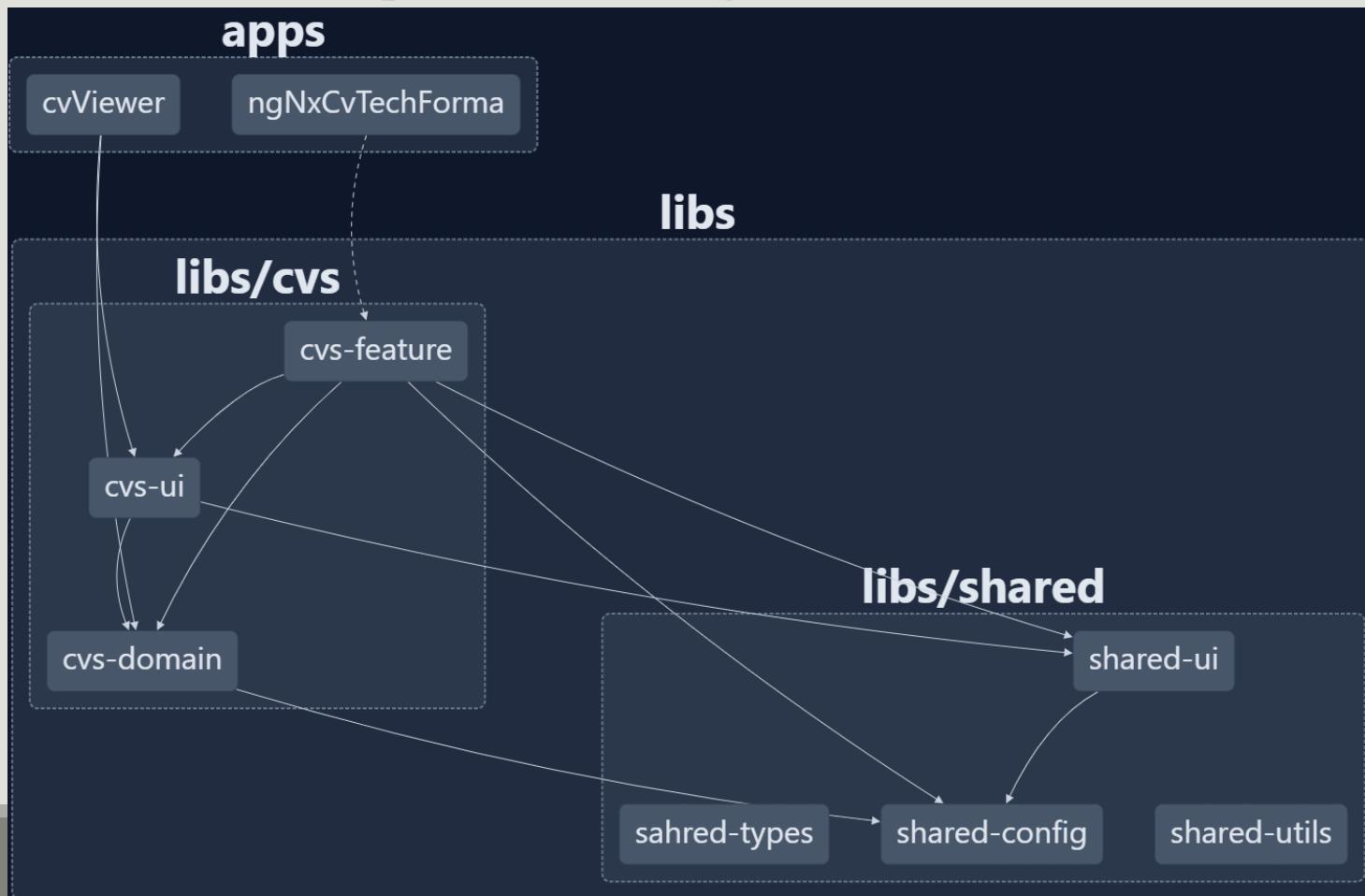
Visualiser les dépendances entre vos applications et vos librairies

- En utilisant les imports de vos librairies, **Nx** peut identifier les dépendances entre les librairies et les dépendances entre vos applications et vos librairies.
- Lancez la commande **nx graph** afin de visualiser ca



Visualiser les dépendances entre vos applications et vos librairies

- Lorsque la relation est en pointillé c'est que l'élément est lazy loaded.



