

Business Case MEAN Stack - Aydogan Emre

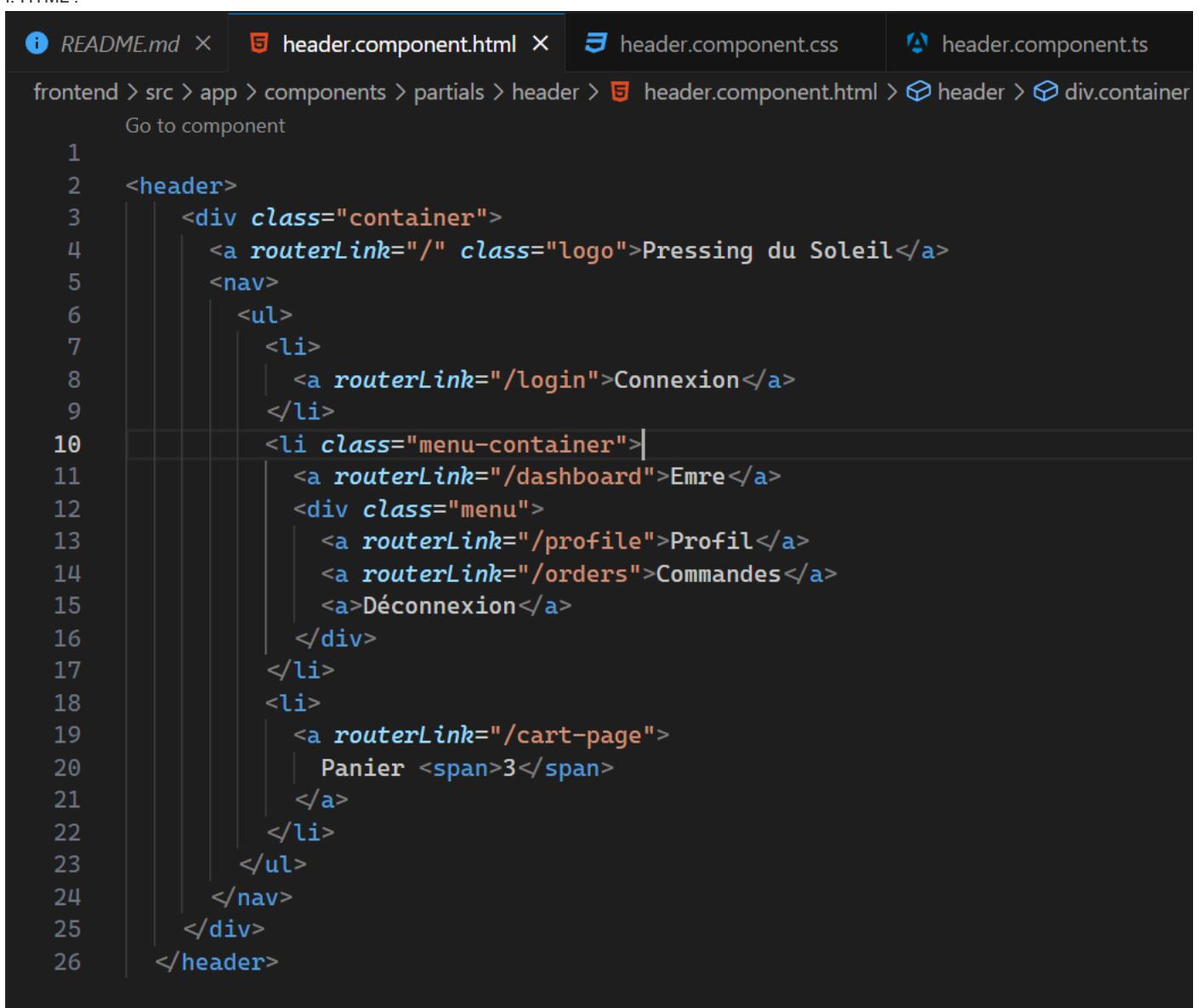
Pré-requis

1. Création du dossier de l'app : bc_v6
2. Installer @angular/cli
3. Création de l'app côté front : ng new frontend --skip-tests (les fichiers de tests ne seront pas utilisés)

Header

1. Générer le Component header dans le dossier frontend.
2. ng g c components/partials/header (création de dossiers afin de mieux s'y retrouver).
3. Appelé le component header dans le fichier app.component.html afin de créer la vue header.

