

CED

Formation Angular

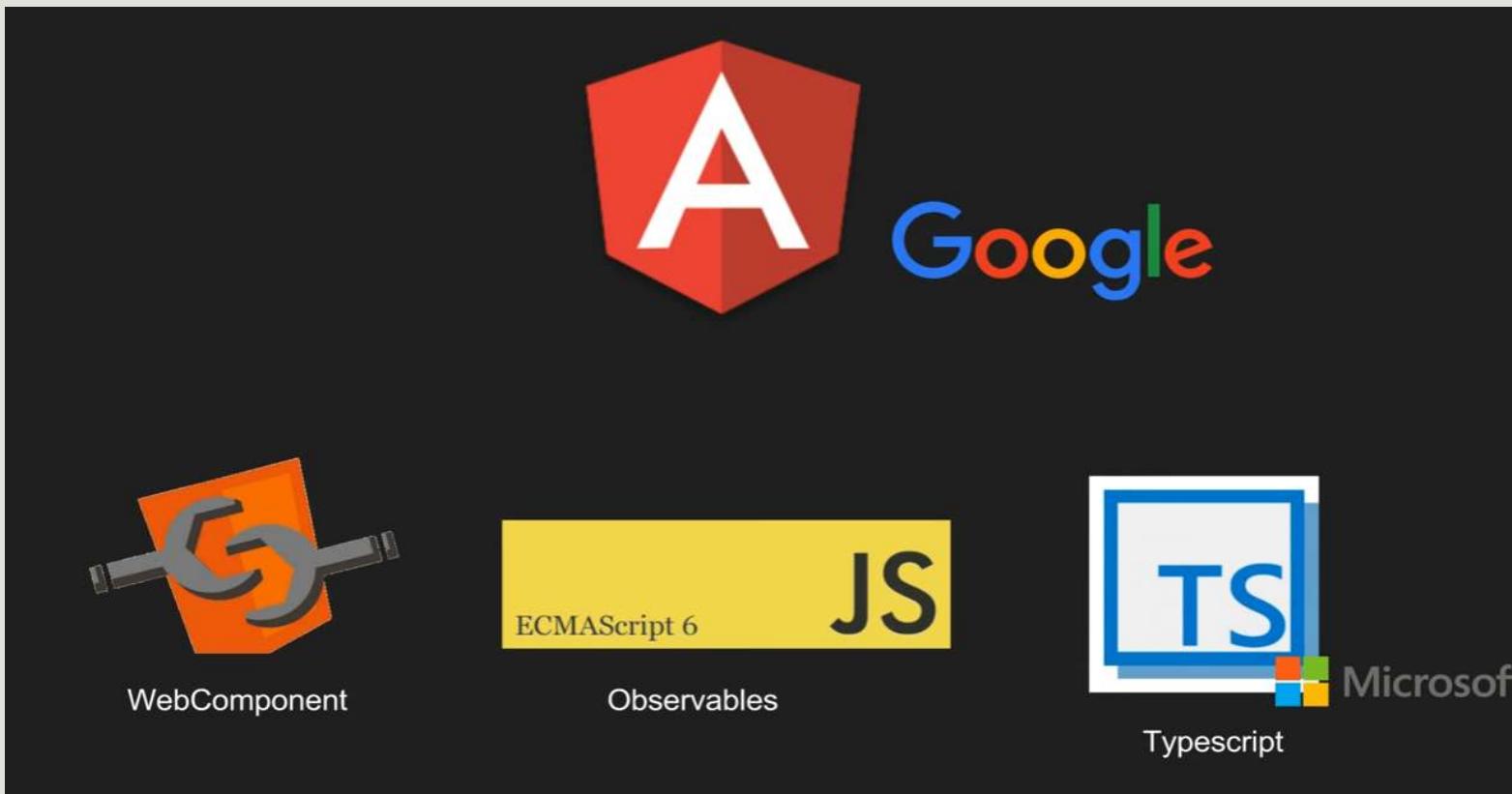
Introduction

AYMEN SELLAOUTI

Plan du Cours

1. Injection de dépendances
2. Router Module
3. RxJS et HTTP
4. Reactive Form
5. Modules
6. Zone et Change Detection

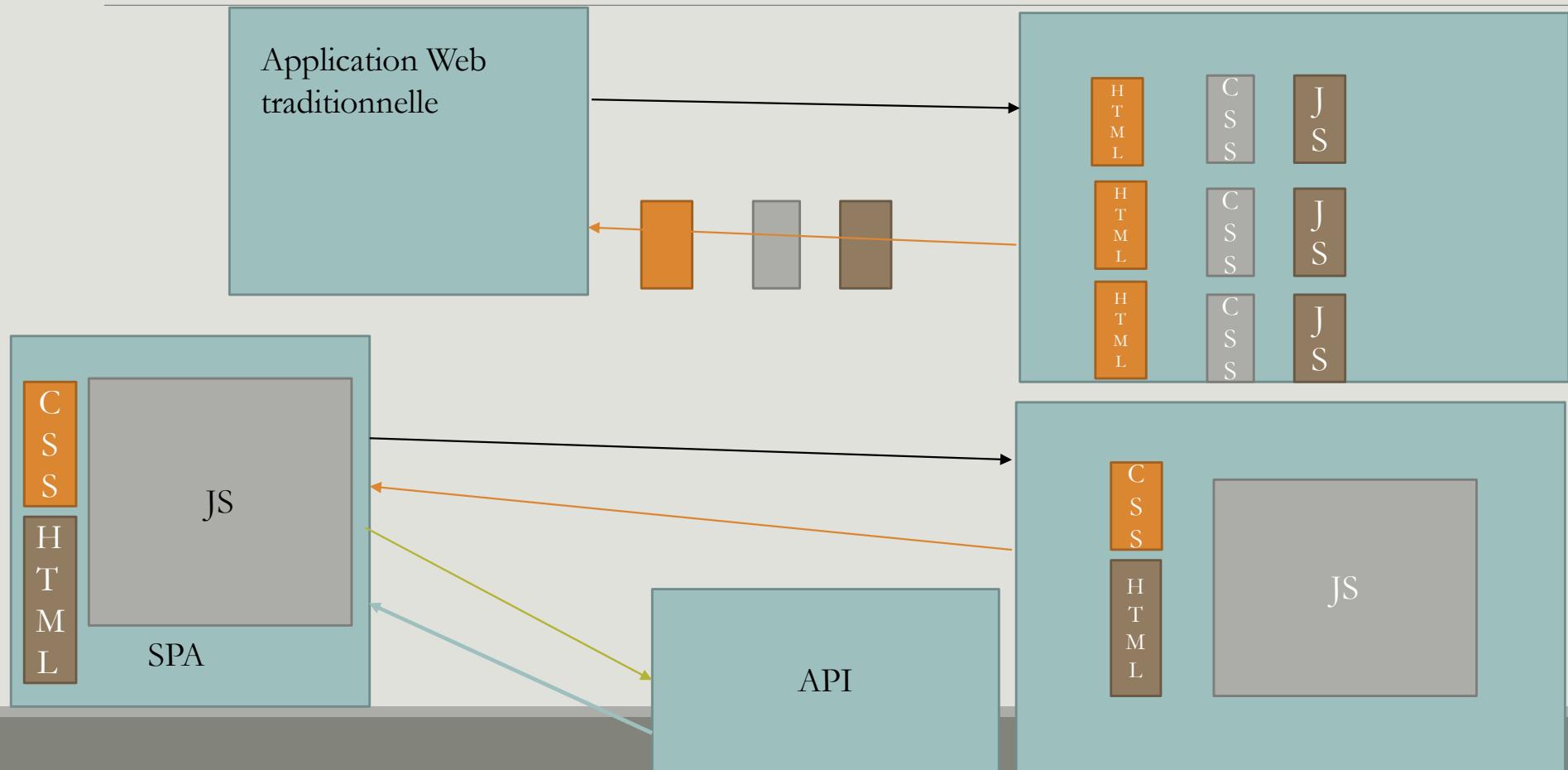
C'est quoi Angular?



C'est quoi Angular?

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

SPA

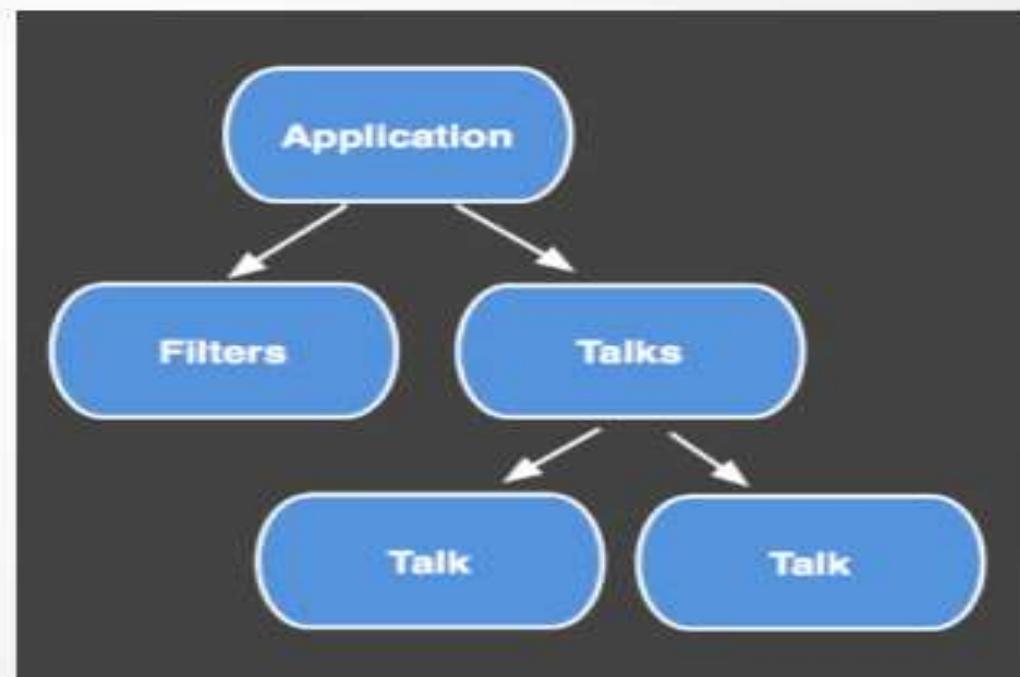


Angular : Arbre de composants

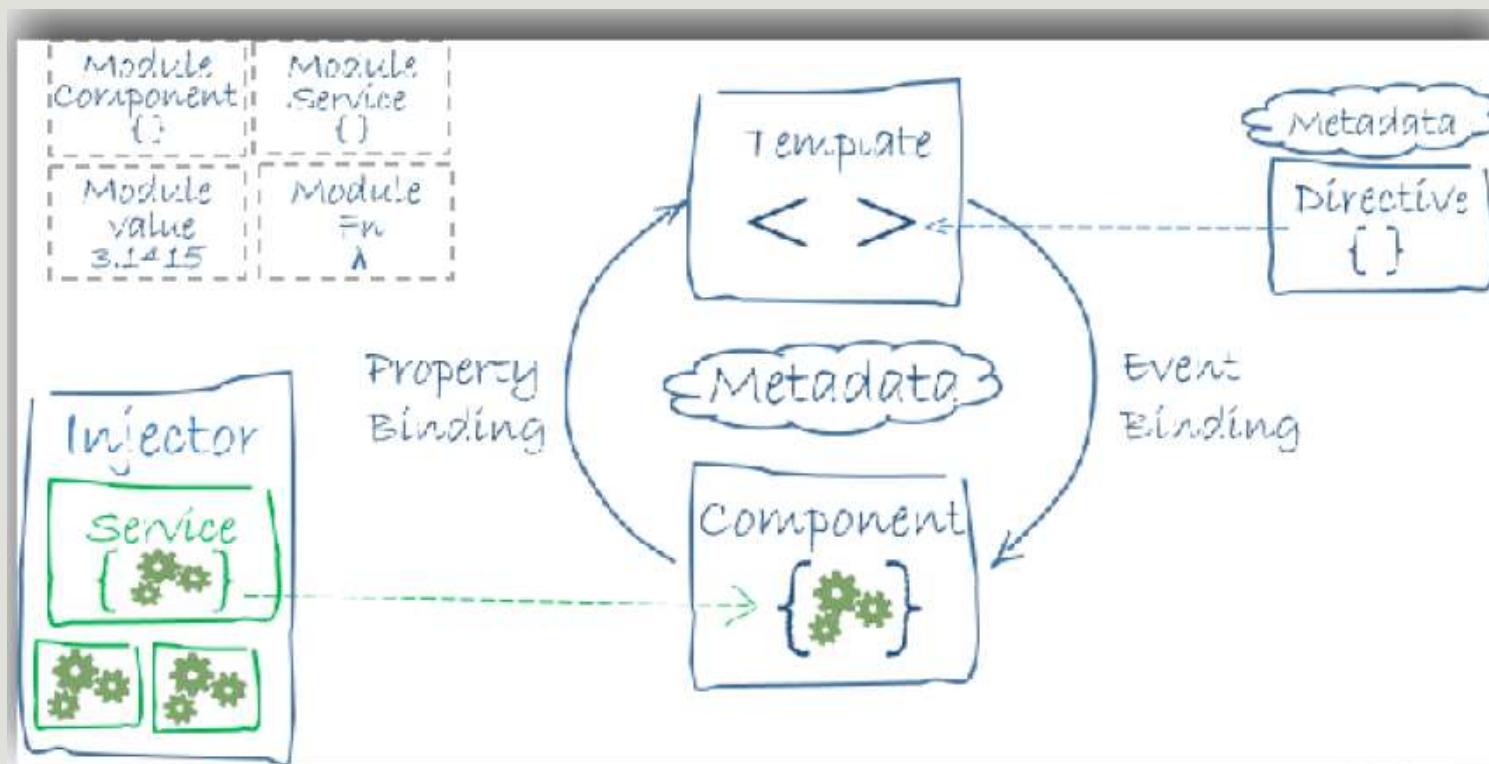
Speaker: Rich Hickey

FILTER

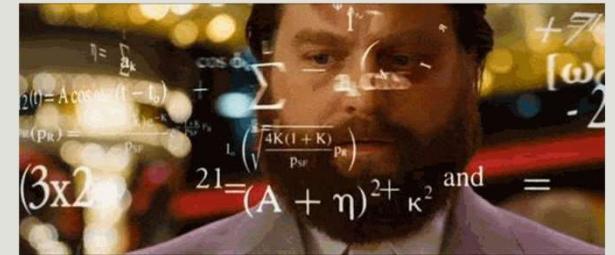
| Rating | Title | Speaker | Action |
|--------|---------------------|-------------|-------------------|
| 9.1 | Are We There Yet? | Rich Hickey | WATCH RATE |
| 8.5 | The Value of Values | Rich Hickey | WATCH RATE |
| 8.2 | Simple Made Easy | Rich Hickey | WATCH RATE |



Architecture Angular



Installation d'Angular Angular Cli



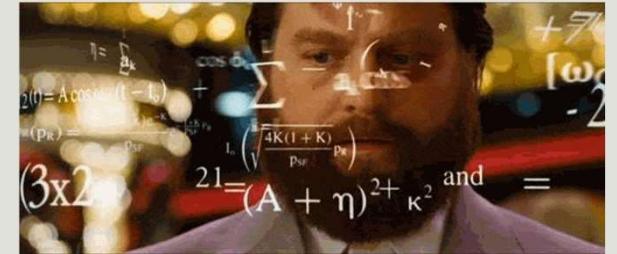
- Nous allons installer notre première application en utilisant [angular Cli](#).
- Si vous avez Node c'est bon, sinon, installer [NodeJs](#) sur votre machine. Vous devez avoir une version de [node nécessaire pour la version Angular que vous installez](#).
- Une fois installé vous disposez de npm qui est le [Node Package Manager](#). Afin de vérifier si vous avez NodeJs installé, tapez `npm -v`.
- Installer maintenant le Cli en tapant la : `npm install -g @angular/cli`
 - `npm install -g @angular/cli@16.0.0` installe la version 16.0.0
 - `npm view @angular/cli` affiche la liste des versions de la cli
- Installer un nouveau projet à l'aide de la commande `ng new nomProjet`
- `npx @angular/cli@16.2.14 new nomProjet`
- Afin d'avoir du help pour le cli tapez `ng help`
- Lancer le projet en utilisant la commande `ng serve`

Angular dépendances

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

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

Installation d'Angular Angular Cli



- Vous pouvez configurer le Host ainsi que le port avec la commande suivante : `ng serve --host leHost --port lePort`

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

Quelques commandes du Cli

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

Ajouter Bootstrap

On peut ajouter Bootstrap de plusieurs façons :

- 1- Via le CDN à ajouter dans le fichier index.html
- 2- En le téléchargeant du site officiel
- 3- Via npm
 - `npm install bootstrap --save`

Ajouter Bootstrap

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

1- En l'ajoutant dans index.html

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

```
"styles": [  
  "./node_modules/bootstrap/dist/css/bootstrap.min.css",  
  "styles.css",  
],  
"scripts": [  
  "./node_modules/jquery/dist/jquery.min.js",  
  "./node_modules/popper.js/dist/umd/popper.min.js",  
  "./node_modules/bootstrap/dist/js/bootstrap.min.js"  
],
```

Ajouter Bootstrap

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

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

Essayer la même chose avec font-awesome.

Angular Les composants

AYMEN SELLAOUTI

Objectifs

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

Qu'est ce qu'un composant (Component)

- Un **composant** est une **classe** qui permet de **gérer une vue**. Il se charge **uniquement** de cette vue là.
Plus simplement, un composant est un fragment HTML géré par une classe JS (component en angular et controller en angularJS)
- Une application Angular est un **arbre de composants**
- La racine de cet arbre est l'application lancée par le navigateur au lancement.
- Un composant est :
 - Composable (normal c'est un composant)
 - Réutilisable
 - Hiérarchique (n'oublier pas c'est un arbre)

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

Quelques exemples

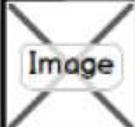
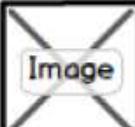
The screenshot displays a user interface for an 'Inventory Management App'. The top bar includes standard window controls (minimize, maximize, close) and a search bar. Below this is a navigation bar with three tabs: 'Home' (white), 'Products' (blue, currently active), and 'Help' (white). A breadcrumb trail 'Products > Products List' is positioned just below the navigation bar. The main content area contains three product items, each with an 'Image' placeholder icon:

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

Annotations on the left side identify the 'Navigation Component' (covering the top bar and part of the navigation bar), the 'Breadcrumbs Component' (covering the breadcrumb trail), and the 'ProductList Component' (covering the main content area with the three products).

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

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

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

Chargement de la classe Component

Le décorateur @Component permet d'ajouter un comportement à notre classe et de spécifier que c'est un Composant Angular.

selector permet de spécifier le tag (nom de la balise) associé ce composant

templateUrl: spécifie l'url du template associé au composant

styleUrls: tableau des feuilles de styles associé à ce composant

Export de la classe afin de pouvoir l'utiliser

Création d'un composant

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

- Manuelle
 - Créer la classe
 - Importer Component
 - Ajouter l'annotation et l'objet qui la décore
 - Ajouter le composant dans le **AppModule(app.module.ts)** dans l'attribut **declarations**

- Cli
 - Avec la commande **ng generate component my-new-component** ou son raccourci **ng g c my-new-component**

Création d'un composant

- La commande `generate` possède plusieurs options

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

Property Binding



Property Binding

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

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

Event Binding

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

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

Property Binding et Event Binding

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

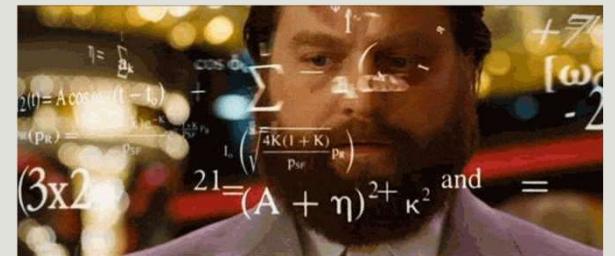
@Component({
  selector: 'inter-interpolation',
  template : `interpolation.html` ,
  styles: []
})
export class InterpolationComponent {
  nom:string ='Aymen Sellaouti';
  age:number =35;
  adresse:string ='Chez moi ou autre part :)';
  getName() {
    return this.nom;
  }
  modifier(newName) {
   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

Component

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

Template

Exercice



- Crée 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

Property Binding et Event Binding

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

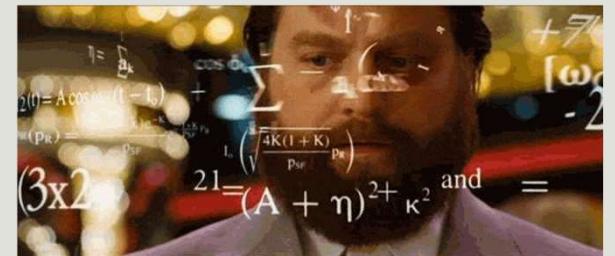
@Component({
  selector: 'app-two-way',
  templateUrl: './two-
way.component.html',
  styleUrls: ['./two-
way.component.css']
})
export class TwoWayComponent {
  two:any="myInitValue";
}
```

Component

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

Template

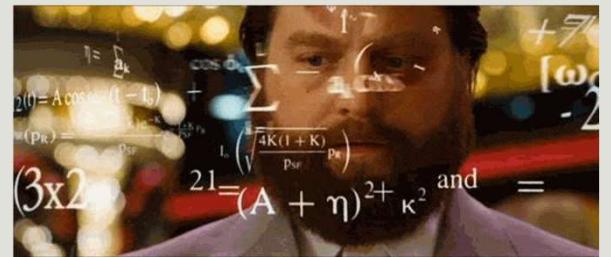
Exercice



- 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

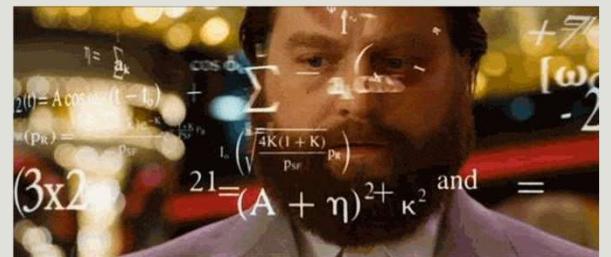
name :

firstname :

job :

path :

Exercice



Two Way Binding

Profile picture:

Sellaouti Aymen
Enseignant

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

Auto Rotation

Nom :

Prénom :

Job :

image :

Citation Favorite :

Décrivez nous votre travail :

Mots clé de votre travail :

Two Way Binding

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

J enseigne aux étudiants les technos du Web
HTML CSS JS PHP Symfony Angular

235 Followers 114 Following 35 Projects

[f](#) [G+](#) [t](#)

Nom :

Prénom :

Job :

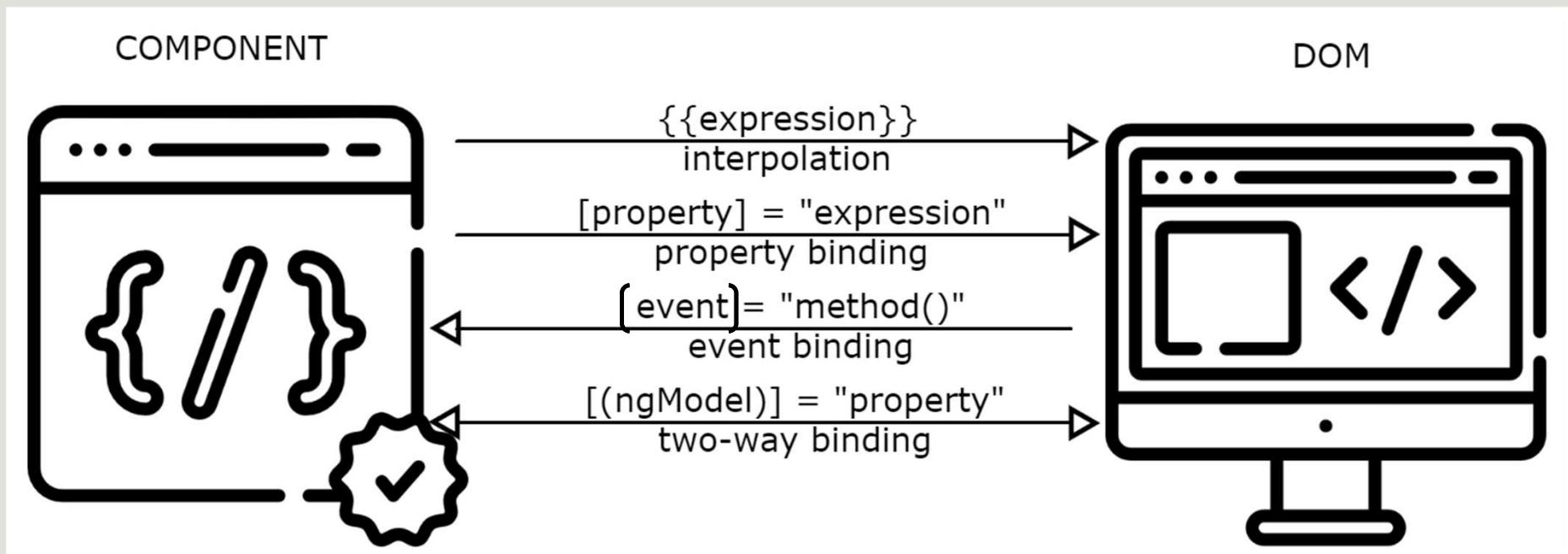
image :

Citation Favorite :

Décrivez nous votre travail :

Mots clé de votre travail :

Résumé : Property Binding



Récap Binding

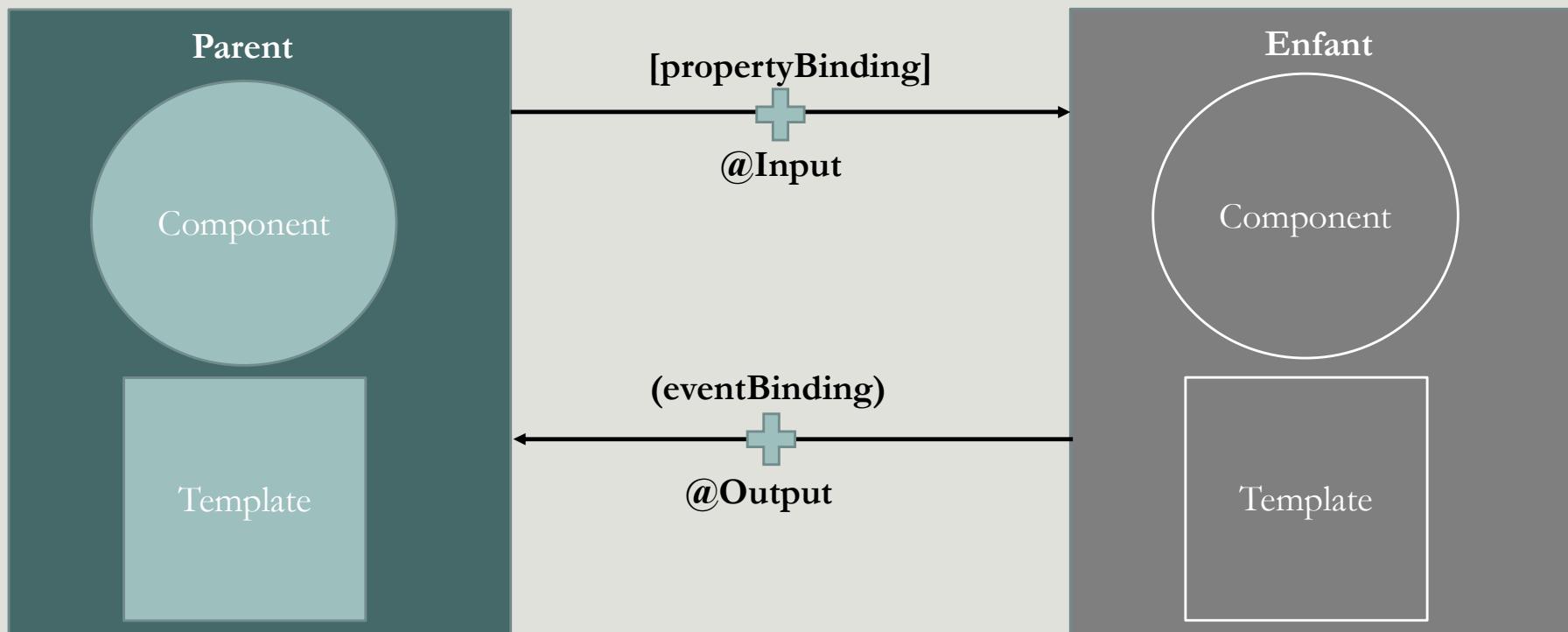
```
<div [style.backgroundColor]="color">  
  Color  
</div>  
  
<input [(ngModel)]="color"  
  type="text"  
  class="form-control"  
>  
  
le contenu de la propriété color est {{color}}  
<button (click)="loggerMesData()">log data</button>  
<br>  
<a (click)="goToCv()">Go to Cv</a>
```

HTML

```
@Component({  
  selector: 'app-color',  
  templateUrl: './color.component.html',  
  styleUrls: ['./color.component.css'],  
  providers: [PremierService]  
)  
export class ColorComponent implements OnInit {  
  color = 'red';  
  constructor() { }  
  
  ngOnInit() {}  
  processReq(message: any) {  
    alert(message);  
  }  
  loggerMesData() {  
    this.premierService.logger('test');  
  }  
  goToCv() {  
    const link = ['cv'];  
    this.router.navigate(link);  
  }  
}
```

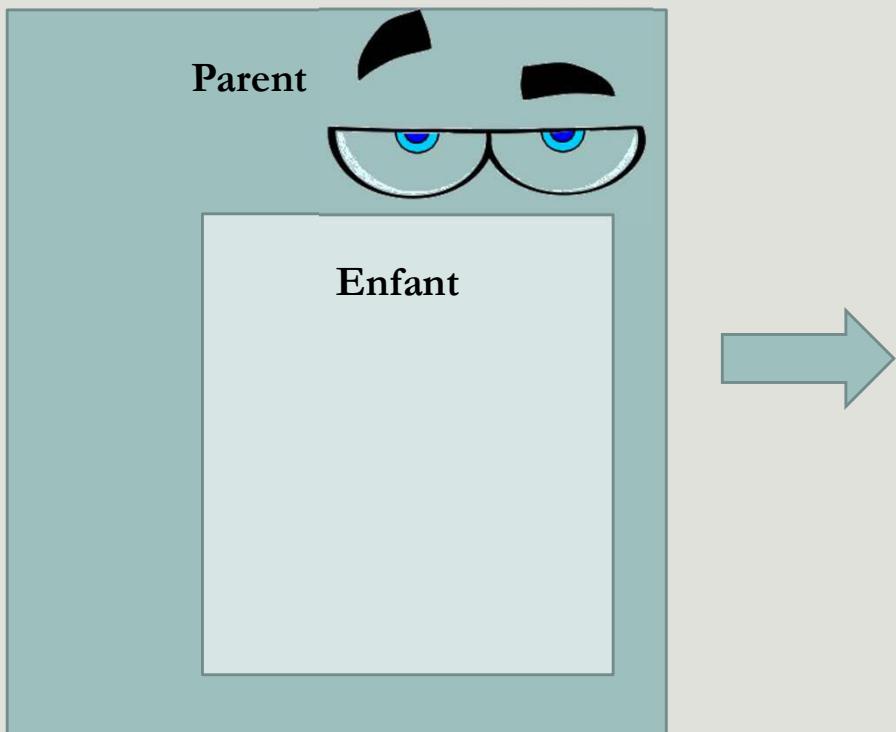
TS

Interaction entre composants



Pourquoi ?

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

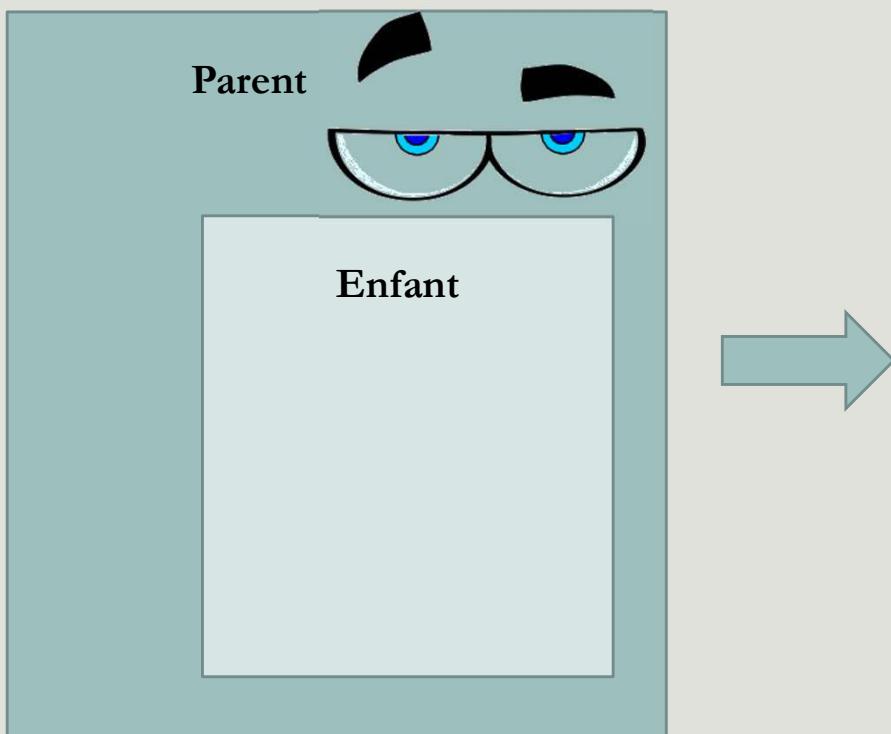


```
import { Component } from '@angular/core';

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

Interaction du père vers le fils

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

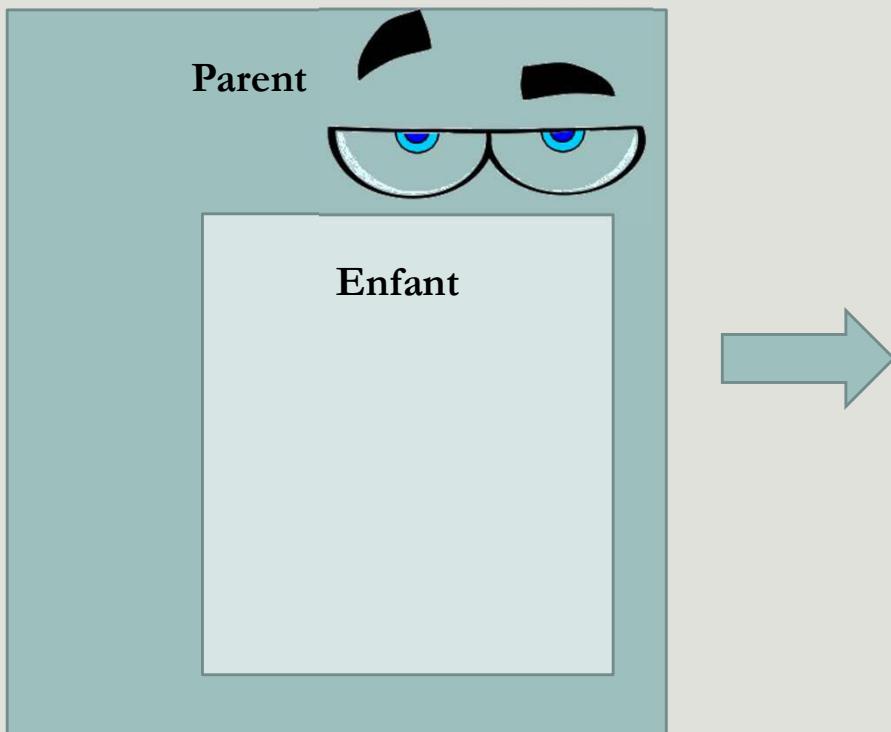


```
import { Component } from '@angular/core';

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

Interaction du père vers le fils

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

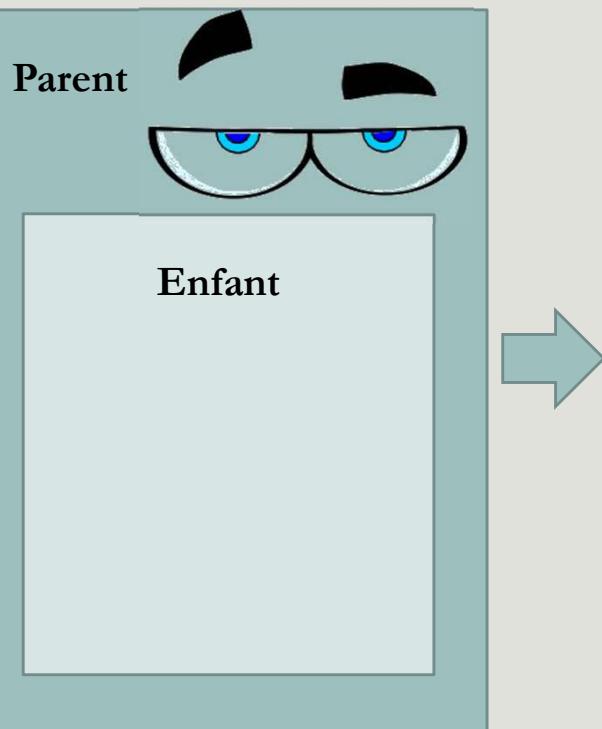


```
import { Component } from '@angular/core';

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

Interaction du père vers le fils

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



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

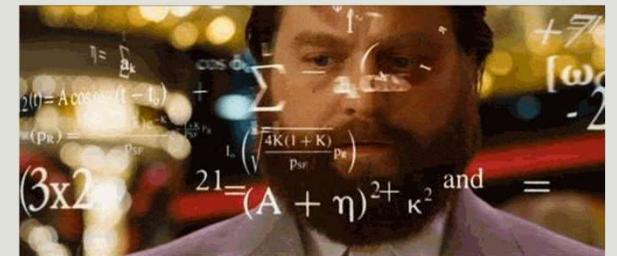
@Component({
selector: 'app-root',
template: `
<p>Je suis le composant père </p>
<forma-fils [external]="title">
</forma-fils>
`,
styles: []
})
export class AppComponent {
  title = 'app works !';
}
```

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

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

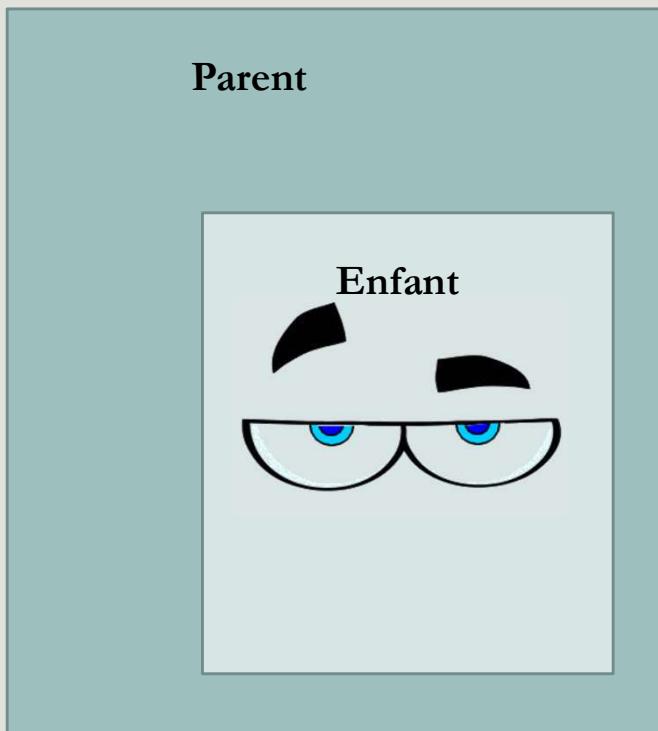
Exercice

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



Interaction du fils vers le père

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



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

Interaction du fils vers le père

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



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

Interaction du père vers le fils

La variable \$event est la variable utilisée pour faire passer des informations.

Parent



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

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

@Component({
  selector: 'bind-output',
  template: `
    <button (click)="incrementer()">+</button> `,
  styleUrls: ['./output.component.css']
})

export class OutputComponent {
  valeur:number=0;
  // On déclare un événement
  @Output() valueChange=new EventEmitter();
  incrementer(){
    this.valeur++;
    this.valueChange.emit(
      this.valeur
    );
  }
}
```

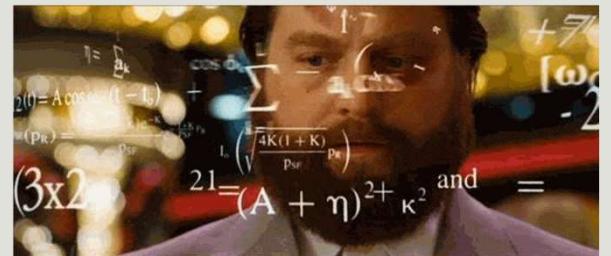
Enfant

```
import { Component } from
'@angular/core';
@Component({
  selector: 'app-root',
  template: `
    <h2> {{result}}</h2>
    <bind-output
      (valueChange)="showValue($event)"
    ></bind-output>
    ,
    styles: [``],
  )
}

export class AppComponent {
  title = 'app works !';
  result:any='N/A';
  showValue(value){
    this.result=value;
  }
}
```

Parent

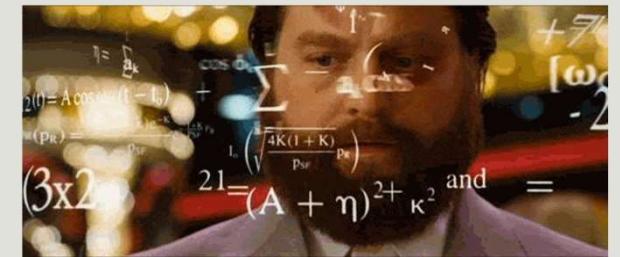
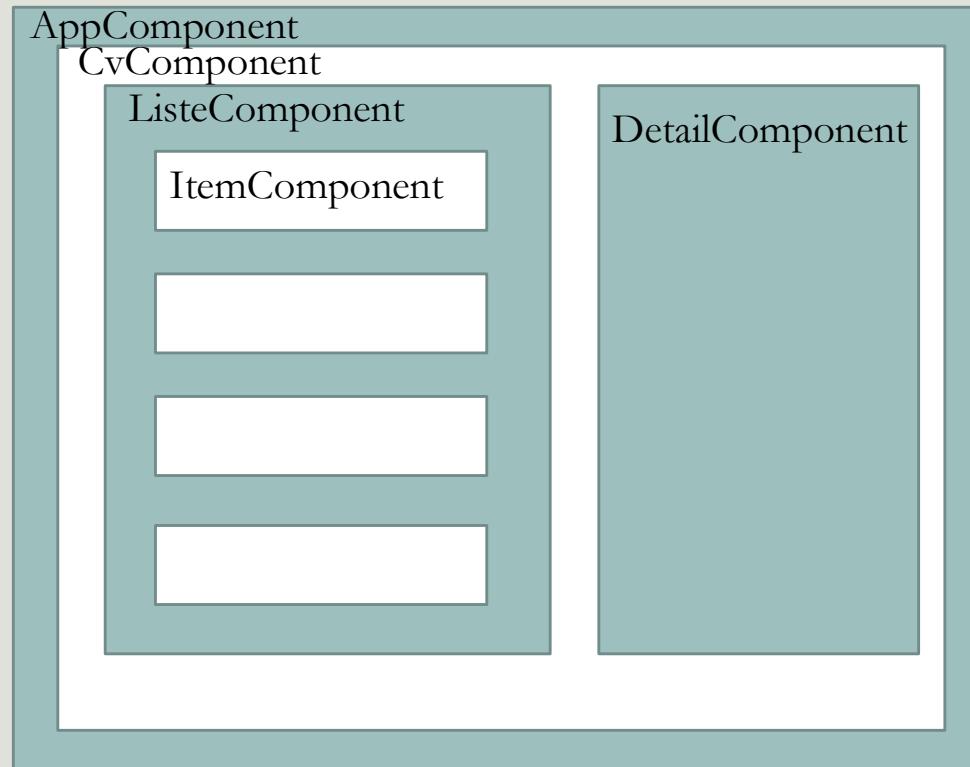
Exercice



