



ANGULAR 4

Jérémy PERROUAULT



ANGULAR 4

Les fondamentaux

PRÉPARER SON ENVIRONNEMENT

Utilisation d'un logiciel de traitement de texte type Sublime, Atom, Notepad++, Eclipse ou autre

Prise en main d'un terminal

Prise en main de navigateurs et de leur console de debug

Préparer le répertoire Projet

Installer Node.JS & NPM

HTML5 / CSS3 / JAVASCRIPT

Les ingrédients de la recette

- « Transformer un site internet en application Web »

| | |
|------------|--------------------------|
| HTML5 | Structure du document |
| CSS3 | Présentation du document |
| JavaScript | Manipulation du document |

Mono-page ?

- Un seul fichier d'accès, plusieurs routes

ANGULAR

Description HTML5 / CSS3

Langages JavaScript et TypeScript

Framework créé par Google

- AngularJS est la première version

Patterns

- MVC (Model-View-Controller)
- MVVM (Model-View-ViewModel) pour le binding
- IoC (Inversion of Control) pour l'injection de dépendances

ANGULAR

AngularJS a été écrit et pensé pour JavaScript

- AngularJS désigne en réalité la version 1.x de Angular

Angular a été écrit et pensé pour TypeScript

- Angular désigne les versions 2.x et supérieures

Il est possible de développer des applications Angular avec

- TypeScript
- JavaScript
- Dart
- Le langage recommandé étant TypeScript

Toute la document est accessible sur <https://angular.io/docs>



TYPESCRIPT

Introduction à TypeScript

GESTIONNAIRE DE PAQUETS

NPM est un gestionnaire de paquets

Commandes utiles

- `$ npm install -g « package »`
 - Pour installer un paquet de façon globale sur la machine
- `$ npm install « package »`
 - Pour installer un paquet utilisé dans un projet
- `$ npm init`
 - Initialiser le projet (prépare le fichier « package.json » dans le répertoire projet)
- `$ npm install`
 - Pour télécharger et installer les dépendances d'un projet
 - S'appuie sur le fichier « package.json »

TYPESCRIPT

Bienvenue dans TypeScript !

- Du JavaScript mais ... Typé
- Qui ne peut pas être interprété par les navigateurs
 - Il est donc compilé en JavaScript

L'extension des fichiers TypeScript est *.ts

Il ajoute de nouvelles fonctionnalités à JS

- Les annotations (appelés décorateurs)
- Le typage de variables (non obligatoire)
- Les classes
- La signature des méthodes
- La généricité

TYPESCRIPT

Déclaration de variables

```
let maVar: number;  
let maVarAssigned: string = "Jérémy";  
let myClient: IClient = new Client();  
let myPersonnes: Array<Personne> = new Array<Personne>();  
let myObject: any;
```

- | | |
|-------------------|---|
| ▪ number | Entier ou flottant |
| ▪ string | Chaîne de caractères |
| ▪ IClient | Objet de type IClient |
| ▪ Array<Personne> | Tableau de personnes |
| ▪ any | Objet dont on ignore le type concret (Object) |

TYPESCRIPT

Déclaration d'une classe

```
class Personne {  
  private nom: string;  
  public prenom: string = "Jérémy";  
  protected age: number = 20;  
  
  constructor(nom: string, prenom?: string) {  
    this.nom = nom;  
    this.prenom = prenom;  
  }  
  
  public getNom(): string {  
    return this.prenom + " " + this.nom;  
  }  
}
```

TYPESCRIPT

Déclaration d'une classe

```
class Personne {  
    protected age: number = 20;  
  
    constructor(private nom: string, private prenom?: string) { }  
  
    public getNom(): string {  
        return this.prenom + " " + this.nom;  
    }  
}
```

TYPESCRIPT

Déclaration d'une classe avec héritage

```
class Client extends Personne {  
  private ca: number = 963258741;  
}
```

