

Département d'Informatique

Filiere : IAAD

A ,U :2023-2024



جامعة مولاي إسماعيل
 UNIVERSITÉ MOULAY ISMAÏL



كلية العلوم
FACULTÉ DES SCIENCES

Université Moulay Ismail faculté des Sciences Meknès

Département d'Informatique

TP N 4 : Angular Framework

Module : Systèmes Distribués

Réalisé par :

- *Illa Meryeme*

Première partie :

Partie 1 :

Cette activité concerne Angular et pour l'appliquer, tout d'abord il faut installer node js et on vérifie dans la ligne de commande sa version

```
c:\> Invite de commandes
node: '20.12.2',
acorn: '8.11.3',
ada: '2.7.6',
ares: '1.27.0',
base64: '0.5.2',
brotli: '1.1.0',
cjs_module_lexer: '1.2.2',
cldr: '44.1',
icu: '74.2',
llhttp: '8.1.2',
modules: '115',
napi: '9',
nghttp2: '1.60.0',
nghttp3: '0.7.0',
ngtcp2: '0.8.1',
openssl: '3.0.13+quic',
simdutf: '4.0.8',
tz: '2024a',
undici: '5.28.4',
unicode: '15.1',
uv: '1.46.0',
uvwasi: '0.0.20',
v8: '11.3.244.8-node.19',
zlib: '1.3.0.1-motley-40e35a7'
}

C:\Users\lenovo>npm --version
10.5.0

C:\Users\lenovo>
```

Après on installe Angular CLI de manière globale et vérifier sa version

```
Sélection C:\Windows\system32\cmd.exe

Global setting: enabled
Local setting: No local workspace configuration file.
Effective status: enabled

Angular CLI: 17.3.6
Node: 20.12.2
Package Manager: npm 10.5.0
OS: win32 x64

Angular:
...
Package          Version
-----
@angular-devkit/architect    0.1703.6 (cli-only)
@angular-devkit/core         17.3.6 (cli-only)
@angular-devkit/schematics   17.3.6 (cli-only)
@schematics/angular          17.3.6 (cli-only)

C:\Users\lenovo>
```

On crée un projet Angular et pour le faire on utilise la commande avec l'utilisation de la version module et pour l'indiquer on utilise flag --no-standalone (pour générer les modules avec le projet)

```
npm install

C:\Users\lenovo>ng new FirstApp --no-standalone
? Which stylesheet format would you like to use? CSS [https://developer.mozilla.org/docs/Web/CSS]
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? No
CREATE FirstApp/angular.json (2961 bytes)
CREATE FirstApp/package.json (1078 bytes)
CREATE FirstApp/README.md (1089 bytes)
CREATE FirstApp/tsconfig.json (889 bytes)
CREATE FirstApp/.editorconfig (290 bytes)
CREATE FirstApp/.gitignore (590 bytes)
CREATE FirstApp/tsconfig.app.json (277 bytes)
CREATE FirstApp/tsconfig.spec.json (287 bytes)
CREATE FirstApp/.vscode/extensions.json (134 bytes)
CREATE FirstApp/.vscode/launch.json (490 bytes)
CREATE FirstApp/.vscode/tasks.json (980 bytes)
CREATE FirstApp/src/main.ts (221 bytes)
CREATE FirstApp/src/favicon.ico (15086 bytes)
CREATE FirstApp/src/index.html (307 bytes)
CREATE FirstApp/src/styles.css (81 bytes)
CREATE FirstApp/src/app/app-routing.module.ts (255 bytes)
CREATE FirstApp/src/app/app.module.ts (411 bytes)
CREATE FirstApp/src/app/app.component.html (20239 bytes)
CREATE FirstApp/src/app/app.component.spec.ts (1094 bytes)
CREATE FirstApp/src/app/app.component.ts (219 bytes)
CREATE FirstApp/src/app/app.component.css (0 bytes)
CREATE FirstApp/src/assets/.gitkeep (0 bytes)
| Installing packages (npm)...
```

```
npm

Nothing to be done.

C:\Users\lenovo>cd FirstApp

C:\Users\lenovo\FirstApp>ng serve
? Would you like to share pseudonymous usage data about this project with the Angular Team at Google under Google's Privacy Policy at https://policies.google.com/privacy. For more details and how to change this setting, see https://angular.io/analytics. Yes

Thank you for sharing pseudonymous usage data. Should you change your mind, the following command will disable this feature entirely:

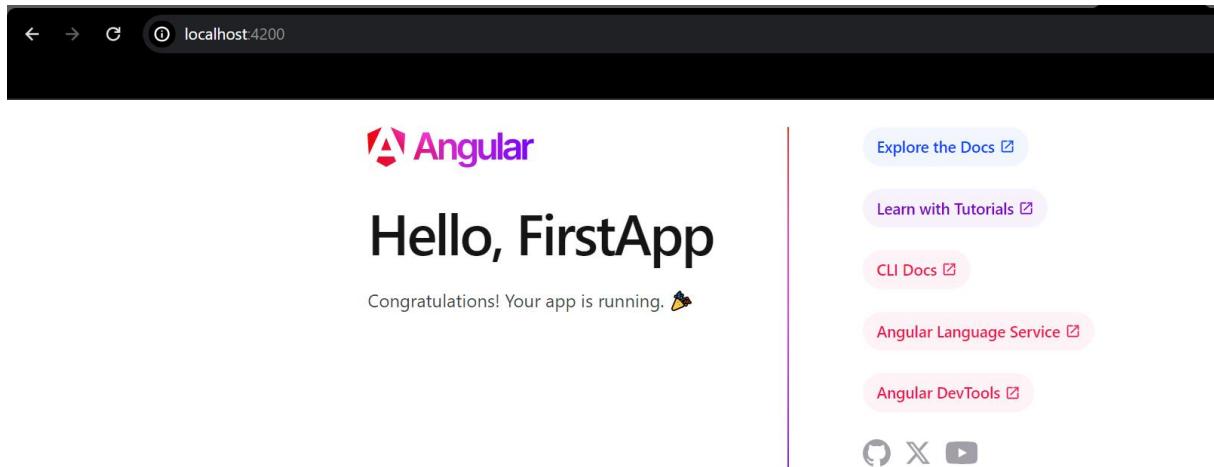
  ng analytics disable

Global setting: enabled
Local setting: enabled
Effective status: enabled
Initial chunk files | Names      | Raw size
polyfills.js        | polyfills  | 83.60 kB
main.js            | main       | 23.00 kB
styles.css         | styles     | 95 bytes
                           | Initial total | 106.69 kB

Application bundle generation complete. [2.554 seconds]

Watch mode enabled. Watching for file changes...
  Local:  http://localhost:4200/
  press h + enter to show help
```

On teste



On utilise l'éditeur Intellij et on fait le même teste dans le terminal de l'éditeur

On crée une petite application avec un petit menu, premièrement on doit installer bootstrap

```
C:\Users\lenovo\FirstApp>npm i bootstrap bootstrap-icons

added 3 packages, and audited 924 packages in 11s

121 packages are looking for funding
  run `npm fund` for details

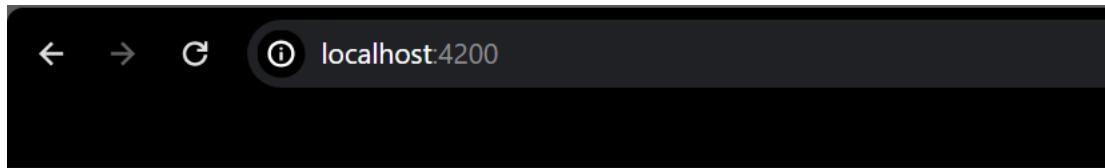
found 0 vulnerabilities
```

On déclare bootstrap dans le fichier angular.json

```
tsconfig.json × index.html × app.component.html × angular.json ×  
  "tsConfig": "tsconfig.app.json",  
  "assets": [  
    "src/favicon.ico",  
    "src/assets"  
  ],  
  "styles": [  
    "src/styles.css",  
    "node_modules/bootstrap/dist/css/bootstrap.min.css"  
  ],  
  "scripts": []
```

On crée 2 boutons dans app.component.html

```
IE.md × index.html × app.component.html × angular.json ×  
<div class="p-3">  
  <ul class="nav nav-pills">  
    <li>  
      <button class="btn btn-outline-primary ms-2">Home</button>  
    </li>  
    <li>  
      <button class="btn btn-outline-primary ms-2">Products</button>  
    </li>  
  </ul>  
</div>
```



[Home](#) [Products](#)

Pour créer les composants on ajoute la commande

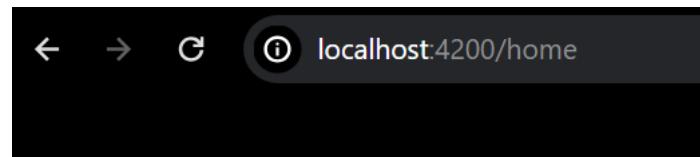
```
C:\Users\lenovo\FirstApp>ng g c home
CREATE src/app/home/home.component.html (20 bytes)
CREATE src/app/home/home.component.spec.ts (610 bytes)
CREATE src/app/home/home.component.ts (201 bytes)
CREATE src/app/home/home.component.css (0 bytes)
UPDATE src/app/app.module.ts (487 bytes)

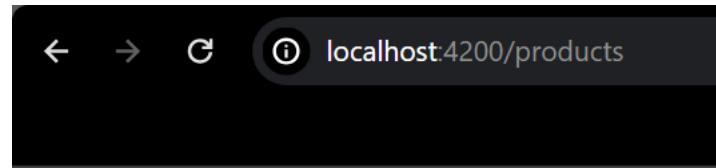
C:\Users\lenovo\FirstApp>ng g c products
CREATE src/app/products/products.component.html (24 bytes)
CREATE src/app/products/products.component.spec.ts (638 bytes)
CREATE src/app/products/products.component.ts (217 bytes)
CREATE src/app/products/products.component.css (0 bytes)
UPDATE src/app/app.module.ts (579 bytes)
```

On va utiliser le système de routage

```
ME.md × index.html × app.component.html × app-routing.module.ts × angular.json ×  
import { RouterModule, Routes } from '@angular/router';  
import { HomeComponent } from './home/home.component';  
import { ProductsComponent } from './products/products.component';  
  
const routes: Routes = [  
  {path: "home", component: HomeComponent},  
  {path: "products", component: ProductsComponent}  
];  
  
@NgModule({  
  imports: [RouterModule.forRoot(routes)],  
  exports: [RouterModule]  
})  
export class AppRoutingModule {}
```

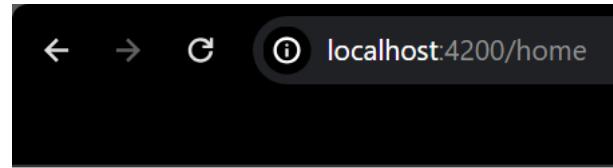
```
ME.md × index.html × app.component.html × app-routing.module.ts × app.component.spec.ts × angular.json ×  
<div class="p-3">  
  <ul class="nav nav-pills">  
    <li>  
      <button routerLink="/home" class="btn btn-outline-primary ms-2">Home</button>  
    </li>  
    <li>  
      <button routerLink="/products" class="btn btn-outline-primary ms-2">Products</button>  
    </li>  
  </ul>  
</div>
```





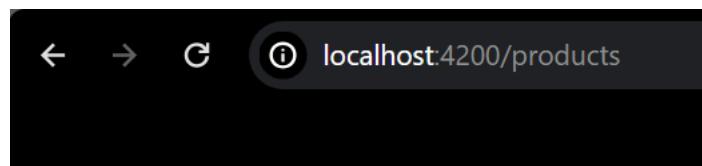
[Home](#) [Products](#)

On ajoute l'élément router-outlet pour afficher les composants home et products



[Home](#) [Products](#)

home works!



[Home](#) [Products](#)

products works!

On veut afficher une liste des produits

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

@Component({
  selector: 'app-products',
  templateUrl: './products.component.html',
  styleUrls: ['./products.component.css'
})

export class ProductsComponent implements OnInit{
  public products= [
    {"id": 1, "name":"Computer", "price":4300},
    {"id": 2, "name":"Printer", "price":3255},
    {"id": 3, "name":"Smartphone", "price":2100},
    {"id": 4, "name":"Mouse", "price":111}
  ];
  constructor() {
  }
  ngOnInit(): void {
  }
}
```

```
<div class="card-body">
  <table class="table">
    <thead>
      <tr>
        <th>ID</th><th>Name</th><th>Price</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let p of products">
        <td>{{p.id}}</td>
        <td>{{p.name}}</td>
        <td>{{p.price}}</td>
      </tr>
    </tbody>
  </table>
</div>
</div>
```

← → ⌂ localhost:4200/products

Home Products

ID	Name	Price
1	Computer	4300
2	Printer	3255
3	Smartphone	2100
4	Mouse	111

On utilise la méthode ngOnInit()

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

@Component({
  selector: 'app-products',
  templateUrl: './products.component.html',
  styleUrls: ['./products.component.css'
})
```

```
export class ProductsComponent implements OnInit{
  public products : any;
  constructor() {
  }
  ngOnInit(): void {
    this.products = [
      {"id": 1, "name":"Computer", "price":4300},
      {"id": 2, "name":"Printer", "price":3255},
      {"id": 3, "name":"Smartphone", "price":2100},
      {"id": 4, "name":"Mouse", "price":111}
    ];
  }
}
```

On fait un test

```
<div class="card-body">
  <table class="table">
    <thead>
      <tr>
        <th>ID</th><th>Name</th><th>Price</th>
      </tr>
    </thead>
    <tbody *ngIf="products">
      <tr *ngFor="let p of products">
        <td>{{p.id}}</td>
        <td>{{p.name}}</td>
        <td>{{p.price}}</td>
      </tr>
    </tbody>
  </table>
</div>
</div>
```

On ajoute un bouton delete qui permet de supprimer les produits

```
<td>{{p.id}}</td>
<td>{{p.name}}</td>
<td>{{p.price}}</td>
<td>
  <button (click)="deleteProduct(p)" class="btn btn-danger">Delete</button>
</td>
</tr>
</tbody>
```

```
app.module.ts × products.component.spec.ts × products.component.ts × home.compo
  ...
  [{"id": 1, "name": "Computer", "price": 4300},
   {"id": 2, "name": "Printer", "price": 3255},
   {"id": 3, "name": "Smartphone", "price": 2100},
   {"id": 4, "name": "Mouse", "price": 111}
  ];
}

deleteProduct(p: any) {
  let index = this.products.indexOf(p);
  this.products.splice(index, 1);
}

}
```



Home Products

ID	Name	Price	
1	Computer	4300	<button>Delete</button>
2	Printer	3255	<button>Delete</button>
3	Smartphone	2100	<button>Delete</button>
4	Mouse	111	<button>Delete</button>

On essaie de supprimer le produit de l'id 4 et ça se passe bien (c'est juste un traitement)

ID	Name	Price	
1	Computer	4300	<button>Delete</button>
2	Printer	3255	<button>Delete</button>
3	Smartphone	2100	<button>Delete</button>

On ajoute un formulaire pour chercher les produits

Importer FormsModule et ReactiveFormsModule pour utiliser les formulaires

```

    ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    ReactiveFormsModule
  ],

```

On ajouter le formulaire avec une méthode search() pour faire la recherche



```
    search() {
      console.log(this.keyword);
      let result: any[] = [];
      for (let p of this.products) {
        if(p.name.includes(this.keyword)) {
          result.push(p);
        }
      }
      this.products= result;
    }
}
```

The screenshot shows a browser window with the URL `localhost:4200/products`. At the top, there are navigation icons for back, forward, and refresh. Below the address bar, there are two buttons: "Home" and "Products". The main content area contains a search bar with the letter "C" and a "Search" button. Below the search bar is a table with three columns: "ID", "Name", and "Price". There is one row of data: ID 1, Name Computer, and Price 4300. To the right of the "Delete" button in the last column is a red rounded rectangle.

ID	Name	Price
1	Computer	4300

On peut utiliser filter pour chercher un produit

```
    search() {
      this.products= this.products.filter((p:any) => p.name.includes(this.keyword));
    }
}
```

Au lieu d'utiliser routerLink, on utilise la deuxième solution

```

    })
}

export class AppComponent {
  title = 'FirstApp';
  public currentRoute : any;

  constructor(private router: Router) {

  }

  gotoHome(): void {
    this.currentRoute= "home";
    this.router.navigateByUrl( url: "/home");
  }

  gotoProducts(): void {
    this.currentRoute= "products";
    this.router.navigateByUrl( url: "/products");
  }
}

```

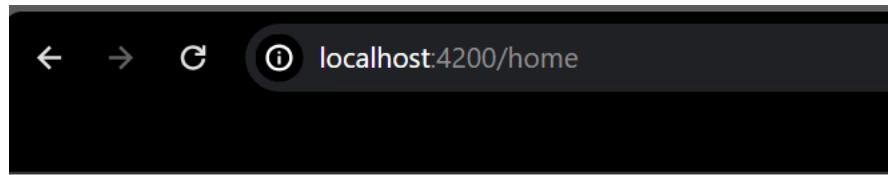
On remplace class par [ngClass]

```

<div class="p-3">
  <ul class="nav nav-pills">
    <li>
      <button (click)="gotoHome()" [ngClass]="currentRoute=='home'? 'btn btn-danger ms-2': 'btn btn-outline-primary ms-2'> Home</button>
    </li>
    <li>
      <button (click)="gotoProducts()" [ngClass]="currentRoute=='products'? 'btn btn-danger ms-2': 'btn btn-outline-primary ms-2'> Products</button>
    </li>
  </ul>
</div>
<router-outlet>

```

Lorsqu'on choisi home le bouton reste en rouge et même chose pour products



Home Component



ID	Name	Price	
1	Computer	4300	<button>Delete</button>
2	Printer	3255	<button>Delete</button>
3	Smartphone	2100	<button>Delete</button>
4	Mouse	111	<button>Delete</button>

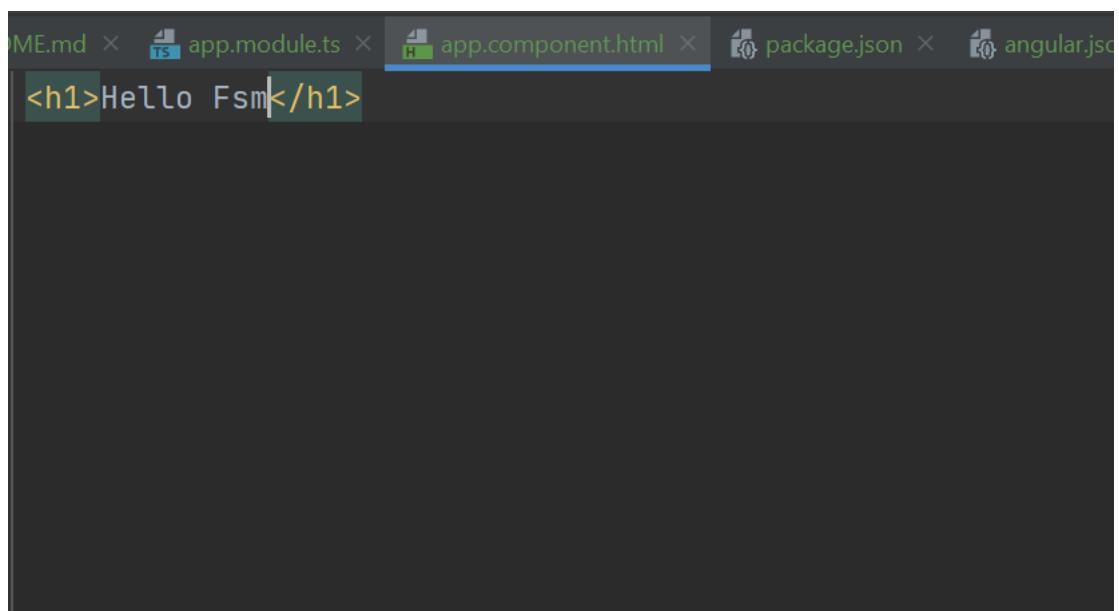
Deuxième Partie :

Partie 1 :

En utilisant la commande ng new fsm-app et on crée le nouveau projet du deuxième partie

```
CREATE fsm-app/tsconfig.app.json (342 bytes)
CREATE fsm-app/tsconfig.spec.json (287 bytes)
CREATE fsm-app/server.ts (1759 bytes)
CREATE fsm-app/.vscode/extensions.json (134 bytes)
CREATE fsm-app/.vscode/launch.json (490 bytes)
CREATE fsm-app/.vscode/tasks.json (980 bytes)
CREATE fsm-app/src/main.ts (256 bytes)
CREATE fsm-app/src/favicon.ico (15086 bytes)
CREATE fsm-app/src/index.html (305 bytes)
CREATE fsm-app/src/styles.css (81 bytes)
CREATE fsm-app/src/main.server.ts (271 bytes)
CREATE fsm-app/src/app/app.component.html (20239 bytes)
CREATE fsm-app/src/app/app.component.spec.ts (948 bytes)
CREATE fsm-app/src/app/app.component.ts (316 bytes)
CREATE fsm-app/src/app/app.component.css (0 bytes)
CREATE fsm-app/src/app/app.config.ts (330 bytes)
CREATE fsm-app/src/app/app.routes.ts (80 bytes)
CREATE fsm-app/src/app/app.config.server.ts (361 bytes)
CREATE fsm-app/src/assets/.gitkeep (0 bytes)
✓ Packages installed successfully.
```

On a besoin de démarrer notre application



```
!ME.md × app.module.ts × app.component.html × package.json × angular.json
<h1>Hello Fsm</h1>
```

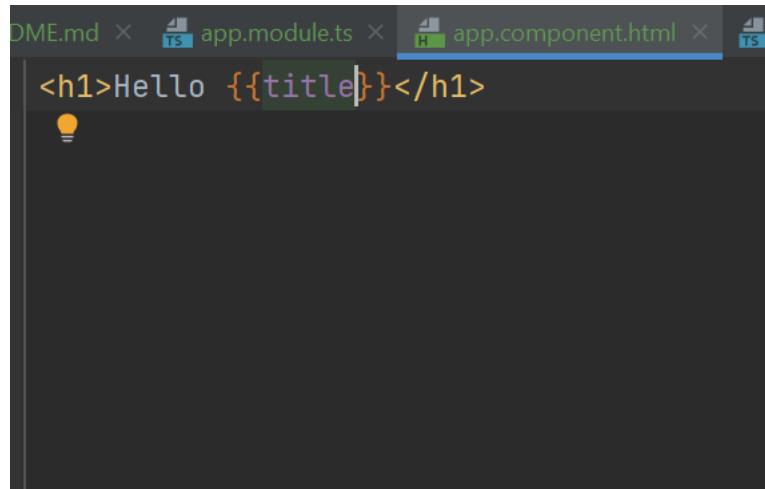


Hello Fsm

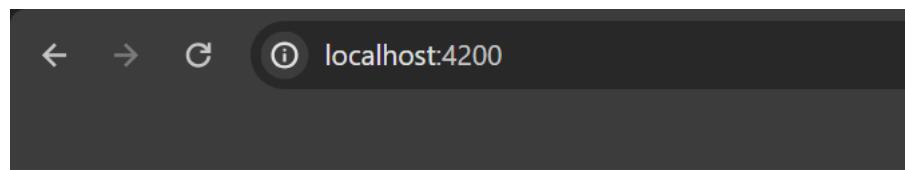
Dans la classe AppComponent, on déclare des variables comme title et si on veux l'afficher dans la partie html on va utiliser le mécanisme de data binding dans lequel angular utilise le directive et faire appel à la variable title (String interpolation)

```
!E.md × ! app.module.ts × ! app.component.html × ! app.component.ts ×
import { Component } from '@angular/core';

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



```
DME.md × app.module.ts × app.component.html ×
<h1>Hello {{title}}</h1>
```

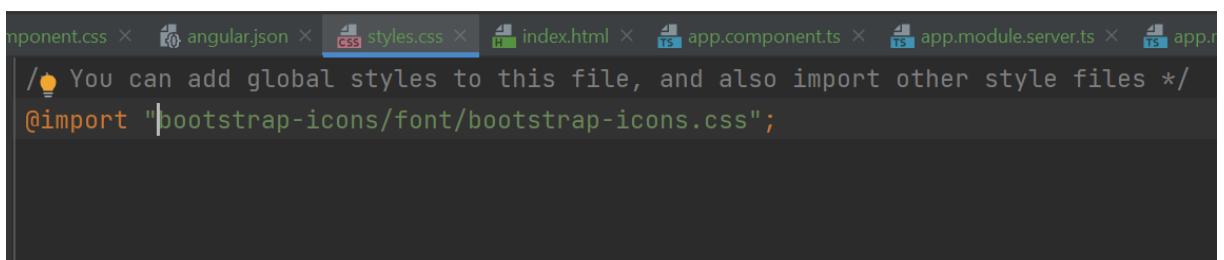


Hello fsm

On a besoin d'installer bootstrap et bootstrap-icons, pour intégrer bootstrap dans l'application

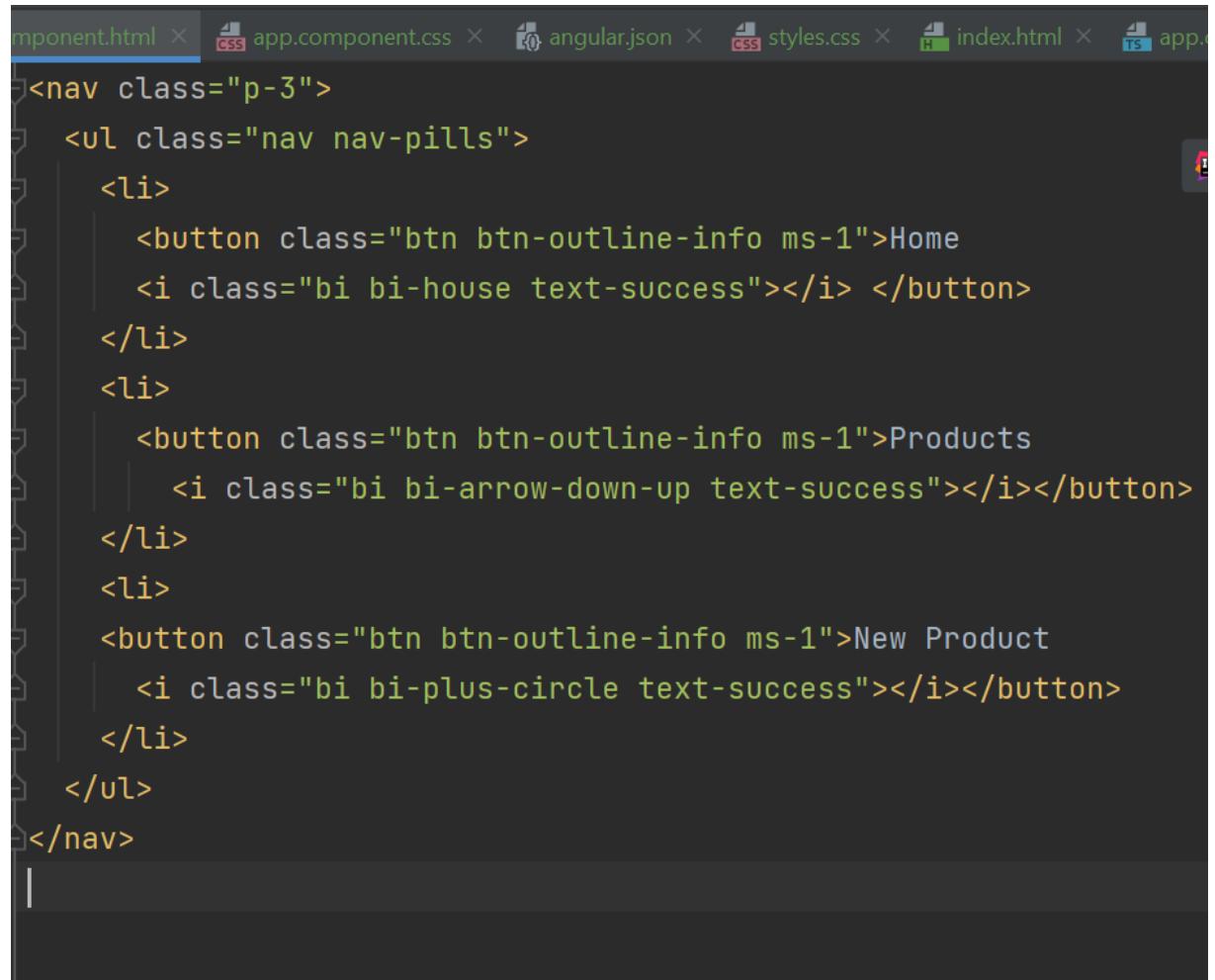


```
"styles": [
  "src/styles.css",
  "node_modules/bootstrap/dist/css/bootstrap.min.css"
],
"scripts": [
  "node_modules/bootstrap/dist/js/bootstrap.bundle.js"
],
```

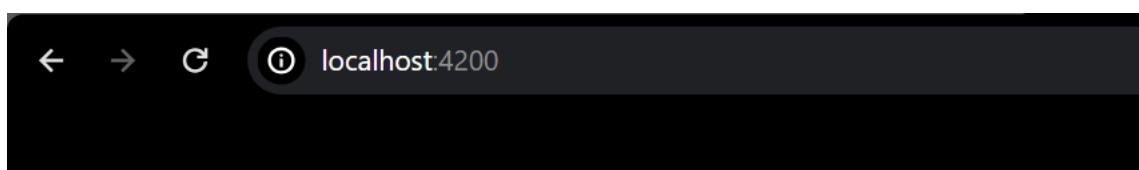


```
component.css × angular.json × styles.css × index.html × app.component.ts × app.module.server.ts × app.r
/ You can add global styles to this file, and also import other style files */
@import "bootstrap-icons/font/bootstrap-icons.css";
```

On crée la barre de navigation qui contient 3 boutons avec leurs icons dans AppComponent.html



```
<nav class="p-3">
  <ul class="nav nav-pills">
    <li>
      <button class="btn btn-outline-info ms-1">Home
        <i class="bi bi-house text-success"></i> </button>
    </li>
    <li>
      <button class="btn btn-outline-info ms-1">Products
        <i class="bi bi-arrow-down-up text-success"></i></button>
    </li>
    <li>
      <button class="btn btn-outline-info ms-1">New Product
        <i class="bi bi-plus-circle text-success"></i></button>
    </li>
  </ul>
</nav>
```



Après on va utiliser un élément router-outlet qui permet d'afficher le rendu après sa génération

On va créer 3 composants home, products et new-product

```
C:\Users\lenovo\angular-projects\fsm-app>ng g c home
CREATE src/app/home/home.component.html (20 bytes)
CREATE src/app/home/home.component.spec.ts (610 bytes)
CREATE src/app/home/home.component.ts (201 bytes)
CREATE src/app/home/home.component.css (0 bytes)
UPDATE src/app/app.module.ts (543 bytes)

C:\Users\lenovo\angular-projects\fsm-app>ng g c products
CREATE src/app/products/products.component.html (24 bytes)
CREATE src/app/products/products.component.spec.ts (638 bytes)
CREATE src/app/products/products.component.ts (217 bytes)
CREATE src/app/products/products.component.css (0 bytes)
UPDATE src/app/app.module.ts (635 bytes)

C:\Users\lenovo\angular-projects\fsm-app>ng g c new-product
CREATE src/app/new-product/new-product.component.html (27 bytes)
CREATE src/app/new-product/new-product.component.spec.ts (653 bytes)
CREATE src/app/new-product/new-product.component.ts (228 bytes)
CREATE src/app/new-product/new-product.component.css (0 bytes)
UPDATE src/app/app.module.ts (737 bytes)
```

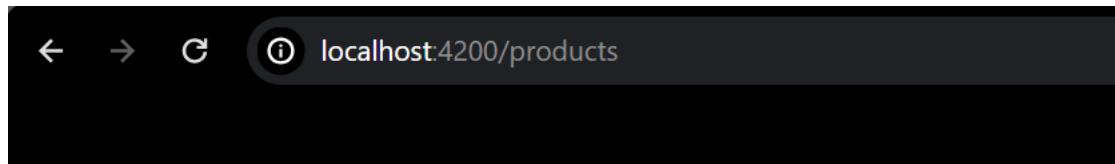
On importe automatiquement les routes avec leurs composants dans app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { ProductsComponent } from './products/products.component';
import { NewProductComponent } from './new-product/new-product.component';

const routes: Routes = [
  {path : "home", component : HomeComponent},
  {path : "products", component : ProductsComponent},
  {path : "newProduct", component : NewProductComponent}
];

```

```
  <ul class="nav nav-pills">
    <li>
      <button routerLink="/home" class="btn btn-outline-success ms-1">Home
        <i class="bi bi-house text-success"></i> </button>
    </li>
    <li>
      <button routerLink="/products" class="btn btn-outline-success ms-1">Products
        <i class="bi bi-arrow-down-up text-success"></i></button>
    </li>
    <li>
      <button routerLink="/newProduct" class="btn btn-outline-success ms-1">New Product
        <i class="bi bi-plus-circle text-success"></i></button>
    </li>
  </ul>
```



Home Products New Product

products works!

On crée le contenu html de chaque composant

```
product.html ×  home.component.html ×  new-product.component.html
<div class="p-3">
  <div class="card">
    <div class="card-body">
      <h3>Products Component</h3>

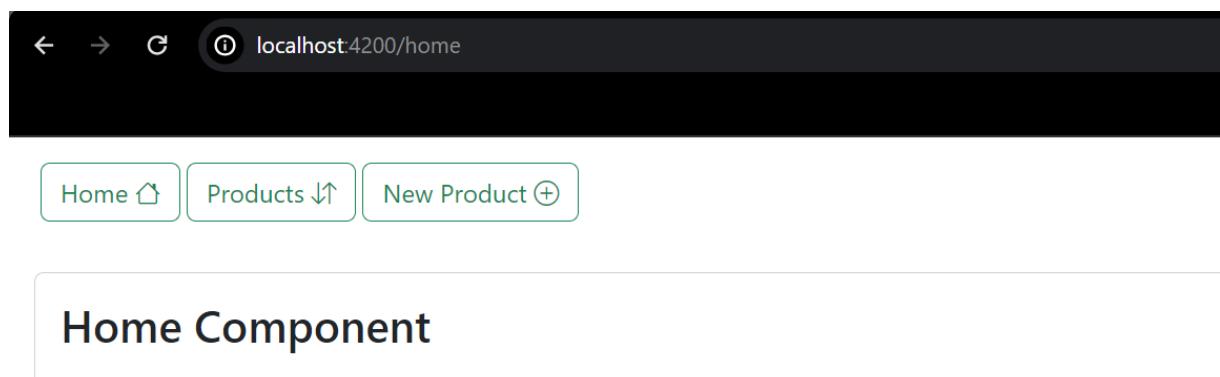
    </div>
  </div>

</div>
```

```
><div class="p-3">
>  <div class="card">
>    <div class="card-body">
>      <h3>Home Component</h3>
>
>    </div>
>  </div>
>
></div>
```

```
><div class="p-3">
>  <div class="card">
>    <div class="card-body">
>      <h3>New Product Component</h3>
>
>    </div>
>  </div>
>
></div>
|
```

On les affiche dans une carte



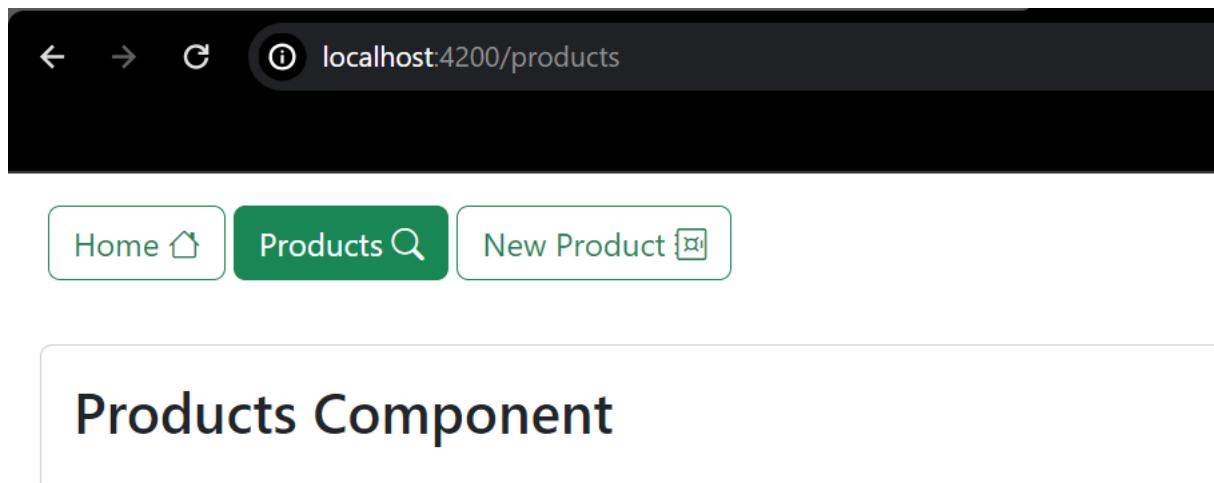
On va faire une boucle pour générer tous les boutons au lieu de faire copier coller

Dans le composant AppComponent, on crée une liste d'actions et on indique l'action courante

```
export class AppComponent {  
  actions : Array<any> =[  
    {title : "Home", "route" : "/home", icon : "house"},  
    {title : "Products", "route" : "/products", icon : "search"},  
    {title : "New Product", "route" : "/newProduct", icon : "safe"},  
  ];  
  currentAction:any;  
  
  setCurrentAction(action: any) {  
    this.currentAction= action;  
  }  
}
```

On faire la boucle

```
<nav class="p-3">  
  <ul class="nav nav-pills">  
    <li *ngFor="let action of actions">  
      <button  
        (click)="setCurrentAction(action)"  
        routerLink="{{action.route}}"  
        [class]={{action==  
          currentAction?'btn btn-success ms-1': 'btn btn-outline-success ms-1'}}>  
        {{action.title}}  
        <i class="bi bi-{{action.icon}}"></i>  
      </button>  
    </li>  
  </ul>  
</nav>
```



Maintenant nous voulons afficher la liste des produits, pour se faire on va travailler avec data binding.

Alors data binding c'est la liaison entre les données qui sont stockées dans la classe des composants avec ce qui est affichée dans la vue.

Le data binding peut se faire dans un sens, dans l'autre ou dans les deux, c'est-à-dire si par exemple nous avons une variable data et nous voulons l'afficher dans la partie vue, on utilise ce qu'on appelle String interpolation (la variable va automatiquement se mettre à jour au niveau de la partie vue).

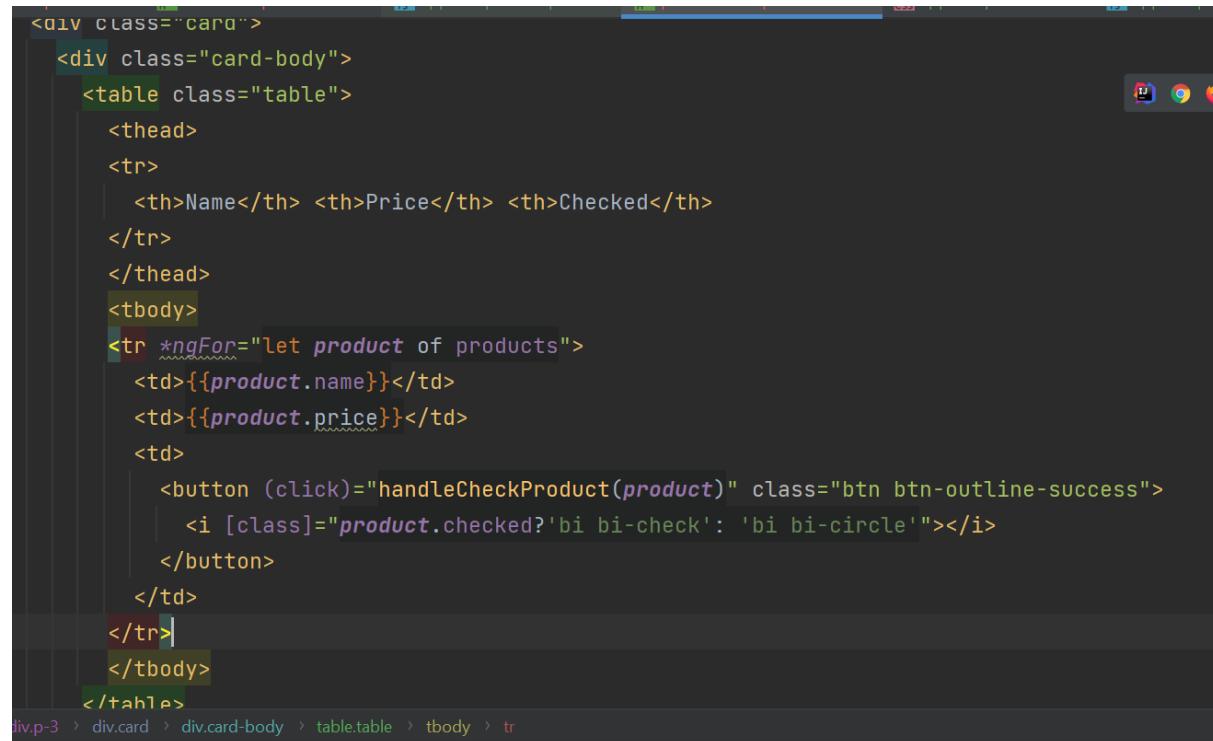
```

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

@Component({
  selector: 'app-products',
  templateUrl: './products.component.html',
  styleUrls: ['./products.component.css'
})
export class ProductsComponent {
  products: Array<any>= [
    {id: 1, name: "Computer", price: 4500, checked: false},
    {id: 1, name: "Printer", price: 4500, checked: true},
    {id: 1, name: "Phone", price: 4500, checked: false}
  ];

  handleCheckProduct(product : any) {
    product.checked =! product.checked;
  }
}

```



```

<div class="card">
  <div class="card-body">
    <table class="table">
      <thead>
        <tr>
          <th>Name</th> <th>Price</th> <th>Checked</th>
        </tr>
      </thead>
      <tbody>
        <tr *ngFor="let product of products">
          <td>{{product.name}}</td>
          <td>{{product.price}}</td>
          <td>
            <button (click)="handleCheckProduct(product)" class="btn btn-outline-success">
              <i [class]="'product.checked?'bi bi-check': 'bi bi-circle'"></i>
            </button>
          </td>
        </tr>
      </tbody>
    </table>
  </div>

```

Name	Price	Checked
Computer	4500	<input type="checkbox"/>
Printer	4500	<input checked="" type="checkbox"/>
Phone	4500	<input type="checkbox"/>

On a besoin d'un backend dans lequel on va gérer les produits en utilisant json-server qui est une application node.js qui permet de mettre en place une rest api en local qui marche bien pour toutes les méthodes

Premièrement, on installe json-server

```
C:\Users\lenovo\angular-projects\fsm-app>npm install json-server

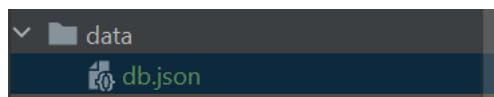
added 40 packages, and audited 967 packages in 4s

132 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Et maintenant pour gérer la base de données, cette dernière elle faut être un fichier json

Dans notre projet on va créer un dossier data à l'intérieur on crée un fichier db.json



Db.json est un objet de json qui contient des collections (un tableau des produits)

On démarre json

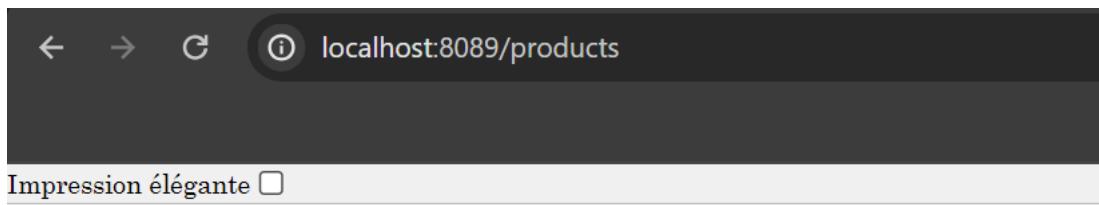
```
C:\Users\lenovo\angular-projects\fsm-app>json-server -w data/db.json -p 8089
--watch/-w can be omitted, JSON Server 1+ watches for file changes by default
JSON Server started on PORT :8089
Press CTRL-C to stop
Watching data/db.json...