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

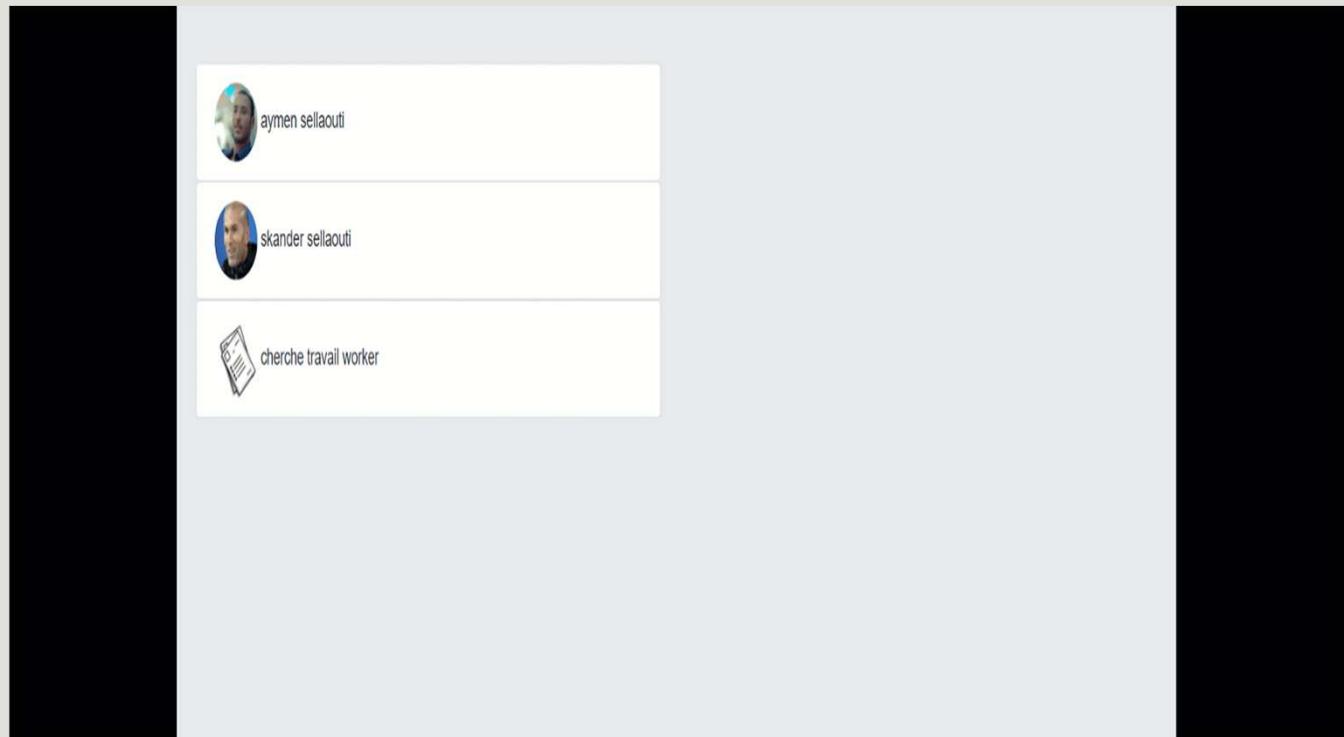
Exercice

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

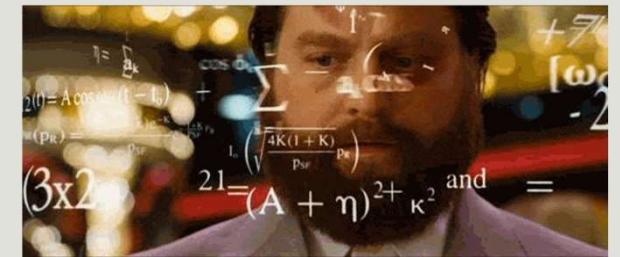
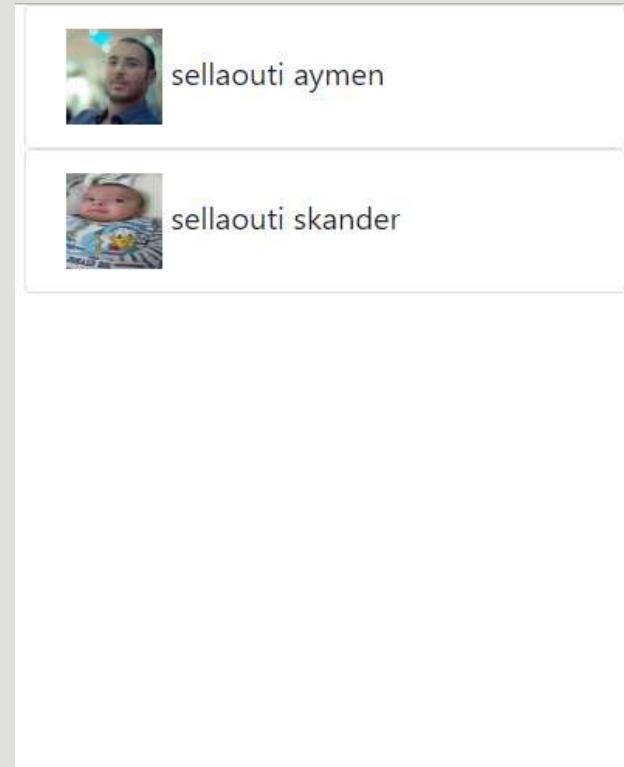
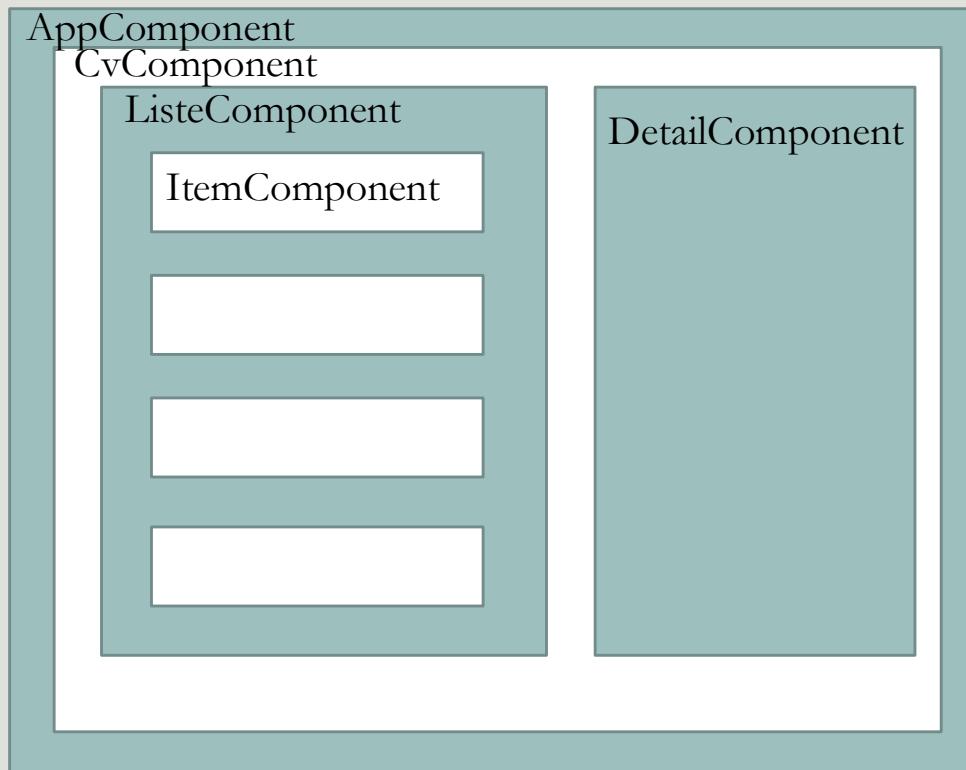


Exercice

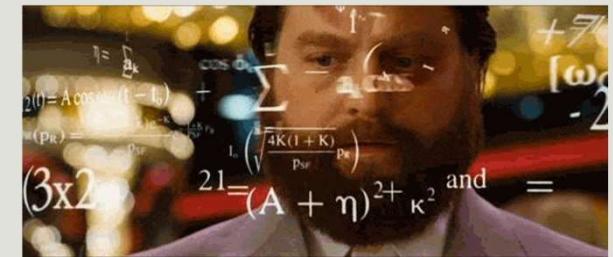
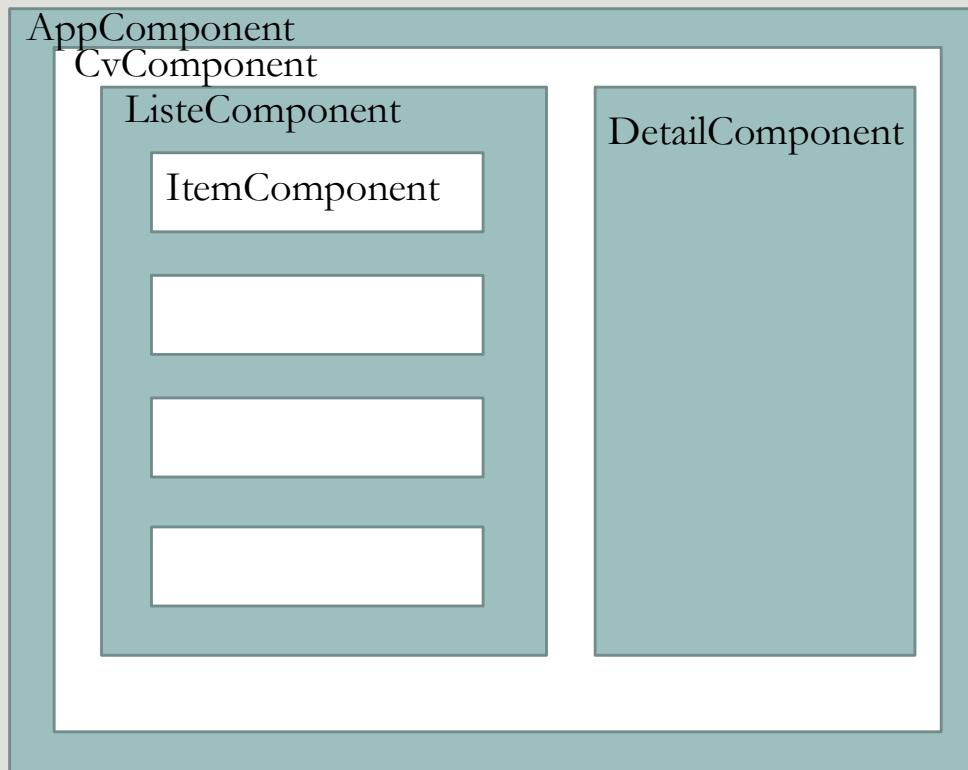
- Le but de cet exercice est de créer une mini plateforme de recrutement.
- La première étape est de créer la structure suivante avec une vue contenant deux parties :
 - Liste des Cvs inscrits
 - Détail du Cv qui apparaîtra au click
- Il vous est demandé juste d'afficher un seul Cv et de lui afficher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



Exercice



Exercice



The screenshot shows a mobile application interface. On the left, there is a list of users with their profile pictures and names: "sellaouti aymen" and "sellaouti skander". On the right, there is a larger view of a user profile for "Aymen Sellaouti", labeled as a "Teacher". Below the name is a circular profile picture of a man. At the bottom right of the screen, there is a button labeled "36" and a "Auto Rotation" icon.

Au click sur le Cv les détails sont affichés

Auto Rotation

Exercice

Un cv est caractérisé par :

id

name

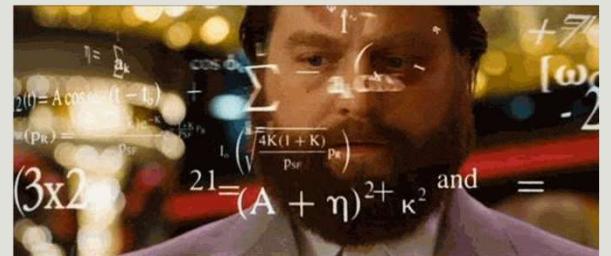
firstname

Age

Cin

Job

path



sellaouti aymen

sellaouti skander

Aymen Sellaouti
Teacher

36

Au click sur le Cv les détails sont affichés

Auto Rotation

50

Angular

Les directives

AYMEN SELLAOUTI

Objectifs

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

Qu'est ce qu'une directive

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

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



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

Qu'est ce qu'une directive

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

Les directives d'attribut (ngStyle)

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

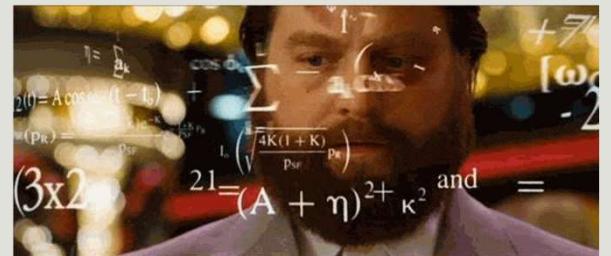
Les directives d'attribut (ngStyle)

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

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

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

Exercice



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

Test

30

arial

Les directives d'attribut (ngClass)

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

Les directives d'attribut (ngClass)

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

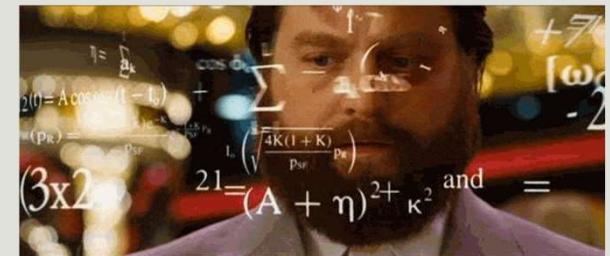
```
// Tableau
<div [ngClass]="['colorer', 'arrierman']"
  class="encadrer">
// Objet

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

Customiser un attribut directive

- Afin de créer sa propre « attribut directive » il faut utiliser un `HostBinding` sur la **propriété** que vous voulez **binder**.
 - Exemple : `@HostBinding('style.backgroundColor')`
`bg:string="red";`
- Si on veut associer un **événement** à notre directive on utilise un `HostListener` qu'on associe à un **événement** déclenchant une méthode .
 - Exemple : `@HostListener('mouseenter')` `mouseover() {`
`this.bg =this.highlightColor;`
`}`
- Afin d'utiliser le HostBinding et le HostListner il faut les importer du `core d'angular`

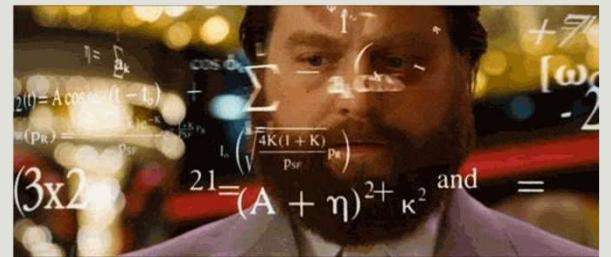
Exercice



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

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

Exercice



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

Styles Computed Event Listeners DOM Breakpoints »

Filter :hover .cls + □

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

Customiser une attribut directive

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

Les directives structurelles

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

Les directives structurelles *ngIf

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

Exemple

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

Les directives structurelles *ngFor

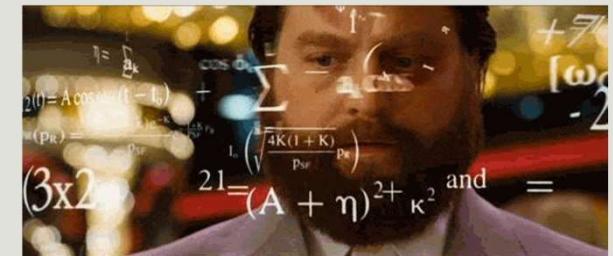
- Permet de répéter un élément plusieurs fois dans le DOM.
- Prend en paramètre les entités à reproduire.
- Fournit certaines valeurs :
 - index : position de l'élément courant
 - first : vrai si premier élément
 - last vrai si dernier élément
 - even : vrai si l'indice est paire
 - odd : vrai si l'indice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

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

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

Exercice (Notre Projet)

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



A screenshot of a web application titled 'CvTech'. The URL bar shows 'localhost:4200'. The page has a navigation bar with links like 'Home', 'Cv', 'Color', 'Task Manager', 'Poc', 'Add Students', and 'Login'. Below the navigation, there is a list of two CVs: 'Aymen Sellaouti' and 'Zineddine Zidan', each with a small profile picture. The rest of the page is mostly blank.

Angular

Les pipes

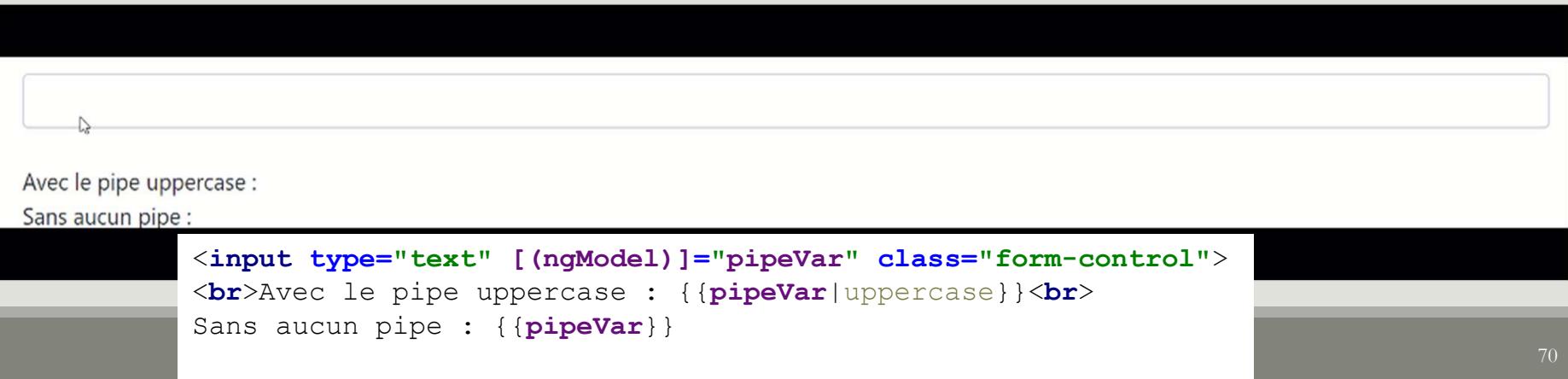
AYMEN SELLAOUTI

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 navigation bar with a back arrow and a search bar containing the text "Angular". Below the navigation bar is a header section with a logo and the word "Angular". The main content area contains a text input field with the placeholder "Type something...". Below the input field, the text "Avec le pipe uppercase :" is followed by the value "HELLO". Further down, the text "Sans aucun pipe :" is followed by the value "Hello".

```
<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 {

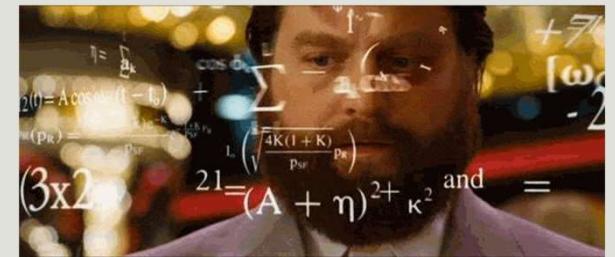
  transform(value: any, args?: any): any {
    switch (value) {
      case 'barca' : return ' blaugrana';
      case 'roma' : return ' giallorossa';
      case 'milan' : return ' rossoneri';
    }
  }
}
```

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

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

Exercice

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



Angular Service et injection de dépendances



AYMEN SELLAOUTI

Objectifs

1. Définir un service
2. INJECTION DE DÉPENDANCES (IOC)
 1. Principes
 2. Configurer son application
 3. L'injection de dépendances : type-based et hiérarchique
 4. Différents types de providers
3. Les services fournis
4. Injection de service

Qu'est ce qu'un service ?



- Un service est une classe qui permet d'exécuter un traitement.
- Permet d'encapsuler des fonctionnalités redondantes permettant ainsi d'éviter la redondance de code.

Component 1

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

Component 2

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

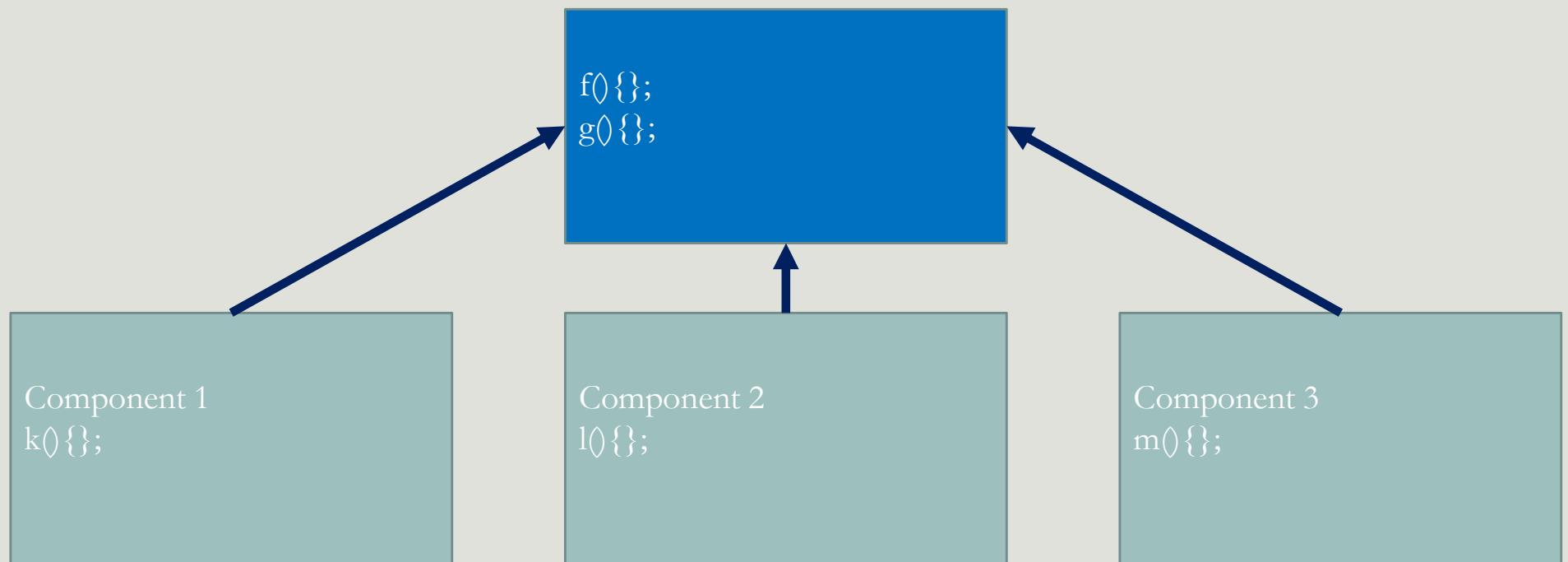
Component 3

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

Redondance de code

Maintenabilité difficile

Qu'est ce qu'un service ?



Qu'est ce qu'un service ?



- Un service est un médiateur entre la vue et la logique
- Fournit un ensemble de fonctionnalités

Qu'est ce qu'un service ?



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

Création d'un service

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

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 linstanciation de ces classes dans les différentes classes qui en dépendent?

Injection de dépendance (DI)



- Déléguer cette tache à une entité tierce.

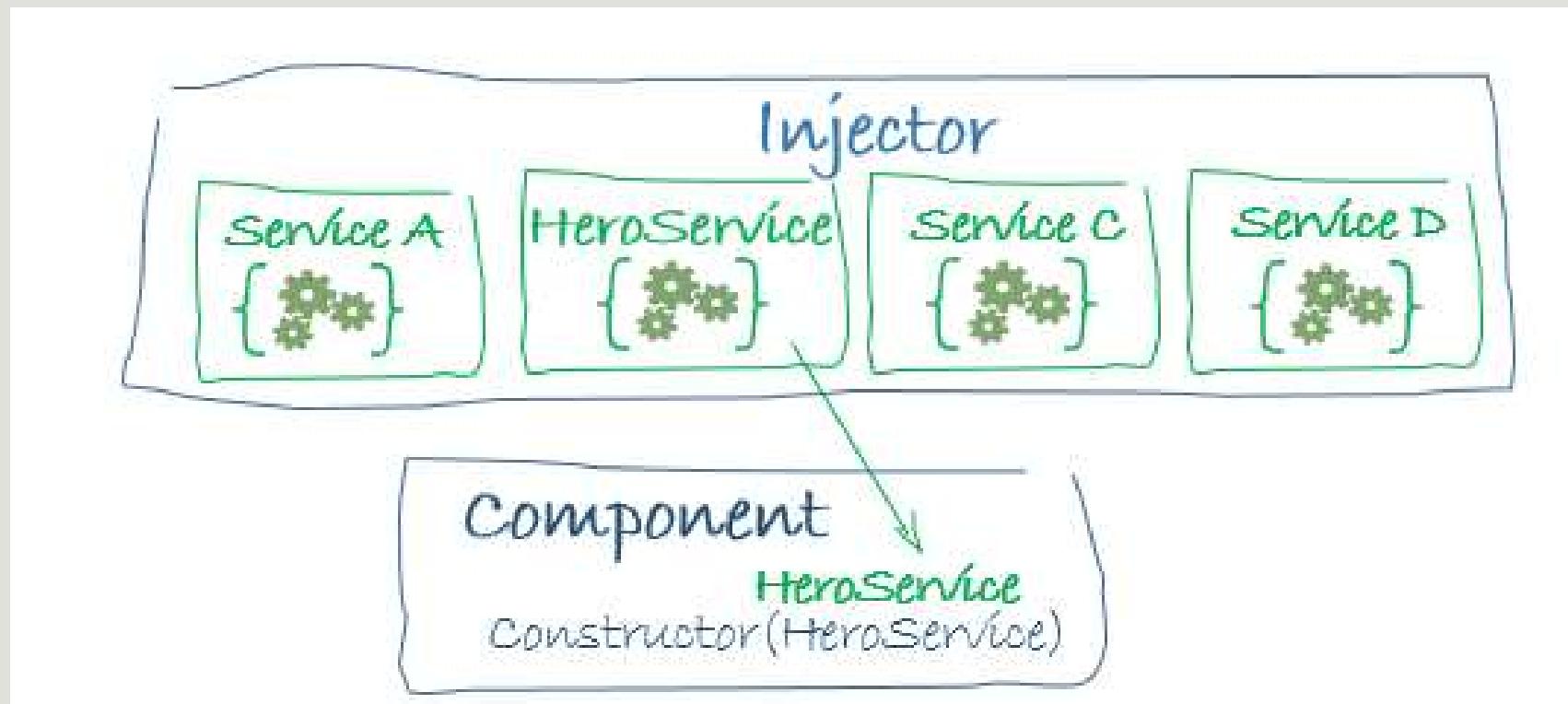
```
Classe A1{  
    Constructor(B b, C c)  
    ...  
}
```

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

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

INJECTOR

Injection de dépendance (DI)



Injection de dépendance (DI)

Que peut on injecter ?

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

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.

Injection de dépendance (DI)

Comment ca fonctionne

- Injection de **dépendance =>** Il nous faut donc une **dépendance**
- Afin de **Lier** cette **dépendance** au **système d'injection de dépendance d'Angular** nous devons **répondre à deux questions**
 - **Comment Angular va créer la dépendance ?**
 - **Quand est ce qu'Angular doit utiliser cette dépendance ?**
- 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**.

Injection de dépendance (DI)

Comment ça fonctionne

- Cette fonction peut être *créée implicitement par Angular* en utilisant quelques conventions simples (le cas le plus répondu) ou *par vous même*.
- 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.

Injection de dépendance (DI)

Comment ca fonctionne

```
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** ?
- Pour ce faire, vous pouvez utiliser des **Tokens**.
- 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 autre 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]  
  }  
]
```

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. 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**, 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) {}
```

@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.

Les providers personnalisés useClass

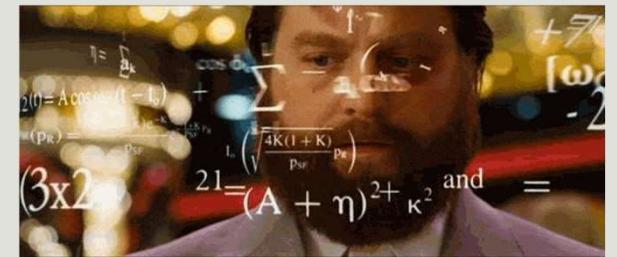
- En utilisant le décorateur **@Injectable**, votre Provider devient encore plus simple puisqu'on n'a plus à spécifier les dépendances qui seront directement déterminées au niveau du constructeur par le Système d'Injection de dépendance d'Angular

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

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

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

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

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 {}
```

Les providers personnalisés multi

- La plupart des dépendances de notre système correspondront **à une seule valeur**, comme par exemple une classe.
- Cependant, il y a des occasions où nous voulons avoir **plusieurs instances pour le même provider**.
- Pour ce faire, ajouter la clé **multi** et mettez la à **true**.
- Au lieu de recevoir **une instance**, vous recevrez **un tableau d'instance**.

```
const AuthenticationInterceptorProvider = {  
  provide: HTTP_INTERCEPTORS,  
  useClass: AuthenticationInterceptor,  
  multi: true  
};
```

Injection de dépendance (DI)

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

@Injectable({
  providedIn: 'root'
})
export class CvService {
  // Ce service est visible pour tout le monde
  constructor() { }

}
```

Injection de dépendance (DI)

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

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

@Injectable()
export class CvService {

  constructor() { }

}
```

Injection de dépendance (DI)

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

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

Chargement automatique du service

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

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

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

Chargement automatique du service providedIn

- La clé **providedIn** peut prendre les valeurs suivantes :
- **root** : L'injecteur au niveau de l'application, c'est ce que vous trouvez dans la plupart des applications.
- **platform** : Un injecteur de plateforme singleton spécial partagé par toutes les applications de la page.
- **any** : Fournit une **instance unique** dans chaque module chargé d'une manière **lazy** tandis que tous les modules chargés en **eager partagent une instance**. Cette option est obsolète.

```
providedIn?: Type<any> | 'root' | 'platform' | 'any' | null;
```

Avantage de l'utilisation du providedIn

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

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.

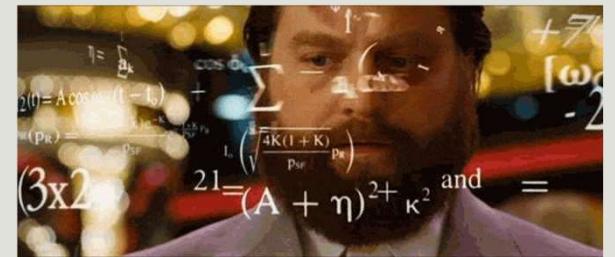
```
providers: [
  {
    provide: 'TODOS_LIST',
    useValue: [{lundi: 'nestJs'}, {mardi: 'Still NestJs'}],
  },
  TodoService,
],
```

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

```
@Controller('todo')
export class TodoController {
  @Inject('TODOS_LIST') todoList;
  constructor() {}
```

Exercice

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



Les providers personnalisés la fonction inject (avant 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**.

```
import {inject, InjectionToken, PLATFORM_ID} from "@angular/core";

export const WINDOW = new InjectionToken<Window>('Window bject', {
  providedIn: 'root',
  factory: () => {
    const platformId = inject(PLATFORM_ID);
    return platformId === 'browser' ? window : {} as Window;
  }
})
```

```
@Injectable({
  providedIn: 'root',
  useFactory: () => {
    const platformId = inject(PLATFORM_ID);
    return platformId === 'browser' ? window : {} as Window;
  }
})
export class WindowService{}
```

Les providers personnalisés la fonction inject (après Angular 14)

- A partir d'Angular 14, cette fonction **n'est plus limitée aux factory**.
- Vous pouvez maintenant **l'utiliser dans vos composants, directives et pipes**.
- Le premier intérêt est le **type safety**
- Il **facilite aussi l'héritage** en externalisant les dépendances de la classe.

DI Hiérarchique

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

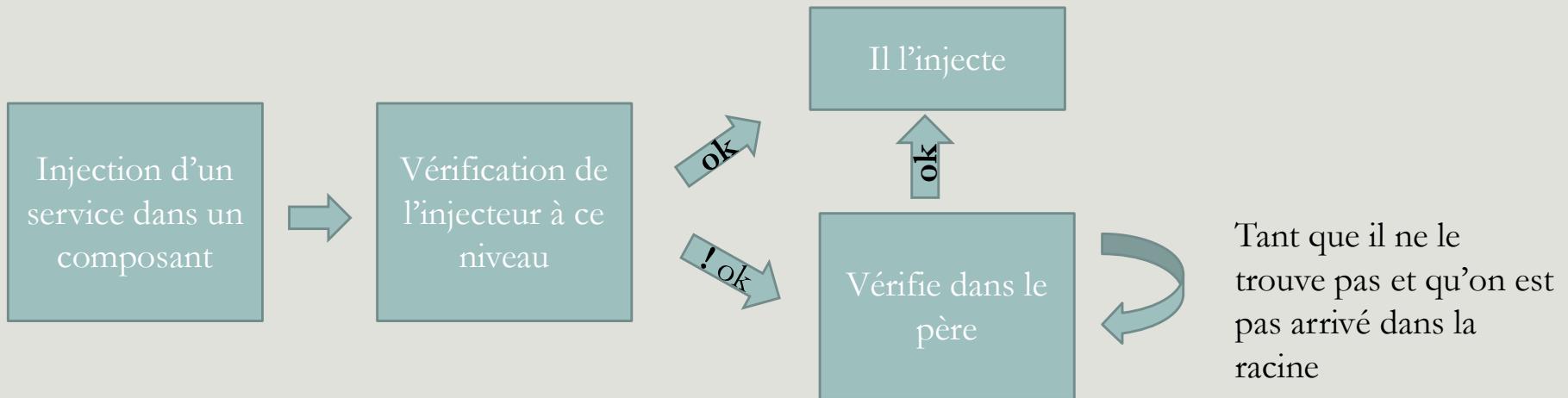
DI Hiérarchique

- Le système d'injection de dépendance d'Angular est **hiérarchique**
- Il possède **deux hiérarchie 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 :

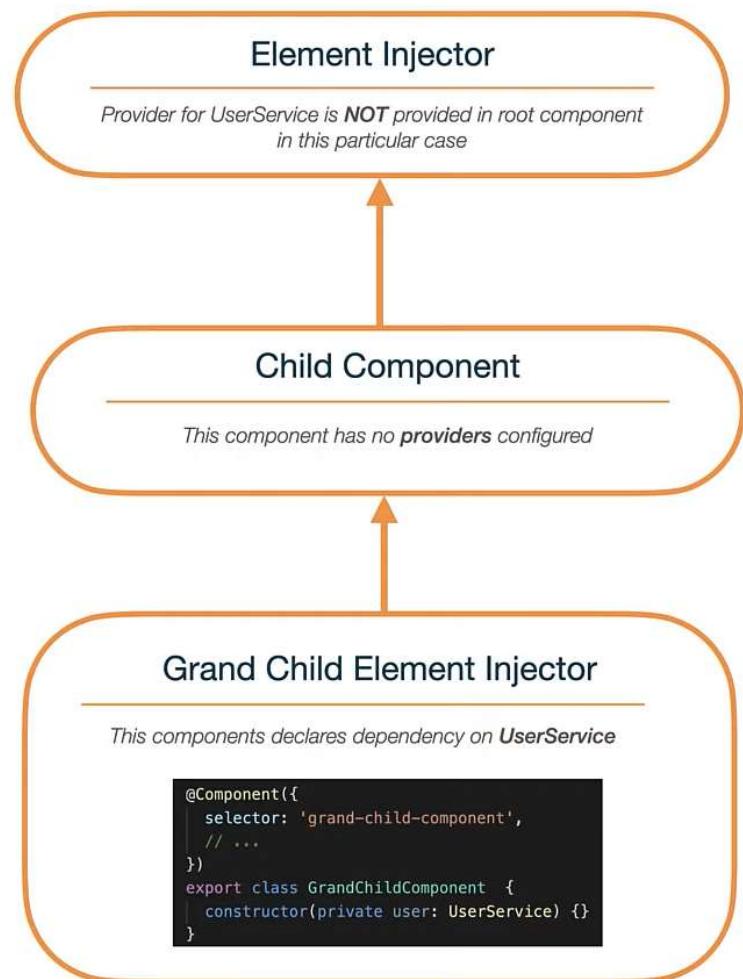


DI Hiérarchique

Hiérarchie d'injecteur niveau Module

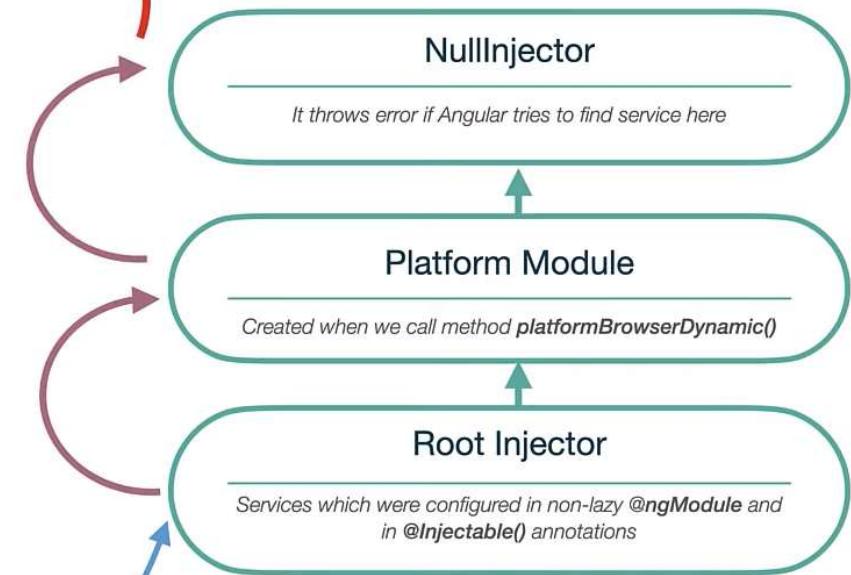
- Si Angular **ne trouve pas de provider** au niveau de la **hiérarchie des composants**, il va aller voir dans la **hiérarchie de module**.
- Le ModuleInjector peut être configuré de deux manières en utilisant :
 - La propriété `@Injectable` pour référencer un **NgModule**, ou '**root**'
 - Le tableau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

Element Injector Hierarchy



Error will be thrown

Module Injector Hierarchy



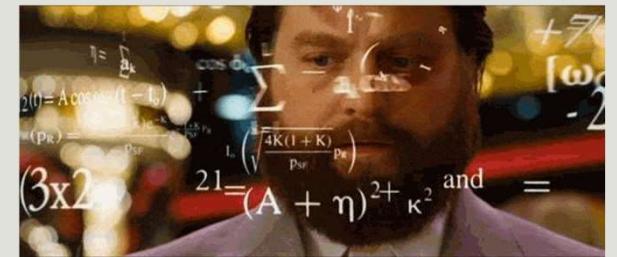
It throws error if Angular tries to find service here

Created when we call method `platformBrowserDynamic()`

Root Injector

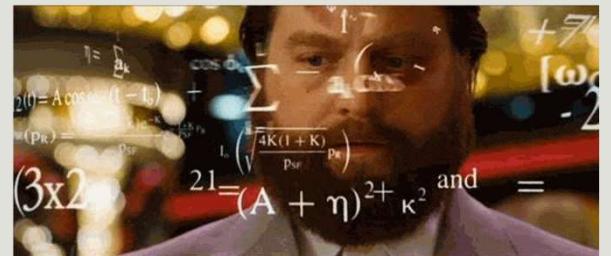
Services which were configured in non-lazy `@NgModule` and in `@Injectable()` annotations

Exercice



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

Exercice



 sellaouti aymen

 sellaouti skander

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

12345678

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

235 Followers | 114 Following | 35 Projects

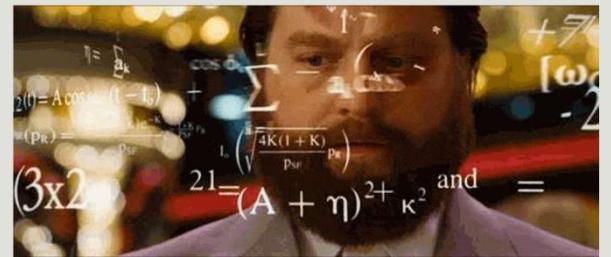
[Embaucher](#)

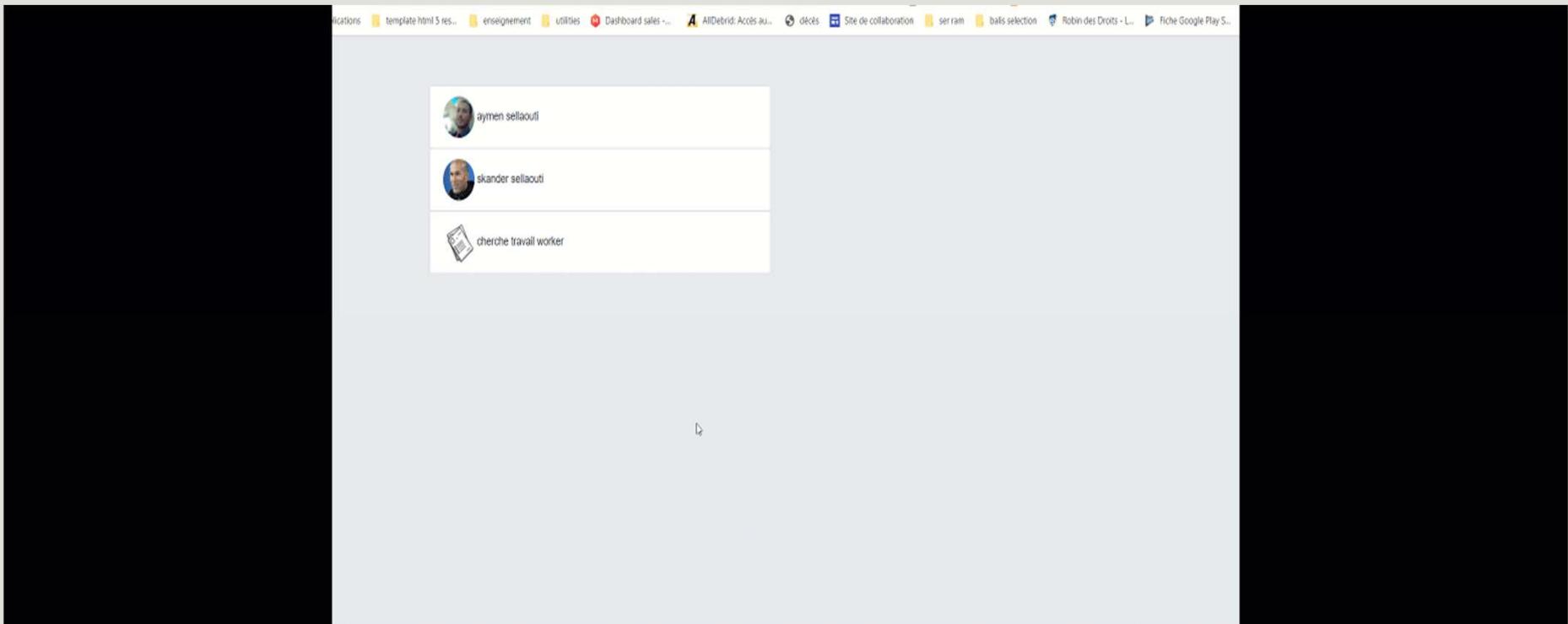
Liste des cvs sélectionnés pour embauche

aymen sellaouti



Exercice


$$x(t) = A \cos(\omega_n(t - t_0)) + \sum_{k=1}^{\infty} \frac{(-1)^k}{k!} e^{-\zeta_k t} \sin(\omega_k t)$$
$$\zeta_k = \sqrt{\frac{4K(1+K)}{p_E}}$$
$$(3x2) \rightarrow 21 = (A + \eta)^2 + \kappa^2 \text{ and } =$$



Angular Routing

AYMEN SELLAOUTI

Objectifs

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

Qu'est ce que le routing

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

Création d'un système de Routing

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

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

Création d'un système de Routing

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

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon"
  href="favicon.ico">
  <link rel="stylesheet"
  href="https://maxcdn.bootstrapcdn.com/bootstrap/
  3.3.7/css/bootstrap.min.css"
  integrity="sha384-
BVYiISIfE1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+P
mSTsz/K68vbDEjh4u"
  crossorigin="anonymous"></head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

1

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

2

App.routing.ts

Création d'un système de Routing

3. Créer la constante qui est un tableau d'objet de type `Routes` représentant chacun la route à décrire.
4. Intégrer les routes à notre application dans le `app` module à travers le `RouterModule` et sa méthode `forRoot`

Création d'un système de Routing