TYPESCRIPT

Déclaration et implémentation d'une interface

```
interface IClient {  
    getCa(): number;  
}
```

```
class Client extends Personne implements IClient {  
    private ca: number = 963258741;  
  
    public getCa(): number {  
        return this.ca;  
    }  
}
```

TYPESCRIPT

Déclaration et utilisation de la généricité

```
interface IClient<T> {  
    getCa(): T;  
}
```

```
class Client extends Personne implements IClient<number> {  
    private ca: number = 963258741;  
  
    public getCa(): number {  
        return this.ca;  
    }  
}
```

```
let myClient: IClient<number> = new Client();
```

TYPESCRIPT

Installation du compilateur TypeScript

- `$ npm install -g typescript`

Vérifier l'installation

- `$ tsc -v`

Compilation

- `$ tsc fichier.ts`

EXERCICE

Démonstration

Installer NPM (NodeJS)

Créer un fichier TypeScript avec une classe

- **Personne**
 - Qui a un nom, un prénom
- **Client qui hérite de Personne**
 - Qui a un CA et une liste de produits
- **Fournisseur qui hérite de Personne**
 - Qui a un nom de société
- **Produit**
 - Qui a un nom, un prix, un fournisseur
 - Qui a une liste de clients

Compiler en JS



PRÉSENTATION DE ANGULAR 4

Développer avec Angular 4

PRÉSENTATION D'ANGULAR

Modèles

- TypeScript

Vues

- HTML5 / CSS3

Contrôleur

- TypeScript

PRÉSENTATION D'ANGULAR

Quelques notions d'Angular

- Import
 - Il faudra importer les éléments dont vous aurez besoin, un peu comme en Java
- Module
 - Conteneur de directives, de composants, de services, ...
- Directive
 - C'est une classe
 - C'est un Component sans vue
- Component
 - Hérite de Directive
 - Composant Angular
 - Composé de vue (template) HTML, d'une classe et éventuellement d'un style CSS

INJECTION DE DÉPENDANCES

Implémentation du Design Pattern IoC

- Un composant n'est plus responsable de sa dépendance
 - Le composant déclare sa dépendance
 - Le composant n'instancie pas sa dépendance

Résolution d'une dépendance

- Basé sur le nom strict de la dépendance (sensible à la casse)

NOMENCLATURES

Règles de nommage

| | |
|-------------------------|----------------------------|
| ▪ AppModule | app.module.ts |
| ▪ AppComponent | app.component.ts |
| ▪ MonComposantComponent | mon-composant.component.ts |
| ▪ ProduitModule | produit.module.ts |
| ▪ AscBoldComponent | asc-bold.component.ts |
| ▪ ProduitDirective | produit.directive.ts |

ARCHITECTURE

L'architecture sera la suivante

- Racine projet
 - .gitignore
 - index.html
 - app (ou src/app)
 - app.component.html
 - app.component.ts
 - app.module.ts
 - main.ts
 - assets (ou src/assets)
 - package.json
 - systemjs.config.js
 - tsconfig.json

DÉMARRAGE

Notre application Angular a besoin d'un « exécuteur » pour démarrer

- On va utiliser Browser, puisque notre application s'exécutera dans un navigateur

BrowserModule (platform-browser)

- Contient le code partagé pour l'exécution au sein d'un navigateur (thread DOM entre autre)

platformBrowserDynamic (platform-browser-dynamic)

- Contient le code côté client qui permet
 - De générer et d'intégrer les templates HTML, avec le binding MVVM, les composants, les directives, ...
 - De gérer l'injection de dépendances (IoC)

DÉMARRAGE

Point d'entrée de l'application

- Fichier *app/main.ts* (qui sera traduit en fichier JS)

Single Page Application (SPA)