♡( ˘◡˘ )

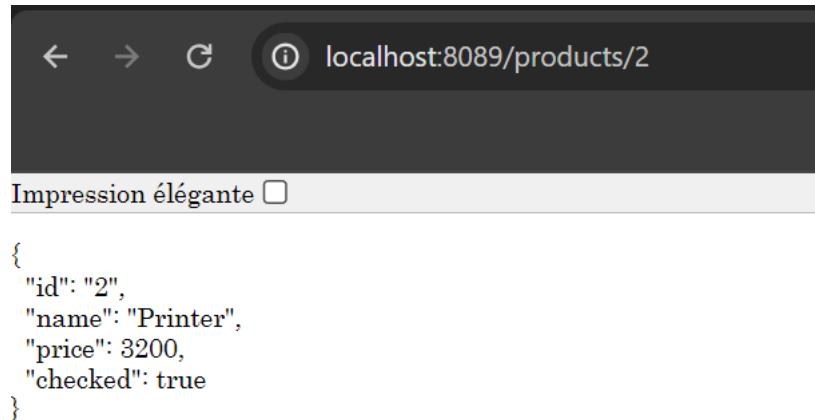
Index:
http://localhost:8089/

Static files:
Serving ./public directory if it exists

Endpoints:
http://localhost:8089/products
```



```
[
  {
    "id": "1",
    "name": "Computer",
    "price": 5600,
    "checked": false
  },
  {
    "id": "2",
    "name": "Printer",
    "price": 3200,
    "checked": true
  },
  {
    "id": "3",
    "name": "Smart Phone",
    "price": 600,
    "checked": false
  }
]
```



A screenshot of a browser window showing the URL `localhost:8089/products/2`. The page content is a JSON object:

```
{  
  "id": "2",  
  "name": "Printer",  
  "price": 3200,  
  "checked": true  
}
```

Au démarrage de notre composant, on a besoin d'envoyer une requête http vers le backend pour chercher la liste des produits, on utilise le module `HttpClientModule` et on l'importe premièrement

```
import { AppComponent } from './app.component';  
import { HomeComponent } from './home/home.component';  
import { ProductsComponent } from './products/products.component';  
import { NewProductComponent } from './new-product/new-product.component';  
import {HttpClientModule} from "@angular/common/http";  
  
@NgModule({  
  declarations: [  
    AppComponent,  
    HomeComponent,  
    ProductsComponent,  
    NewProductComponent  
  ],  
  imports: [  
    BrowserModule,  
    AppRoutingModule,  
    HttpClientModule  
  ],  
  providers: [  
    provideClientHydration()  
]
```

Et deuxièmement on fait l'injection des dépendances et après mon composant s'affiche il m'affiche une liste des produits

J'ai envoyé la requête et je n'attends pas le retour (asynchrone)

```

        styleUrl: './products.component.css'
    })
}

export class ProductsComponent implements OnInit{
    products : Array<any>=[];
    constructor(private http: HttpClient) {
        this.http=http;
    }

    ngOnInit() {
        this.http.get<Array<any>>( url: "http://localhost:8089/products")
            .subscribe( observerOrNext: {
                next : data => {
                    this.products= data
                },
                error : err => {
                    console.log(err);
                }
            })
    }
}

ProductsComponent

```

On ajoute un autre produit dans la base de données

```

{
    "products": [
        {
            "id": 1 , "name": "Computer", "price": 5600, "checked": false
        },
        {
            "id": 2, "name": "Printer", "price": 3200, "checked": true
        },
        {
            "id": 3, "name": "Smart Phone", "price": 600, "checked": false
        },
        {
            "id": 3, "name": "Smart Phone", "price": 600, "checked": false
        }
    ]
}

```

On a comme résultat

Name	Price	Checked
Computer	5600	<input type="checkbox"/>
Printer	3200	<input checked="" type="checkbox"/>
Smart Phone	600	<input type="checkbox"/>
Smart Phone	600	<input type="checkbox"/>

Lorsqu'on fait check product il faut qu'il soit modifier dans le niveau backend et on veux seulement changer l'attribut checked donc on utilise patch pour envoyer une requête http vers l'url indiquée

```

handleCheckProduct(product : any) {
    this.http.patch<any>( url: `http://localhost:8089/products/${product.id}` ,
        body: {checked:product.checked}).subscribe( observerOrNext: {
            next : updatedProduct => {
                product.checked =! product.checked;
                //this.getProducts();
            }
        })
}

```

ProductsComponent > handleCheckProduct() > next()

On a comme résultat

Name	Price	Checked
Computer	5600	<input type="checkbox"/>
Printer	3200	<input checked="" type="checkbox"/>
Smart Phone	600	<input type="checkbox"/>
Smart Phone	600	<input type="checkbox"/>

On va structurer notre code et pour cela on va introduire la notion de service et pour créer une service avec angular par

```
C:\Users\lenovo\angular-projects\fsm-app>ng g s services/product
CREATE src/app/services/product.service.spec.ts (378 bytes)
CREATE src/app/services/product.service.ts (145 bytes)
```

Dans notre service on va créer des méthodes

```
import { Injectable } from '@angular/core';
import {HttpClient} from "@angular/common/http";
import {Observable} from "rxjs";

@Injectable({
  providedIn: 'root'
})
export class ProductService {

  constructor(private http: HttpClient) { }

  public getProducts(): Observable<any> {
    return this.http.get<Array<any>>(`url: "http://localhost:8089/products"`);
  }

  public checkProduct(product: any): Observable<any> {
    return this.http.patch<any>(`url: "http://localhost:8089/products/${product.id}"`,
      body: {checked:product.checked});
  }
}
```

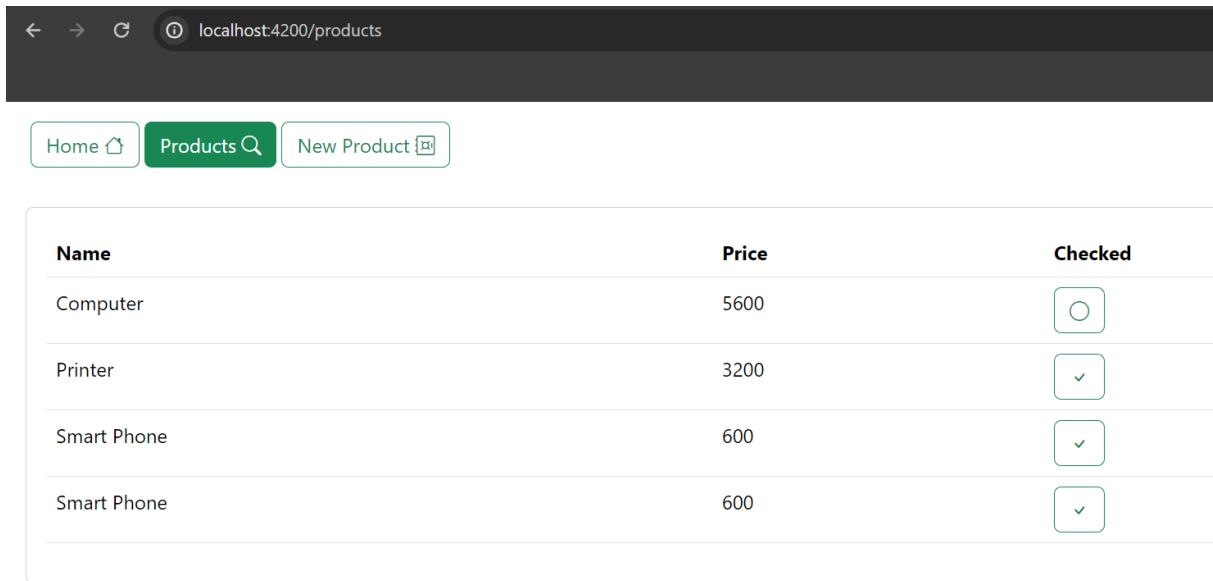
ProductService > constructor() > http

Externally added files can be added to Git

Dans le constructeur de productsComponent on change le module HttpClient par
ProductService

```
@Component({
  selector: 'app-products',
  templateUrl: './products.component.html',
  styleUrls: ['./products.component.css'
})
export class ProductsComponent implements OnInit{
  products : Array<any>=[];
  constructor(private productService: ProductService) {
    this.productService=productService;
  }
}
```

On démarre l'application



Name	Price	Checked
Computer	5600	<input type="checkbox"/>
Printer	3200	<input checked="" type="checkbox"/>
Smart Phone	600	<input checked="" type="checkbox"/>
Smart Phone	600	<input checked="" type="checkbox"/>

Il n'est pas obligatoire de déclarer le module ProductService dans providers parce qu'on a l'attribut providedIn : 'root' c'est-à-dire que cet service est disponible dans la racine du projet donc automatiquement dès le démarrage c'est un service qui va être instancier et on peut l'injecter dans n'importe quel composant de l'application de toutes les modules

Nous avons besoin de créer un modèle dans lequel on crée une interface et on déclare les types de données qu'on manipule dans le frontend

```
export interface Product {  
    id : number,  
    name : String,  
    price: number,  
    checked : boolean  
}  
|
```

Au lieu de retourner des objets observable <any> on retourne une liste de produits et on spécifie aussi chaque produit

```
import {ProductService} from "../services/product.service";  
import {Product} from "../model/product.model";  
  
@Component({  
    selector: 'app-products',  
    templateUrl: './products.component.html',  
    styleUrls: ['./products.component.css'  
})  
export class ProductsComponent implements OnInit{  
    products : Array<Product>=[];  
    constructor(private productService: ProductService) {  
        this.productService=productService;  
    }  
  
    ngOnInit() {  
        this.getProducts();  
    }  
}
```

The screenshot shows a code editor with several tabs at the top: cts.component.spec.ts, product.service.spec.ts, product.service.ts, app.module.ts, and products.component.ts. The products.component.ts tab is active. The code in the editor is:

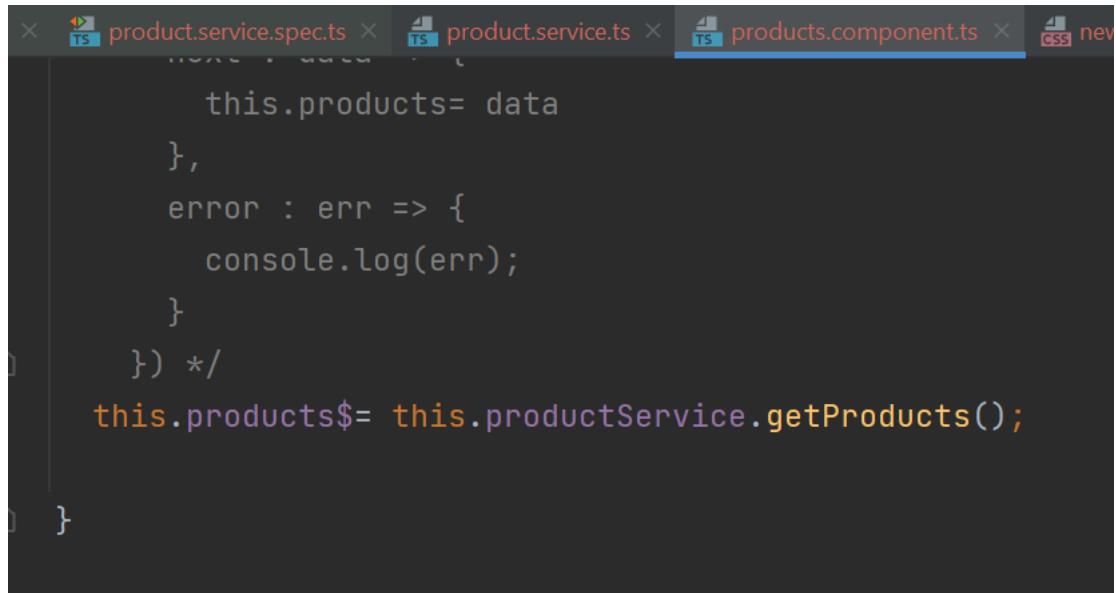
```
    }
  })
}

handleCheckProduct(product : Product) {
  this.productService.checkProduct(product).subscribe( observerOrNext: {
    next : updatedProduct => {
      product.checked =! product.checked;
      //this.getProducts();
    }
  })
}
```

On utilise une autre méthode pour la fonction getProducts()

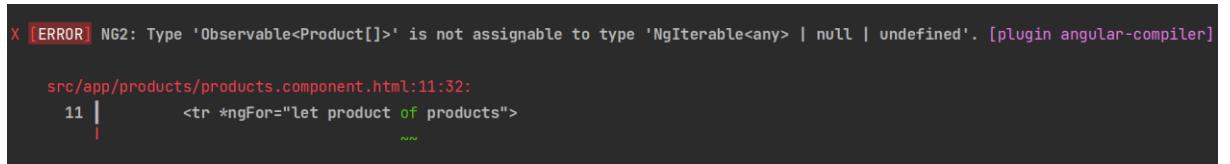
Si on une variable de type Observable par convention on utilise un variable qui se termine par \$ et ça ne marche pas

```
export class ProductsComponent implements OnInit{
  products$! : Observable<Array<Product>>;
  constructor(private productService: ProductService) {
    this.productService=productService;
  }
}
```



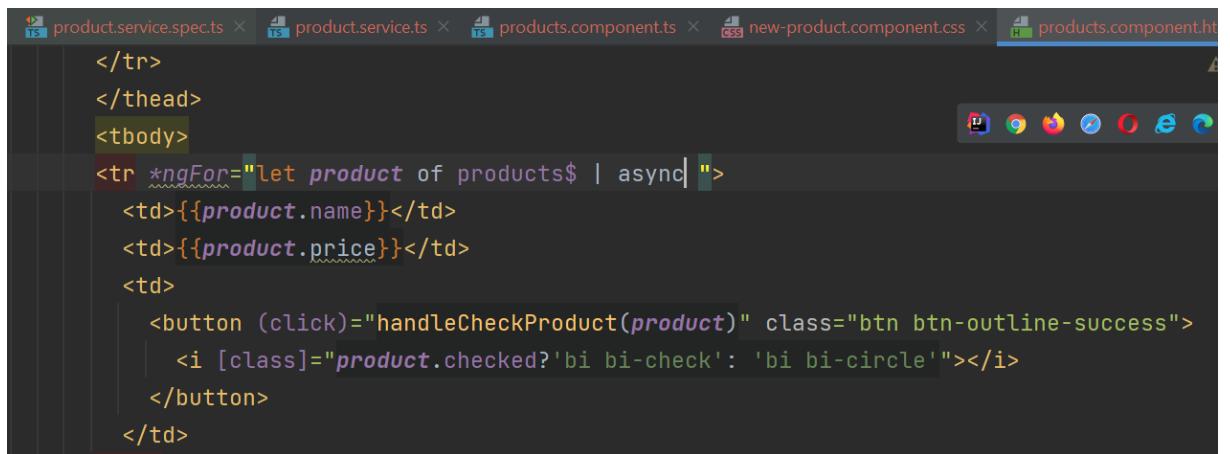
```
        this.products= data
    },
    error : err => {
        console.log(err);
    }
}) */
this.products$= this.productService.getProducts();

}
}
```



```
x [ERROR] NG2: Type 'Observable<Product[]>' is not assignable to type 'NgIterable<any> | null | undefined'. [plugin angular-compiler]
src/app/products/products.component.html:11:32:
11 |         <tr *ngFor="let product of products">
|             
```

Pour résoudre ce problème on ajoute async a products pour faire subscribe automatiquement et il parcourt la liste



```
</tr>
</thead>
<tbody>
<tr *ngFor="let product of products$ | async" >
    <td>{{product.name}}</td>
    <td>{{product.price}}</td>
    <td>
        <button (click)="handleCheckProduct(product)" class="btn btn-outline-success">
            <i [class]="product.checked?'bi bi-check': 'bi bi-circle'"></i>
        </button>
    </td>

```

localhost:4200/products

Name	Price	Checked
Computer	5600	<input checked="" type="checkbox"/>
Printer	3200	<input checked="" type="checkbox"/>
Smart Phone	600	<input type="checkbox"/>
Smart Phone	600	<input type="checkbox"/>

On ajoute un bouton pour supprimer un produit

```
</td>
<td>
  <button (click)="handleDelete(product)" class="btn btn-outline-danger">
    <i class="bi bi-trash"></i>
  </button>
</td>
</tr>
```

On déclare la fonction pour la suppression

```
public deleteProduct(product: Product) {
  return this.http.delete<any>( url: `http://localhost:8089/products/${product.id}` );
}

}

handleDelete(product : Product) {
  this.productService.deleteProduct(product).subscribe( observerOrNext: {
    next:value => {
      this.getProducts();
    }
  }
}
```

On a comme résultat

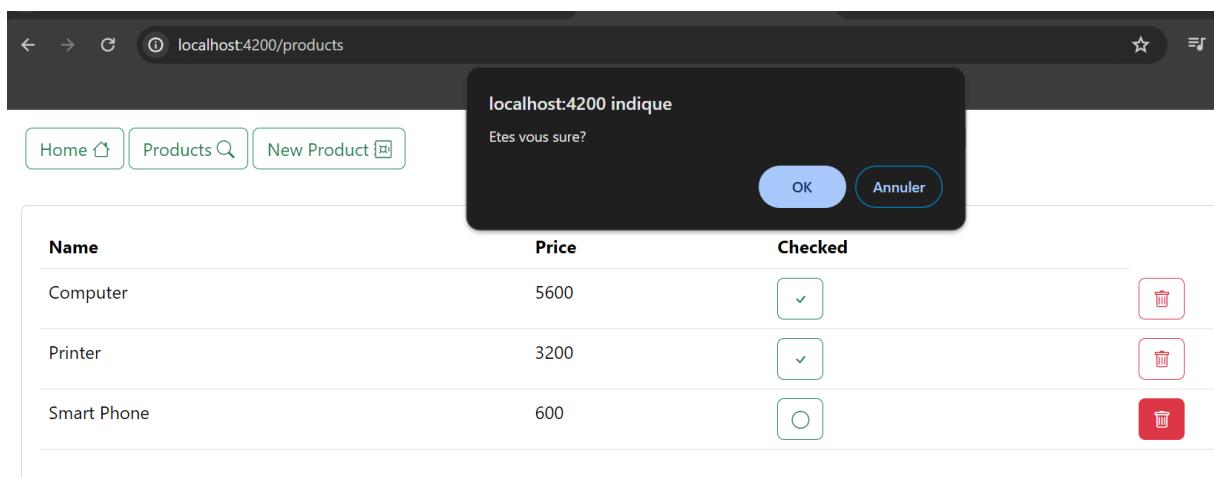
Name	Price	Checked	
Computer	5600	<input checked="" type="checkbox"/>	
Printer	3200	<input checked="" type="checkbox"/>	
Smart Phone	600	<input type="checkbox"/>	

C'est acceptable mais on peut faire mieux c'est supprimer au niveau du frontend

```
export class ProductsComponent implements OnInit{
  products : Array<Product>= [];
  constructor(private productService: ProductService) {
    this.productService=productService;
  }
  ngOnInit() {
    this.getProducts();
  }
  getProducts() {
    this.productService.getProducts()
      .subscribe( observerOrNext: {
        next : data => {
          this.products= data
        },
        error : err => {
          console.log(err);
        }
      })
    //this.products$= this.productService.getProducts();
  }
}
```

```
        handleDelete(product : Product) {
            if(confirm("Etes vous sure?"))
                this.productService.deleteProduct(product).subscribe( observerOrNext: {
                    next:value => {
                        //this.getProducts();
                        this.products= this.products.filter(p => p.id != product.id);
                    }
                }
            )
        }
    }
```

On a le résultat



On va créer un formulaire pour ajouter des produits

Tout d'abord on importe un module ReactiveFormsModule

```
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    ReactiveFormsModule
  ],
  providers: [
    provideClientHydration()
```

Dans le composant New Product, on déclare le constructeur et redéfinir la méthode ngOnInit()

```
  styleUrls: ['./new-product.component.css'
])
export class NewProductComponent implements OnInit{

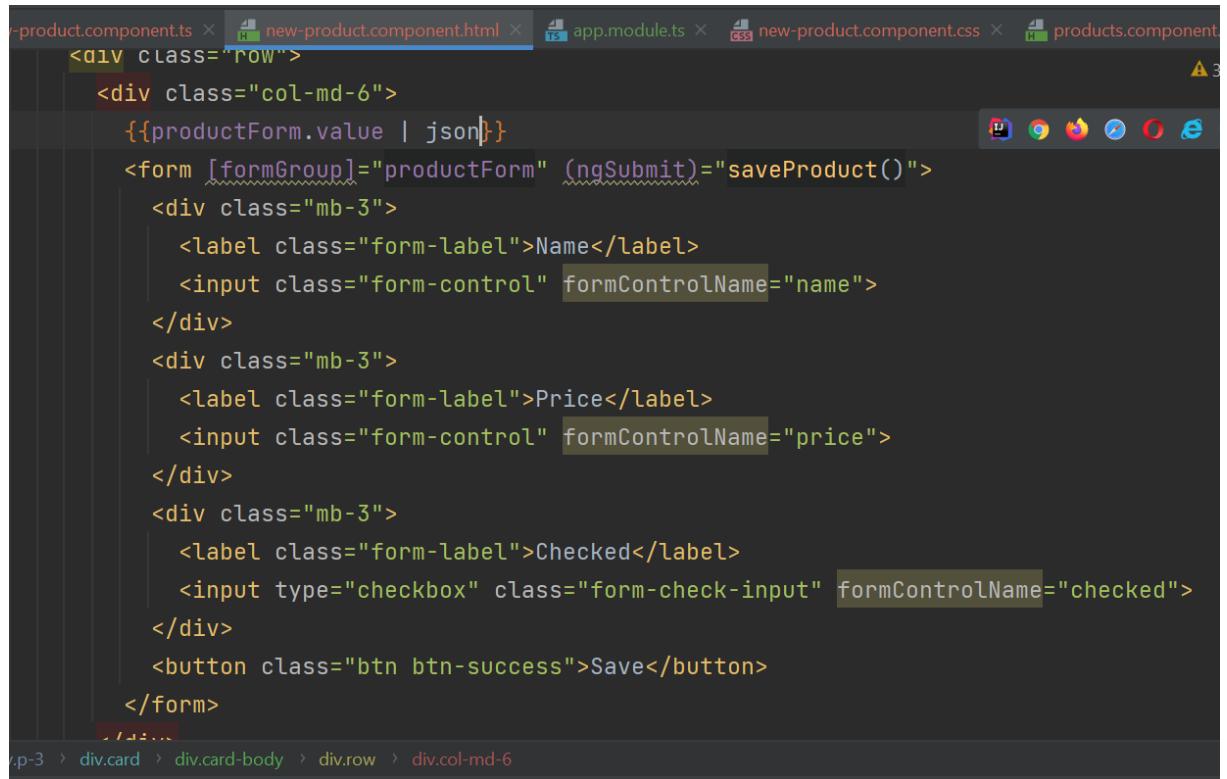
  public productForm!: FormGroup;

  constructor(private fb: FormBuilder) {

  }

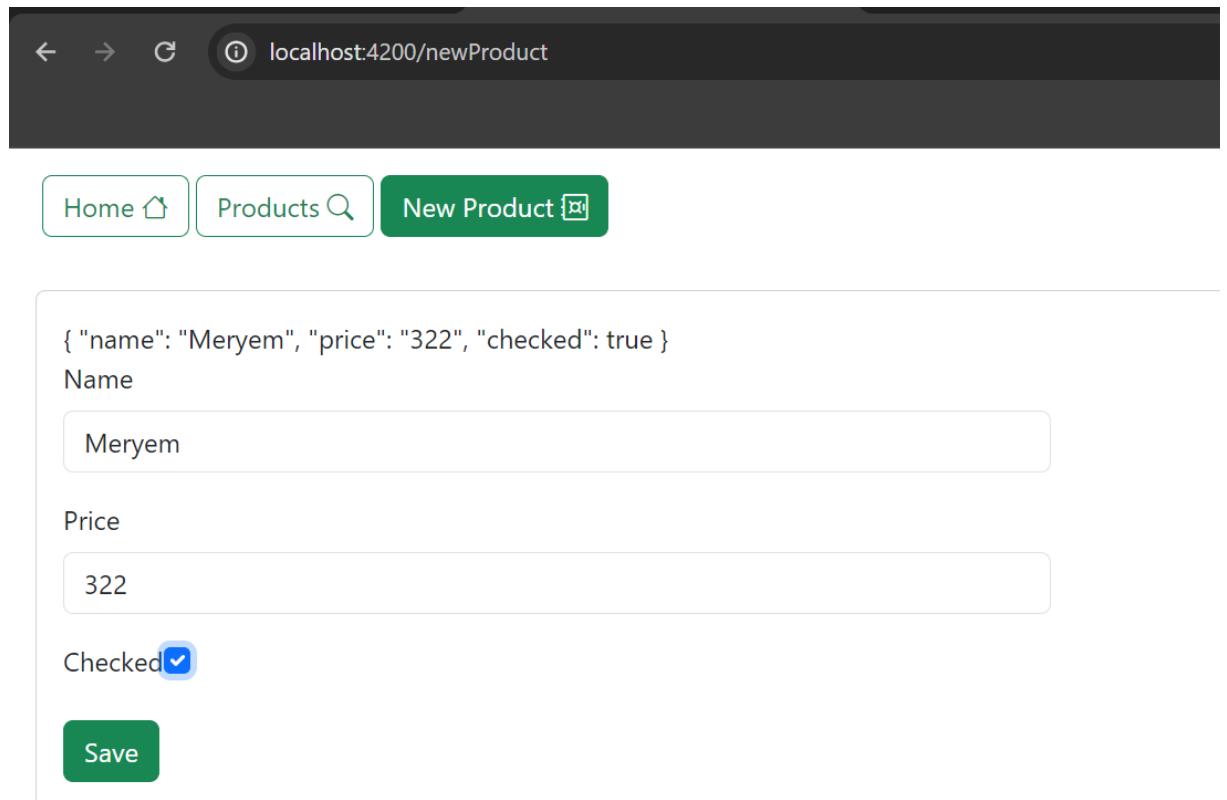
  ngOnInit(): void {
    this.productForm=this.fb.group( controls: {
      name: this.fb.control( formState: '' ),
      price: this.fb.control( formState: '' ),
      checked: this.fb.control( formState: false ),
    });
  }
}
```

Maintenant, on crée le formulaire dans le composant html



```
<div class="row">
  <div class="col-md-6">
    {{productForm.value | json}}
    <form [formGroup]="productForm" (ngSubmit)="saveProduct()">
      <div class="mb-3">
        <label class="form-label">Name</label>
        <input class="form-control" formControlName="name">
      </div>
      <div class="mb-3">
        <label class="form-label">Price</label>
        <input class="form-control" formControlName="price">
      </div>
      <div class="mb-3">
        <label class="form-label">Checked</label>
        <input type="checkbox" class="form-check-input" formControlName="checked">
      </div>
      <button class="btn btn-success">Save</button>
    </form>
  </div>
</div>
```

On démarre l'application



{ "name": "Meryem", "price": "322", "checked": true }

Name

Price

Checked

Save

On récupère le formulaire par la fonction SaveProduct()

```

    saveProduct(product: Product): Observable<Product>{
      return this.http.post<Product>( url: `http://localhost:8089/products`, product);
    }
}

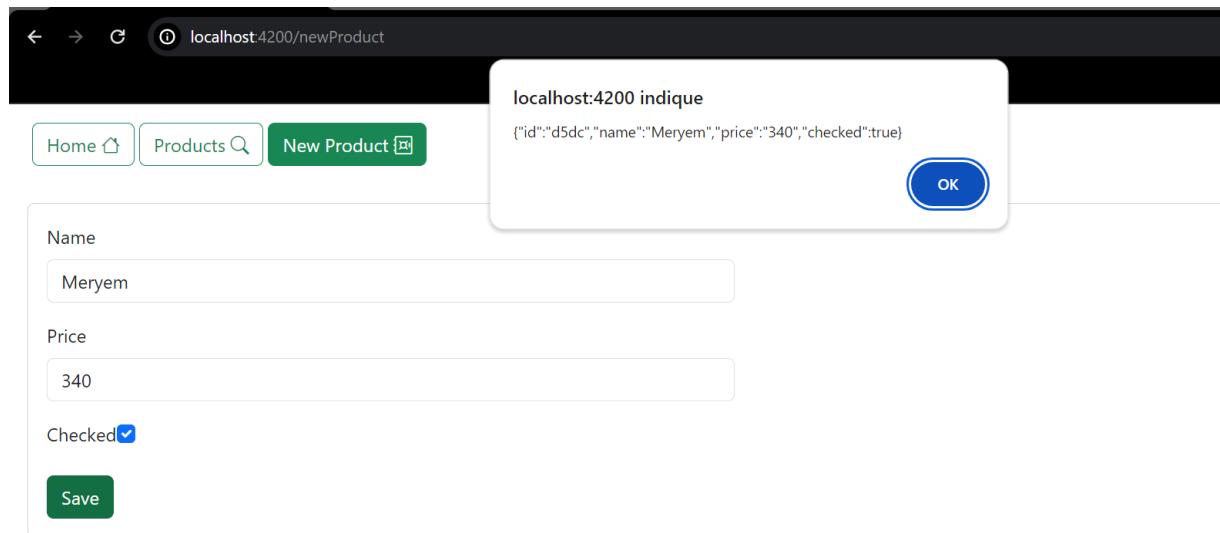
```

```

    saveProduct() {
      let product: Product= this.productForm.value;
      this.productService.saveProduct(product).subscribe( observerOrNext: {
        next : data| => {
          alert(JSON.stringify(data));
        },
        error : err => {
          console.log(err);
        }
      });
    }
}

```

On démarre l'application



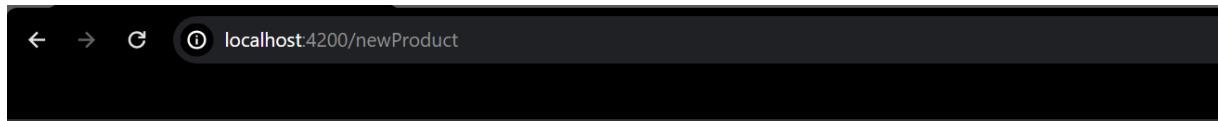
On ajoute disabled qui nous a permis de désactiver le bouton save jusqu'au remplir le formulaire

```

      <input type="checkbox" class="form-check-input" formControlName="checked">
    </div>
    <button [disabled]="productForm.invalid" class="btn btn-success">Save</button>
  </form>
</div>

</div>

```



Home Products New Product

Name

Price

Checked

Save

On ajoute une barre de recherche pour chercher des produits

```
<div class="p-3">
  <div class="card">
    <div class="card-body">
      <div class="card-body">

        <input class="p-1" type="text" [(ngModel)]="keyword">
        <button (click)="searchProducts()" class="btn btn-outline-success">

          <i class="bi bi-search"></i>
        </button>

      </div>
    </div>
  </div>
```

```
public searchProducts(keyword: string): Observable<Array<Product>> {
  return this.http.get<Array<Product>>(`http://localhost:8089/products?name=${keyword}`);
}
```

```
searchProducts() {
  this.productService.searchProducts(this.keyword).subscribe(observerOrNext: {
    next: value => { this.products = value; }
  })
}
```

On démarre l'application

Name	Price	Checked
Printer	3200	

Il nous reste la pagination, pour cela on va transmettre des paramètres à nos méthodes dans le service

```

    providedIn: 'root'
})
export class ProductService {

  constructor(private http: HttpClient) { }

  public getProducts(page: number=1, size:number=4): Observable<Array<Product>> {
    return this.http.get<Array<Product>>(`http://localhost:8089/products?page=${page}&_limit=${size}`);
  }
  public checkProduct(product: Product): Observable<Product> {
    return this.http.patch<Product>(`http://localhost:8089/products/${product.id}`, {
      body: {checked:product.checked}
    });
  }
  public deleteProduct(product: Product) {
    return this.http.delete<any>(`http://localhost:8089/products/${product.id}`);
  }
  saveProduct(product: Product): Observable<Product> {
  }

  getProducts() {
    this.productService.getProducts( page: 1, size: 2)
      .subscribe( observerOrNext: {
        next : data => {
          this.products= data
        },
        error : error => {
        }
      })
  }
}

```

On a comme résultat

Name	Price	Checked
Computer	5600	<input checked="" type="checkbox"/>
Printer	3200	<input type="checkbox"/>

Donc json-server il gère la pagination en backend

Partie 2:

On essaie de compléter la pagination, pour cela on doit lire les headers en utilisant observe 'response'

```
export class ProductService {  
  constructor(private http: HttpClient) {}  
  
  public getProducts(page: number=1, size:number=4) {  
    return this.http.get(`http://localhost:8089/products?_page=${page}&_limit=${size}`, {observe: 'response'});  
  }  
}
```

Maintenant on revient vers notre composant, alors lorsqu'on appelle la méthode getProducts on retourne un objet responsable ça veut dire un objet de type http Response, et forcément un petit souci se trouve, pour le sauver on met resp.body as Product[] c'est comme on dit que c'est un tableau de produits, la deuxième des choses c'est qu'on doit récupérer totalCount :

```

    //, current page, number ...
constructor(private productService: ProductService) {

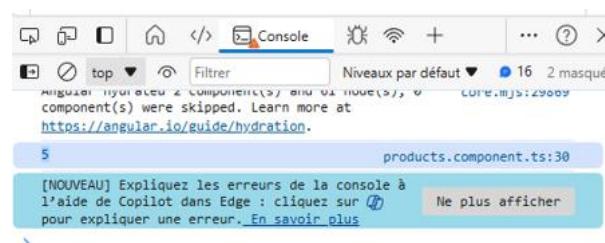
}

ngOnInit() {
  this.getProducts();
}

getProducts() {
  this.productService.getProducts({ page: 1, size: 2 })
    .subscribe(observerOrNext: {
      next : (resp: HttpResponse<Object>) => {
        this.products = resp.body as Product[];
        let totalProducts: number = parseInt(resp.headers.get('x-total-count')!) ;

        console.log(totalProducts);
      },
      error : err => {
        console.log(err);
      }
    })
}

```



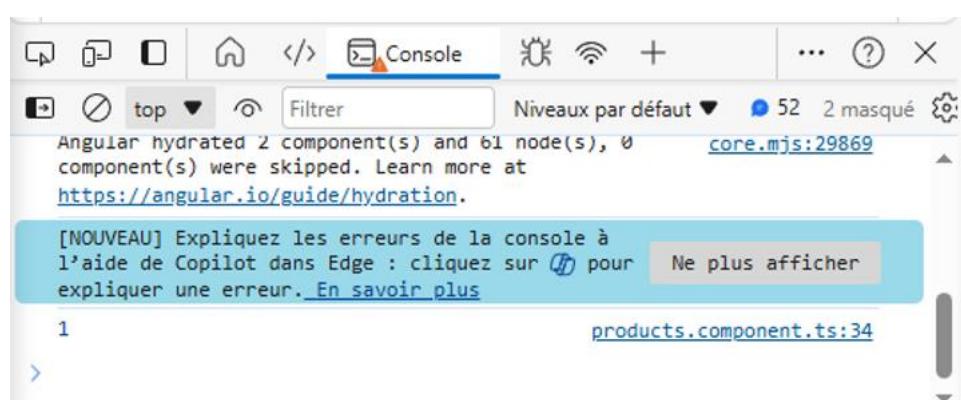
Maintenant qu'on a vu le nombre de page récupéré est 5, on passe pour calculer le nombre total des pages, donc nous avons besoin dans notre composant de déclarer les trois variables totalPages, pageSize et currentPage :

```

pageSize: number=3;
currentPage: number = 1;
constructor(private productService: ProductService) {

}
ngOnInit() {
  this.getProducts();
}
getProducts() {
  this.productService.getProducts(this.currentPage, this.pageSize)
    .subscribe( observerOrNext: {
      next : (resp : HttpResponse<Object>) => {
        this.products = resp.body as Product[];
        let totalProducts: number=parseInt(resp.headers.get('x-total-count')!)!;
        this.totalPages=Math.floor( x: totalProducts / this.pageSize);
        if(totalProducts % this.pageSize != 0) {
          this.totalPages= this.totalPages + 1;
        }
      }
    })
}

```



On affiche les pages après le tableau et tous ce qui reste c'est qu'il nous faut colorier ou bien sélectionner la page courante :

Name	Price	Checked	
Computer	5600	<input checked="" type="checkbox"/>	
New Printer	3200	<input checked="" type="checkbox"/>	
Smart	600	<input checked="" type="checkbox"/>	

On peut le faire aussi en utilisant [class] au lieu de [ngClass] :

```

</table>
<ul class="nav nav-pills">
  <li *ngFor="let page of getPagesArray()" >
    <button (click)="handleGotoPage(page)" 
      [ngClass]="currentPage==(i+1)?'btn btn-success m-1':'btn btn-outline-success m-1'" 
      class="btn m-1">
      {{ page }}
    </button>
  </li>
</ul>

```

Maintenant il nous reste un petit problème, c'est que tous les produits s'affiche dans la même page, parce qu'on n'a pas initialisé le pageSize et currentPage.

Alors on va régler le problème, on va envoyer le currentPage et pageSize en paramètres de notre méthode searchProducts.

L'autre problème qu'on trouve c'est de refaire ce que nous avons déjà fait dans la méthode getProducts, (c'est le calcul des pages totales), pour cela on va changer le nom de la méthode getProducts en searchProducts et en envoie le variable keyword dans leur paramètres.

```

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help fsm-app - products.component.ts
fsm-app src app products products.component.ts
Project fsm-app C:\Users\21263\ideaProjects\Asmaea...
  > angular
  > .idea
  > vscode
  > data db.json
  > node_modules library root
  > src
    > app
      > home
        home.component.css
        home.component.html
        home.component.spec.ts
        home.component.ts
      > model
        product.model.ts
      > new-product
        new-product.component.css
        new-product.component.html
        new-product.component.spec.ts
        new-product.component.ts
      > products
        products.component.css
        products.component.html
        products.component.spec.ts
        products.component.ts
      > services
        productService.spec.ts
        productService.ts
        app.component.css
  Favorites
  Structure
  Event Log
  TODO Problems Terminal Profiler
  IntelliJ IDEA 2021.3 available // Update... (39 minutes ago)
  TypeScript 41:9 (552 chars, 14 line breaks) CRLF UTF-8 2 spaces*
  
```

The screenshot shows the IntelliJ IDEA interface with the fsm-app project open. The products.component.ts file is being edited. The code is as follows:

```

21 }
22
23 ngOnInit() {
24   this.searchProducts();
25 }
26
27 searchProducts(){
28   this.productService.searchProducts(this.keyword, this.currentPage, this.pageSize)
29     .subscribe( observerOrNext: {
30       next : (resp : HttpResponse<Object>) => {
31         this.products = resp.body as Product[];
32         let totalProducts : number = parseInt(resp.headers.get('x-total-count')!);
33         this.totalPages = Math.floor(totalProducts / this.pageSize);
34         if (totalProducts % this.pageSize != 0){
35           this.totalPages = this.totalPages + 1;
36         }
37       },
38       error : err => {
39         console.log(err);
40       }
41     })
42   //this.products$ = this.productService.getProducts();
43 }
44
45 handleCheckProduct(product : Product) {
46   this.productService.checkProduct(product).subscribe( observerOrNext: {
47     ProductsComponent > searchProducts()
  
```

On change aussi dans notre service, et on teste pour voir les produits s'affiche dans plusieurs pages telles que le size de chaque page est 3, et lorsqu'on fait la recherche il s'affiche les pages contenant les produits recherchés.