```
import {Route, RouterModule} from  
"@angular/router";  
import {CvComponent} from "./cv/cv.component";  
import {HeaderComponent} from  
"./header.component";  
  
const APP_ROUTES : Routes = [  
  {path: '', component:CvComponent},  
  {path:'onlyHeader', component:HeaderComponent}  
];  
  
export const ROUTING =  
RouterModule.forRoot(APP_ROUTES);
```

App.routing.ts

2

3

4

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

4

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

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

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

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

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

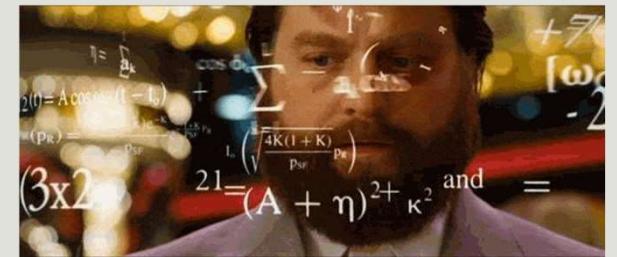
Syntaxe minimaliste d'une route

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

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

Exercice

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



Déclencher une route routerLink

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

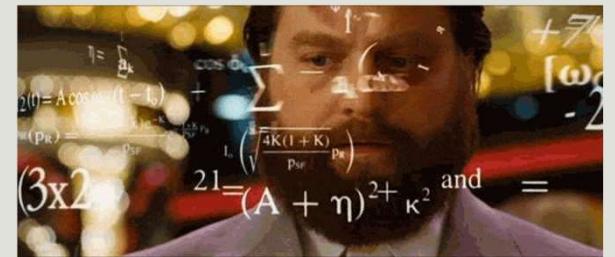
```
<li><a [routerLink]="'todo'" routerLinkActive="active">Gérer les  
cvs</a></li>
```

Déclencher une route routerLink

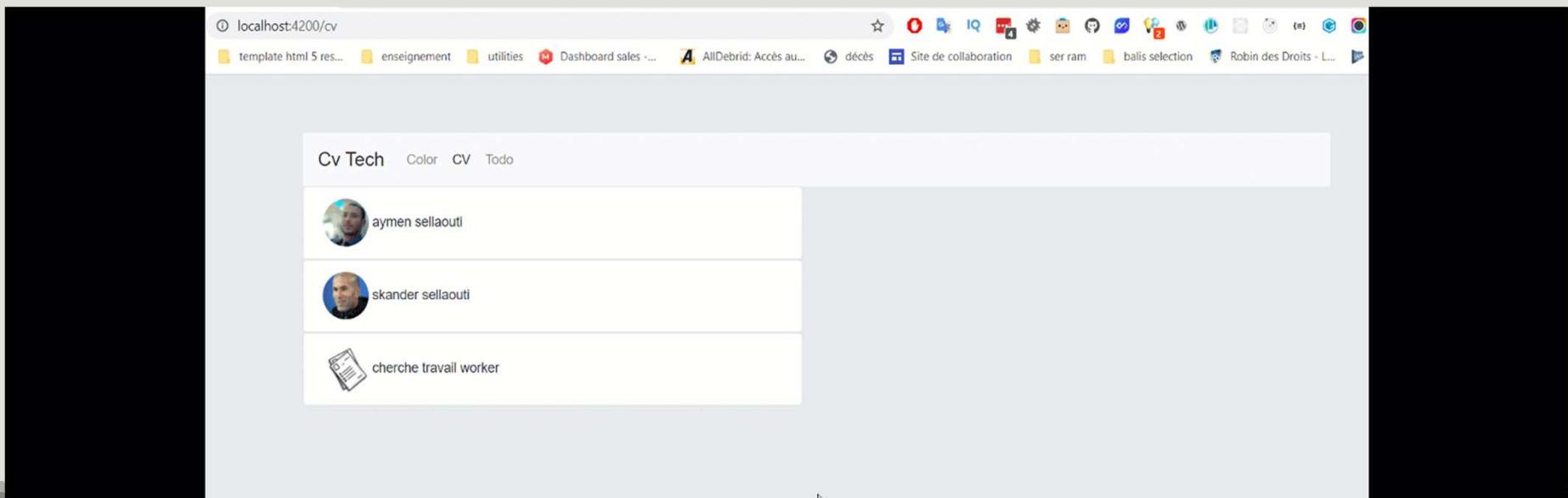
- **routerLinkActive="active"** va associer la classe active à l'uri cible ainsi qu'à tous ses ancêtres.
- Par exemple si on a l'uri 'cv/liste' la classe active sera ajouté à cet uri ainsi qu'à l'uri 'cv' et 'list'.
- Pour identifier uniquement l'uri cible, ajouter la directive suivante :

[routerLinkActiveOptions] = "{exact: true}"

Exercice



- Faites en sorte d'avoir un composant Header dans votre application qui permet d'afficher l'ensemble de vos liens.
- En cliquant sur un lien, le composant qui lui est associé doit être affiché.



Déclencher une route à partir du composant

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

Déclencher une route à partir du composant

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

Les paramètres d'une route

- Afin de spécifier à notre router qu'un segment d'une route est un paramètre, il suffit d'y ajouter ':' devant le nom de ce segment.
- Exemple
 - /cv/:id permet de dire que la root contient au début cv ensuite un paramètre de root appelé id.

Récupérer les paramètres d'une route

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

Récupérer les paramètres d'une route ActivatedRoute / snapshot

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

Récupérer les paramètres d'une route ActivatedRoute / snapshot

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

Récupérer les paramètres d'une route ActivatedRoute / snapshot

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

Récupérer les paramètres d'une route ActivatedRoute / snapshot

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

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

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

Passer le paramètre à travers le tableau de routerLink

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

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

Les queryParameters

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

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

Les queryParameters

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

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

Récupérer Les queryParameters

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

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

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

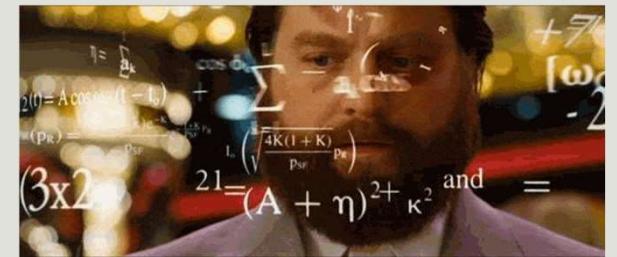
La route joker

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

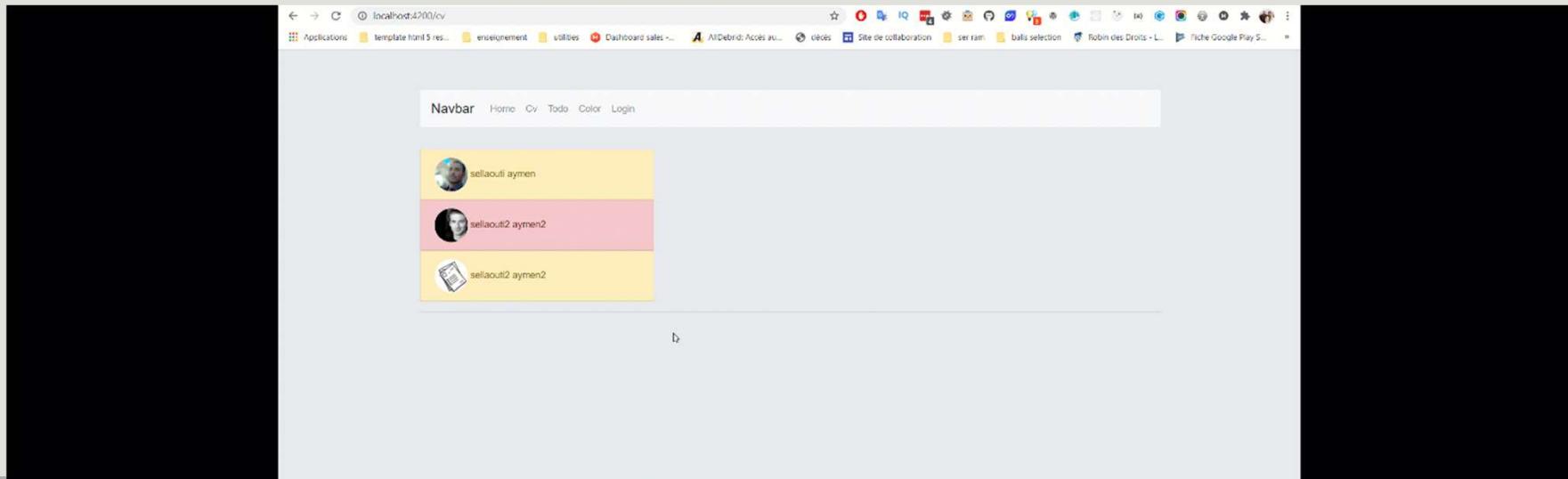
Exemple

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

Exercice



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



Angular Form

AYMEN SELLAOUTI

Approche de gestion de FORM

1. Approche basée Template
2. Approche réactive

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
- 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

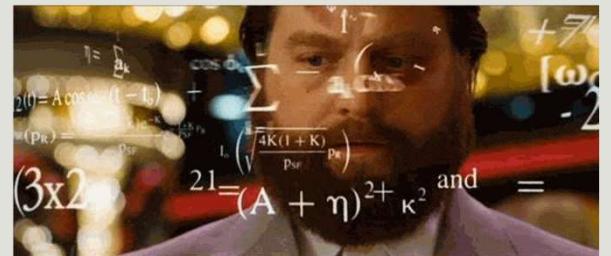
La propriété valid de ngForm permet de vérifier si le formulaire est valid ou non en se basant sur les validateurs qu'ils contient.

Approche basée Template NgForm

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

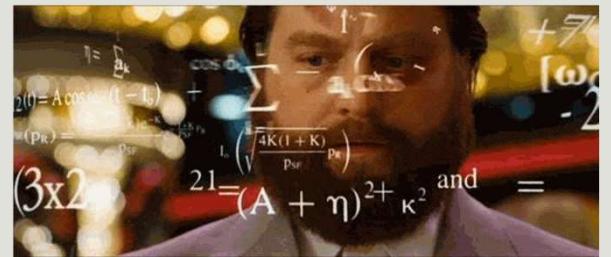
- **dirty** : informe sur le fait que l'une des propriétés du formulaire a été modifié ou non
- **Valid / invalid** : informe si le formulaire est valide ou non
- **Untouched / touched** : informe si le formulaire est touché ou non
- **pristine** : le formulaire n'a pas été touché, c'est l'opposé du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

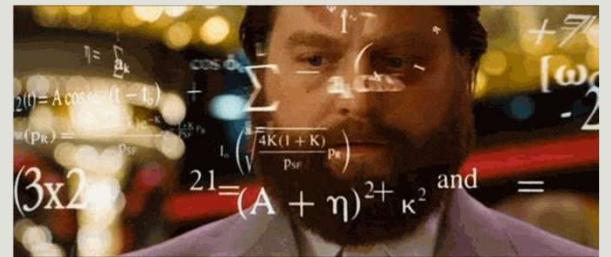
A screenshot of a web browser window showing a login form. The URL bar indicates the page is at `localhost:4200/login`. The browser's toolbar is visible at the top. The main content area contains a navigation bar with links to Home, Cv, Todo, Color, and Login. Below the navigation bar is a form with two input fields labeled "Email:" and "password:", each with a clear icon. A green "Login" button is positioned below the password field. The background of the browser window is black.

Approche basée Template

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

- Pour accéder à l'objet form et ces propriétés nous avons utilisé `#notreForm=«ngForm»`
- Pour les champs du formulaire c'est la même chose mais au lieu du ngForm c'est un **ngModel**
`#notreChamp=« ngModel »`

Exercice



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

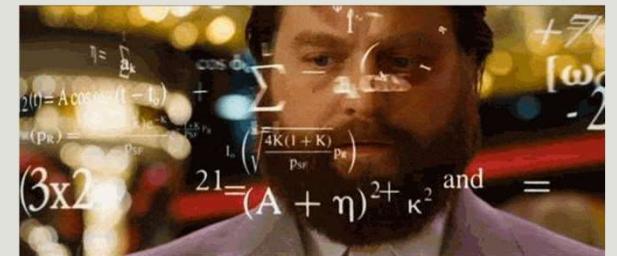
A screenshot of a web browser window showing a login form. The URL bar indicates the page is `localhost:4200/login`. The form has four fields: 'email', 'password', and two redacted sections on the left and right sides. Below the form is a green 'Login' button. The browser's toolbar and various icons are visible at the top.

Approche basée Template

Associer des valeurs par défaut aux champs

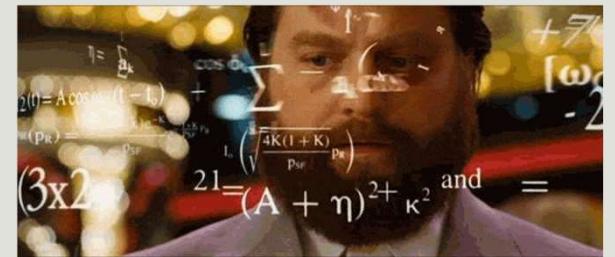
- Pour associer des valeurs par défaut aux champs d'un formulaire associé à Angular il faut le faire à partir du composant.
- Afin de gérer les valeur du formulaire à partir du composant il faut du binding.
- Au lieu d'avoir juste la primitive ngModel associée au contrôle d'un élément on ajoute le **property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

Exercice

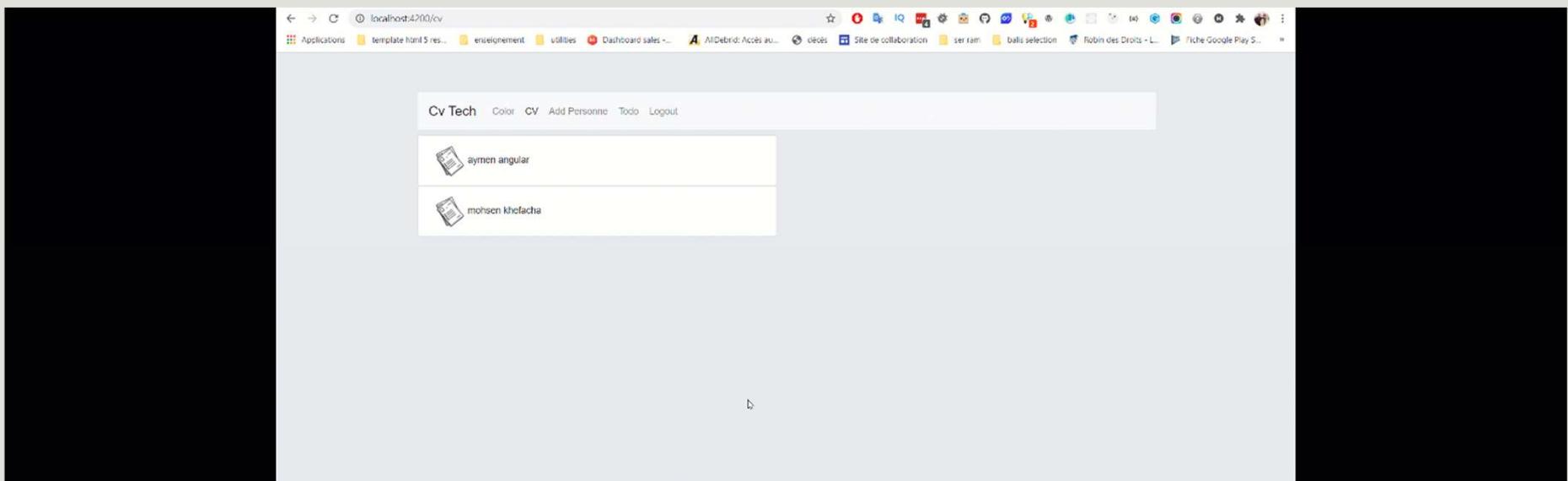


- Ajouter la valeur par défaut « myUserName » au champ username.

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

AYMEN SELLAOUTI

HTTP

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

Le Module HttpClient

Programmation Asynchrone

Programmation non bloquante.

Les promesses

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

(https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Utiliser_les_promesses)

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

Promesse

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

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

Qu'est ce que la programmation réactive

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

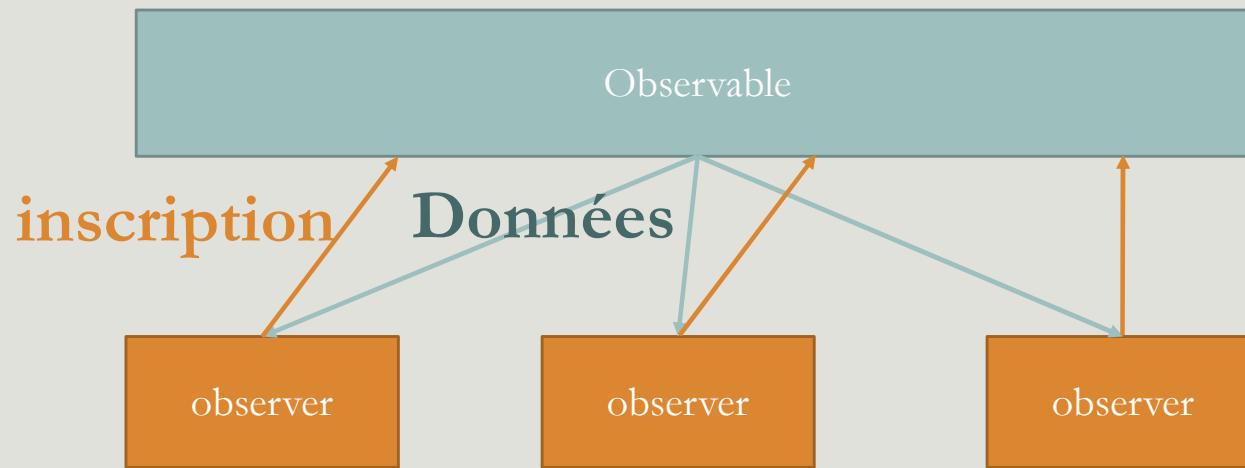
Programmation reactive =

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

Le pattern « Observer »

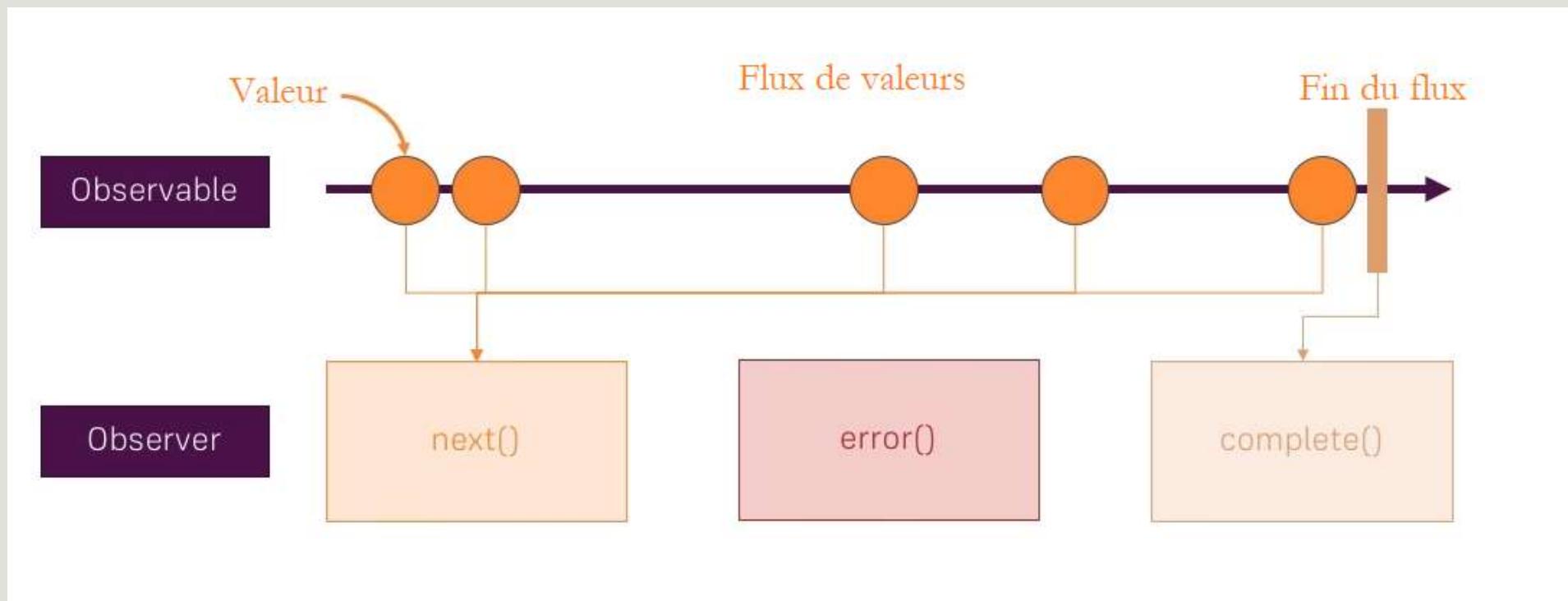
- Le patron de conception **Observateur** permet à un objet de garder la trace d'autres objets, intéressés par l'état de ce dernier.
- Il définit une relation entre objets de type un-à-plusieurs.
- Lorsque l'état de cet objet **change**, il **notifie** ces **observateurs**.

Observables, Observers et subscriptions



traitement

Fonctionnement



Promesse Vs Observable

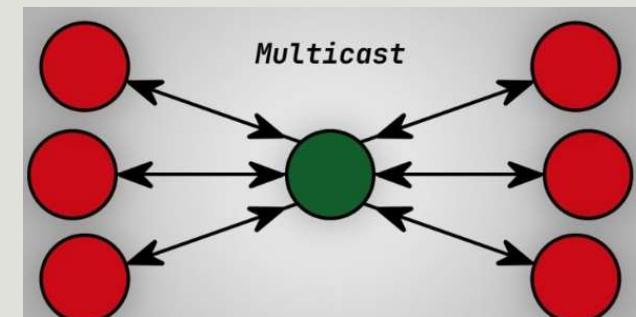
| Promesse | Observable |
|--|---|
| Un promesse gère un seul événement | Un observable gère un « flux » d'événements. |
| Non annulable. | Annulable. |
| Traitement immédiat. | Lazy : le traitement n'est déclenché qu'à la première utilisation du résultat. |
| Deux méthodes uniquement (then/catch). |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

Observable

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

Hot Vs Cold Observable

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



asyncPipe

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

Les opérateurs de l'observable

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

Quelques opérateurs utiles de l'Observable

map

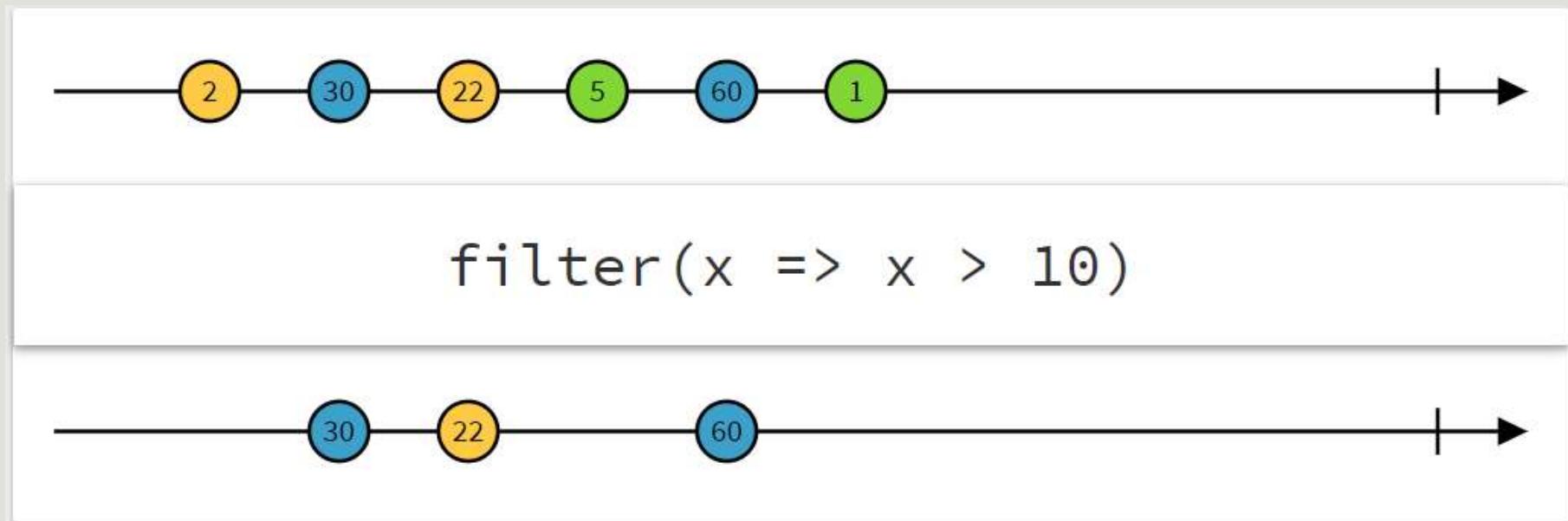


`map(x => 10 * x)`

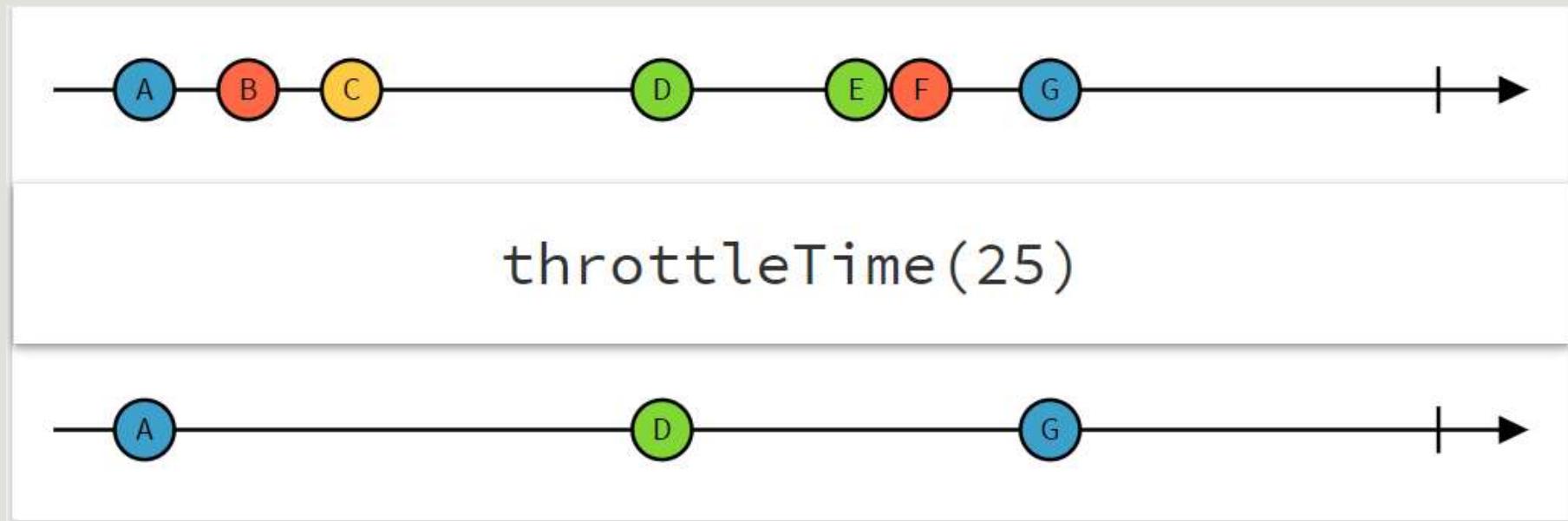


Quelques opérateurs utiles de l'Observable

filter



Quelques opérateurs utiles de l'Observable



Quelques opérateurs utiles de l'Observable

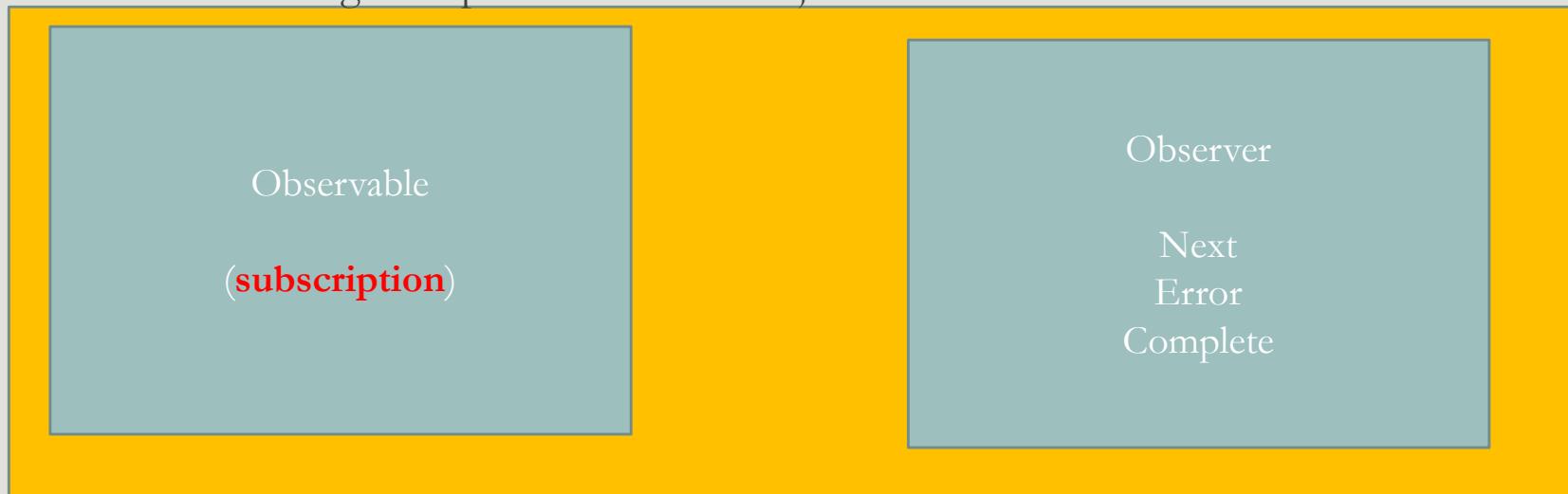
<https://angular.io/guide/rx-library>

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

<http://rxmarbles.com/>

Les subjects

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

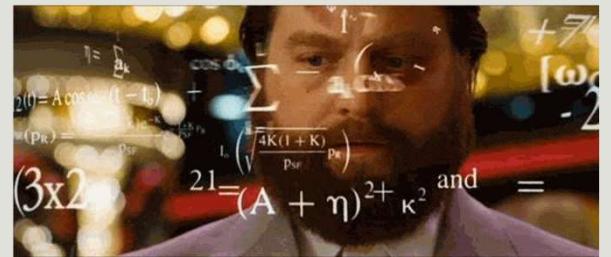


Les subjects



Exercice

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



Installation de HTTP

- Le module permettant la consommation d'API externe s'appelle le HTTP MODULE.
- Afin d'utiliser le module HTTP, il faut l'importer de @angular/common/http (@angular/http dans les anciennes versions) `import {HttpClientModule} from "@angular/common/http";`
- Il faudra aussi l'ajouter dans le fichier module.ts dans le tableau d'imports.

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

Installation de HTTP

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

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

Interagir avec une API Get Request

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

Interagir avec une API Get Request

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

Interagir avec une API POST Request

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

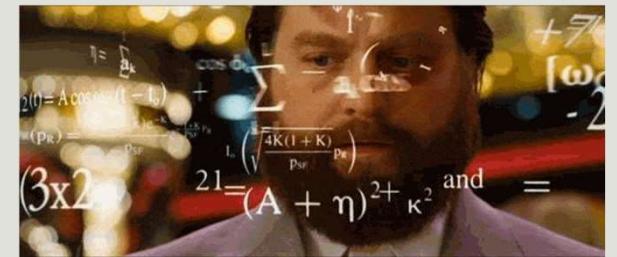
Interagir avec une API POST Request

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

Documentation

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

Exercice



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

Les headers

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

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

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

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

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

Les paramètres

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

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

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

Authentification

How does Authentication work?



Ajouter le token dans la requête

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

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

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

Ajouter le token dans la requête

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

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

Sécuriser vos routes

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

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

Guard

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

Guard / canActivate

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

Guard / canActivate

1

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

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor() {
  }
  // route contient la route appelé
  // state contiendra le futur état du routeur de l'application qui devra passer la validation du guard
  // https://vsavkin.com/routeur-angular-comprendre-1%C3%A9tat-du-routeur-5e15e729a6df
  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {
    if (// your condition) {
      return true;
    }
    return false;
  }
}
```

Guard / canActivate

2

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

App.module.ts

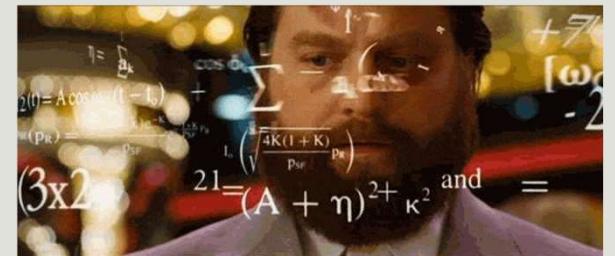
Guard / canActivate

3

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

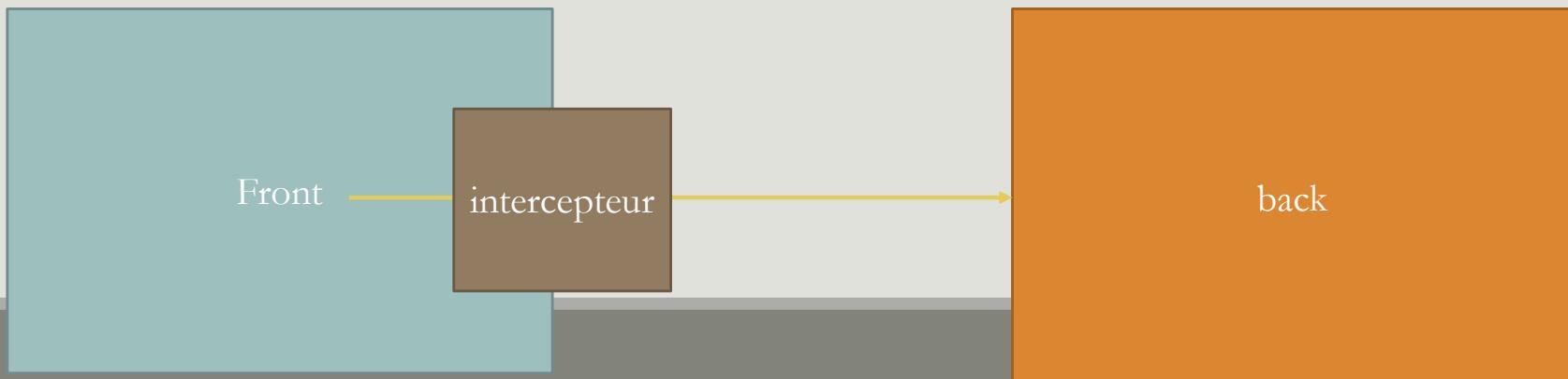
Exercice

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



Les intercepteurs

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



Les intercepteurs

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

Les intercepteurs

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

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

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

Les intercepteurs : changer la requête

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

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

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

Cloner une requête

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

Aller plus loin avec RxJS

AYMEN SELLAOUTI

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** 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 |

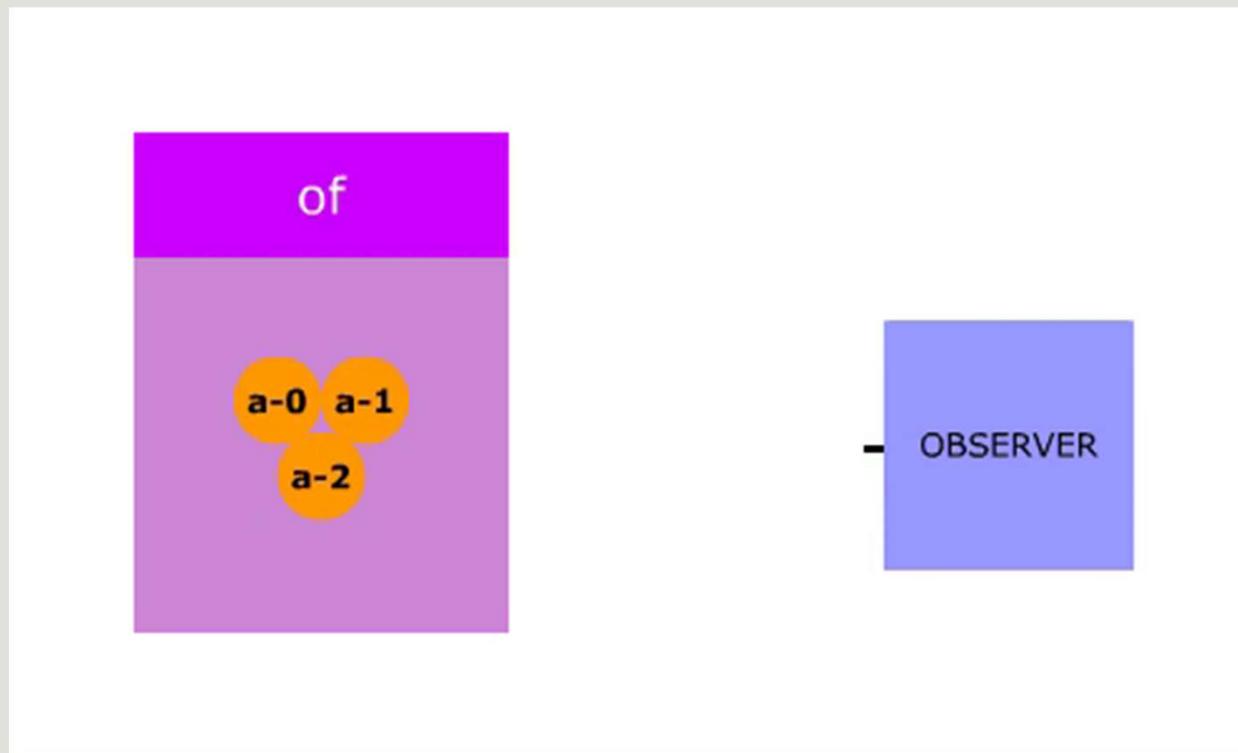
Les opérateurs de création of

- 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

tap

- L'opérateur **tap** est utilisé pour gérer les effets de bord.
- En effet, il n'a aucun effet sur le flux de données et il vous permet de faire des opérations mais sans toucher au flux.

```
from([1, {name: 'aymen'}, 3])
    .pipe(
        tap((data) => console.log(data))
    )
)
```

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'événement.
- 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.

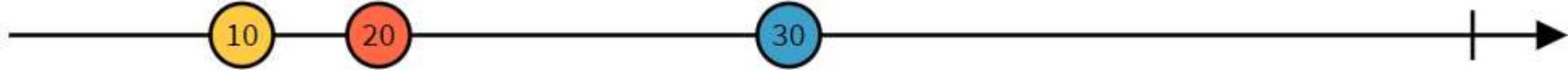
```
export class FromEventComponent implements AfterViewInit {  
  @ViewChild('btn') button!: ElementRef;  
  onClickButton$: Observable<Event>;  
  ngAfterViewInit() {  
    this.onClickButton$ = fromEvent(this.button.nativeElement, 'click');  
  }  
}
```

Quelques opérateurs utiles de l'Observable opérateur pipable

map



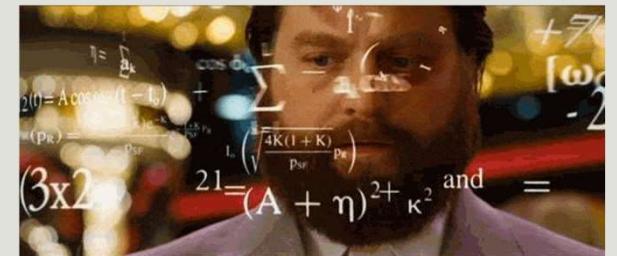
`map(x => 10 * x)`



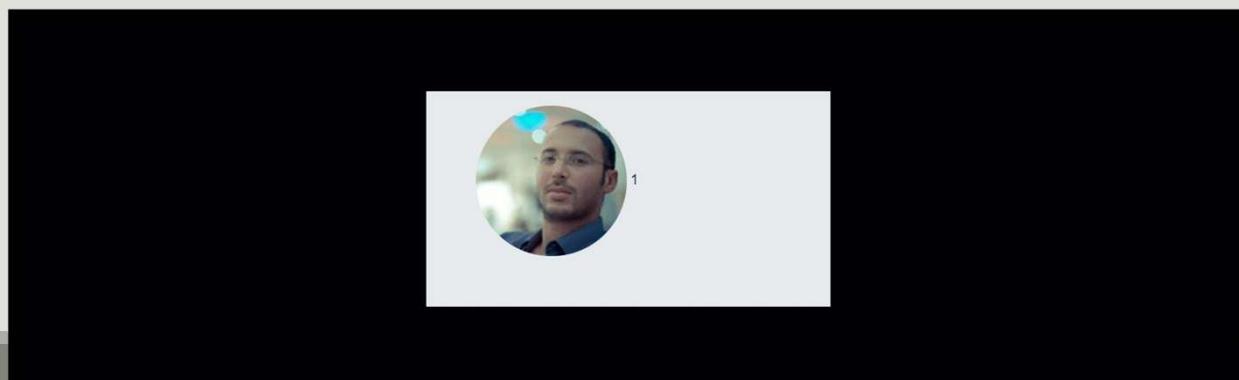
asyncPipe

- **async** Pipe est un pipe qui permet d'afficher directement les valeurs émises par 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.

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.



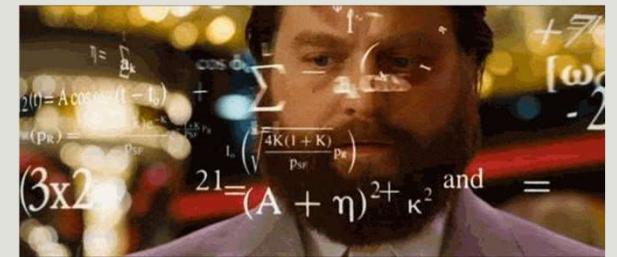
asyncPipe

L'opérateur as

- Si vous utilisez le même observable plusieurs fois dans votre template vous serez amené à réutiliser `async` plusieurs fois.
- Afin d'éviter ça, vous pouvez utiliser l'opérateur **as** qui vous permettra de récupérer le résultat émis par votre source observable dans une variable que vous pourrez utiliser plusieurs fois.

```
<ng-container *ngIf="data$ | async as data">  
  
  </ng-container>
```

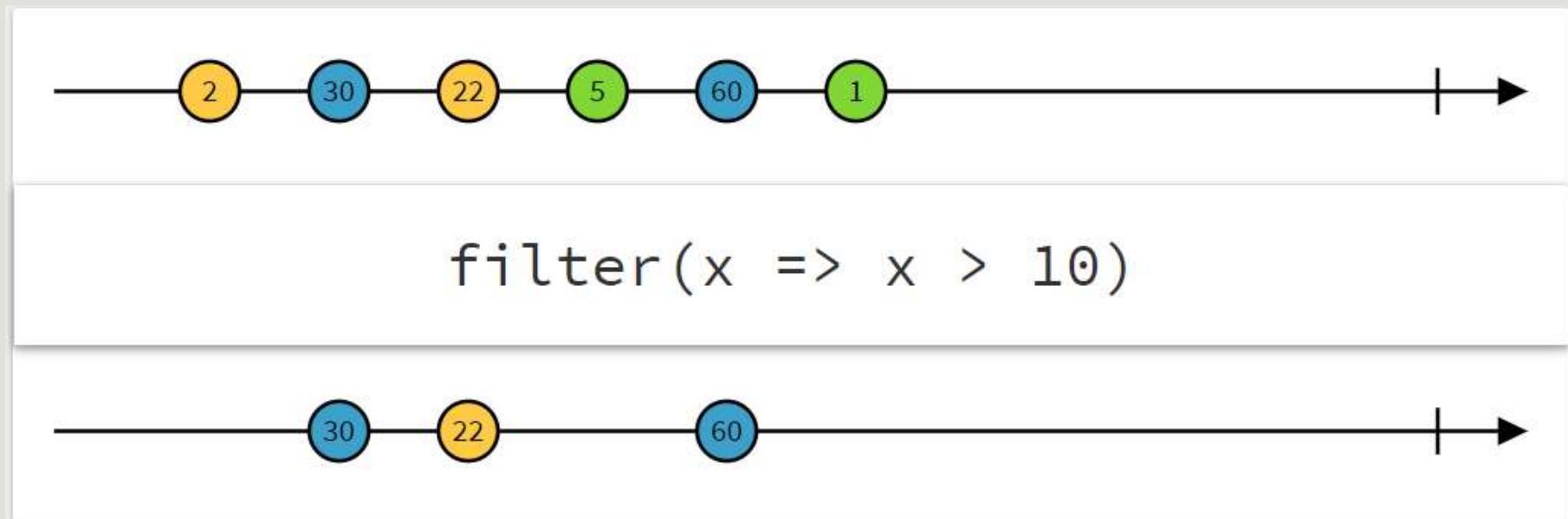
Exercice



- Utilisez le `asyncPipe` afin de réécrire le **DetailsCvComponent**
- Ps : Ignorer pour le moment la partie Gestion des erreurs on y reviendra après.

Quelques opérateurs utiles de l'Observable opérateur pipable

filter



Quelques opérateurs utiles de l'Observable

opérateur pipable

take

- L'opérateur **take** prend des valeurs de la source observable **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 **complete** à l'observateur.

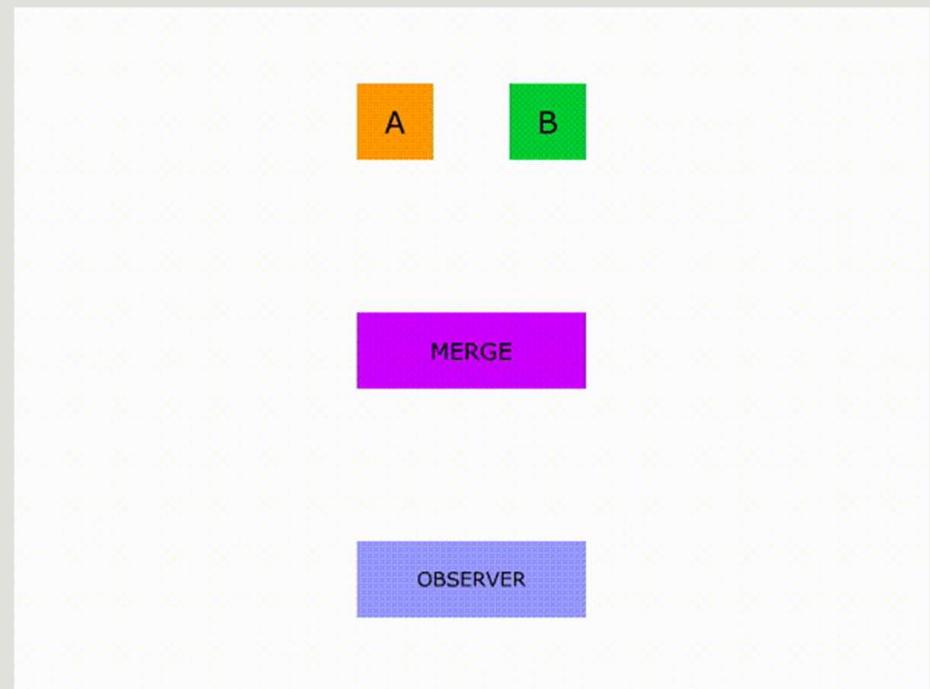


Quelques opérateurs utiles de l'Observable

opérateur pipable

merge

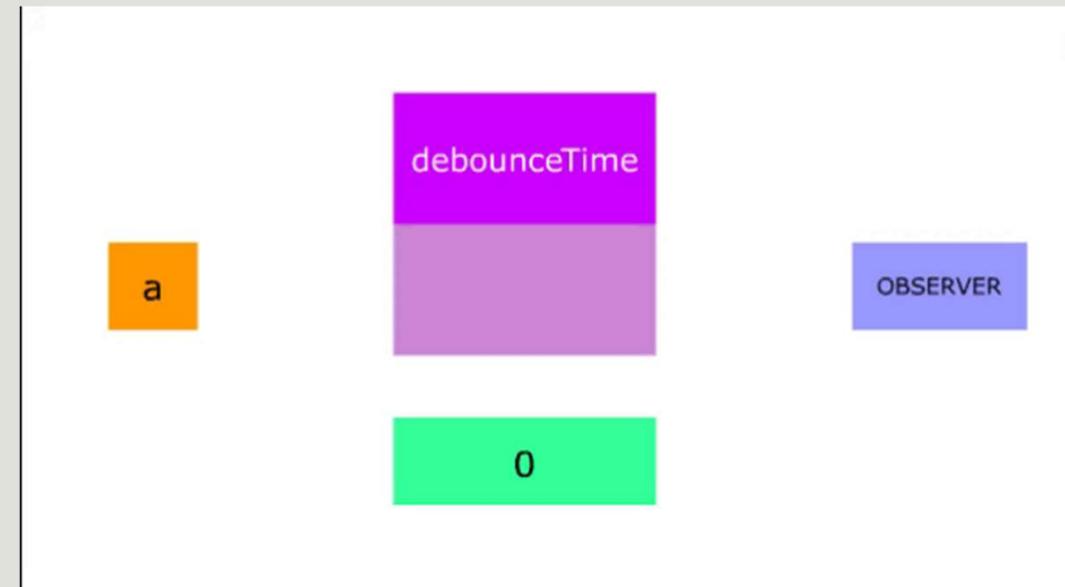
- **merge** combine un certain nombre de flux observables et émet simultanément toutes les valeurs de chaque flux d'entrée donné.
- Au fur et à mesure que les valeurs d'une séquence combinée sont produites, ces valeurs sont émises dans le flux résultant.
- Utilisez cet opérateur si vous n'êtes pas concerné par l'ordre des émissions de flux et si vous êtes simplement intéressé par toutes les valeurs provenant de plusieurs flux combinés comme si elles étaient produites par un seul flux.