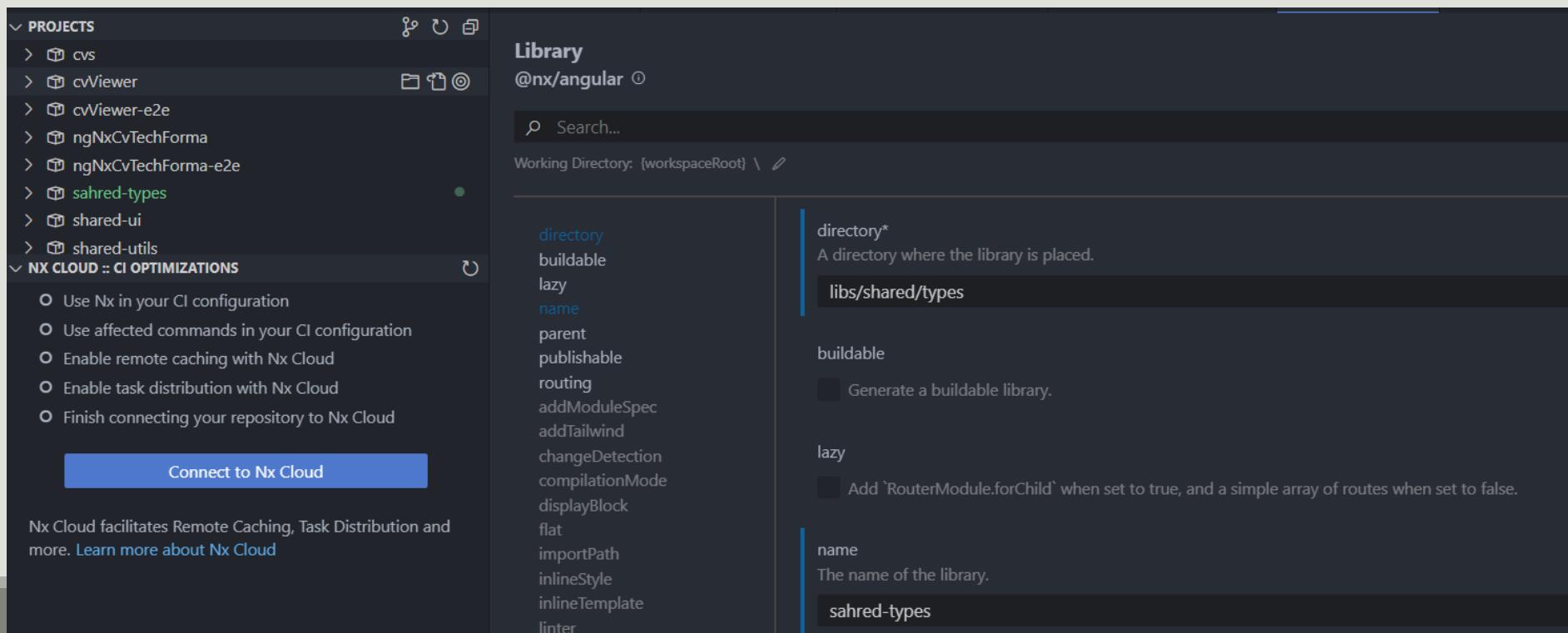
Architecture Applicative

Les librairies locales

Créer une librairie via Nx

- Afin de créer votre librairie, vous pouvez passer par Nx Console ou la commande utilisant votre générateur.