```

@Injectable({
  providedIn: 'root'
})
export class ProductService {

  constructor(private http:HttpClient) { }

  public searchProducts(keyword: string="", page: number = 1, size: number = 4){
    return this.http.get(
      `http://localhost:8089/products?name_like=${keyword}_page=${page}&_limit=${size}`
      , {observe: "response"});
  }
}
  
```

Il y'a encore un petit problème c'est que quand on est dans la première ou la dernière page et on supprime les produits, la page reste vide mais afficher, pour cela on doit ajouter des trucs pour que le numéro de la page s'incrémente ou décrémente selon si on est au début ou à la fin.

Après on va passer pour faire la modification des produits, et on va commencer par la partie html pour afficher le bouton éditer :

```

<td>{{product.name}}</td>
<td>{{product.price}}</td>
<td>
    <button (click)="handleCheckedProduct(product)" class="btn btn-outline-success">
        <i [class]="'bi bi-check'">'bi bi-circle'"></i>
    </button>
</td>
<td>
    <button (click)="handleDelete(product)" class="btn btn-outline-danger">
        <i class="bi bi-trash"></i>
    </button>
</td>
<td>
    <button (click)="handleEdit(product)" class="btn btn-outline-danger">
        <i class="bi bi-pencil"></i>
    </button>
</td>
</tr>

```

Name	Price	Checked	
Computer	5600		
New Printer	3200		
Smart	600		

1 2 3 4

Puis on va pour générer un nouveau composant qui permet d'édition le produit :

```
C:\Users\lenovo\angular-projects\fsm-app>ng g c edit-product
CREATE src/app/edit-product/edit-product.component.html (28 bytes)
CREATE src/app/edit-product/edit-product.component.spec.ts (660 bytes)
CREATE src/app/edit-product/edit-product.component.ts (232 bytes)
CREATE src/app/edit-product/edit-product.component.css (0 bytes)
UPDATE src/app/app.module.ts (1098 bytes)
```

Pour aller vers edit-product on est besoin de créer une route d'abord, cette route il va contenir un paramètre id, on ajoute les paramètres dans angular après un slash :

```

import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { ProductsComponent } from './products/products.component';
import { NewProductComponent } from './new-product/new-product.component';
import { EditProductComponent } from './edit-product/edit-product.component';

const routes: Routes = [
  {path : "home", component : HomeComponent},
  {path : "products", component : ProductsComponent},
  {path : "newProduct", component : NewProductComponent},
  {path : "editProduct/:id", component : EditProductComponent}
];

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

```

On passe pour définir notre méthode handleEdit qu'on a cité dans la partie html, mais tout d'abord on va injecter le router, après on teste :

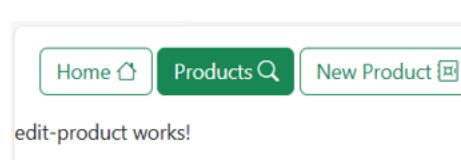
```

  |   |
  |   })
  |

  handleGotoPage(page: number) {
    this.currentPage = page;
    this.searchProducts();
  }

  handleEdit(product : Product) {
    this.router.navigateByUrl(`url: `/editProduct/${product.id}`)
  }
}

```



Maintenant on revient vers notre composant pour faire éditer, on commence par l'injection d'activatedRoute c'est pour le route active, on a snapshot qui fait un copie pour notre route, et params pour indiquer le paramètre qui nous intéresse :

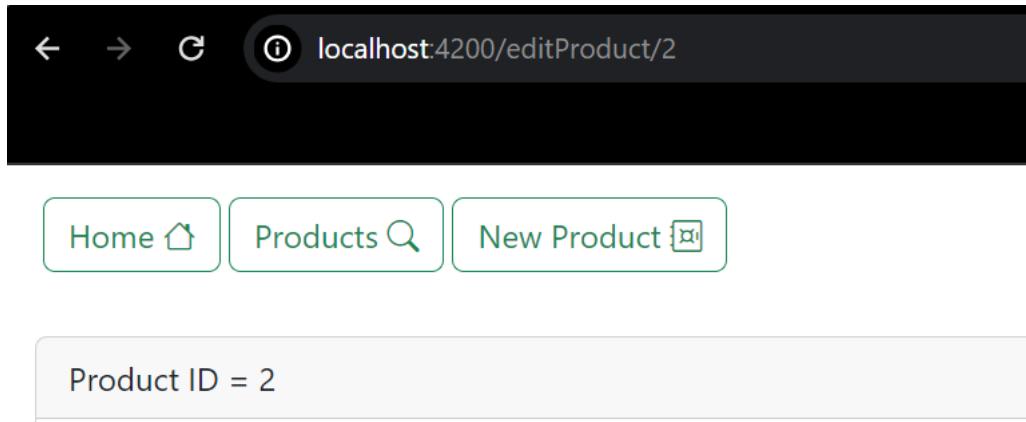
```
import {ActivatedRoute} from '@angular/router';

@Component({
  selector: 'app-edit-product',
  templateUrl: './edit-product.component.html',
  styleUrls: ['./edit-product.component.css']
})
export class EditProductComponent implements OnInit{
  productId!: number;
  constructor(private activatedRoute : ActivatedRoute) {
  }
  ngOnInit() {
    this.productId = this.activatedRoute.snapshot.params['id'];
  }
}
```

Pour les tester, on revient vers la partie html :

```
<div class="p-3">
  <div class="card">
    <div class="card-header">
      Product ID = {{productId}}
    </div>
    <div class="card-body">

    </div>
  </div>
</div>
```



On passe pour que nous envoyions une requête au backend pour récupérer le produit, pour l'éditer :

```
export class EditProductComponent implements OnInit{
  productId!: number;
  productFormGroup!: FormGroup;
  constructor(private activatedRoute: ActivatedRoute,
    private productService: ProductService,
    private fb: FormBuilder) {}
  ngOnInit() {
    this.productId = this.activatedRoute.snapshot.params['id'];
    this.productService.getProductById(this.productId).subscribe({
      next : (product) => {
        this.productFormGroup = this.fb.group( controls: {
          id : this.fb.control(product.id),
          name: this.fb.control(product.name, [Validators.required]),
          price: this.fb.control(product.price, [Validators.required]),
          checked : this.fb.control(product.checked)
        }),
        error : error => {
          console.log(error);}});}}
```

On définit la méthode `getProductById` dans notre service :

```
duct.component.ts × product.service.ts × edit-product.component.html ×
}

public deleteProduct(product:Product){
  return this.http.delete<any>( url: `http://localhost:8089/products/${product.id}` );
}

saveProduct(product: Product):Observable<Product> {
  return this.http.post<Product>( url: `http://localhost:8089/products` ,
  product);
}
getProductById(productId: number) {
  return this.http.get<Product>( url: `http://localhost:8089/products/${productId}` );
}
```

On a initialisé notre formulaire, donc il reste de l'afficher donc revenant à la partie du html :

```
<div class="card-header">
  Product ID = {{productId}}
</div>
<div class="card-body" *ngIf="productFormGroup">
  <form [formGroup]="productFormGroup" (ngSubmit)="updateProduct()">
    <div class="mb-3">
      <label class="form-label">Name</label>
      <input class="form-control" formControlName="name">
    </div>
    <div class="mb-3">
      <label class="form-label">Price</label>
      <input class="form-control" formControlName="price">
    </div>
    <div class="mb-3">
      <label class="form-label">Checked</label>
      <input type="checkbox" class="form-check-input" formControlName="checked">
    </div>
    <button [disabled]="productFormGroup.invalid" class="btn btn-success">Save</button>
  </form>
</div>
```

localhost:4200/editProduct/2

Home Products New Product

Product ID = 2

Name

New Printer

Price

3200

Checked

Save

Et en fin on définit la méthode dans notre service aussi :

```

    }
    updateProduct(product: Product): Observable<Product> {
      return this.http.put<Product>( url: `http://localhost:8089/products/${product.id}`, product);
    }
  }
}

```

Partie 3 :

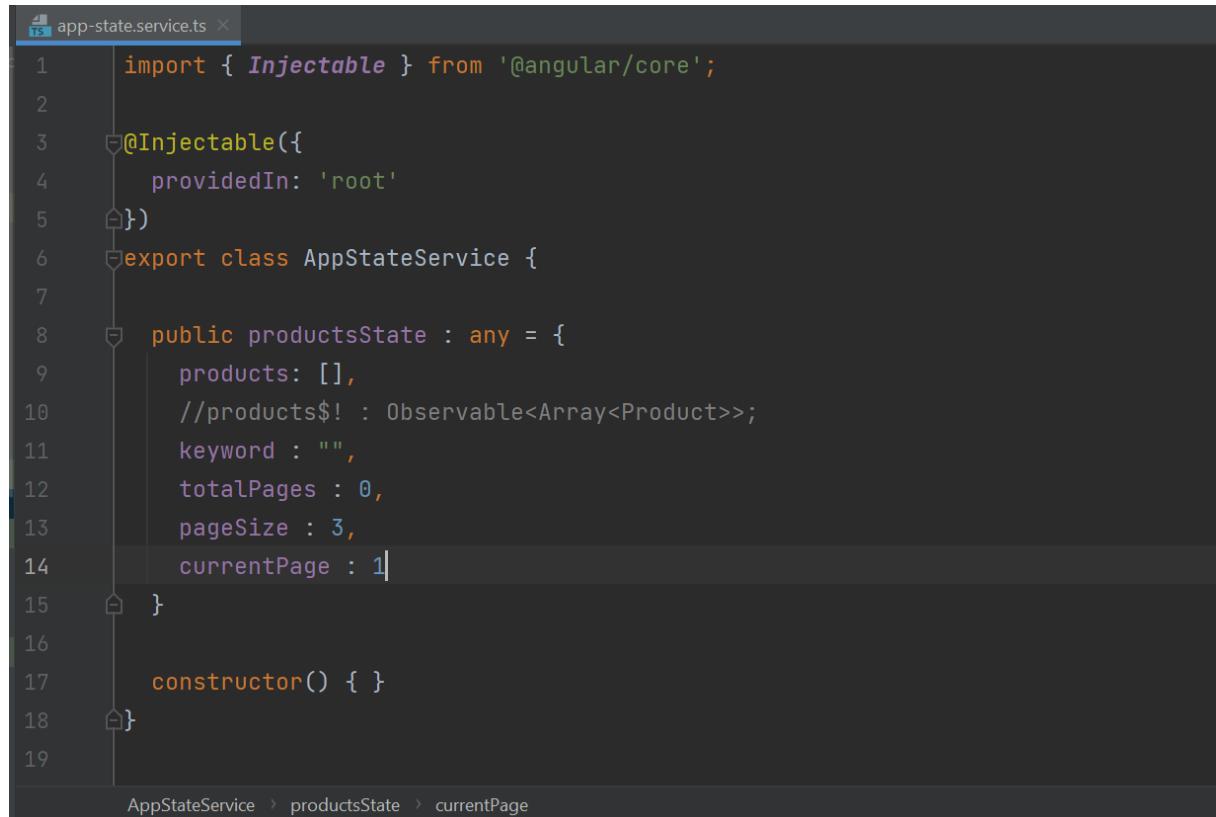
Supposons nous avons un composant qui permet de faire un dashboard mais ce dashboard il dépend des données qui se trouve dans l'autre composant, alors maintenant on va tomber dans un problème, c'est que nous avons en fait les deux composant, comment ils vont communiquer ? donc on a une hiérarchie des composants, ça veut dire le premier composant utilise les données qui sont déclarer dans l'autre composant.

Pour qu'il peut les utiliser on peut commencer par des solutions basé sur des inputs et des outputs, mais c'est vraiment par une bonne solution, alors c'est la raison pour laquelle souvent il est plus utile quand nous avons des données de notre application qui ont besoin d'être utilisé par plusieurs composants, alors dans cette situation-là, il est plus important de centraliser les données de l'application dans un service, donc on va appeler app state, c'est-à-dire c'est un

service dans lequel on va mettre l'état de l'application ce qui nous facilite la gestion de nos données, alors pour cela on va faire un réfactoring de notre application.

```
C:\Users\lenovo\angular-projects\fsm-app>ng g s services/app-state
CREATE src/app/services/app-state.service.spec.ts (384 bytes)
CREATE src/app/services/app-state.service.ts (146 bytes)
```

Dans notre service app-state, on va créer une variable qu'on va appeler productsState, puis on va déplacer les données de states de notre composant products à notre service app-state :



```
app-state.service.ts
1 import { Injectable } from '@angular/core';
2
3 @Injectable({
4   providedIn: 'root'
5 })
6 export class AppStateService {
7
8   public productsState : any = {
9     products: [],
10    //products$! : Observable<Array<Product>>;
11    keyword : "",
12    totalPages : 0,
13    pageSize : 3,
14    currentPage : 1
15  }
16
17  constructor() { }
18}
```

AppStateService > productsState > currentPage

Alors maintenant pour pouvoir les utiliser dans products, et pour gérer les erreurs qu'on voit partout, et bien tout simplement on va injecter le service app-state en public pour que l'on utilise directement dans la partie template, et on appelle les données de notre service :

```
export class ProductsComponent implements OnInit{
  public products : Array<Product> = [];
  //products$! : Observable<Array<Product>>;
  public keyword : string = "";
  totalPages : number = 0;
  pageSize : number = 3;
  currentPage : number = 1;
  constructor(private productService : ProductService,
              private router : Router,
              public appState: AppStateService) {
    }
}
```

On va faire la même chose pour la partie html, et on teste :

```
<div class="card">
  <div class="card-body">
    <div class="card-body">
      <input class="p-1" type="text" [(ngModel)]="appState.productsState.keyword">
      <button (click)="searchProducts()">
        <i class="bi bi-search"></i>
      </button>
    </div>
    <table class="table">
      <thead>
        <tr>
          <th>Name</th><th>Price</th><th>Checked</th>
        </tr>
      </thead>
      <tbody>
        <tr *ngFor="let product of appState.productsState.products">
          <td>{{product.name}}</td>
          <td>{{product.price}}</td>
          <td>
```

The screenshot shows a user interface for a dashboard. At the top, there are three buttons: "Home" with a house icon, "Products" with a magnifying glass icon, and "New Product" with a plus icon. Below these is a search bar with a magnifying glass icon. The main area contains a table with columns "Name", "Price", and "Checked". The table has three rows of data: "Computer" (Price 5600), "New Printer" (Price 3200), and "Smart" (Price 600). Each row has three icons: a green checkmark, a red trash can, and a green pencil. At the bottom of the table are four numbered buttons: 1 (dark green), 2 (light green), 3 (light green), and 4 (light green).

Name	Price	Checked
Computer	5600	
New Printer	3200	
Smart	600	

Pourquoi on fait ça ? Ça va nous faciliter les choses parce que supposons nous voulons créer un composant dashboard par exemple, on va le créer d'abord :

```
C:\Users\lenovo\angular-projects\fsm-app>ng g c dashboard
CREATE src/app/dashboard/dashboard.component.html (25 bytes)
CREATE src/app/dashboard/dashboard.component.spec.ts (645 bytes)
CREATE src/app/dashboard/dashboard.component.ts (221 bytes)
CREATE src/app/dashboard/dashboard.component.css (0 bytes)
UPDATE src/app/app.module.ts (1192 bytes)
```

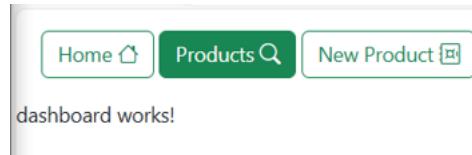
Puis on revient à notre partie html de l'application et on affiche le dashboard :

```

<nav class="p-3">
  <ul class="nav nav-pills">
    <li *ngFor="let action of actions">
      <button
        (click)="setCurrentAction(action)"
        routerLink="{{action.route}}"
        [class]="action==currentAction
          ?'btn btn-success ms-1': 'btn btn-outline-success ms-1'"
        {{action.title}}
        <i class="bi bi-{{action.icon}}"></i>
      </button>
    </li>
  </ul>
</nav>
<app-dashboard></app-dashboard>
<router-outlet>
</router-outlet>

```

nav.p-3



Pour qu'on affiche des statistiques de notre application, on va à la partie TypeScript de notre composant dashboard, on injecte le service dans un constructeur (là on voit l'intérêt de centraliser les données dans un service), et on définit la méthode totalCheckedProducts que nous utiliserons dans la partie html, et qui nous retourne le nombre des produits checker dans une page :

```

import { Component } from '@angular/core';
import {AppStateService} from "../services/app-state.service";

@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css'
})
export class DashboardComponent {
  constructor(public appState: AppStateService) {
  }
  totalCheckedProducts() {
    let checkedProducts =
      this.appState.productsState.products.filter((p : any) => p.checked == true);
    return checkedProducts.length;
  }
}

```

On passe à la partie html pour créer les labels dans lequel on va afficher nos statistiques :

```


- Pages : {{appState.productsState.totalPages}}
- Pages : {{appState.productsState.pageSize}}
- Pages : {{appState.productsState.totalProducts}}
- Checked : {{totalCheckedProducts()}}

```

Pour afficher le nombre total des produits, on a ajouté totalProducts avec nos données centralisé :

```

})
export class AppStateService {

    public productsState : any = {
        products: [],
        //products$! : Observable<Array<Product>>;
        keyword : "",
        totalPages : 0,
        pageSize : 3,
        currentPage : 1,
        totalProducts: 0
    }
}

constructor() { }

}

```

Puis on le récupérer et stocker dans la variable totalProducts :

```

cts.component.ts ✘
}

ngOnInit() {
    this.searchProducts();
}

searchProducts(){
    this.productService.searchProducts(this.keyword, this.currentPage, this.pageSize)
        .subscribe( observerOrNext: {
            next : (resp : HttpResponse<Object>) => {
                this.products = resp.body as Product[];
                let totalProducts : number = parseInt(resp.headers.get('x-total-count')!)!
                this.appState.productsState.totalProducts = totalProducts;
                this.totalPages = Math.floor( x.totalProducts / this.pageSize);
                if (totalProducts % this.pageSize != 0){
                    this.totalPages = this.totalPages + 1;
                }
            },
            error : err => {

```

ProductsComponent > searchProducts() > next() > totalProducts

Le problème qu'on a maintenant c'est lorsqu'on supprime un produit le nombre total des produits ne change que lorsqu'on recharge la page, pour cela et dans le même composant on

revient à notre méthode handleDelete, et on va appeler la méthode searchProducts qui va recharger les données :

```
handleDelete(product : Product) {
  if (confirm("Etes vous sûre?"))
    this.productService.deleteProduct(product).subscribe( observerOrNext: {
      next : value => {
        // this.getProducts();
        //this.appState.productsState.products =
        // this.appState.productsState.products.filter((p : any) => p.id != product.id);
        this.searchProducts();
      }
    })
}
```

The screenshot shows a web application interface. At the top, there are three buttons: "Home" (disabled), "Products" (highlighted in green), and "New Product". Below these are four status indicators: "Pages :4", "Size :3", "Total :10", and "Checked :3". The main content area displays a table of products with columns for Name, Price, and Checked status. The table contains three rows: Computer (5600), New Printer (3200), and Smart (600). Each row has a "check" icon, a "trash" icon, and an "edit" icon. At the bottom of the table are page navigation buttons labeled 1, 2, 3, and 4.

Name	Price	Checked
Computer	5600	
New Printer	3200	
Smart	600	

On fait un petit refactoring à notre code, donc au lieu d'appeler un attribut on va créer une méthode dans laquel on utilise des trois points pour qu'on faire copier tous les attributs de productsState, c'est ce qu'on appelle un objet immuable :

The screenshot shows a code editor with the tab bar at the top. The active tab is 'app-state.service.ts' (indicated by a blue background and white 'TS' icon). There are other tabs for 'cts.component.ts' and another partially visible component. The code itself is a TypeScript class definition:

```
cts.component.ts x app-state.service.ts x
})
export class AppStateService {

    public productsState : any = {
        products: [],
        //products$!: Observable<Array<Product>>;
        keyword : "",
        totalPages : 0,
        pageSize : 3,
        currentPage : 1,
        totalProducts: 0
    }

    constructor() { }

    public setProductState(state: any) : void {
        this.productsState= {...this.productsState, ...state}
    }
}

AppStateService > setProductState() > productsState
```

On change aussi au niveau de products, par exemple au lieu d'aller directement stocker dans des variables, on va déclarer des variables et on change des données en utilisant la méthode setProductState :

```

searchProducts(){
  this.productService.searchProducts(
    this.appState.productsState.keyword,
    this.appState.productsState.currentPage,
    this.appState.productsState.pageSize)
  .subscribe( observerOrNext: {
    next : (resp : HttpResponse<Object> ) => {
      this.products = resp.body as Product[];
      let totalProducts : number = parseInt(resp.headers.get('x-total-count')!)!;
      //this.appState.productsState.totalProducts = totalProducts;
      this.totalPages = Math.floor( x: totalProducts / this.appState.productsState.pageSize);
      if (totalProducts % this.appState.productsState.pageSize != 0){
        ++this.totalPages;
      }
      this.appState.setProductState({
        products : this.products,
        totalProducts : totalProducts,
        totalPages: this.totalPages
      })
    }
  },
  ProductsComponent > searchProducts() > next()

```

On revient vers notre service product, comme nous voyons le host ce n'est pas bien de les mettre directement, ou de le coder en dur le host de notre URL, donc ce que nous pouvons faire c'est dans un premier temps de déclarer une variable que nous appelons par exemple host et en lui donner comme valeur notre host :

```

import {Observable} from "rxjs";
import {Product} from "../model/product.model";

@Injectable({
  providedIn: 'root'
})
export class ProductService {

  private host: string = "http://localhost:8089";

  constructor(private http : HttpClient) { }

  public searchProducts(keyword : string="", page : number=1, size : number=4){
    return this.http.get(
      url: `http://localhost:8089/products?name_like=${keyword}&_page=${page}&_limit=${size}`,
      options: {observe : 'response'});
  }

  public checkProduct(product : Product):Observable<Product>{

```

On revient vers notre service app-state pour ajouter des messages, pour qu'il s'affiche un spinner pour que nous indique que les données sont en chargement :

```

export class AppStateService {

    public productsState : any = {
        products: [],
        //products$! : Observable<Array<Product>>;
        keyword : "",
        totalPages : 0,
        pageSize : 3,
        currentPage : 1,
        totalProducts: 0,
        status: "",
        errorMessage:""
    }

    constructor() { }

    public setProductState(state: any) : void {
        this.productsState= {...this.productsState, ...state}
    }
}

AppStateService > productsState > errorMessage

```

On revient dans notre composant products, dans searchProducts avant d'envoyer la requête en peut aller vers AppState et on accède directement à la méthode setProductState et on change la valeur de status en LOADING, et qu'une fois les données sont chargé on met status en LOADED et dans la partie des erreurs on met status en ERROR et en transmis un message d'erreur :

```

//this.appState.productsState.totalProducts = totalProducts;
this.totalPages = Math.floor( x totalProducts / this.appState.productsState.pageSize);
if (totalProducts % this.appState.productsState.pageSize != 0){
    ++this.totalPages;
}
this.appState.setProductState({
    products : this.products,
    totalProducts : totalProducts,
    totalPages: this.totalPages,
    status: "LOADED"
})
},
error : err => {
    this.appState.setProductState({
        status: "ERROR",
        errorMessage: err
    })
    console.log(err);
}

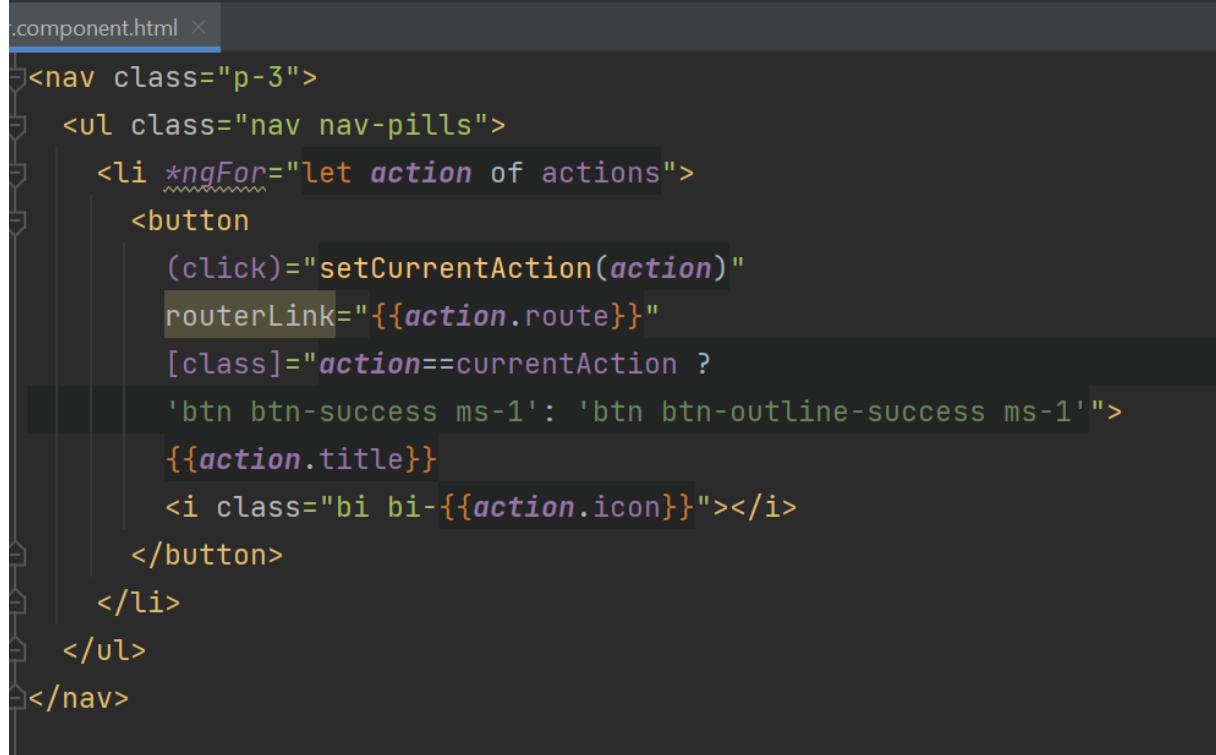
```

ProductsComponent > searchProducts() > error()

Maintenant on passe pour afficher un spinner dans navbar, mais avant on doit générer un composant navbar :

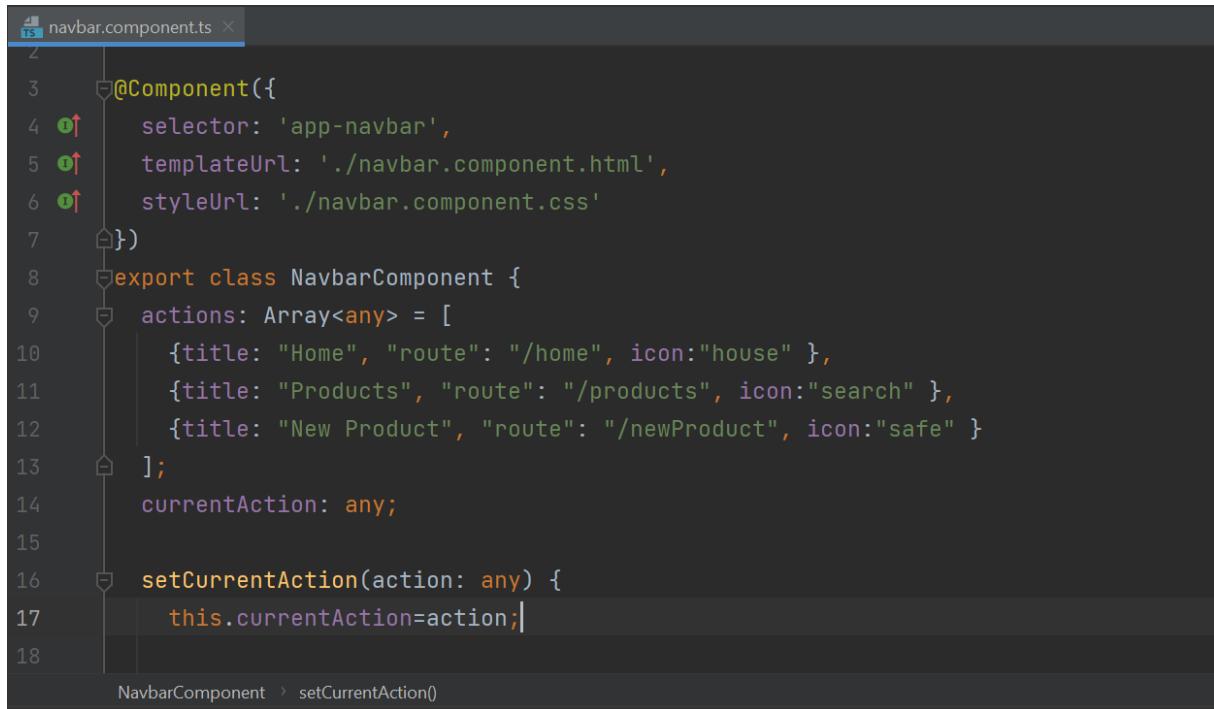
```
C:\Users\lenovo\angular-projects\fsm-app>ng g c navbar
CREATE src/app/navbar/navbar.component.html (22 bytes)
CREATE src/app/navbar/navbar.component.spec.ts (624 bytes)
CREATE src/app/navbar/navbar.component.ts (209 bytes)
CREATE src/app/navbar/navbar.component.css (0 bytes)
UPDATE src/app/app.module.ts (1276 bytes)
```

Puis on prend le code html de navbar de notre application et on le mettre dans le fichier html de notre composant navbar :



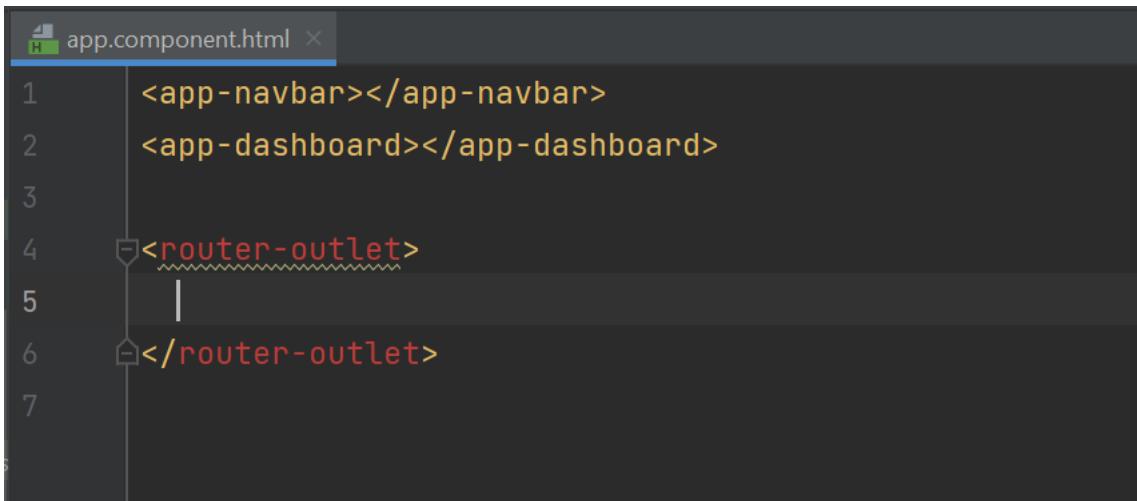
```
.component.html ×
<nav class="p-3">
  <ul class="nav nav-pills">
    <li *ngFor="let action of actions">
      <button
        (click)="setCurrentAction(action)"
        routerLink="{{action.route}}"
        [class]="action==currentAction ?
          'btn btn-success ms-1': 'btn btn-outline-success ms-1'">
        {{action.title}}
        <i class="bi bi-{{action.icon}}"></i>
      </button>
    </li>
  </ul>
</nav>
```

On prend le code qui contient les variables que nous utilisons dans la partie html, du composant de l'application vers notre composant navbar :



```
navbar.component.ts
2
3   @Component({
4     selector: 'app-navbar',
5     templateUrl: './navbar.component.html',
6     styleUrls: ['./navbar.component.css']
7   })
8   export class NavbarComponent {
9     actions: Array<any> = [
10       {title: "Home", "route": "/home", icon:"house" },
11       {title: "Products", "route": "/products", icon:"search" },
12       {title: "New Product", "route": "/newProduct", icon:"safe" }
13     ];
14     currentAction: any;
15
16     setCurrentAction(action: any) {
17       this.currentAction = action;
18     }
}
NavbarComponent > setCurrentAction()
```

N'oubliez pas d'ajouter app-navbar dans la partie html du composant de l'application :



```
app.component.html
1   <app-navbar></app-navbar>
2   <app-dashboard></app-dashboard>
3
4   <router-outlet>
5   |
6   </router-outlet>
7
```

The screenshot shows a web application interface for managing products. At the top, there's a header with a back arrow, forward arrow, refresh icon, and the URL 'localhost:4200/products'. Below the header is a navigation bar with three buttons: 'Home' (with a house icon), 'Products' (with a magnifying glass icon), and 'New Product' (with a plus icon). Underneath the navigation bar, there are four status indicators: 'Pages :4' (red), 'Size :3' (blue), 'Total :10' (green), and 'Checked :3' (blue). The main content area contains a table with the following data:

Name	Price	Checked		
Computer	5600	<input checked="" type="checkbox"/>		
New Printer	3200	<input checked="" type="checkbox"/>		
Smart	600	<input checked="" type="checkbox"/>		

Below the table is a navigation bar with four buttons labeled 1, 2, 3, and 4.

Maintenant à l'intérieur de navbar on ajoute notre spinner :

```
<button
  (click)="setCurrentAction(action)"
  routerLink="{{action.route}}"
  [class]="action==currentAction ? 'btn btn-success ms-1': 'btn btn-outline-success ms-1'"
  {{action.title}}
  <i class="bi bi-{{action.icon}}"></i>
</button>
</li>
<li>
  | <div class="spinner-border text-primary ms-2" role="status">
    </div>
</li>
</ul>
</nav>
```

The screenshot shows a user interface for managing products. At the top, there are three buttons: "Home" (with a house icon), "Products" (with a magnifying glass icon), and "New Product" (with a plus icon). Below these is a navigation bar with four items: "Pages :4" (red), "Size :3" (blue), "Total :10" (green), and "Checked :3" (blue). The main content area contains a search bar with a magnifying glass icon. A table lists three products:

Name	Price	Checked
Computer	5600	<input checked="" type="checkbox"/>
New Printer	3200	<input checked="" type="checkbox"/>
Smart	600	<input checked="" type="checkbox"/>

Below the table is a pagination section with buttons labeled 1, 2, 3, and 4.

Mais on ne veut pas que le spinner s'affiche que lorsque le status est en loading, pour cela on injecte `appState` dans notre composant :

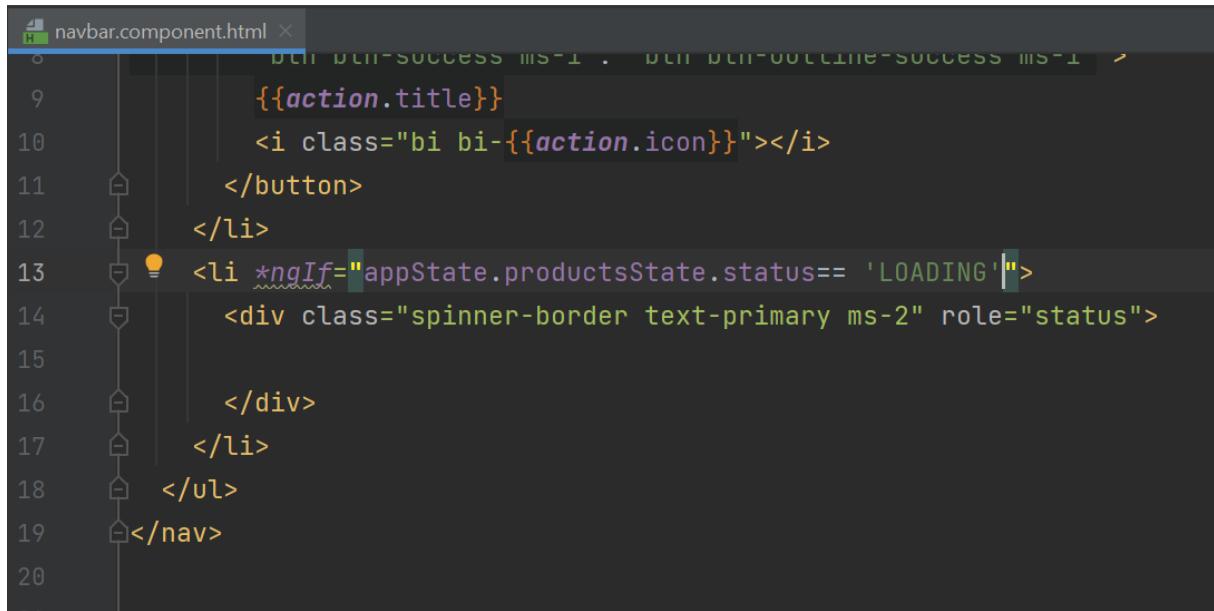
```

1 navbar.component.ts
2
3   selector: 'app-navbar',
4   templateUrl: './navbar.component.html',
5   styleUrls: ['./navbar.component.css']
6 }
7
8 export class NavbarComponent {
9   actions: Array<any> = [
10     {title: "Home", "route": "/home", icon:"house" },
11     {title: "Products", "route": "/products", icon:"search" },
12     {title: "New Product", "route": "/newProduct", icon:"safe" }
13   ];
14
15   currentAction: any;
16   constructor(public appState : AppStateService) {
17   }
18
19   setCurrentAction(action: any) {
20     this.currentAction=action;
21   }

```

NavbarComponent > constructor()

Et on revient dans la partie html pour faire un test avec `ngIf` :

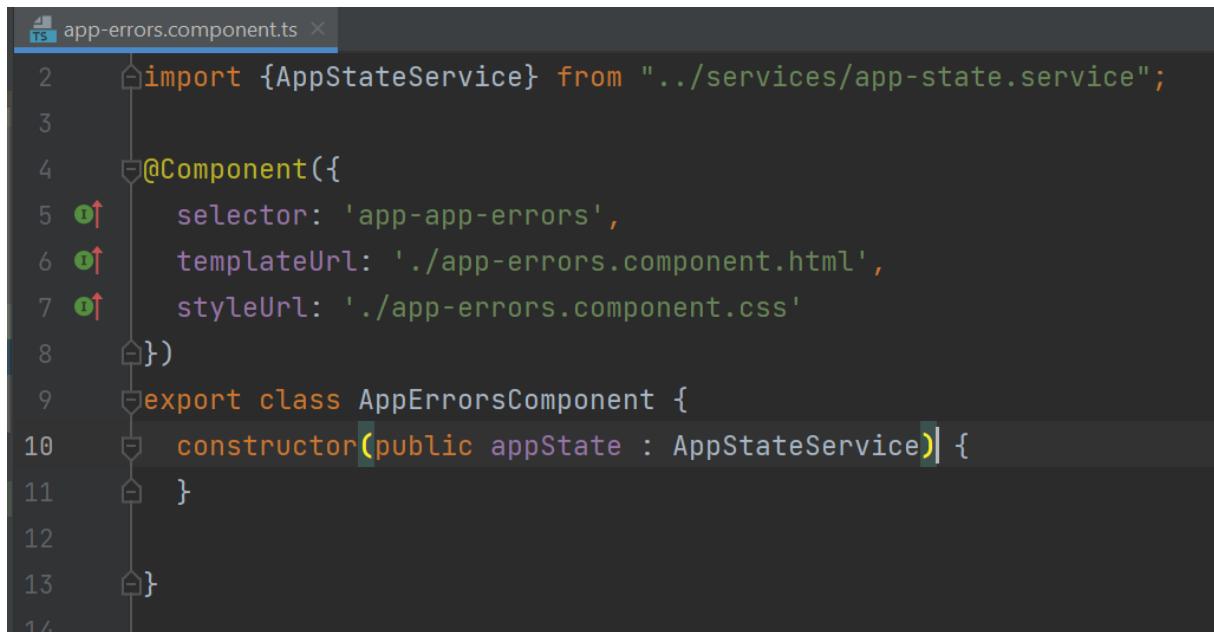