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

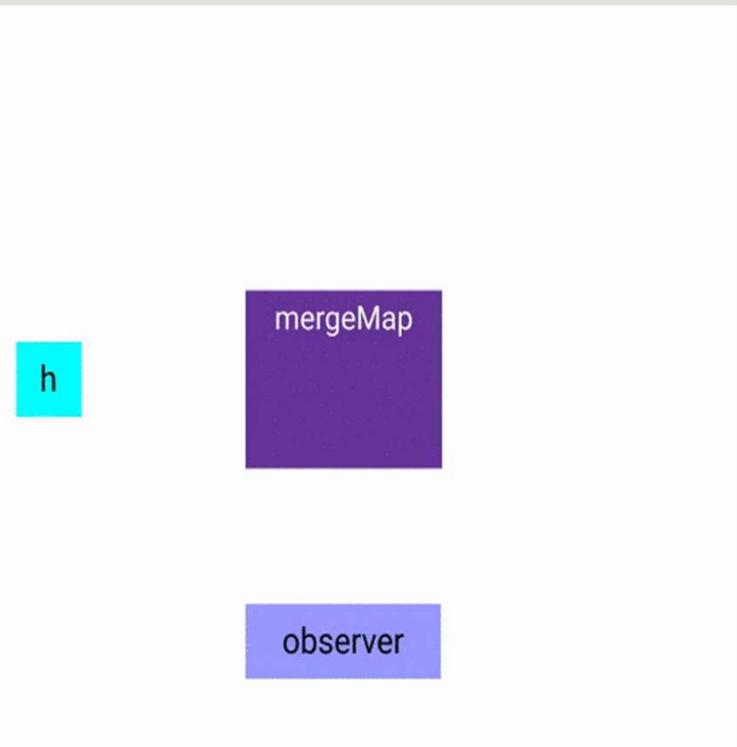
MergeMap

- 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'aplatissement/ 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);
  });
}
```

| observable | obs1 | 1 |
|------------|------|---|
| observable | obs2 | 1 |
| observable | obs1 | 2 |
| observable | obs1 | 3 |
| observable | obs2 | 2 |
| observable | obs1 | 4 |
| 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(
    mergeMap((param) => thisrxjsService.timerObs(param.timer,
param.name))
  ).subscribe(
    (data) => console.log(data)
  )
}
```

Les opérateurs d'applatissement/ flattening operators

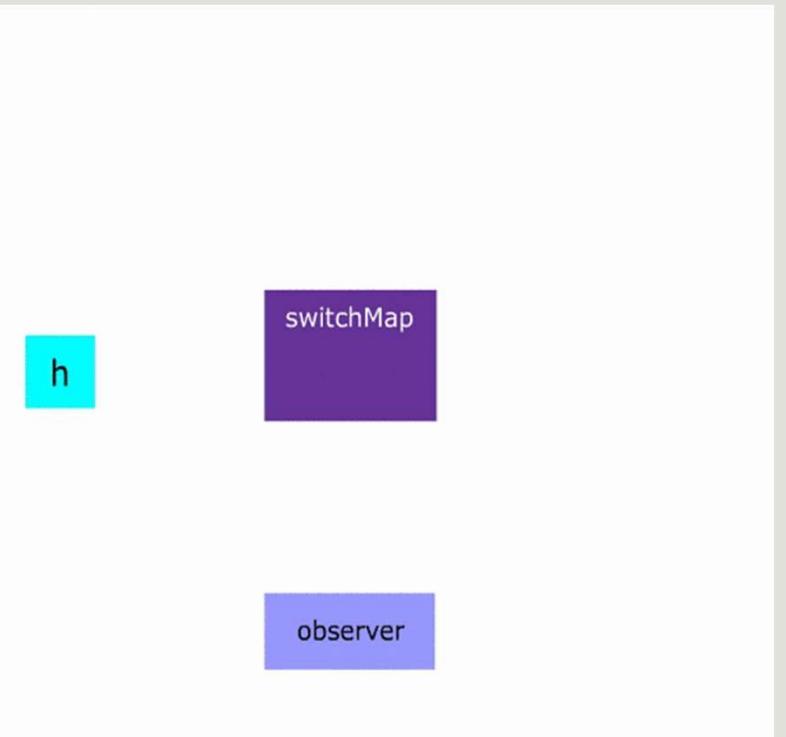
SwitchMap

- 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'aplatissement/ 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 et 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) => this.rxsService.timerObs(param.timer,
    param.name))
  ).subscribe(
    (data) => console.log(data)
  )
}
```

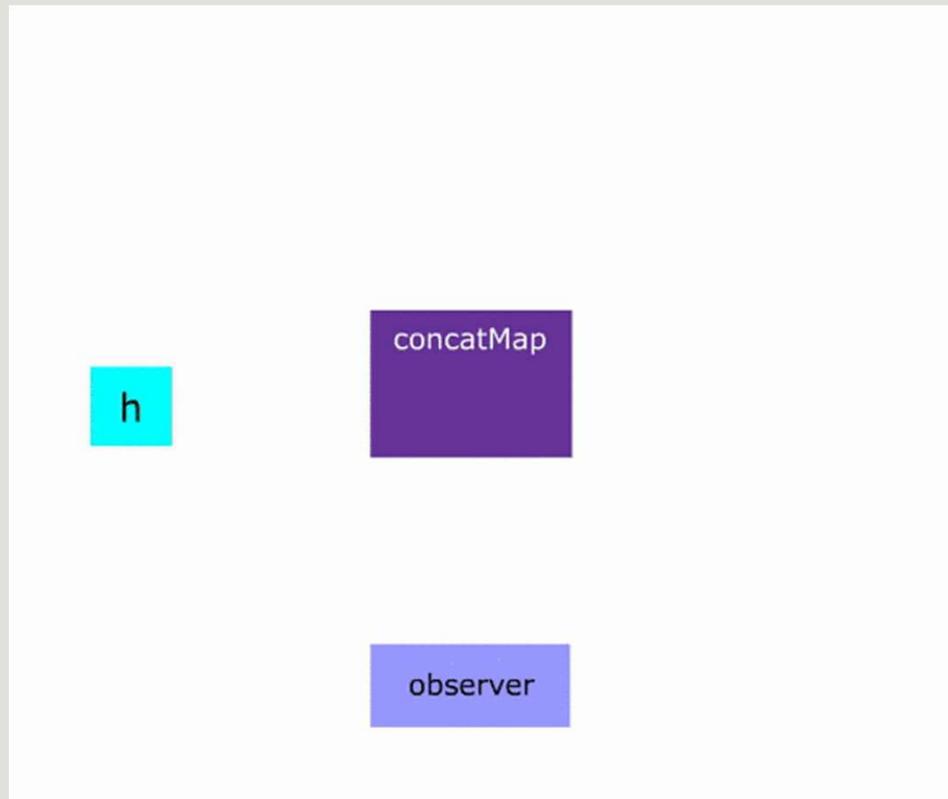
```
observable obs2 1
observable obs2 2
observable obs2 3
observable obs2 4
```

Les opérateurs d'applatissement/ flattening operators

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



<https://indepth.dev/reference/rxjs/operators/>

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 |

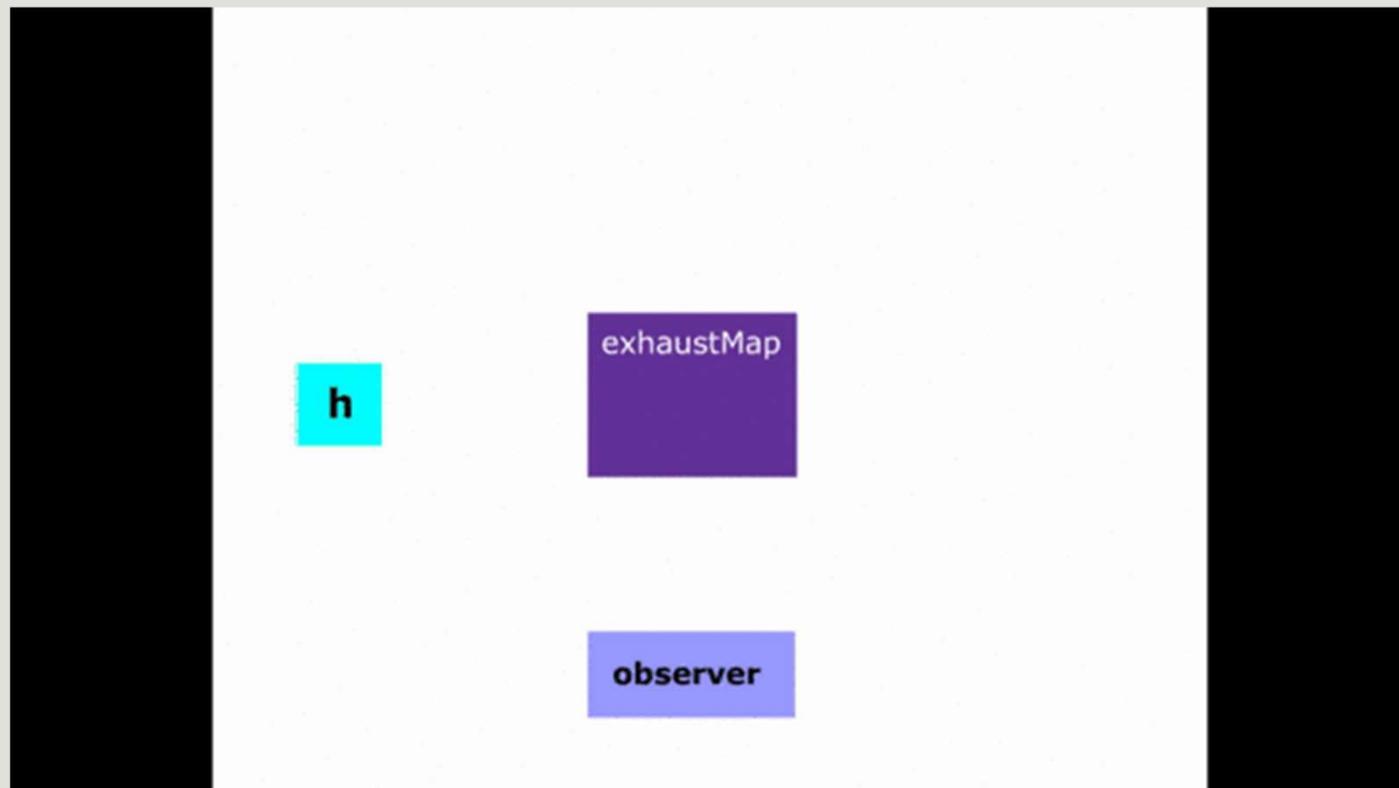
```
params = [
  {name: 'obs1', timer: 1000, iteration: 4 },
  {name: 'obs2', timer: 1500, iteration: 4 },
];
constructor(private rxjsService: RxjsService) {
  from(this.params).pipe(
    concatMap((param) => thisrxjsService.timerObs(param.timer,
param.name))
  ).subscribe(
    (data) => console.log(data)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

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

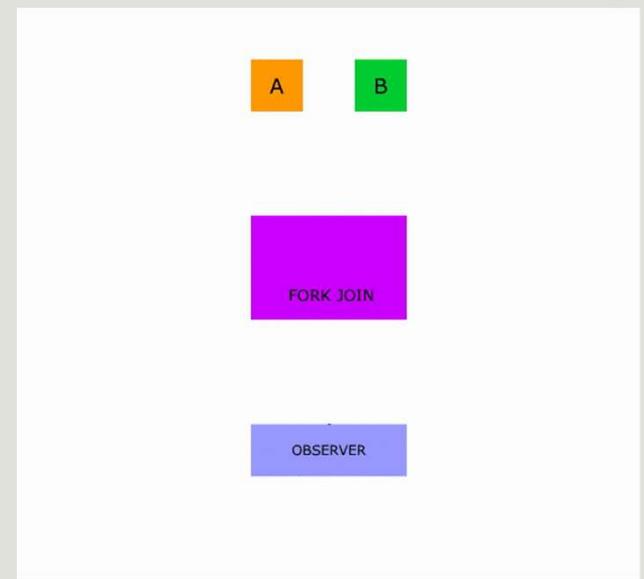
```
timerObs(timer: number, name: string, iteration = 4) {
  return new Observable((observer: Observer<string>) => {
    let i = 0;
    const x = setInterval(() => {
      if (i >= iteration) {
        observer.complete();
        clearInterval(x);
      }
      observer.next(`observable ${name} ${++i}`);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

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 erreur

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

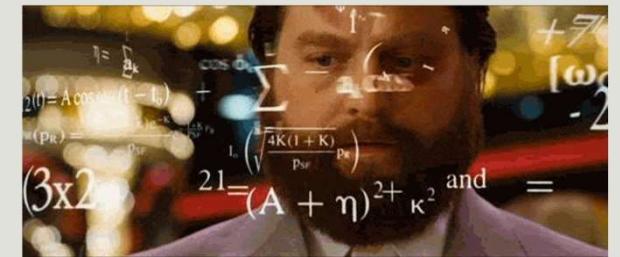
catchError

- 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("something went wrong");  
}),
```

Exercice



- Reprenez les composants CvComponent et DetailsCvComponent.
- Utilisez l'async pipe à chaque fois que vous le pouvez.
- Utilisez catchError pour gérer les erreurs.

Les opérateurs RxJs

Gestion des erreur

l'opérateur retry

- RxJS dispose d'un **opérateur de nouvelle tentative** très pratique, qui est utile dans les situations où une demande de données échoue.
- Par exemple, vous essayez de charger des données à partir d'une API et cela génère des erreurs.
- L'opérateur **retry**, va réessayer avant de déclencher une erreur.

```
return this.http.get<Cv[]>(API.cv).pipe(  
    retry(5)  
);
```

Les opérateurs RxJs

Gestion des erreur

l'opérateur retry

- L'opérateur **retry** peut aussi accepter un objet avec les paramètres suivants:
 - **count** : le nombre maximal de tentative
 - **delay** : number | ((error: any, retryCount: number) => ObservableInput<any>)

Le nombre de secondes avant de relancer ou un notifier qui peut être un Subject et qui à chaque émission déclenchera un nouvel essai.

```
return this.http.get<Cv[]>(API.cv).pipe(  
  retry({  
    delay: 3000,  
    count: 3  
  })  
);
```

Quelques opérateurs utiles de l'Observable

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

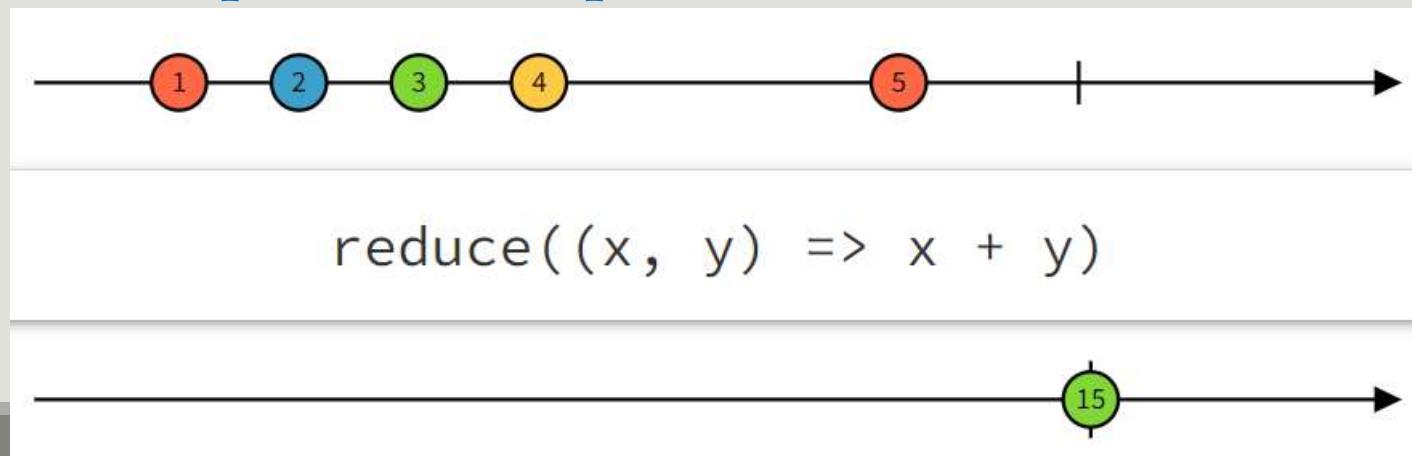
<http://reactivex.io/rxjs/manual/overview.html#operators>

<http://rxmarbles.com/>

Quelques opérateurs utiles de l'Observable

opérateur pipable reduce

- **reduce** est un opérateur qui permet de réduire un flux en une valeur égale à l'opération passé en paramètre.
- Cette fonction prend en paramètre la valeur accumulé et la valeur actuelle.
- **Il émet uniquement lorsque le flux se termine.**

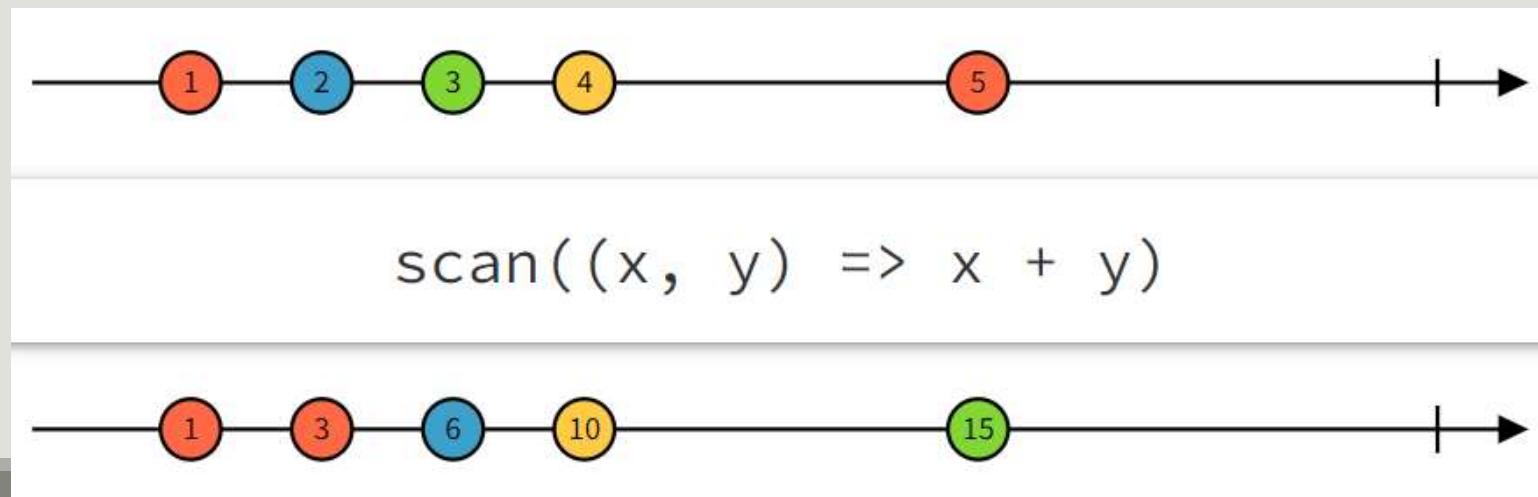


Quelques opérateurs utiles de l'Observable

opérateur pipable

scan

- scan est un opérateur qui ressemble à **reduce** sauf qu'il émet une valeur à chaque réception d'une nouvelle valeur recu du flux.
- Cette fonction prend en paramètre la valeur accumulé et la valeur actuelle.



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.

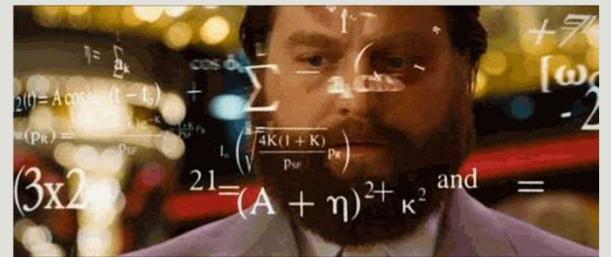
Les ReplaySubject

- 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

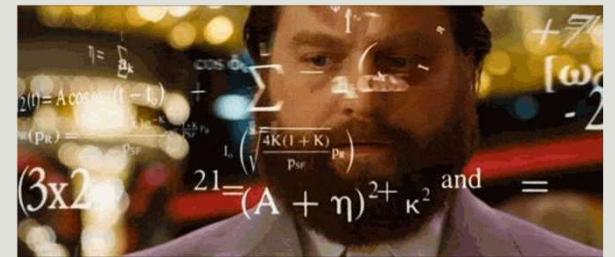
- Alors que BehaviorSubject et ReplaySubject stockent tous deux des valeurs, **AsyncSubject** fonctionne un peu différemment. L'AsyncSubject est une variante de l'objet où **seule la dernière valeur de l'exécution Observable est envoyée à ses abonnés, et uniquement lorsque l'exécution est terminée.**

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 (son email), à savoir qu'il est authentifié ou non.
- Choisissez la bonne structure

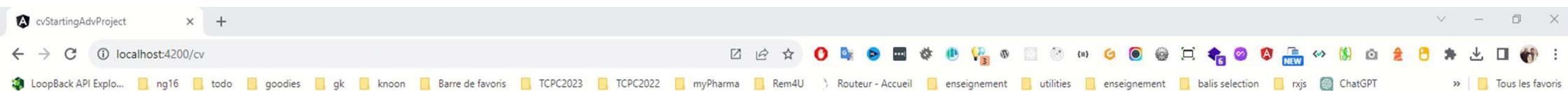
Exercice



- Nous voulons reproduire l'interface suivante qui permet de récupérer au fur et à mesure les produits en utilisant l'API suivante :

<https://dummyjson.com/products>
(<https://dummyjson.com/docs/products>)

- Au départ afficher un nombre de produit (12)
- En cliquant sur le bouton plus de produits, récupérer au fur et à mesure d'autre produits.
- S'il ne reste aucun produit, arrêter d'appeler l'API
- Faites en sorte que le code soit déclarative.



RxJS Home Cv Add Cv Todo Products Logout

TUESDAY, SEPTEMBER 26, 2023 AT 3:32:01 PM GMT+01:00



Nidhal Jelassi



Aymen Sellaouti

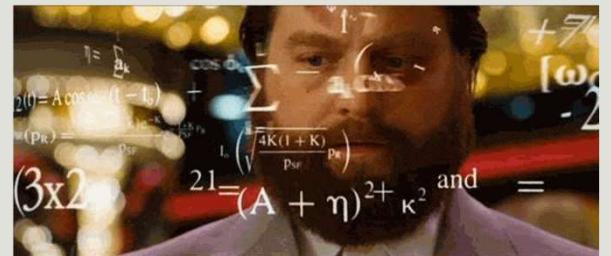


Skander Sellaouti

Footer



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.
- Créer un composant autocomplete, Ajouter y un champ input. En saisissant des caractères, la liste des choix doit automatiquement changer et n'afficher que les cvs qui contiennent la chaîne saisie.
- En sélectionnant un des choix, afficher le Cv correspondant.
- Pensez à optimisez votre code en minimisant les appels http afin de ne pas spammer votre serveur
- Faites en sorte d'avoir le code le plus récursive possible.

Cv Aymen Sellaouti

Informations

Name Sellaouti
Firstname Aymen
Job Teacher
Age 40
Cin 123456

267

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 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 peut pas utiliser l'async pipe ?

```
export class TestUnsbscribeComponent 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();
  }
}
```

Angular Routing Avancée

AYMEN SELLAOUTI

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: n
    ► 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 aimerais 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

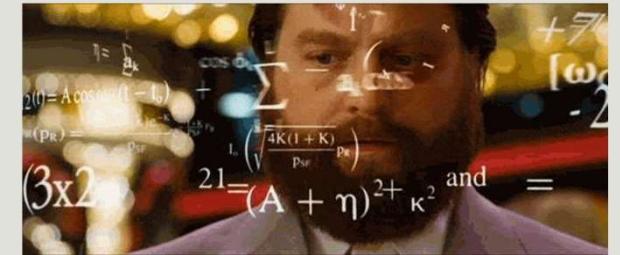
```
const CV_ROUTE: Routes = [
  {
    path: 'cv',
    component: CvComponent,
    children: [
      {path: 'detail/:id', component: DetailCvComponent },
      {path: 'addPersonne', component: FormPersonneComponent },
    ]
  }
];
```

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.

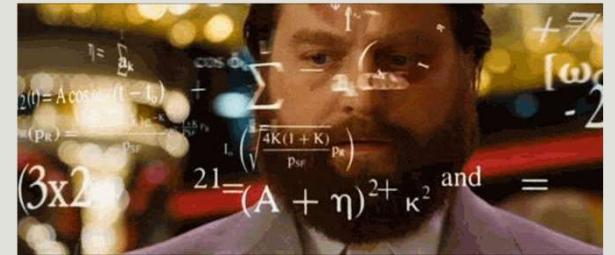
Exercice

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

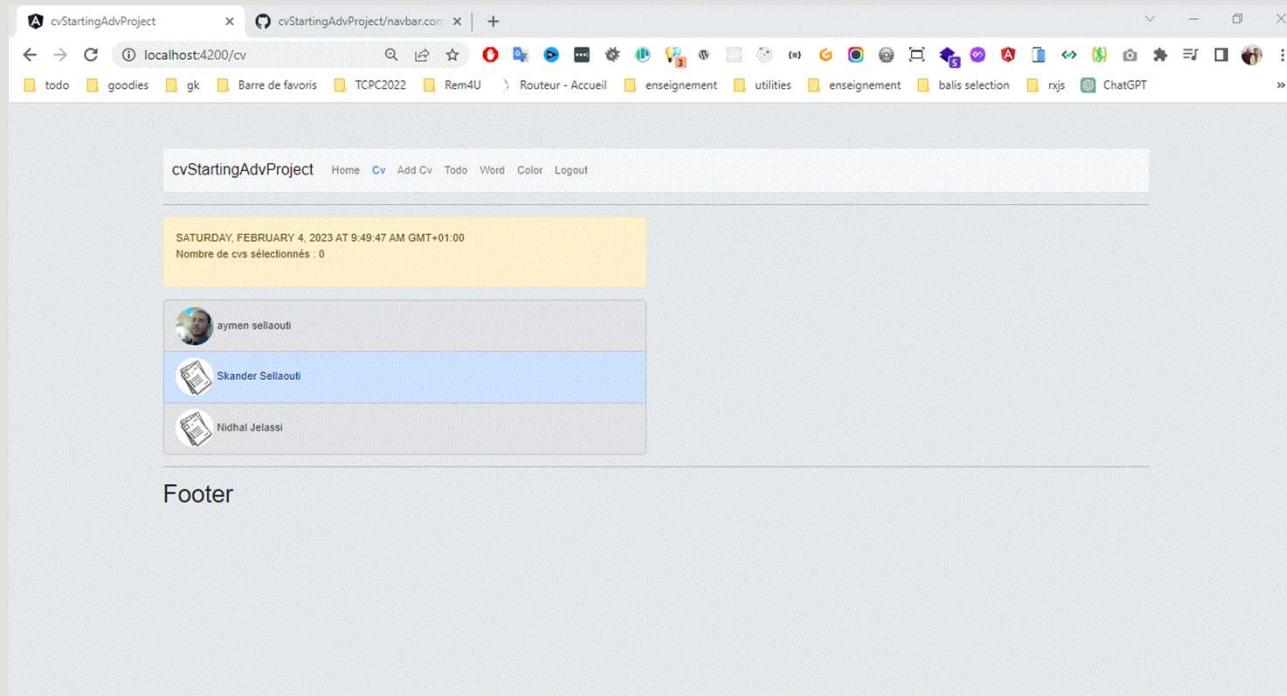
Nidhal Jelassi

Footer

294

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  
  }  
},
```

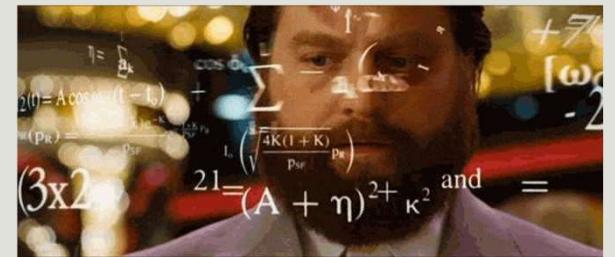
Router Resolver

- 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]  
},
```

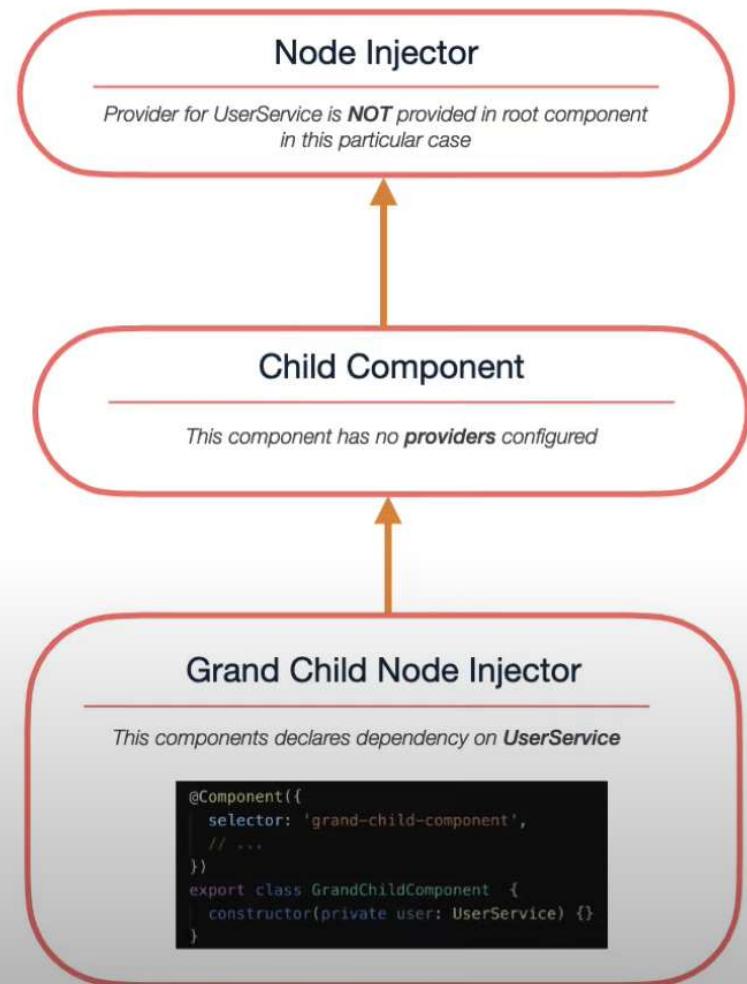
Route Provider

A partir d'Angular 14

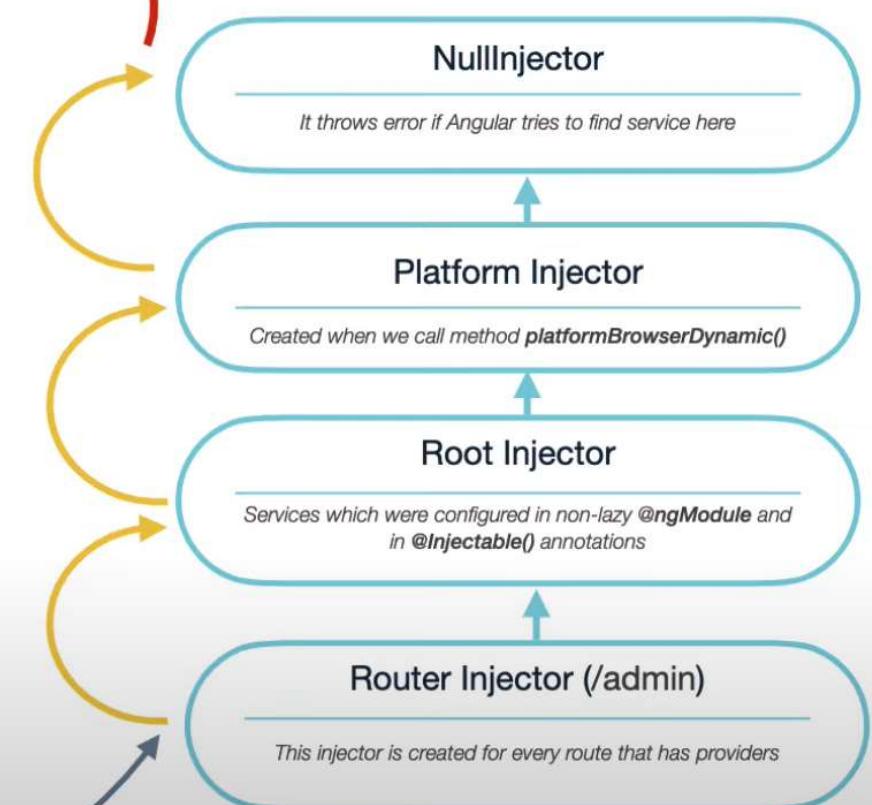
- Ceci va créer un nouvel *Injector* qui sera appelé juste après l'*element Injector*

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

Node Injector Hierarchy



Environment Injector Hierarchy



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 ça permet d'avoir un **formulaire testable**
- Finalement ceci permet de plus facilement **générer des Validateurs personnalisés.**

Reactive form

Créer un formulaire

- 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**

Reactive form

Créer un formulaire

```
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

```
constructor(controls: TControl, validatorOrOpts?: ValidatorFn | ValidatorFn[] | AbstractControlOptions | null, asyncValidator?: AsyncValidatorFn | AsyncValidatorFn[] | null);
  controls: eTypedOrUntyped<TControl, TControl, {
    [key: string]: AbstractControl<any>;
  }>;
```

```
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';
}
```

Reactive form

Créer un formulaire

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

Reactive form

Créer un formulaire

- 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
1 styleUrls: ['./reactive.component.css']
2
3 }
4
5 export class ReactiveForm {
6   constructor(private fb: FormBuilder) {
7     this.form = fb.group({
8       email: ['', Validators.required]
9     });
10  }
11
12  form: FormGroup;
}

reactive.component.html
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" formControlName="email" />
7   </div>
8 </form>
```

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 files open. On the left is a TypeScript file named 'FormComponent.ts' containing the following code:

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

On the right is an HTML file named 'FormComponent.html' containing the following template:

```
<div>  
    <label for="email">Email</label>  
    <input  
        class="form-control" type="email" id="email"  
        formControlName="email" placeholder="Enter email"/>  
</div>  
    <div>  
        <label for="username">Username</label>  
        <input  
            class="form-control" type="text" id="username"  
            formControlName="username" placeholder="Username"/>  
</div>
```

Both the 'email' FormControl in the TypeScript code and the 'email' input field in the HTML template have their 'formControlName' attributes highlighted with a red box.

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 is the TypeScript code:

```
process() {
  console.log(this.form);
}
```

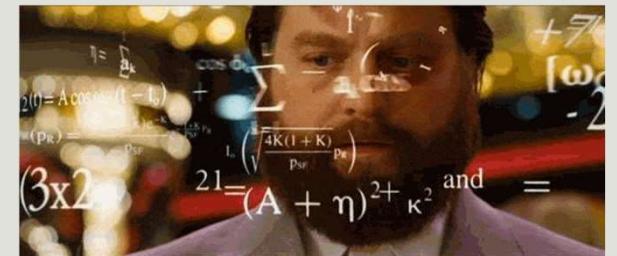
On the right is the corresponding HTML template:

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

The code editor highlights the 'process()' method and the 'Submit' button's click event handler.

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.]),  
  ↪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

- 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.]),
  ↪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

- 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.

```
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

```
class Validators {  
    static min(min: number): ValidatorFn  
    static max(max: number): ValidatorFn  
    static required(control: AbstractControl<any, any>): ValidationErrors | null  
    static requiredTrue(control: AbstractControl<any, any>): ValidationErrors | null  
    static email(control: AbstractControl<any, any>): ValidationErrors | null  
    static minLength(minLength: number): ValidatorFn  
    static maxLength(maxLength: number): ValidatorFn  
    static pattern(pattern: string | RegExp): ValidatorFn  
    static nullValidator(control: AbstractControl<any, any>): ValidationErrors | null  
    static compose(validators: ValidatorFn[]): ValidatorFn | null  
    static composeAsync(validators: AsyncValidatorFn[]): AsyncValidatorFn | null  
}
```

Reactive form

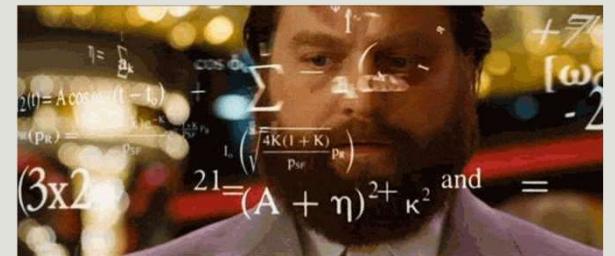
Les Validateurs offerts par Angular

- 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.

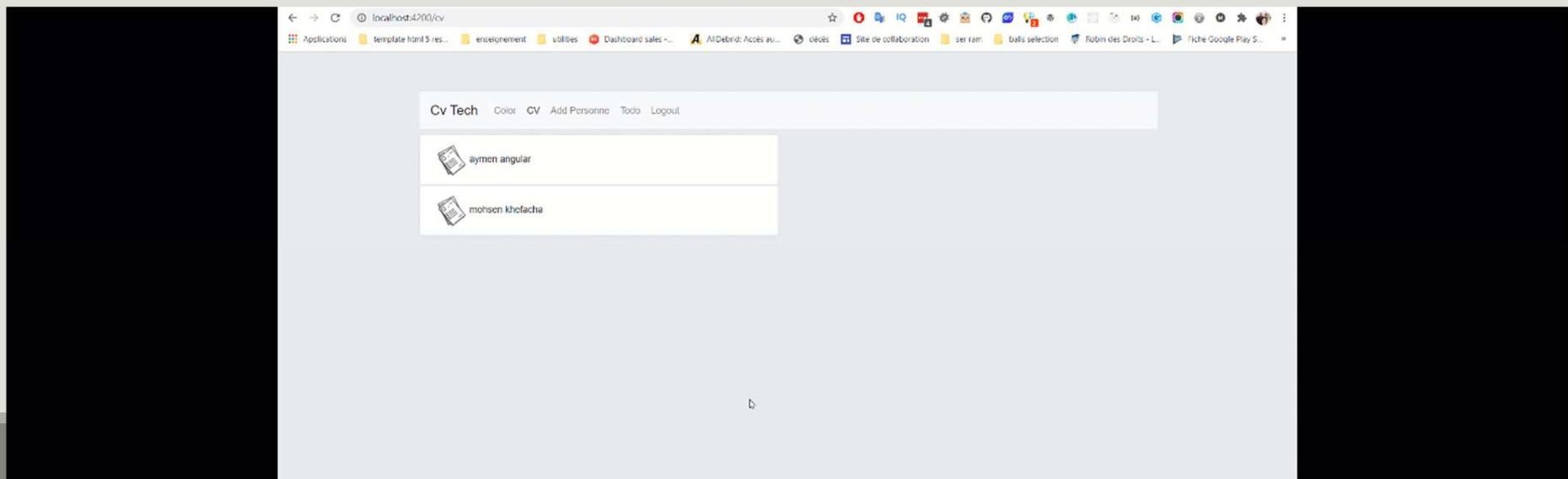
```
formGroup: FormGroup = new FormGroup({  
    name: new FormControl(null, {  
        validators: [Validators.required, Validators.minLength(3)],  
        asyncValidators: [],  
        updateOn: "change"  
    }),  
    age: new FormControl(null),  
});
```

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.

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

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

Reactive form

Manipulez les valeurs de votre form

- En deuxième paramètre de ces deux fonctions, vous pouvez passer un objet d'options avec deux propriétés:
 - **onlySelf**: Angular vérifie l'état de validation du formulaire, chaque fois qu'il y a un changement de valeur. La **validation commence à partir du contrôle** dont la valeur a été modifiée et **se propage au FormGroup** de niveau supérieur. Il s'agit du comportement par défaut. Si vous ne souhaitez pas **qu'Angular vérifie la validité de l'ensemble du formulaire, chaque fois que vous modifiez la valeur à l'aide de setValue ou patchValue, définissez onlySelf à true.**

Reactive form

Manipulez les valeurs de votre form

- En deuxième paramètre de ces deux fonctions, vous pouvez passer un objet d'options avec deux propriétés:
 - **emitEvent**: Les forms Angular émettent deux événements. L'un est **ValueChanges** et l'autre est **StatusChanges**. L'événement ValueChanges est émis chaque fois que la valeur du formulaire est modifiée. L'événement StatusChanges est émis chaque fois qu'angular calcule l'état de validation du formulaire. C'est le comportement par défaut Nous pouvons empêcher que cela se produise, en définissant l'emitEvent à false

Reactive form

Manipulez les valeurs de votre form

```
this.form.patchValue(  
  { firstname: "Aymen" },  
  {  
    emitEvent: false,  
    onlySelf: true,  
  }  
);
```

Reactive form

Suivre les modifications de vos FormControl et de vos FormGroup

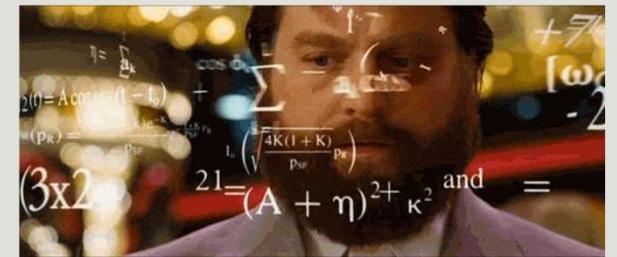
- **FormControl et FormGroup**, vous fournissent deux Observables permettant le suivi des changements de valeur et de status.
- **ValueChanges** est un événement déclenché par les formulaires Angular chaque fois que la valeur de FormControl, FormGroup ou FormArray change. L'observable obtient la dernière 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

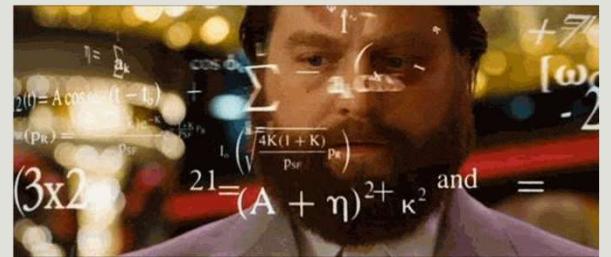
- **statusChanges** est un événement déclenché par les formulaires Angular **chaque fois que Angular calcule le statut de validation** de FormControl, FormGroup ou FormArray. Il renvoie un observable afin que vous puissiez vous y abonner. L'observable obtient le dernier état du contrôle.

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;  
}
```

```
export declare type ValidationErrors = {  
  [key: string]: any;  
};
```

Reactive form

Customiser vos validateurs

- 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 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 un objet de type **ValidationErrors**.

Reactive form

Customiser vos validateurs

- L'objet **ValidationErrors** peut avoir comme propriétés **les multiples erreurs trouvées** (généralement une seule) et comme valeurs les détails de chaque erreur.
- Si nous voulons simplement indiquer qu'une erreur s'est produite, **sans fournir plus de détails**, nous pouvons simplement **renvoyer true** comme valeur d'une propriété error dans l'objet **ValidationErrors**.

```
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);  
  };  
}
```

Reactive form

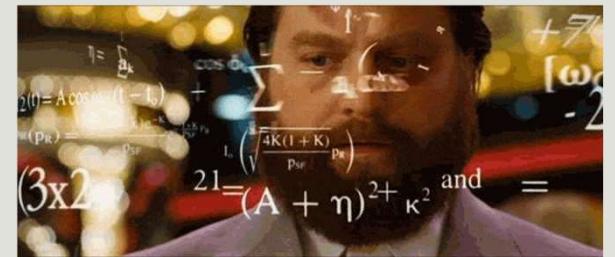
Customiser vos validateurs

Validateurs Asynchrones

```
findUserByEmail(value: any): Observable<ValidationErrors | null> {
  return new Observable<ValidationErrors | null>(
    (observer) => {
      setTimeout(
        () => {
          const date = new Date();
          if (date.getTime() % 2) {
            observer.next(null);
          } else {
            observer.next({userExists: true});
          }
        }, 1500);
    }
  );
}
```

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

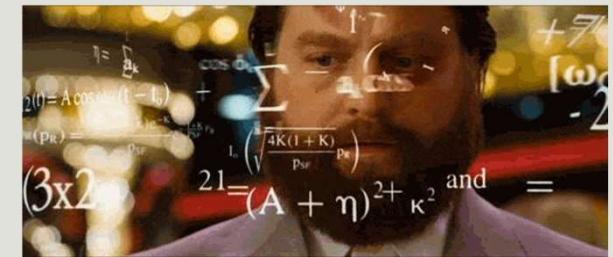
Customiser vos validateurs

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.

Reactive form

FormArray

- Dans les Reactive Form, un formulaire est défini à l'aide des API FormControl et FormGroup, ou à l'aide de l'API FormBuilder dans la plupart des cas, où tous **les champs** d'un formulaire sont bien **connus à l'avance**, permettant de définir un modèle statique.
- Imaginons d'autres situations plus avancées mais encore fréquemment rencontrées où le **formulaire est beaucoup plus dynamique** et où tous **les champs du formulaire ne sont pas connus à l'avance**
- Prenons le cas d'un formulaire dynamique dans lequel des contrôles de formulaire sont ajoutés ou supprimés du formulaire par l'utilisateur, en fonction de son interaction avec l'interface utilisateur.

Reactive form

FormArray

- Un **FormArray**, tout comme un **FormGroup**, est également un **conteneur de FormControl**.
- Contrairement à un FormGroup, un conteneur **FormArray ne nous oblige pas à connaître tous les contrôles à l'avance**, ainsi que leurs noms.
- En fait, un FormArray peut avoir un **nombre indéterminé de FormControl**, en commençant par zéro ! Les contrôles peuvent ensuite être ajoutés et supprimés **dynamiquement** en fonction de la façon dont l'utilisateur interagit avec l'interface utilisateur.

Reactive form

FormArray

- Chaque contrôle aura alors une **position numérique** dans le tableau des contrôles de formulaire, **au lieu d'un nom unique**.
- Les contrôles de formulaire peuvent être ajoutés ou supprimés du modèle de formulaire à tout moment lors de l'exécution à l'aide de l'API FormArray.
- Pensez à spécifier un getter pour votre FormArray

```
this.fg2 = this.formBuilder.group({  
  name: [null],  
  age: [0],  
  skills: new FormArray([]),  
});
```

```
get skills() {  
  return this.fg2.get('skills') as FormArray;  
}
```

Reactive form

FormArray

- Afin de déclarer un **FormArray** dans un **FormGroup**, il suffit de définir une **propriété de type un objet FormArray** et de **l'initialiser à un tableau vide** ou contenant **des FormControl ou FormGroup** selon votre cas d'utilisation.

```
this.form = new FormGroup({  
    credentials: new FormGroup({  
        email: new FormControl(null, [Validators.required, Validators.email]),  
        password: new FormControl(null, [Validators.required,  
createPasswordStrengthValidator]),  
    }),  
    username: new FormControl(null, [Validators.required]),  
    age: new FormControl(null, [Validators.required, Validators.pattern('[0-1]?\\d{1,2}')]),  
    skills: new FormArray([]),  
});
```

Reactive form

FormArray

- Afin de déclarer un **FormArray** avec le service **FormBuilder**, il suffit d'appeler la méthode **array de votre FormBuilder** et de lui passer **un tableau vide** ou contenant **des group** selon votre cas d'utilisation.

```
this.form = this.formBuilder.group({  
    credentials: this.formBuilder.group({  
        email: [null, {validators: [Validators.required, Validators.email]}],  
        password: [null, {validators: [Validators.required, createPasswordStrengthValidator()]}],  
    }),  
    username: [null, {validators: [Validators.required]}],  
    age: [null, {validators: [Validators.required, Validators.pattern('[0-1]?\\d{1,2}')]}],  
    skills: this.formBuilder.array([])  
});
```

Reactive form

FormArray

- Dans la partie Template, vous devez identifier un **container groupant tous ses éléments.**
- Ajouter y la directives **formArrayName** et associez la au **champ FormArray**.
- **Boucler sur les éléments contenu dans votre FormArray.** Si c'est des FormGroup, **binder la propriété formGroupName (formControlName si vous avez uniquement des FormControl)** avec l'indice du FormGroup dans le array.

```
<div formArrayName="skills">
  Skills <button (click)="addSkill()" class="btn btn-success">Add Skill</button>
  <div
    *ngFor="let skillControl of skills.controls; let i = index"
    [formGroupName]="i">
    <input type="text" class="form-control" formControlName="name">
  </div>
```

Reactive form

FormArray

Voici les méthodes les plus couramment utilisées disponibles dans l'API FormArray :

- **controls** : il s'agit d'un **tableau** contenant tous les **FormControl** qui font partie du FormArray
- **length** : C'est la **longueur totale du tableau**
- **at(index)** : renvoie le **FormControl** à une position de tableau donnée
- **push(control)** : ajoute un nouveau **FormControl** à la fin du tableau
- **insert(index, control, option)** : ajoute un nouveau **FormControl** à la position index
- **removeAt(index)** : supprime un **FormControl** à une position donnée du tableau
- **getRawValue()** : Obtient les **valeurs de tous les contrôles** de formulaire, via la propriété **control.value** de chaque **FormControl**.
- **setValue(value, option, emitEvent)**: Définit la valeur de FormArray. Il accepte un tableau qui correspond à la structure du contrôle.

Reactive form

FormArray

Ajout dynamique d'éléments

- Afin d'ajouter un élément dynamiquement, vous pouvez utiliser l'API FormArray et sa méthode push.
- Préparer l'élément à ajouter et pusher le dans votre Array

```
addSkill(): void {
  const skillFormGroup = this.formBuilder.group({
    name: [null, Validators.required]
  });
  this.skills.push(skillFormGroup);
}
```

Angular

Les modules

AYMEN SELLAOUTI

Qu'est ce qu'un module

- C'est une **classe** avec une décoration **NgModule**
- Un module est un **conteneur** qui englobe un **ensemble de fonctionnalités** liées.
- Les applications Angular sont **modulaire**.
- Un application Angular contient **au moins un Module** : AppModule.
- Un Module Angular peut contenir des composants, des provider de service...
- Une application simple est généralement composé d'un seul module. Par contre dès que votre application grandit penser à la séparer en Modules.
- Chaque module **vie séparée** des autres modules. Par défaut **il n'expose rien**, tous ce qui est à l'intérieur du module **reste uniquement dans le module tant qu'on ne l'exporte pas**.
- Lorsqu'on importe un module, on **importe réellement tous ce qu'il exporte**.

Rôle d'un module

- Organise votre application en des briques fonctionnelles
- Etend votre application avec des librairies externes
- Permet d'agréger et d'exporter des briques fonctionnels
- Faciliter le développement et la maintenance de votre application

Définition d'un module

- L'annotation (decorator) `@NgModule` identifie un module Angular.
- L'annotation prend en paramètre un objet spécifiant à Angular comment compiler et lancer l'application.
 - `imports` : tableau contenant les modules utilisés.
 - `declarations` : tableau contenant les classes de vue appartenant à ce module, i.e. composants, directives et pipes de l'application.
 - `exports` : tableau des classes de vues à exporter.
 - `providers` : déclaration des services
 - `bootstrap` : indique le composant exécuter au lancement de l'application et elle ne concerne que le module racine.

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

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

declarations

- Dans la partie **declarations**, faite en sorte que chaque composant, directive ou pipe soit associé à **un et un seul module**. **Ne déclarer pas un même composant dans deux modules différents.**
- Ne déclarer que les composants, directives et les pipes dans cette partie.
- Tous les composants, directives et les pipes déclarés sont **privés par défaut**. Ils ne sont accessible que pour les composants, directives et les pipes déclarés dans le même module.
- Pour utiliser un de ces éléments à l'extérieur du module, il faudra penser à les exporter.

Routing

- Vous pouvez créer les routes de vos modules de la même manière que pour le routing de l'AppModule.

- Cependant, au lieu d'utiliser RouterModule.forRoot vous devez utiliser la méthode **forChild** du RouterModule.