- *index.html*

INITIALISATION

Il existe des outils permettant d'initialiser un projet Angular rapidement

- Angular CLI par exemple

Mais en partant de rien

- 4 fichiers sont nécessaires à la racine du projet



NOUVELLE APPLI — SANS OUTILS

Initialiser une application sans
outil

NOUVELLE APPLI — SANS OUTILS

4 fichiers sont nécessaires à la racine du projet

- `index.html`
 - Fichier principal de l'application Angular
- `package.json`
 - Ce fichier décrit le projet et ses dépendances (pour NPM)
- `systemjs.config.js`
 - Angular a choisi le système de configuration SystemJS par défaut
 - Permet de charger les modules Angular, et d'assembler l'application de manière cohérente
- `tsconfig.json`
 - Ce fichier décrit la configuration de la compilation de l'application Angular en JS
 - **GIT : vous pourrez ignorer le répertoire « outDir » défini dans ce fichier, si différent du répertoire sources**

NOUVELLE APPLI — SANS OUTILS

Il faudra ensuite définir 4 autres fichiers dans le répertoire « *app* »

- `app.component.html`
 - Fichier du template du composant principal
- `app.component.ts`
 - Classe TS du composant principal
- `app.module.ts`
 - Classe TS du module principal
- `main.ts`
 - Fichier d'entrée pour l'application Angular (un fichier TS)

NOUVELLE APPLI — SANS OUTILS

Installation des dépendances (définies dans *package.json*)

- Se placer dans le répertoire du projet
- \$ npm install
- A l'issu de cette commande, un répertoire *node_modules* est créé à la racine
 - Il contient les dépendances JS
 - GIT : ignorer ce répertoire !

NOUVELLE APPLI — SANS OUTILS

On a besoin au minimum d'un composant

- Le composant racine, qui sera exécuté par le module racine
- Par convention, ce composant s'appelle AppComponent
- Fichier *app.component.ts*

On a besoin au minimum d'un module

- Le module racine
- Par convention, ce module s'appelle AppModule
- Fichier *app.module.ts*

NOUVELLE APPLI — SANS OUTILS

Démarrage de l'application

- `$ npm start`
 - Compilation des fichiers TS en fichiers JS
 - Le navigateur s'ouvre tout seul
 - Cette commande écoute la modification des fichiers
 - Lorsque les fichiers sont modifiés, ils seront recompilés, et la page du navigateur sera rafraichie automatiquement
 - Chargement des fichiers
 - `index.html > main.ts > app.module.ts > app.component.ts`



NOUVELLE APPLI — ANGULAR CLI

Initialiser une application avec
Angular CLI

NOUVELLE APPLI — ANGULAR CLI

Installation de Angular CLI

- `$ npm install -g @angular/cli@latest`

Création d'un nouveau projet

- `$ ng new nom_projet`

Démarrage du projet

- `$ ng serve`
 - Compilation des fichiers TS en fichiers JS
 - Cette commande écoute la modification des fichiers
 - Lorsque les fichiers sont modifiés, ils seront recompilés, et la page du navigateur sera rafraichie automatiquement
 - Chargement des fichiers
 - `index.html > main.ts > app.module.ts > app.component.ts`

NOUVELLE APPLI — ANGULAR CLI

Angular CLI permet d'initialiser rapidement une nouvelle application

Mais permet aussi de générer des nouveaux composants, services, ...

Toute la documentation : <https://github.com/angular/angular-cli/wiki>



LES PREMIÈRES NOTIONS

Les fondamentaux

LES PREMIÈRES NOTIONS

Lorsque vous utilisez un module, un composant, une directive, un service, ...

- Dans le script TS, il faut déclarer l'import de ce dont vous avez besoin (un peu comme en Java)

```
import { quelquechose } from 'quelquepart';  
import { unechose, autrechose } from 'ailleurs';  
import { MonModule } from './monfichiersansextension';
```

LES PREMIÈRES NOTIONS

Contenu du fichier *app.component.ts* (Composant principal)

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

@Component({
  selector: 'eshop-app',
  templateUrl: 'app/app.component.html',
  styleUrls: ['app/app.component.css']
})
export class AppComponent {
  private prenom: string = "Jérémy";
}
```

Sélecteur HTML qu'on utilisera
pour appeler ce composant

Fichier HTML de la vue du
composant

Fichier(s) CSS de la vue du
composant

LES PREMIÈRES NOTIONS

Contenu du fichier *app.module.ts* (Module principal)