I. HTML :



```
1 <header>
2   <div class="container">
3     <a routerLink="/" class="logo">Pressing du Soleil</a>
4     <nav>
5       <ul>
6         <li>
7           <a routerLink="/login">Connexion</a>
8         </li>
9         <li class="menu-container">
10           <a routerLink="/dashboard">Emre</a>
11           <div class="menu">
12             <a routerLink="/profile">Profil</a>
13             <a routerLink="/orders">Commandes</a>
14             <a>Déconnexion</a>
15           </div>
16         </li>
17         <li>
18           <a routerLink="/cart-page">
19             Panier <span>3</span>
20           </a>
21         </li>
22       </ul>
23     </nav>
24   </div>
25 </header>
```

4. Ajouter du CSS pour le style.

I. Visuellement :

Models

1. Model Produits
2. Création de data.ts histoire d'avoir des datas pour l'instant avec quoi travailler.
 - i. Ajouter les produits
3. Ajouter les images dans le dossier assets
4. Création du Service Produits : `ng g s services/product`
 - I. Méthode Get afin de récupérer tous les produits => `getAll():Product[] { return sample_products; }`
 - II. Injecter le Service Product dans le contructor du component dont je veux utiliser :

```
products:Product[] = []; constructor(private productService:ProductService) { this.products = productService.getAll(); }
```
5. Création du Component Home `ng g c component/pages/home`, appelé la vue comme précédemment dans le fichier `app.component.html`.
6. Installer un package afin d'avoir les étoiles pour un petit coté visuel `npm i ng-starrating`
7. Créez un premier visuel dans le component Home en appelant les données du fichier `data.ts` en se basant sur le model `Product.ts` :

```
<ul>
  <li *ngFor="let product of products">
    <a routerLink="/product/{{product.id}}">
      <img [src]="product.imageUrl" [alt]="product.name"/>
      <div class="content">
        <div class="name">
          {{product.name}}
        </div>
        <span class="favorite {{product.favorite?'':'not'}}">
          ♥
        </span>
        <div class="stars">
          <star-rating
            [value]="product.stars"
            [totalstars]="5"
            checkedcolor="red"
            uncheckedcolor="black"
            size="22px"
            [readonly]="true">
          </star-rating>
        </div>
        <div class="product-item-footer">
          <div class="category">
            <span *ngFor="let category of product.categorys">
              {{category}}
            </span>
          </div>
          <div class="price">
            <span>
              {{ product.price }}
            </span>
          </div>
        </div>
      </div>
    </a>
  </li>
</ul>
```

Visuellement avec du css :

Pressing du Soleil

Connexion Emre Panier 3



Barre de recherche et Routing Angular

1. Ajouter cette méthode à Product Service

I.

```
getAllProductsSearchTerm(searchTerm:string) { return this.getAll().filter( product => product.name.toLowerCase().includes(searchTerm)) }
```

Pour chaque produit cela vérifie le nom du produit contient bien le mot recherché sans se soucier si les lettres sont en majuscules ou minuscules.

2. Ajouter les routes

I. Dans le fichier app-routing.module.ts nous déclarons les routes de l'application:

```
const routes: Routes = [ {path: '', component: HomeComponent}, {path: 'search/:searchTerm', component: HomeComponent} ];
```

II. Plus Besoin d'appeler le component Home dans l'app.component.ts :

```
<app-header></app-header> <router-outlet></router-outlet>
```

Les routes seront gérés dans le fichier app-routing.module.ts :

```
frontend > src > app > app-routing.module.ts > ...
1 import { NgModule } from '@angular/core';
2 import { RouterModule, Routes } from '@angular/router';
3 import { HomeComponent } from './components/pages/home/home.component';
4
5 const routes: Routes = [
6   {path: '', component: HomeComponent},
7   {path: 'search/:searchTerm', component: HomeComponent}
8 ];
9
10 @NgModule({
11   imports: [RouterModule.forRoot(routes)],
12   exports: [RouterModule]
13 })
14 export class AppRoutingModule { }
```