Exemple pour le TodoModule

```
import { CommonModule } from "@angular/common";
import { NgModule } from "@angular/core";
import { FormsModule } from "@angular/forms";

import { TodoComponent } from "./todo.component";
import { TodoRoutingModule } from "./todo.routing";

@NgModule({
  declarations: [TodoComponent],
  imports: [CommonModule, FormsModule, TodoRoutingModule],
  // Si vous n'avez pas besoin de TodoComponent à l'extérieur,
  // ne l'exporter pas
  exports: [TodoComponent],
})
export class TodoModule {}
```

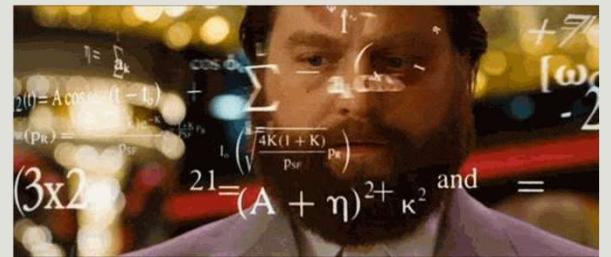
```
import { NgModule } from "@angular/core";
import { Route, RouterModule } from "@angular/router";
import { NF404Component } from "../nf404/nf404.component";
import { TodoComponent } from "./todo.component";

const routes: Route[] = [
  { path: "todo", component: TodoComponent },
  { path: "**", component: NF404Component },
];

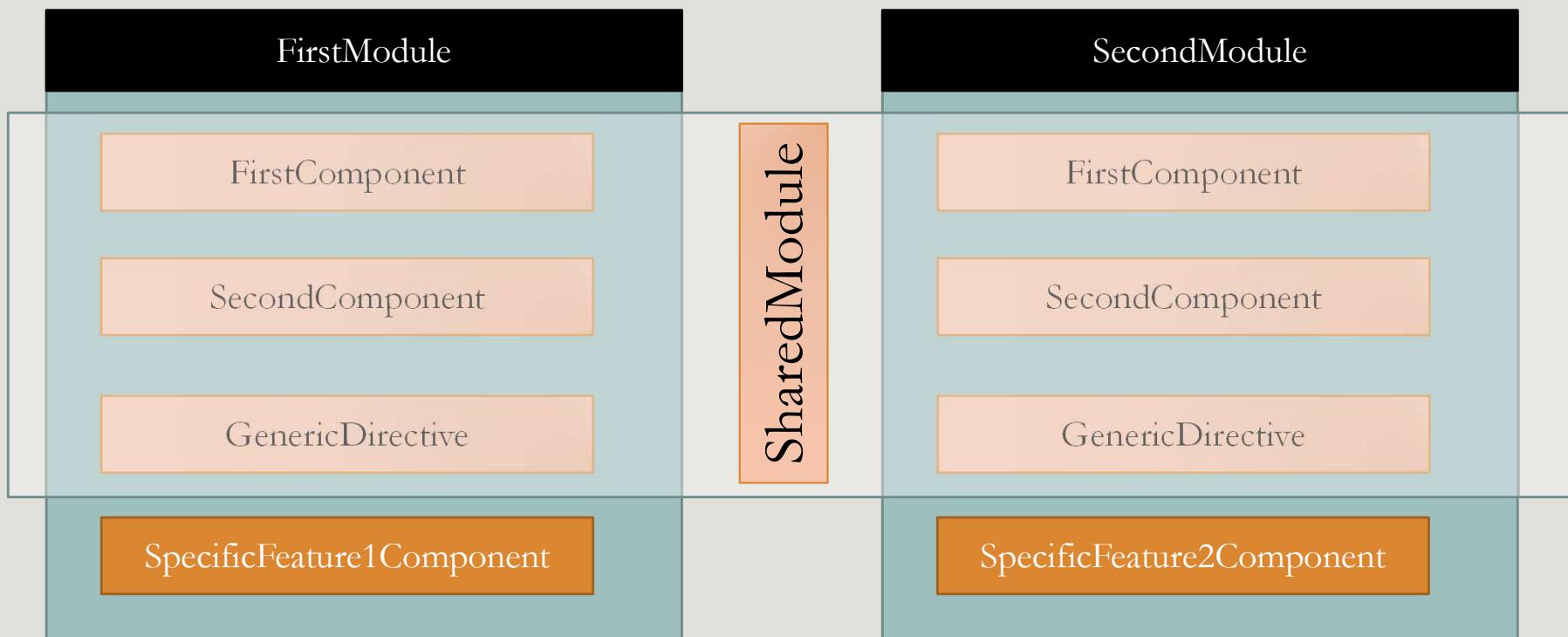
@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule],
})
export class TodoRoutingModule {}
```

Exercice

- Implémentez le CvModule.



Shared Module

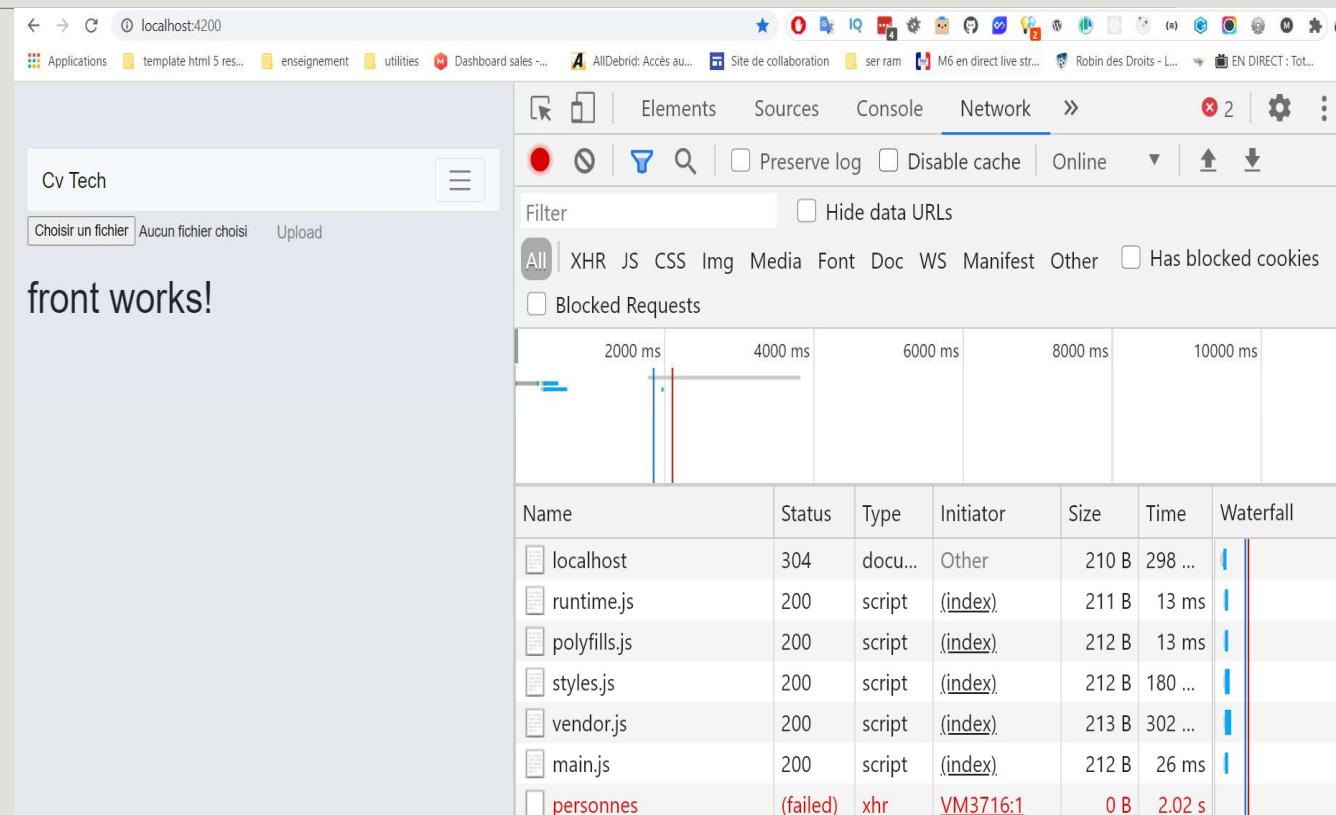


Shared Module

```
@NgModule({
  declarations: [
    FirstComponent,
    SecondComponent,
    GenericDirective,
  ],
  imports: [
    CommonModule
  ]
  exports: [
    FirstComponent,
    SecondComponent,
    GenericDirective,
  ]
})
export class SharedModule { }
```

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.



Lazy Loading

➤ Afin d'implémenter le *Lazy Loading*, on doit suivre les étapes suivantes :

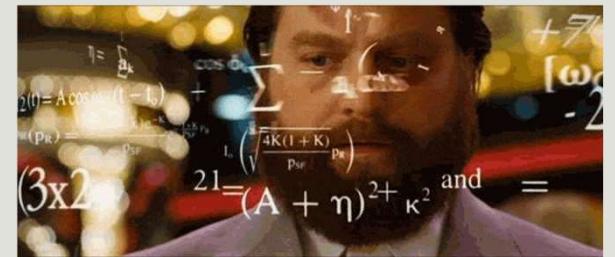
1. Le **module** à charger doit **lui-même gérer sa partie routing**
2. Au niveau du routing principal (AppRoutingModule), créer une **route**, ajouter lui un path ‘ça sera le **préfixe** de toutes les routes du module’, et ajouter une nouvelle clé qui est **loadChildren**. Cette clé va informer angular 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  
  ),  
},
```

Exercice

- Testez le lazy loading avec le CvModule



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

- 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

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 ciblé 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 {
      return of(null);
    }
  }
}
```



Angular State Management NgRx Store

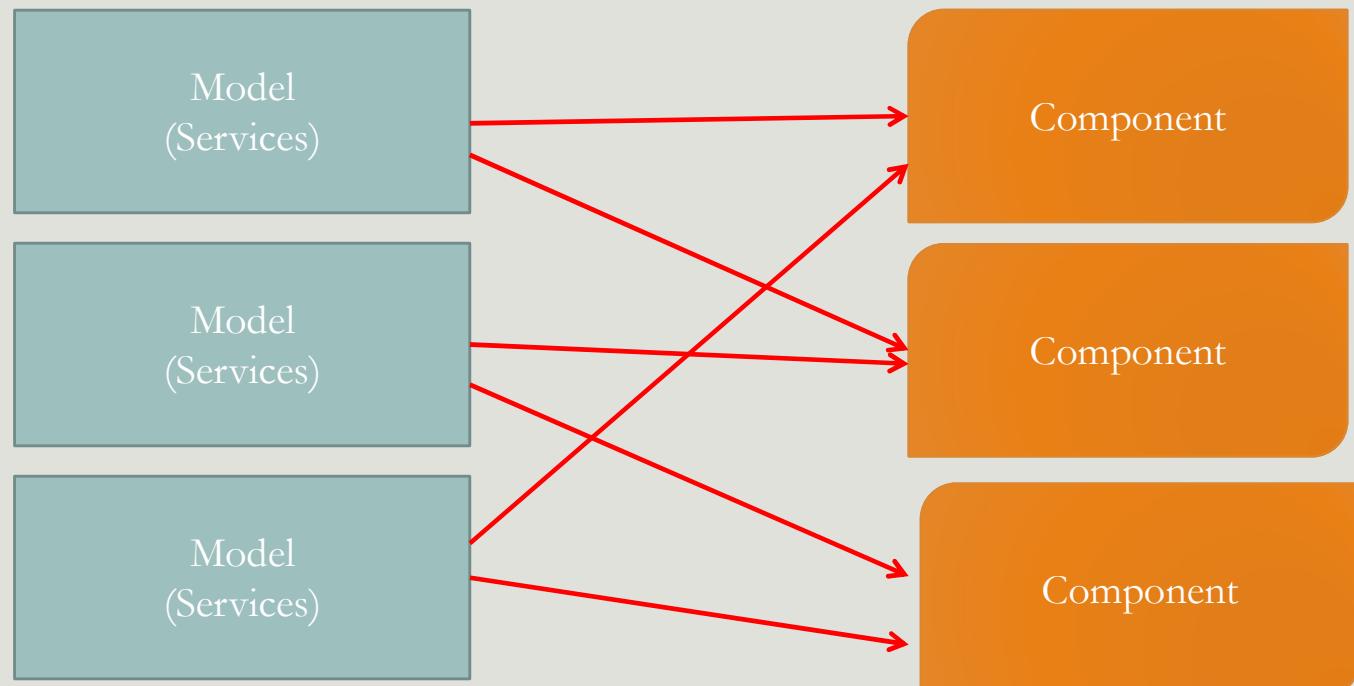
AYMEN SELLAOUTI

NgRx

Introduction



- Dans la plupart de nos applications, nous **gérons des composants qui communiquent avec des models** afin de gérer leur **état**.
- Plus l'application **grandi** et plus ca devient **compliqué de gérer l'état** de l'application qui est éparpillé dans les composants et les services.



NgRx

Introduction



- Solution : Gestion centralisée de l'état de votre application



REDUX

NgRx

Introduction



Composant

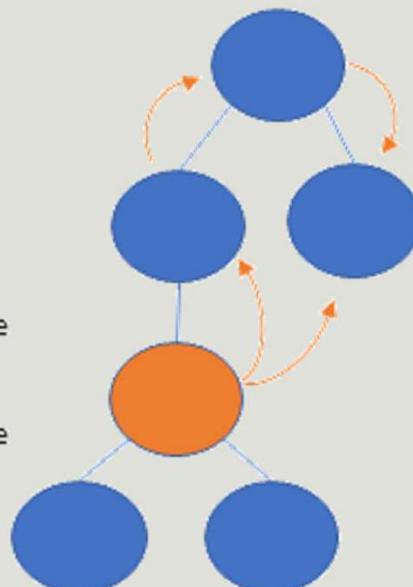


Composant souhaitant
modifier une donnée

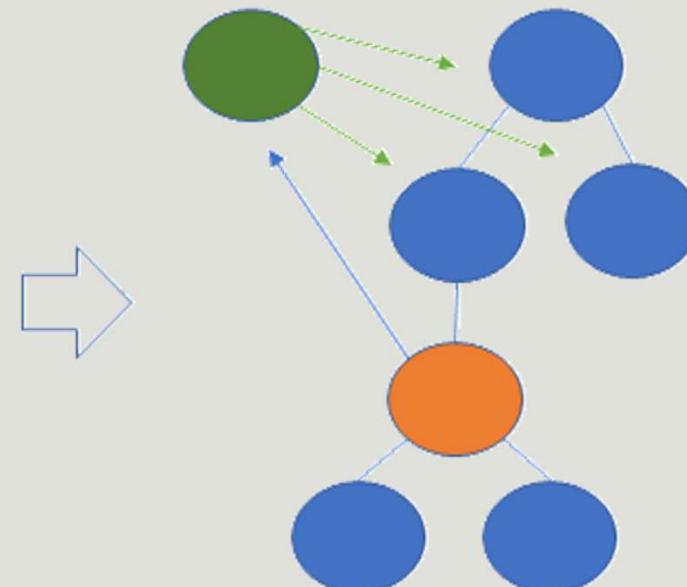


Composant de gestion du state

- > Modifie directement la donnée
- > Demande une modification
- > Notifie d'un nouvel état



Spaghetti



State managed

NgRx

Introduction

Redux



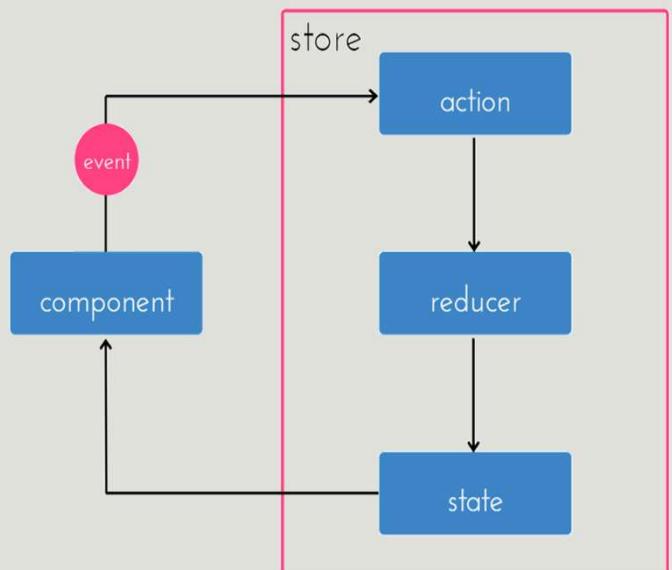
➤ **Store** : C'est un objet contenant le state de votre application. **Le store n'est pas le state.**

➤ **State** : C'est la source de **vérité unique**. C'est un objet **immutable** qui **centralise l'état de votre application**.

➤ L'état est la **représentation de votre application** à un moment **donnée**.

➤ Les changement de l'état sont faites par des **fonctions pures**. Ces fonctions n'ont **aucun effet de bord** et **ne modifient aucune donnée**. Elle ne font que **retourner une nouvelle valeur de l'état**.

➤ Les fonctions pures ne **se basent** que **sur les paramètres d'entrées** et **ne gèrent pas les éléments externes**. Ceci implique que pour une **même entrée**, on aura toujours le **même résultat de sortie**.





NgRx

- NgRx est une implémentation du **pattern Redux**.
- Elle permet de **gérer l'état de votre application**.
- NgRx est un **framework** pour créer des applications réactives dans Angular. NgRx fournit des bibliothèques pour:
 - **Gérer l'état** global et local de votre application.
 - **Isoler des effets de bord** permettant d'avoir une architecture de composants plus propre.
 - **Gérer la collection** de vos entités.
 - **S'intégrer avec le routeur** d'Angular.

NgRx

Quand L'utiliser



- NgRx présente plusieurs avantages, cependant **il ne faut pas toujours l'utiliser.**
- Dans la documentation officielle de NgRx, on vous demande de réfléchir avant d'entamer l'utilisation de NgRx dans votre application.
- L'idée est de **ne pas ajouter une couche de complexité** à votre code **si vous n'en avez pas besoin.**

NgRx

Quand L'utiliser



- Le principe à suivre s'appelle **SHARI**. Il permet de voir quand utiliser NgRx :
 - **Shared**: Est ce que votre State est partagé par plusieurs composants et services ?
 - **Hydrated**: Est ce que votre State doit être conservé et hydraté lors des recharges de pages
 - **Available**: Est ce que le State doit être récupéré quand vous changer de routes.
 - **Retrieved**: Est ce que votre State doit être récupéré à travers des effets de bord, e.g. une requête HTTP.
 - **Impacted**: Est ce que le State est impacté par plusieurs components

NgRx

Quand L'utiliser



- Voici les cas d'usages les plus présents :
 - Quand plusieurs composants et services dépendent de la même portion du state
 - Chaque fois que vous refactorisez votre composant en plusieurs morceaux, vous aurez une énorme probabilité d'introduire un système de gestion d'état.
 - Imaginez que vous ayez une grosse application avec beaucoup de pagination de filtres, etc. Si vous quittez votre page pour modifier quelque chose à la page 5 et que vous revenez ensuite, si vous êtes dans l'état initial de la page (page1 et pas de filtre) c'est vraiment une mauvaise expérience. Dans ce cas, un système de gestion d'état qui gère tous vos filtres, pagination, etc. vous aidera à conserver l'état.

NgRx

Quand L'utiliser



- D'autre part si on revient à la création de Flux et la première apparition de Redux, les problèmes majeurs qu'avait à gérer la team Facebook sont :
 - **Plusieurs acteurs modifient les données**, généralement c'est le serveur et le client.
 - Le problème de '*Extraneous props*' ou d'**@Input étrangers**'. C'est-à-dire qu'on trouve dans les composants des propriétés qui n'ont rien à voir avec l'état du composant mais qui servent juste à récupérer des paramètres à passer aux composant fils ou père.

NgRx

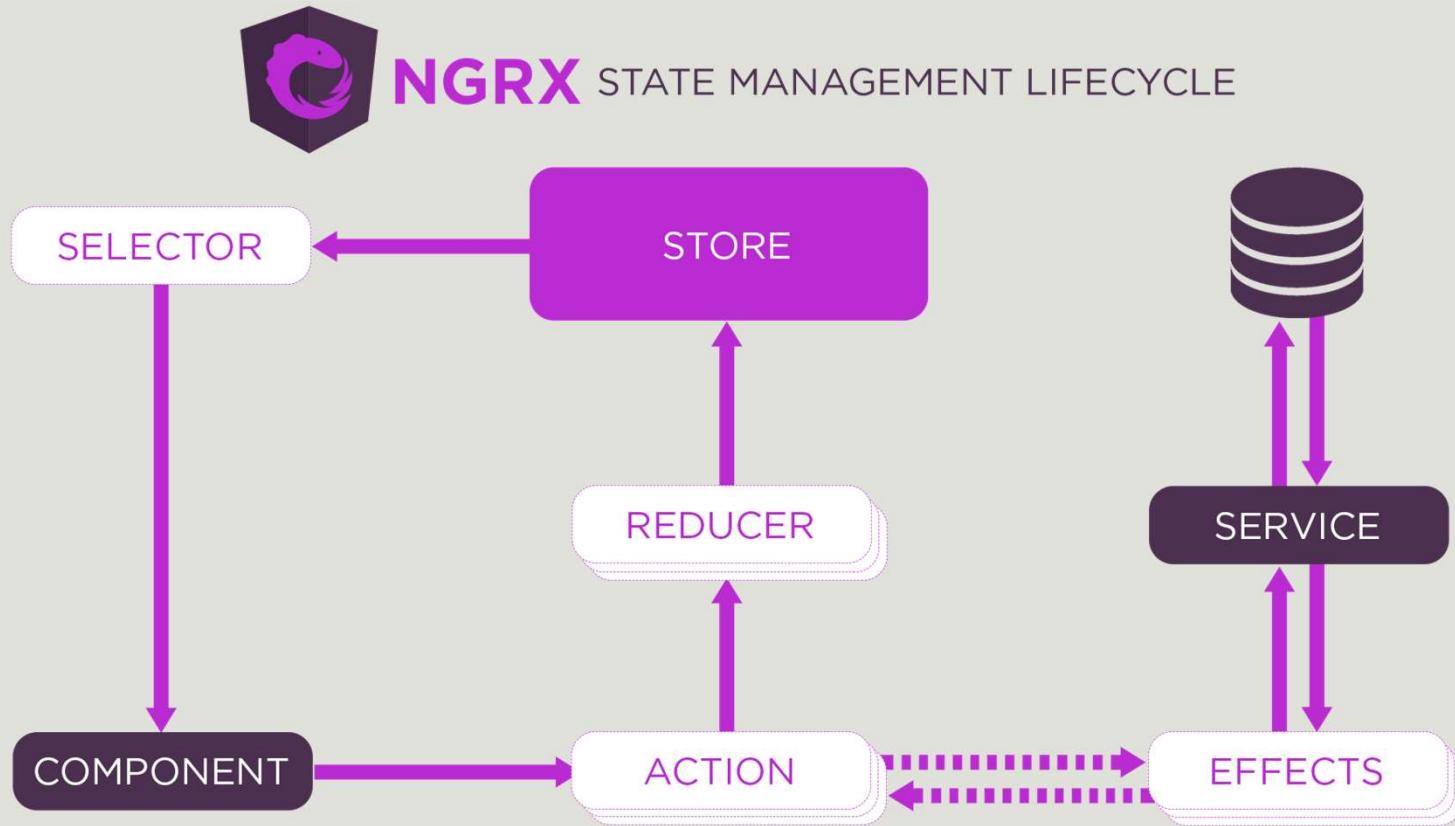
Quand L'utiliser



- [https://dev.to/alfredoperez/my-notes-from-ngrx-workshop-from-
ngconf-2020-part-1-introduction-h8l](https://dev.to/alfredoperez/my-notes-from-ngrx-workshop-from-ngconf-2020-part-1-introduction-h8l)
- https://www.youtube.com/watch?v=omnwu_etHTY&ab_channel=AngularConnect
- [https://dev.to/alfredoperez/angular-service-to-handle-state-using-
behaviorsubject-4818](https://dev.to/alfredoperez/angular-service-to-handle-state-using-behaviorsubject-4818)
- <https://blog.angular-university.io/angular-2-redux-ngrx-rxjs/>

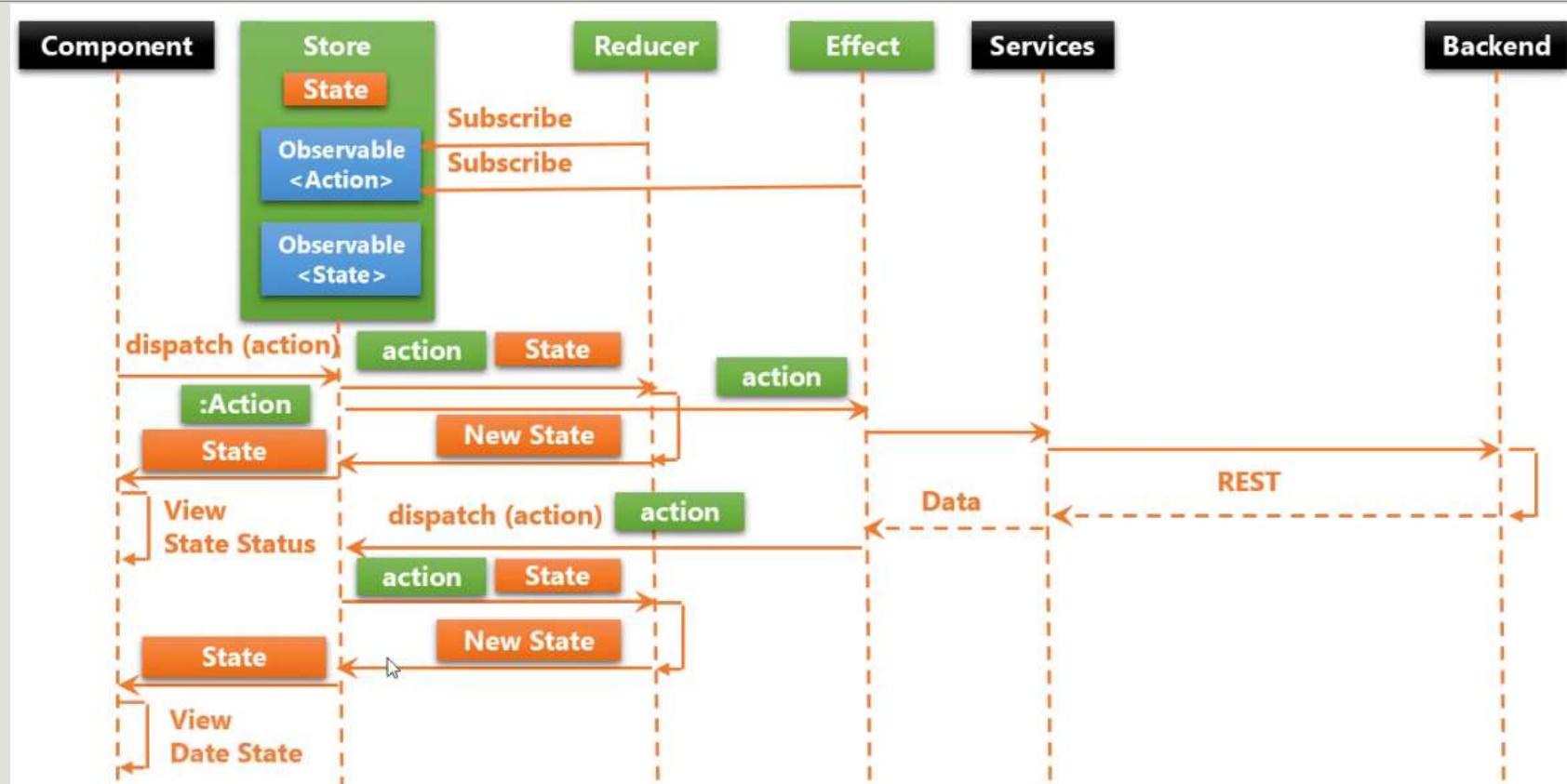
https://www.youtube.com/watch?v=yYiO-kjmLAc&ab_channel=RainerHahnekamp

NgRx Architecture



<https://ngrx.io/>

NgRx Architecture



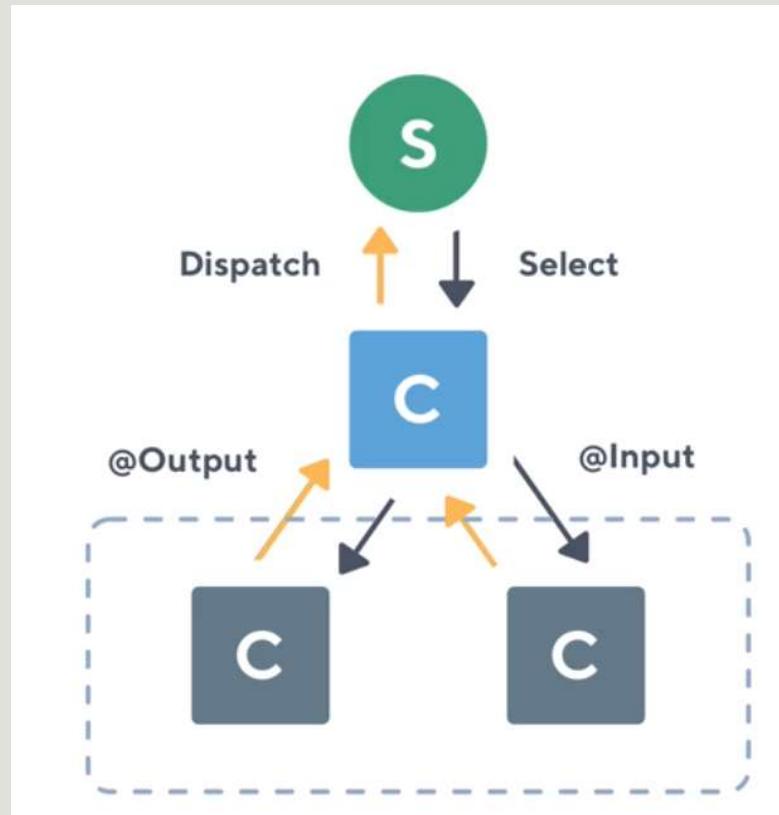
https://www.youtube.com/watch?v=pjZoTOqzl_A

NgRx Architecture



| Smart Component | Presentational Component |
|-------------------------------|--|
| Se préoccupe du Store | Ne se préoccupe pas du store |
| Dispache les actions | Invoque les callbacks via les <code>@Output</code> |
| Récupère les données du Store | Lit les datas des <code>@Input</code> |

NgRx Architecture



NgRx Installation



➤ Vous pouvez installer cette bibliothèque vous pouvez utiliser les commandes suivantes :

- npm install @ngrx/store
- yarn add @ngrx/store
- ng add @ngrx/store

```
imports: [
  BrowserModule,
  BrowserAnimationsModule,
  RouterModule.forRoot(routes),
  StoreModule.forRoot({}, {})
],
```

➤ L'utilisation de la commande ng add déclenchera les schematics installant la bibliothèques et ajoutant les fichiers et la configuration nécessaire.

➤ Le module **StoreModule** sera ajouté et configuré

NgRx

Installation / debugger



- Vous pouvez débugger votre application avec le redux store devtools.
- Il faut d'abord installer l'extension Redux DevTools

<https://github.com/reduxjs/redux-devtools/>

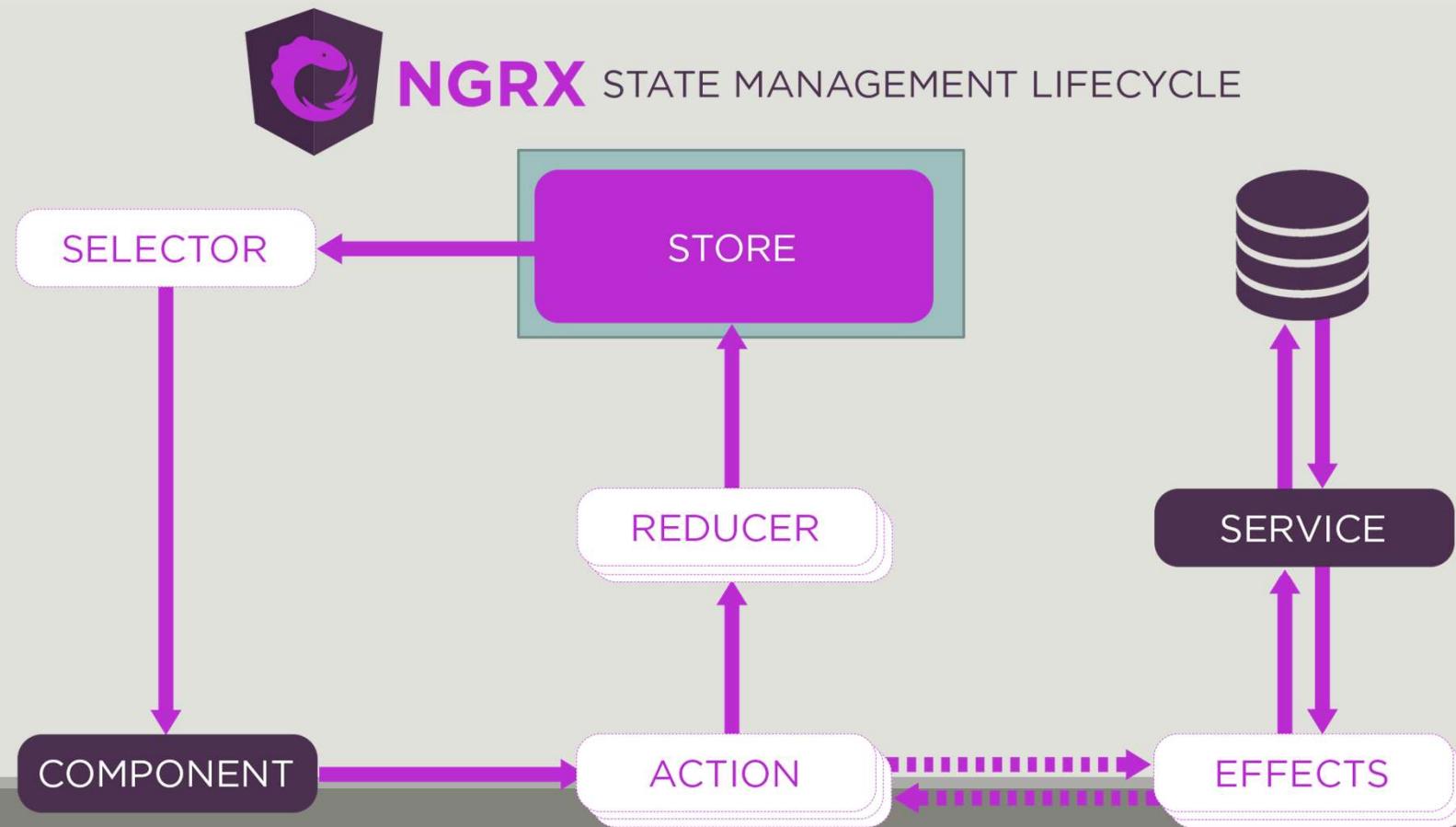
- Pour l'installer lancer la commande suivante :

ng add @ngrx/store-devtools

```
imports: [
  StoreModule.forRoot({}, {}),
  StoreDevtoolsModule.instrument({ maxAge: 25, logOnly: environment.production })
],
```

<https://ngrx.io/guide/store-devtools>

NgRx Architecture



NgRx Store



- Afin de récupérer votre store, vous devez l'injecter.
- Une fois fait, le store est représenté par un observable qui n'offre aucune méthode permettant de manipuler le state.
- Etant un objet générique, vous pouvez lui spécifier l'objet représentant votre état (le state).

```
export interface AppState {}
```

index.ts

```
constructor(  
  private store: Store<AppState>  
)
```

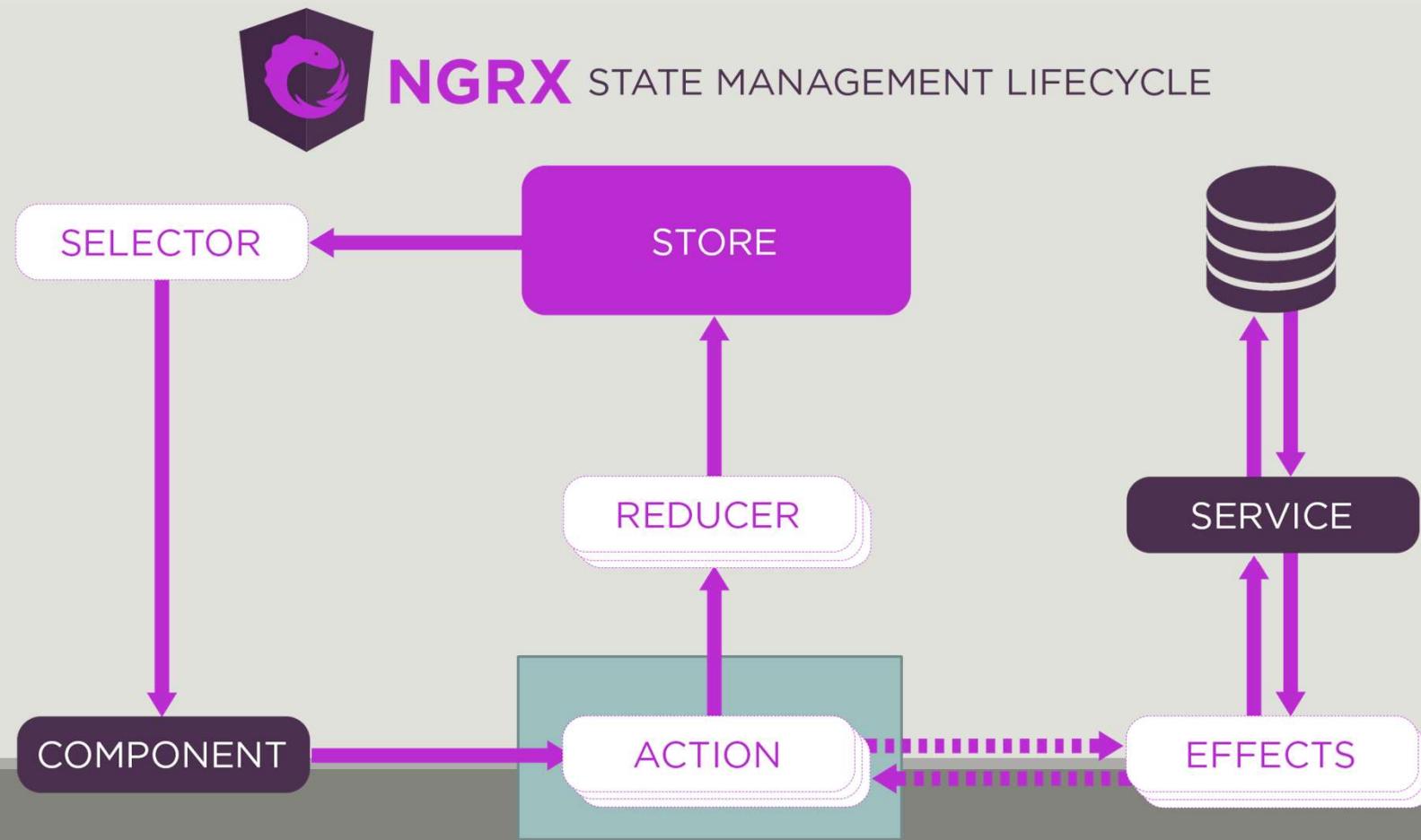
```
this.store.|  
  addReducer  
  complete  
  dispatch  
  error  
  forEach  
  lift  
  next  
  pipe  
  removeReducer  
  select  
  subscribe
```

NgRx Store API



- **dispatch(action: Action)** : cette méthode permet de **déclencher une action** pour mettre à jour l'état de l'application. Elle prend en paramètre un objet de type **Action** décrivant son **type et les données associées**.
- **pipe(select(selector: any))** : cette méthode permet de **sélectionner une partie de l'état** de l'application à partir d'un sélecteur. Elle retourne un **Observable** qui émet **les valeurs de l'état sélectionné**.
- **subscribe()** : cette méthode permet de **s'abonner aux changements de l'état de l'application**. Elle prend en paramètre des fonctions callback pour gérer les changements de l'état, les erreurs et la fin de l'abonnement.

NgRx Architecture



NgRx Actions



- Les **actions** sont l'un des **principaux composants de NgRx**.
- Les actions **expriment des événements** uniques qui se produisent dans votre application.
- Que ce soit des événements **internes** de votre utilisateur, ou des événements **externes** via le réseau ou tout autre événement, les actions sont là pour les décrire.
- Les actions NgRx doivent implémenter **l'interface Action**.
- La propriété **type** représente le type qui est l'identifiant de l'action.
- Vous disposez aussi d'une propriété **payload** qui contiendra les informations à fournir en cas de besoin avec votre action

```
interface Action {  
  type: string;  
}
```

<https://ngrx.io/guide/store/actions>

NgRx Actions



- La propriété type suit une **convention de nommage** : **[Source] Event**
- Ceci permet de définir le **contexte** et de spécifier **quelle catégorie d'action** il s'agit et **d'où une action a été dispatchée**.

```
'[Login Page] User Login'
```

- Vous ajoutez des **propriétés** à une action pour fournir un contexte ou des métadonnées supplémentaires pour une action.

```
{  
  type: '[Login Page] User Login',  
  username: string;  
  password: string;  
}
```

NgRx Actions



➤ NgRx nous fournit une méthode **createAction** qui vous permet de créer une action et une fonction **props** qui permet de spécifier le **type du payload** donnant plus de robustesse à votre code.

```
import { createAction, props } from '@ngrx/store';
import { User } from './model/user.model';

const enum ActionsEnum {
  'LOGIN' = '[Login Page] User Login'
}

export const loginAction = createAction(
  ActionsEnum.LOGIN,
  props<{user: User}>()
)
```

NgRx Actions



```
import { createAction, props } from '@ngrx/store';
import { User } from './model/user.model';
const action = createAction('[Source] action simple');
action();
const action = createAction('[Source] action simple', props<User>());
action({
  id: "1",
  name: "aymen",
  email: "aymen@gmail.com",
});
const action = createAction('[Source] action simple',(user: User, prefix: string) => ({
  ...user,
  name: `${prefix}${user.name}`,
}) );
const user: User = { /* ... */ };
action(user, '@');
```

<https://ngrx.io/guide/store/actions>

NgRx Actions



- Afin de déclencher une action, vous devez utiliser la méthode **dispatch** de votre **store**.
- Cette méthode prend en **paramètre** une **action**.
- En dispatchant cette action, on informe le store qu'un événement s'est déclenché et qu'il faut **Notifier** les **reducers** et les '**effects**'.

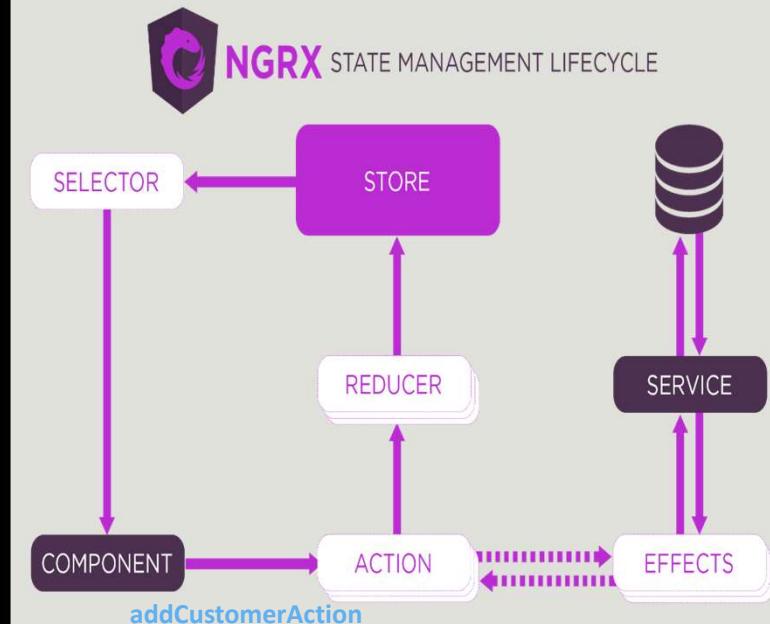
```
this.store.dispatch(loginAction({user}));
```

The screenshot shows the NgRx Store DevTools interface. On the left, there's an 'Inspector' panel with a 'filter...' input and a 'Commit' button. The main area is titled 'NgRx Store DevTools' and has tabs for 'Action', 'State', and 'Diff'. The 'Action' tab is selected. It displays a table with columns 'Action', 'Tree', 'Chart', and 'Raw'. Below the table, there's a code editor showing a JSON object:

```
1 | {
2 |   user: {
3 |     id: 1,
4 |     email: 'test@angular-university.io'
5 |   },
6 |   type: '[Login Page] User Login'
7 | }
```

NgRx Actions

```
@Component({
  selector: 'app-add-customer',
  templateUrl: './add-customer.component.html',
  styleUrls: ['./add-customer.component.css'],
})
export class AddCustomerComponent {
  constructor(public store: Store<AppState>) {}
  name = '';
  email = '';
  addCustomer() {
    const newCustomer = new Customer(uuidv4(), this.name, this.email);
    this.store.dispatch(
      customersActions.addCustomerAction({ customer: newCustomer })
    );
  }
}
export interface AppState {
  customers: Customer[];
}
```



NgRx

Actions

Bonne pratiques



Il existe quelques règles pour écrire de bonnes actions dans votre application.

- **Commencer par les actions** : écrivez des actions avant de développer des fonctionnalités pour avoir **une vue globale de ce que vous aller implémenter**.
- **Catégoriser** : **catégorisez** les actions **en fonction de la source** de l'événement.
- **Soyer généreux en actions** : les actions sont peu coûteuses à écrire, donc plus vous écrivez d'actions, mieux vous exprimez les flux dans votre application.

NgRx

Actions

Bonne pratiques



- C'est une bonne pratique de définir vos actions à côté des fonctionnalités qui les utilisent

```
└── books\  
    ├── actions\  
    │   books-api.actions.ts  
    │   books-page.actions.ts  
    └── index.ts
```

NgRx Actions



Les groupes d'actions (Actions Groups)

- Vu que le **groupement des actions de même sources est une bonne pratique**, NgRx à partir de la version 13 offre la possibilité de le faire via les **ActionGroup**.
- Pour ce faire, utiliser la fonction **createActionGroup**.
- Elle prend en paramètre un **objet avec deux propriétés** à renseigner:
 - **source** : qui représente **la source** et qui sera **ajouté automatiquement à toutes les actions du groupe**.
 - **events** : qui est un **objet** contenant **les actions du groupe** avec pour **clé** le **nom de l'action** (il sera transformé en camelCase pour récupérer l'action) et pour **valeur** la **props** si elle existe sinon la fonction **emptyProps**.

NgRx Actions

Les groupes d'actions (Actions Groups)



```
export const exampleActions = createActionGroup({  
  source: "Group Actions",  
  events: {  
    "Load Users": emptyProps(),  
    "Add User": props<{ user: User }>(),  
  },  
});
```

```
export const exampleActions = createActionGroup({  
  source: "Group Actions",  
  events: {  
    loadUsers: emptyProps(),  
    addUser: props<{ user: User }>(),  
  },  
});
```

```
this.store.dispatch(exampleActions.addUser({ user }));
```

NgRx Reducers



- Un **Reducer** est une **fonction JS pure**.
- Elle est appelée suite à un **événement** (une **Action**) et reçoit en paramètre **l'état** actuel de l'application (le **state**) ainsi que **l'action dispatché** par le store (qui contient le **payload** s'il existe).
- En fonction du **type de l'action** et de son **payload**, le **reducer** retourne le **nouvel état (state)** de l'application (du store).
- Les reducers **doivent** garder **l'immutabilité** du state, ceci nous permet d'avoir un **historique** des différentes versions de l'état de l'application.