`npx nx generate @nx/angular:library --directory=libs/shared/types --name=sahred-types`

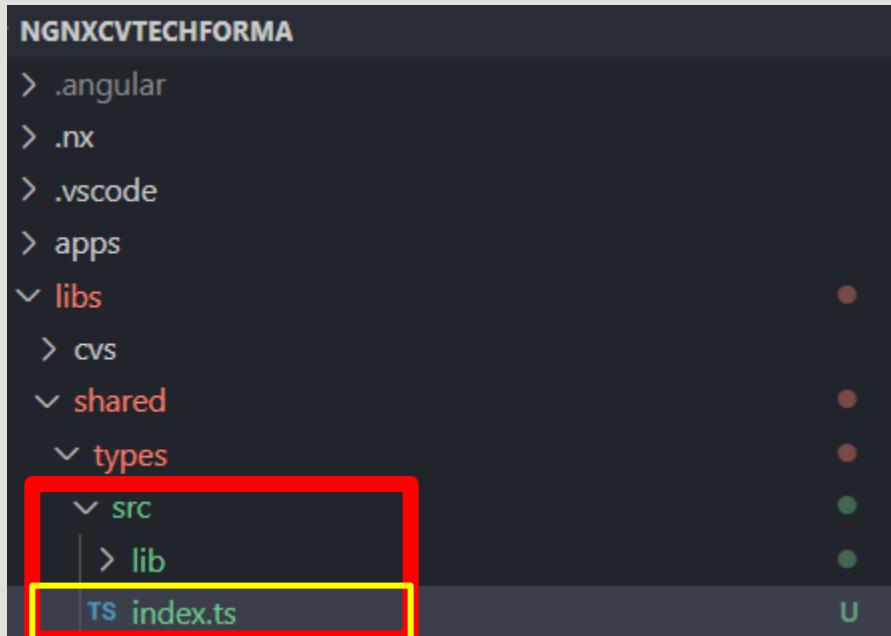


Architecture Applicative

Les librairies locales

Le contexte privé

- Tout ce qui se trouve dans la librairie est **considéré privé par défaut**.
- Tout **ce qui doit être visible** doit être **exporté via le fichier index.ts** sous le dossier source de votre librairie. C'est ce qu'on appelle **barrel**.



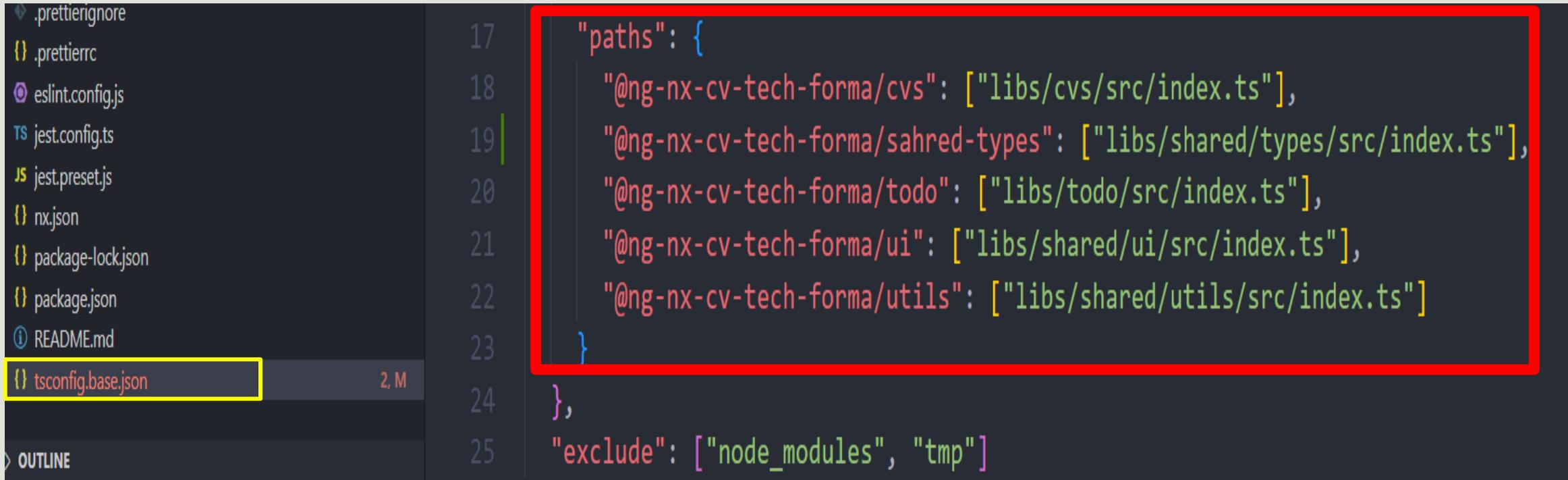
A screenshot of a code editor showing a file named 'index.ts'. The path is 'libs > shared > types > src > index.ts'. The code contains one line: `export * from './lib/shared-types/shared-types.component';`. The entire line is highlighted with a yellow box.