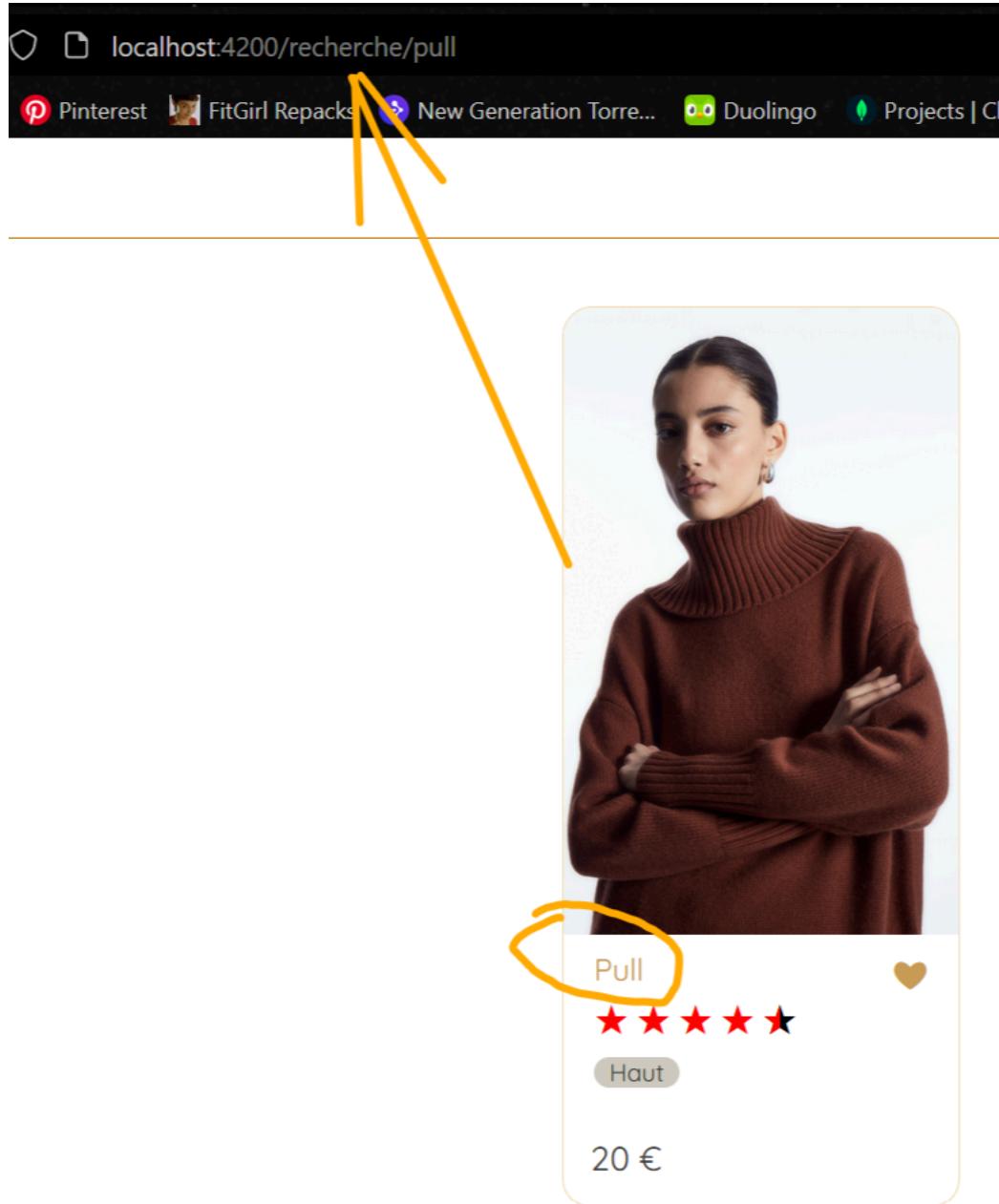
III. {path: 'recherche/:rechercheTerm', component: HomeComponent} : Lorsque l'on navigue à "recherche/quelquechose" dans l'URL, cela charge le component Home.

Le "quelquechose" est ce que nous cherchons.

IV. activatedRoute.params.subscribe((params) => {...}) : Cela écoute les changements dans les paramètres de l'URL. Chaque fois que l'URL change, cette fonction est déclenchée avec les nouveaux paramètres.

```
`if(params.rechercheTerm) {...}` : Vérifie si un terme de recherche est présent dans les paramètres de l'URL. Si oui, cela  
`this.products = this.productService.getAllProductsSearchTerm(params.rechercheTerm);` : Si le terme de recherche est présent  
Sinon (`else`), cela charge simplement tous les produits en utilisant `productService.getAll()`.
```

3. Afficher les résultats



4. Création du component Search : `ng g c components/partials/search`

5. Ajouter le component Search dans le component Home tout en haut : `<app-search></app-search>`

6. Search Component :

The screenshot shows a code editor with three tabs: README.md, search.component.html, and search.component.ts. The search.component.ts tab is active, displaying the following TypeScript code:

```
frontend > src > app > components > partials > search > search.component.ts > SearchComponent > const
● 1 import { Component, OnInit } from '@angular/core';
  2 import { ActivatedRoute } from '@angular/router';
  3 import { Router } from '@angular/router';
  4
  5 @Component({
  6   selector: 'app-search',
  7   templateUrl: './search.component.html',
  8   styleUrls: ['./search.component.css']
  9 })
10 export class SearchComponent implements OnInit {
11   searchTerm = '';
12
13 constructor(activatedRoute: ActivatedRoute, private router: Router) {
14   activatedRoute.params.subscribe((params) =>{
15     if (params.searchTerm) this.searchTerm = params.searchTerm;
16   });
17 }
18
19 ngOnInit(): void {
20 }
21
22 search(term: string): void {
23   if (term)
24     this.router.navigateByUrl('/recherche/' + term)
25 }
26
27 }
28
```

activatedRoute permet d'accéder aux paramètres de l'URL, tandis que router est utilisé pour la navigation vers une autre page.
Écoute des changements dans l'URL:

Dans le constructor, le component écoute les changements dans les paramètres de l'URL à l'aide de activatedRoute.params.subscribe(...).

Permettant de mettre à jour le terme de recherche (searchTerm) chaque fois que l'utilisateur modifie le terme de recherche dans l'URL.

Méthode search(term: string):

Cette méthode est déclenchée lorsque l'utilisateur effectue une recherche.

Elle prend le terme de recherche en tant qu'argument.

Si le terme de recherche n'est pas vide, elle utilise le router pour naviguer vers une nouvelle URL qui contient le terme de recherche, par exemple /recherche/t-shirt.