NgRx Reducers



- Afin de créer un reducer et à partir de la version 8 de NgRx, vous pouvez utiliser le helper **createReducer**.
- **createReducer** prend en paramètre **l'état initial du state**, et la **fonction à déclencher** pour une **action donnée**.
- Dans les anciennes versions de RxJs, le travail se faisant avec un switch qui selon le type de l'action exécutait le traitement nécessaire.
- A partir de la version 8 vous pouvez utiliser le helper **on** qui prend en premier paramètre l'action à gérer et en second paramètre la fonction à exécuter. **Cette fonction reçoit le state actuel et l'action et retourne le nouveau state.**

NgRx Reducers



```
const scoreboardReducer = createReducer(  
  initialState,  
  on(ScoreboardPageActions.homeScore,  
    state => ({ ...state, home: state.home + 1 })  
,  
  on(ScoreboardPageActions.awayScore,  
    state => ({ ...state, away: state.away + 1 })  
,  
  on(ScoreboardPageActions.resetScore,  
    state => ({ home: 0, away: 0 })),  
  on(ScoreboardPageActions.setScores,  
  (state, { game }) => ({ home: game.home, away: game.away }))  
);
```

```
export function reducer(  
  state = initialState,  
  action: Scoreboard.ActionsUnion  
) : State {  
  switch (action.type) {  
    case Scoreboard.ActionTypes.IncrementHome: {  
      return { ...state, home: state.home + 1 };  
    }  
    case Scoreboard.ActionTypes.IncrementAway: {  
      return { ...state, away: state.away + 1 };  
    }  
    case Scoreboard.ActionTypes.Reset: {  
      return action.payload;  
    }  
    default: {  
      return state;  
    }  
  }  
}
```

<https://ngrx.io/guide/store/reducers>

NgRx Reducers



- Pensez pour chaque reducer à définir **une interface décrivant la portion du state qu'il gère.**
- Définissez **le state initiale de cette portion.**
- Exportez maintenant votre reducer et définissez le au niveau du store module.

```
export interface AuthState {  
  user: User;  
}  
  
export const initialAuthState: AuthState = {  
  user: null,  
};  
  
export const authReducer = createReducer(  
  initialAuthState,  
  on(AuthActions.loginAction, (state, action) => {  
    return {  
      user: action.user,  
    };  
  })  
);
```

NgRx

Reducers (Root State)

Enregistrement de l'état principale



- L'état de votre application est défini comme **un seul grand objet**.
- L'enregistrement des reducers pour gérer des parties de votre état vous permet de définir uniquement les clés avec des valeurs associées dans l'objet.
- Pour enregistrer le Store global dans votre application, utilisez la méthode **forRoot du StoreModule**.
- La méthode **forRoot()** prend un objet avec comme clé l'identifiant du reducer et comme valeur le reducer.
- En enregistrant le state avec forRoot, vous rendez le state disponible dès le lancement de l'application. `StoreModule.forRoot({home: fromHomeReducer.reducer}),`

<https://ngrx.io/guide/store/reducers>

NgRx

Reducers (Feature State)

Enregistrement de l'état des fonctionnalités



- Les **états fonctionnels (Feature State)** se comportent de la **même manière** que l'état principal.
- Chaque module fonctionnel aura son état qu'il va gérer mais qui sera aussi une partie de l'état principal.
- Pour les **états fonctionnels**, utilisez la méthode **forFeature** de votre **StoreModule**.

```
StoreModule.forFeature("auth", authReducer),
```

NgRx

Reducers (Feature State)

Enregistrement de l'état des fonctionnalités



```
StoreModule.forRoot("auth", authReducer),
```

The screenshot shows the NgRx Store DevTools interface. On the left, the 'Inspector' panel lists commits: '@ngrx/store/init' at 1:38:23.98, '@ngrx/store/update-reducers' at +00:00.01, and '[Login Page] User Login' at +02:24.28. On the right, the 'NgRx Store DevTools' panel displays the 'State' tab in 'Tree' mode. It shows a tree structure under the 'auth' feature, with a single node: 'user (pin): { id: 1, email: "test@angular.io" }'.

<https://ngrx.io/guide/store/reducers>



Sélectionner des éléments du store

- Afin de **récupérer les données de votre state** vous pouvez utiliser plusieurs méthodes.
- Le Store que vous utilisez pour dispatcher des actions vous permet aussi d'accéder au state global de votre application.
- Le **store** étant un **Observable**, vous pouvez vous y **inscrire** et **récupérer le state** de l'application.

```
this.store.subscribe((state) => {
  console.log("state : ", state);
  console.log("autState : ", state['auth']);
});
```



```
state : ► {auth: ...}
autState : ► {user: ...}
```

NgRx

Sélectionner des éléments du store



- Afin de sélectionner la partie que vous voulez, utilisez l'opérateur **map** de rxjs.

```
this.isLoggedIn$ = this.store.pipe(  
    map((state) => !! state["auth"].user)  
);
```

- Le problème ici est que cette **opération va se répéter à chaque fois** alors que le résultat risque d'être le même. Une **fonction pure** qui a le même input retournera toujours le même Output.

- La Solution est donc d'utiliser l'opérateur de RxJs **distinctUntilChanged**.

```
this.isLoggedIn$ = this.store.pipe(  
    map(  
        (state) => state["auth"]  
    ),  
    distinctUntilChanged()  
);
```



Sélectionner des éléments du store

select operator



- Afin de nous aider dans cette démarche, NgRx nous offre l'opérateur select qui réalise un **map** selon une **fonction pure** et ne déclenche le flux de l'observable qui si le résultat change en utilisant **distinctUntilChanged**.
- Elle prend en paramètre le **state** et elle applique une fonction de mapping.

```
this.isLoggedIn$ = this.store.select(  
  (state) => !!state["auth"].user  
);
```

NgRx

Sélecteurs



- Le problème avec le **select** c'est qu'il effectue à **chaque fois l'opération de map**. Dans certains cas, elle peut être très couteuse, sa répétition pose donc un problème de performance.
- Le Store fournit la fonction **createSelector** afin d'optimiser cette sélection. En effet ce helper permet d'avoir de la :
 - Portabilité
 - Mémorisation
 - Composition
 - Testabilité
 - Sécurité de type (Type Safety)

<https://ngrx.io/guide/store/selectors>

NgRx Sélecteurs



- Lors de l'utilisation de **createSelector**, NgRx **garde la trace** des derniers arguments avec lesquels votre fonction de sélection a été appelée.
- Étant donné que les sélecteurs **sont des fonctions pures**, le dernier résultat peut être renvoyé lorsque les arguments correspondent **sans ré invoquer** votre fonction de sélection.
- Cette pratique est connue sous le nom de **mémorisation**.
- Un sélecteur est donc tout simplement une fonction de **mapping** avec de la **mémoire**.
- **createSelector** retourne un objet de type **MemoizedSelector**.

NgRx

Sélecteurs



- La fonction **createSelector** prend en paramètre **un ensemble de sélecteurs (8 maximum)** permettant de sélectionner ce dont vous avez besoin pour mapper votre state suivi de la **fonction de map qui est toujours le dernier paramètre**.
- La fonction de map **récupère comme paramètres le résultat de l'ensemble des sélecteurs** passé en paramètre avec elle.

```
import { createSelector } from '@ngrx/store';

export interface FeatureState {
  counter: number;
}

export interface AppState {
  feature: FeatureState;
}

export const selectFeature = (state: AppState) => state.feature;

export const selectFeatureCount = createSelector(
  selectFeature,
  (state: FeatureState) => state.counter
);
```

NgRx Sélecteurs



```
import { createSelector } from '@ngrx/store';

export interface User {
  id: number;
  name: string;
}

export interface Book {
  id: number;
  userId: number;
  name: string;
}

export interface AppState {
  selectedUser: User;
  allBooks: Book[];
}
```

```
export const selectUser = (state: AppState) => state.selectedUser;
export constselectAllBooks = (state: AppState) => state.allBooks;

export const selectVisibleBooks = createSelector(
  selectUser,
 selectAllBooks,
  (selectedUser: User, allBooks: Book[]) => [
    if (selectedUser && allBooks) {
      return allBooks.filter((book: Book) => book.userId === selectedUser.id);
    } else {
      return allBooks;
    }
  ]
);
```



Sélecteurs de fonctionnalités

featureSelector



- Afin de **centraliser** et de **typer** la partie de votre state qui correspond à une fonctionnalité particulière (généralement le state du module), vous pouvez utiliser les **featureSelector**.
- Pour ce faire, utilisez la méthode **createFeatureSelector** pour passer au **featureStateType** que vous souhaitez et passer lui comme paramètre la **clé** représentant la partie du state que vous voulez utiliser.

```
export const selectUser = (state: AppState) => state.auth;
```

```
export const authFeatureSelector = createFeatureSelector<AuthState>("auth");
```

NgRx

Sélecteurs de fonctionnalités createFeature



- Il existe trois éléments principaux de la gestion globale de l'état avec `@ngrx/store` : les actions, les réducteurs et les sélecteurs.
- Pour un état de fonctionnalité particulier, nous créons un réducteur pour gérer les transitions d'état en fonction des actions et des sélecteurs distribués pour obtenir des tranches de l'état de fonctionnalité.
- Nous devons également définir un nom de fonctionnalité nécessaire pour enregistrer le réducteur de fonctionnalités dans le magasin NgRx.
- Par conséquent, nous pouvons considérer la fonctionnalité NgRx comme un **regroupement** du **nom de la fonctionnalité**, du **reducer** de fonctionnalité **et des sélecteurs** pour l'état de la fonctionnalité particulier.



<https://ngrx.io/guide/store/feature-creators>

NgRx

Sélecteurs de fonctionnalités createFeature



- Afin d'éviter le code répétitif généré avec la création des différents sélecteurs, NgRx propose à partir de la version 16 la fonction **createFeature**
- La fonction **createFeature** réduit le code répétitif dans les fichiers de sélection en générant des sélecteurs enfants pour chaque propriété d'état de fonctionnalité.
- Elle accepte un objet contenant un nom de fonctionnalité et un reducer de fonctionnalité comme argument d'entrée :

```
export const appFeature = createFeature({  
  name: "app",  
  reducer: reducer,  
});
```

<https://ngrx.io/guide/store/feature-creators>

NgRx

Sélecteurs de fonctionnalités createFeature

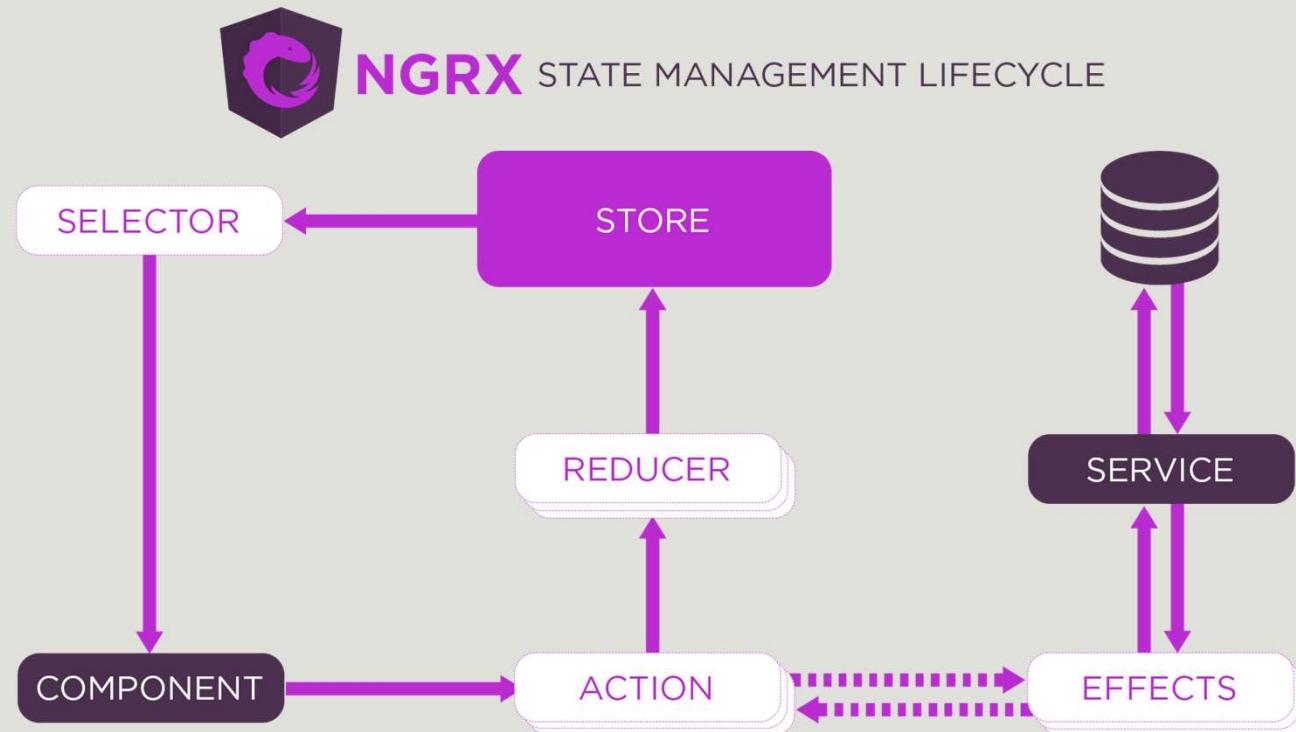


- Afin d'ajouter vos propres sélecteurs, vous pouvez utiliser l'option **extraSelectors**.
- L'option **extraSelectors** accepte une **fonction** qui **prend les sélecteurs générés** comme arguments d'entrée et **renvoie un objet de tous les sélecteurs supplémentaires**.

```
export const todoFeature = createFeature({
  name: "todo",
  reducer: todoReducer,
  extraSelectors: ({ selectTodos }) => {
    const selectCompletedTodos = createSelector(selectTodos, (todos) =>
      todos.filter((todo) => todo.completed)
    );
    const selectIncompleteTodos = createSelector(selectTodos, (todos) =>
      todos.filter((todo) => !todo.completed)
    );
    return { selectCompletedTodos, selectIncompleteTodos };
  },
});
```

NgRx

Les effects



<https://ngrx.io/guide/effects>

NgRx

Les effects



- Dans vos applications, vous avez tout le temps besoin d'interagir avec **les effets de bord** (side effects) comme par exemple communiquer avec une API.
- Dans les applications basées sur les services, les composants sont responsables de l'interaction avec les ressources externes directement via les services.
- Ceci pose un problème architectural. Nous voulons **éviter cette communication direct entre composant et service** et **isoler le traitement des effets de bord**.
- C'est là où interviennent les **effects**.

<https://ngrx.io/guide/effects>

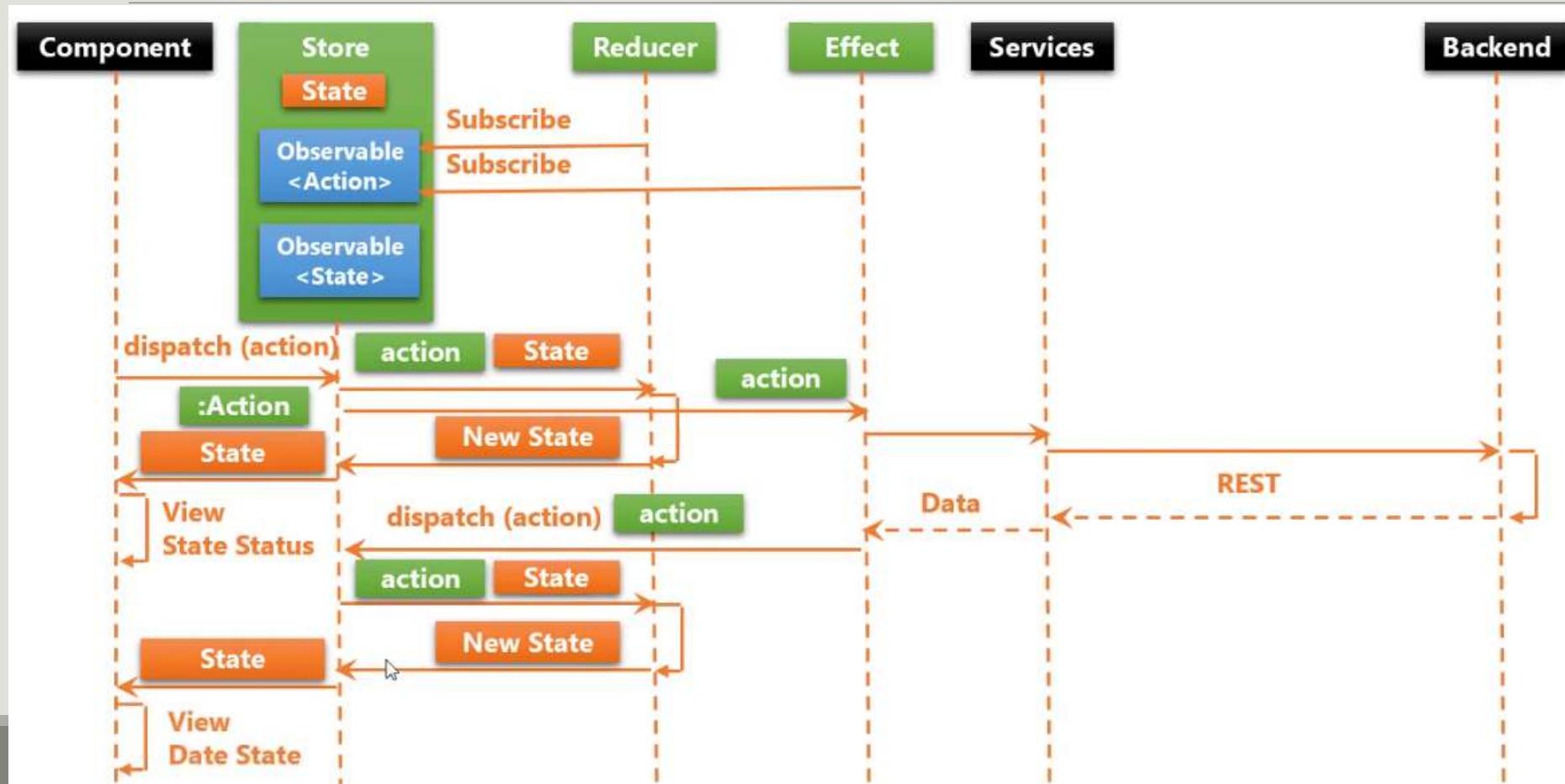
NgRx

Les effects



- Les **effects isolent les effets de bord des composants**, permettant d'avoir des composants plus purs qui sélectionnent les états et déclenchent des évènements.
- Les effects sont des services qui **écoutent** un observable de **chaque action dispatchée depuis le Store**.
- Les effects **filtrent ces actions** en fonction du **type** d'action qui les intéresse. Cela se fait à l'aide d'un opérateur.
- Les effets exécutent des tâches synchrones ou asynchrones et **renvoient une nouvelle action**.

NgRx Architecture



NgRx

Les effects

Installation



- Vous pouvez installer la package en utilisant **npm** via la commande :

npm install @ngrx/effects

- A partir de la version 6 il est recommandé d'utiliser la commande **add** du cli qui va aussi configurer votre application avec le module installé.

ng add @ngrx/effects@latest

- Cette commande fera les fonctionnalités suivantes :

- Mettre à jour **package.json** dependencies avec avec **@ngrx/effects**.
- Exécuter **npm install** pour installer ces dépendances.
- Mettre à jour votre **src/app/app.module.ts** et le tableau de la clé **imports** avec **EffectsModule.forRoot()**.

NgRx

Les effects

Configuration



- Pour configurer votre module au niveau du module principale utiliser la méthode forRoot.
- Dans les modules de fonctionnalités (Feature Module) utiliser la méthode forFeature.

```
imports: [
  //...
  StoreModule.forRoot(reducers),
  StoreDevtoolsModule.instrument({
    maxAge: 25,
    logOnly: environment.production,
  }),
  EffectsModule.forRoot([])
],
```

```
imports: [
  CommonModule,
  ReactiveFormsModule,
  MatCardModule,
  MatInputModule,
  MatButtonModule,
  RouterModule.forChild([{ path: "", component: LoginComponent }]),
  StoreModule.forFeature("auth", authReducer),
  EffectsModule.forFeature([])
],
```

NgRx

Les effects



- Les effects se basent sur **l'écoute sur les actions** qui sont **dispatchées**.
- Afin d'écouter sur les actions, NgRx vous offre **le service Actions** qui étends de la class **Observable** (n'oublier pas NgRx est basée sur RxJs).
- Injecter le et faire le traitement nécessaire selon le type de l'action.

```
import { Actions } from "@ngrx/effects";  
  
constructor(private actions$: Actions) {}
```

NgRx

Les effects



- Maintenant que vous disposez d'un moyen d'écouter sur vos actions, créer votre effects, NgRx vous offre le helper **createEffect**.
<https://ngrx.io/api/effects/createEffect>
- Cette api prend en paramètre **une fonction de mapping** qui **selon le type de votre action**, va **mapper l'observable**.
- **Généralement** vous avez deux possibilité :
 - mapper l'observable à une **action** qui sera traitée par le reducer
 - ou faire un **traitement** et **arrêter** le **process** (exemple : sauvegarder le user connecté dans le local storage).

NgRx

Les effects



- **createEffect** va retourner un nouvel Observable qui retourne une Action. Si vous ne voulez pas le faire, ajouter l'option **{dispatch: false}**
- Afin de tester le type de l'action sans passer par un switch ou un if, NgRx vous offre le helper **ofType**

NgRx

Les effects



```
import { Injectable } from '@angular/core';
import { Actions, createEffect, ofType } from '@ngrx/effects';
import { EMPTY } from 'rxjs';
import { map, mergeMap, catchError } from 'rxjs/operators';
import { MoviesService } from './movies.service';

@Injectable()
export class MovieEffects {

  loadMovies$ = createEffect(() => this.actions$.pipe(
    ofType('[Movies Page] Load Movies'),
    mergeMap(() => this.moviesService.getAll()
      .pipe(
        map(movies => ( { type: '[Movies API] Movies Loaded Success', payload: movies } )),
        catchError(() => EMPTY)))));

  constructor(private actions$: Actions, private moviesService: MoviesService) {}
}
```

<https://ngrx.io/guide/effects>

NgRx

Les effects



```
import { Injectable } from "@angular/core";
import { Actions, createEffect, ofType } from "@ngrx/effects";
import { tap } from "rxjs/operators";
import { AuthActions } from "../ngrx/action-types";

@Injectable()
export class AuthEffects {
  login$ = createEffect(
    () => this.actions$.pipe(
      ofType(AuthActions.loginAction),
      tap(action => localStorage.setItem('user', JSON.stringify(action.user)))
    ),
    {dispatch: false}
  )
  constructor(private actions$: Actions) {}
}
```

<https://ngrx.io/guide/effects>

PWA

- Les **Progressive Web Apps** (PWA) sont des applications Web qui utilisent les dernières technologies pour se rapprocher de la fiabilité et des performances des applications mobiles natives.
- Les PWA **peuvent s'installer sur la tablette ou le mobile** de l'utilisateur et possèdent leur propre icône, sans avoir besoin de passer par un app store. Elles fonctionnent sur tous les supports, elles sont progressives et responsives.
- Elles se lancent en plein écran et peuvent **émettre des notifications push**.
- Elles sont **indexables** par les moteurs de recherche, contrairement aux applications natives, elles sont donc plus simple à trouver.
- Elles sont **partageables avec un lien** sans installation (contrairement à une application mobile).

PWA

Les services workers

- L'une des technologies utilisée par les PWA sont les **service workers**. Ce sont de scripts qui **tournent dans les navigateurs** et sont **responsables de la mise en cache** des ressources nécessaires à une application.
- Ils fonctionnent comme un **proxy du côté du navigateur** : ils **interceptent toutes les requêtes HTTP** sortantes effectuées par l'application et **décident de comment y répondre** (Passer par le cache ou utiliser le réseau).
- Les **service workers sont toujours actifs** même après la fermeture d'un onglet.

PWA

Les services Workers

-
- Quand elles sont lancées depuis l'écran de l'utilisateur, les **services workers** permettent aux PWA de **se charger instantanément**, et ce **même hors ligne**.
 - Elles sont **à jour** grâce aux services workers.
 - Disponibles en **production** uniquement **en HTTPS** et en **développement** sur **localhost**.
 - Les **avantages en termes de performances** de **l'installation de tous nos bundles** Javascript et CSS sur le navigateur de l'utilisateur, c'est qu'elles rendent le démarrage de l'application **beaucoup plus rapide**.

Les services workers dans Angular

- Depuis sa version 5, **Angular nous propose sa propre API** pour **gérer les service workers**.
- Le Service Worker d'Angular peut **mettre en cache toutes sortes de contenus** dans le **Cache Storage** du navigateur.
- Le service worker Angular est comme un cache installé du côté du navigateur de l'utilisateur. Son objectif est de **répondre aux requêtes** effectuées par l'application Angular **pour obtenir des ressources ou des données grâce au cache local**.
- Comme tous les caches, il y a des **règles paramétrables** pour décider **comment** le système de **cache fonctionne**.

Les services workers dans Angular

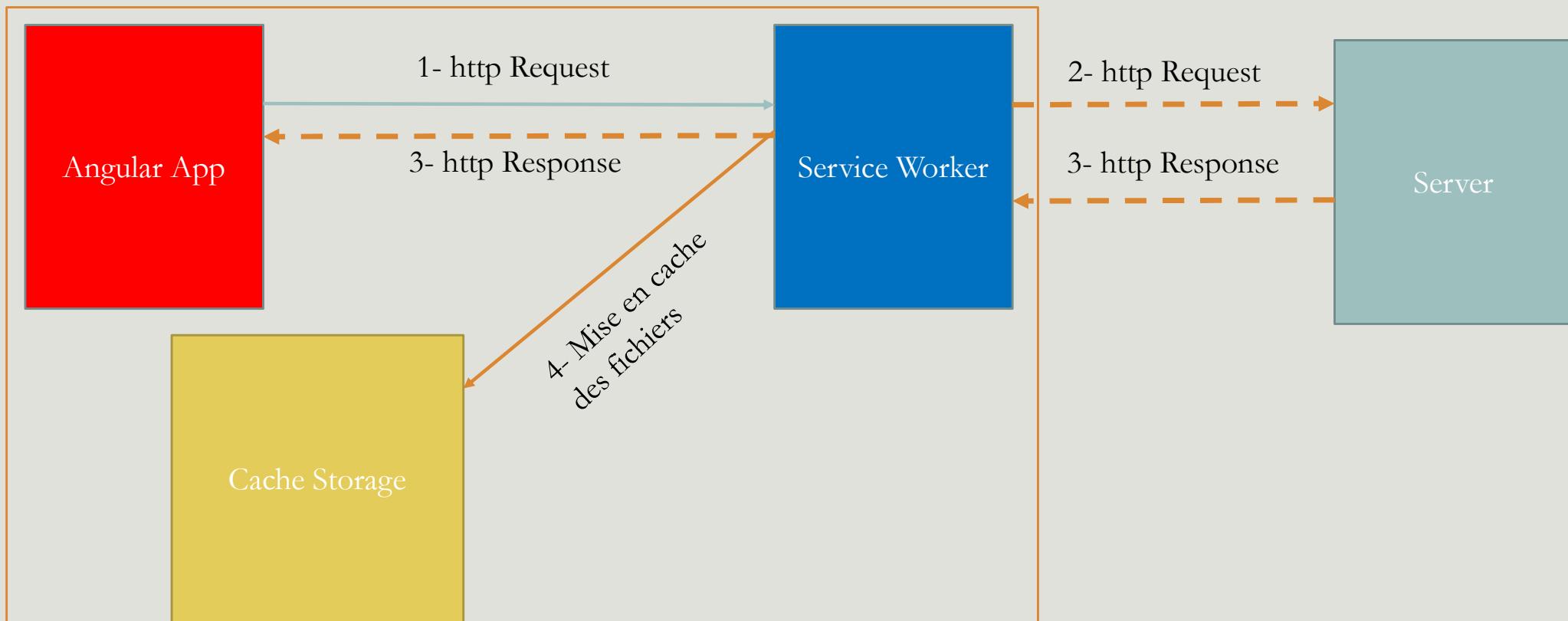
- Le service worker Angular est implémenté de la manière suivante :
- Lors du **premier chargement** de l'application, il **met en cache les parties de l'application configurées pour être cachées**.
- Lorsque **l'utilisateur rafraîchit l'application**, il obtient la **dernière version de l'application mise en cache**.
- La **mise à jour** de l'application est faite **en arrière plan**, pendant que l'utilisateur utilise la version précédente qui est en cache. Lorsque la nouvelle version est totalement chargée, la mise à jour est prête.
- Lorsque l'utilisateur **rechargera** une nouvelle fois **l'application** il aura la **dernière version** en cache et il obtiendra la **version mise à jour**.

Les services workers dans Angular

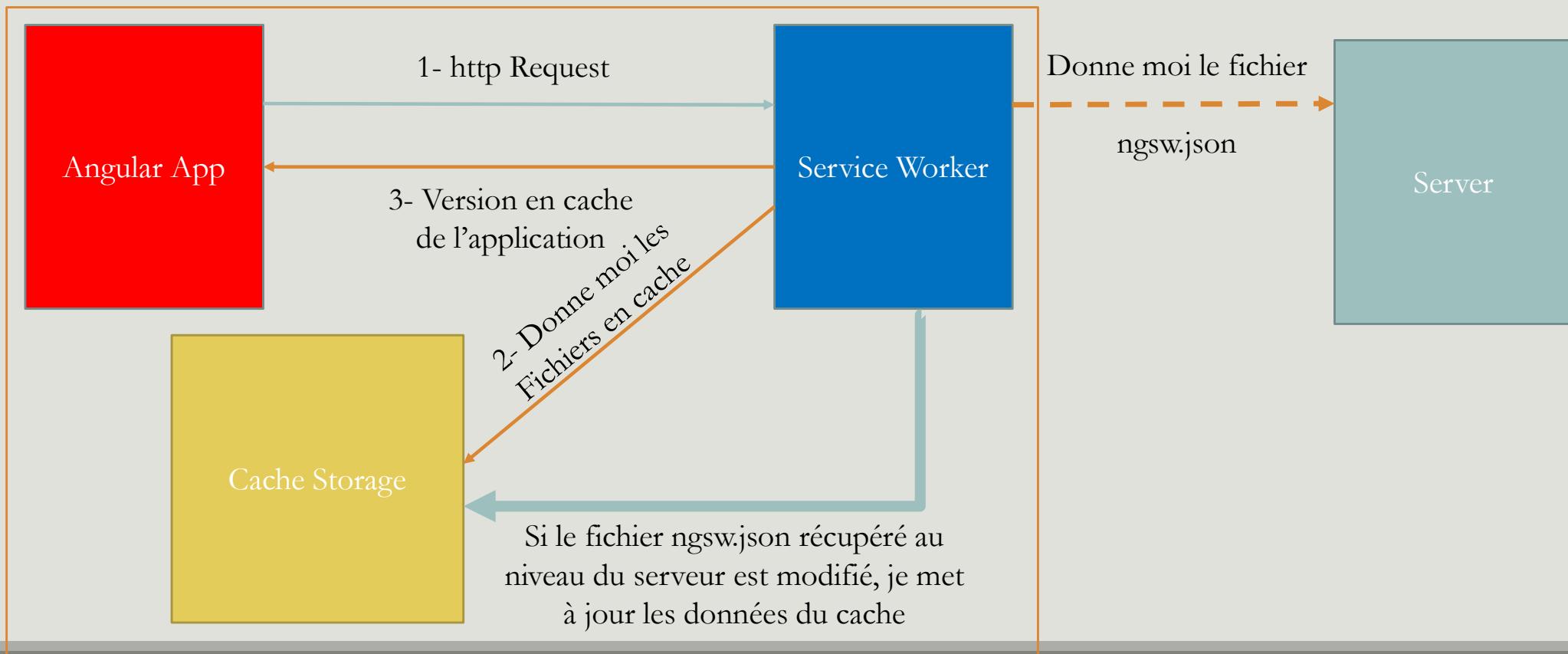
- Pour obtenir ce comportement, le service worker Angular charge un fichier **ngsw.json** (**angular** pour **ng**, et **service worker** pour **sw**) depuis le serveur. Ce fichier **décrit les ressources** à mettre en cache et permet de savoir la version de tous les fichiers mis en cache.
- Lorsque l'application est **mise à jour**, le **contenu de ce fichier change**, et le **service worker est informé qu'une nouvelle version de l'application est disponible** et doit être **téléchargée et mise en cache**.
- Ce fichier est généré à partir d'un fichier de configuration **ngsw-config.json**.

Les services workers dans Angular

1^{er} chargement de l'application



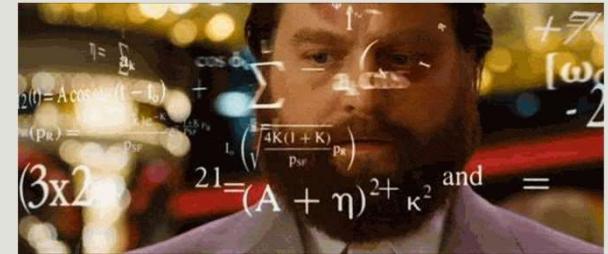
Les services workers dans Angular n éme chargement



Les services workers dans Angular

- Afin de mettre en place votre Service worker utiliser la commande
ng add @angular/pwa --project NOM_DU_PROJET
- Cette commande va permettre d'effectuer les actions suivantes :
 - Ajouter la dépendance **@angular/service-worker**
 - Activer le support du service worker par le CLI
 - Importer et **activer le service worker** dans le module racine **app.module**
 - Mets à jour le fichier **index.html** pour importer le **manifest.webmanifest** et des métas tags pour les couleurs
 - Créer les fichiers **d'icônes** pour pouvoir ajouter la PWA à l'écran d'accueil sur mobile
 - Crée le fichier **ngsw-config.json** qui permet notamment de configurer la mise en cache

Exercice



- Installer le module pwa d'Angular
- Builder votre projet via **ng build**
- Si vous avez le serveur http-client installé, utiliser le pour tester votre application. Sinon installer le via la commande **npm i -g http-server**
- Lancer la commande **http-server -c-1 dist/nomDeVotreProjet**
- Accéder à l'url **localhost:8080**, vérifier que ca marche puis ouvrez le devtools de votre navigateur.
- Allez dans l'onglet **application** et vérifier les éléments **Manifest** et **Service Workers**.
- Mettez vous en mode **offline** et vérifier que votre application marche encore.

Exercice

$$I(t) = A \cos(\omega t - I_0) + \sum_{n=1}^{\infty} \frac{(-1)^n}{n} \frac{e^{-\zeta_n t}}{\omega_n^2 - \omega^2} \sin(\omega_n t)$$
$$(P_R) = \frac{(1-e^{-Kt})}{P_{RF}} e^{-\frac{L}{P_{RF}} t} L \left(\sqrt{\frac{4K(1+K)}{P_{RF}}} \right) P_R$$
$$(3x2) \quad 21 = (A + \eta)^2 + \kappa^2 \text{ and } =$$

Sources Network Performance Memory Application Security Lighthouse Recorder Performance insights

Application

- Manifest
- Service Workers
- Storage

Storage

- Local Storage
- Session Storage
- IndexedDB
- Web SQL
- Cookies
- Private State Tokens
- Interest Groups
- Shared Storage
- Cache Storage

Background Services

- Back/forward cache
- Background Fetch
- Background Sync
- Bounce Tracking Mitigation
- Notifications
- Payment Handler

Service Workers

http://localhost:8080/

Source [ngsw-worker.js](#)

Received 05/08/2023 16:45:41

Status ● #75 activated and is running [stop](#)

Clients http://localhost:8080/ [focus](#)

Push [Push](#)

Sync [Sync](#)

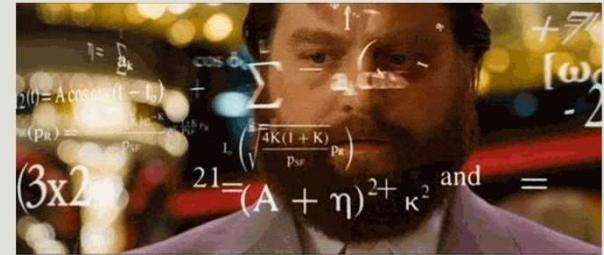
Periodic Sync [Periodic Sync](#)

Update Cycle

| Version | Update Activity | Timeline |
|---------|-----------------|------------|
| ▶ #75 | Install | 1 |
| ▶ #75 | Wait | 1 |
| ▶ #75 | Activate | ██████████ |

Exercice

- Visiter votre dossier de build et vérifier les changements qui ont été réalisés


$$I \approx \sum_{k=1}^{\infty} I_k + \cos \phi \sum_{k=1}^{\infty} -\frac{I_k}{4 \pi k r_0}$$
$$I(t) = A \cos(\omega t - I_0)$$
$$I_R(p_R) = \frac{(A + K)}{p_R} e^{-\frac{(A + K)}{p_R} r_0} \cos\left(\frac{2\pi}{p_R} r_0\right)$$
$$(3x2) \quad 2I = (A + \eta)^2 + \kappa^2 \text{ and } =$$

Les services workers

ngsw-config.json

- Le fichier **ngsw-config.json** permet de **spécifier quels fichiers et quelles données provenant des URLs spécifiées doivent être mise en cache** par le service worker et également **comment ils doivent être mis à jour**.
- Ce fichier est ensuite utilisé par le CLI pendant le build en production (ng build).
- **ngsw-config.json** utilise le format JSON. Tous les chemins de fichiers doivent commencer par /, qui correspond au répertoire de déploiement — généralement dist/<nom-projet> dans les projets CLI.
- Les modèles utilisent un format limité qui sera converti en interne en regex

Les services workers

ngsw-config.json

- ****** permet de matcher **0 ou plusieurs niveaux de chemin** (`/niv1/niv2/../nivn`).
- **?** Correspond exactement à un caractère à l'exception de /
- **!** Marque le modèle comme étant négatif, ce qui signifie que seuls les fichiers qui ne correspondent pas au modèle sont inclus.
- ***** permet de matcher **0 ou plusieurs caractères en excluant /**.
 - `/**/*.html` spécifie tous les fichiers HTML
 - `/*.html` spécifie uniquement les fichiers HTML à la racine

```
{  
  "$schema": "./node_modules/@angular/service-worker/config/schema.json",  
  "index": "/index.html",  
  "assetGroups": [  
    {  
      "name": "app",  
      "installMode": "prefetch",  
      "resources": {  
        "files": [  
          "/favicon.ico", "/index.html", "/manifest.webmanifest", "/*.css", "/*.js"  
        ]  
      }  
    },  
    {  
      "name": "assets",  
      "installMode": "lazy",  
      "updateMode": "prefetch",  
      "resources": {  
        "files": [  
          "/assets/**",  
          "/*.(svg|cur|jpg|jpeg|png|apng|webp|avif|gif|otf|ttf|woff|woff2)"  
        ]  
      }  
    }  
  ]  
}
```

Les services workers

ngsw-config.json

Les AssetGroups

- Les **AssetGroups** sont les ressources qui sont mises à jour en même temps que l'application.
- La propriété **assetGroups** est un **tableau** qui contient des asset groups, qui **décrivent un type de ressources** et la **stratégie de mise en cache**.
- Elles peuvent inclure des ressources chargées depuis le **serveur**, depuis des **CDNs** ou autres **sources externes**.
- Vous pouvez utiliser les **regex** pour matcher des URLs dont seule une partie peut être connue par avance.

```
interface AssetGroup {  
  name: string;  
  installMode?: 'prefetch' | 'lazy';  
  updateMode?: 'prefetch' | 'lazy';  
  resources: {  
    files?: string[];  
    urls?: string[];  
  };  
  cacheQueryOptions?: {  
    ignoreSearch?: boolean;  
  };  
}
```

Les services workers

ngsw-config.json

Les assetGroups - installMode

- Le mode d'installation, **installMode**, permet de paramétrer la **stratégie de mise en cache initiale d'une ressource**.
- Il existe deux types de stratégies de mise en cache :
 - **prefetch** : permet de **télécharger toutes les ressources listées pendant que la version actuelle de l'application est mise en cache**. Cette stratégie permet de s'assurer que toutes les **ressources listées** seront **disponibles**, même **hors connexion**.
 - **lazy** : les ressources ne sont **mise en cache** que **lorsque des requêtes pour les charger sont émises**.

Les services workers

ngsw-config.json

Les assetGroups - updateMode

- Le mode de mise à jour, `updateMode`, permet de paramétriser la stratégie de **mise à jour des ressources mises en cache** lorsqu'une **nouvelle version** de l'application est **disponible**.
- Il existe deux types de stratégies de mise à jour des ressources qui ont changé depuis la dernière version de l'application :
 - `prefetch` : permet de dire au service worker de **télécharger et de mettre en cache** les ressources qui ont été modifiées **immédiatement**.
 - `lazy` : permet de dire au service worker **d'attendre** que ces **ressources soient requêtées** pour les télécharger de nouveau **afin de les mettre à jour** dans le cache. Pour activer ce mode de mise à jour, il **est nécessaire que le mode d'installation soit aussi lazy**.

Les services workers

ngsw-config.json

Les assetGroups - **ressources**

- Les ressources peuvent être de deux types.
- Les **fichiers**, **files**, sont une liste de regex qui match un ou plusieurs fichiers dans le dossier du build (public ou dist généralement).
- Les **urls** sont une **liste de regex** qui **match des urls lors de l'exécution de l'application**. Elles permettent par exemple de **mettre en cache des polices de caractères**.

```
"resources": {  
    "files": [  
        "/favicon.ico",  
        "/index.html",  
        "/manifest.webmanifest",  
        "/*.css",  
        "/*.js"  
    ]  
}
```

Les services workers manifest.webmanifest

- Le fichier **manifest.webmanifest** d'une application Web fournit des informations la concernant dans un fichier JSON afin de pouvoir l'installer sur l'écran d'accueil d'un appareil mobile.
- Les propriétés sont :
 - Le **name** ou le **short_name** sera le nom utilisé sur l'écran d'accueil du mobile sous l'icône de votre application une fois que celle-ci aura été installée.
 - Le **theme_color** définit la couleur du thème de l'application. Cette couleur sera utilisée à certains endroits par le système d'exploitation (Android par exemple, va utiliser cette couleur dans le gestionnaire des applications lancées).

Les services workers manifest.webmanifest

- **background_color** est la couleur de fond de base de l'application web. Cette valeur peut être utilisée par le navigateur comme couleur de fond lorsque le fichier **manifest.webmanifest** est disponible avant que la feuille de style de l'application soit chargée.
- **display** est le mode d'affichage de l'application web sur mobile lorsqu'elle est installée. La valeur recommandée est **standalone** pour la plupart des utilisations. L'application ressemblera à une application native et se comportera comme telle (<https://developer.mozilla.org/en-US/docs/Web/Manifest/display>).
- **icons** sont les images qui vont servir d'icônes pour l'application dans différents tailles sur l'appareil mobile ou la tablette.

Les services workers

Le dossier Build

ngsw.json

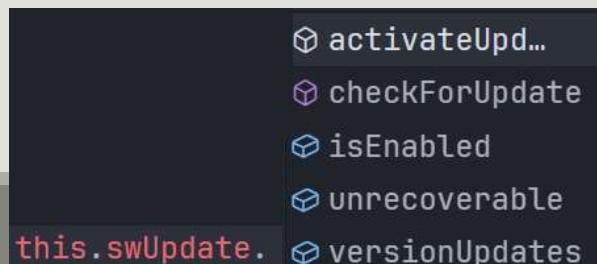
- Ce fichier est généré à **chaque build du projet**.
- Il s'agit du fichier de configuration d'exécution que le Service Worker d'Angular utilise.
- Ce fichier est **construit sur la base du fichier ngs-w-config.json** et contient toutes les informations nécessaires au Service Worker d'Angular pour **savoir au moment de l'exécution quels fichiers il doit mettre en cache et quand**.
- C'est une **version étendue du fichier ngs-w-config.json**, où toutes les **URL génériques** ont été **appliquées et remplacées** par les chemins **de tous les fichiers qui leur correspondaient**.

```
{  
  "configVersion": 1,  
  "index": "/index.html",  
  "assetGroups": [  
    {  
      "name": "app", ...  
      "urls": [  
        "/favicon.ico",  
        "/index.html", ...]  
    },  
    {  
      "name": "assets",...  
      "urls": [  
        "/assets/icons/icon-128x128.png",  
        "/assets/icons/icon-144x144.png", ...  
      ]  
    }  
  ],  
  "hashTable": {  
    "/assets/icons/icon-128x128.png": "b1f06c7d714abb4a0cc4d1f7c954feb26826e4c4",  
    "/assets/icons/icon-144x144.png": "96a0d629765c1bc85d9263c6bc83094ca776d267",  
    ...  
  },  
}
```

Les services workers

Le service swUpdate

- Le service **SwUpdate** permet **d'accéder à plusieurs événements** qui indiquent le moment où le **service worker a trouvé une mise à jour** disponible de l'application **ou lorsqu'il a activé une mise à jour**, c'est-à-dire lorsqu'il sert le contenu de l'application mise à jour.
- Ce service permet trois opérations :
 - D'être **notifié lorsqu'une mise à jour est disponible** : c'est-à-dire une nouvelle version de l'application qui sera chargée lors du rafraîchissement de la page.
 - Demander au service worker de **vérifier si une mise à jour est disponible**.
 - Demander au service worker **d'activer la dernière version**



Les services workers

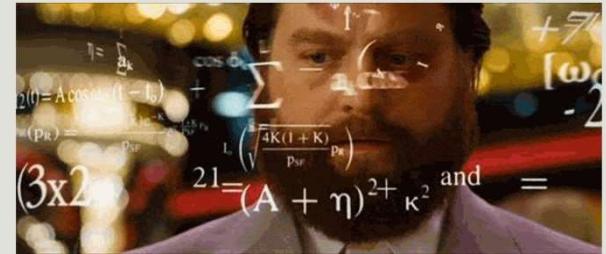
Le service swUpdate

versionUpdate

➤ La propriété **versionUpdates** est un observable de SwUpdate et qui émet quatre événements :

| | |
|---------------------------------------|--|
| VersionDetectedEvent | Émis lorsque le service worker a détecté une nouvelle version de l'application sur le serveur et est sur le point de commencer à la télécharger. |
| NoNewVersionDetectedEvent | Émis lorsque le service worker a vérifié la version de l'application sur le serveur et n'a pas trouvé de nouvelle version. |
| VersionReadyEvent | Émis lorsqu'une nouvelle version de l'application est disponible pour être activée par les clients. Il peut être utilisé pour informer l'utilisateur d'une mise à jour disponible ou l'inviter à actualiser la page. |
| VersionInstallationFailedEvent | Émis lorsque l'installation d'une nouvelle version a échoué. Il peut être utilisé à des fins de journalisation/surveillance. |

Exercice



- Faites en sorte d'avoir une alerte lors de la détection d'une nouvelle version de votre application.
- Cette alerte devra notifier l'utilisateur qu'une nouvelle version de l'application a été détecté et s'il veut recharger la page.

Les services workers

ngsw-config.json

Les dataGroups

- Les dataGroups permettent de mettre en cache la réponse de requêtes à des APIs.
- Les requêtes pour des données ne sont pas versionnées comme les assets. Elles sont mises en cache suivant des paramètres configurés manuellement.

```
export interface DataGroup {  
    name: string;  
    urls: string[];  
    version?: number;  
    cacheConfig: {  
        maxSize: number;  
        maxAge: string;  
        timeout?: string;  
        strategy?: 'freshness' | 'performance';  
    };  
    cacheQueryOptions?: {  
        ignoreSearch?: boolean;  
    };  
}
```

Les services workers

ngsw-config.json

Les dataGroups

- **name**, permet d'identifier chaque dataGroup de manière unique.
- **urls** contient un tableau avec des regex qui vont permettre la mise en cache des réponses des requêtes aux URLs qui matcheront ces regex.
- **version** permet d'indiquer que le format des données de l'API a changé et que l'ancien n'est plus compatible avec l'application. Il permet donc d'indiquer au service worker qu'il est nécessaire de ne pas utiliser les données mises en cache depuis l'API. Cette propriété est un nombre qui s'incrémente manuellement en partant de 0.