```
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 { }
```

Le module a besoin de
BrowserModule

On démarre avec
AppComponent

On utilise le composant
AppComponent

Définition de la classe
principale

LES PREMIÈRES NOTIONS

Dans @NgModule

- import importe les modules requis pour le module en question
- declarations liste des composants, directives utilisés par le module
- bootstrap liste des composants utilisés pour démarrer l'application
 - On peut démarrer l'application de multiples façons ; au sein d'un navigateur, on va utiliser le bootstrapping Browser

LES PREMIÈRES NOTIONS

Sélecteur HTML déclaré dans le composant

Quant au contenu du body du fichier HTML *index.html*

```
<body>
  <eshop-app>Chargement de l'application Angular ...</eshop-app>

  <!-- Scripts Angular ... -->
</body>
```

LES PREMIÈRES NOTIONS

Contenu du fichier *main.ts*

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';  
import { AppModule } from './app.module';  
  
platformBrowserDynamic(  
  .bootstrapModule(AppModule);
```

- Dans ce fichier, on indique qu'on exécute notre module principal, avec platform-browser-dynamic
- C'est pour cette raison que AppModule a besoin d'importer BrowserModule

EXERCICE

Initialiser et démarrer le projet (avec Angular CLI)

Tester et vérifier que le module s'exécute bien

- Le terminal reste en attente d'une modification de fichier
- L'adresse <http://localhost:4200/> est disponible



PREMIERS PAS

Développer avec Angular

PREMIERS PAS — EXPRESSION LANGUAGE

Angular mappe des données au HTML en utilisant des EL

- {{ expression }}

Exemples

```
<p>{{ 5 + 5 }}</p>
```

```
<p>Le client est {{ prenom }} {{ nom }}</p>
```

```
<p>Le client est {{ client.prenom }} {{ client.nom }}</p>
```

EXERCICE

Afficher le prénom dans le paragraphe

BINDING — LIAISON DES DONNÉES

Il est possible de lire une donnée avec les expressions

Il est également possible de les modifier

- Avec un input par exemple, ou un select
 - Utilisation d'une directive *ngModel*
- Puisqu'utilisation d'un input (ou select, ou autre) qui sont habituellement dans un **formulaire HTML**
 - Le module **FormsModule** est nécessaire
 - Importer FormsModule
 - Dans le module AppModule
 - Depuis *@angular/forms*

```
import { FormsModule } from '@angular/forms';
```

EXERCICE

Tester ce code : dans le template HTML

```
<input type="text" [(ngModel)]="prenom" />  
<p>{{ prenom }}</p>
```

Que fait ngModel ?

dans le composant TS

```
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'eshop-app',  
  templateUrl: 'app/app.component.html'  
})  
export class AppComponent {  
  private prenom: string = "Jérémy";  
}
```


BINDING — LIAISON DES DONNÉES

ngModel lie les données du modèle à la vue, et inversement !

- C'est ce qu'on appelle un *data-binding Two-Way*

Angular s'appuie sur le pattern *MVVM* pour le binding

- Comme vu dans l'exercice précédent
- Chaque modification d'un côté entraîne une mise à jour de l'autre côté

EXERCICE

Dans la vue

- Ajouter un bouton qui appelle une fonction qui change la valeur de *prenom*
- Créer cette fonction dans la classe AppComponent

```
<button (click)="changePrenom()">CHANGER</button>
```

Quel est le résultat ?

TEMPLATES

Dans la vue des exercices précédents

- Utilisation de
 - {{ ... }}
 - [(...)]="..."
 - (...)= "..."
- Il en existe d'autres
 - *...="..."
 - [...]="..."
 - bind-...="..."
 - bindon- ...="..."