7. HTML :

README.md M

search.component.html U X

frontend > src > app > components > partials > search > search.component.html > div

Go to component

```
1  <div>
2    <input #s type="text" placeholder="Rechercher un Produit ... "
3      |   (keyup.enter)="search(s.value)"
4      |   [value]="searchTerm"
5    />
6    <button (click)="search(s.value)">
7      |   Rechercher
8    </button>
9  </div>
```

I. Résultat avec une faute:

The screenshot shows a web browser window with the URL `localhost:4200/recherche/chemi`. The search bar contains the text "chemi". A button labeled "Rechercher" is visible. To the right, there is a product card for a man wearing a white shirt and dark trousers. The card includes the word "Chemise", a 5-star rating, a "Haut" button, and a price of "15 €".

II. Résultat avec du CSS :

veste

Rechercher



Tags des Produits

1. Création de Model Tag

I. Ajouter des données d'exemple pour l'instant dans data.ts

II.

```
const sample_tags: Tag[] = [ { name: 'All', count: 8 }, { name: 'Été', count: 2 }, { name: 'Automne', count: 2 }, { name: 'Hiver', count: 3 },
```

2. Ajouter 2 méthodes au Product Service

- méthode getAllTags :

```
getAllTags(): Tag[] { return sample_tags; }
```

Elle utilise une variable appelée sample_tags qui contient tous les tags

- méthode getAllProductsByTag :

```
getAllProductsByTag(tag: string): Product[] { return tag == "All"? this.getAll(): this.getAll().filter( product => product.tags?. includes(
```

Si le tag est "All" alors tous les produits doivent être retournés donc la fonction renvoie simplement tous les produits en utilisant this.getAll()

Sinon la fonction filtre les produits en fonction du tag spécifié

Elle parcourt tous les produits et vérifie si le tag spécifié est inclus dans les tags du produit

Si c'est le cas, le produit est ajouté à un tableau qui sera retourné

3. Ajouter la Route Tags

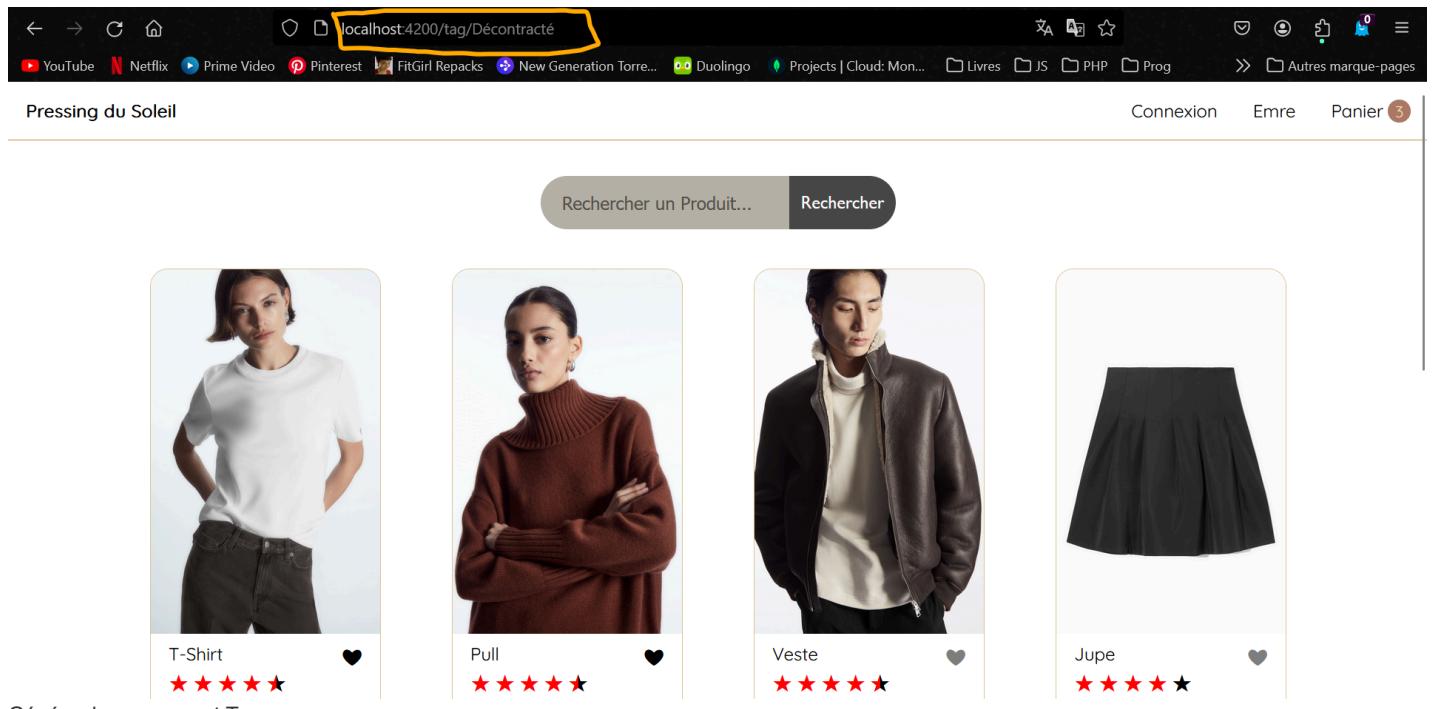
I. {path: 'tag/:tag', component: HomeComponent} (Similaire à la route Recherche)

4. Afficher les résultats dans le component Home

I. Si un paramètre appelé tag est présent dans l'URL cela signifie qu'un tag spécifique a été sélectionné. Dans ce cas il utilise la méthode

getAllProductsByTag() du service productService pour obtenir les produits correspondants.