Architecture Applicative

Les librairies locales

Définition de paths

- Lorsque vous créer une librairie, un path est automatiquement définie pour elle par nx au niveau du fichier tsconfig.base.json de votre workspace.



The screenshot shows a code editor with the file `tsconfig.base.json` open. The file contains configuration for multiple libraries:

```
17 "paths": {  
18   "@ng-nx-cv-tech-forma/cvs": ["libs/cvs/src/index.ts"],  
19   "@ng-nx-cv-tech-forma/shared-types": ["libs/shared/types/src/index.ts"],  
20   "@ng-nx-cv-tech-forma/todo": ["libs/todo/src/index.ts"],  
21   "@ng-nx-cv-tech-forma/ui": ["libs/shared/ui/src/index.ts"],  
22   "@ng-nx-cv-tech-forma/utils": ["libs/shared/utils/src/index.ts"]  
23 },  
24 },  
25 "exclude": ["node_modules", "tmp"]
```

A red box highlights the `"paths": { ... }` block, which defines paths for five different libraries: `cvs`, `shared-types`, `todo`, `ui`, and `utils`. The `tsconfig.base.json` file is also highlighted with a yellow border in the sidebar.

Architecture Applicative

Les librairies locales

Définition de paths

- Maintenant vous pouvez accéder à vos librairies comme si c'était des packages npm standard.