TEMPLATES

| Syntaxe | Direction de l'information | Type | Exemples |
|--|--|---|--|
| <code>{{ expression }}</code> <code>[attr]="expression"</code> <code>bind-attr="expression"</code> | Sens unique Depuis la source de données vers la vue | Interpolation Propriété Attribut Classe Style | <code>{{ couleur }}</code> <code>[style.color]="couleur"</code> <code>bind-style.color="couleur"</code> |
| <code>(evenement)="traitement"</code> <code>on-evenement="traitement"</code> | Sens unique Depuis la vue vers la source de données | Evènement | <code>(click)="fonction()"</code> <code>on-dblclick="fonction()"</code> <code>(mouseenter)="fonction()"</code> |
| <code>[(attr)]="expression"</code> <code>bindon-attr="expression"</code> | Dans les deux sens | - | <code>[(ngModel)]="couleur"</code> <code>bindon-ngModel="couleur"</code> |
| <code>*directive</code> | Depuis la source de données vers la vue Permet d'encapsuler l'élément dans une zone tampon, manipulée par Angular | - | <code>*ngIf="true"</code> <code>*ngFor="let e of list"</code> <code>*ngSwitchCase=""valeur""</code> |

EXERCICE

Ajouter une liste de choix à la vue (select HTML)

- Quelques couleurs (noir, vert, jaune, bleu)

Ajouter un input de type « color »

Lorsque l'utilisateur change la couleur

- La couleur du paragraphe change en fonction de la sélection !
- Utiliser uniquement ngModel

EXERCICE

Reprendre la classe Produit (nom, prix)

- Créer un nouveau fichier « produit.ts »
- Le composant utilise cette classe, la vue aussi
- La classe sera dans un fichier à part : penser à l'import !

Dans la vue

- Ajouter un input pour la saisie du nom et du prix
- La paragraphe affiche le nom et le prix
 - Et ne doit s'afficher que lorsque le nom est rempli, et que le prix est différent de 0 !
 - Si le prix est négatif, le background du paragraphe applique une classe CSS qui fait devenir le paragraphe rouge
 - Avec une transition !
 - `[class.nom-classe] ou [ngClass]="{ 'nom-class': true }"`



LES COMPOSANTS

Les composants Angular

COMPONENTS

Un composant est une fonctionnalité front-end

- Intégré à un module
- Sépare la partie logique de la partie manipulation DOM
- Réutilisable

Quelques questions à se poser avant de se lancer

- Quel est son rôle ?
- Doit-il prendre place dans le vue HTML ? (avoir une structure HTML, une vue)
- Quels sont ses paramètres / attributs d'entrée ?

COMPONENTS

Dans le fichier *nom-composant.component.ts*

- On a besoin de l'annotation **@Component**, donc on l'importe

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

- On déclare le composant, avec une classe, en l'annotant de **@Component**

```
@Component({  
  selector: 'nom-composant,[nom-composant]',  
  templateUrl: 'app/asc-bold.component.html'  
})  
export class NomComposantComponent { }
```

On précise le sélecteur qu'on utilisera en HTML
'nom-sélecteur' balise HTML
'[nom-sélecteur]' attribut HTML
Ici, on fait les 2

```
<p nom-composant>...</p>
```

```
<nom-composant>...</nom-composant>
```

COMPONENTS

Dans la classe du composant

- On y déclare ce dont on a besoin
 - Attributs, méthodes, ...
- Pour binder une action utilisateur sur l'ensemble du composant
 - On peut utiliser (evenement) dans la vue
 - Ou, plus intéressant, utiliser l'annotation **@HostListener** sur les méthodes
 - Pour ça, il nous faut importer **HostListener**, de **@angular/core**

```
@HostListener('click') onClick() {  
  //Faire quelquechose au clique  
}
```

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

COMPONENTS

Dans la vue du composant (*nom-composant.component.html*)

- On définit la structure HTML
- On peut inclure un contenu en utilisant la balise ng-content

```
<div> <!-- Conteneur du composant -->
  <p>Le paragraphe du composant</p>

  <!-- Contenu issu de la vue qui a appelé ce composant -->
  <ng-content></ng-content>
</div>
```