```
9     {{action.title}}
10    <i class="bi bi-{{action.icon}}"></i>
11    </button>
12  </li>
13  <li *ngIf="appState.productsState.status == 'LOADING'">
14    <div class="spinner-border text-primary ms-2" role="status">
15      </div>
16  </li>
17 </ul>
18 </nav>
19
20
```

Maintenant on crée un composant que l'on appelle app-errors :

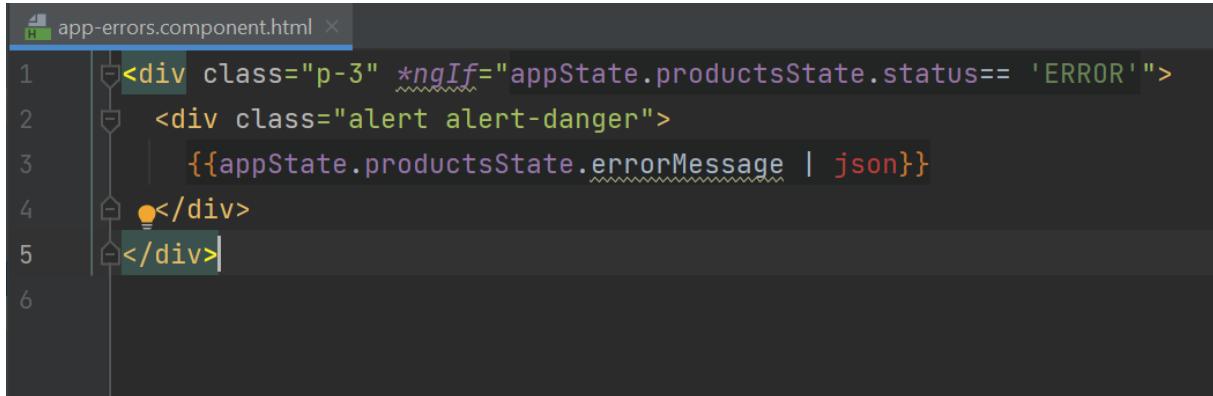
```
C:\Users\lenovo\angular-projects\fsm-app>ng g c app-errors
CREATE src/app/app-errors/app-errors.component.html (26 bytes)
CREATE src/app/app-errors/app-errors.component.spec.ts (646 bytes)
CREATE src/app/app-errors/app-errors.component.ts (224 bytes)
CREATE src/app/app-errors/app-errors.component.css (0 bytes)
UPDATE src/app/app.module.ts (1374 bytes)
```

De même on va injecter le service appState :



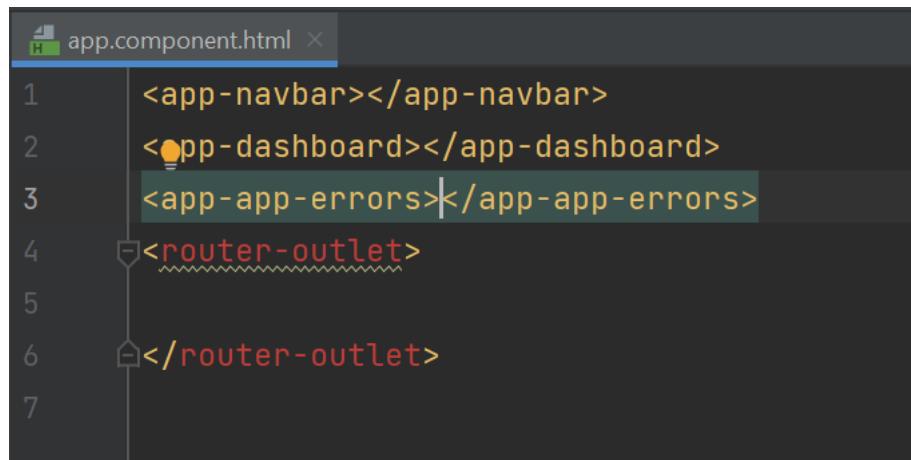
```
1 import {AppStateService} from "../services/app-state.service";
2
3 @Component({
4   selector: 'app-app-errors',
5   templateUrl: './app-errors.component.html',
6   styleUrls: ['./app-errors.component.css']
7 })
8 export class AppErrorsComponent {
9   constructor(public appState : AppStateService) {
10   }
11 }
12
13 }
```

On passe pour la partie html, dans laquel on met une condition d'afficher le message d'erreur en format json uniquement si le status égale ERROR :

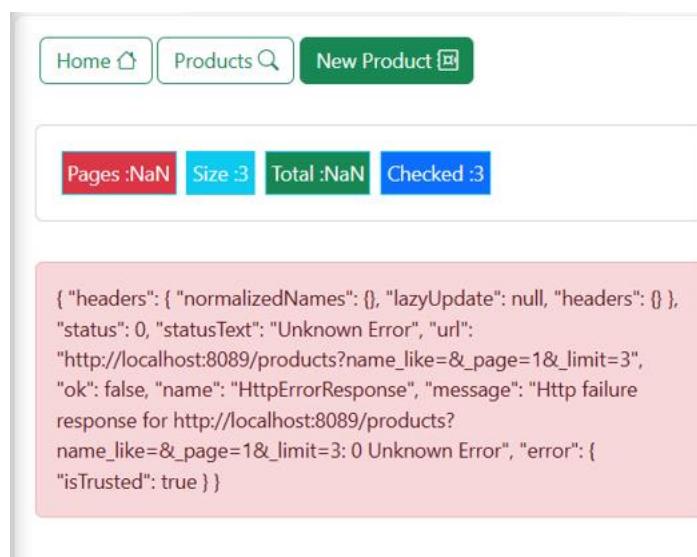


```
1 <div class="p-3" *ngIf="appState.productsState.status== 'ERROR' "gt;
2   <div class="alert alert-danger">
3     {{appState.productsState.errorMessage | json}}
4   </div>
5 </div>
6
```

N'oubliez pas d'ajouter app-app-errors à la partie html du composant de l'application pour qu'il s'affiche :



```
1 <app-navbar></app-navbar>
2 <app-dashboard></app-dashboard>
3 <app-app-errors></app-app-errors>
4 <router-outlet>
5
6 </router-outlet>
7
```



Maintenant ce qui nous intéresse dans tout ça c'est le message, donc va lui afficher seul :

```

1 <div class="p-3" *ngIf="appState.productsState.status== 'ERROR' ">
2   <div class="alert alert-danger">
3     {{appState.productsState.errorMessage.message}}
4   </div>
5 </div>
6

```

Name	Price	Checked
------	-------	---------

Pour simuler les applications du réseau lent il suffit de choisir Lente 3G :

Nom	Statut	Type	Initiateur	Taille	Durée	Rempli par
products?name_like=&_page=1&_limit=3	200	xhr	products.component.ts:34	674 B	2.04 s	

On remarque que à chaque fois, on doit envoyer le status avant la requête et après, et en état d'erreur dans chaque composant et ça dérange vraiment, alors la meilleure solution pour ce genre de chose est d'utiliser ce qu'on appelle un intercepteur http. L'intercepteur http c'est un service qui intercepte toutes les requêtes http, à chaque fois que nous envoyons une requête http il va l'intercepter avant qu'il envoie.

Et si ce qu'on veut, on envoie une requête mais avant qu'il soit envoyée, on va mettre le state de l'application à loading, une fois qu'on reçoit la réponse on le met en loaded, et le problème est réglé.

Donc on va créer un intercepteur que l'on appelle app-http :

```
C:\Users\lenovo\angular-projects\fsm-app>ng g interceptor app-http
CREATE src/app/app-http.interceptor.spec.ts (506 bytes)
CREATE src/app/app-http.interceptor.ts (157 bytes)
```

Dans cet intercepteur on va utiliser clone pour créer une copie de la requête, et supposons nous voulons mettre un header on peut utiliser req.headers et on retourne la requête :

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

@Injectable()
export class AppHttpInterceptor implements HttpInterceptor {
  constructor() {}

  intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    let req = request.clone({ update: {
      headers: request.headers.set("Authorization", "Bearer JWT")
    }});
    return next.handle(req);
  }
}
```

Et pour qu'il fonctionne, on doit le déclarer dans le module précisément dans providers :

```

imports: [
  BrowserModule,
  AppRoutingModule,
  HttpClientModule,
  ReactiveFormsModule,
  FormsModule
],
providers: [
  provideClientHydration(),
  { provide: HTTP_INTERCEPTORS, useClass: AppHttpInterceptor, multi: true}
],
bootstrap: [AppComponent]
)
export class AppModule { }

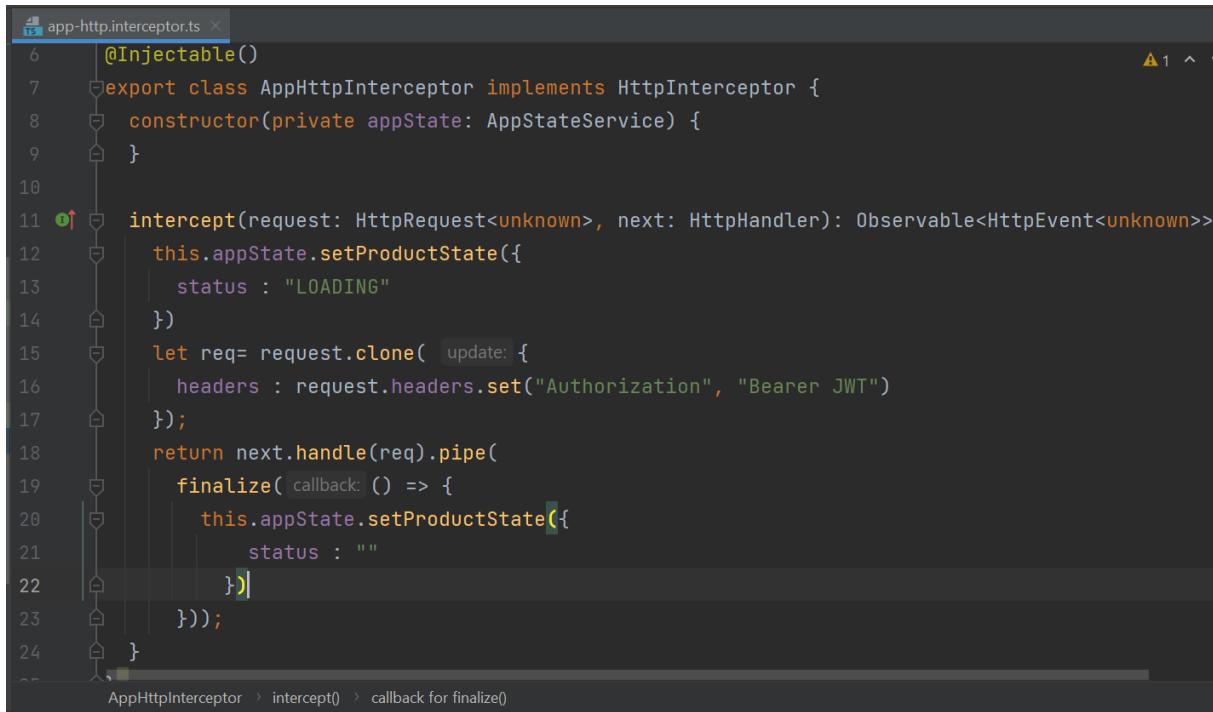
```

Si nous inspectons nos requêtes, nous allons trouver dans les headers, Authorization pour qu'on connaisse que notre intercepteur se fonctionne :

The screenshot shows the Chrome DevTools Network tab. At the top, there are buttons for 'Pages: 4', 'Size: 3', 'Total: 10', and 'Checked: 3'. Below the table, the Network tab is selected, showing a timeline from 50000 ms to 400000 ms. A specific request to 'products...' is highlighted. In the 'En-têtes' (Headers) section, the 'Authorization' header is visible with the value 'Bearer JWT'.

Nom	En-têtes	Charge utile	Aperçu	Réponse	Initiateur	Expiration du délai
bootstra...						
favicon.ico						
products...	Accept: application/json, text/plain, */*					
products...	Accept-Encoding: gzip, deflate, br, zstd					
products...	Accept-Language: fr,fr-FR;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6					
22 demandes	Authorization: Bearer JWT					

Maintenant on va injecter appState dans notre intercepteur, pour qu'il fasse loading avant qu'il retourne la requête, et donc une fois que la réponse arrive on va l'écouter en utilisant pipe :



```
6  @Injectable()
7  export class AppHttpInterceptor implements HttpInterceptor {
8      constructor(private appState: AppStateService) {
9          }
10
11     intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>>
12     {
13         this.appState.setProductState({
14             status : "LOADING"
15         })
16         let req= request.clone( { update: {
17             headers : request.headers.set("Authorization", "Bearer JWT")
18         }};
19         return next.handle(req).pipe(
20             finalize( callback: () => {
21                 this.appState.setProductState({
22                     status : ""
23                 });
24             }
25         );
26     }
27 }
```

AppHttpInterceptor > intercept() > callback for finalize()

Maintenant on va voir l'autre solution pour le spinner c'est de créer un subject, qui est un type d'observable particulier qui peut lui-même être un observateur, alors on va créer un service que nous appelons loading :

```
C:\Users\lenovo\angular-projects\fsm-app>ng g s services/loading
CREATE src/app/services/loading.service.spec.ts (378 bytes)
CREATE src/app/services/loading.service.ts (145 bytes)
```

Alors dans notre loading service on va déclarer un subject que l'on appelle isLoading, et on créer deux méthodes showLoadingSpinner, dans laquelle on utilise next, ça veut dire on va émettre, ou bien on va envoyer à n'importe quel composant qui écoute un true, et la méthode hideLoadingSpinner dans laquelle on utilise next en false :

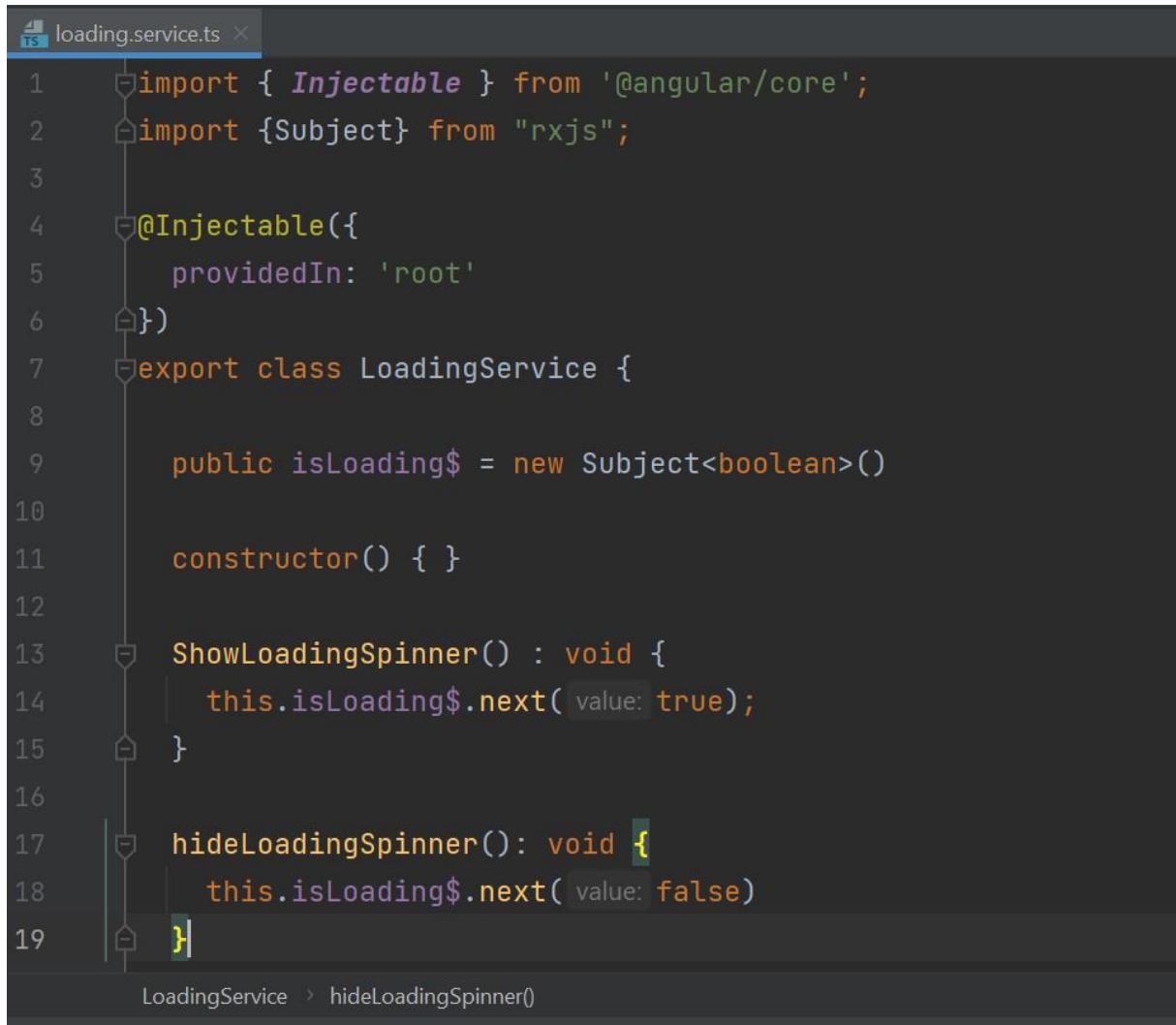
```
5
6     @Injectable()
7     export class AppHttpInterceptor implements HttpInterceptor {
8         constructor(private appState: AppStateService) {
9     }
10
11    intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
12        this.appState.setProductState({
13            status : "LOADING"
14        })
15        let req= request.clone( update: {
16            headers : request.headers.set("Authorization", "Bearer JWT")
17        });
18        return next.handle(req).pipe(
19            finalize( callback: () => {
20                this.appState.setProductState({
21                    status : ""
22                })
23            })
24    )
25 }
```

Alors comment on va l'utiliser ? On va vers notre intercepteur, et on injecte le service loading, et on fait appelle à nos méthodes :

```
12     }
13
14    intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
15        /*this.appState.setProductState({
16            status : "LOADING"
17        }) */
18        let req= request.clone( update: {
19            headers : request.headers.set("Authorization", "Bearer JWT")
20        });
21        return next.handle(req).pipe(
22            finalize( callback: () => {
23                /*this.appState.setProductState({
24                    status : ""
25                }) */
26                this.loadingService.hideLoadingSpinner();
27            })
28        }
29    }
30 }
```

AppHttpInterceptor > intercept() > callback for finalize()

On met le subject en public, et on ajoute un \$ au nom du subject, pour indiquer que c'est un observable :



```
loading.service.ts
1 import { Injectable } from '@angular/core';
2 import {Subject} from "rxjs";
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class LoadingService {
8
9   public isLoading$ = new Subject<boolean>()
10
11  constructor() { }
12
13  ShowLoadingSpinner(): void {
14    this.isLoading$.next( value: true);
15  }
16
17  hideLoadingSpinner(): void {
18    this.isLoading$.next( value: false)
19 }
```

LoadingService > hideLoadingSpinner()

Maintenant on l'utilise directement dans navbar, en injectant le service loading :

```
navbar.component.ts x
7  templateUrl: './navbar.component.html',
8  styleUrls: ['./navbar.component.css'
9  })
10 export class NavbarComponent {
11   actions: Array<any> = [
12     {title: "Home", "route": "/home", icon:"house" },
13     {title: "Products", "route": "/products", icon:"search" },
14     {title: "New Product", "route": "/newProduct", icon:"safe" }
15   ];
16   currentAction: any;
17   constructor(public appState : AppStateService,
18             public loadingService : LoadingService) {
19   }
20
21   setCurrentAction(action: any) {
22     this.currentAction = action;
23   }
24
25 }
```

On change dans la partie html aussi :

```
1  <nav class="p-3">
2    <ul class="nav nav-pills">
3      <li *ngFor="let action of actions">
4        <button
5          (click)="setCurrentAction(action)"
6          routerLink="{{action.route}}"
7          [class]="action==currentAction ? 'btn btn-success ms-1' : 'btn btn-outline-success ms-1'"
8          >{{action.title}}
9          <i class="bi bi-{{action.icon}}"></i>
10         </button>
11       </li>
12       <li *ngIf="loadingService.isLoading$ | async">
13         <div class="spinner-border text-primary ms-2" role="status">
14           <div>
15             </div>
16           </div>
17         </li>
18       </ul>
19     </nav>
```

nav.p-3 > ul.nav.nav-pills > li > div.spinner-border.text-primary.ms-2

On peut faire autre solution, c'est de déclarer un attribut que l'o appelle isLoading :

```
8     styleUrls: './navbar.component.css'
9   })
10  export class NavbarComponent {
11    actions: Array<any> = [
12      {title: "Home", "route": "/home", icon:"house" },
13      {title: "Products", "route": "/products", icon:"search" },
14      {title: "New Product", "route": "/newProduct", icon:"safe" }
15    ];
16    currentAction: any;
17    public isLoading : boolean = false;
18    constructor(public appState : AppStateService,
19                public loadingService : LoadingService) {
20      this.loadingService.isLoading$.subscribe( observerOrNext: {
21        next : (value : T ) =>{
22          this.isLoading = value;
23        }
24      })
25    }
26  }
```

On vérifie directement dans la partie html, si la variable isLoading est égale à true :

```
1 <nav class="p-3">
2   <ul class="nav nav-pills">
3     <li *ngFor="let action of actions">
4       <button
5         (click)="setCurrentAction(action)"
6         routerLink="{{action.route}}"
7         [class]={{action==currentAction ? 'btn btn-success ms-1' : 'btn btn-outline-success ms-1'}}
8         {{action.title}}>
9         <i class="bi bi-{{action.icon}}"></i>
10        </button>
11      </li>
12      <li *ngIf="isLoading == true">
13        <div class="spinner-border text-primary ms-2" role="status">
14          </div>
15      </li>
16    </ul>
17  </nav>
```

nav.p-3 > ul.nav.nav-pills > li

Partie 4 :

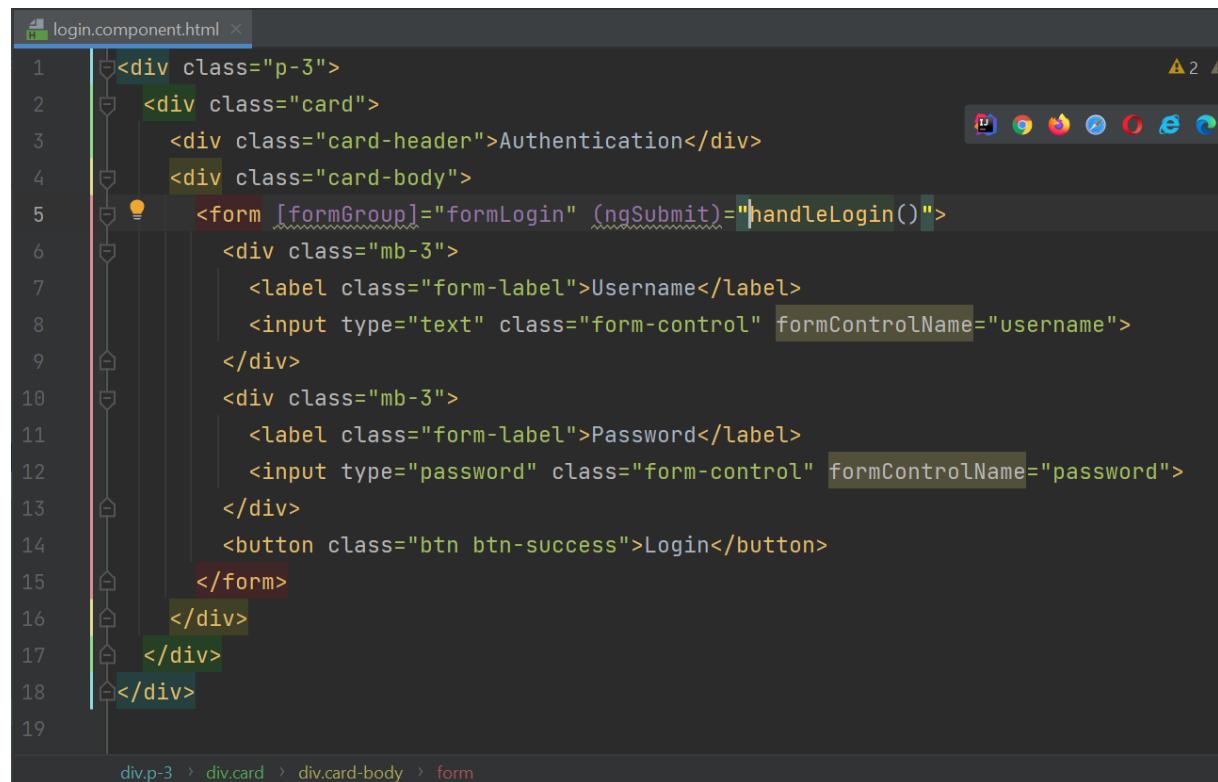
Dans cette partie, on va continuer sur la même application pour montrer les éléments de base pour mettre en place une partie authentification.

On va faire juste quelque chose temporaire, on va dire un fake backend, donc ce qui fait nous allons monter notre système d'authentification sur quelque chose qui n'est pas vraiment sécurisé dans la base, mais après par la suite on va revenir au partie backend encore pour travailler avec Spring pour mettre en place Rest API, et on va la sécurisé, donc on va essayer de voir avec Spring Security, Json web token, comment aborder la sécurité, et comment sécuriser le backend. Et après par la suite, on va adapter en quelque sorte notre frontend, que nous avons développé avec Angular à notre backend pour sortir avec un exemple d'application complet.

Maintenant nous voulons qu'on démarre l'application, va démarrer avec un formulaire d'authentification, alors pour se faire, nous aurons besoin de créer un composant login :

```
C:\Users\lenovo\angular-projects\fsm-app>ng g c login
CREATE src/app/login/login.component.html (21 bytes)
CREATE src/app/login/login.component.spec.ts (617 bytes)
CREATE src/app/login/login.component.ts (205 bytes)
CREATE src/app/login/login.component.css (0 bytes)
UPDATE src/app/app.module.ts (1811 bytes)
```

Passons pour créer un formulaire simple dans notre fichier login.html, et on fait la liaison de notre formulaire et nos champs avec ce qu'on va déclarer dans le component à la suite, aussi on crée une méthode handleLogin qui affiche les valeurs d'authentification :



```
login.component.html
1  <div class="p-3">
2    <div class="card">
3      <div class="card-header">Authentication</div>
4      <div class="card-body">
5        <form [formGroup]="formLogin" (ngSubmit)="handleLogin()">
6          <div class="mb-3">
7            <label class="form-label">Username</label>
8            <input type="text" class="form-control" formControlName="username">
9          </div>
10         <div class="mb-3">
11           <label class="form-label">Password</label>
12           <input type="password" class="form-control" formControlName="password">
13         </div>
14         <button class="btn btn-success">Login</button>
15       </form>
16     </div>
17   </div>
18 </div>
```

Donc bien sûr, on va comme la dernière fois dans login component, déclarer un formulaire, et définir la méthode handleLogin :

```
1 login.component.ts x
2
3 ①selector: 'app-login',
4 ①templateUrl: './login.component.html',
5 ①styleUrl: './login.component.css'
6 }
7
8 export class LoginComponent implements OnInit{
9     formLogin!: FormGroup;
10    constructor(private fb: FormBuilder) {
11    }
12
13 ②ngOnInit(): void {
14     this.formLogin = this.fb.group( controls: {
15         username : this.fb.control( formState: ""),
16         password: this.fb.control( formState: "")
17     })
18 }
19
20 handleLogin() {
21     console.log(this.formLogin.value)
22 }
23 }
```

LoginComponent > ngOnInit()

Après on ajoute la route de ce composant, puis on ajoute un path pour qu'il nous rediriger vers la page login si on fait rien :

```
app-routing.module.ts ×
1 import ...
7
8 const routes: Routes = [
9 {path : "home", component : HomeComponent},
10 {path : "products", component : ProductsComponent},
11 {path : "newProduct", component : NewProductComponent},
12 {path : "editProduct/:id", component : EditProductComponent},
13 {path : "", redirectTo : "login", pathMatch : "full"}]
14 ];
15
16 @NgModule({
17 imports: [RouterModule.forRoot(routes)],
18 exports: [RouterModule]
19 })
20 export class AppRoutingModule { }
```

On démarre l'application, on voit que notre formulaire a été bien afficher :

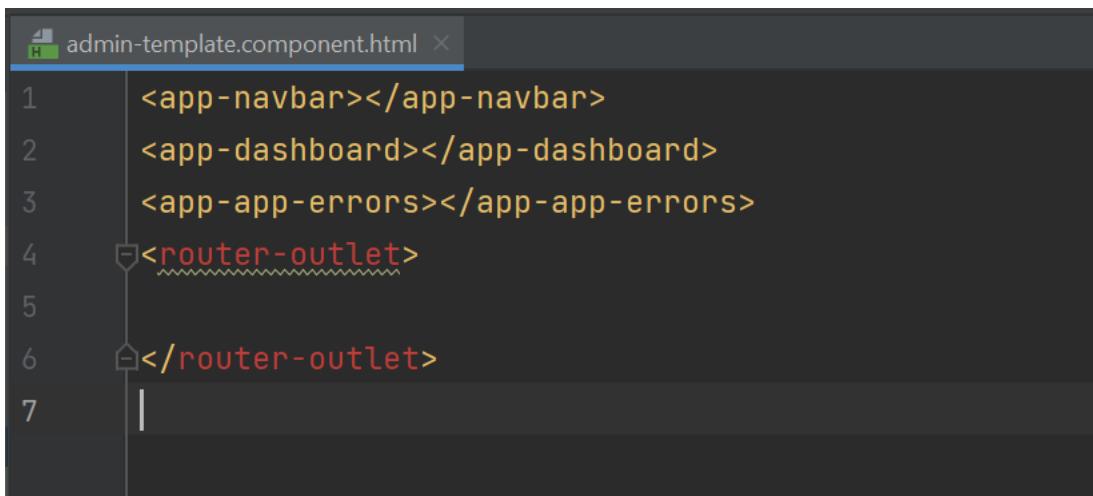
The screenshot shows a web browser window with the URL `localhost:4200/login`. At the top, there are navigation icons for back, forward, and refresh. Below the address bar, there are three buttons: `Home`, `Products`, and `New Product`. The main content area contains a horizontal row of four buttons labeled `Pages : 0`, `Pages : 3`, `Pages : 0`, and `Checked : 0`. Below this is a section titled `Authentication` with fields for `Username` and `Password`, and a `Login` button.

Le problème maintenant, c'est qu'on ne veut pas afficher les autres composants, lorsqu'on est dans la page d'authentification, ce qui fait, on va créer donc un composant ou bien une template qui concerne la partie admin, c'est-à-dire il y a une interface qui ne va être visible

qu'une fois nous avons authentifiées. Pour cela le plus simple c'est de créer un composant que l'on appelle admin-template :

```
c:\Users\lenovo\angular-projects\fsm-app>ng g c admin-template
CREATE src/app/admin-template/admin-template.component.html (30 bytes)
CREATE src/app/admin-template/admin-template.component.spec.ts (674 bytes)
CREATE src/app/admin-template/admin-template.component.ts (240 bytes)
CREATE src/app/admin-template/admin-template.component.css (0 bytes)
UPDATE src/app/app.module.ts (1925 bytes)
```

Puis on va supprimer le header de notre fichier app.component.html et on le replace dans le fichier html de notre composant admin-template :



```
admin-template.component.html ×
1  <app-navbar></app-navbar>
2  <app-dashboard></app-dashboard>
3  <app-app-errors></app-app-errors>
4  <router-outlet>
5
6  </router-outlet>
7  |
```

Alors on veut afficher que le formulaire d'authentification, et celui-ci qui va nous permettre d'aller vers les autres pages, donc ce qui fait on va changer les routes, c'est à dire on va ajouter un path vers notre composant admin-template, et on remplace les autres à l'intérieur de ce composant on utilisant le tableau children :

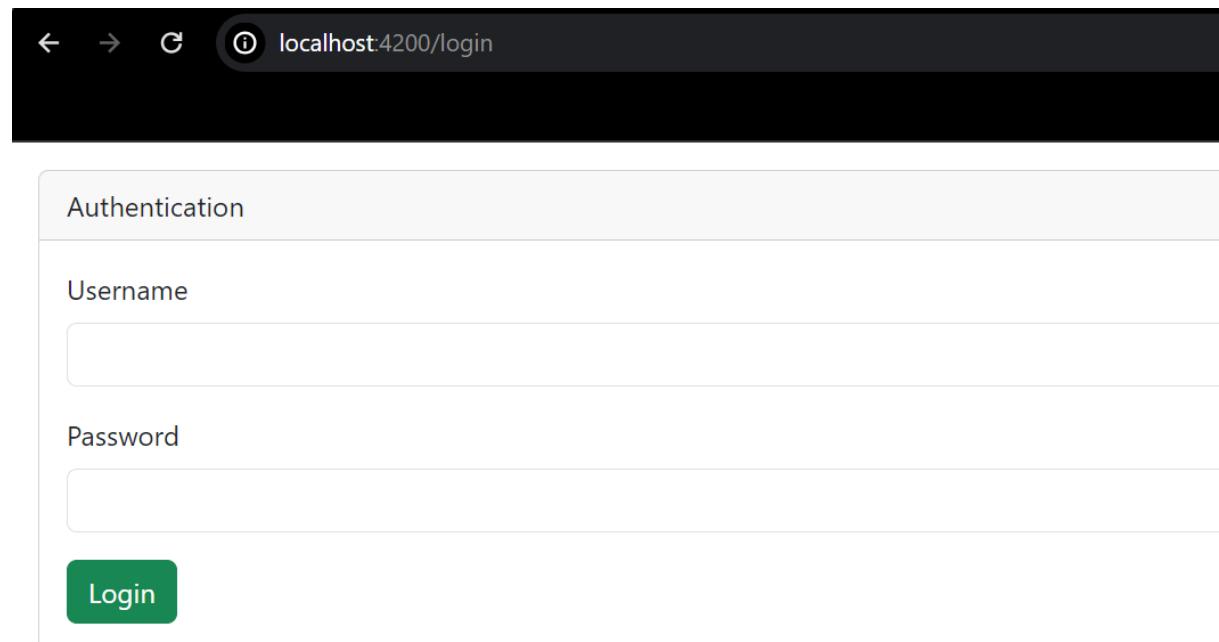
```

import { HomeComponent} from "./home/home.component";
import {ProductsComponent} from "./products/products.component";
import {NewProductComponent} from "./new-product/new-product.component";
import {EditProductComponent} from "./edit-product/edit-product.component";
import {AdminTemplateComponent} from "./admin-template/admin-template.component";
import { LoginComponent} from "./login/login.component";

const routes: Routes = [
  {path : "login", component : LoginComponent},
  {
    path : "admin", component : AdminTemplateComponent, children : [
      {path : "products", component: ProductsComponent},
      {path : "newProduct", component : NewProductComponent},
      {path : "editProduct/:id", component: EditProductComponent}
    ],
  },
  {path : "home", component : HomeComponent},
  {path : "products", component : ProductsComponent},
]
routes > children > component

```

On redémarre l'application, pour voir que nous avons le formulaire sans header, et que pour accéder à un composant il faut créer /admin/composant :



The screenshot shows a web browser window with the following details:

- Address Bar:** localhost:4200/login
- Page Title:** Authentication
- Form Fields:**
 - Username: An input field.
 - Password: An input field.
- Buttons:**
 - A green rectangular button labeled "Login".

localhost:4200/admin/products					
Home		Products		New Product	
<hr/>					
Pages :4	Size :3	Total :10	Checked :3		
<hr/>					
<input type="text"/> <input type="button" value="Search"/>					
<hr/>					
Name	Price	Checked			
Computer	5600	<input checked="" type="checkbox"/>	<input type="button" value="Delete"/>	<input type="button" value="Edit"/>	
New Printer	3200	<input checked="" type="checkbox"/>	<input type="button" value="Delete"/>	<input type="button" value="Edit"/>	
Smart	600	<input checked="" type="checkbox"/>	<input type="button" value="Delete"/>	<input type="button" value="Edit"/>	
<hr/>					
<input type="button" value="1"/>	<input type="button" value="2"/>	<input type="button" value="3"/>	<input type="button" value="4"/>		
<hr/>					

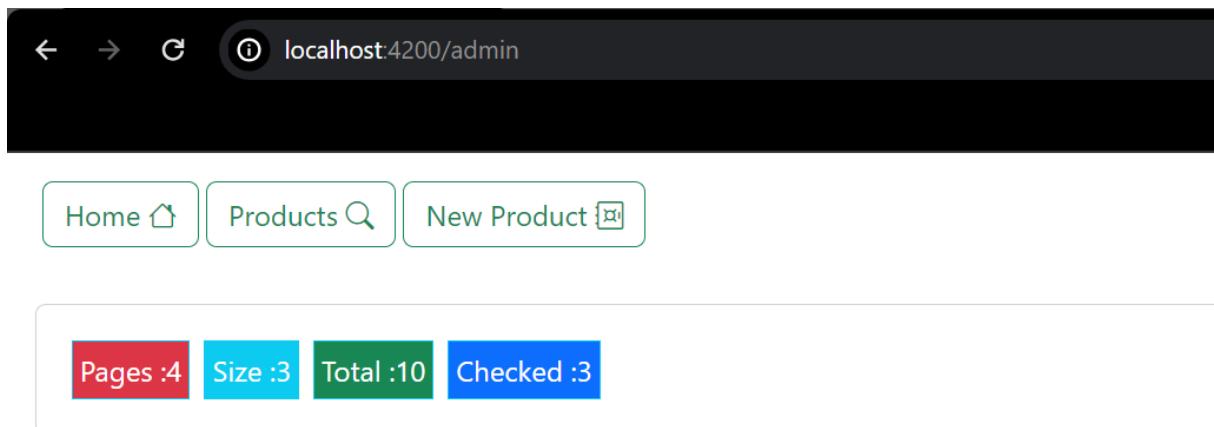
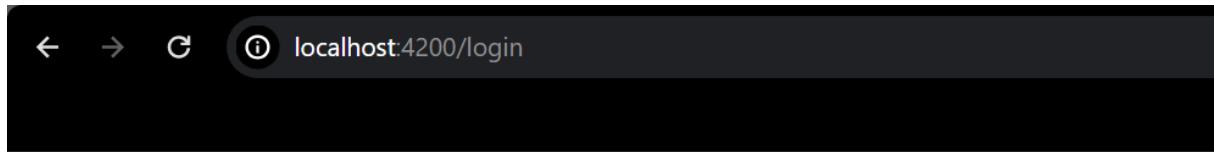
Mais la suppression et la modification ne fonctionne pas, car on doit modifier les URLs en ajoutant le /admin (on va le faire après), alors ça c'est la première de chose, alors maintenant on va vers notre login et en fait un test que lorsqu'on fait l'authentification avec username admin et password 123 et on clique sur le bouton il affiche le composant admin-template

```

ngOnInit(): void {
  this.formLogin = this.fb.group( controls: {
    username : this.fb.control( formState: ""),
    password: this.fb.control( formState: "")
  })
}

handleLogin() {
  console.log(this.formLogin.value)
  if(this.formLogin.value.username == "admin" && this.formLogin.value.password=="1234"){
    this.router.navigateByUrl( url: "/admin")
  }
}

```



Mais on voie que les boutons ne fonctionnent pas, donc c'est le temps de changer les routes :

```
navbar.component.ts ×
6   selector: 'app-navbar',
7   templateUrl: './navbar.component.html',
8   styleUrls: ['./navbar.component.css']
9  })
10 export class NavbarComponent {
11   actions: Array<any> = [
12     {title: "Home", "route": "/home", icon:"house" },
13     {title: "Products", "route": "/admin/products", icon:"search" },
14     {title: "New Product", "route": "/admin/newProduct", icon:"safe" }
15   ];
16   currentAction: any;
17   public isLoading : boolean = false;
18   constructor(public appState : AppStateService,
19             public loadingService : LoadingService) {
20     /* this.loadingService.isLoading$.subscribe({
21       next : (value ) =>{
22         this.isLoading = value;
23       }
24     }) */
25   }
  NavbarComponent > constructor()
```

```
products.component.ts ×
77   // this.getProducts();
78   //this.appState.productsState.products =
79   // this.appState.productsState.products.filter((p : any) => p.id != product.id);
80   this.searchProducts();
81 }
82 }
83 }
84
85 handleGotoPage(page: number) {
86   this.currentPage = page;
87   this.searchProducts();
88 }
89
90 handleEdit(product : Product) {
91   this.router.navigateByUrl(url: `/admin/editProduct/${product.id}`)
92 }
```

Donc on redémarre l'application, pour voir que tout se passe bien :

Product List			
Name	Price	Checked	Action
Computer	5600	<input checked="" type="checkbox"/>	trash edit
New Printer	3200	<input checked="" type="checkbox"/>	trash edit
Smart	600	<input checked="" type="checkbox"/>	trash edit

Pages :4 Size :3 Total :10 Checked :3

Home Products New Product

Jusqu'à maintenant on n'a pas fait l'authentification car on n'a pas protégé, ce que nous allons faire à la suite en utilisant les guards.

Le guard c'est un service Angular qui va tout simplement vérifier si les routes nécessitent une condition.