II. Test dans l'URL :



5. Générer le component Tags

ng g c components/partials/tags

I. l'ajouter au Home component

II. TS-HTML-CSS :

```
TS: tags?:Tag[];  
constructor(productService: ProductService) { this.tags = productService.getAllTags(); }
```

Récupèrons tous les tags à partir du service productService en appelant la méthode getAllTags() .

Afin de les stocker dans la variable tags du composant, les tags seront disponibles pour être utilisés dans le template HTML.

HTML:

```
frontend > src > app > components > partials > tags > tags.component.html > div  
Go to component  
1  <div *ngIf="tags">  
2    <a *ngFor="let tag of tags" routerLink="/tag/{{ tag.name }}">  
3      {{ tag.name }}({{ tag.count }})  
4    </a>  
5  </div>
```

Visuellement :

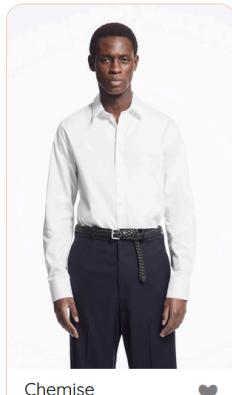
Rechercher un Produit...

Rechercher

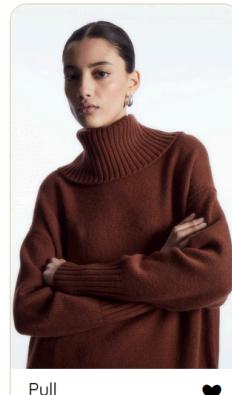
All(8) Été(2) Automne(2) Hiver(2) Printemps(6) Décontracté(5) Sportif(1) Formel(1) Élégant(5) Moderne(4)



T-Shirt



Chemise



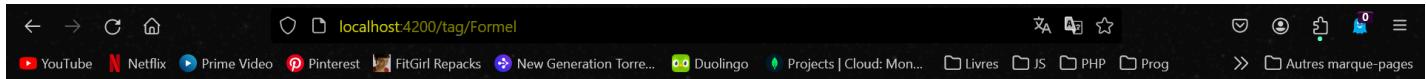
Pull



Veste

CSS + Tag Formel :

Résultat de Chemise :



Rechercher un Produit...

Rechercher

Tous(8)

Été(3)

Automne(2)

Hiver(2)

Printemps(6)

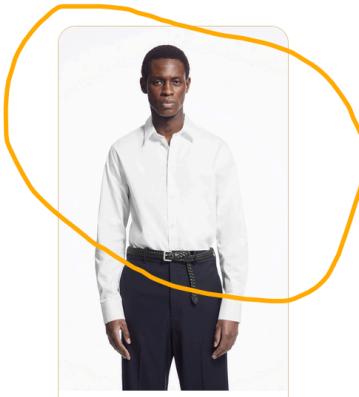
Décontracté(5)

Sportif(1)

Formel(1)

Élégant(4)

Moderne(4)



Page du Produit

1. Ajouter la méthode `getProductById` au fichier `product.service.ts`:

```
getProductById(productId: string){ return this.getAll().find(product => product.id == productId) ?? new Product(); }
```

I. La méthode `getAll` récupère tous les produits, la méthode `find()` recherche le produit dont l'id correspond à l'identifiant passé en argument `productId`, si le produit est trouvé il est renvoyé, si rien n'est trouvé elle renverra `undefined` une valeur par défaut `new Product()` est renvoyé. Cela me garantit que la méthode renvoie toujours quelque chose même si aucun produit n'est trouvé.

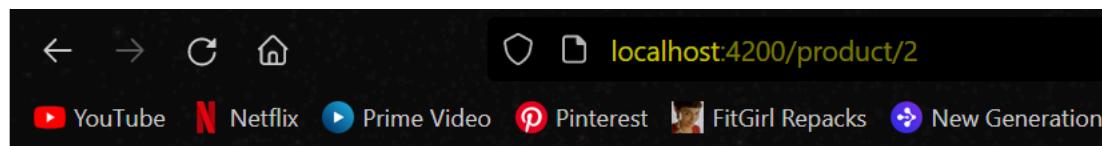
2. Générer le component Page Produit

I. `ng g c components/pages/product-page` (respecter l'arborescence du projet afin de s'y retrouver au mieux)

3. Ajouter la route

I. `{ path: 'product/:id', component: ProductPageComponent }` dans le fichier `app-routing.module.ts`

II. Lorsque l'on clique sur un produit :



Pressing du Soleil

product-page works!