- « Un contenu ... » sera positionné à l'endroit où se trouve ng-content dans le composant

```
<p nom-composant>Un contenu ...</p>
<nom-composant>Un contenu ...</nom-composant>
```

COMPONENTS

Chaque composant doit être dans un module, il faut donc

- Importer ce composant dans le module
- Le déclarer dans la liste des déclarations

EXERCICE

Créer un composant « asc-bold » qui :

- Met en gras le contenu
- Affiche une alerte « on a cliqué » au clique sur le contenu

```
<p asc-bold>Le contenu à mettre en gras ...</p>
```

COMPONENTS

Dans la classe du composant

- On peut récupérer des informations brutes ou bindées sur le modèle de données

- Utiliser l'annotation **@Input**

- Pour ça, il nous faut importer **Input**, de `@angular/core`

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

- On place cette annotation sur l'attribut à récupérer

```
export class AscBoldComponent {  
  @Input('titre') title: string;  
  @Input() text: string;  
}
```

```
<p nom-composant titre="Le super titre" [text]="produit.nom">Un contenu ...</p>
```

- En utilisant les principes de binding dans la vue
 - Sans les crochets Texte brute
 - Avec les crochets Binding vers une donnée existante dans le modèle

EXERCICE

Créer un composant « asc-bold-element » qui :

- Présente le nom du produit saisi avec une phrase « préfixe »
- Met en gras le nom
- Affiche une alerte « nom » au clique sur le contenu

```
<asc-bold-element prefix="Voici le nom" [text]="produit.nom"></asc-bold-element>
```

EXERCICE — ALLER PLUS LOIN

Créer un composant qui affiche un tooltip au survol de la souris

- CSS
 - bordure arrondie
 - couleur rose #FF5588
 - padding 10px 20px
- Scope: redéfini, nous avons besoin du **texte** du tooltip
- Doit fonctionner en attribut et en élément HTML

```
<p asc-tooltip asc-tooltip-titre="Et voilà un tooltip !">Voilà un message !</p>
```

```
<asc-tooltip asc-tooltip-titre="Et voilà un tooltip !">Voilà un message !</asc-tooltip>
```




LES DIRECTIVES

Les directives Angular

DIRECTIVES

Une directive est une fonctionnalité front-end

- Intégrée à un module
- Il n'y a pas de vue associée
- Certaines sont natives à Angular et sont préfixées par « ng »
 - `ngModel`, `ngClass`, `ngIf`, `ngFor`, `ngSwitch`, `ngSwitchCase`, ...

DIRECTIVES

Trois types de directives

- Les composants
- Les directives d'attributs
 - Modifient le comportement des éléments HTML et/ou de leurs attributs
 - Représentée généralement sous forme d'attribut HTML

```
<p ma-directive>Le contenu</p>
```

- Les directives structurelles
 - Modifient la structure HTML des éléments sur lesquels elles s'appliquent
 - **ngIf** et **ngFor** par exemple

DIRECTIVES

ngClass

- Permet d'appliquer une classe en fonction d'une condition

```
<p ng-class="{ 'une-classe': nom, 'attention': prix < 0 }">...</p>
```

DIRECTIVES

ngIf

- Afficher un bloc en fonction d'une condition vérifiée

```
<div *ngIf="condition_vrai">  
  <p>C'est vrai</p>  
</div>
```

```
<div [ngIf]="condition_vrai">  
  <p>C'est vrai</p>  
</div>
```

DIRECTIVES

ngIf

- Afficher un bloc en fonction d'une condition vérifiée

```
<div *ngIf="condition_vrai; else sinon_on_fait_ca">  
  <p>C'est vrai</p>  
</div>
```

```
<ng-template #sinon_on_fait_ca>  
  <p>C'est faux</p>  
</ng-template>
```

DIRECTIVES

ngFor

- Effectue une boucle for dans la vue
- Se place sur l'élément à répéter

```
<ul>  
  <li *ngFor="let element of list">{{ element }}</li>  
</ul>
```

```
<tr *ngFor="let p of produits">  
  <td>{{ p.nom }}</td>  
  <td>{{ p.prix }}</td>  
</tr>
```