The screenshot shows a code editor with two panes. The left pane displays the project structure of an Angular application named 'ngNxCvTechForma'. It includes an 'apps' folder containing an 'ngNxCvTechForma' module, which has an 'app' folder with files like 'app.component.css', 'app.component.html', 'app.component.spec.ts', and 'app.component.ts'. Below 'ngNxCvTechForma' is an 'e2e' folder. The right pane shows the 'app.component.ts' file. A yellow box highlights the import statement for 'CvsComponent': `import { CvsComponent } from '@ng-nx-cv-tech-forma/cvs';`. The 'app.component.html' file is also shown, with a yellow box highlighting the component tag: `<lib-cvs/>`. The status bar at the bottom indicates the path: 'apps > ngNxCvTechForma > src > app > app.component.html > app-nx-welcome'.

```
apps > ngNxCvTechForma > src > app > ts app.component.ts > AppComponent
1 import { Component } from '@angular/core';
2 import { RouterModule } from '@angular/router';
3 import { NxWelcomeComponent } from './nx>Welcome.component';
4 import { CvsComponent } from '@ng-nx-cv-tech-forma/cvs';

5
6 @Component({
7   standalone: true,
8   imports: [NxWelcomeComponent, RouterModule, CvsComponent],
9   selector: 'app-root',
10  templateUrl: './app.component.html',
11  styleUrls: ['./app.component.css'],
12 })
13 export class AppComponent {
14   title = 'ngNxCvTechForma';

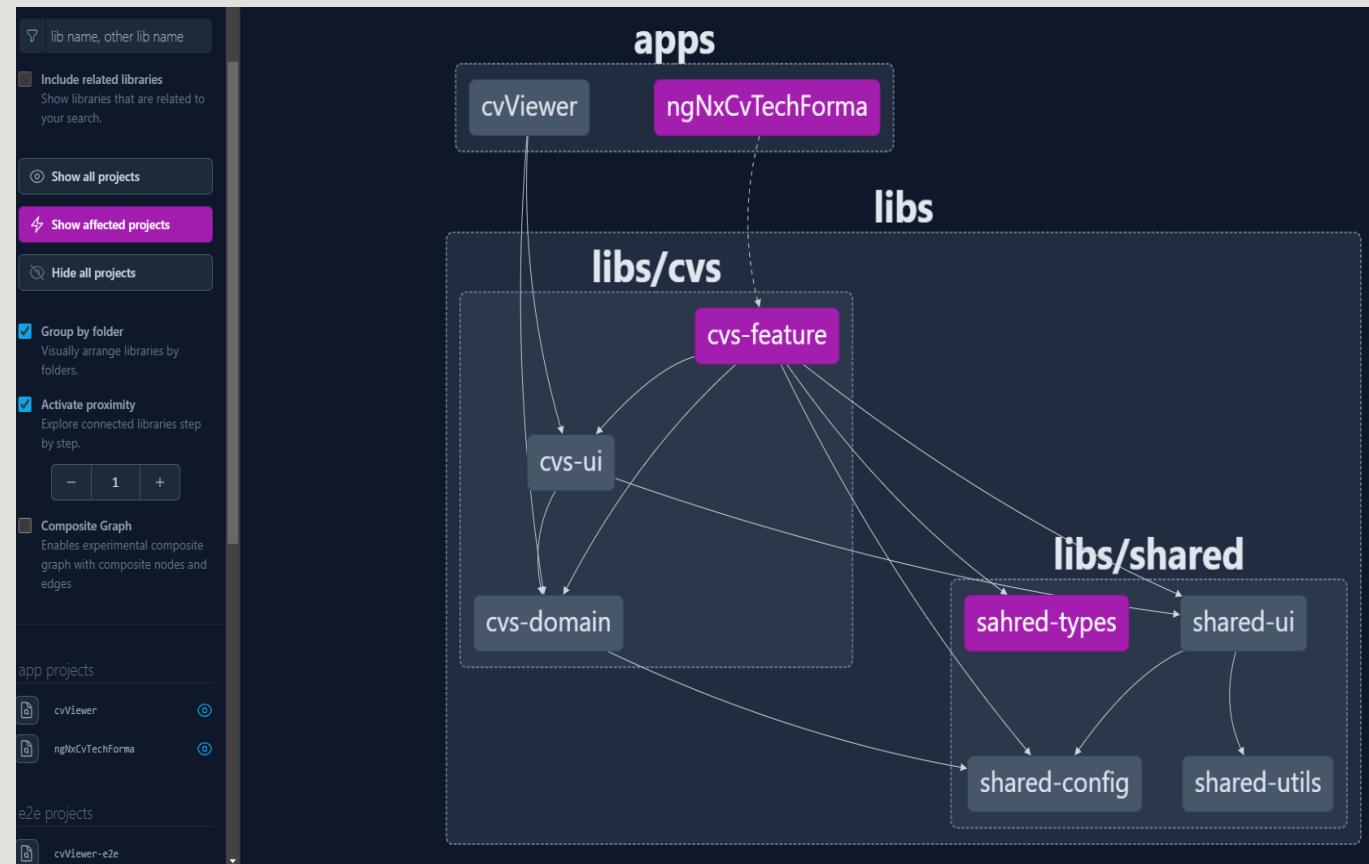
< app.component.html M >
apps > ngNxCvTechForma > src > app > app.component.html > app-nx-welcome
Go to component
1 <app-nx>Welcome></app-nx>Welcome>
2 <lib-cvs/>
3 <router-outlet></router-outlet>
4
```

Optimisez vos tâches avec **affected**

- À mesure que votre espace de travail s'agrandit, la relance de vos tests, vos lints et vos build de tous les projets **deviendra de plus en plus lente**.
- Pour résoudre ce problème, il est préférable de **relancer ces commandes** avec **uniquement les parties concernées par les changements effectués**.
- Nx est fourni avec une commande **affected** qui permet de gérer ça pour vous.
- À l'aide de cette commande, Nx **détermine l'ensemble minimal de projets affectés** par la modification et **exécute uniquement les tâches sur ces projets affectés**
- Pour ce faire, tapez la commande nx **affected -t <vosTaches>**
- nx **affected -t test build lint**