Maintenant on va vers notre fichier db.json pour ajouter une collection des utilisateurs qui ont le droit d'accéder à l'application, on utilise les tokens qui contiennent le jwt token qu'on a pris du site jwt.io et on a encodé le password avec base64 :

```
db.json x
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

"users": [
  {
    "id": "user1",
    "password": "MTIzNA==",
    "roles": ["USER"],
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyMSIsImlhCI6MTUxNjIzD"
  },
  {
    "id": "user2",
    "password": "MTIzNA==",
    "roles": ["USER"],
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyMiIsImlhCI6MTUxNjIzD"
  },
  {
    "id": "admin",
    "password": "MTIzNA==",
    "roles": ["USER", "ADMIN"],
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZG1pbkiIsImlhCI6MTUxNjIzD"
  }
]
```

Voilà la liste des utilisateurs sous format json :



```
Impression élégante 
{
  "id": "user1",
  "password": "MTIzNA==",
  "roles": [
    "USER"
  ],
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyMSIsImlhhdCl6MTUxNjIzMjOTAyMiwiZXhwIjoxNTE2MjM5MDIyLCJpc3MiOiJodHRwOi8vbG9jYWxob3N0Ojg4ODgvYXV0aClIsIm5iZiI6MTUxNjIzOTAyMiwiZmlyc3RuYW1IjoiTWWVyeWVtliwbGFzdG5hbWUiOjJTExBliwiZW1haWwiOiJtZXJ5ZW1AZ21haWwuY29tIiwiem9sZXMiOlsiVVNFUiJdfQ.loJdx4bITj9xKqhHYohHF9h2G8NgxrcVQ4RxeS2wyQ"
},
{
  "id": "user2",
  "password": "MTIzNA==",
  "roles": [
    "USER"
  ],
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyMSIsImlhhdCl6MTUxNjIzMjOTAyMiwiZXhwIjoxNTE2MjM5MDIyLCJpc3MiOiJodHRwOi8vbG9jYWxob3N0Ojg4ODgvYXV0aClIsIm5iZiI6MTUxNjIzOTAyMiwiZmlyc3RuYW1IjoiTWWVyeWVtliwbGFzdG5hbWUiOjJTExBliwiZW1haWwiOiJtZXJ5ZW1AZ21haWwuY29tIiwiem9sZXMiOlsiVVNFUiJdfQ.deLwW7FY5U7mMbUDTYwpAbjo5DFnIEpynGYbsGnE0"
},
{
  "id": "admin",
  "password": "MTIzNA==",
  "roles": [
    "USER",
    "ADMIN"
  ],
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZG1pbisImlhhdCl6MTUxNjIzMjOTAyMiwiZXhwIjoxNTE2MjM5MDIyLCJpc3MiOiJodHRwOi8vbG9jYWxob3N0Ojg4ODgvYXV0aClIsIm5iZiI6MTUxNjIzOTAyMiwiZmlyc3RuYW1IjoiTWWVyeWVtliwbGFzdG5hbWUiOjJTExBliwiZW1haWwiOiJtZXJ5ZW1AZ21haWwuY29tIiwiem9sZXMiOlsiVVNFUiJdfQ.IkFETUOI19.QyOUUGJ8oHP"
}
}
```

Maintenant on va créer un autre service que l'on va appeler auth :

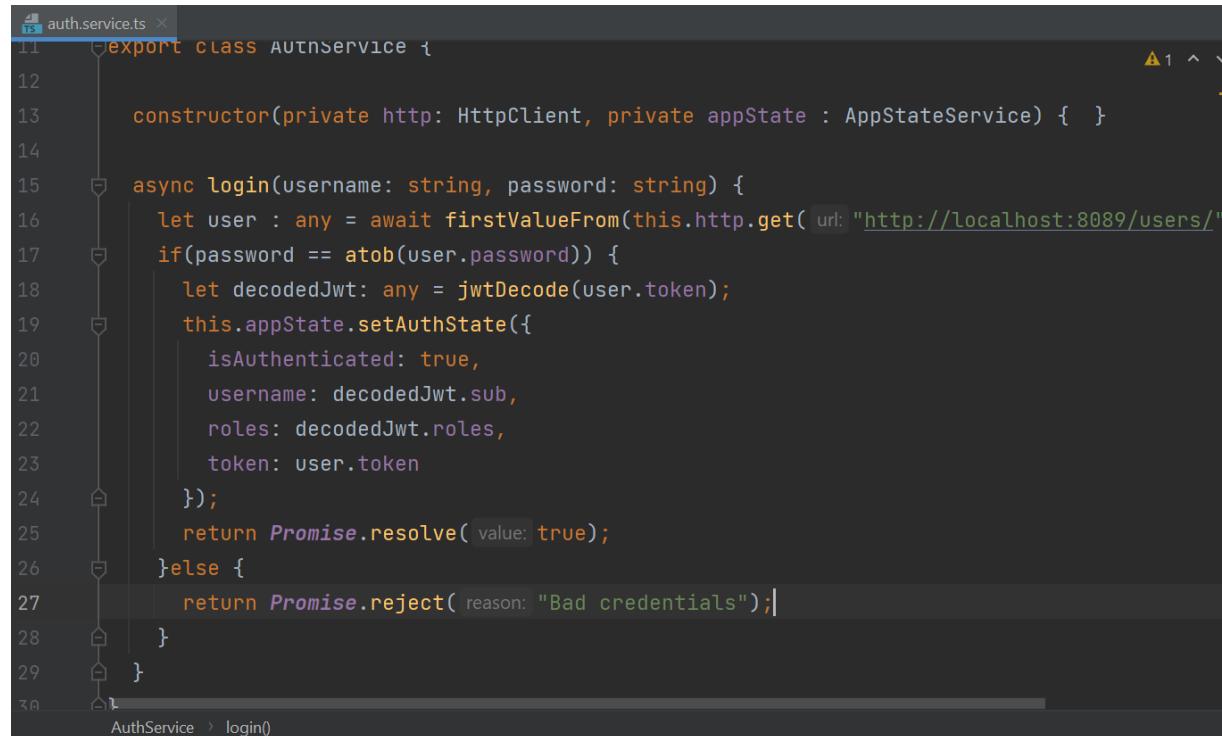
```
C:\Users\lenovo\angular-projects\fsm-app>ng g s auth
CREATE src/app/auth.service.spec.ts (363 bytes)
CREATE src/app/auth.service.ts (142 bytes)
```

On passe pour créer une méthode « async » que l'on appelle login, et qu'il attend à récupérer l'utilisateur en utilisant le mot « await », et le « firstValueFrom » qui rend l'observable en promise, et après il décode le mot de passe avec « atob », puis le vérifier, ensuite on utilise jwtDecode après qu'on l'installe pour décoder notre token afin de récupérer nos données, si le mot de passe est correct tout se passe bien, sinon un message d'erreur se lever.

Voilà d'abord comment installer jwt-decoder :

```
Terminal: Local × Local (2) × Local (3) × +  
  
C:\Users\lenovo\angular-projects\fsm-app>npm i jwt-decode  
  
added 1 package, and audited 928 packages in 4s  
  
121 packages are looking for funding  
  run `npm fund` for details  
  
1 high severity vulnerability  
  
To address all issues, run:  
  npm audit fix  
  
Run `npm audit` for details.
```

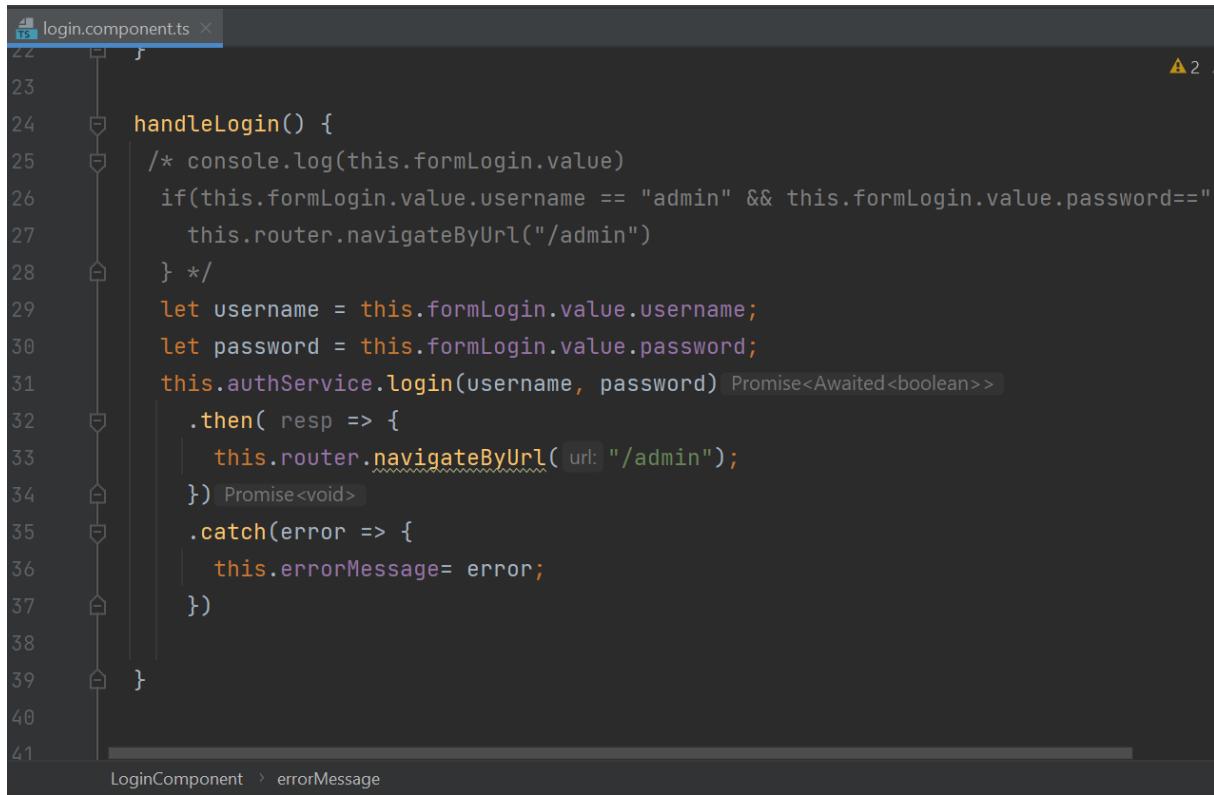
Et voilà le code de cette partie :



The screenshot shows a code editor window with a dark theme. The file being edited is `auth.service.ts`. The code defines a class `AutnService` with a constructor taking `HttpClient` and `AppStateService`. It contains an asynchronous `login` method that sends a GET request to `http://localhost:8089/users/` with the provided `username` and `password`. The password is decoded using `atob`. If the password matches the user's stored password, the user's token is decoded using `jwtDecode`, and the `AppStateService` is updated with the user's authenticated status, username, roles, and token. A `Promise.resolve(true)` is returned if successful. If the password does not match, a `Promise.reject("Bad credentials")` is returned. The code editor shows syntax highlighting for keywords like `export`, `class`, `async`, and `private`, as well as variable names and comments.

```
11 export class AutnService {  
12   constructor(private http: HttpClient, private appState : AppStateService) { }  
13  
14   async login(username: string, password: string) {  
15     let user : any = await firstValueFrom(this.http.get( url: "http://localhost:8089/users/"  
16     if(password == atob(user.password)) {  
17       let decodedJwt: any = jwtDecode(user.token);  
18       this.appState.setAuthState({  
19         isAuthenticated: true,  
20         username: decodedJwt.sub,  
21         roles: decodedJwt.roles,  
22         token: user.token  
23       });  
24       return Promise.resolve( value: true);  
25     }else {  
26       return Promise.reject( reason: "Bad credentials");  
27     }  
28   }  
29 }
```

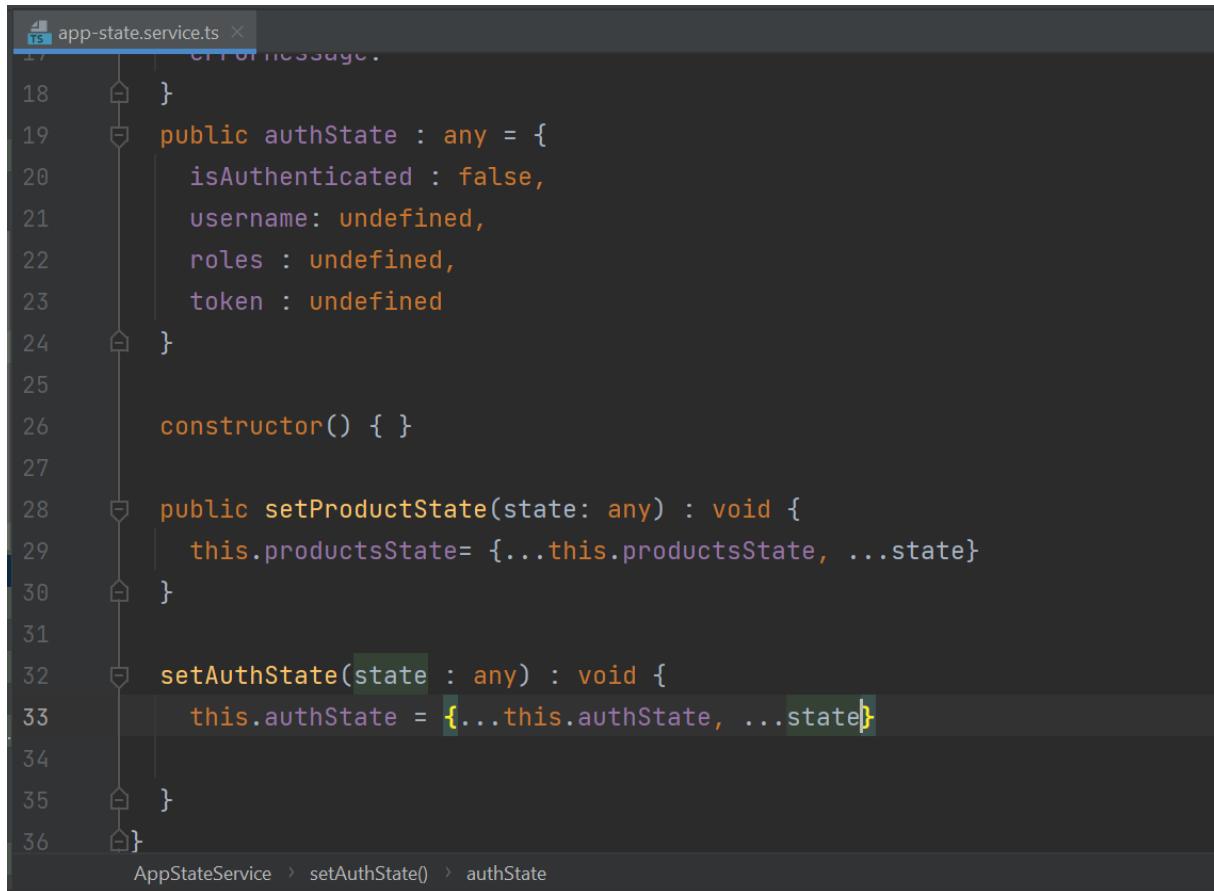
Après on va vers le login, et on injecte notre authService, et on spécifie la route que l'on diriger vers si le tout est bien, sinon on récupère le message qu'il va s'afficher :



```
login.component.ts
22
23
24     handleLogin() {
25         /* console.log(this.formLogin.value)
26         if(this.formLogin.value.username == "admin" && this.formLogin.value.password=="123456")
27             this.router.navigateByUrl("/admin")
28         }*/
29         let username = this.formLogin.value.username;
30         let password = this.formLogin.value.password;
31         this.authService.login(username, password) Promise<Awaited<boolean>>
32             .then( resp => {
33                 this.router.navigateByUrl(url: "/admin");
34             }) Promise<void>
35             .catch(error => {
36                 this.errorMessage= error;
37             })
38
39     }
40
41
```

LoginComponent > errorMessage

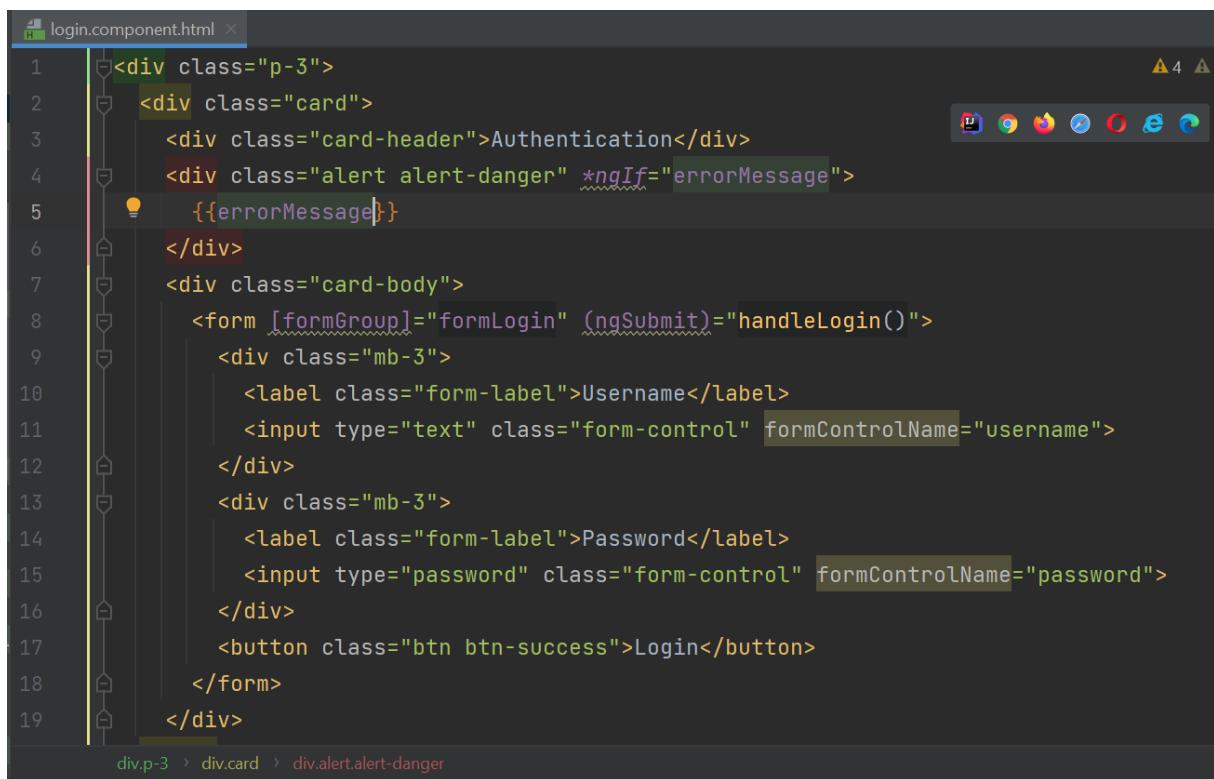
On va maintenant pour ajouter l'authState dans le fichier app-state.service, donc on déclare une variable appState dans laquelle on va stocker les informations de l'utilisateur, et on va créer aussi une méthode setAuthState dans laquelle on va créer une copie et on va ajouter les attributs qui se trouve dans state :



```
17     errorMessage.
18   }
19   public authState : any = {
20     isAuthenticated : false,
21     username: undefined,
22     roles : undefined,
23     token : undefined
24   }
25
26   constructor() { }
27
28   public setProductState(state: any) : void {
29     this.productsState= {...this.productsState, ...state}
30   }
31
32   setAuthState(state : any) : void {
33     this.authState = {...this.authState, ...state}
34   }
35 }
36 }
```

AppStateService > setAuthState() > authState

On ajoute quelque code dans le fichier html du composant login, pour qu'il affiche le message d'erreur :



```
1 <div class="p-3">
2   <div class="card">
3     <div class="card-header">Authentication</div>
4     <div class="alert alert-danger" *ngIf="errorMessage">
5       {{errorMessage}}
6     </div>
7     <div class="card-body">
8       <form [formGroup]="formLogin" (ngSubmit)="handleLogin()">
9         <div class="mb-3">
10           <label class="form-label">Username</label>
11           <input type="text" class="form-control" formControlName="username">
12         </div>
13         <div class="mb-3">
14           <label class="form-label">Password</label>
15           <input type="password" class="form-control" formControlName="password">
16         </div>
17         <button class="btn btn-success">Login</button>
18       </form>
19     </div>
```

div.p-3 > div.card > div.alert.alert-danger

On démarre l'application, et on essaye de lui donner un mot de passe incorrect :

Authentication

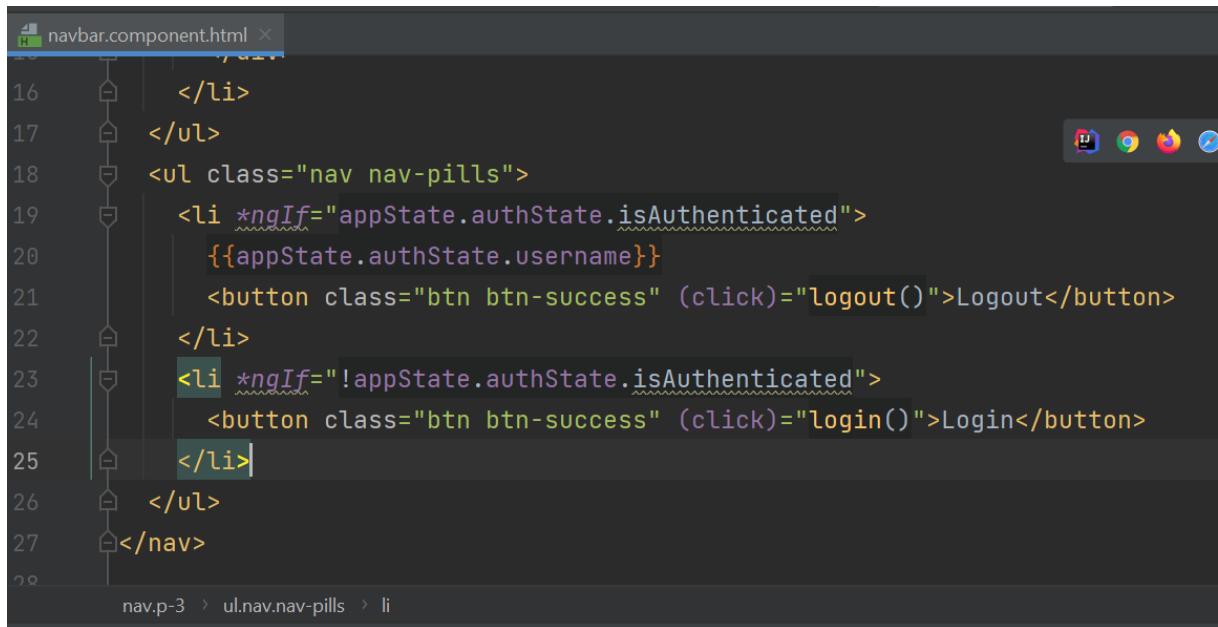
Bad credentials

Username

Password

Login

Maintenant on veut afficher à côté les informations de l'utilisateur qu'il est connecté, pour cela on va vers le navbar et on fait un test, c'est que si l'utilisateur est connecté on affiche un bouton logout sinon on affiche login :



```

16   </li>
17 </ul>
18 <ul class="nav nav-pills">
19   <li *ngIf="appState.authState.isAuthenticated">
20     {{appState.authState.username}}
21     <button class="btn btn-success" (click)="logout()>Logout</button>
22   </li>
23   <li *ngIf="!appState.authState.isAuthenticated">
24     <button class="btn btn-success" (click)="login()>Login</button>
25   </li>
26 </ul>
27 </nav>
28
  nav,p-3  > ul.nav.nav-pills  > li

```

Puis on définit les méthodes logout et login :

```
navbar.component.ts
```

```
30
31    }
32
33    logout() {
34        this.appState.authState= {};
35        this.router.navigateByUrl( url: "/logout");
36
37    }
38
39    login() {
40        this.router.navigateByUrl( url: "/login")
41
42    }
43
44}
```

NavbarComponent > logout()

Voilà comment s'affiche :

The screenshot shows a web application interface for managing products. At the top, the URL bar shows `localhost:4200/admin/products`. The header includes navigation links for `Home`, `Products`, and `New Product`, along with a user session indicator for `admin` and a `Logout` button. Below the header, there are filters: `Pages :4`, `Size :3`, `Total :10`, and `Checked :3`. The main content area displays a table of products:

Name	Price	Checked		
Computer	5600	<input checked="" type="checkbox"/>		
New Printer	3200	<input checked="" type="checkbox"/>		
Smart	600	<input checked="" type="checkbox"/>		

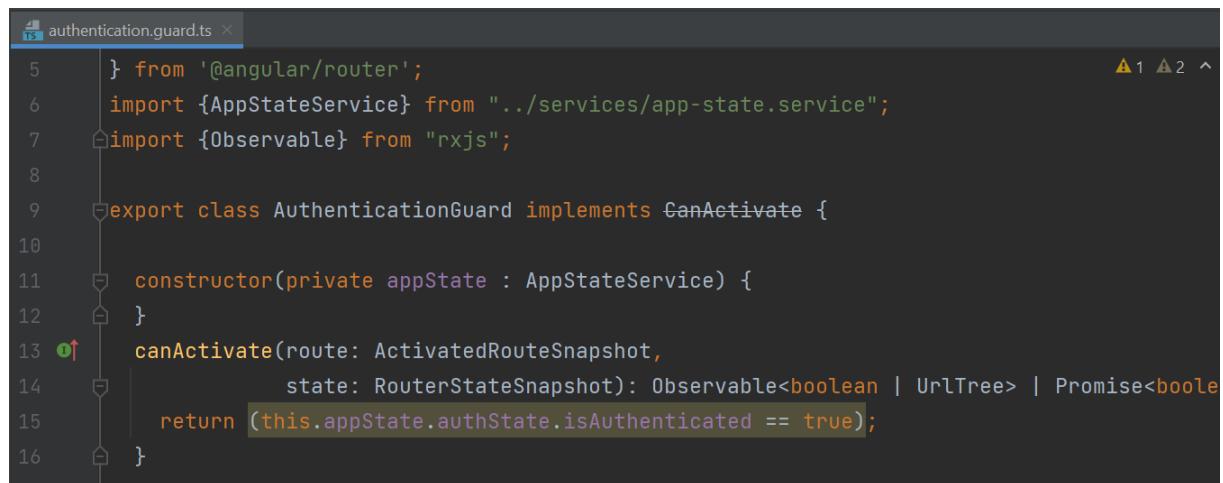
At the bottom of the table, there are navigation buttons labeled 1, 2, 3, and 4, with the number 1 highlighted in green.

C'est le temps maintenant de protéger les routes par des autorisations, donc là qu'on va utiliser les guards, allons pour les créer :

```
C:\Users\lenovo\angular-projects\fsm-app>ng g g guards/authentication
? Which type of guard would you like to create? CanActivate
CREATE src/app/guards/authentication.guard.spec.ts (518 bytes)
CREATE src/app/guards/authentication.guard.ts (143 bytes)
```

```
C:\Users\lenovo\angular-projects\fsm-app>ng g g guards/authorization
? Which type of guard would you like to create? CanActivate
CREATE src/app/guards/authorization.guard.spec.ts (514 bytes)
CREATE src/app/guards/authorization.guard.ts (142 bytes)
```

On va vers authentication guard et on retourne l'utilisateur s'il est authentifié :



```
authentication.guard.ts
5 } from '@angular/router';
6 import {AppStateService} from "../services/app-state.service";
7 import {Observable} from "rxjs";
8
9 export class AuthenticationGuard implements CanActivate {
10
11   constructor(private appState : AppStateService) {
12   }
13   canActivate(route: ActivatedRouteSnapshot,
14     state: RouterStateSnapshot): Observable<boolean | UrlTree | Promise<boolean> {
15     return (this.appState.authState.isAuthenticated == true);
16   }
}
```

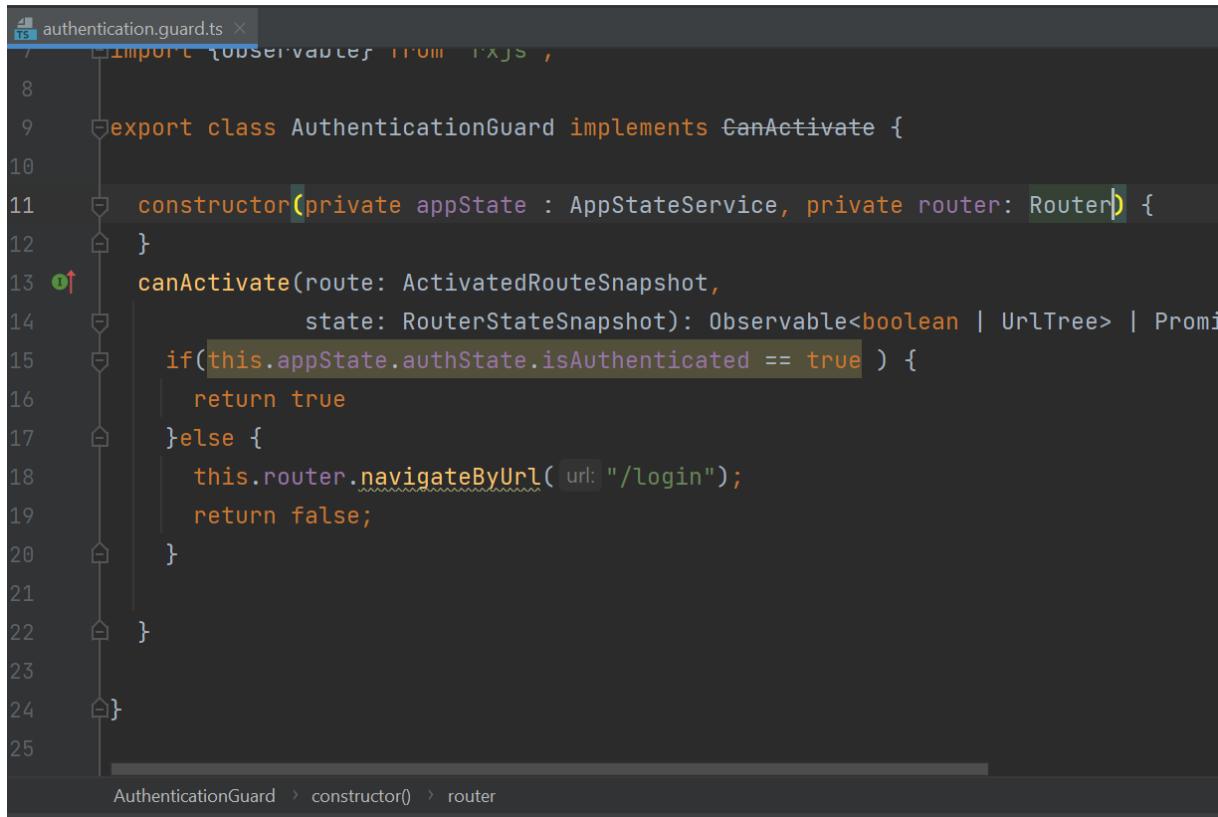
Pour pouvoir utiliser ce guard, on va vers le module des routes, et on ajoute can activate à notre route admin, pour que nous disions que cette route est protégée avec ce guard :



```
app-routing.module.ts
7   implements HomeComponent, AdminTemplateComponent, AdminTemplateComponent,
8   LoginComponent} from "./login/login.component";
9   AuthenticationGuard} from "./guards/authentication.guard";
10
11   routes: Routes = [
12     {"login", component : LoginComponent},
13
14     {"admin", component : AdminTemplateComponent, canActivate: [AuthenticationGuard], children: [
15       {"path : "products", component: ProductsComponent},
16       {"path : "newProduct", component : NewProductComponent},
17       {"path : "editProduct/:id", component: EditProductComponent}
18     ]
19   }
```

Pour les tester on va démarrer l'application et on tape /admin manuellement et on voie que ça ne marche pas

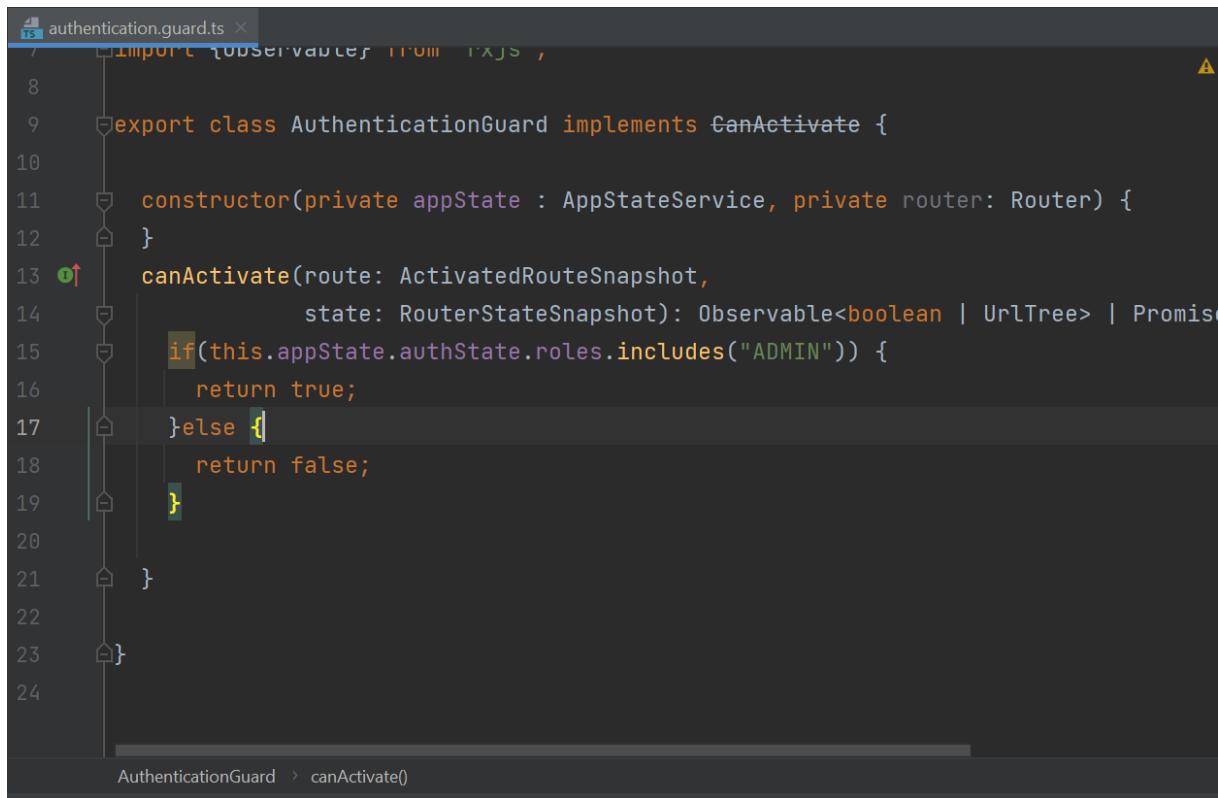
Maintenant on veut qu'il nous redirige vers la page login, au lieu d'afficher une page vide, pour ça on change un peu dans le guard :



```
authentication.guard.ts
1 import { UserService } from './UserService';
2
3 export class AuthenticationGuard implements CanActivate {
4   constructor(private appState : AppStateService, private router: Router) {}
5   canActivate(route: ActivatedRouteSnapshot,
6               state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> {
7     if(this.appState.authState.isAuthenticated == true) {
8       return true;
9     } else {
10       this.router.navigateByUrl('/login');
11       return false;
12     }
13   }
14 }
15
16
17
18
19
20
21
22
23
24 }
```

AuthenticationGuard > constructor() > router

Alors maintenant on passe pour qu'on donne le droit de supprimer, d'ajouter et d'éditer un produit qu'à l'admin, pour cela on va vers le guard authorization pour qu'on fait une vérification est ce que les rôles de l'utilisateur contient le rôle admin pour le donner le droit :



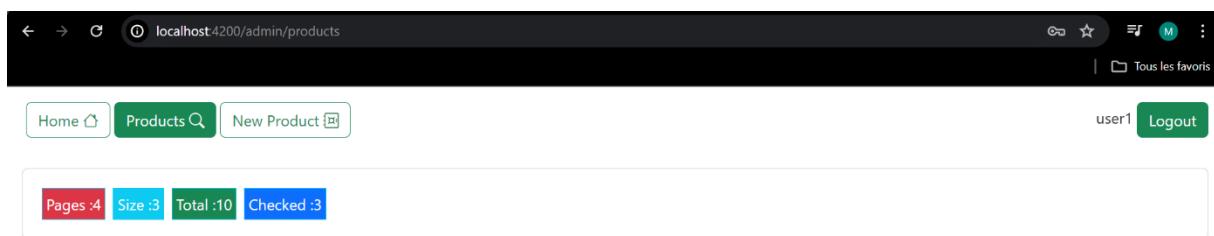
```
authentication.guard.ts
1 import { UserService } from './UserService';
2
3 export class AuthenticationGuard implements CanActivate {
4   constructor(private appState : AppStateService, private router: Router) {}
5   canActivate(route: ActivatedRouteSnapshot,
6               state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> {
7     if(this.appState.authState.roles.includes("ADMIN")) {
8       return true;
9     } else {
10       return false;
11     }
12   }
13 }
14
15
16
17
18
19
20
21
22
23
24 }
```

AuthenticationGuard > canActivate()

Dans le module des routes, on ajoute can Activate, à notre composant d'ajout :

```
app-routing.module.ts
7   {path: 'admin', component: AdminTemplateComponent, canActivate: [AuthenticationGuard], children: [
8     {inComponent: true, component: LoginComponent},
9     {path: 'login', component: LoginComponent},
10    {path: 'products', component: ProductsComponent},
11    {path: 'newProduct', component: NewProductComponent, canActivate: [AuthenticationGuard]},
12    {path: 'editProduct/:id', component: EditProductComponent}
13  ],
14  {path: 'home', component: HomeComponent},
15  {path: 'products', component: ProductsComponent},
16  {path: 'newProduct', component: NewProductComponent}
17 }
18
19
20
21
22
23
```

On teste l'application avec l'user1, et on voit qu'il ne lui donne pas le droit d'ajouter un produit :



Alors on va créer un composant que l'on appelle not-authorized :

```
C:\Users\lenovo\angular-projects\fsm-app>ng g c not-authorized
CREATE src/app/not-authorized/not-authorized.component.html (30 bytes)
CREATE src/app/not-authorized/not-authorized.component.spec.ts (674 bytes)
CREATE src/app/not-authorized/not-authorized.component.ts (240 bytes)
CREATE src/app/not-authorized/not-authorized.component.css (0 bytes)
UPDATE src/app/app.module.ts (2039 bytes)
```

Et on va créer un peu de code pour afficher un message d'erreur :

```
not-authorized.component.html
1 <div class="p-3">
2   <div class="card">
3     <div class="card-body">
4       <div class="alert alert-danger">
5         You are not authorized
6       </div>
7     </div>
8   </div>
9 </div>
10
```

On ajoute ce composant dans notre route admin :

```
app-routing.module.ts
11
12 const routes: Routes = [
13   {path : "login", component : LoginComponent},
14   {
15     path : "admin", component : AdminTemplateComponent, canActivate: [AuthenticationGuard]
16       {path : "products", component: ProductsComponent},
17       {path : "newProduct", component : NewProductComponent, canActivate: [AuthenticationGuard]}
18       {path : "editProduct/:id", component: EditProductComponent},
19       {path : "notAuthorized", component: NotAuthorizedComponent}
20   }],
21   {path : "", redirectTo : "login", pathMatch : "full"}
22 ];
23
```

Après on ajoute une ligne de code dans notre guard pour qu'il affiche le message d'erreur aux utilisateurs qui n'ont pas le droit d'ajouter un produit :

```
authorization.guard.ts
12 export class AuthorizationGuard implements CanActivate {
13   constructor(private appState : AppStateService, private router: Router) {
14   }
15
16   canActivate(route: ActivatedRouteSnapshot,
17             state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
18     if (this.appState.authState.roles.includes("ADMIN")) {
19       return true;
20     } else {
21       this.router.navigateByUrl(url: "/admin/notAuthorized")
22       return false;
23     }
24   }
25 }
26
```

Maintenant on passe pour faire la même chose pour l'édition :

```

12
13 const routes: Routes = [
14   {path : "login", component : LoginComponent},
15   {
16     path : "admin", component : AdminTemplateComponent, canActivate: [AuthenticationGuard], children : [
17       {path : "products", component: ProductsComponent},
18       {path : "newProduct", component : NewProductComponent, canActivate: [AuthenticationGuard]},
19       {path : "editProduct/:id", component: EditProductComponent, canActivate: [AuthorizationGuard]},
20       {path : "notAuthorized", component: NotAuthorizedComponent}
21     ],
22     {path : "", redirectTo : "login", pathMatch : "full"}
23   };
24 ]

```

Pour la suppression elle n'y a pas une route, donc va la masquer, aussi que pour l'édition et le check :

```

20   <td>{{product.price}}</td>
21   <td>
22     <button (click)="handleCheckProduct(product)"
23       class="btn btn-outline-success">
24       <i [class]="product.checked?'bi bi-check':'bi bi-circle'"></i>
25     </button>
26   </td>
27   <td *ngIf="appState.authState.roles.includes( value: 'ADMIN')">
28     <button (click)="handleDelete(product)" class="btn btn-outline-danger">
29       <i class="bi bi-trash"></i>
30     </button>
31   </td>
32   <td *ngIf="appState.authState.roles.includes( value: 'ADMIN')">
33     <button (click)="handleEdit(product)" class="btn btn-outline-success">
34       <i class="bi bi-pencil"></i>
35     </button>
36   </td>
37   </tr>
38 </tbody>

```

div.p-3 > div.card > div.card-body > table.table > tbody > tr > td

Voilà le résultat :

Name	Price	Checked
Computer	5600	
New Printer	3200	
Smart	600	

Pages: 4 | Size: 3 | Total: 10 | Checked: 3

1 2 3 4

On peut améliorer les guards pour ne pas écrire le rôle ADMIN ou autre, pour cela on créer une liste des rôles dans le module des routes :

```
12
13 const routes: Routes = [
14   {path : "login", component : LoginComponent},
15   {
16     path : "admin", component : AdminTemplateComponent, canActivate: [AuthenticationGuard],
17     {path : "products", component: ProductsComponent},
18     {path : "newProduct", component : NewProductComponent,
19       data : {requiredRoles : 'ADMIN'}
20     },
21     {path : "editProduct/:id", component: EditProductComponent,
22       data : {requiredRoles: 'ADMIN'}
23     },
24     {path : "notAuthorized", component: NotAuthorizedComponent}
25   ],
26   {path : "", redirectTo : "login", pathMatch : "full"}
27 ]
28 
```

Pour le guard maintenant, au lieu d'accéder à admin on va utiliser la route d'ActivatedRoute pour accéder à notre rôle :

```
12 export class AuthorizationGuard implements CanActivate {
13   constructor(private appState : AppStateService, private router: Router) {
14   }
15
16   canActivate(route: ActivatedRouteSnapshot,
17     state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | U
18   if (this.appState.authState.roles.includes(route.data['requiredRoles'])) {
19     return true;
20   } else {
21     this.router.navigateByUrl(url: "/admin/notAuthorized")
22     return false;
23   }
24 }
25
26 }
```

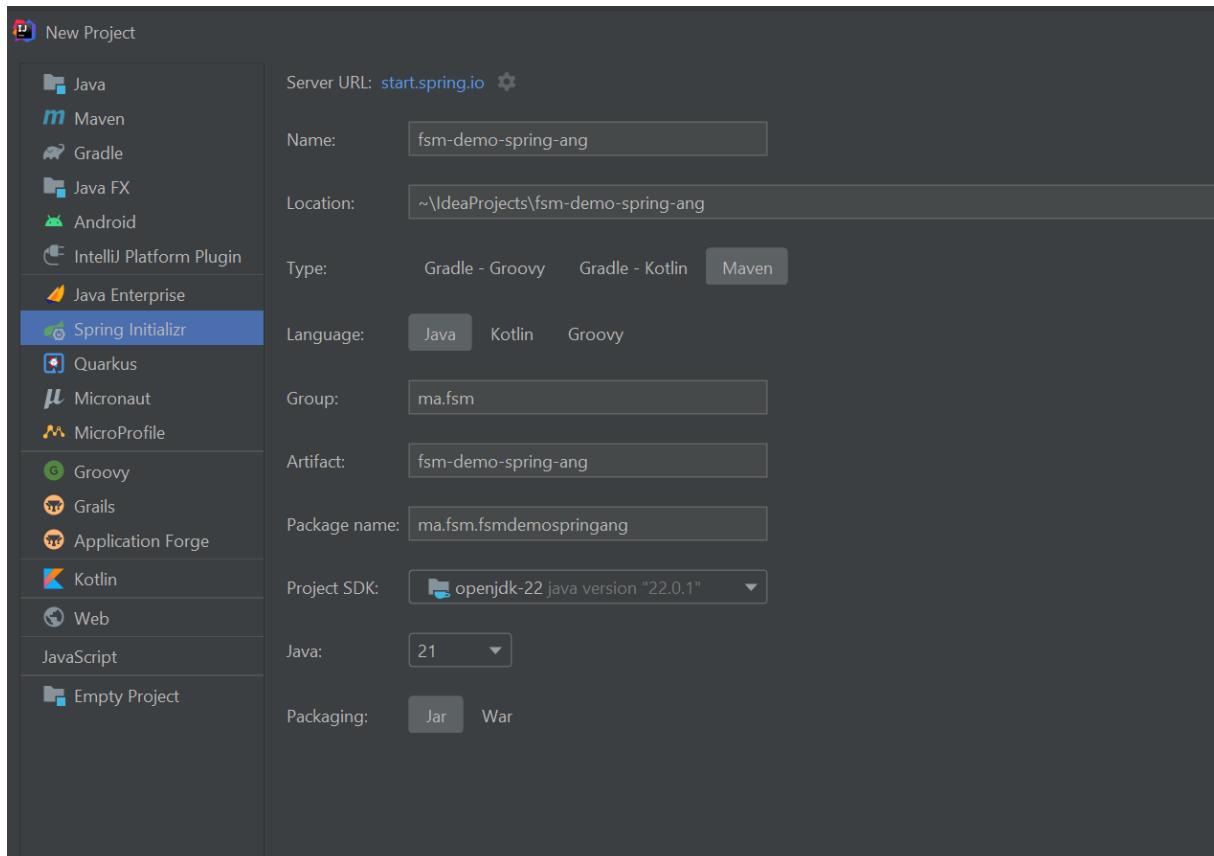
NB : tout ce qu'on fait dans cette partie est un faux backend, on a protégé juste le frontend

Troisième partie :

Partie 1 :

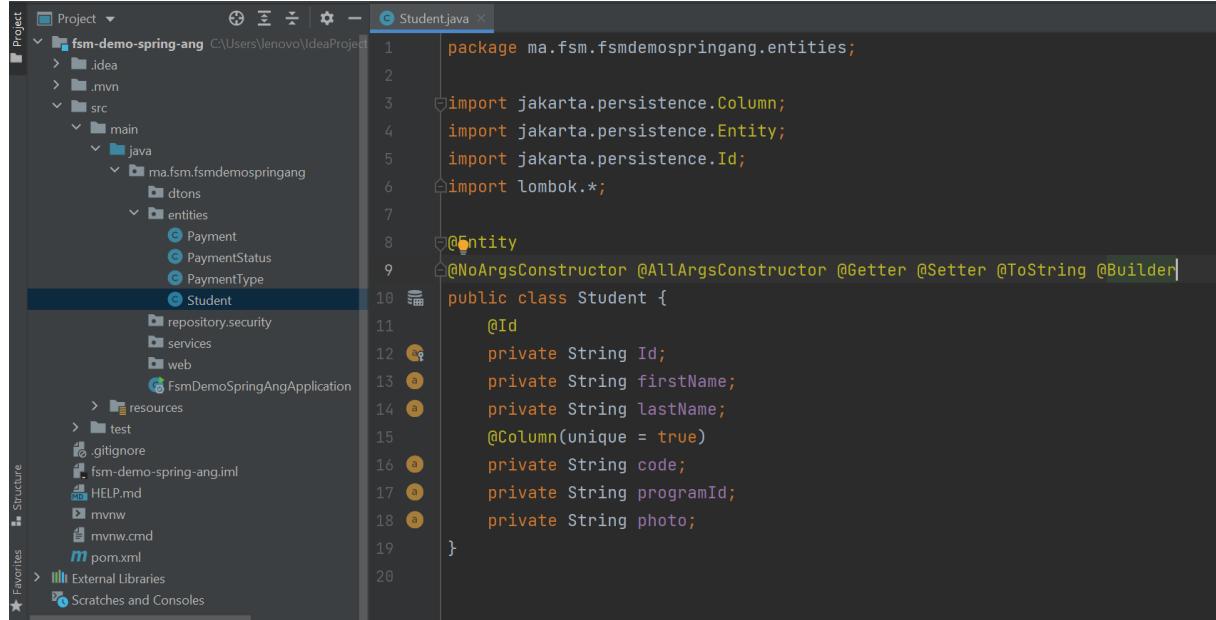
Dans cette partie, on va créer une autre application qui gère des étudiants qui ont fait plusieurs paiements, on travaille sur le backend puis le frontend et ensuite on va voir comment les combiner.