4. TS-HTML-CSS

TS :

```
product!: Product;
```

La variable product est déclarée avec le type Product. Le ! garantie que cette variable sera initialisée avant qu'elle soit utilisée

Le constructeur prend deux arguments :

```
`constructor(activatedRoute: ActivatedRoute, productService: ProductService)`
```

Il écoute les changements dans les paramètres de l'URL à l'aide de:

```
`activatedRoute.params.subscribe(...)`.
```

```
`{
  activatedRoute.params.subscribe((params) =>{
    if (params.id)
      this.product = productService.getProductById(params.id);
  })
}`
```

Lorsqu'un nouvel id de produit est fourni dans les paramètres de l'URL, la méthode `getProductById()` du service `productService` est appelée pour récupérer les détails du produit.

Le produit récupéré est alors assigné à la variable `product`.

II. HTML-CSS Résultat :



Jean



★★★★★

Bas

Printemps

Automne

Moderne

Décontracté

État du Vêtement : Neuf Bon état Abîmé

Matériaux : Cuir Daim Velours Cotton Soie Laine

Satin Denim

Services: (2 maximums) Repassage Nettoyage à sec

Traitement anti-taches Réparation de vêtements

Prix: €30.00

Ajouter Au Panier

Page Panier

1. Création d'un model de Structure Panier

I.

```
import { Product } from "./Product"; export class CartItem{ constructor(public product: Product){ } quantity: number = 1;
```

2. Création d'un model de Panier

I.

```
import { CartItem } from "./CartItem"; export class Cart{ items: CartItem[] = []; totalPrice: number = 0; totalCount: number = 0;
```

3. Générer un service Panier

I. ng g s services/cart

4. Permettre d'ajouter/supprimer/modifier/changer la quantité d'un ou plusieurs Produit(s) dans son panier :

I. addCart(product: Product): void: Ajoute un produit au panier. Vérifie si le produit est déjà dans le panier. Si c'est le cas il fait rien. Sinon il ajoute un nouvel élément au panier avec ce produit.

II. removeFromCart(productId: string): void Supprime un produit du panier en filtrant les éléments du panier pour ne garder que ceux dont l'ID du produit ne correspond pas à celui fourni.

III. changeQuantity(productId: string, quantity: number): Modifie la quantité d'un produit dans le panier en cherchant l'élément correspondant dans le panier et en mettant à jour sa quantité et son prix en conséquence

IV. clearCart(): Vide à zéro le panier en le réinitialisant à un nouveau panier vide

V. getCartObservable(): Observable<Cart> : Renvoie un Observable qui émet des mises à jour du panier. Les composants peuvent s'abonner à cet Observable pour être informés des changements dans le panier

VI. setCartToLocalStorage(): void: Met à jour le panier dans le localStorage du navigateur. Calcul le prix total et le nombre total d'articles dans le panier puis sauvegarde le panier dans le localStroage et notifie les abonnés à l'Observable

VII. getCartFromLocalStorage(): Cart: Récupère le panier à partir du stockage local du navigateur. S'il n'existe pas, il crée un nouveau panier vide

5. Générer un component page Panier

I. Ajouter la route

```
{ path: 'panier', component: CartComponent },  
II. TS-HTML-CSS
```

TS.

Properties :

```
cart!: Cart;
```

Stock les données du panier actuel.

Constructor:

```
Injection du service de panier CartService
```

Abonnement à l'Observable du panier: Lorsque le composant est initialisé il s'abonne à l'Observable du panier à l'aide de la méthode `getCartObservable()` du service `CartService`. À chaque mise à jour du panier il met à jour la propriété `cart` du composant

```
removeFromCart(cartItem: CartItem): Cette méthode est appelée lorsqu'un utilisateur souhaite supprimer un élément du panier, elle utilise le service CartService pour supprimer l'élément du panier en fonction de l'ID du produit
```

```
changeQuantity(cartItem: CartItem, quantityInString: string): Cette méthode est appelée lorsqu'un utilisateur souhaite modifier la quantité d'un élément dans le panier, elle prend la nouvelle quantité (fournie sous forme de chaîne de caractères) et la convertit en nombre entier. Ensuite, elle utilise le service CartService pour mettre à jour la quantité de l'élément dans le panier en fonction de l'ID du produit associé à cet élément
```

HTML & CSS, Résultat :

Mon Panier

	Chemise	3	€45.00	<button>Supprimer</button>
	Pull	2	€40.00	<button>Supprimer</button>
	Veste	1	€30.00	<button>Supprimer</button>

Total: 6
Prix: €115.00

[Passer au Paiement](#)

Plus qu'à rendre dynamique l'affichage des éléments présents dans le panier situé dans le header :

dans le fichier `header.component.ts` :

Propriété :

```
cartQuantity : Stock le nombre total d'articles dans le panier.
```

Au début elle est initialisée à 0

Constructor :

Injection du `cartService`

Abonnement à l'Observable du panier, lorsque le composant est initialisé il s'abonne à l'Observable du panier avec la méthode `getCartObservable()` du service `CartService`

À chaque mise à jour du panier la fonction de rappel est déclenchée avec le nouveau panier

Mise à jour de `cartQuantity` A chaque fois qu'une mise à jour du panier est reçue le nombre total d'articles dans le panier (`newCart.totalCount`) est assigné à la propriété `cartQuantity`. Cela met à jour dynamiquement le nombre d'articles affiché dans le header

HTML :

```
<li> Panier <span *ngIf="cartQuantity">{{ cartQuantity }}</span> </a> </li>
```

Panier 6

NOT FOUND Barre de Recherche & Panier

1. Générer le component dans le dossier partials

```
I. ng g c components/partials/not-found
```

II. dans le fichier TS :

Initialisation de 4 décorateurs `@Input()` sont utilisés pour définir des propriétés d'entrée, ces valeurs peuvent être passées au composant depuis son parent.