Optimisez vos tâches avec affected Visualisez les éléments affectés

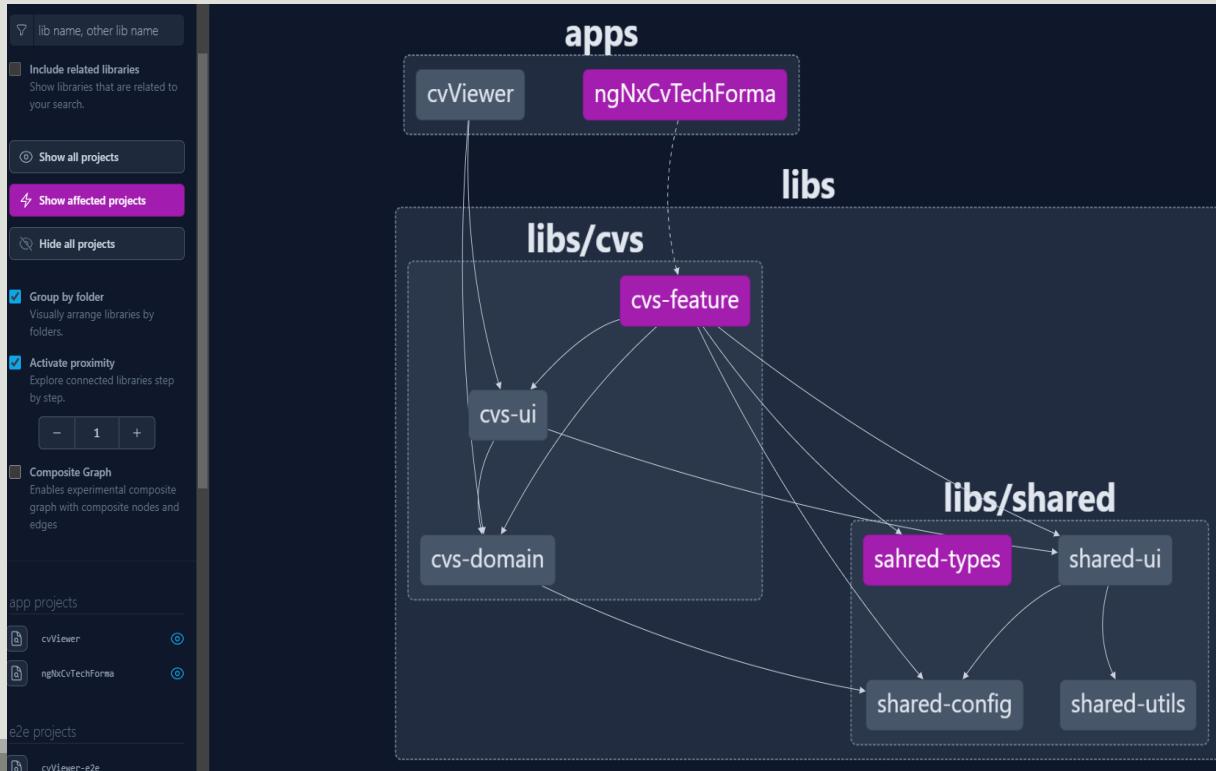
- Vous pouvez afficher le graph en spécifiant l'option **--affected** pour visualiser les **éléments affectés**.
- En changeant uniquement la lib **shared-types**, Nx est capable d'identifier ses dépendances.



Optimisez vos tâches avec affected

Exécuter des tâches uniquement sur les éléments affectés

- En lançant la commande `nx affected -t build lint test`, Nx ne se chargera que des librairies et projets affectés.



```
$ nx affected -t lint build test  
NX  Affected criteria defaulted to --base=master --head=HEAD  
  
✓  nx run sahred-types:lint (3s)  
✓  nx run sahred-types:test (5s)  
✓  nx run cvs-feature:lint (2s)  
✓  nx run cvs-feature:test (14s)  
✓  nx run ngNx CvTechForma:test (9s)  
✓  nx run ngNx CvTechForma:build:production (9s)  
✓  nx run ngNx CvTechForma:lint (2s)  
✓  nx run ngNx CvTechForma-e2e:lint (2s)  
  
NX  Successfully ran targets lint, build, test for 4 projects (17s)
```