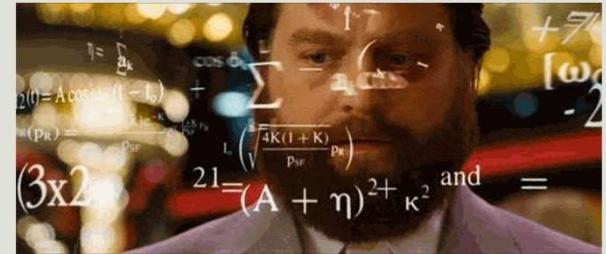
Les services workers

ngsw-config.json

Les dataGroups

- **maxSize** correspond au nombre maximal de réponses d'API en cache.
- **maxAge** permet d'indiquer combien de temps les données mises en cache peuvent être conservées. Elle se paramètre par exemple : 2d12h (avec d pour days, h, m, s et u pour heures, minutes, secondes et millisecondes respectivement).
- **strategy** peut être définie entre, **performance** et **freshness**.
 - **Performance**: le service worker va répondre en priorité avec les données en cache.
 - **freshness** le service worker va d'abord effectuer une requête réseau et ne répondra avec le cache que si cette requête est en timeout.
- **timeout** permet de spécifier la durée avant de considérer que le réseau est en timeout. Le service worker Angular va attendre ce temps avant d'utiliser la réponse en cache (si il est configuré en stratégie freshness comme nous allons le voir).

Exercice



- Ajouter dans votre composant l'affichage de la liste des users récupérées depuis <https://jsonplaceholder.typicode.com/users>
- Faites en sorte que vos appels d'apis soient cachées.
- Rebuilder votre application et tester son fonctionnement en offline.

Les services workers

Résumé

- Pour résumer, voici comment les nouvelles versions d'application sont gérées par Angular Service Worker :
 - Pour chaque **nouvelle version**, un **nouveau ngsw.json** est disponible
 - Pour le **premier rechargement** de l'application après le déploiement de la nouvelle version - Angular Service Worker **détecte le nouveau ngsw.json** et **charge tous les nouveaux fichiers en arrière-plan**
 - Pour le **deuxième rechargement** après le déploiement de la nouvelle version - **l'utilisateur voit la nouvelle version**

Les Push Notifications

- Les **push notifications web** sont des informations que vous pouvez envoyer à vos utilisateurs s'ils ont accepté d'en recevoir, et ce même si votre site est fermé.
- Pour recevoir une notification il suffit que leur navigateur soit ouvert, même en tâche de fond.
- Vous pouvez penser aux **notifications push sur mobile**.
- Afin de faire ça, vous avez besoin d'un **script qui fonctionne en tache de fond**. C'est le **Service Worker**

Les Push Notifications

- Les **push notifications web** sont une combinaison de **Web Push** et de **notification**.
- **L'API de notification** du browser vous permet **d'afficher des notifications à votre utilisateur**. Ceci est **indépendant de l'API Web Push**.

```
if ('Notification' in window) {  
    console.log(`Vous posséder l'API de Notification`);  
    if (window.Notification.permission === 'granted') {  
        console.log('Permission accordée');  
        new window.Notification('Ceci st ma première notification!');  
    }  
}
```

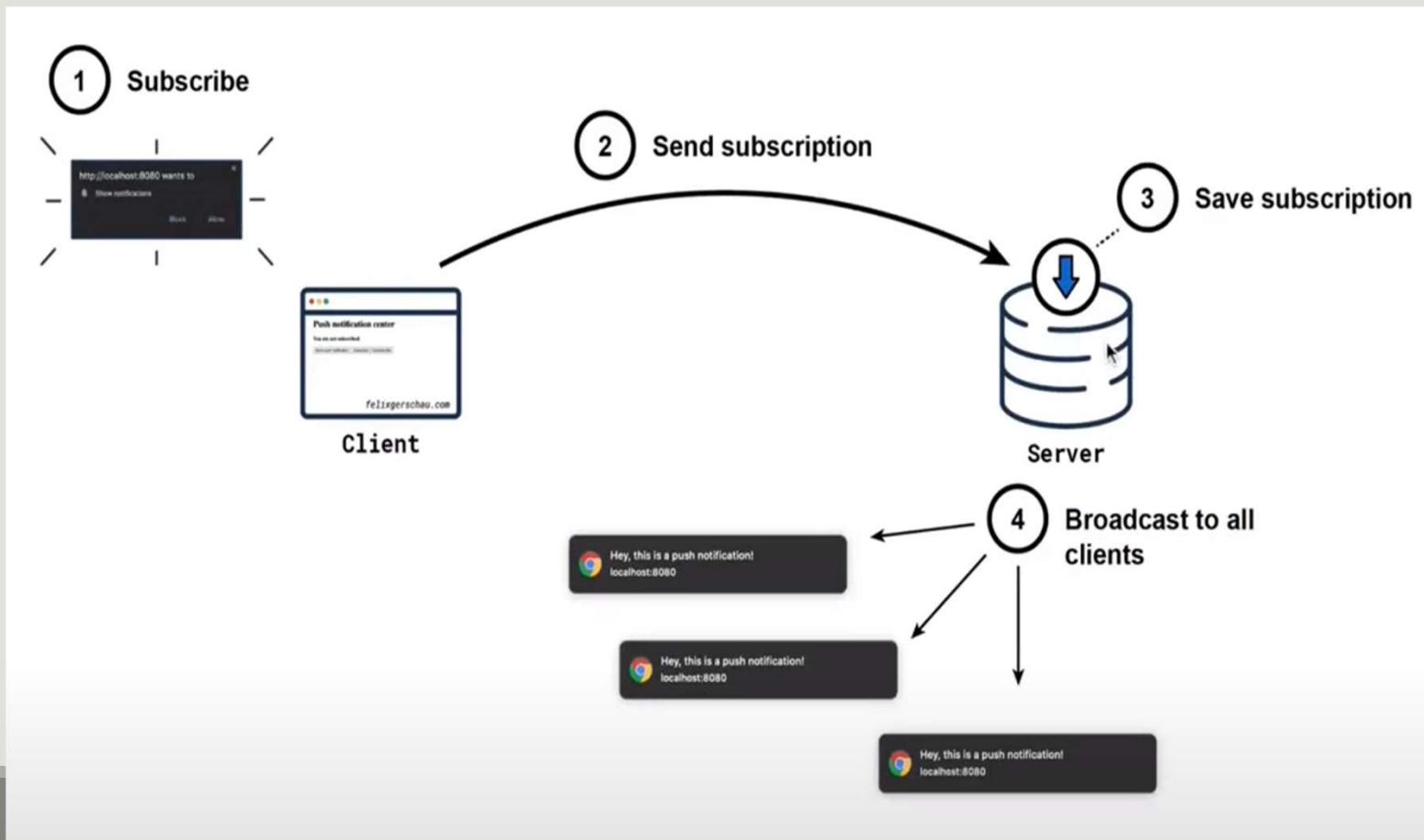
Les Push Notifications

- Le **push API** permet aux **applications Web** de **recevoir des messages d'un serveur**, même **lorsque le site Web n'est pas ouvert**.
- Nous pouvons utiliser cette API pour recevoir des messages puis les afficher, en utilisant l'API de notification.
- Cette **API est uniquement disponible via les service workers**, qui permettent d'exécuter du code en arrière-plan lorsque l'application n'est pas ouverte.

Les Push Notifications

- Cependant, **les navigateurs ne nous permettent pas d'envoyer les notifications** de notre **serveur directement au navigateur** de l'utilisateur.
- Ceci est du à la peur de **spammer** l'utilisateur avec des notifications.
- Au lieu de cela, **seuls certains serveurs spécifique au navigateur pourront envoyer des notifications** à un navigateur donné.
- Ces serveurs sont connus sous le nom de **Browser Push Service**.
- Notez que par exemple, le service Push utilisé par Chrome (Firebase Cloud Messaging) est différent de celui utilisé par Firefox (autoupush).

Les Push Notifications



Les Push Notifications

- Afin de pouvoir délivrer un message à **un utilisateur donné et uniquement à cet utilisateur**, le **Service Push identifie l'utilisateur de manière anonyme**, garantissant la **confidentialité de l'utilisateur**.
- Afin que le Service Push puisse analyser le comportement du serveur, il doit pouvoir l'identifier.
- Afin de faire ça on va utiliser le protocole **VAPID** (Voluntary Application Server Identification)

Les Push Notifications VAPID

- Une paire de clés VAPID est **une paire de clés publique/privée** cryptographique utilisée de la manière suivante :
- **La clé publique** est utilisée comme **identifiant unique du serveur pour abonner l'utilisateur** aux notifications envoyées par ce serveur
- **La clé privée doit être gardée secrète** et utilisée par le serveur d'application pour **signer les messages**, avant de les envoyer au service Push pour livraison.
- Afin de générer les clés, installer la librairie web-push côté serveur

npm install -g web-push

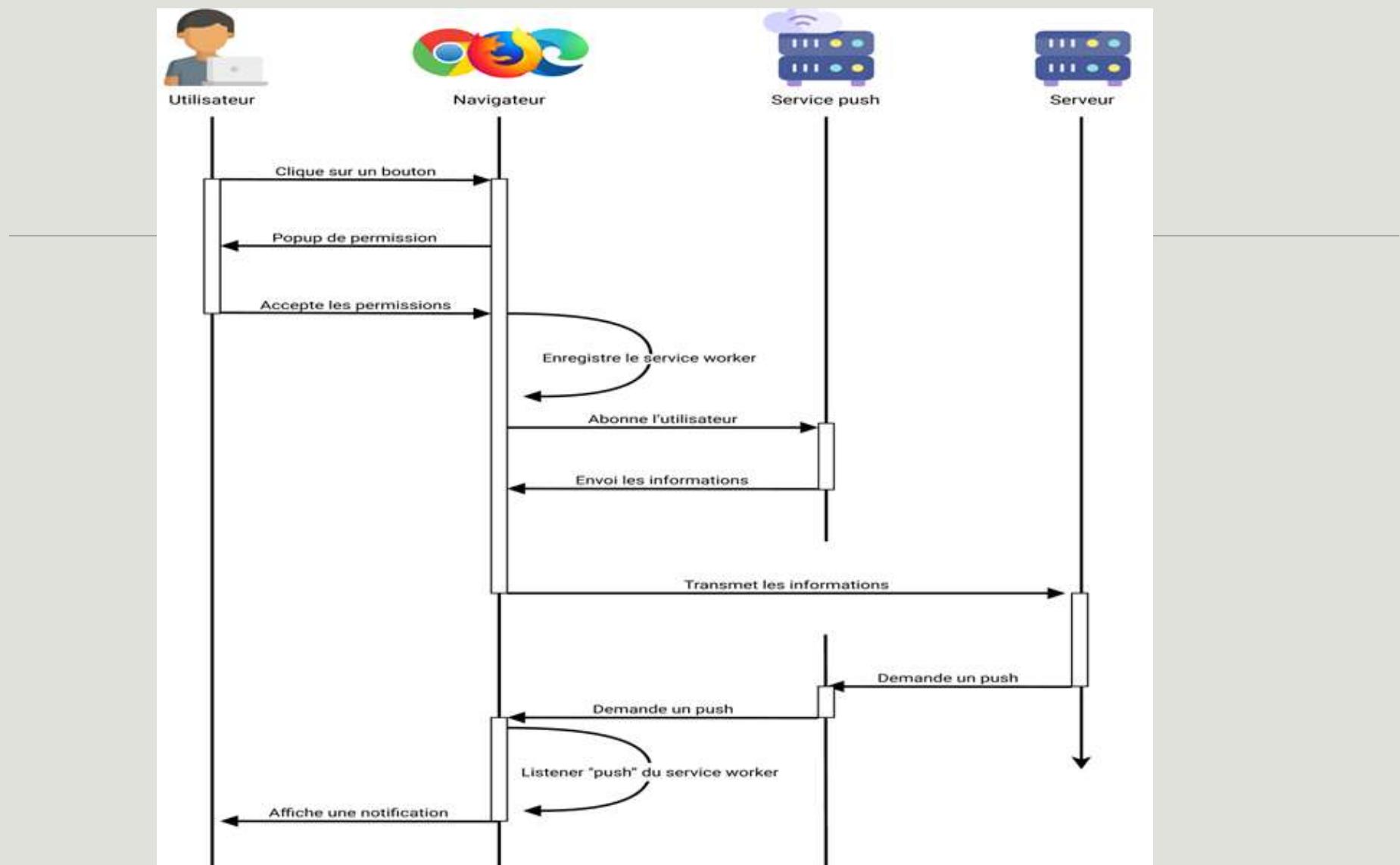
- Pour générer la paire de clés nécessaire au protocole VAPID :

web-push generate-vapid-keys --json

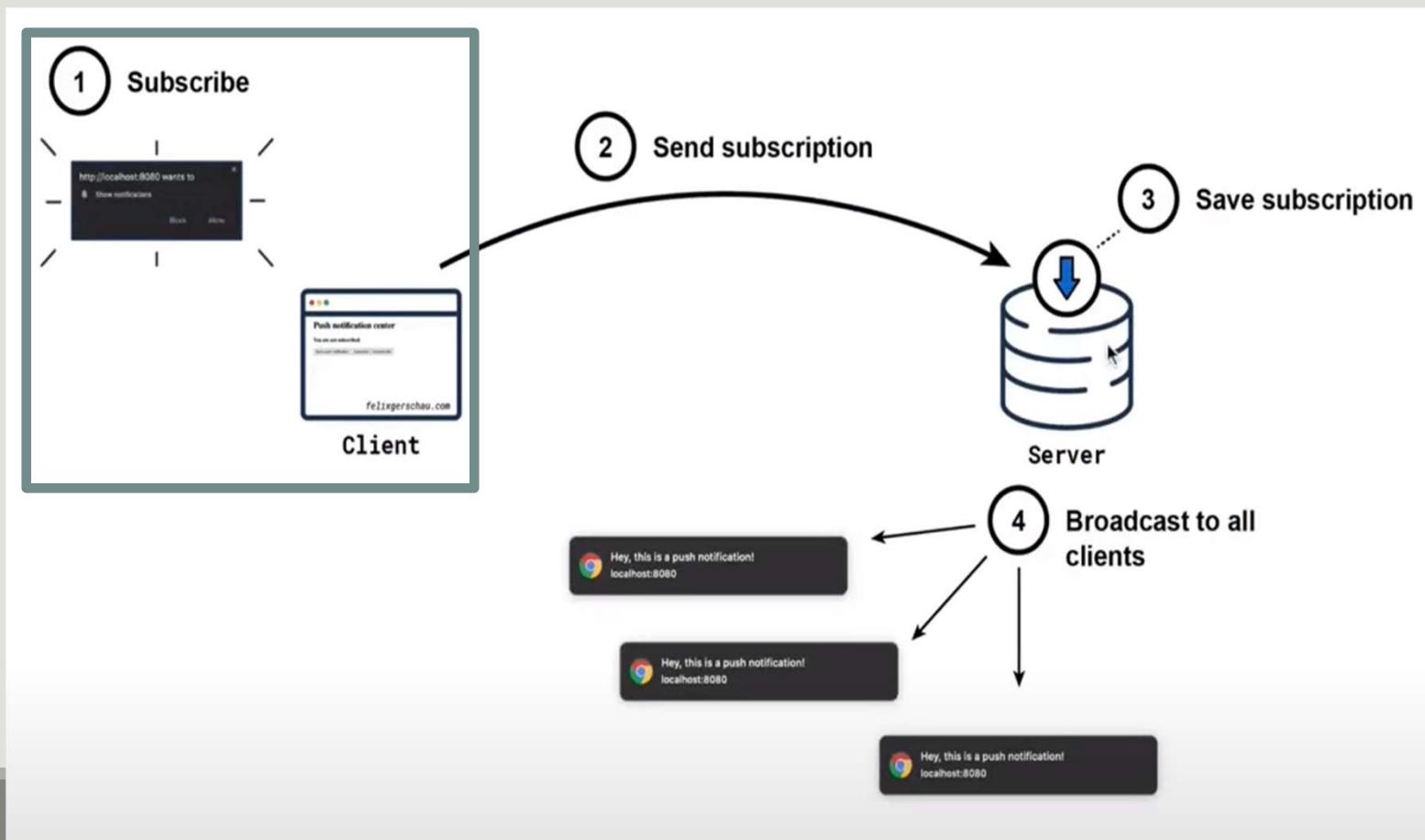
Les Push Notifications VAPID

- Pour notre serveur de notification, voici les deux clés utilisées :

```
vapidKeys = {  
    publicKey:  
        'BCucKI9WG2aEbOhigJ6y_3Z28GIe0jp9QjaZtXXUABjGoUMup6IoYUrTSM  
        h72MxP3t6eVoV6cZ1doEGMm9cRJ2Q',  
    privateKey: 'OQazMChcws-F21vMTt81rlpRr4eZsWkR9yQlZAf0IW0',  
};
```



Les Push Notifications



Les Push Notifications

1- Inscription au service push

- Côté client : autorisation et PushSubscription
- La première étape est d'inscrire un utilisateur pour qu'il puisse recevoir des notifications de votre application.
- Il faut d'abord obtenir son autorisation avec une pop-up spécifique, native au navigateur.
- Ensuite, il est nécessaire d'obtenir du navigateur une **PushSubscription**.
- Cette **PushSubscription** contient toutes les informations nécessaires pour pouvoir envoyer une notification à un utilisateur donné.
- Cette inscription utilise l'API push.

Les Push Notifications

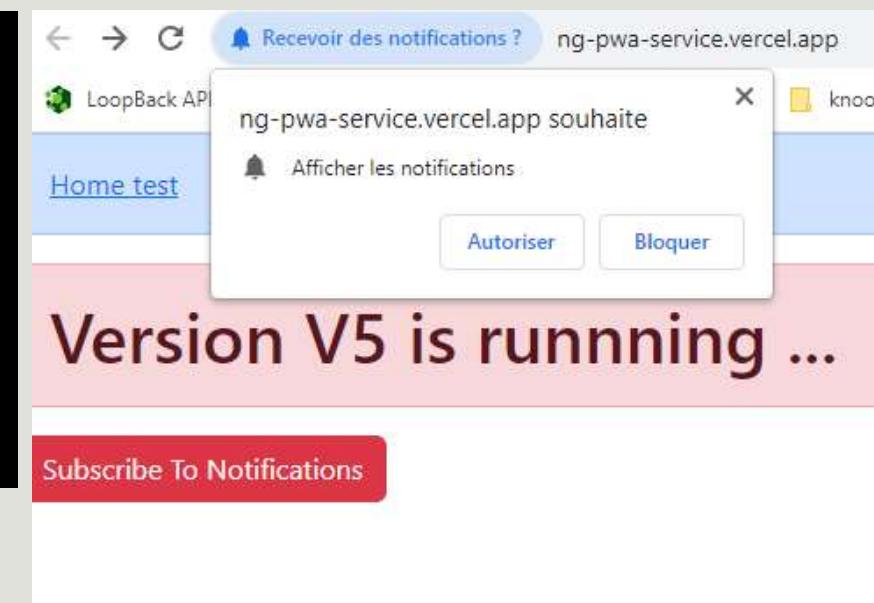
Quelles sont les étapes pour permettre l'activation des notifications web push ?

- Afin de recevoir cette **PushSubscription**, injecter le service SwPush offert par votre **ServiceWorkerModule**.
- Utilisez ensuite sa méthode **requestSubscription** qui prend en paramètre un **objet d'option** dont le **paramètre obligatoire est la propriété serverPublicKey** qui correspondra à la **clé publique de votre VAPID**.
- Lorsque vous lappelez, cette méthode va déclencher un popup demandant à l'utilisateur s'il accepte ou pas d'être notifié (ceci se fait une seule fois).
- S'il accepte la fonction retourne une **promesse** qui va émettre la **PushSubscription**.
- **S'il n'accepte pas vous n'aurez aucun moyen de lui redemander.**

Les Push Notifications

Quelles sont les étapes pour permettre l'activation des notifications web push ?

```
this.swPush
.requestSubscription({
  serverPublicKey: VAPID.publicKey,
})
.then((sub:PushSubscription) => {
  // Faites ce que vous voulez avec cet objet
})
.catch((err) => {//en cas de refus}
);
```



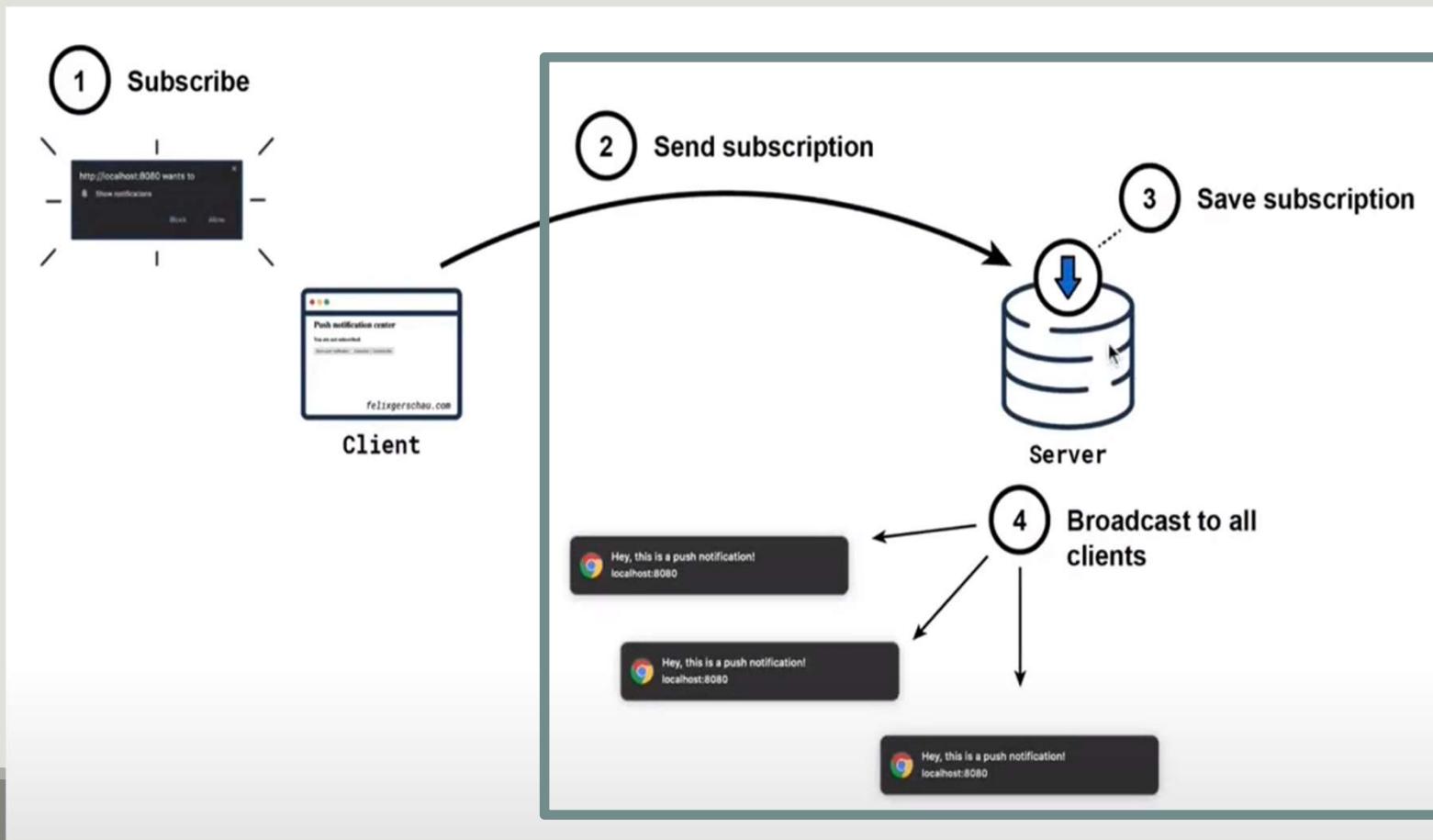
Les Push Notifications

Quelles sont les étapes pour permettre l'activation des notifications web push ?

- L'objet Push Subscription contient les informations suivantes :

```
{  
  subscription: {  
    endpoint: 'https://fcm.googleapis.com/fcm/send/e9YicbR...',  
    expirationTime: null,  
    keys: {  
      p256dh: 'BJIRfMw6Va3GG2u...',  
      auth: 'xBnPrD-I0_xNzikHRuxCxg'  
    }  
  }  
}
```

Les Push Notifications



Les Push Notifications

Quelles sont les étapes pour permettre l'activation des notifications web push ?

- 2 - Le client envoi une demande d'inscription au serveur en y incluant l'objet subscription.
- 3 - Côté serveur : Le serveur doit sauvegarder le subscription
- Votre serveur **reçoit la PushSubscription** envoyée par votre application cliente et **l'enregistre habituellement dans une base de données pour pouvoir envoyer des notifications à l'utilisateur.**
- Pour envoyer une notification, vous allez devoir **passer par un service push** qui va ensuite l'envoyer à l'utilisateur, et ce à des fins de contrôles.
- Si le **navigateur n'est pas en ligne**, il va la **mettre en attente** jusqu'à ce que l'utilisateur l'ouvre.

Les Push Notifications

Quelles sont les étapes pour permettre l'activation des notifications web push ?

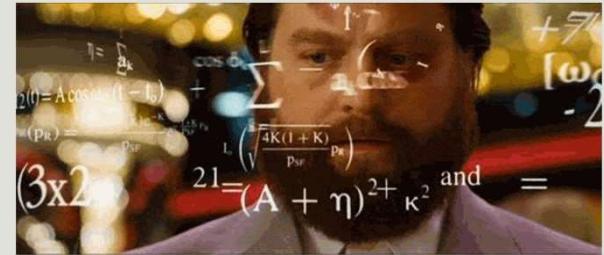
```
subscribeToNotifications() {
  this.swPush
    .requestSubscription({
      serverPublicKey: VAPID.publicKey,
    })
    .then((sub:PushSubscription) => {
      console.log({sub});
      this.http
        .post('https://nest-push-ghxv.vercel.app/notifications', sub)
        .subscribe((data) => console.log({ data: data }));
    })
    .catch((err) =>
      console.error('Could not subscribe to notifications', err)
    );
}
```

```
private sendPushNotification(subscription) {
  webpush
    .sendNotification(
      subscription,
      JSON.stringify({
        notification: {
          title: 'Our first push notification',
          body: 'Here you can add some text',
        },
      }),
      options,
    )
    .then((log) => {
      console.log('Push notification sent.');
      console.log(log);
    })
    .catch((error) => {
      console.log(error);
    });
}
```

Exercice

- Connecter vous au service de notification suivant et tester la réception d'une notification.

<https://nest-push-ghxv.vercel.app/notifications>



Angular

Les stratégies de compilation

- Il existe **deux techniques de compilation** dans Angular:
 - **Just-In-Time (JIT) Compilation:** La compilation JIT se produit à l'exécution du code. Le compilateur Angular est intégré au navigateur et compile le code à chaque fois que l'application est lancée. Cela signifie que le code est compilé en temps réel.
 - **Ahead-of-Time (AOT) Compilation:** La compilation AOT se produit à la génération du code, avant que l'application ne soit exécutée. Le compilateur Angular compile le code en un fichier JavaScript statique qui est exécuté directement par le navigateur sans nécessiter de compilation supplémentaire.

Angular

Les stratégies de compilation

JIT

Compile your code with plain
TypeScript (tsc)

Decorators (@Component, etc)
invoke the compiler and
generate Ivy static fields

*Truly just in time - compilation
happens when the field is read*

AOT

ngc does at build time
what the decorators would
have done at runtime

Avoid the cost of runtime
compilation

Angular

Les stratégies de compilation

Pourquoi choisir le AOT

- **Rendu plus rapide Avec AOT:** le navigateur télécharge une **version pré-compilée** de l'application. Le navigateur charge le code exécutable afin qu'il puisse restituer l'application immédiatement, sans attendre de compiler l'application au préalable.
- **Moins de requêtes asynchrones** Le compilateur intègre des modèles HTML externes et des feuilles de style CSS dans le JavaScript de l'application, **éliminant ainsi les requêtes ajax distinctes pour ces fichiers source.**
- **Taille de téléchargement du framework Angular plus petite** Il n'est pas nécessaire de télécharger **le compilateur Angular** si l'application est déjà compilée. Le compilateur est à peu près la **moitié d'Angular lui-même**, donc l'omettre réduit considérablement la charge utile de l'application.
- **Cependant ceci n'est pas toujours valide pour les grandes applications** puisque on charge le code compilé qui **est plus lourd. PENSEZ AU LAZY LOADING**

<https://angular.io/guide/aot-compiler#choosing-a-compiler>

Angular

Les stratégies de compilation

Pourquoi choisir le AOT

- **Déetecter les erreurs de modèle plus tôt** Le compilateur AOT détecte et signale les erreurs de liaison de modèle **pendant l'étape de construction** avant que les utilisateurs ne puissent les voir.
- **Meilleure sécurité** AOT **compile les modèles** et composants HTML dans des fichiers JavaScript bien avant qu'ils ne soient servis au client. **Sans modèles à lire et sans évaluation HTML ou JavaScript risquée côté client**, il y a **moins de possibilités d'attaques par injection**.

Mesurer les performances

- Afin **d'optimiser le chargement** de votre application, vous devez faire en sorte d'avoir le **bundle le plus optimisé possible**.
- Afin d'analyser votre bundle, vous pouvez utiliser l'outil source-map-explorer :
<https://github.com/danvk/source-map-explorer>
- Afin d'installer cet outil lancez la commande :
npm install source-map-explorer --save-dev
- Lancez le build en incluant source-map : **ng build --source-map**
- Vous allez remarquez l'existence de **fichiers '.map'**

Mesurer les performances

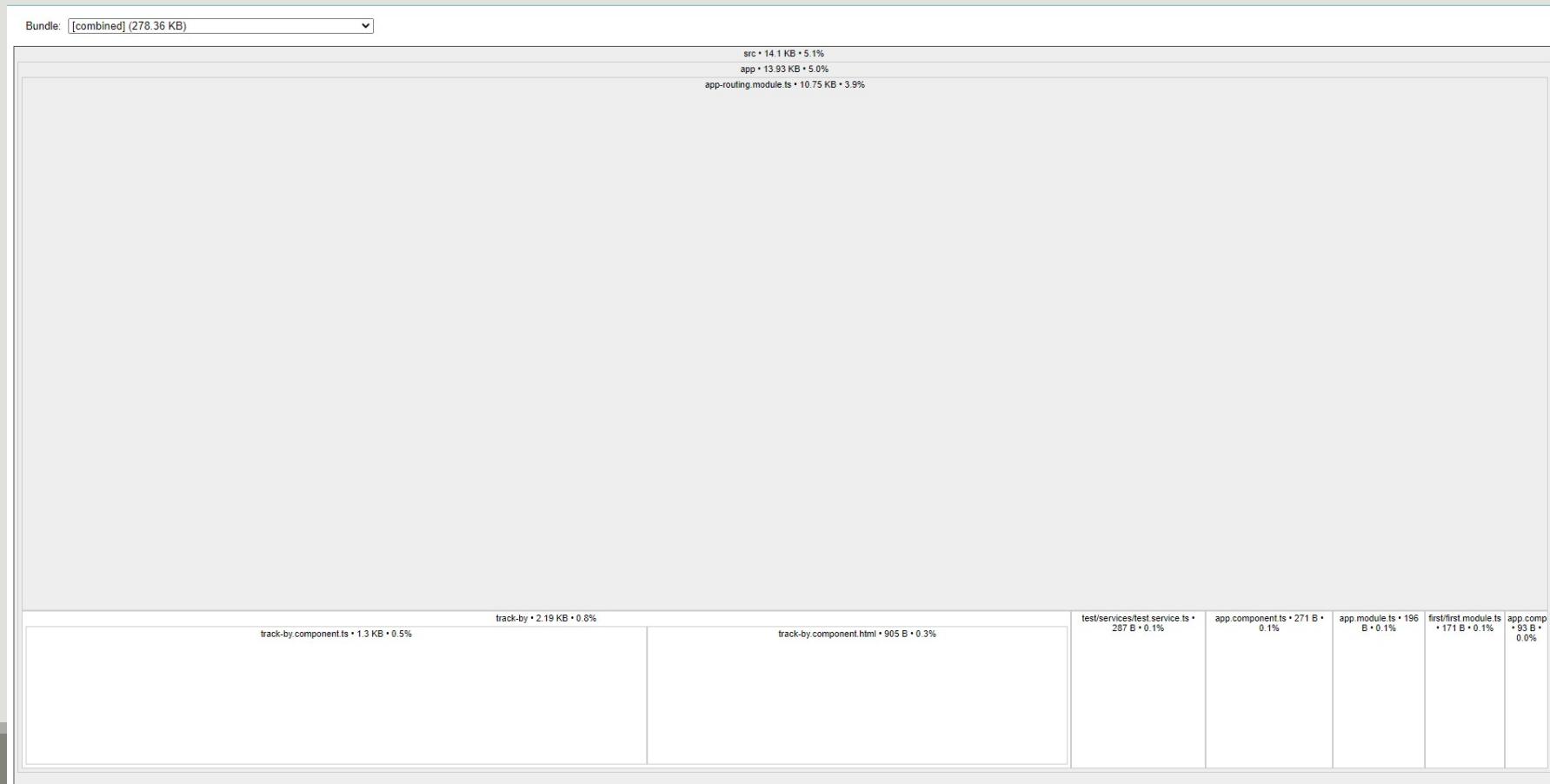
- Maintenant pour visualiser la sourceMap de votre code lancer la commande :

source-map-explorer dist//*.js**

- Préférez ajouter la commande dans votre fichier **package.json**

```
"explorer":"source-map-explorer dist/**/*.js",
"explorer:main":"source-map-explorer dist/**/main*.js"
```

Mesurer les performances



Mesurer les performances

- Vous pouvez utiliser le plugin **lighthouse**
- Il vous permet de **générer un rapport** sur votre application sur plusieurs catégories

<https://chrome.google.com/webstore/detail/lighthouse/blipmdconlkpinefehnmjammfjpmpbjk?hl=fr>

The image shows the Lighthouse extension interface in Google Chrome. It has three main sections: 'Mode' (Navigation (Default) selected), 'Device' (Mobile selected), and 'Categories' (Performance, Accessibility, Best practices, SEO, and Progressive Web App all checked). A large blue button at the top right says 'Analyze page load'.

Generate a Lighthouse report

Analyze page load

Mode [Learn more](#)

Navigation (Default)
 Timespan
 Snapshot

Device

Mobile
 Desktop

Categories

Performance
 Accessibility
 Best practices
 SEO
 Progressive Web App

Mesurer les performances

- Maintenant pour visualiser la **sourceMap** de votre code lancer la commande :

source-map-explorer dist//*.js**

- Préférez l'ajout de la commande dans votre fichier **package.json**

```
"explorer":"source-map-explorer dist/**/*.js",
"explorer:main":"source-map-explorer dist/**/main*.js"
```

Mesurer les performances

- Dans **angular.json**, vous pouvez configurer **source-map** avec la propriété **sourceMap**.
- Elle prend comme **paramètre** un **booléen** ou un **objet** avec les **4 options possibles**

```
"development": {  
    "buildOptimizer": false,  
    "optimization": false,  
    "vendorChunk": true,  
    "extractLicenses": false,  
    "sourceMap": {  
        "hidden": false,  
        "scripts": true,  
        "styles": true,  
        "vendor": false  
    },
```

Mesurer les performances

Quelques techniques pour améliorer votre Bundle Size

- En analysant le contenu de votre bundle, vous allez remarquer que certaines **bibliothèques** prennent **beaucoup d'espace**.
- Si l'une de vos bibliothèque prend beaucoup d'espace, que'elle est implémenté en CommonJS et que vous l'utilisez pour juste **une ou deux fonctionnalités, essayer de les implémentez**.
- Si elle présente une **grande complexité**, essayez de chercher une autre **bibliothèque moins lourde**.
- Préférez les **Bibliothèques ES6 et +**.

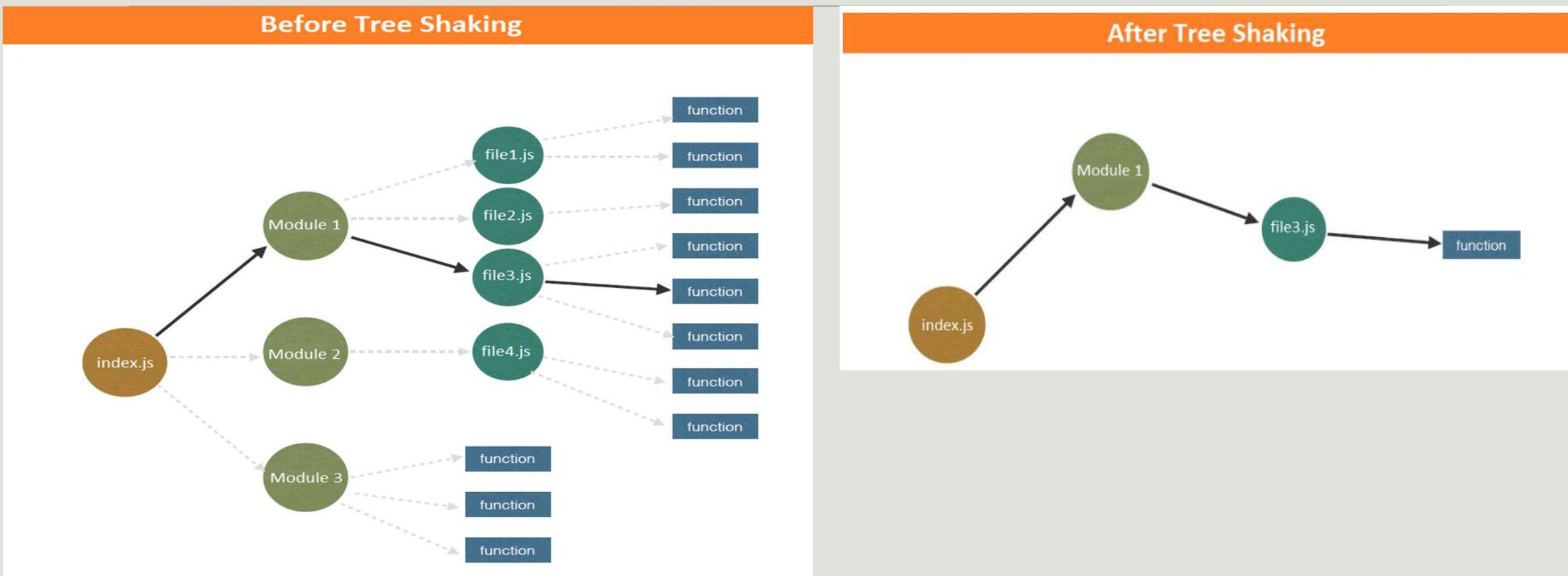
Mesurer les performances

Quelques techniques pour améliorer votre Bundle Size

- Vous devez changer votre code afin qu'il s'aligne et qu'il soit optimal pour le **tree-shaking**.
- Le **tree-shaking** est le mécanisme qui permet **d'éliminer le code mort**; C'est-à-dire **le code qui ne sert à rien**.
- **Webpack** lors de l'opération du bundling va **détecter le dead** code et le **marquer comme ‘module non utilisé’**.
- Ensuite des outils comme **UglifyJS**, vont **uglifyer** votre code et en même temps **éliminer le dead code**.
- L'idée est donc **d'aider Webpack à trouver ce code inutile**.

Mesurer les performances

Quelques techniques pour améliorer votre Bundle Size



Mesurer les performances

Quelques techniques pour améliorer votre Bundle Size

- **N'importez jamais toutes les fonctions de votre bibliothèques.**
- Ceci impliquera que vous avez besoin de toutes la bibliothèque. Ceci implique que leur code, même si vous **ne l'utilisez jamais**, sera inclus dans votre bundle et **ne sera pas comptabilisé comme dead code**.

```
import * as moment from 'moment';
import * as _ from 'lodash';
```

- Dans le cas ici de moment, la solution est d'implémenter votre fonction ou d'utiliser une bibliothèque ES6+ telle que date-fns et qui soit compatible avec le tree-shaking

Mesurer les performances

Quelques techniques pour améliorer votre Bundle Size

- Pour loadash, préférez ‘**lodash-es**’ qui **exporte chaque fonction à part**.

```
import * as _ from 'lodash';

import { pick, clone } from 'lodash-es';
```

Mesurer les performances

Quelques techniques pour améliorer votre Bundle Size

- Pour la **partie css**. Si vous importez des fichiers css au niveau de vos composants plusieurs fois, **ceci impliquera la duplication de ses styles**.
- **N'importez que ce qui est spécifique dans votre composant.**

Angular Zone et Change Détection

AYMEN SELLAOUTI

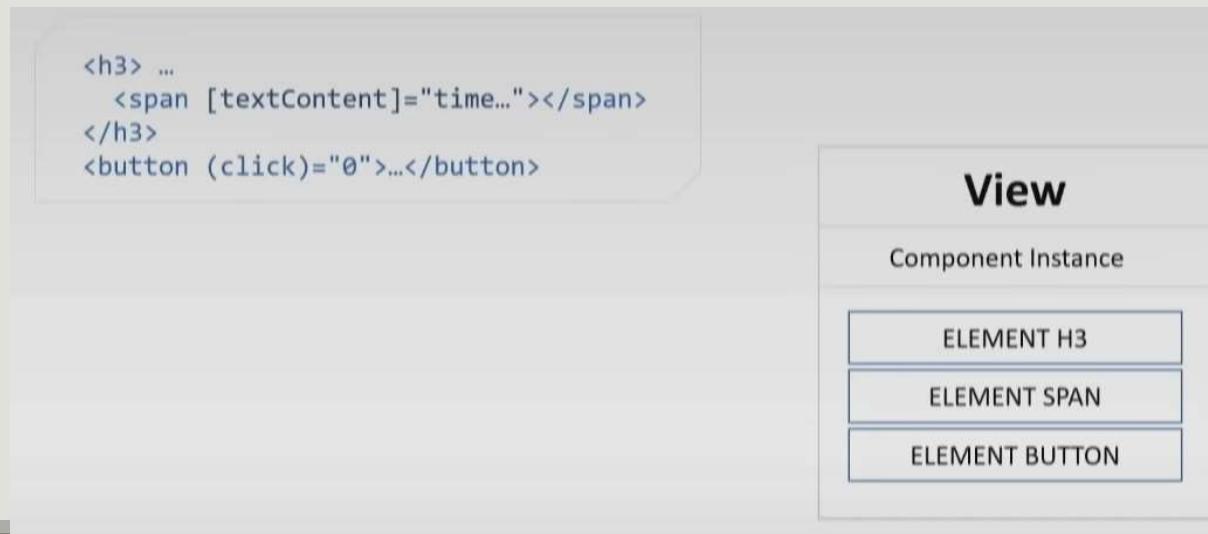
Change Detection

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**.

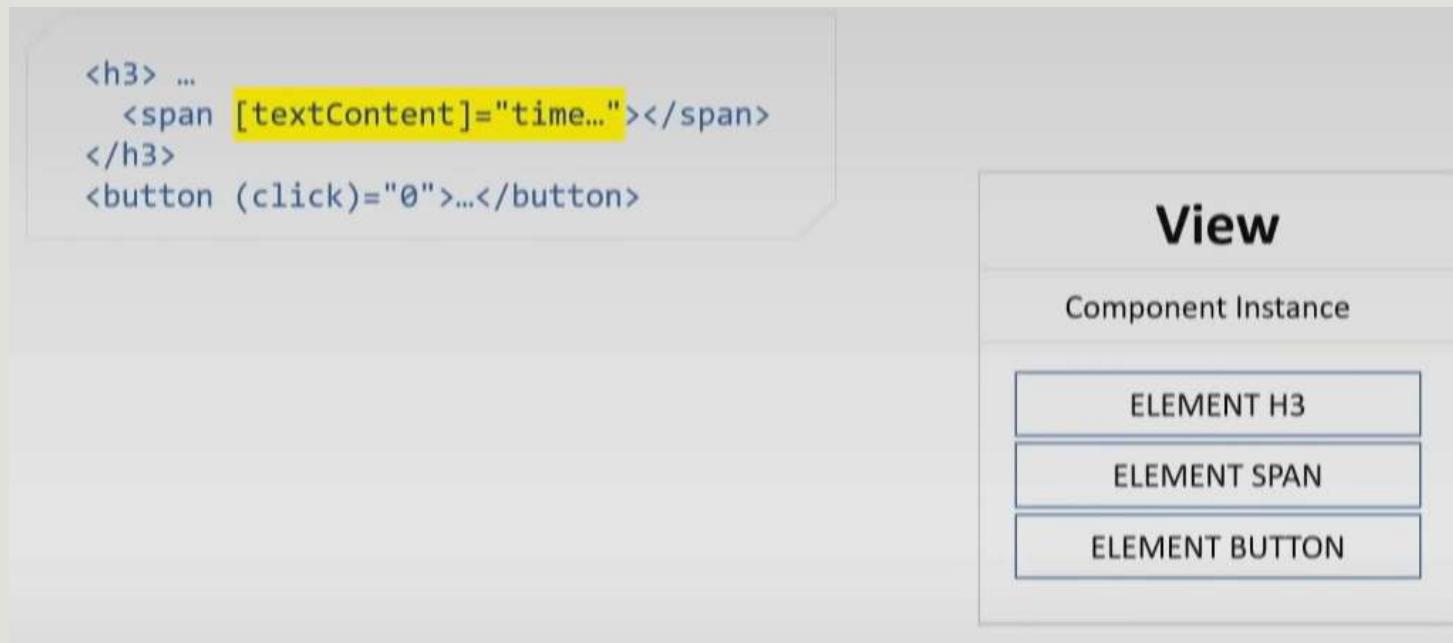
Change Detection

➤ 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.