Les propriétés `visible`, `notFoundMessage`, `resetLinkText`, `resetLinkRoute` sont toutes des propriétés d'entrée qui peuvent être définies à partir du composant parent lors de l'utilisation du composant `NotFoundComponent` dans le template.

Les propriétés d'entrée permettent de personnaliser l'apparence et le comportement du component selon les besoins, dans mon cas un message et un lien.

2. l'Ajouter aux pages nécessaires...

I. HTML du component notFound :

```
<div *ngIf="visible"> {{ notFoundMessage }} <a [routerLink]="resetLinkRoute">{{ resetLinkText }}</a></div>
```

Celà affiche ou masque l'élément en fonction de la valeur de la propriété visible du component, si visible est évalué à true donc le contenu à l'intérieur de la balise `<div>` sera affiché sinon il sera masqué

II. Panier HTML :

`[visible]="!cart || !cart.items.length"` : Cette propriété d'entrée visible détermine si le component "NotFoundComponent" est visible ou non.

Elle est définie en fonction de la condition si panier est vide.

Si le panier est vide (`!cart || !cart.items.length` évalué à true) le component "NotFoundComponent" sera visible.

`notFoundMessage = "Votre panier est vide."` : le message à afficher.

`resetLinkText="Retour à l'accueil"` : C'est le lien permettant à l'utilisateur de revenir à la page d'accueil

Résultat avec du CSS :

The screenshot shows the header of the website with the logo "Sunny Laundry" and navigation links for "Accueil", "Nos Produits", "Connexion", "Emre", and "Panier". Below the header, the main content area has a heading "Mon Panier". In the center, there is a message "Votre panier est vide." with a button labeled "Retour à l'accueil" below it.

Page Produits :

Même principe que le panier, redirige vers la liste des produits.

Résultat de la recherche ne donnant rien 'azerty' :

The screenshot shows the header of the website with the logo "Sunny Laundry" and navigation links for "Accueil", "Nos Produits", "Connexion", "Emre", and "Panier". Below the header, there is a search bar with the input "azerty" and a "Rechercher" button. Below the search bar, there is a row of category filters: "Tous(8)", "Été(3)", "Automne(2)", "Hiver(2)", "Printemps(6)", "Décontracté(5)", "Sportif(1)", "Formel(1)", "Élégant(4)", and "Moderne(4)". In the main content area, there is a message "Aucun N'élément Trouvé!" with a button labeled "Retourne à la Recherche de ton Produit" below it.

Page Single Produit :

Produit Inexistant [Retour à l'accueil](#)

BACK-END

Ce Projet est MEAN STACK, Utilisation de MongoDB, Express, (Angular) et NodeJS

1. Connexion au Back

I. Création du dossier backend à la racine du Projet, à coté du dossier frontend.

II. Se situé dans le dossier backend : `npm init -y` dans le terminal, puis `npm install typescript`

III. Création du fichier de config dans le dossier back : `tsconfig.json`.

IV. Création du fichier `.gitignore` afin de ne pas push certains fichiers, surtout le dossier `node_modules`.

V. Copier le fichier `data.ts` au `backend/src` (nos samples data du début), suppression des imports, remplacer les tableau `Product`, `Tag`, `Service` en `any`.

VI. Installation de express cors : `npm install express cors`

VII. dans le fichier `package.json`, les dépendances ont été correctement installés (cors et express à l'instant) :

```
"dependencies": {  
  "cors": "^2.8.5",  
  "express": "^4.19.2",  
  "typescript": "^5.4.4"  
}
```

VIII. Création de `server.ts`

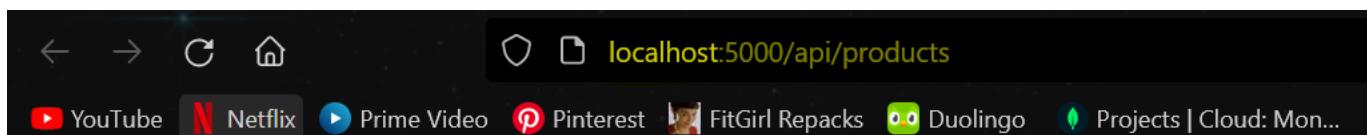
IX. `npm install ts-node --save-dev`, `npm install nodemon --save-dev`

X. En dessous de script nous ajouté notre config `start` :

```
"scripts": { "start": "cd src && nodemon server.ts", "test": "echo \"Error: no test specified\" && exit 1" } plus qu'a lancé le serveur avec  
npm start !
```

```
> backend@1.0.0 start  
> cd src && nodemon server.ts  
  
[nodemon] 3.1.0  
[nodemon] to restart at any time, enter 'rs'  
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: ts,json  
[nodemon] starting 'ts-node server.ts'  
Le site est hébergé sur http://localhost:5000
```