Tout d'abord on va créer l'application, et on ajoute les dépendances :



Puis on va créer les packages que nous avons besoin.

On commence par les entities, on a deux Student et Payment et pour cette dernière on a deux classes de type énumération, on a défini aussi les annotations que nous avons l'habitude d'utiliser :



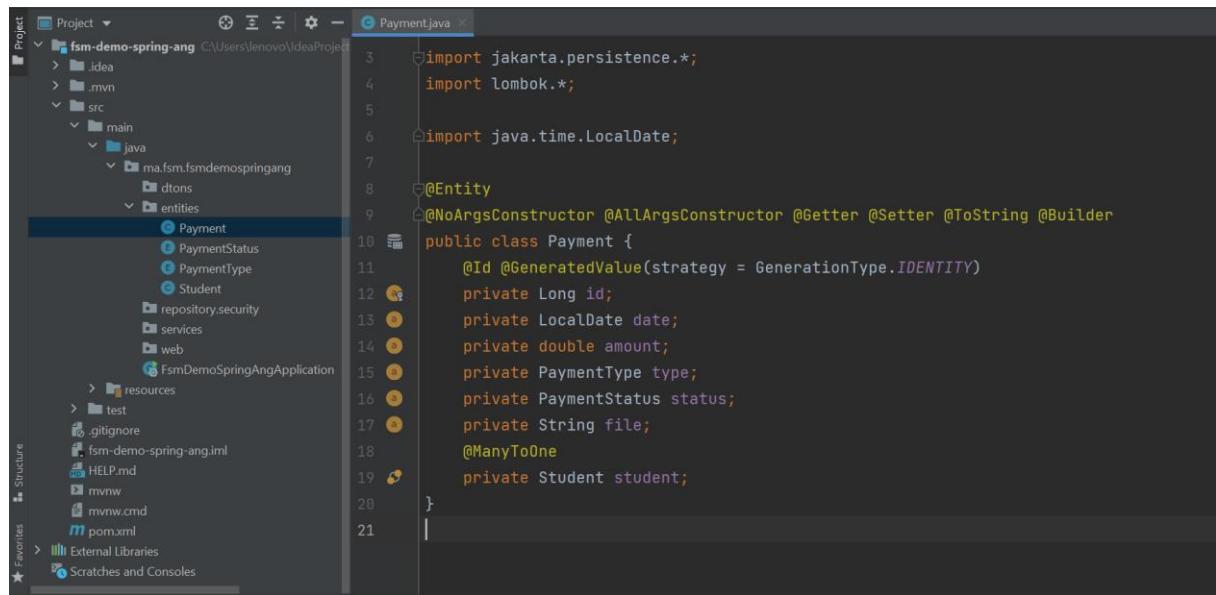
The screenshot shows the IntelliJ IDEA interface with the 'Student.java' file open in the editor. The code defines a class 'Student' with various fields and annotations. The project structure on the left shows the 'src' directory containing 'main' and 'java' packages, with 'entities' being the current active package. Other files like 'FsmDemoSpringAngApplication.java' and 'pom.xml' are also visible.

```
package ma.fsm.fsmdemospringang.entities;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import lombok.*;

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@ToString
@Builder
public class Student {

    @Id
    private String Id;
    private String firstName;
    private String lastName;
    @Column(unique = true)
    private String code;
    private String programId;
    private String photo;
}
```



The screenshot shows the IntelliJ IDEA interface with the 'Payment.java' file open in the editor. The code defines a class 'Payment' with various fields and annotations. The project structure on the left shows the 'src' directory containing 'main' and 'java' packages, with 'entities' being the current active package. Other files like 'FsmDemoSpringAngApplication.java' and 'pom.xml' are also visible.

```
import jakarta.persistence.*;
import lombok.*;
import java.time.LocalDate;

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@ToString
@Builder
public class Payment {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private LocalDate date;
    private double amount;
    private PaymentType type;
    private PaymentStatus status;
    private String file;
    @ManyToOne
    private Student student;
}
```

```

package ma.fsm.fsmdemospingang.entities;

public enum PaymentStatus {
    CREATED, VALIDATED, REJECTED
}

```

```

package ma.fsm.fsmdemospingang.entities;

public enum PaymentType {
    CASH, CHECK, TRANSFER, DEPOSIT
}

```

Après on va dans les repository pour gérer les étudiants, on va créer une interface StudentRepository, dans laquelle on va définir une méthode pour chercher un étudiant par son code, et une deuxième méthode qui va retourner la liste des étudiants d'une filière donnée :

```

package ma.fsm.fsmdemospingang.repository;

import ma.fsm.fsmdemospingang.entities.Student;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface StudentRepository extends JpaRepository<Student, String> {
    Student findByCode(String code);
    List<Student> findByNameId(String programId);
}

```

Puis de la même manière, on va créer une deuxième repository pour gérer les paiements, donc on va créer une interface PaymentRepository, dans laquelle on va définir une méthode pour retourner les paiements de l'étudiant, et une deuxième méthode pour retourner les paiements selon un status, et aussi une troisième pour chercher la liste des paiements par type :

```

1 package ma.fsm.fsmdemospringang.repository;
2
3 import ma.fsm.fsmdemospringang.entities.Payment;
4 import ma.fsm.fsmdemospringang.entities.PaymentStatus;
5 import ma.fsm.fsmdemospringang.entities.PaymentType;
6 import org.springframework.data.jpa.repository.JpaRepository;
7
8 import java.util.List;
9
10 public interface PaymentRepository extends JpaRepository<Payment, Long> {
11     List<Payment> findByStudentCode(String code);
12     List<Payment> findByStatus(PaymentStatus status);
13     List<Payment> findByType(PaymentType type);
14 }
15
16

```

Après on veut faire un test, dans notre application on va remplir d'abord notre base de données avec des données, puis on va créer des paiements pour chaque étudiant :

```

28     return args -> {
29         studentRepository.save(Student.builder().Id(UUID.randomUUID().toString())
30             .firstName("Meryeme").code("112233").programId("SDIA")
31             .build());
32         studentRepository.save(Student.builder().Id(UUID.randomUUID().toString())
33             .firstName("Asmae").code("112244").programId("SDIA")
34             .build());
35         studentRepository.save(Student.builder().Id(UUID.randomUUID().toString())
36             .firstName("Chaymae").code("112255").programId("IAAD")
37             .build());
38         studentRepository.save(Student.builder().Id(UUID.randomUUID().toString())
39             .firstName("Fatima").code("112266").programId("BDCC")
40             .build());
41
42         PaymentType[] paymentTypes = PaymentType.values();
43         Random random = new Random();
44         studentRepository.findAll().forEach(st -> {
45             for (int i=0; i<10; i++){
46                 int index = random.nextInt(paymentTypes.length);
47                 Payment payment = Payment.builder()
48                     .amount(1000+(int)(Math.random()*20000))
49                     .type(paymentTypes[index]);
50             }
51         });
52     }
53 }

```

Avant de tester on va configurer notre base de données :

```

1 spring.application.name=fsm-demo-spring-ang
2 spring.datasource.url=jdbc:h2:mem:students-db
3 spring.h2.console.enabled=true
4 server.port=8021
5

```

Et on va créer un web service, un Rest controller dans notre package web, on crée un RestController que l'on appelle PaymentRestController, dans lequel on va injecter StudentRepository et PaymentRepository via le constructeur, et on va créer une méthode pour consulter la liste de tous les paiements pour les retourner, et une pour consulter les paiements sachant l'id, et autre pour consulter tous les étudiants, et une pour consulter un étudiant sachant son code, autre pour consulter les étudiants d'une filière, et une pour les paiements de l'étudiants, et aussi une pour les paiements par status, et la dernière pour consulter les paiements par type :

```

1 package com.example.fsdemospringang.web;
2
3 import com.example.fsdemospringang.entities.Payment;
4 import com.example.fsdemospringang.repositories.PaymentRepository;
5 import com.example.fsdemospringang.repositories.StudentRepository;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.web.bind.annotation.*;
8
9 @RestController
10 @RequestMapping("/payments")
11 public class PaymentRestController {
12     @Autowired
13     private PaymentRepository paymentRepository;
14     @Autowired
15     private StudentRepository studentRepository;
16
17     @GetMapping(path = "/payments")
18     public List<Payment> paymentsByType(@RequestParam PaymentType type){
19         return paymentRepository.findByType(type);
20     }
21
22     @GetMapping(path = "/payments/{id}")
23     public Payment getPaymentById(@PathVariable Long id){
24         return paymentRepository.findById(id).get();
25     }
26
27     @GetMapping(path = "/students")
28     public List<Student> allStudents(){
29         return studentRepository.findAll();
30     }
31
32     @GetMapping(path = "/students/{code}")
33     public Student getStudentByCode(@PathVariable String code){
34         return studentRepository.findByCode(code);
35     }
36
37     @GetMapping(path = "/studentsByProgramId")
38     public List<Student> getStudentsByProgramId(@RequestParam String programId){
39         return studentRepository.findByProgramId(programId);
40     }
41 }

```

```

17  @RestController
18  public class PaymentRestController {
19      private StudentRepository studentRepository;
20      private PaymentRepository paymentRepository;
21
22      public PaymentRestController(StudentRepository studentRepository, PaymentRepository paymentRepository) {
23          this.studentRepository = studentRepository;
24          this.paymentRepository = paymentRepository;
25      }
26
27  }
28
29  @GetMapping(path = "/payments")
30  public List<Payment> allPayments() {
31      return paymentRepository.findAll();
32  }
33
34  @GetMapping(path = "/students/{code}/payments")
35  public List<Payment> paymentsByStudent(@PathVariable String code) {
36      return paymentRepository.findByStudentCode(code);
37  }
38

```

On passe pour tester l'application, on trouve une exception qui signifie que dans le mapping de notre Rest Controller on a une ambiguïté :

```

public List<Payment> paymentsByType(@RequestParam PaymentType type) {
    return paymentRepository.findByType(type);
}

@GetMapping(path = "/payments/{id}")
public Payment getPaymentById(@PathVariable Long id) {
}

```

Caused by: java.lang.IllegalStateException: Ambiguous mapping. Cannot map 'paymentRestController' method
ma.fsm.fsmdemospingang.web.PaymentRestController#paymentsByStatus(PaymentStatus)
to {GET [/payments]}: There is already 'paymentRestController' bean method
ma.fsm.fsmdemospingang.web.PaymentRestController#allPayments() mapped. <6 internal lines>
at java.base/java.util.LinkedHashMap.forEach(LinkedHashMap.java:986) ~[na:na] <5 internal lines>
at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.invokeInitMethods(AbstractAutowireCapableBeanFactory.java:174) ~[spring-beans-5.3.11.jar:5.3.11]
at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.java:149) ~[spring-beans-5.3.11.jar:5.3.11]
... 16 common frames omitted

Justement ce problème se lève car on a /payments utilisé plusieurs fois, donc allons pour régler le problème on changeant le mapping des méthodes paymentsByStatus, et PaymentsByType, puis on lance l'application pour voir qu'il n'a pas de conflit :

The screenshot shows an IDE interface with the following details:

- Project Structure:** Shows a Maven project structure with a .mvn folder and a src folder containing main, java, resources, and test folders.
- Code Editor:** Displays `PaymentRestController.java` with annotations for `@GetMapping` methods. The code includes methods for getting payments by status, type, and ID.
- Run Tab:** Shows the application is running as `FsmDemoSpringAngApplication`.
- Console Tab:** Displays application logs from 2024-06-14T20:19:44.089+02:00 to 2024-06-14T20:19:44.716+02:00. The logs include INFO and WARN messages related to the application's startup and configuration.

On teste l'application :

```

[{"firstName": "Meryeme", "lastName": null, "code": "112233", "programId": "SDIA", "photo": null, "id": "fb4aa54c-24a9-4cd5-b163-58dfe06f64ca"}, {"firstName": "Asmae", "lastName": null, "code": "112244", "programId": "SDIA", "photo": null, "id": "f1deca9af-71d1-4e86-a66d-0435235d0bd7"}, {"firstName": "Chaymae", "lastName": null, "code": "112255", "programId": "IAAD", "photo": null, "id": "269c6c04-b8e8-47aa-901c-8ddb43953c2e"}, {"firstName": "Fatima", "lastName": null, "code": "112266", "programId": "BDCC", "photo": null, "id": "7d51199a-b9ed-4a57-8769-b33f728921bd"}]

```

Pour tester les API Rest Full, on a un moyen beaucoup plus pratique c'est d'utiliser la documentation swagger, donc pour le faire on ajoute une dépendance à notre fichier pom.xml :

```

23     
```

```

24     
```

```

25         </dependency>
26         <!--<dependency>
27             <groupId>org.springframework.boot</groupId>
28             <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
29         </dependency>-->
30         <dependency>
31             <groupId>org.springdoc</groupId>
32             <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
33             <version>2.5.0</version>
34         </dependency>

```

On redémarre l'application et on va dans le serveur pour faire appel au swagger-ui.html :

The screenshot shows the Swagger UI interface for a REST API. At the top, it displays the URL `localhost:8021/swagger-ui/index.html`. Below the header, there's a navigation bar with the Swagger logo, the path `/v3/api-docs`, and a green "Explore" button. The main content area is titled "OpenAPI definition v0 OAS 3.0". Under the "Servers" section, the URL `http://localhost:8021 - Generated server url` is listed. The "payment-rest-controller" section contains three GET methods:

- `GET /students`
- `GET /studentsByProgramId`
- `GET /students/{code}`

Nous avons vu comment s'affiche, donc on passe pour tester toutes les méthodes, on cite ici juste un exemple, et on peut appliquer pour les autres :

The screenshot shows the Swagger UI interface for the `getStudentsByProgramId` method. The URL in the address bar is `localhost:8021/swagger-ui/index.html#/payment-rest-controller/getStudentsByProgramId`. The "Responses" section includes a "Curl" block with the following command:

```
curl -X 'GET' \
'http://localhost:8021/studentsByProgramId?programId=IAAD' \
-H 'accept: */*'
```

Below the curl command is the "Request URL" field containing `http://localhost:8021/studentsByProgramId?programId=IAAD`. The "Server response" section shows a 200 status code with a "Response body" block containing the following JSON:

```
[{"firstName": "Chavmae"}]
```

On va ajouter une méthode aussi qui permet de mettre à jour le statut de paiement, et on redémarre l'application :

```
PaymentRestController.java
64     @GetMapping(path = "/studentsByProgramId")
65     public List<Student> getStudentsByProgramId(@RequestParam String programId) {
66         return studentRepository.findByProgramId(programId);
67     }
68
69     @PutMapping("/payments/{id}")
70     public Payment updatePaymentStatus(@RequestParam PaymentStatus status, @PathVariable Long id)
71     {
72         Payment payment= paymentRepository.findById(id).get();
73         payment.setStatus(status);
74         return paymentRepository.save(payment);
75     }
76 }
```

Pour tester on actualise le server, et on modifier par exemple l'id 1 en VALIDATED :

The screenshot shows the Swagger UI interface for testing the `updatePaymentStatus` endpoint. The URL is `localhost:8021/swagger-ui/index.html#/payment-rest-controller/updatePaymentStatus`.

Request Body:

Name	Description
<code>status</code> * required	<code>VALIDATED</code>
<code>string</code> (query)	
<code>id</code> * required	<code>1</code>
<code>integer(\$int64)</code> (path)	

Buttons: Execute, Clear

Responses:

Curl:

```
curl -X 'PUT' \
'http://localhost:8021/payments/1?status=VALIDATED' \
-H 'accept: */*'
```

Request URL:

```
http://localhost:8021/payments/1?status=VALIDATED
```

Server response:

Code: 200

Response body:

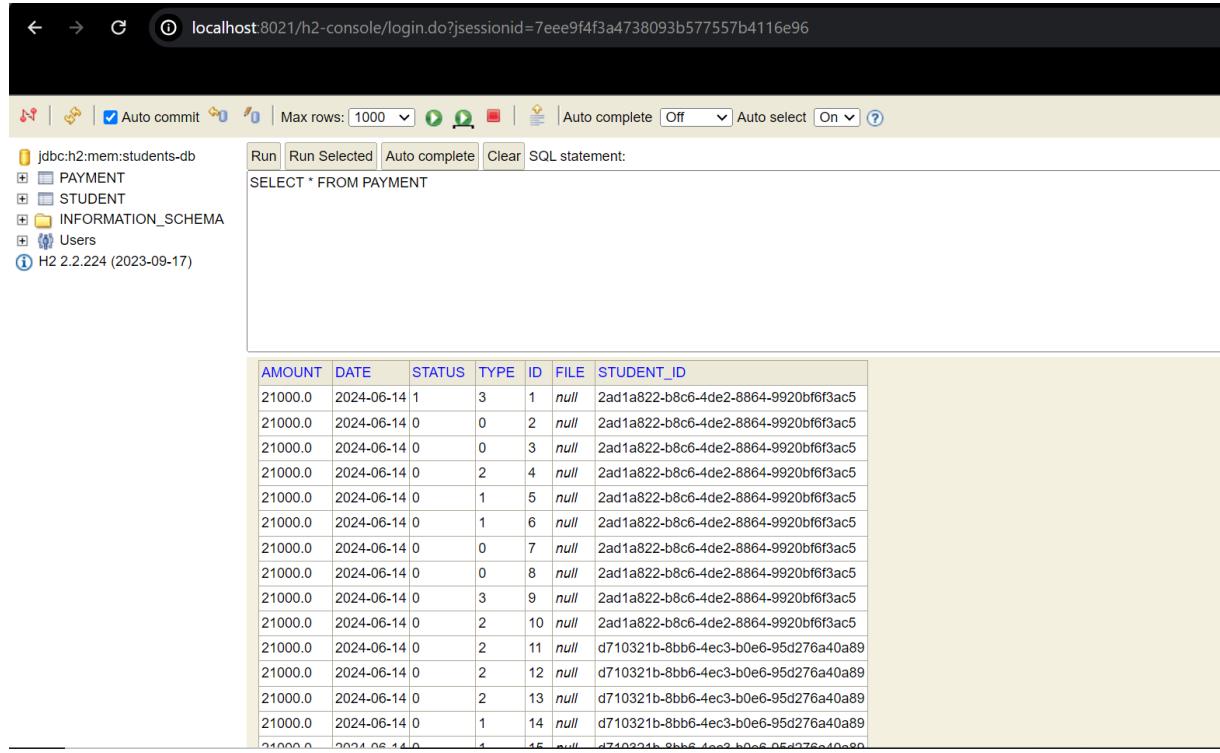
```
{
    "_id": 1,
    "date": "2024-06-14",
    "amount": 21000,
    "type": "DEPOSIT",
    "status": "VALIDATED",
    "file": null,
    "student": {
        "firstName": "Meryeme",
        "lastName": null,
        "code": "112233",
        "programId": "SDIA",
        "photo": null,
        "id": "2ad1a822-b8c6-4de2-8864-9920bf6f3ac5"
    }
}
```

Response headers:

```
connection: keep-alive
content-type: application/json
date: Fri, 14 Jun 2024 19:18:02 GMT
keepalive-timeout: 60
transfer-encoding: chunked
```

Responses:

Les données sont stockées dans la base de données h2, donc voilà on a bien les deux tables student et payment :



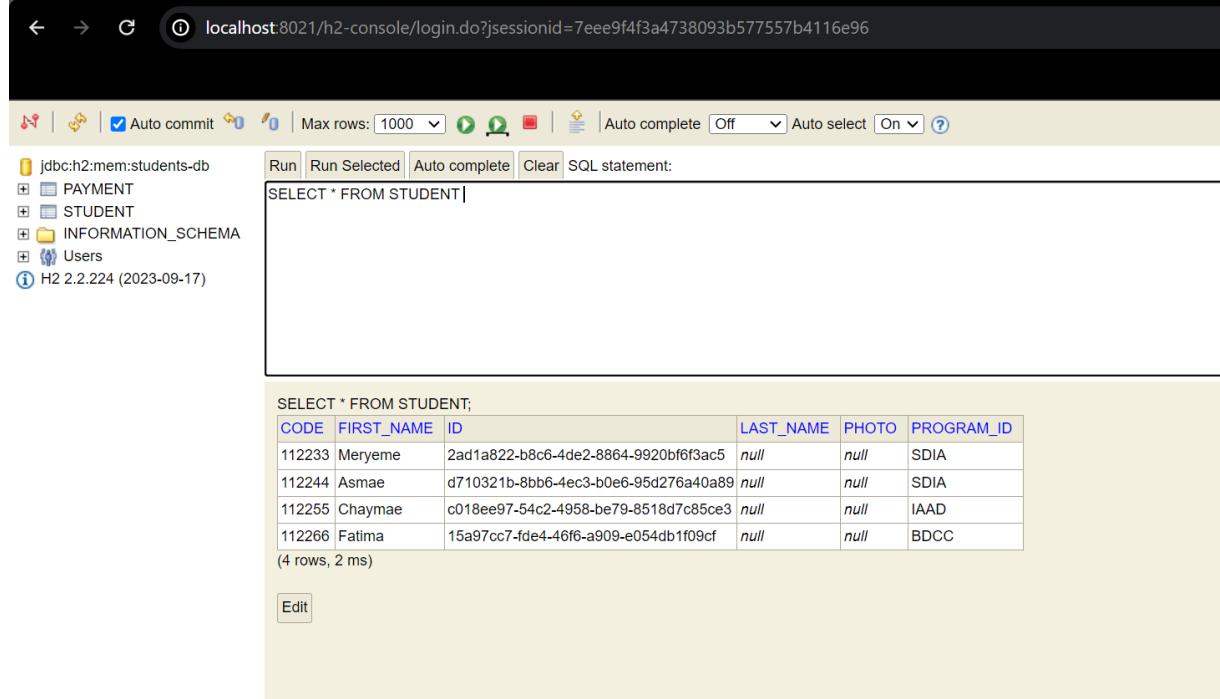
The screenshot shows the H2 Database Console interface. The left sidebar lists the database schema: jdbc:h2:mem:students-db, PAYMENT, STUDENT, INFORMATION_SCHEMA, and Users. The right pane contains a SQL statement editor and a results grid.

SQL Statement:

```
SELECT * FROM PAYMENT
```

Results:

AMOUNT	DATE	STATUS	TYPE	ID	FILE	STUDENT_ID
21000.0	2024-06-14	1	3	1	null	2ad1a822-b8c6-4de2-8864-9920bf6f3ac5
21000.0	2024-06-14	0	0	2	null	2ad1a822-b8c6-4de2-8864-9920bf6f3ac5
21000.0	2024-06-14	0	0	3	null	2ad1a822-b8c6-4de2-8864-9920bf6f3ac5
21000.0	2024-06-14	0	2	4	null	2ad1a822-b8c6-4de2-8864-9920bf6f3ac5
21000.0	2024-06-14	0	1	5	null	2ad1a822-b8c6-4de2-8864-9920bf6f3ac5
21000.0	2024-06-14	0	1	6	null	2ad1a822-b8c6-4de2-8864-9920bf6f3ac5
21000.0	2024-06-14	0	0	7	null	2ad1a822-b8c6-4de2-8864-9920bf6f3ac5
21000.0	2024-06-14	0	0	8	null	2ad1a822-b8c6-4de2-8864-9920bf6f3ac5
21000.0	2024-06-14	0	3	9	null	2ad1a822-b8c6-4de2-8864-9920bf6f3ac5
21000.0	2024-06-14	0	2	10	null	2ad1a822-b8c6-4de2-8864-9920bf6f3ac5
21000.0	2024-06-14	0	2	11	null	d710321b-8bb6-4ec3-b0e6-95d276a40a89
21000.0	2024-06-14	0	2	12	null	d710321b-8bb6-4ec3-b0e6-95d276a40a89
21000.0	2024-06-14	0	2	13	null	d710321b-8bb6-4ec3-b0e6-95d276a40a89
21000.0	2024-06-14	0	1	14	null	d710321b-8bb6-4ec3-b0e6-95d276a40a89
21000.0	2024-06-14	0	1	15	null	d710321b-8bb6-4ec3-b0e6-95d276a40a89



The screenshot shows the H2 Database Console interface. The left sidebar lists the database schema: jdbc:h2:mem:students-db, PAYMENT, STUDENT, INFORMATION_SCHEMA, and Users. The right pane contains a SQL statement editor and a results grid.

SQL Statement:

```
SELECT * FROM STUDENT
```

Results:

CODE	FIRST_NAME	ID	LAST_NAME	PHOTO	PROGRAM_ID
112233	Meryeme	2ad1a822-b8c6-4de2-8864-9920bf6f3ac5	null	null	SDIA
112244	Asmae	d710321b-8bb6-4ec3-b0e6-95d276a40a89	null	null	SDIA
112255	Chaymae	c018ee97-54c2-4958-be79-8518d7c85ce3	null	null	IAAD
112266	Fatima	15a97cc7-fde4-46f6-a909-e054db1f09cf	null	null	BDCC

(4 rows, 2 ms)

Edit

Donc il nous reste d'ajouter un paiement, c'est l'occasion de voir aussi comment faire upload.

On va créer une méthode pour ajouter un paiement :

```
PaymentRestController.java
79         return paymentRepository.save(payment);
80     }
81     @PostMapping(path = "/payments", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
82     public Payment savePayment(@RequestParam MultipartFile file, LocalDate date, double amount,
83                               PaymentType type, String studentCode) throws IOException {
84         Path folderPath = Paths.get(System.getProperty("user.home"), ...more: "fsm-data", "payments");
85         if(!Files.exists(folderPath)) {
86             Files.createDirectories(folderPath);
87         }
88         String fileName= UUID.randomUUID().toString();
89         Path filePath = Paths.get(System.getProperty("user.home"), ...more: "fsm-data", "payments", fileName+".pdf");
90         Files.copy(file.getInputStream(), filePath);
91         Student student = studentRepository.findByCode(studentCode);
92         Payment payment = Payment.builder()
93             .date(date).type(type).student(student).amount(amount)
94             .file(filePath.toUri().toString())
95             .status(PaymentStatus.CREATED).build();
96         return paymentRepository.save(payment);
97     }
```

On teste et on voie que le paiement a été créé avec le file :

Curl

```
curl -X 'POST' \
  'http://localhost:8021/payments?date=2024-11-11&amount=43000&type=CASH&studentCode=112233' \
  -H 'accept: */*' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@Cours_NLP_P4.pdf;type=application/pdf'
```

Request URL

```
http://localhost:8021/payments?date=2024-11-11&amount=43000&type=CASH&studentCode=112233
```

Server response

Code	Details
200	Response body
	{ "id": 42, "date": "2024-11-11", "amount": 43000, "type": "CASH", "status": "CREATED", "file": "file:///c:/Users/lenovo/fsm-data/payments/ee2152d0-5d68-4eed-89ff-77671f35db66.pdf", "student": { "firstName": "Meryeme", "lastName": null, "code": "112233", "programId": "SDIA", "photo": null, "id": "965c652b-3c57-46e1-8d48-257ae798247a" } }

On vérifie si le file existe, et s'il est dans le bon endroit, et que ce file a été bien créé dans la base de données :

```
Sélection Invite de commandes
Microsoft Windows [version 10.0.19045.4529]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\lenovo>cd fsm-data/payments

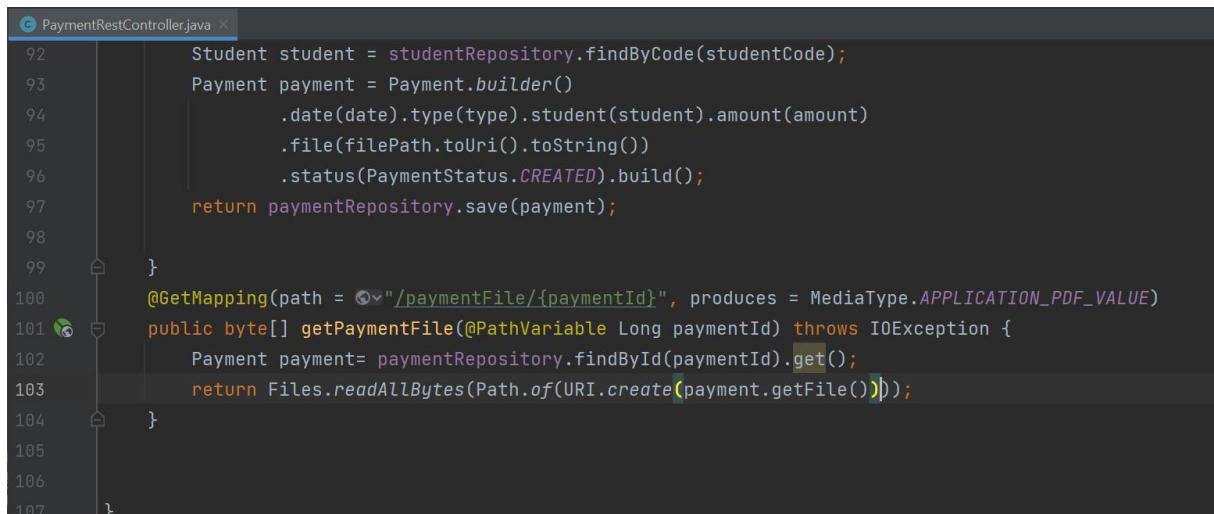
C:\Users\lenovo\fsm-data\payments>dir
Le volume dans le lecteur C n'a pas de nom.
Le numéro de série du volume est 0A91-1184

Répertoire de C:\Users\lenovo\fsm-data\payments

14/06/2024 22:09    <DIR>      .
14/06/2024 22:09    <DIR>      ..
14/06/2024 22:07           52 519 280d927b-8255-44d8-96d1-167d1fcd2c3e.pdf
14/06/2024 22:09           480 501 ee2152d0-5d68-4eed-89ff-77671f35db66.pdf
              2 fichier(s)      533 020 octets
              2 Rép(s)   154 775 076 864 octets libres

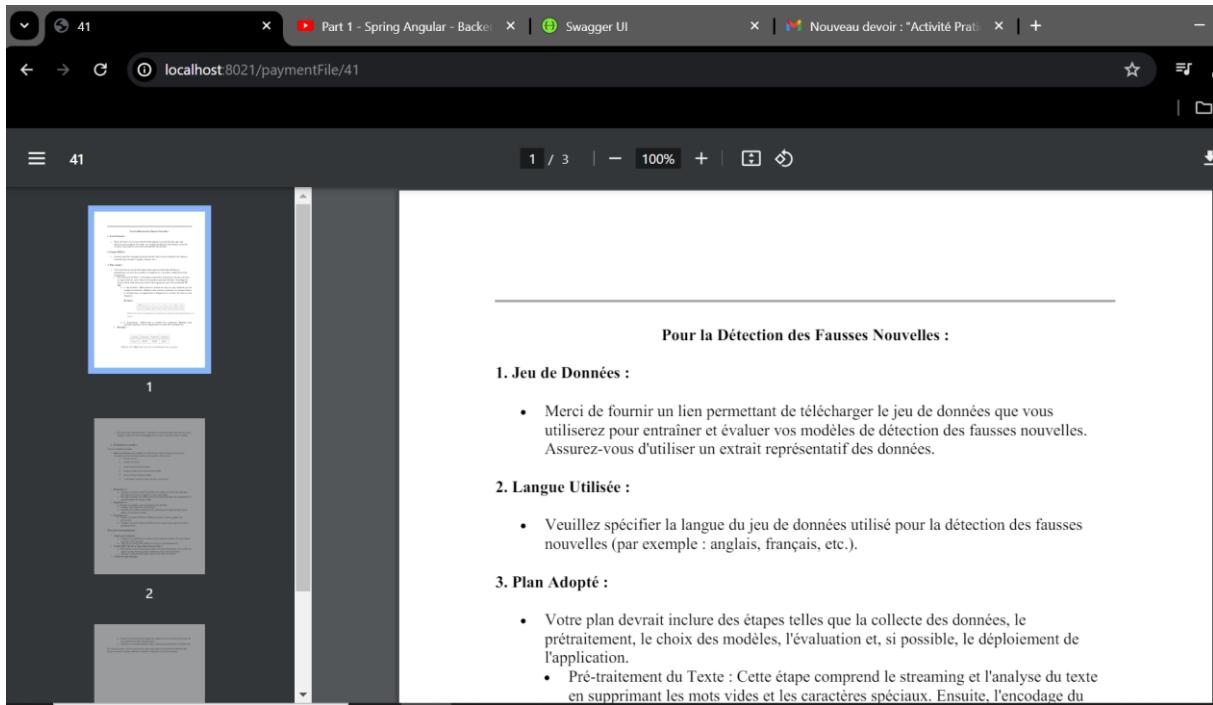
C:\Users\lenovo\fsm-data\payments>
```

Maintenant on va créer une méthode qui nous a permis de consulter ce file :



```
PaymentRestController.java
92     Student student = studentRepository.findByCode(studentCode);
93     Payment payment = Payment.builder()
94         .date(date).type(type).student(student).amount(amount)
95         .file(filePath.toUri().toString())
96         .status(PaymentStatus.CREATED).build();
97     return paymentRepository.save(payment);
98
99 }
100 @GetMapping(path = @Value("/paymentFile/{paymentId}"), produces = MediaType.APPLICATION_PDF_VALUE)
101 public byte[] getPaymentFile(@PathVariable Long paymentId) throws IOException {
102     Payment payment= paymentRepository.findById(paymentId).get();
103     return Files.readAllBytes(Path.of(URI.create(payment.getFile())));
104 }
105
106
107 }
```

On teste dans le server :



Pour respecter les bonnes pratiques on remplace les méthodes ayant un traitement dans un service `PaymentService` que l'on injecte et on utilise dans le `PaymentRestController`:

```

PaymentService.java ×
82     payment.setStatus(status);
83     return paymentRepository.save(payment);
84 }
85 @PostMapping(path = "/payments", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
86 public Payment savePayment(@RequestParam MultipartFile file, LocalDate date, double amount,
87                             PaymentType type, String studentCode) throws IOException {
88     Path folderPath = Paths.get(System.getProperty("user.home"), ...more: "fsm-data", "payments");
89     if(!Files.exists(folderPath)) {
90         Files.createDirectories(folderPath);
91     }
92     String fileName= UUID.randomUUID().toString();
93     Path filePath = Paths.get(System.getProperty("user.home"), ...more: "fsm-data", "payments");
94     Files.copy(file.getInputStream(), filePath);
95     Student student = studentRepository.findByCode(studentCode);
96     Payment payment = Payment.builder()
97         .date(date).type(type).student(student).amount(amount)
98         .file(filePath.toUri().toString())
99         .status(PaymentStatus.CREATED).build();
100    return paymentRepository.save(payment);
101}

```

Partie 2 :

Dans cette partie on va créer le frontend avec angular.

Dans le même projet précédent on va créer un dossier que l'on appelle `frontend-ang`, et on créer notre projet angular dans le terminal comme suit :

```
C:\Users\lenovo\IdeaProjects\fsm-demo-spring-ang\frontend-ang>ng new frontend-ang-app --directory=./ --no-standalone
? Which stylesheet format would you like to use? CSS [ https://developer.mozilla.org/docs/Web/CSS ]
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? No
CREATE angular.json (3000 bytes)
CREATE package.json (1085 bytes)
CREATE README.md (1095 bytes)
CREATE tsconfig.json (889 bytes)
CREATE .editorconfig (290 bytes)
CREATE .gitignore (590 bytes)
CREATE .vscode/extensions.json (134 bytes)
CREATE .vscode/launch.json (490 bytes)
```

Alors maintenant on va installer Angular Material (C'est un framework de design qui à peu près nous donne les mêmes fonctionnalités que nous trouvons dans bootstrap ou encore mieux avec des web components) :

```
C:\Users\lenovo\IdeaProjects\fsm-demo-spring-ang\frontend-ang>ng add @angular/material
? Would you like to share pseudonymous usage data about this project with the Angular Team at Google under Google's Privacy Policy at https://policies.google.com/privacy. For more details and how to change this setting, see https://angular.io/analytics. Yes
```

Thank you for sharing pseudonymous usage data. Should you change your mind, the following command will disable this feature entirely:

```
ng analytics disable
```

```
Global setting: enabled
Local setting: enabled
Effective status: enabled
```

On démarre l'application :

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure with modules: fsm-demo-spring-ang, .idea, .mvn, frontend-ang, .angular, .vscode, node_modules (library root), and src/app.
- Code Editor:** The PaymentService.java file is open in the editor.
- Terminal:**
 - Shows the command: `ng serve`
 - Output:
 - Application bundle generation complete. [3.319 seconds]
 - Watch mode enabled. Watching for file changes...
 - Local: <http://localhost:4200/>
 - press h + enter to show help
- File Status:** Shows file sizes for styles.css (93.20 kB), polyfills.js (86.27 kB), main.js (23.13 kB), and an initial total of 202.60 kB.