DIRECTIVES

ngSwitch / ngSwitchCase / ngSwitchDefault

- Afficher un contenu selon la valeur d'une donnée

```
<div [ngSwitch]="personne.genre">  
  <p *ngSwitchCase="0">C'est un monsieur</p>  
  <p *ngSwitchCase="1">C'est une madame</p>  
  <p *ngSwitchDefault>C'est quelqu'un qui a des doutes, qui se pose des questions.</p>  
</div>
```


EXERCICE

Créer une liste de produits

- nom, prix
- Afficher cette liste de produits dans un tableau HTML
 - Colonnes Nom, Prix, Infos
- Dans la colonne Infos
 - Si le prix est < 0 , afficher « Négatif »
 - Si le prix est > 0 , afficher « Positif »
 - Si le prix est $= 0$, afficher « Neutre »

Créer un formulaire « nom, prix »

- Ajouter un produit dans la liste des produits en cliquant sur le bouton de validation
- Utiliser `Array.prototype.push(object)`
- Le tableau doit se mettre à jour automatiquement

EXERCICE

Ajouter la possibilité de filtrer les produits sur leur nom

- Le filtre se traite dans le composant
- Utiliser la syntaxe suivante pour vous aider

```
this.produits.filter(p => p.nom.indexOf("valeur") !== -1);
```

- Après l'ajout d'un produit , le filtre doit continuer de s'appliquer

DIRECTIVES

Dans le fichier *nom-directive.directive.ts*

- On a besoin de l'annotation **@Directive**, donc on l'importe

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

- On déclare la directive, avec une classe, en l'annotant de **@Directive**

```
@Directive({  
  selector: '[nom-directive]'  
})  
export class NomDirectiveDirective { }
```

```
<p nom-directive>...</p>
```

DIRECTIVES

Dans la classe de la directive

- Il est possible de faire tout ce que peut faire un composant
- Simplement, la directive n'a pas de vue associée

DIRECTIVES

Dans la classe de la directive

- On peut récupérer l'élément sur lequel cette directive s'applique
 - Utiliser **ElementRef**
 - Pour ça, il nous faut importer **ElementRef**, de `@angular/core`

```
import { Directive, ElementRef } from '@angular/core';
```

- On injecte **ElementRef** dans le constructeur de la directive

- **ATTENTION**
- Ici vous avez un accès direct au DOM

```
export class NomDirectiveDirective {  
  constructor(private el: ElementRef) {  
    //this.el.nativeElement donne accès à l'élément  
  }  
}
```

DIRECTIVES

Pour manipuler plus facilement l'élément, dans la classe de la directive

- On peut récupérer un **Renderer** pour ajouter un style, une classe, modifier sa structure HTML
 - Utiliser **Renderer2** (**Renderer** est déprécié)
 - Pour ça, il nous faut importer **Renderer2**, de `@angular/core`

```
import { Directive, ElementRef, Renderer2 } from '@angular/core';
```

- On injecte **Renderer2** dans le constructeur de la directive

```
export class NomDirectiveDirective {  
  constructor(private el: ElementRef, private renderer: Renderer2) { }  
}
```

DIRECTIVES

Chaque directive doit être dans un module, il faut donc

- Importer cette directive dans le module
- La déclarer dans la liste des déclarations



LES ROUTES

Le routage avec Angular

ROUTES

Une seule page HTML, plusieurs points d'accès

- L'application aura toujours un seul et unique fichier : *index.html*
- C'est Angular qui va
 - Gérer et interpréter les routes
 - Dispatcher vers le bon composant (un peu à la manière de *Spring MVC*)
- On utilisera des composants pour jouer le rôle de contrôleurs

ROUTES

Il est possible de configurer des routes « partielles »

- Composées d'un ou de plusieurs paramètres
- « /route/:param/:param2 »
- « /produit/:id »
- « /fournisseur/:id/produits »