Résultat sur l'URL : `http://localhost:5000/api/products` :



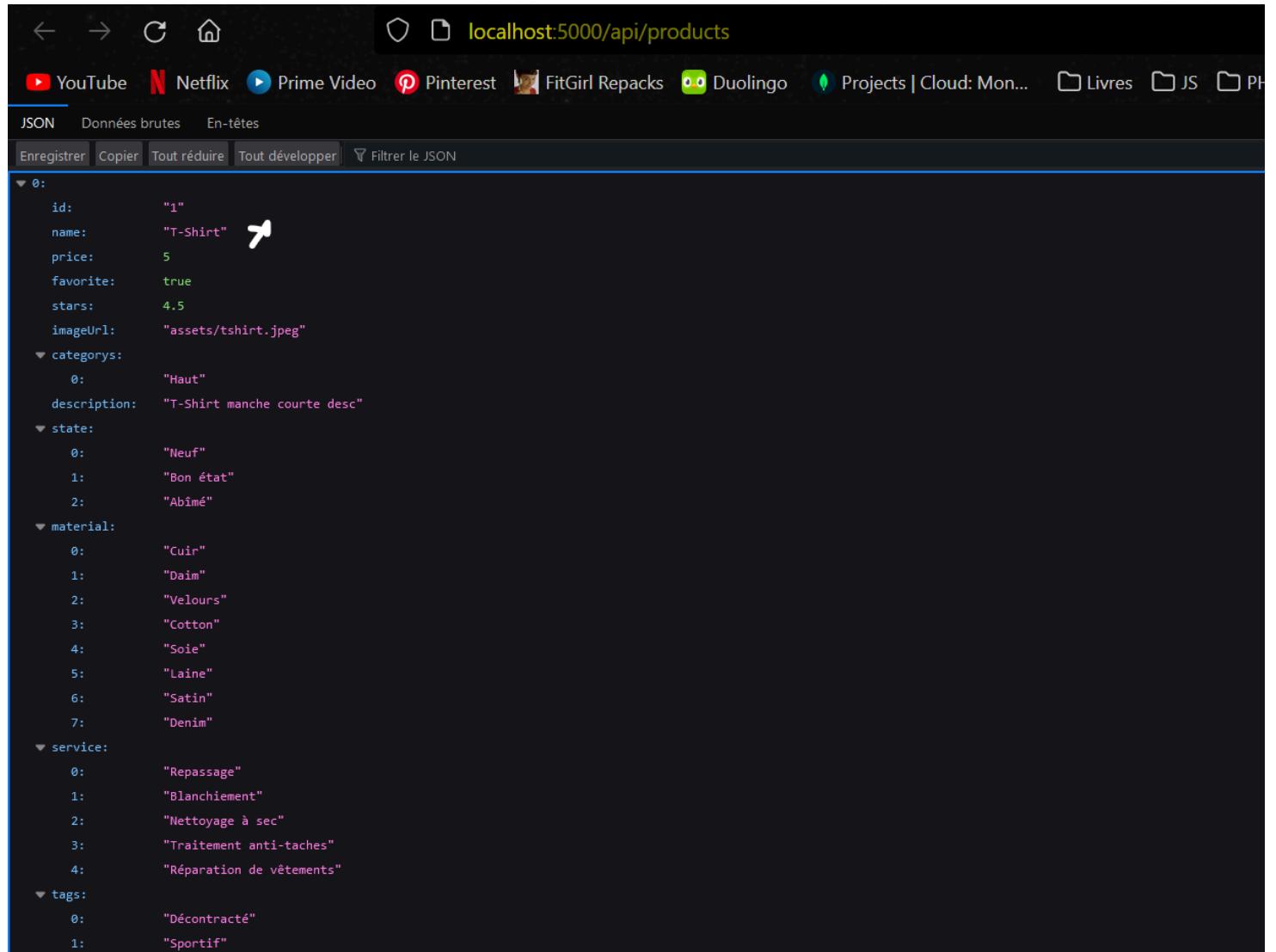
bonjour le monde!

Tout fonctionne !

A la place de ce bonjour je vais plutot inmplémenter mes produits.

```
app.get("/api/products", (req, res) => {     res.send(sample_products);   })
```

Plus qu'à recharger la page :



```
0:
  id: "1"
  name: "T-Shirt" 
  price: 5
  favorite: true
  stars: 4.5
  imageUrl: "assets/tshirt.jpeg"
  categorys:
    0: "Haut"
    description: "T-Shirt manche courte desc"
  state:
    0: "Neuf"
    1: "Bon état"
    2: "Abîmé"
  material:
    0: "Cuir"
    1: "Daim"
    2: "Velours"
    3: "Cotton"
    4: "Soie"
    5: "Laine"
    6: "Satin"
    7: "Denim"
  service:
    0: "Repassage"
    1: "Blanchiment"
    2: "Nettoyage à sec"
    3: "Traitement anti-taches"
    4: "Réparation de vêtements"
  tags:
    0: "Décontracté"
    1: "Sportif"
```

Nos Produits sont bien là au format JSON 😊