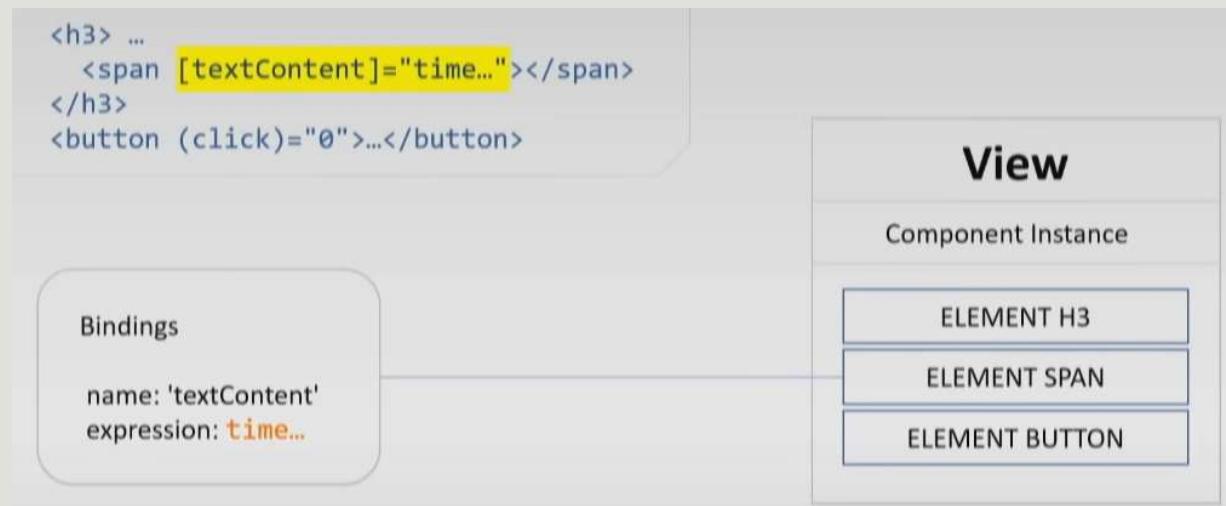
Change Detection

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



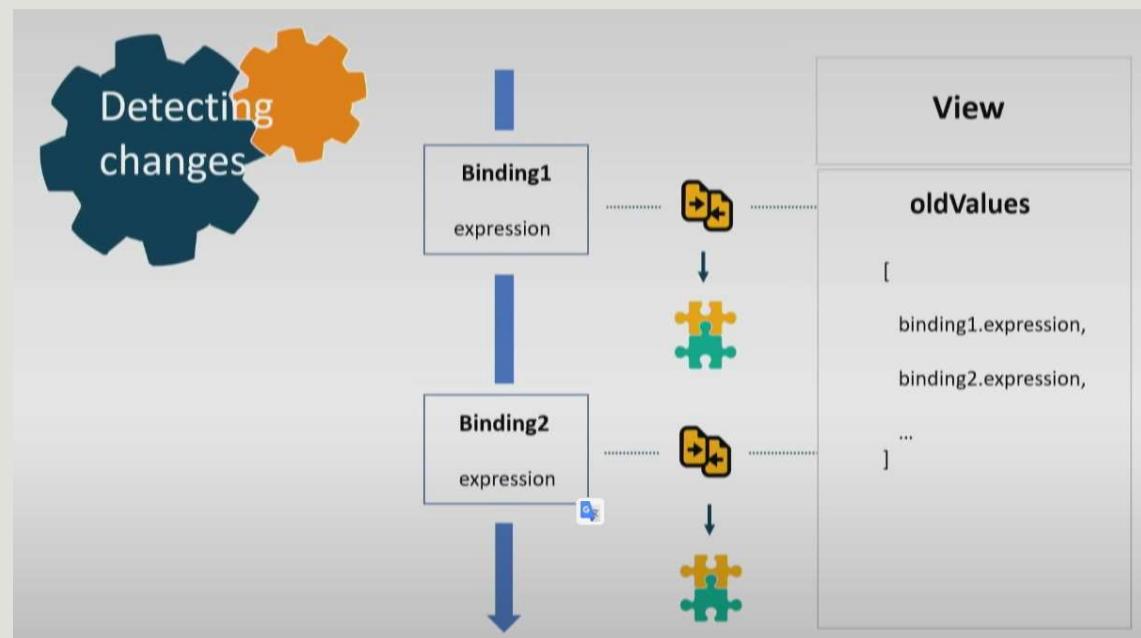
Change Detection

- 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

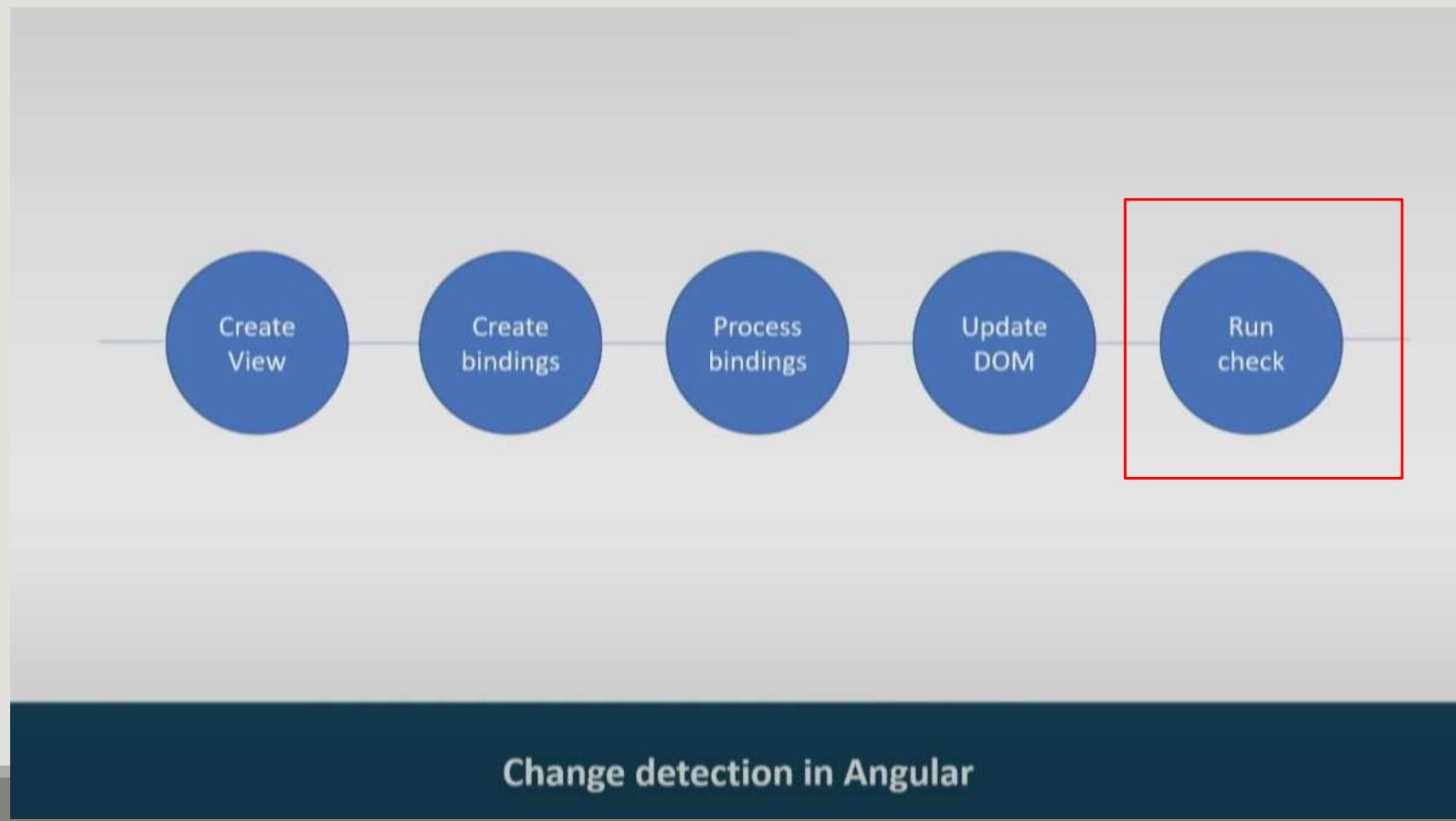


Change Detection

- 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 susceptible 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

https://www.youtube.com/watch?v=wlqAK2hMnKM&list=PLLf-VBOaoVkJE9fTBZdq1CfnRa2TtL_Kb&index=12&ab_channel=StepanSuvorov

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);
```

https://www.youtube.com/watch?v=wlqAK2hMnKM&list=PLLf-VBOaoVkJ9fTBZdq1CfnRa2TtL_Kb&index=12&ab_channel=StepanSuvorov

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 déclencheur** 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
```

https://www.youtube.com/watch?v=wlqAK2hMnKM&list=PLLf-VBOaoVkJE9fTBZdq1CfnRa2TtL_Kb&index=12&ab_channel=StepanSuvorov

Change Detection

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

Change Detection

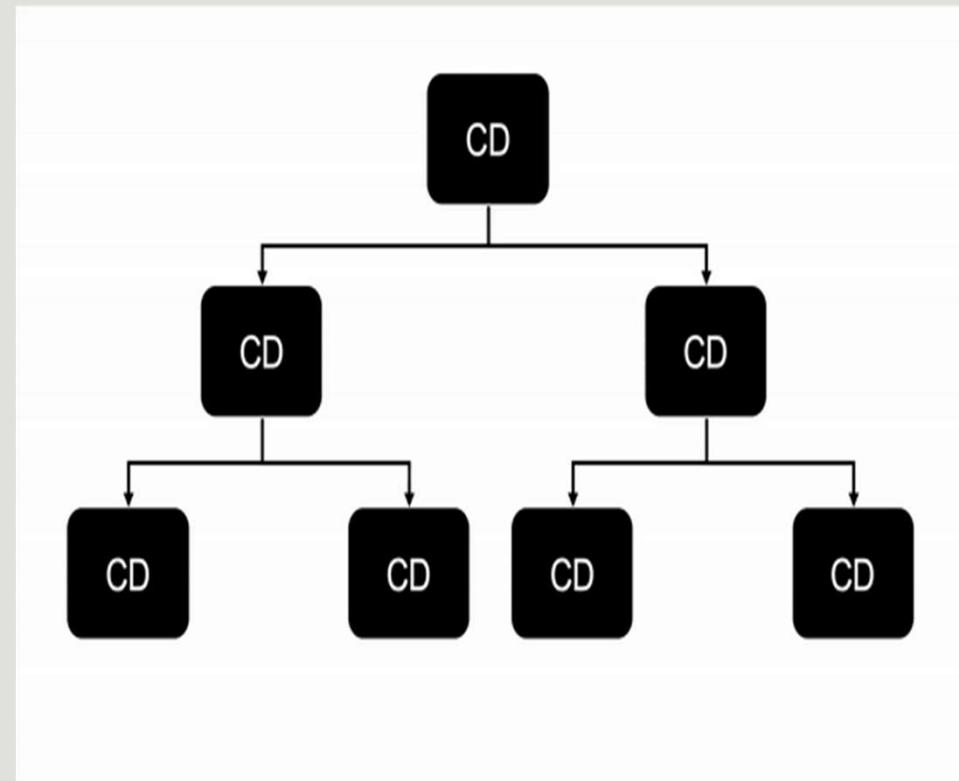
La stratégie par défaut

- 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

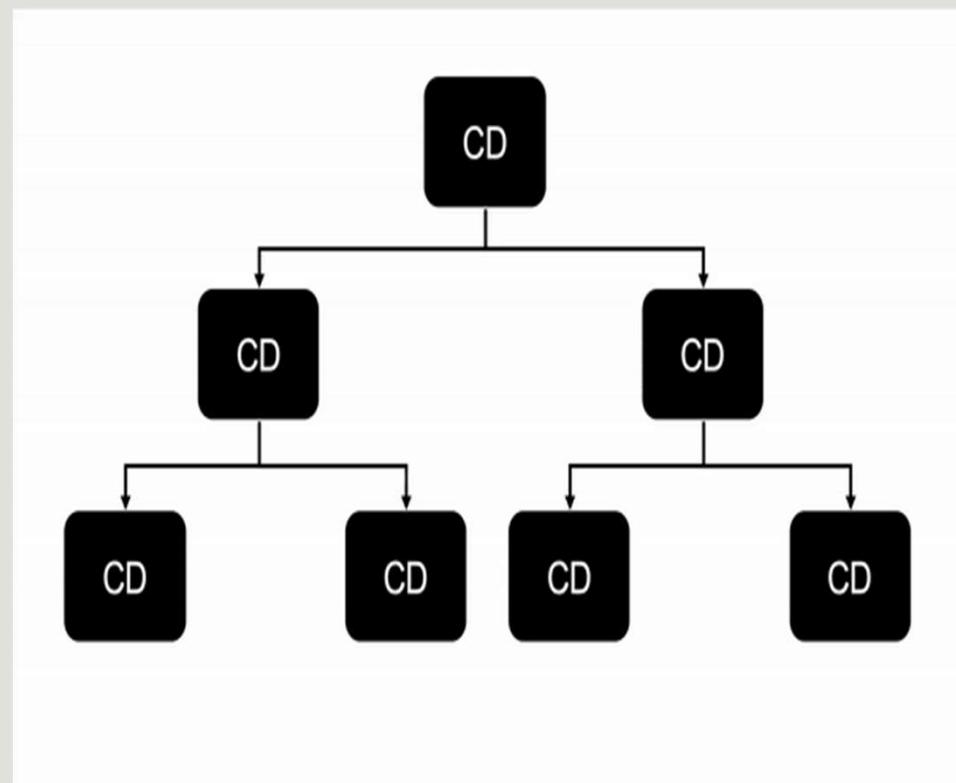
- Ici, dans **tous les composants** de l'arbre de composant, le **change detector alloué à chaque composant**, compare la valeur courante et la valeur précédente des propriétés.
- Si la **valeur change**, il va marquer une propriété **isChanged à true**.



Change Detection

La stratégie par défaut

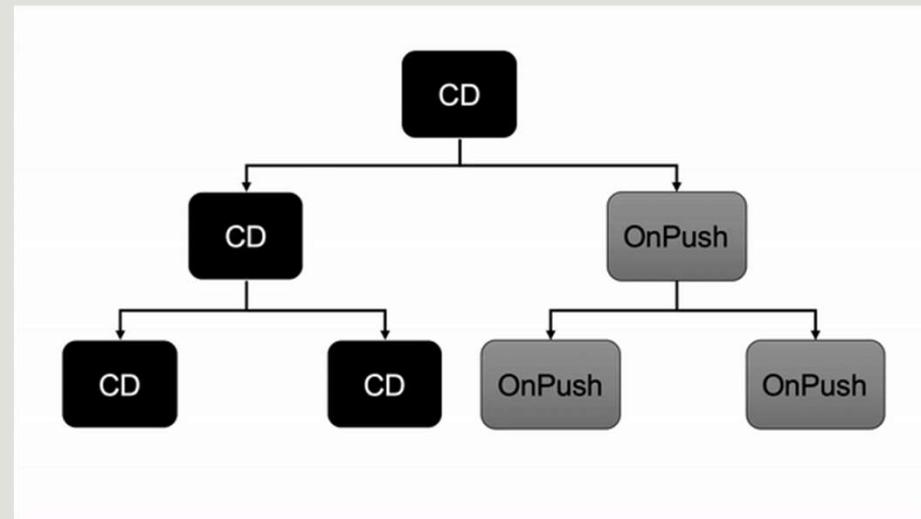
- Cette première stratégie est assez **performante pour les application petite et moyenne.**
- Ceci est du au fait qu'Angular est **très rapide pour le Change détection** pour chaque composant en utilisant la technique du **inline-caching**.
- Cependant, ceci peut ne plus suffire pour les application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

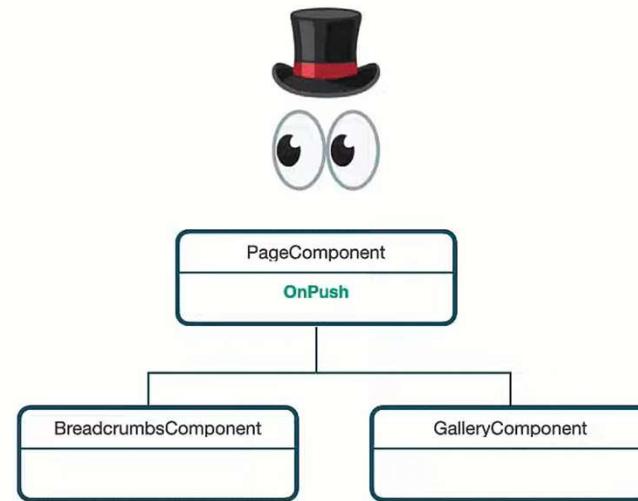


Change Detection

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  
})
```

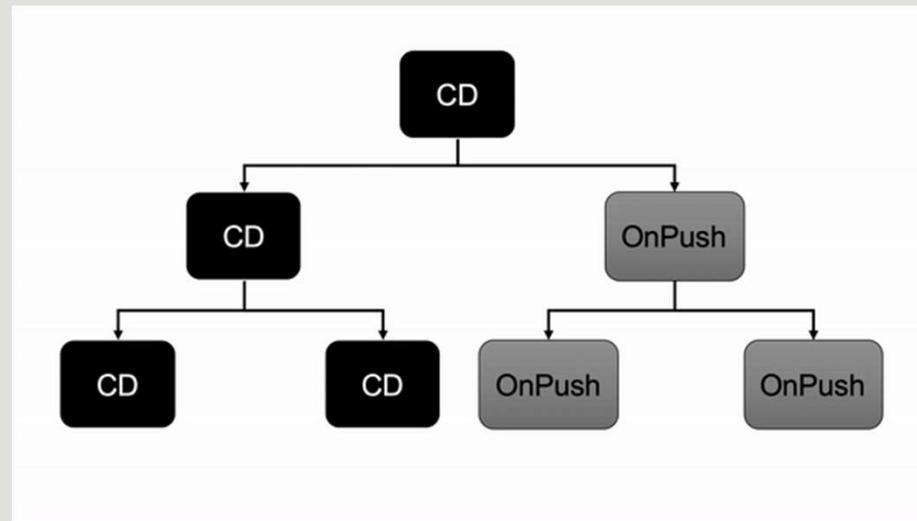


https://www.youtube.com/watch?v=WAu7omIoerM&ab_channel=DecodedFrontend

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 **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.**



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)

asyncPipe

- **async** Pipe est un pipe qui permet d'afficher directement les valeurs émises par 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.

asyncPipe

L'opérateur as

- Si vous utilisez le même observable plusieurs fois dans votre template vous serez amené à réutiliser `async` plusieurs fois.
- Afin d'éviter ça, vous pouvez utiliser l'opérateur **as** qui vous permettra de récupérer le résultat émis par votre source observable dans une variable que vous pourrez utiliser plusieurs fois.

```
<ng-container *ngIf="data$ | async as data">  
  
  </ng-container>
```

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).

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 ou vous avez des évènement 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.

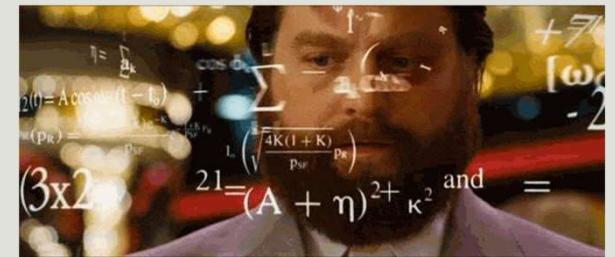
npm i memo-decorator

Pipe pure et impure

- Par défaut un pipe t considéré comme une fonction pure
- Une fonction est dite pure si elle :
 - Ne provoque pas de side effect.
 - Renvoie la même valeur pour les mêmes paramètres.
- Lorsque le pipe est pur, la méthode transform() est invoquée uniquement lorsque ses arguments d'entrée changent.

```
@Pipe({  
    name: 'myPipe',  
    pure: true  
})
```

Exercice



- Créer un composant contenant un input de type texte et une ol
- A chaque fois que vous écrivez dans l'input, afficher le contenu dans un paragraphe (**utilisez ngModel**).
- Dans le composant initialiser un tableau avec **100 valeurs entre 20 et 30**.
- Dans le ol afficher la valeur de chacun des éléments du tableau et en façade la valeur de cet élément lorsque on lui applique la fonction suivante:
 - $f(x) = 2f(x-1) + 3f(x-2)$ si $n > 1$ et $f(n) = 1$ si $n == 0$ ou 1
- Tester l'input. Que remarquez vous lorsque vous écrivez dans l'input ?

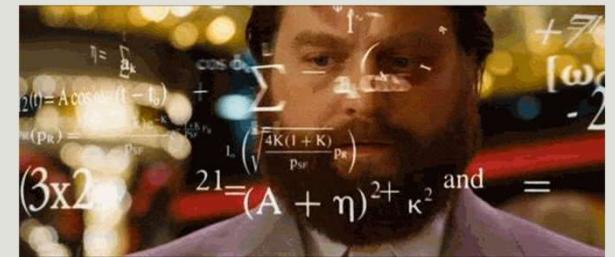
Memo

- Il existe une bibliothèque **memo-decorator** qui permet de mémoriser des fonctions pures.
- Elle permet en l'utilisant de cacher les résultats des fonctions pures et de les réutiliser.
- Pour l'installer utiliser la commande **npm i memo-decorator**

```
import memo from 'memo-decorator';
@memo()
pureFonction(n: number): number { //...}
```

Exercice

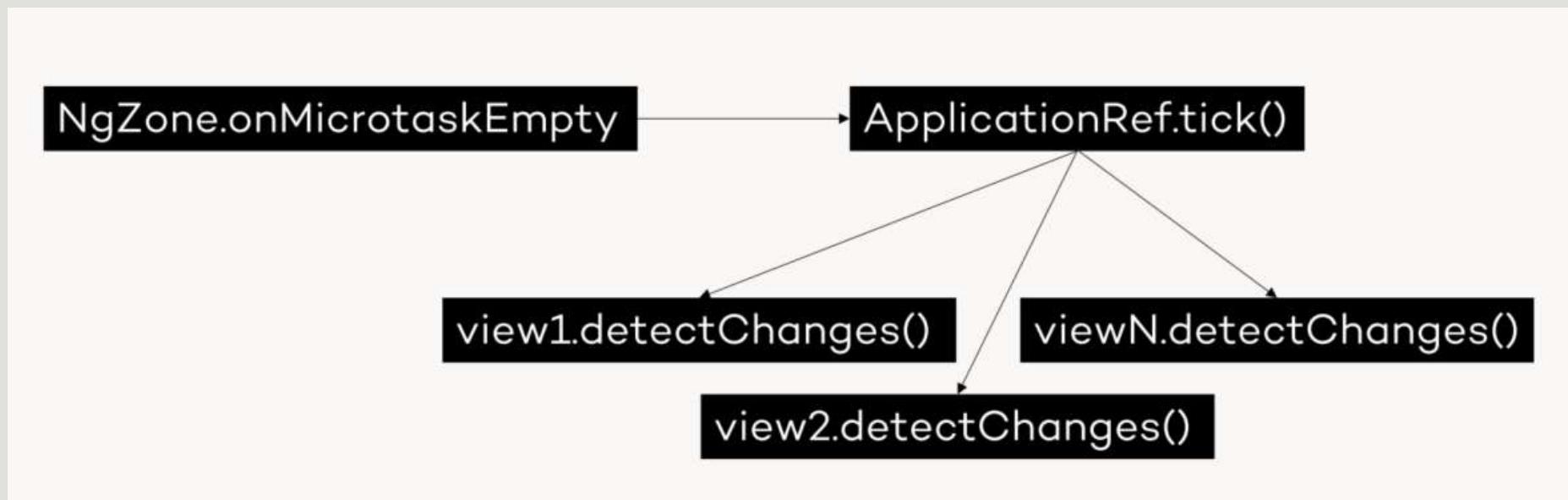
- Utilisez les pipes et utiliser la bibliothèque memo pour optimiser votre pipe



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

- Dans **certains cas d'utilisation**, ce mode de fonctionnement peut causer des problèmes et **ralentir l'application**.
- Prenons le cas des **événements fréquents** qu'on combinera avec des Change Detection ayant beaucoup de calcul :
 - mousemove,
 - des scroll,
 - un setInterval avec des interval courts.

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 détection 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();  
}
```

Change Detection

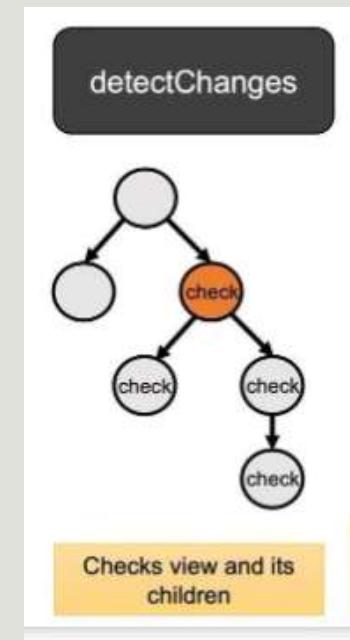
Le changement déclenché manuellement

- Il existe trois méthodes pour déclencher manuellement les detections 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

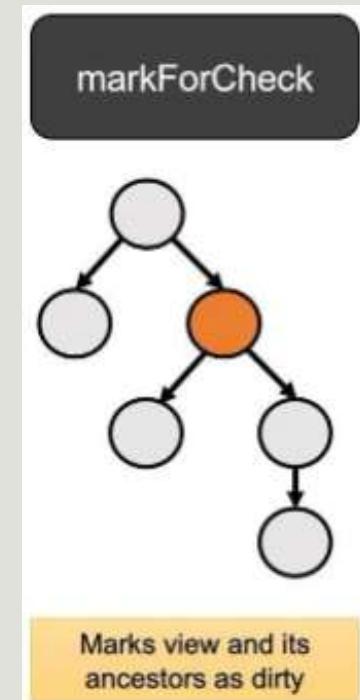
- `detectChanges()` du **ChangeDetectorRef** qui **exécute la détection de changement** sur **ce composant et ses enfants** en gardant à l'esprit la stratégie de détection de changement.
- Il peut être utilisé en combinaison avec `detach()` pour implémenter des contrôles de détection de changement locaux.



Change Detection

Le changement déclenché manuellement

➤ `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**.

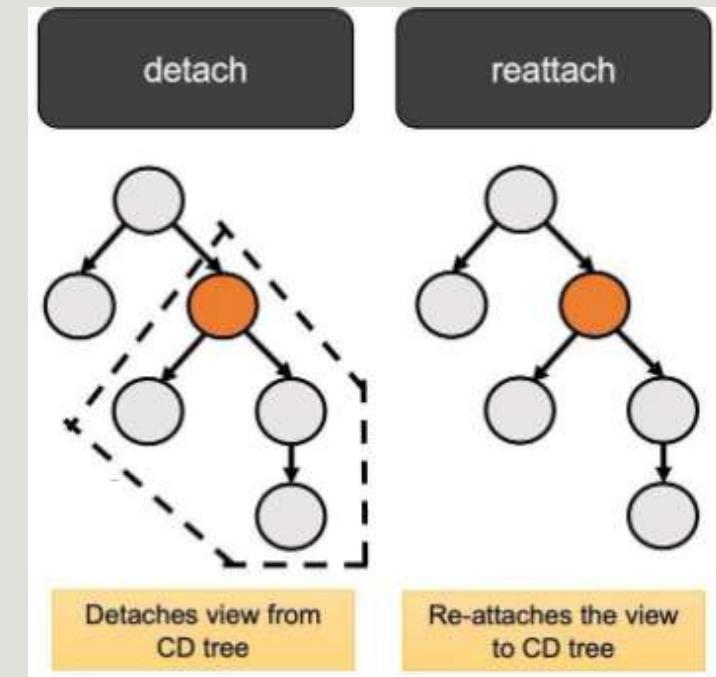


Change Detection

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();  
}
```



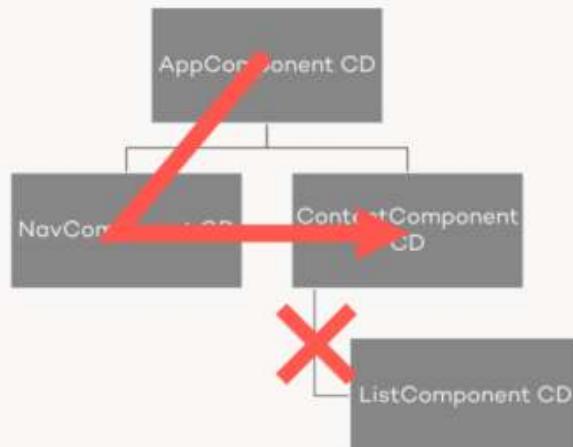
Change Detection

Détacher un composant du change detection

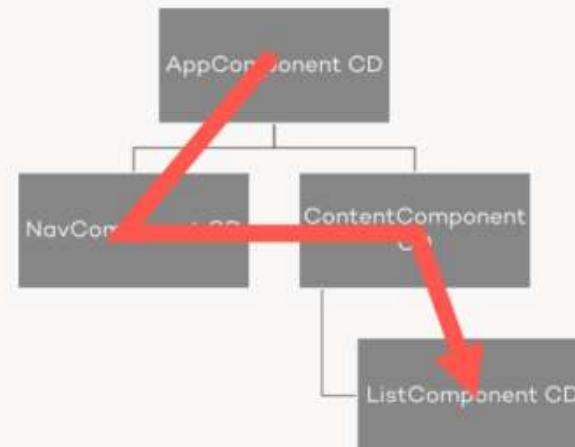
Change Detector

Detaching Components

```
changeDetector.detach();
```

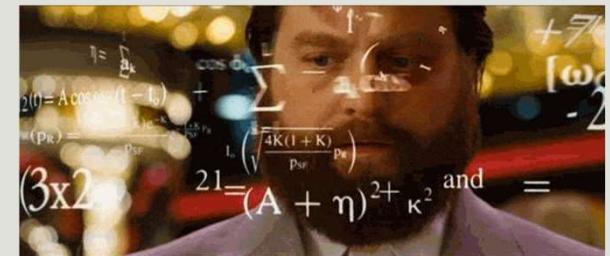


```
changeDetector.reattach();
```



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.

Les Signaux



AYMEN SELLAOUTI





Qu'est ce qu'un Signal ?

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



Qu'est ce qu'un Signal ?

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



Pourquoi intégrer les signaux



Reactivity Everywhere

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



Precision Updates

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



Lightweight Dependencies

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

Change Detection

- Quand un événement se déclenche dans votre application, Angular **parcourt tout votre arbre de composants pour chercher où les modifications doivent être effectués.**
- 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

API

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

Créer un signal via l'api signal()

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

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

Récupérer la valeur d'un signal

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

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

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

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

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

computed()

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

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

computed()

-
- ▶ Pour identifier un signal qui a changé, et donc si on doit exécuté 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 traitement conditionnel.

```
@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 dépendence

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

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

computed()

Modifier un signal dans un computed

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

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

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

computed()

Les computed, autres propriétés

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

Writablesignals et signals

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

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

Signals et service

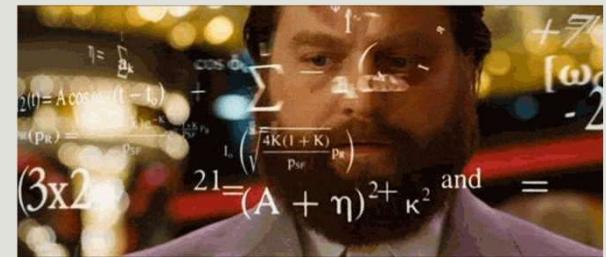
- Les signaux facilitent la réactivité dans Angular
- Nous pouvons voir un signal comme une source de vérité unique
- Il est donc logique de le partager et de permettre à toute personne intéressé d'y accéder.
- Cependant, pensez à les encapsuler pour garantir le control sur la modification.

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

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

Exercice

- Reproduisez cette interface en utilisant les signaux.
- Un Todo possède un id, name, content et un statut.
- Le statut est une énumération
- A chaque changement de statut, l' élément va automatiquement changer de liste



```
export enum TodoStatusEnum {  
  WAITING = 'waiting',  
  IN_PROGRESS = 'in progress',  
  DONE = 'done',  
}
```

A screenshot of a React application interface. At the top, there is a header with input fields for 'name' (containing 'I') and 'content', and a red 'add' button. Below the header are three tabs: 'Waiting' (grey), 'In Progress Todos' (yellow, active), and 'Done Todos' (light green). The 'In Progress Todos' tab contains a single item: 'I'. The background is light grey.

L'API effect()

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

L'API effect()

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

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

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

L'API effect()

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

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

L'API effect()

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

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

effect()

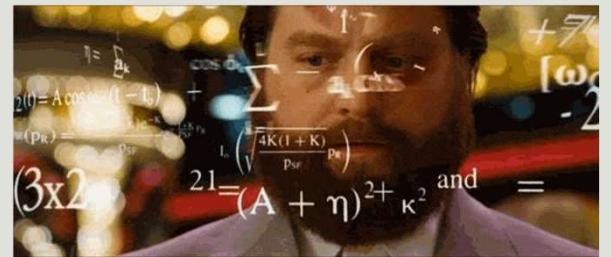
Exclure un signal de l'arbre de dépendance

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

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

Exercice

- Utiliser les signaux à la place du subject pour la gestion de la sélection d'un Cv

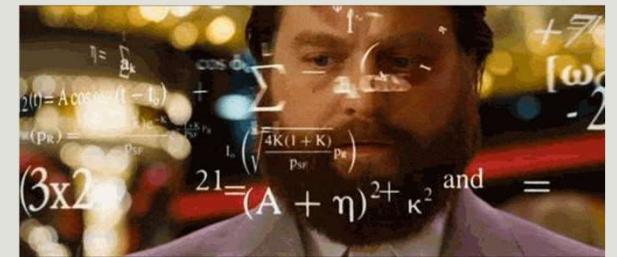


signaux et rxJs

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**

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.
- Créer un autre composant qui contient un champ de texte de type number.
- Ce composant doit appeler le composant isEven et lui passer en paramètre la valeur de l'input.



Signal Input

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

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

Signal Input

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

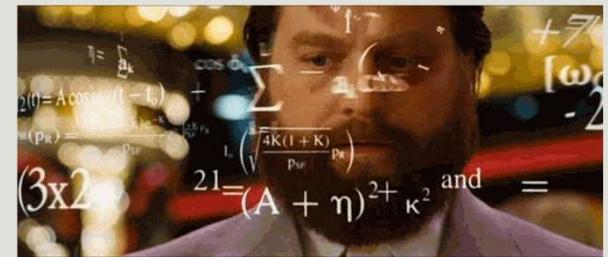
```
// optional  
counter = input<number>();  
// Required  
counter = input.required<number>();  
// Valeur par défaut = 0  
counter = input.required<number>(0);
```

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

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

Signal Input

- Reprenez cet exemple en utilisant les Signal Input

0 is even

Les directives structurelles

- ▶ Comme déjà dit, les directives structurelles d'Angular sont les directives qui manipulent les éléments du DOM, comme **@if**, **@for** et **@switch**.
- ▶ Avant la version 17, ces directives avaient une autre syntaxe avec ***ngIf**, ***ngFor** et **[ngSwitch]**.
- ▶ Elles sont appliquées sur **l'élément HOST** (l'élément sur lequel la directive est appliquée).

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.

```
<button
  *ngIf="authService.isAuthenticated()"
  (click)="deleteCv(cv)"
  class="btn btn-danger">
  Delete
</button>
```

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

Control flow

@if

- ▶ Si vous utilisez l'ancienne version `*ngIf` avec des standalone component, vous êtes dans l'obligation d'importer la directive.
- ▶ `@If` est nativement reconnu par le compilateur, vous n'avez pas besoin de l'importer.

```
@Component({
  standalone: true,
  template: `<div
    *ngIf="condition">Content</div>`,
  imports: [NgIf],
})
export class MyComponent {}
```

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.**

```
<div *ngIf="connectedUser; else disconnectedMessage">
  Hello {{ connectedUser.name}}
</div>

<ng-template #disconnectedMessage>
  <div>Merci de vous connectez</div>
</ng-template>
```

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

<ul>
  @for (episode of episodes; track episode.id) {
    <li
      [ngClass]={`${ odd: $odd, bgfonce: $first }`}
    >
      Episode {{ $index + 1 }} : {{ episode.title }}
    </li>
  }
</ul>
```

Control flow

@For track

5
8
4

- ▶ 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>  
}  
</ul>
```

Control flow

@For

track, pourquoi c'est devenu obligatoire ?

- ▶ Dans son introduction de `@for`, Minko Gechev un des chefs de projets Angular a argumenté le choix de rendre `@For` Obligatoire

We often see performance problems in apps due to the lack of `trackBy` function in `*ngFor`. A few differences in `@for` are that `track` is mandatory to ensure fast diffing performance. In addition, it's way easier to use since it's just an expression rather than a method in the component's class.

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 bloc 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

```
<div [ngSwitch]="streamingService">
  <div *ngSwitchCase="'AppleTV'">Ted Lasso</div>
  <div *ngSwitchCase="'Disney+'>Mandalorian</div>
  <div *ngSwitchDefault>Peaky Blinders</div>
</div>
@switch(streamingService) {
  @case ('Disney+') {
    <div>'Mandalorian'</div>
  } @case ('AppleTV') {
    <div>'Ted Lasso'</div>
  } @default {
    <div>'Peaky Blinders'</div>
  }
}
```

```
@Component({
  standalone: true,
  imports: [NgSwitch, NgSwitchCase, NgSwitchDefault],
})
export class SwitchComponent {
  streamingService = 'Netflix';
}
```

Control flow

Migration automatique

- ▶ Si vous voulez migrer votre ancien code et utilisez le nouveau work flow, angular vous fournit une commande qui le fait pour vous :

`ng g @angular/core:control-flow`

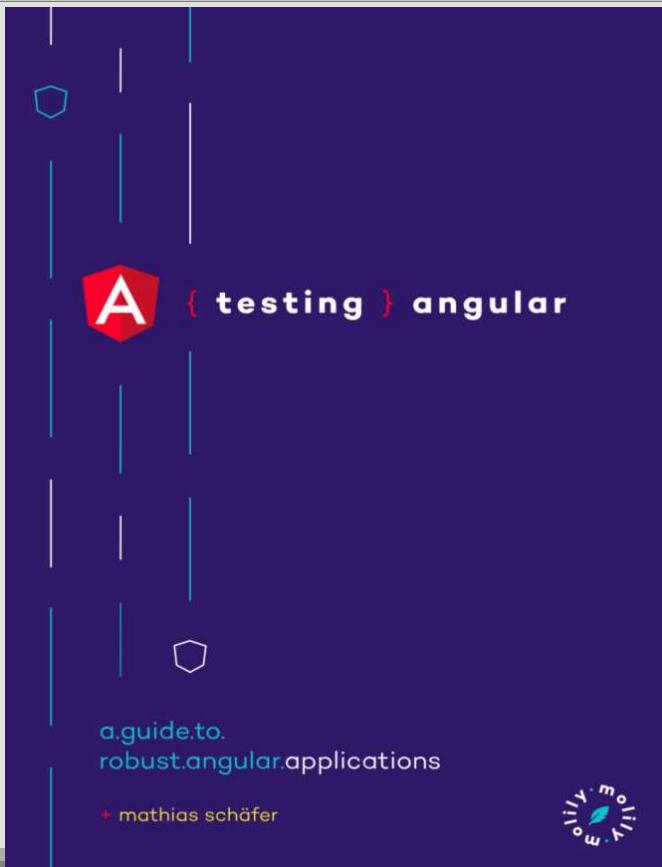
Cette fonctionnalité est encore en developer Preview (Angular 17.2)

Angular Tests Unitaires et E2E

AYMEN SELLAOUTI



Références



Tests unitaires : Introduction

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

Tests unitaires : Pourquoi

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

Tests unitaires : Partout ?

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

Angular et les tests unitaires

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

Jasmin et Karma

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



Jasmine

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



Lancement d'un test

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

`ng test`

Concepts de base de jasmin describe (suite)

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

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

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

Concepts de base de jasmin

Specification (**it**)

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

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

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

Concepts de base de jasmin

Specification (**it**)

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

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

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

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

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

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

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

Concepts de base de jasmin

Attente (Expectation)

- Dans la phase **d'affirmation (Assert)**, le test **compare** la **sortie** ou la **valeur de retour réelle** à la **sortie ou à la valeur de retour attendue**. S'ils sont **identiques**, le test **réussit**. S'ils **diffèrent**, le test **échoue**.
- Afin de gérer 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**.

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

Jasmin matchers

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

Jasmin matchers

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

Concepts de base de jasmin

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

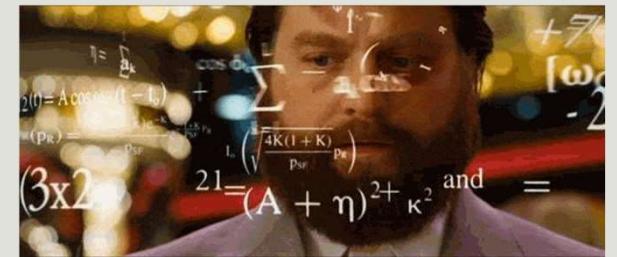
Exemple **expect(etatActuel).toBe(etatExpecté)**

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

Concepts de base de jasmin

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

Exercice



- Récupérer le Répo suivant :

<https://github.com/aymensellaouti/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**.

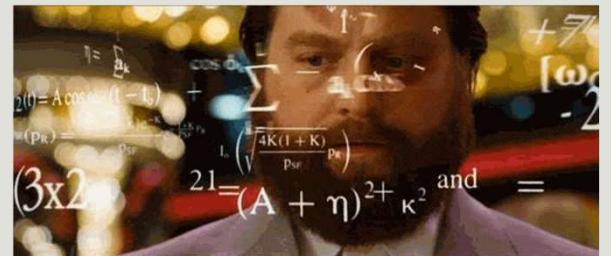
Concepts de base de jasmin

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**.
- **afterEach** : prend en paramètre une **callback** et la **répète** après chaque suite **describe**.

Exercice

- Mettez à jour vos tests.



Tests E2E

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

Tests E2E

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

Tests E2E

Comment ça marche ?

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

Tests E2E

Cypress

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

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

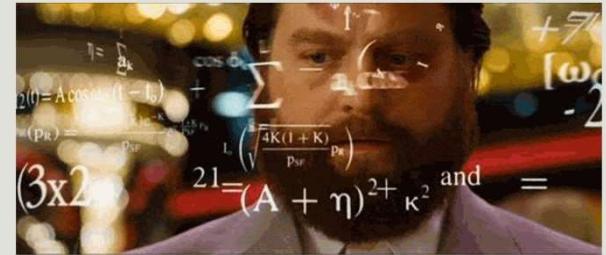
Tests E2E

Cypress

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

Exercice

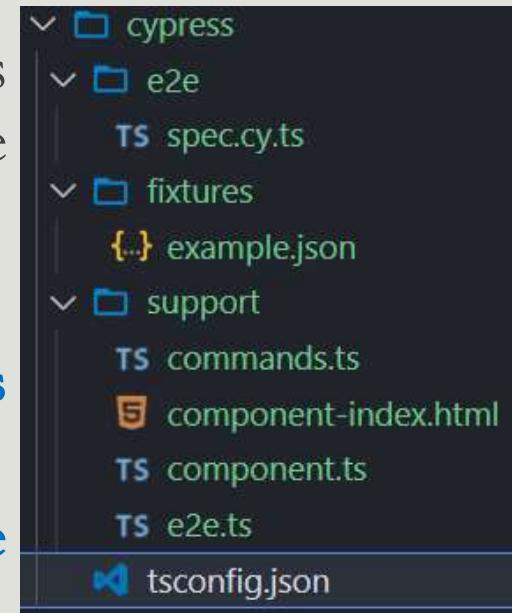
- Installez Cypress et regarder les différents fichiers ajoutés


$$I \approx \sum_{k=1}^{\infty} I_k = A \cos(\omega t - I_0) + \sum_{k=1}^{\infty} C_k e^{-\frac{t}{T_k}} \cos\left(\frac{2\pi k}{T_k} t\right)$$
$$(P_R) = \frac{(A + \eta)^2 + \kappa^2}{P_{RF}} = \frac{1}{L_c} \left(\sqrt{\frac{4K(I+K)}{P_{RF}}} P_R \right)$$
$$(3x2)$$
$$2I = (A + \eta)^2 + \kappa^2 \text{ and } =$$

Tests E2E

Le dossier cypress

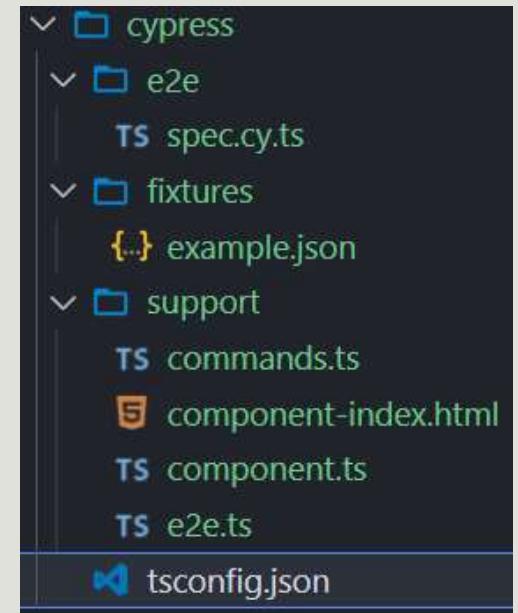
- Le dossier **cypress** généré contient :
 - Une configuration **tsconfig.json** pour tous les fichiers TypeScript spécifiquement dans ce répertoire,
 - Un répertoire **e2e** pour les **tests E2E**,
 - Un répertoire de **support** pour **les commandes personnalisées** et autres assistants de test,
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

$$I \approx \sum_k \cos(\omega_k t - I_0) + \dots$$
$$I(t) = A \cos(\omega t - I_0) + \dots$$
$$p_R = \frac{A e^{-K t}}{p_{sp}} e^{\frac{i \omega_k}{p_{sp}} t} L_0 \left(\sqrt{\frac{4K(I+K)}{p_{sp}}} p_R \right)$$
$$(3x2) \quad 2I = (A + \eta)^2 + \kappa^2 \text{ and } =$$

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

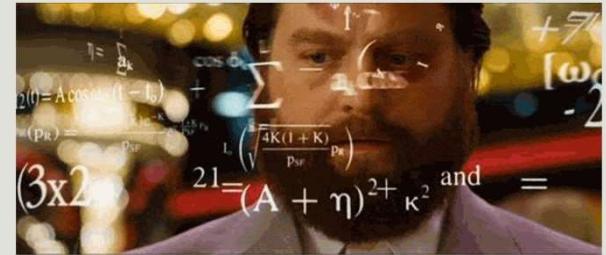
- **cy.root()**: Cette méthode permet de sélectionner la racine du document HTML. **Exemple:** cy.root().should('have.class', 'racine')
- **cy.children()**: Cette méthode permet de sélectionner les enfants d'un élément donné. **Exemple:** cy.get('.élément-parent').children().should('have.length', '3')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

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

Tests E2E

Test des requêtes HTTP API Mockés / intercept, mockez une réponse

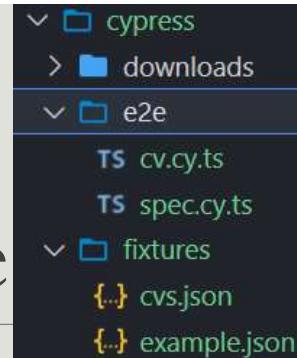
➤ Lorsque vous utilisez intercept suivez les étapes suivantes:

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

Tests E2E

Test des requêtes HTTP

API Mockés / intercept, mockez une réponse



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

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

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

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

Tests E2E

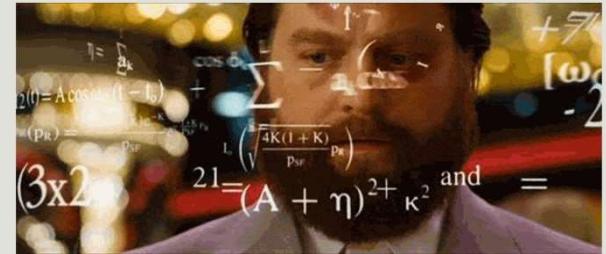
Test des requêtes HTTP

API Mockés / intercept, affecter un alias

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

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

Exercice



- Faites 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

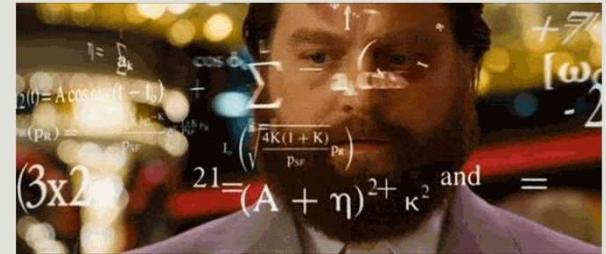
<https://docs.cypress.io/api/commands/should>

Tests E2E

Les assertions

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

Exercice



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

Tests E2E

Location

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

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

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

Tests E2E

Déclencher des actions

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

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

Tests E2E

Déclencher des actions

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

Tests E2E

Déclencher des actions

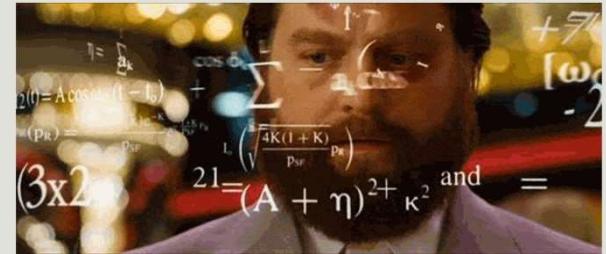
- Afin de simuler le click sur un caractère spécial, entre, insert, delete, utilisez la syntaxe suivante:

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

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

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

Exercice



- Ecrivez un test qui permet de tester le fonctionnement de la sélection d'un cv via le click sur le bouton détails.
- Utilisez des fixtures
- Simulez un click sur le bouton détails du premier élément.
- Vérifiez qu'en cliquant sur le bouton, vous accéder à la bonne url
- Vérifier que les infos du cv sont bien affichées

aymen.sellaouti@gmail.com