ROUTES

Dans le module principal

- Importer **RouterModule** et **Routes** de `@angular/router`
- Configuration des routes dans une constante

```
//Configuration des routes
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'produit', component: ProduitComponent },
  { path: 'produit/:id', component: ProduitDetailComponent },
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: '**', component: PageNotFoundComponent }
];
```

Composant « Contrôleur »

pathMatch est important : on veut rediriger seulement si le path est strictement "

- Dans la déclaration des imports, importer **RouterModule** en appelant sa méthode `forRoot()`

```
RouterModule.forRoot(routes)
```

ROUTES

Le template du composant principal doit implémenter la balise router-outlet

```
<router-outlet></router-outlet>
```

- C'est à cet emplacement qu'iront les vues des « contrôleurs »

ROUTES

Dans les vues, il est possible de générer les liens, avec ou sans les paramètres

```
<a [routerLink]="['/']">Lien vers l'accueil</a>  
<a [routerLink]="['/produit']">Lien vers les produits</a>  
<a [routerLink]="[p.nom]">Lien vers un produit</a>  
<a [routerLink]="['/produit', p.nom]">Lien vers un produit</a>
```

Et pour naviguer sans le rafraichissement de la page

- Il suffit d'utiliser l'attribut « routerLinkActive »
 - Qui a aussi pour effet de sélectionner la classe à appliquer si le lien est actif (navigation active)

```
<a [routerLink]="['/']" routerLinkActive="active">Lien vers l'accueil</a>  
<a [routerLink]="['/produit']" routerLinkActive="active">Lien vers les produits</a>  
<a [routerLink]="[p.nom]" routerLinkActive="active">Lien vers un produit</a>  
<a [routerLink]="['/produit', p.nom]" routerLinkActive="active">Lien vers un produit</a>
```

ROUTES

Dans un contrôleur, il est possible de récupérer les paramètres

- On a besoin de **ActivatedRoute**, disponible dans *@angular/router*
- Ensuite, on l'injecte dans le constructeur

```
constructor(private route: ActivatedRoute) { }
```

- Pour lire un paramètre de chemin (PathVariable)

```
this.route.params.subscribe(params => {  
  console.log(params);  
});
```

- Pour lire un paramètre de requête

```
this.route.queryParams.subscribe(params => {  
  console.log(params);  
});
```

ROUTES

Dans un contrôleur, il est possible de récupérer le **Router** pour rediriger par exemple

- On a besoin de **Router**, disponible dans *@angular/router*
- Ensuite, on l'injecte dans le constructeur

```
constructor(private router: Router) { }
```

- Pour rediriger vers une page

```
this.router.navigate(['/produit', 42]);
```

ROUTES — AUTHENTIFICATION

Il est possible d'empêcher l'accès à une route selon des conditions

- On va utiliser un « Guard »
- Utiliser un service (**UserService** par exemple)
- Implémenter une méthode `canActivate()` qui retourne un booléen
- Dans les routes concernées, ajouter l'attribut *canActivate* et y préciser le service qui s'en charge

```
//Configuration des routes
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'produit', component: ProduitComponent, canActivate: [ UserService ] },
  { path: 'produit/:id', component: ProduitDetailComponent , canActivate: [ UserService ] },
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: '**', component: PageNotFoundComponent }
];
```


EXERCICE

Créer un contrôleur **HomeController**

- N'affiche qu'un H1

Créer un contrôleur **ProduitComponent**

- Affiche la liste des produits
- Affiche le formulaire d'ajout d'un produit

Créer un contrôleur **ProduitDetailComponent**

- Attend un paramètre de chemin
- Affiche les informations du produit (informations bidons, pas important pour le moment)

EXERCICE

Implémenter bootstrap (CSS)

Créer un CRUD

- Pouvoir lister les produits
- En ajouter
- Les modifier
- Les supprimer

Pour aller plus loin (bonus)

- Pouvoir modifier un produit directement depuis le tableau (des inputs s'affichent dans les cellules)
- Chaque ligne de produit est un composant **ProduitCrudRowComponent**

```
<tr produit-row [produit]="p"></tr>
```