Hello, frontend-ang-app

Congratulations! Your app is running. 🎉

[Explore the Docs](#)

[Learn with Tutorials](#)

[CLI Docs](#)

[Angular Language Service](#)

[Angular DevTools](#)

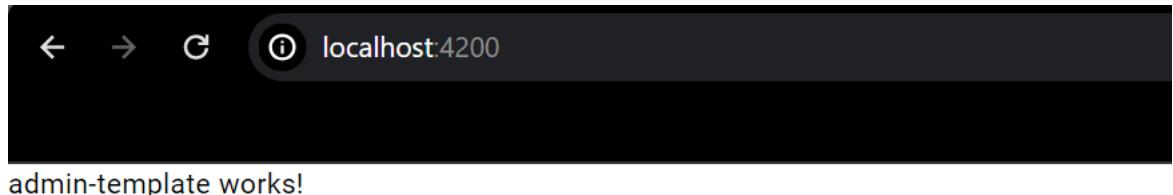


Donc ce qu'on fait, on va générer un composant que l'on appelle admin-template :

```
C:\Users\lenovo\IdeaProjects\fsm-demo-spring-ang\frontend-ang>ng g c admin-template
CREATE src/app/admin-template/admin-template.component.html (30 bytes)
CREATE src/app/admin-template/admin-template.component.spec.ts (674 bytes)
CREATE src/app/admin-template/admin-template.component.ts (240 bytes)
CREATE src/app/admin-template/admin-template.component.css (0 bytes)
UPDATE src/app/app.module.ts (643 bytes)
```

On va donc afficher ce composant dans app.component :

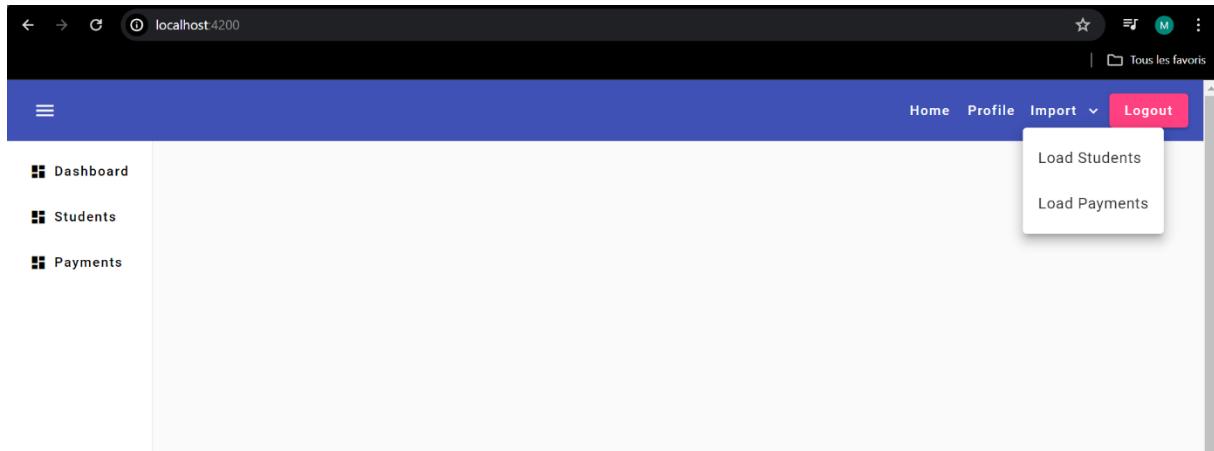
```
app.component.html ×
1  <app-admin-template></app-admin-template>
2
```



Maintenant on crée une barre de navigation dans admin-template, avec une sidebar :

```
admin-template.component.html
1  <mat-toolbar color="primary">
2    <button mat-icon-button>
3      <mat-icon>menu</mat-icon>
4    </button>
5    <span style="flex: auto"></span>
6    <button mat-button>Home</button>
7    <button mat-button>Profile</button>
8    <button mat-button [matMenuTriggerFor]="#importMenu">
9      <mat-icon iconPositionEnd>keyboard_arrow_down</mat-icon>
10     Import
11   </button>
12   <mat-menu #importMenu>
13     <button mat-menu-item>Load Students</button>
14     <button mat-menu-item>Load Payments</button>
15   </mat-menu>
16   <button mat-raised-button color="accent">
17     Logout
18   </button>
19 </mat-toolbar>
```

The code editor shows the `admin-template.component.html` file. It contains an `mat-toolbar` component with a primary color. Inside the toolbar, there is a `button` with `mat-icon-button` class, which contains a `mat-icon` element with the value `menu`. Following this is a `span` element with a `flex: auto` style. There are two `button` elements with `mat-button` class, labeled `Home` and `Profile`. Below these is another `button` with `mat-button` class and a `[matMenuTriggerFor]` attribute set to `#importMenu`. This button contains a `mat-icon` element with the `iconPositionEnd` attribute and the value `keyboard_arrow_down`, followed by the word `Import`. A `mat-menu` component is triggered by this button, with the id `#importMenu`. Inside the menu, there are two `button` elements with `mat-menu-item` class, labeled `Load Students` and `Load Payments`. The entire toolbar structure is closed with a final `</mat-toolbar>`.



Pour que ces balises d'angular material fonctionnent on a ajouté des modules :

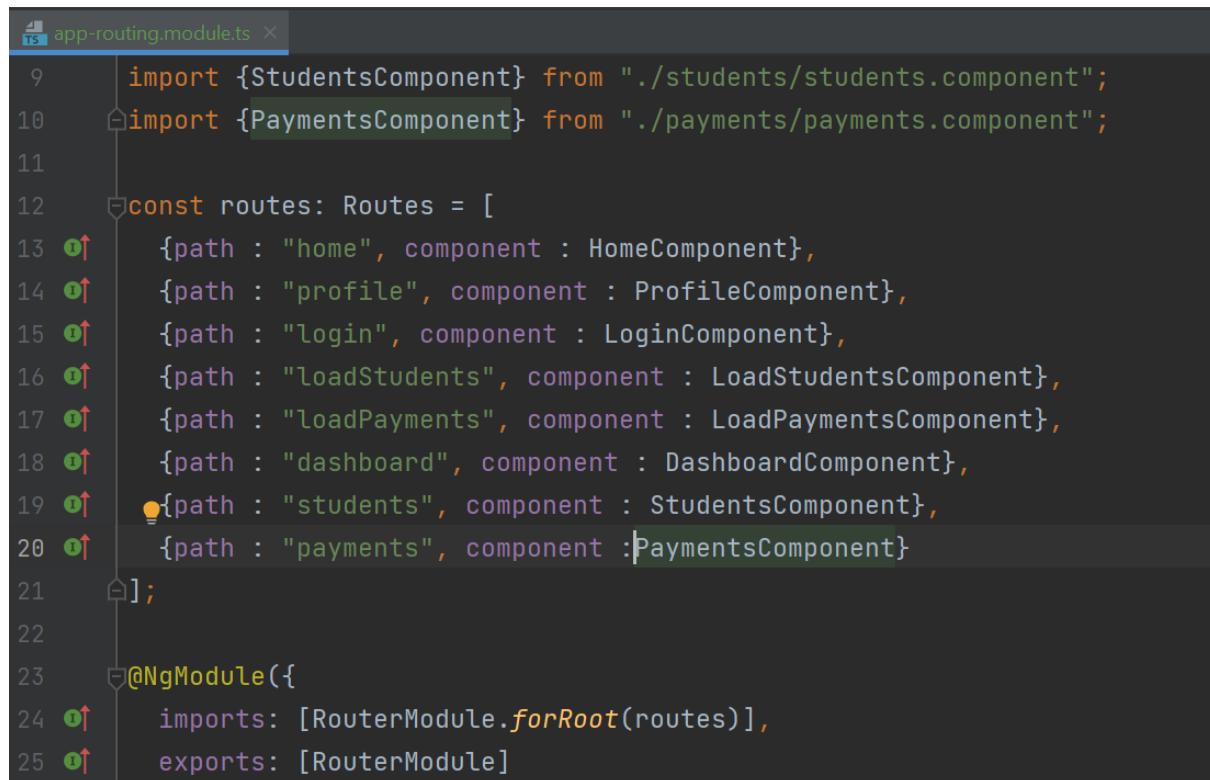
```
app.module.ts
19 import {MatTableModule} from "@angular/material/table";
20 import {MatPaginatorModule} from "@angular/material/paginator";
21 import {MatSortModule} from "@angular/material/sort";
22
23 @NgModule({
24   declarations: [
25     AppComponent,
26     AdminTemplateComponent,
27   ],
28   imports: [
29     BrowserModule,
30     AppRoutingModule,
31     MatToolbarModule,
32     MatButtonModule,
33     MatIconModule,
34     MatInputModule,
35     MatSidenavModule,
36     MatListModule,
```

The image shows a code editor window with the file 'app.module.ts' open. The code defines an NgModule with declarations and imports. The imports section includes several Angular Material modules: MatTableModule, MatPaginatorModule, MatSortModule, BrowserModule, AppRoutingModule, MatToolbarModule, MatButtonModule, MatIconModule, MatInputModule, MatSidenavModule, and MatListModule. There are also two small red arrows with question marks pointing upwards from the 'declarations' and 'imports' sections towards the 'NgModule' decorator.

Alors on passe pour faire notre système de navigation, d'abord on va créer les composants (on va citer ici un seul exemple de home et c'est le même pour les autres) :

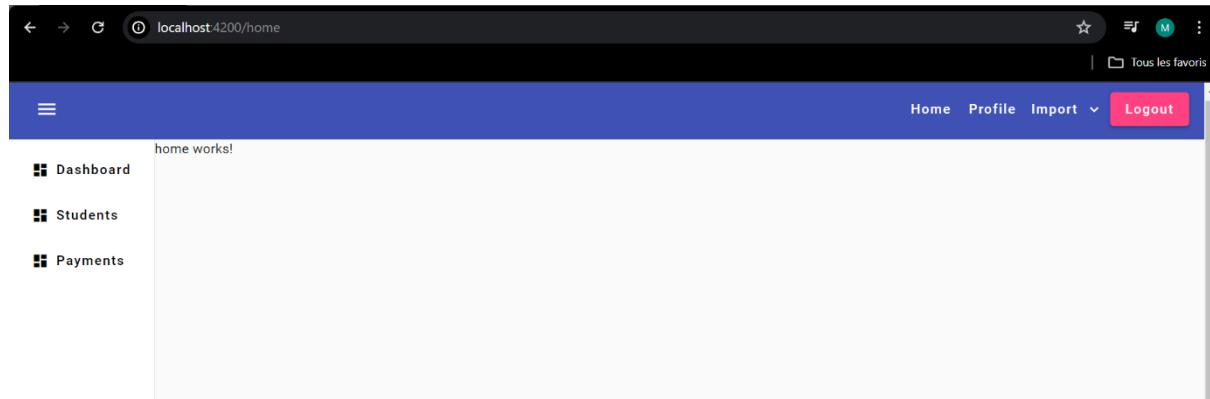
```
C:\Users\lenovo\IdeaProjects\fsm-demo-spring-ang\frontend-ang>ng g c home
CREATE src/app/home/home.component.html (20 bytes)
CREATE src/app/home/home.component.spec.ts (610 bytes)
CREATE src/app/home/home.component.ts (201 bytes)
CREATE src/app/home/home.component.css (0 bytes)
UPDATE src/app/app.module.ts (1843 bytes)
```

Puis on va déclarer les routes :

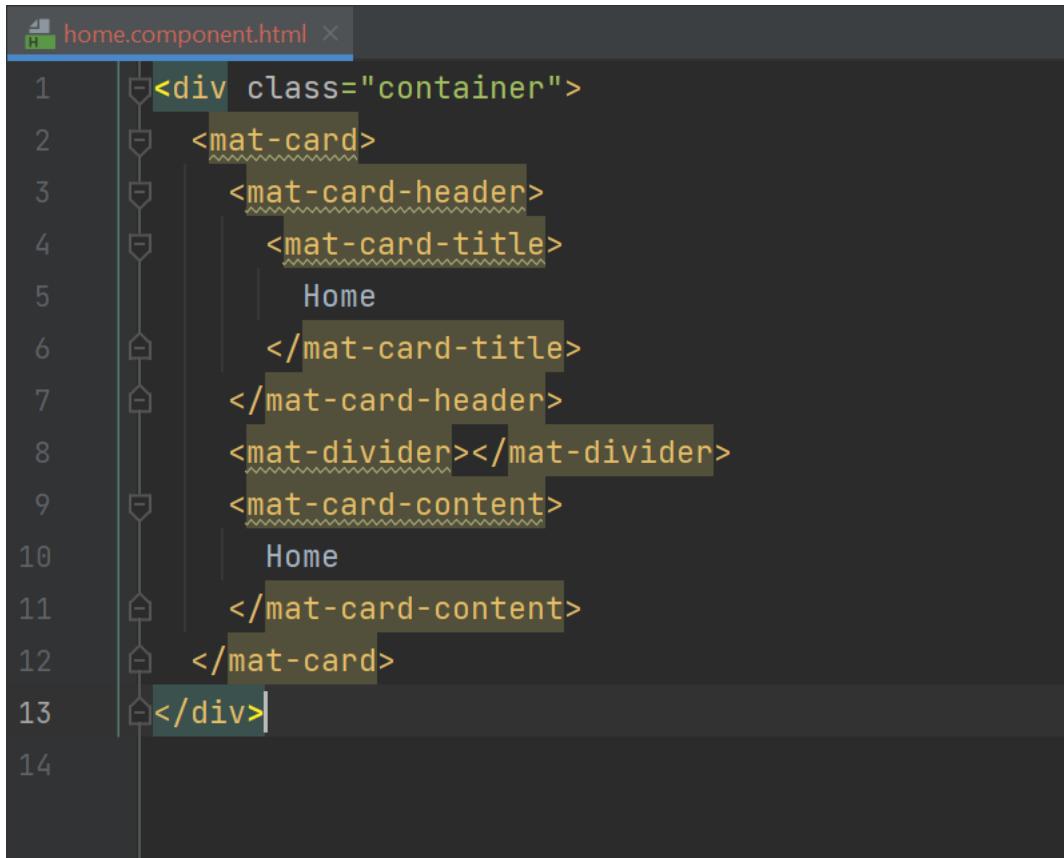


```
app-routing.module.ts
9 import {StudentsComponent} from "./students/students.component";
10 import {PaymentsComponent} from "./payments/payments.component";
11
12 const routes: Routes = [
13   {path : "home", component : HomeComponent},
14   {path : "profile", component : ProfileComponent},
15   {path : "login", component : LoginComponent},
16   {path : "loadStudents", component : LoadStudentsComponent},
17   {path : "loadPayments", component : LoadPaymentsComponent},
18   {path : "dashboard", component : DashboardComponent},
19   {path : "students", component : StudentsComponent},
20   {path : "payments", component : PaymentsComponent}
21 ];
22
23 @NgModule({
24   imports: [RouterModule.forRoot(routes)],
25   exports: [RouterModule]
```

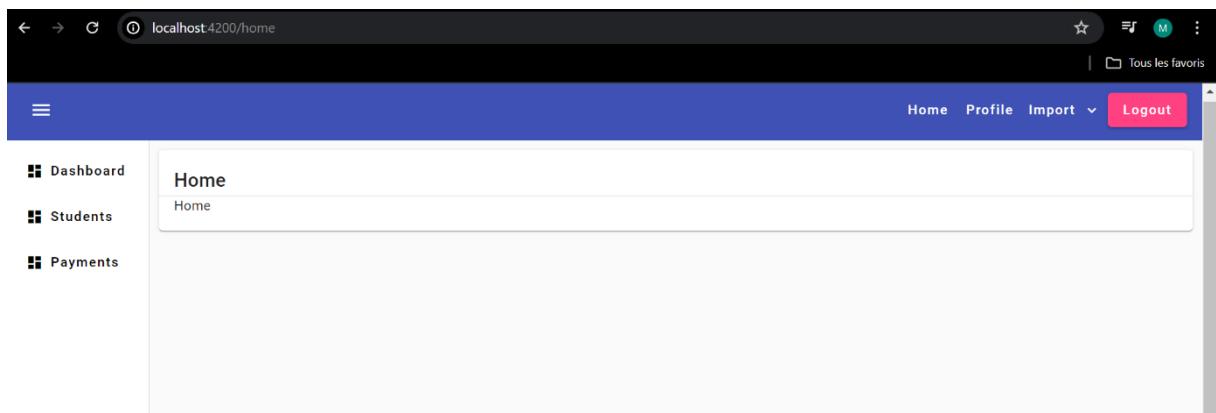
Donc on va utiliser ces routes dans notre système de navigation, et on voie qu'il marche bien :



Maintenant on va mettre à jour nos composants, en prend comme exemple home, et on le fait de la même manière pour les autres et on teste :



```
home.component.html
1 <div class="container">
2   <mat-card>
3     <mat-card-header>
4       <mat-card-title>
5         Home
6       </mat-card-title>
7     </mat-card-header>
8     <mat-divider></mat-divider>
9     <mat-card-content>
10    Home
11  </mat-card-content>
12 </mat-card>
13 </div>
14
```



Il nous reste le bouton de menu, qu'il doit afficher et masquer le menu, c'est simple on va créer un id pour le menu que l'on appelle dans le bouton en utilisant toggle :

```

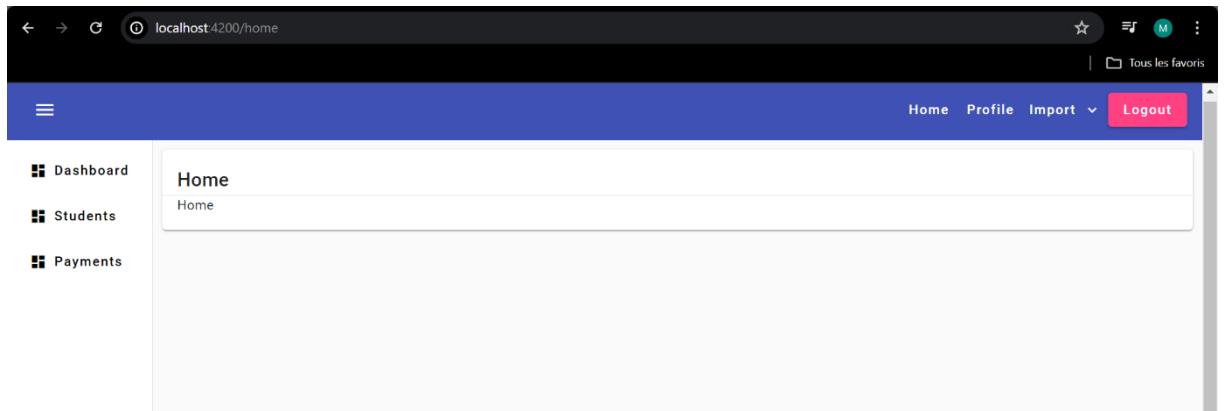
1  <mat-toolbar color="primary">
2    <button mat-icon-button (click)="myDrawer.toggle()">
3      <mat-icon>menu</mat-icon>
4    </button>
5    <span style="flex: auto"></span>
6    <button mat-button routerLink="/home">Home</button>
7    <button mat-button routerLink="/profile">Profile</button>
8    <button mat-button [matMenuTriggerFor]="importMenu">
9      <mat-icon iconPositionEnd>keyboard_arrow_down</mat-icon>
10     Import
11   </button>
12   <mat-menu #importMenu>
13     <button mat-menu-item routerLink="/loadStudents">Load Students</button>
14     <button mat-menu-item routerLink="/loadPayments">Load Payments</button>
15   </mat-menu>
16   <button mat-raised-button color="accent" routerLink="/login">
17     Logout
18   </button>
19 </mat-toolbar>

```

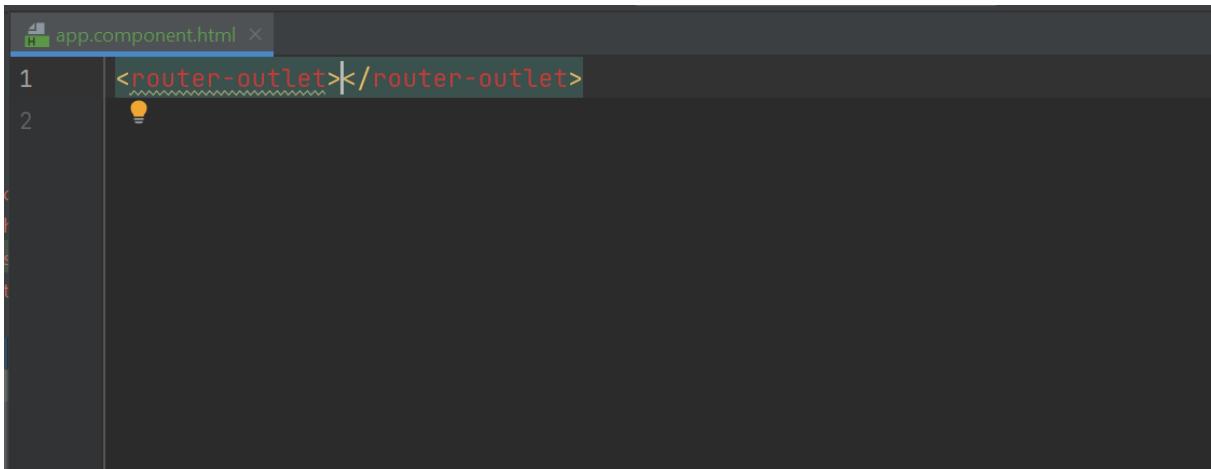
mat-drawer-container > mat-drawer > myDrawer

Externally added file

[View File](#) [Always](#)

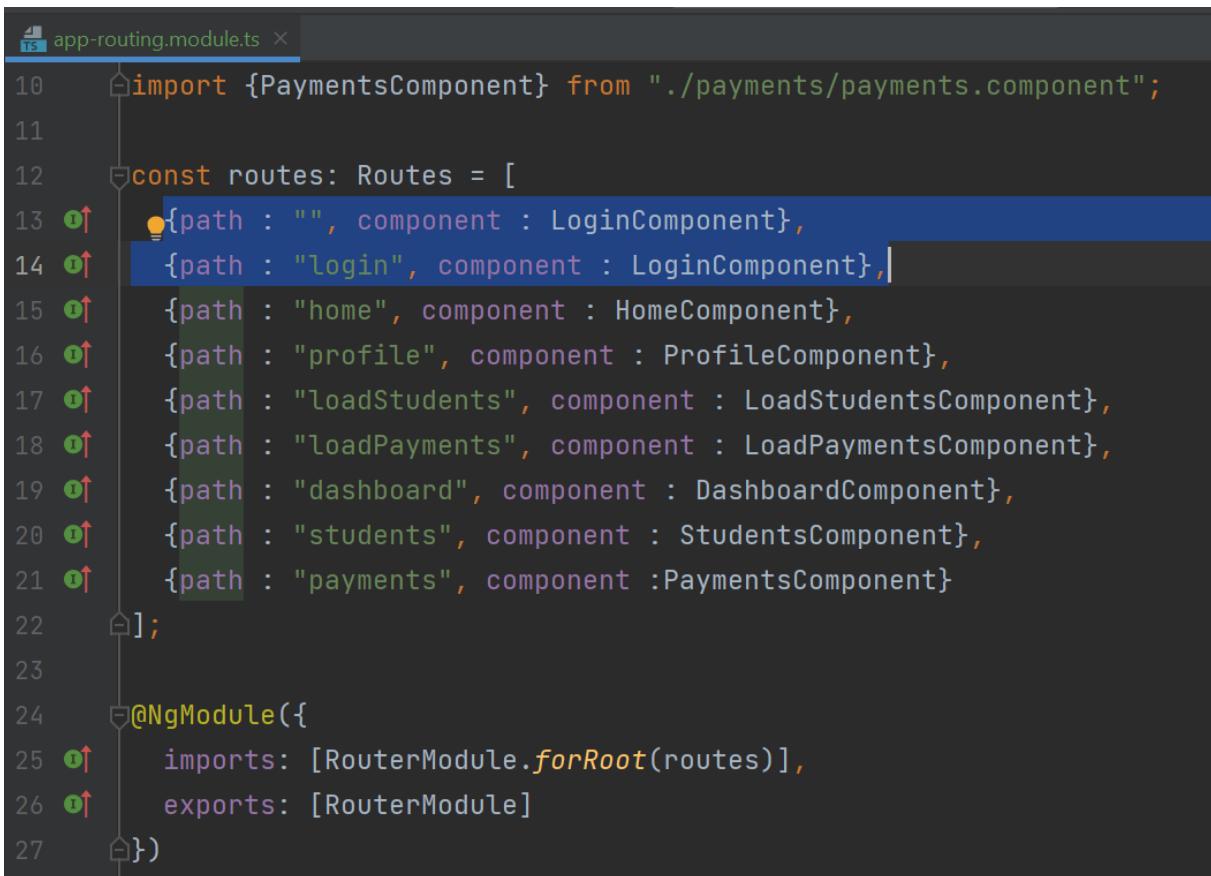


On veut que notre application quand il démarre affiche un formulaire d'authentification, alors dans app.component on va utiliser router-outlet au lieu d'admin-template :



```
1 <router-outlet></router-outlet>
2
```

Après on suppose que la route par défaut c'est login, alors on le déplace en haut et on ajoute une autre route pour que s'il ne fait rien il affiche le formulaire d'authentification aussi :



```
10 import {PaymentsComponent} from "./payments/payments.component";
11
12 const routes: Routes = [
13   {path : "", component : LoginComponent},
14   {path : "login", component : LoginComponent},
15   {path : "home", component : HomeComponent},
16   {path : "profile", component : ProfileComponent},
17   {path : "loadStudents", component : LoadStudentsComponent},
18   {path : "loadPayments", component : LoadPaymentsComponent},
19   {path : "dashboard", component : DashboardComponent},
20   {path : "students", component : StudentsComponent},
21   {path : "payments", component : PaymentsComponent}
22 ];
23
24 @NgModule({
25   imports: [RouterModule.forRoot(routes)],
26   exports: [RouterModule]
27 })
```

Alors allons pour compléter la page d'authentification :

```
login.component.html
```

```
1  <div class="container">
2    <mat-card class="login-form" >
3      <mat-card-header>
4        <mat-card-title>
5          Authentication
6        </mat-card-title>
7      </mat-card-header>
8      <mat-divider></mat-divider>
9      <mat-card-content>
10     <mat-form-field appearance="outline">
11       <mat-label>Username or Email</mat-label>
12       <input matInput>
13     </mat-form-field>
14     <mat-form-field appearance="outline">
15       <mat-label>Password</mat-label>
```

div.container > mat-card.login-form > mat-card-content > mat-form-field

A screenshot of a login form card. The card has a title "Authentication" at the top. It contains two input fields: one labeled "Username or Email" and another labeled "Password". At the bottom right of the card is a blue "Login" button.

Jusqu'à maintenant on n'a pas un système d'authentification en backend, on va bricoler un qui est basique en frontend.

Pour cela maintenant on va voir comment récupérer les données du formulaire, tout d'abord on va importer un module qui s'appelle `ReactiveFormsModule` :

```
43
44     ],
45     imports: [
46         BrowserModule,
47         AppRoutingModule,
48         MatToolbarModule,
49         MatButtonModule,
50         MatIconModule,
51         MatMenuModule,
52         MatSidenavModule,
53         MatListModule,
54         MatCardModule,
55         MatFormFieldModule,
56         MatInputModule,
57         ReactiveFormsModule,
```

AppModule > imports

Maintenant dans notre composant login, on va déclarer notre formulaire, et l'initialiser :

```
5     selector: 'app-login',
6     templateUrl: './login.component.html',
7     styleUrls: ['./login.component.css'
8   })
9   export class LoginComponent implements OnInit {
10     public loginForm!: FormGroup;
11     constructor(private fb: FormBuilder) {
12     }
13     ngOnInit(): void {
14       this.loginForm = this.fb.group( controls: {
15         username: this.fb.control( formState: '' ),
16         password: this.fb.control( formState: '' )
17       })
18     }
19   }
```

LoginComponent > ngOnInit() > loginForm > password

On doit faire la data binding, ça veut dire lier les champs du formulaire avec ses variables :

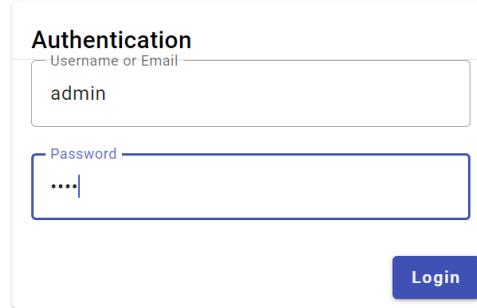
```
login.component.html
```

```
1 <div class="container">
2   {{loginForm.value | json}}
3   <mat-card class="login-form" [formGroup]="loginForm">
4     <mat-card-header>
5       <mat-card-title>
6         Authentication
7       </mat-card-title>
8     </mat-card-header>
9     <mat-divider></mat-divider>
10    <mat-card-content>
11      <mat-form-field appearance="outline">
12        <mat-label>Username or Email</mat-label>
13        <input matInput formControlName="username">
14      </mat-form-field>
15      <mat-form-field appearance="outline">
16        <mat-label>Password</mat-label>
17        <input matInput type="password" formControlName="password">
18      </mat-form-field>
19    </mat-card-content>
```

div.container > mat-card.login-form



```
{"username": "admin", "password": "1234"}
```



On va ajouter un click dans notre bouton qui fait appelle à la méthode login qu'on va créer après :

```
login.component.html
6     Authentication
7         </mat-card-title>
8     </mat-card-header>
9     <mat-divider></mat-divider>
10    <mat-card-content>
11        <mat-form-field appearance="outline">
12            <mat-label>Username or Email</mat-label>
13            <input matInput formControlName="username">
14        </mat-form-field>
15        <mat-form-field appearance="outline">
16            <mat-label>Password</mat-label>
17            <input matInput type="password" formControlName="password">
18        </mat-form-field>
19    </mat-card-content>
20    <mat-card-actions align="end">
21        <button (click)="Login()" mat-raised-button color="primary">Login</button>
22    </mat-card-actions>
23 </mat-card>
24 </div>
```

div.container > mat-card.login-form > mat-card-actions > button

Passons pour définir la méthode login dans laquelle on va récupérer le username et le mot de passe :

```
login.component.ts
8  })
9  export class LoginComponent implements OnInit {
10    public loginForm!: FormGroup;
11    constructor(private fb: FormBuilder) {
12    }
13  ↑ ngOnInit(): void {
14    this.loginForm = this.fb.group( controls: {
15      username: this.fb.control( formState: '' ),
16      password: this.fb.control( formState: '' )
17    })
18  }
19
20  login() {
21    let username = this.loginForm.value.username;
22    let password = this.loginForm.value.password;
23  }
24
25  }
26
LoginComponent > login()
```

Nous allons en train de créer un système d'authentification basique, pour cela on doit créer un service :

```
C:\Users\lenovo\IdeaProjects\fsm-demo-spring-ang\frontend-ang>ng g s services/auth
CREATE src/app/services/auth.service.spec.ts (363 bytes)
CREATE src/app/services/auth.service.ts (142 bytes)
```

Dans ce service on fait une vérification du mot de passe saisi par l'utilisateur est le mot de passe stockées déjà, on précise les rôles aussi pour chaque utilisateur :

```
component.ts × auth.service.ts ×
```

```
})
}

export class AuthService {
    public users: any= {
        admin: {password: '1234', roles : ['STUDENT', 'ADMIN']},
        user1: {password: '1234', roles: ['STUDENT']}
    }

    constructor() { }

    public login(username: string, password: boolean) {
        if(this.users[username] && this.users[username]['password']==password) {
            return true;
        }else {
            return false;
        }
    }
}
AuthService > login()
```

Revenons maintenant à notre composant login pour terminer la définition de la méthode login :

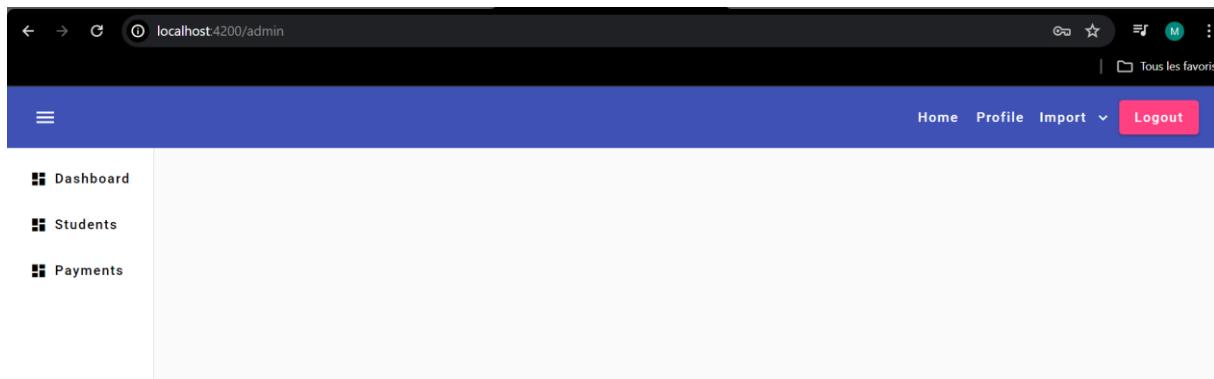
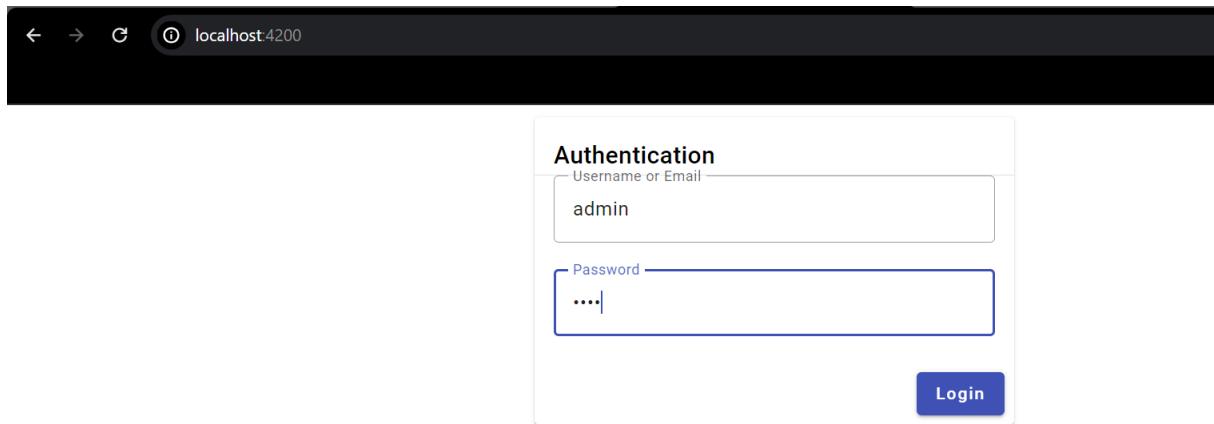
```
login.component.ts ×
```

```
21
22
23     login() {
24         let username = this.loginForm.value.username;
25         let password = this.loginForm.value.password;
26         let auth: boolean = this.authService.login(username, password);
27         if(auth == true) {
28             this.router.navigateByUrl(url: "/admin");
29         }
30     }
31 }
32 }
33 }
```

N'oublions pas d'ajouter la route d'admin :

```
app-routing.module.ts
11 import {AdminTemplateComponent} from "./admin-template/admin-template.component";
12
13 const routes: Routes = [
14   {path: "", component: LoginComponent},
15   {path: "login", component: LoginComponent},
16   {path: "admin", component: AdminTemplateComponent}, // This line is highlighted
17   {path: "home", component: HomeComponent},
18   {path: "profile", component: ProfileComponent},
19   {path: "loadStudents", component: LoadStudentsComponent},
20   {path: "loadPayments", component: LoadPaymentsComponent},
21   {path: "dashboard", component: DashboardComponent},
```

On va tester, on se connecte avec le compte admin :



Maintenant on doit changer la structure des routes, les tous doivent être des children de la route admin :

```
app-routing.module.ts ×
12
13     const routes: Routes = [
14       {path : "", component : LoginComponent},
15       {path : "login", component : LoginComponent},
16       {path : "admin", component : AdminTemplateComponent, children : [
17         {path : "home", component : HomeComponent},
18         {path : "profile", component : ProfileComponent},
19         {path : "loadStudents", component : LoadStudentsComponent},
20         {path : "loadPayments", component : LoadPaymentsComponent},
21         {path : "dashboard", component : DashboardComponent},
22         {path : "students", component : StudentsComponent},
23         {path : "payments", component : PaymentsComponent}
24       ] },
25     ];
26   @NgModule({
```

Mais à condition de changer les liens dans admin-template en ajoutant /admin :

```
admin-template.component.html ×
1 <mat-toolbar color="primary">
2   <button mat-icon-button (click)="myDrawer.toggle()">
3     <mat-icon>menu</mat-icon>
4   </button>
5   <span style="flex: auto"></span>
6   <button mat-button routerLink="/admin/home">Home</button>
7   <button mat-button routerLink="/admin/profile">Profile</button>
8   <button mat-button [matMenuTriggerFor]="importMenu">
9     <mat-icon iconPositionEnd>keyboard_arrow_down</mat-icon>
10    Import
11  </button>
12  <mat-menu #importMenu>
13    <button mat-menu-item routerLink="/admin/loadStudents">Load Students</button>
14    <button mat-menu-item routerLink="/admin/loadPayments">Load Payments</button>
15  </mat-menu>
16  <button mat-raised-button color="accent" routerLink="/login">
17    Logout
18  </button>
19</mat-toolbar>
```

Notre système d'authentification n'est pas complet car on doit garder l'utilisateur connecté, alors on fait ça dans notre service auth :

```
auth.service.ts ×
5
6     export class AuthService {
7         public users: any= {
8             admin: {password: '1234', roles : ['STUDENT', 'ADMIN']},
9             user1: {password: '1234', roles: ['STUDENT']}
10        }
11        public username: any;
12        public isAuthenticated : boolean= false;
13        public roles: string[] = [];
14
15        constructor() { }
16
17        public login(username: string, password: boolean) {
18            if(this.users[username] && this.users[username]['password']==password) {
19                this.username=username;
20                this.isAuthenticated= true;
21                this.roles= this.users[username]['roles'];
22                return true;
23            }else {
24                return false;
25            }
26        }
27    }
28
29    AuthService > login()
```

Alors on n'a besoin d'afficher l'utilisateur connecté donc on va vers admin-template et on injecte le service auth, puis on créer un bouton dans la partie html en faisons un teste si l'utilisateur est authentifié, on affiche son nom, ainsi on va ajouter un click dans le bouton logout :

```
admin-template.component.html ×
5         <span style="flex: auto"></span>
6         <button mat-button routerLink="/admin/home">Home</button>
7         <button mat-button routerLink="/admin/profile">Profile</button>
8         <button mat-button [matMenuTriggerFor]="importMenu">
9             <mat-icon iconPositionEnd>keyboard_arrow_down</mat-icon>
10            Import
11        </button>
12        <mat-menu #importMenu>
13            <button mat-menu-item routerLink="/admin/loadStudents">Load Students</button>
14            <button mat-menu-item routerLink="/admin/loadPayments">Load Payments</button>
15        </mat-menu>
16        <button mat-button *ngIf="authService.isAuthenticated">
17            {{authService.username}}
18        </button>
19        <button mat-raised-button color="accent" (click)="logout()" >
20            Logout
21        </button>
22    </mat-toolbar>
23    <mat-drawer-container>
```

On va créer la méthode logout dans le composant admin-template, qui est à son tour fait appelle à la méthode logout du service auth :

```
1 import {AuthService} from "../services/auth.service";
2
3
4 @Component({
5   selector: 'app-admin-template',
6   templateUrl: './admin-template.component.html',
7   styleUrls: ['./admin-template.component.css']
8 })
9 export class AdminTemplateComponent {
10
11   constructor(public authService: AuthService) {
12     }
13
14   logout() {
15     this.authService.logout();
16   }
17 }
18
19
```

AdminTemplateComponent > logout()

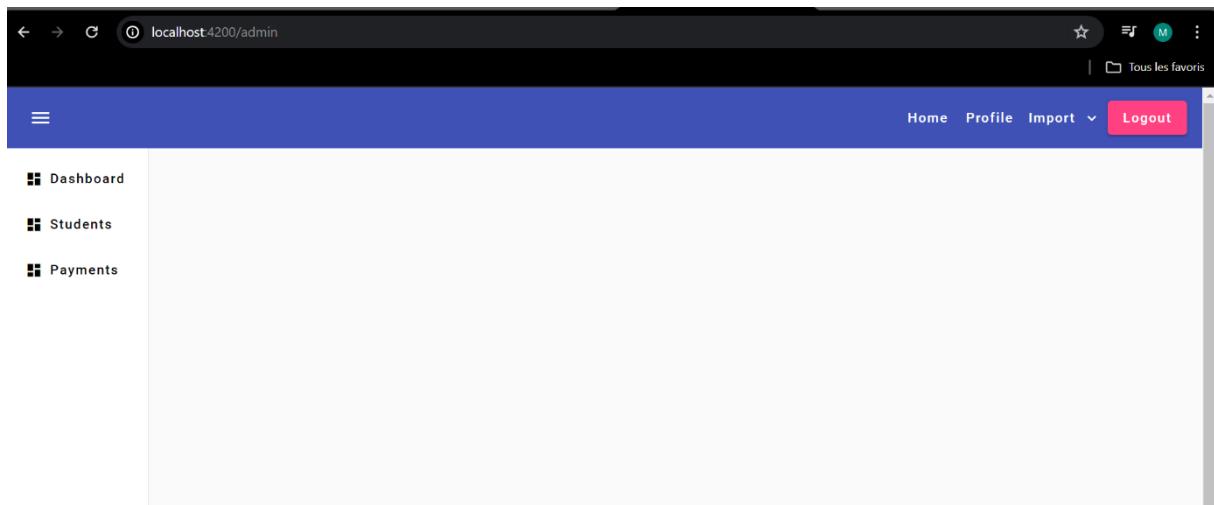
Allons pour redéfinir cette dernière méthode dans notre service :

```

19     if(this.users[username] && this.users[username]['password']==password) {
20         this.username=username;
21         this.isAuthenticated= true;
22         this.roles= this.users[username]['roles'];
23         return true;
24     }else {
25         return false;
26     }
27 }
28
29 logout() {
30     this.isAuthenticated= false;
31     this.roles = [];
32     this.username= undefined;
33     this.router.navigateByUrl(url: '/login');
34 }
35 }
36 }
37

```

AuthService > constructor() > router

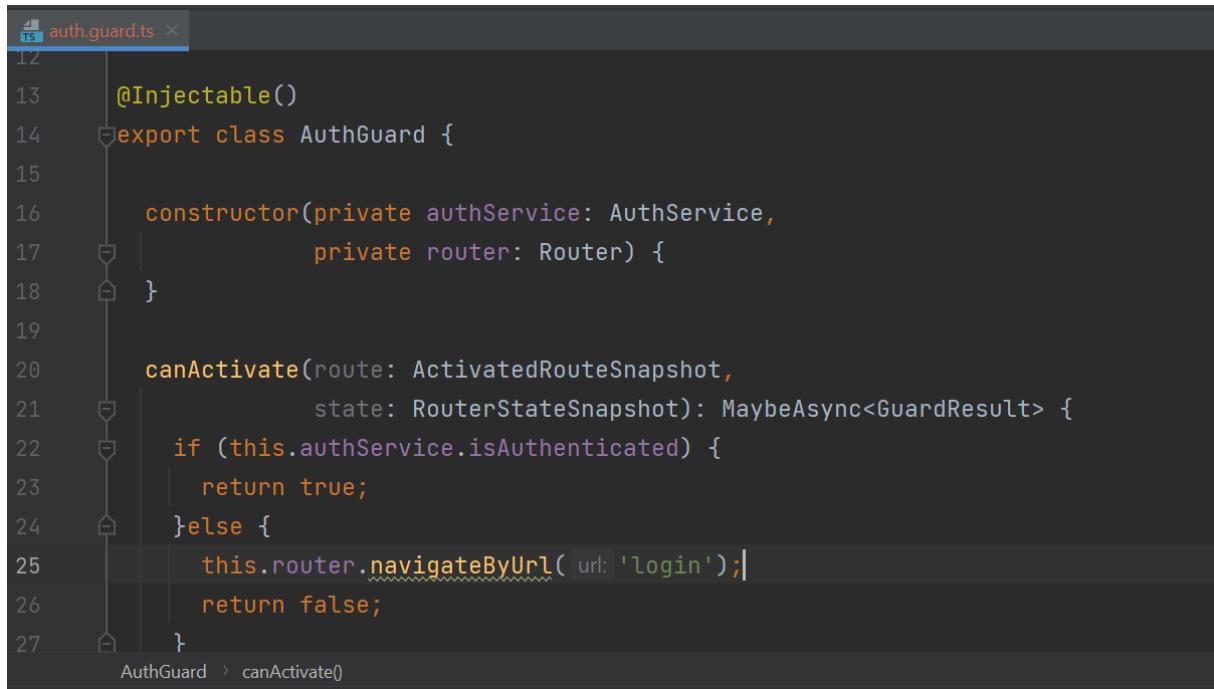


Mais on a encore un problème c'est que on peut accéder à quelle page via l'url, pour se régler ça on doit protéger les routes, en utilisant guards.

Donc on va créer le guard auth :

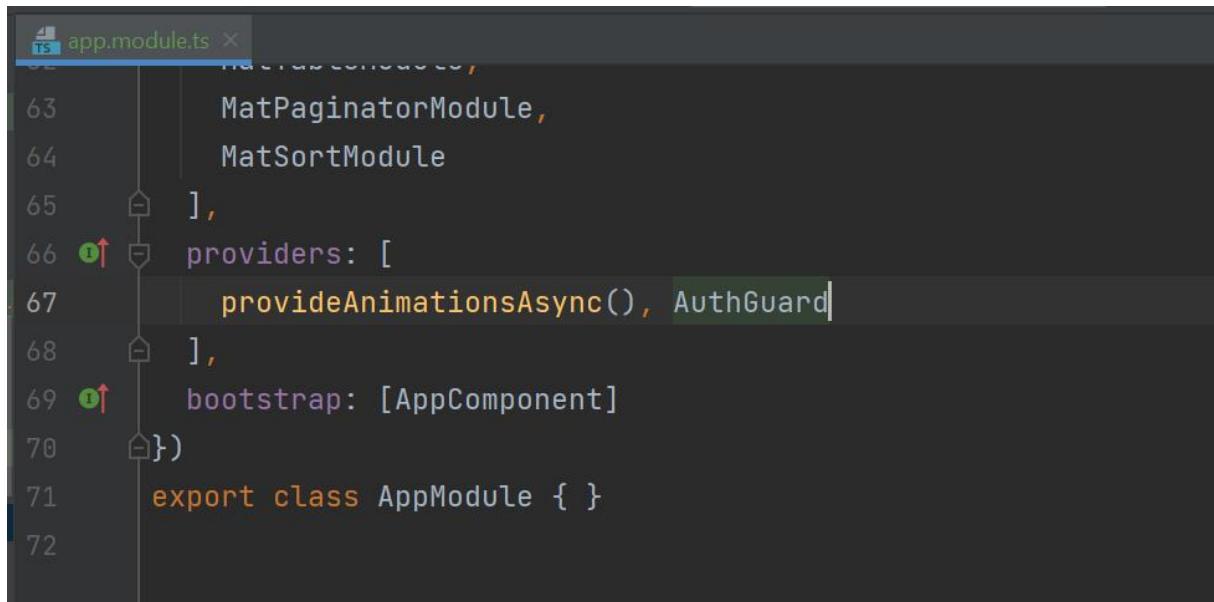
```
C:\Users\lenovo\IdeaProjects\fsm-demo-spring-ang\frontend-ang>ng g g guards/auth
? Which type of guard would you like to create? (Press <space> to select, <a> to toggle all, <i> to invert selection, and <enter> to to
? Which type of guard would you like to create? CanActivate
CREATE src/app/guards/auth.guard.spec.ts (478 bytes)
CREATE src/app/guards/auth.guard.ts (133 bytes)
```

Donc on ajoute une ligne de code pour que le guard vérifie si l'utilisateur est connecté ou non :



```
auth.guard.ts ×
12
13     @Injectable()
14     export class AuthGuard {
15
16         constructor(private authService: AuthService,
17                     private router: Router) {
18     }
19
20         canActivate(route: ActivatedRouteSnapshot,
21                      state: RouterStateSnapshot): MaybeAsync<GuardResult> {
22             if (this.authService.isAuthenticated) {
23                 return true;
24             } else {
25                 this.router.navigateByUrl(url: 'login');
26                 return false;
27             }
28         }
29     }
30
31     AuthGuard > canActivate()
```

Alors pour que ce service fonctionne on doit le déclarer dans le module premièrement :



```
app.module.ts ×
12
13     MatPaginatorModule,
14     MatSortModule
15     ],
16     providers: [
17         provideAnimationsAsync(), AuthGuard
18     ],
19     bootstrap: [AppComponent]
20 })
21     export class AppModule { }
```

Deuxièmement il faut protéger les routes :

```

11 import {AdminTemplateComponent} from "./admin-template/admin-template.com
12 import {AuthGuard} from "./guards/auth.guard";
13
14 const routes: Routes = [
15   {path: "", component: LoginComponent},
16   {path: "login", component: LoginComponent},
17   {path: "admin", component: AdminTemplateComponent,
18     canActivate: [AuthGuard],
19     children: [
20       {path: "home", component: HomeComponent},
21       {path: "profile", component: ProfileComponent},
22       {path: "loadStudents", component: LoadStudentsComponent},
23       {path: "loadPayments", component: LoadPaymentsComponent},
24       {path: "dashboard", component: DashboardComponent}

```

The screenshot shows the project structure on the left and the app-routing.module.ts file on the right. The file contains the same routing configuration as the previous screenshot, but with more inline error markers (red and yellow arrows) appearing on various lines, indicating issues with the code.

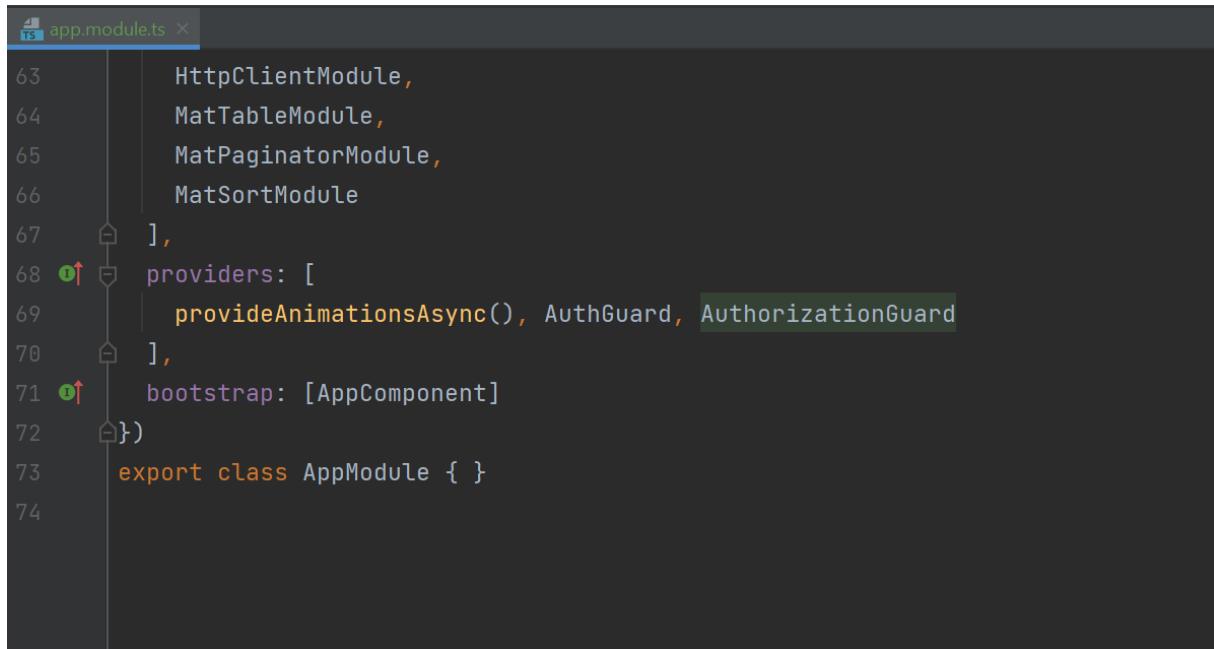
```

12 import {AuthGuard} from "./guards/auth.guard";
13
14 const routes: Routes = [
15   {path: "", component: LoginComponent},
16   {path: "login", component: LoginComponent},
17   {path: "admin", component: AdminTemplateComponent,
18     canActivate: [AuthGuard],
19     children: [
20       {path: "home", component: HomeComponent},
21       {path: "profile", component: ProfileComponent},
22       {path: "loadStudents", component: LoadStudentsComponent},
23       {path: "loadPayments", component: LoadPaymentsComponent},
24       {path: "dashboard", component: DashboardComponent},
25       {path: "students", component: StudentsComponent},
26       {path: "payments", component: PaymentsComponent}
27     ],
28   },
29
30   @NgModule({
31     imports: [RouterModule.forChild(routes)],
32     exports: [RouterModule]
33   })
34   export class AppRoutingModule { }

```

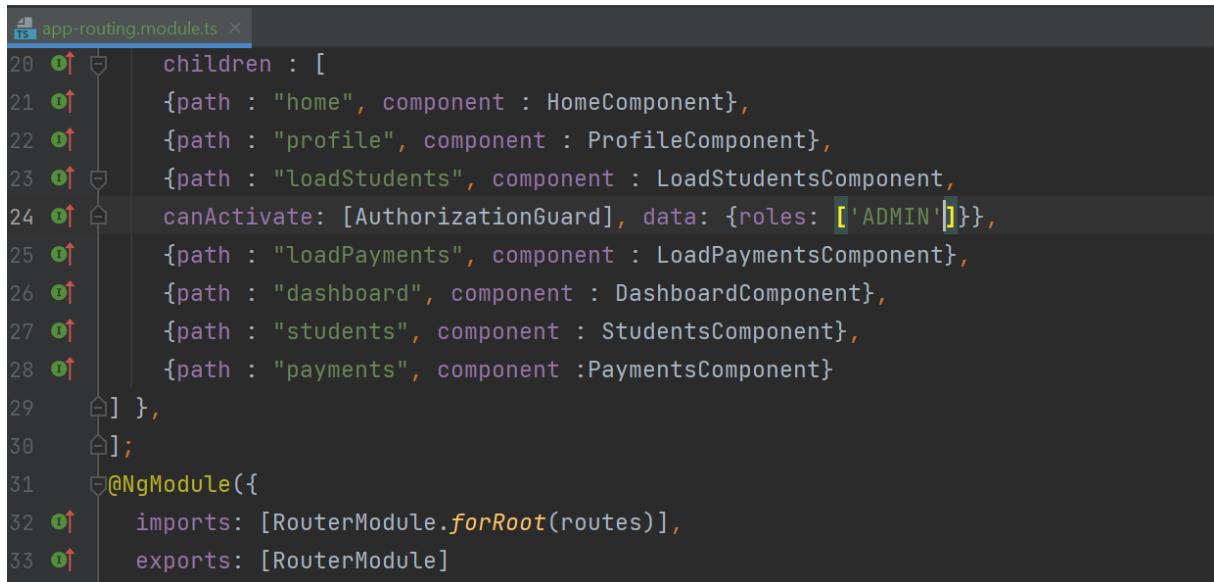
Maintenant on passe pour ajouter autre guard pour que l'on diversifié les roles d'Admin et User.

On crée un guard authorization, et on ajoute ce guard dans le module des routes :



```
63     HttpClientModule,
64     MatTableModule,
65     MatPaginatorModule,
66     MatSortModule
67   ],
68   providers: [
69     provideAnimationsAsync(),
70     AuthGuard,
71     AuthorizationGuard
72   ],
73   bootstrap: [AppComponent]
74 })  
export class AppModule { }
```

Après on applique ce guard aux routes que nous voulons :

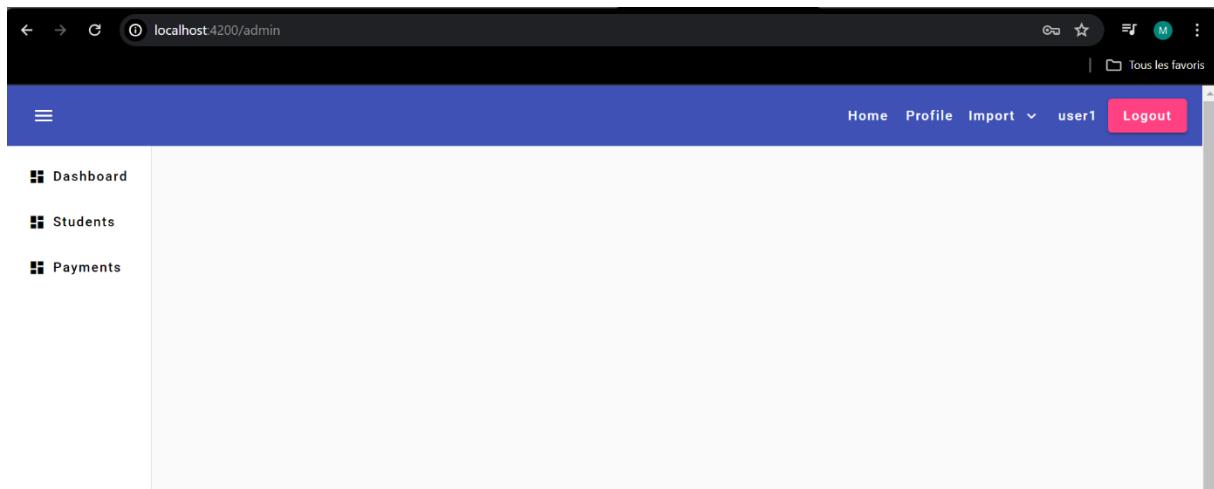


```
20   children : [
21     {path : "home", component : HomeComponent},
22     {path : "profile", component : ProfileComponent},
23     {path : "loadStudents", component : LoadStudentsComponent,
24       canActivate: [AuthorizationGuard], data: {roles: ['ADMIN']}},
25     {path : "LoadPayments", component : LoadPaymentsComponent},
26     {path : "dashboard", component : DashboardComponent},
27     {path : "students", component : StudentsComponent},
28     {path : "payments", component : PaymentsComponent}
29   ],
30 ];
31 @NgModule({
32   imports: [RouterModule.forRoot(routes)],
33   exports: [RouterModule]
```

Revenons à notre guard, on va récupérer les roles, et on vérifie si l'un des roles existe comme ceux qui sont requis, puis on teste l'application en se connectant en tant que user pour voir que load students et payments ne fonctionnent pas :

```
authorization.guard.ts
10 import {Injectable} from '@angular/core';
11 import {AuthService} from '../services/auth.service';
12
13 @Injectable()
14 export class AuthorizationGuard {
15
16   constructor(private authService : AuthService,
17               private router : Router) {
18   }
19
20   canActivate(route: ActivatedRouteSnapshot,
21               state: RouterStateSnapshot): MaybeAsync<GuardResult> {
22     if (this.authService.isAuthenticated){
23       let requiredRoles = route.data['roles'];
24       let userRoles = this.authService.roles;
25       for (let role of userRoles){
26         if (requiredRoles.includes(role)){
27           return true;
28         }
29     }
30   }
31 }
```

AuthorizationGuard > canActivate() > userRoles



Dès qu'ils ne fonctionnent pas donc on va les masquées, pour cela on ajoute une condition :

```

1 <button mat-icon-button (click)="myDrawer.toggle()">
2   <mat-icon>menu</mat-icon>
3 </button>
4 <span style="flex: auto"></span>
5 <button mat-button routerLink="/admin/home">Home</button>
6 <button mat-button routerLink="/admin/profile">Profile</button>
7 <button *ngIf="authService.roles.includes('ADMIN')" mat-button [matMenuTriggerFor]>
8   <mat-icon iconPositionEnd>keyboard_arrow_down</mat-icon>
9   Import
10 </button>
11 <mat-menu #importMenu>
12   <button mat-menu-item routerLink="/admin/LoadStudents">Load Students</button>
13   <button mat-menu-item routerLink="/admin/loadPayments">Load Payments</button>
14 </mat-menu>
15 <button mat-button *ngIf="authService.isAuthenticated">
16   {{authService.username}}
17 </button>
18 <button mat-raised-button color="accent" (click)="logout()">
19   Logout
20 </button>

```

Maintenant, on veut afficher la liste des payments, pour cela on revient vers notre composant payments, pour injecter le module HttpClient qu'on a ajouté dans nos modules, et dans ngOnInit, on va envoyer une requête vers le backend pour récupérer le backend et stocker les données :

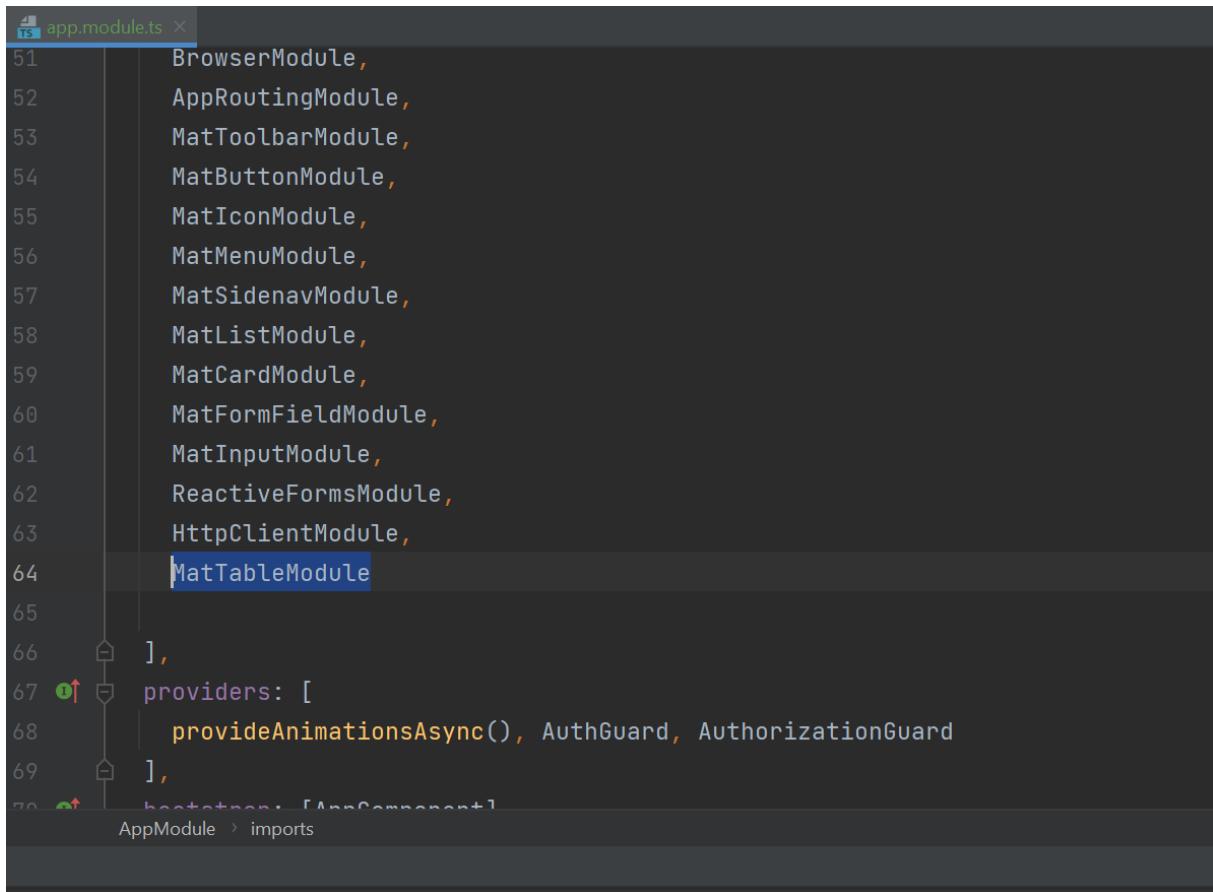
```

1 <!-- templateUrl: './payments.component.html',
2 styleUrls: ['./payments.component.css'] -->
3
4
5 @Component({
6   selector: 'app-payments',
7   templateUrl: './payments.component.html',
8   styleUrls: ['./payments.component.css']
9 })
10 export class PaymentsComponent implements OnInit{
11   public payments: any;
12
13   constructor(private http: HttpClient) {
14   }
15
16   ngOnInit(): void {
17     this.http.get( url: "http://localhost:8021/payments" ).subscribe( observerOrNext: {
18       next : data => {
19         this.payments = data;
20       },
21       error: err => {
22         console.log(err);
23       }
24     })
25   }
26 }

```

Donc maintenant on va utiliser un matériel table, alors on revient vers notre fichier html du composant payments et on coller le code qu'on a appris de la documentation.

Donc on va d'abord pour ajouter le module MatTableModule :



```
TS app.module.ts ×
51   BrowserModule,
52   AppRoutingModule,
53   MatToolbarModule,
54   MatButtonModule,
55   MatIconModule,
56   MatMenuModule,
57   MatSidenavModule,
58   MatListModule,
59   MatCardModule,
60   MatFormFieldModule,
61   MatInputModule,
62   ReactiveFormsModule,
63   HttpClientModule,
64   MatTableModule
65
66 ],
67 providers: [
68   provideAnimationsAsync(), AuthGuard, AuthorizationGuard
69 ],
70 bootstrap: [AppComponent]
```

AppModule > imports

Puis on doit déclarer ce qu'on appelle dataSource et l'initialiser, et aussi en déclare displayedColumns pour afficher les colonnes voulus dans le type script de notre composant :

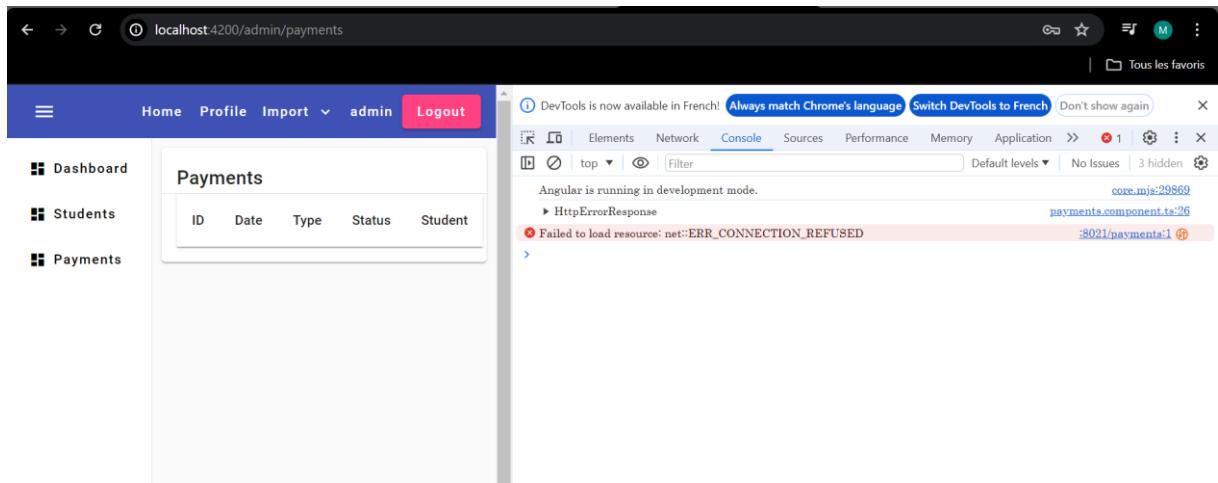
```

32     <td mat-cell *matCellDef="let element"> {{element.type}} </td>
33   </ng-container>
34
35   <!-- status Column -->
36   <ng-container matColumnDef="status">
37     <th mat-header-cell *matHeaderCellDef> Status </th>
38     <td mat-cell *matCellDef="let element"> {{element.status}} </td>
39   </ng-container>
40
41   <!-- firstName Column -->
42   <ng-container matColumnDef="firstName">
43     <th mat-header-cell *matHeaderCellDef> Student </th>
44     <td mat-cell *matCellDef="let element"> {{element.student.firstName}} </td>
45   </ng-container>
46
47   <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
48   <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
49 </table>
50

```

div.container > mat-card > mat-card-content > table.mat-elevation-z1 > ng-container > td

On teste et en trouve une erreur c'est que l'url est protégé par CORS, en fait ce qui se passe c'est que nous avons le backend dans un domaine et nous avons le frontend dans un autre domaine, alors techniquement parlant on n'a pas le droit de récupérer une page à partir d'un domaine x et d'envoyer une requête http vers un domaine y, c'est pour ça il faut demander l'autorisation du backend :



Après on va voir ça avec spring security, maintenant on va faire quelque chose de plus simple, on va vers notre contrôleur dans le backend et on va ajouter l'annotation `@CrossOrigin(' * ')`, cet étoile veut dire que le backend autorise n'importe quelles pages de n'importe quelles domaines de faire appel à ces services :

```

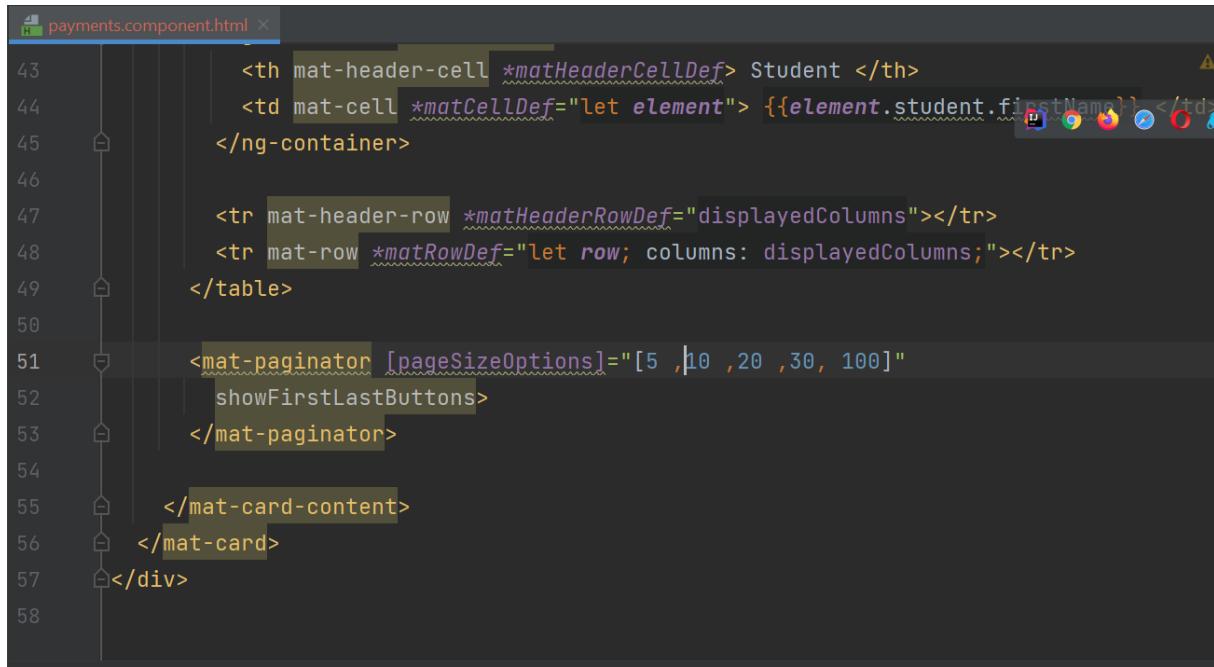
19 import java.time.LocalDate;
20 import java.util.List;
21 import java.util.UUID;
22
23 @RestController
24 @CrossOrigin("*")
25 public class PaymentRestController {
26     private StudentRepository studentRepository;
27     private PaymentRepository paymentRepository;
28     private PaymentService paymentService;
29
30     public PaymentRestController(StudentRepository studentRepository, PaymentRepository paymentRepository, PaymentService paymentService) {
31         this.studentRepository = studentRepository;
32         this.paymentRepository = paymentRepository;
33         this.paymentService = paymentService;
34     }
35
36     @GetMapping(path = "/payments")
37     public List<Payment> allPayments() { return paymentRepository.findAll(); }
38
39

```

On démarre l'application, et cette fois ci on voie que les payments s'affichent :

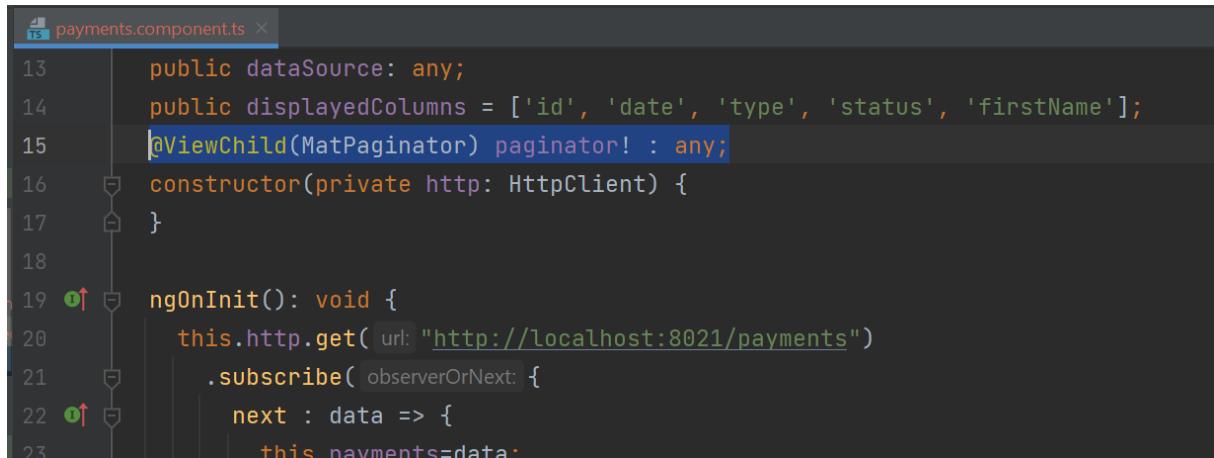
ID	Date	Type	Status	Student
1	2024-06-15	CHECK	CREATED	Meryeme
2	2024-06-15	CASH	CREATED	Meryeme
3	2024-06-15	CASH	CREATED	Meryeme
4	2024-06-15	CHECK	CREATED	Meryeme
5	2024-06-15	CASH	CREATED	Meryeme
6	2024-06-15	DEPOSIT	CREATED	Meryeme

Alors maintenant on va faire la pagination, on prend juste le code de mat-paginator du documentation et on l'ajoute à notre table après qu'on a ajouté dans les modules :



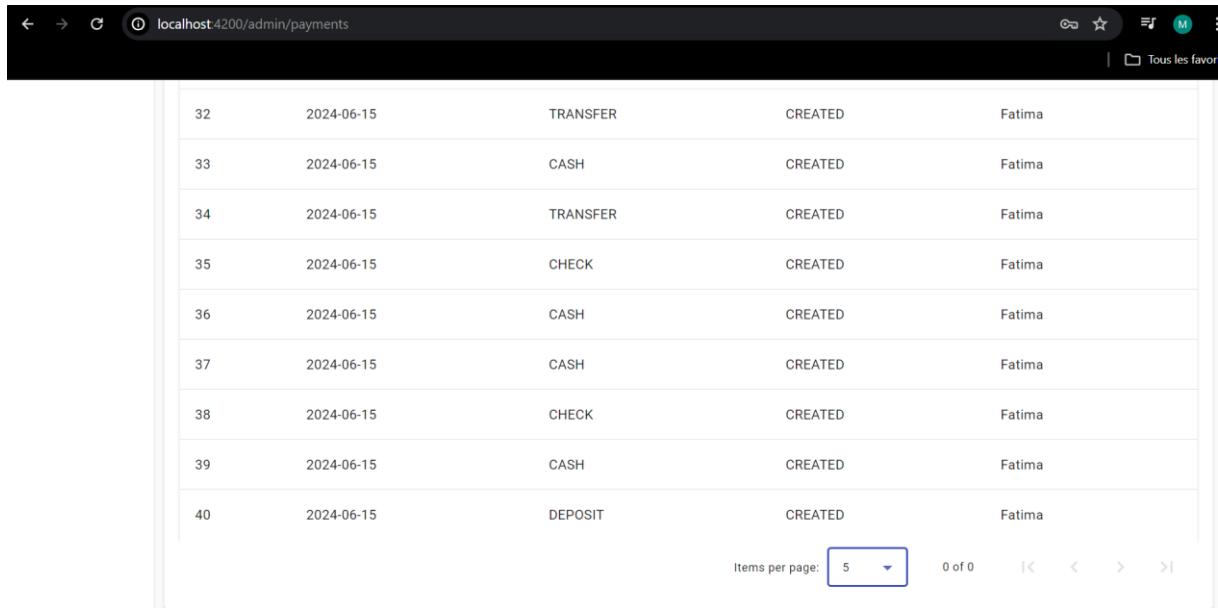
```
43     <th mat-header-cell *matHeaderCellDef> Student </th>
44     <td mat-cell *matCellDef="let element"> {{element.student.firstName}} </td>
45   </ng-container>
46
47   <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
48   <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
49 </table>
50
51   <mat-paginator [pageSizeOptions]=[5 ,10 ,20 ,30 , 100]>
52     showFirstLastButtons
53   </mat-paginator>
54
55   </mat-card-content>
56 </mat-card>
57 </div>
58
```

Puis dans le type script il nous faut déclarer un @ViewChild(MatPaginator) que l'on va appeler paginator, et qu'il va chercher dans la partie html un objet de type MatPaginator et on va l'affecter à la variable paginator :



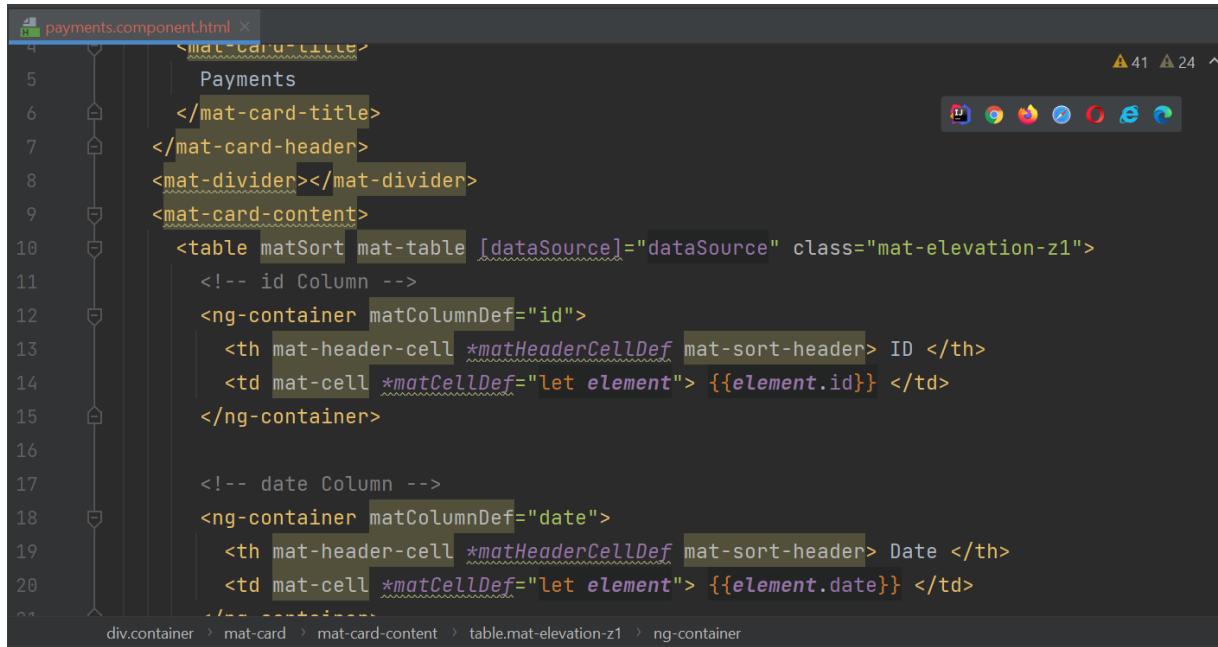
```
13   public dataSource: any;
14   public displayedColumns = ['id', 'date', 'type', 'status', 'firstName'];
15   @ViewChild(MatPaginator) paginator!: any;
16   constructor(private http: HttpClient) {
17   }
18
19   ngOnInit(): void {
20     this.http.get(url: "http://localhost:8021/payments")
21       .subscribe(observerOrNext: {
22         next : data => {
23           this.payments=data;
```

On voie maintenant la pagination :



32	2024-06-15	TRANSFER	CREATED	Fatima
33	2024-06-15	CASH	CREATED	Fatima
34	2024-06-15	TRANSFER	CREATED	Fatima
35	2024-06-15	CHECK	CREATED	Fatima
36	2024-06-15	CASH	CREATED	Fatima
37	2024-06-15	CASH	CREATED	Fatima
38	2024-06-15	CHECK	CREATED	Fatima
39	2024-06-15	CASH	CREATED	Fatima
40	2024-06-15	DEPOSIT	CREATED	Fatima
Items per page: 5 0 of 0 < < > >				

Maintenant on va ajouter le tri, pour cela il nous faut ajouter dans table de la partie html l'élément matSort après qu'on a ajouté dans les modules, la deuxième chose qu'on va faire est dans chaque colonne on ajoute mat-sort-header :



```

payments.component.html
1<div>
2  <mat-card>
3    <mat-card-header>
4      <mat-card-title>Payments</mat-card-title>
5    </mat-card-header>
6    <mat-divider></mat-divider>
7    <mat-card-content>
8      <table matSort mat-table [dataSource]="dataSource" class="mat-elevation-z1">
9        <!-- id Column -->
10       <ng-container matColumnDef="id">
11         <th mat-header-cell *matHeaderCellDef mat-sort-header> ID </th>
12         <td mat-cell *matCellDef="let element"> {{element.id}} </td>
13       </ng-container>
14
15        <!-- date Column -->
16        <ng-container matColumnDef="date">
17          <th mat-header-cell *matHeaderCellDef mat-sort-header> Date </th>
18          <td mat-cell *matCellDef="let element"> {{element.date}} </td>
19        </ng-container>
20
21      </table>
22    </mat-card-content>
23  </mat-card>
24</div>

```

Et la dernière chose c'est qu'on va ajouter un autre @ViewChild de type MatSort :

```
payments.component.ts
```

```
11  })
12  export class PaymentsComponent implements OnInit{
13      public payments: any;
14      public dataSource: any;
15      public displayedColumns = ['id', 'date', 'type', 'status', 'firstName'];
16      @ViewChild(MatPaginator) paginator!: any;
17      @ViewChild(MatSort) sort!: MatSort;
18      constructor(private http: HttpClient) {
19      }
20
21  ngOnInit(): void {
```

Et comme ça on fait le tri :

The screenshot shows a web application interface. At the top, there is a navigation bar with icons for back, forward, search, and other browser functions. The URL is 'localhost:4200/admin/payments'. Below the navigation bar, there is a sidebar with two items: 'Students' and 'Payments'. The 'Payments' item is selected and highlighted in blue. To the right of the sidebar is a table titled 'Payments'. The table has columns: ID, Date, Type, Status, and Student. There are five rows of data:

ID	Date	Type	Status	Student
1	2024-06-15	TRANSFER	CREATED	Meryeme
2	2024-06-15	CHECK	CREATED	Meryeme
3	2024-06-15	DEPOSIT	CREATED	Meryeme
4	2024-06-15	TRANSFER	CREATED	Meryeme
5	2024-06-15	CHECK	CREATED	Meryeme

At the bottom of the table, there is a pagination control with a dropdown for 'Items per page' set to 5, and links for '1 - 5 of 40', '<', '>', and '>>'. The entire table is contained within a light gray box.