

SYSTÈMES DISTRIBUÉS

MASTER INTELLIGENCE
ARTIFICIELLE ET ANALYSE
DES DONNÉES

RÉALISÉE PAR :
ANOADA NOHAYLA

ENCADRÉ PAR :
M. MOHAMED YOUSSEFI

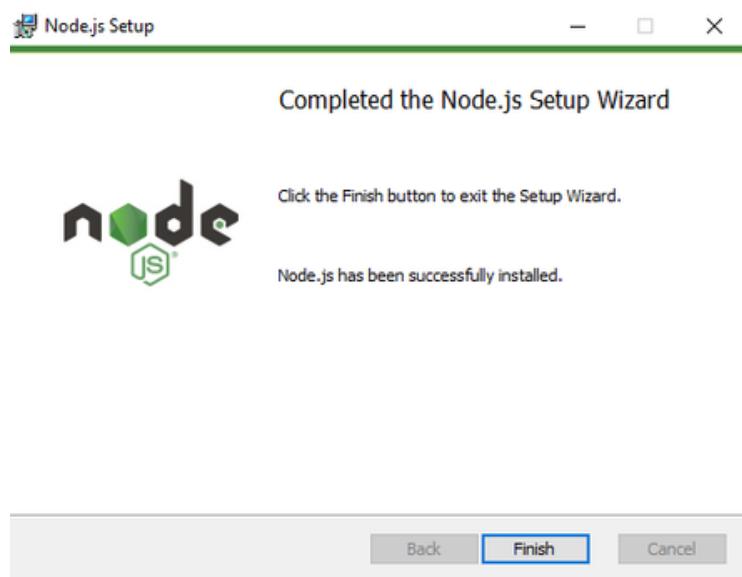


ACTIVITÉ PRATIQUE N° 4

ANGULAR - SPRING

PARTIE 1 :

1. TÉLÉCHARGEMENT ET INSTALLATION DE NODE.JS



téléchargement et installation de Node.js depuis le site officiel. L'installation a été confirmée par l'assistant d'installation.

2. VÉRIFICATION DE L'INSTALLATION DE NPM

```
C:\Users\nohayla>npm --version  
10.5.2
```

3. VÉRIFICATION DE L'ANGULAR CLI

```
C:\Users\nohayla>ng version

Angular CLI: 17.3.7
Node: 20.13.1
Package Manager: npm 10.5.2
OS: win32 x64

Angular: undefined
...
Package          Version
-----
@angular-devkit/architect    0.1703.7 (cli-only)
@angular-devkit/core         17.3.7 (cli-only)
@angular-devkit/schematics   17.3.7 (cli-only)
@schematics/angular          17.3.7 (cli-only)
```

4. INSTALLATION DE JSON-SERVER

```
C:\Users\nohayla>npm i -g json-server@0.17.4
added 116 packages in 15s
```

PARTIE 2:

1. CRÉATION DU PROJET ANGULAR

```
C:\Users\nohayla\Desktop\study\school\Master\S2\Systèmes_Distribués\Tp\Anoada_Nohayla_Tp4
>ng new Anoada_Nohayla_Tp4_1
? 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)? Yes
CREATE Anoada_Nohayla_Tp4_1/angular.json (2908 bytes)
CREATE Anoada_Nohayla_Tp4_1/package.json (1305 bytes)
CREATE Anoada_Nohayla_Tp4_1/README.md (1098 bytes)
CREATE Anoada_Nohayla_Tp4_1/tsconfig.json (889 bytes)
CREATE Anoada_Nohayla_Tp4_1/.editorconfig (290 bytes)
CREATE Anoada_Nohayla_Tp4_1/.gitignore (629 bytes)
CREATE Anoada_Nohayla_Tp4_1/tsconfig.app.json (342 bytes)
CREATE Anoada_Nohayla_Tp4_1/tsconfig.spec.json (287 bytes)
CREATE Anoada_Nohayla_Tp4_1/server.ts (1759 bytes)
CREATE Anoada_Nohayla_Tp4_1/.vscode/extensions.json (134 bytes)
CREATE Anoada_Nohayla_Tp4_1/.vscode/launch.json (490 bytes)
CREATE Anoada_Nohayla_Tp4_1/.vscode/tasks.json (980 bytes)
CREATE Anoada_Nohayla_Tp4_1/src/main.ts (256 bytes)
CREATE Anoada_Nohayla_Tp4_1/src/favicon.ico (15086 bytes)
CREATE Anoada_Nohayla_Tp4_1/src/index.html (316 bytes)
CREATE Anoada_Nohayla_Tp4_1/src/styles.css (81 bytes)
CREATE Anoada_Nohayla_Tp4_1/src/main.server.ts (271 bytes)
CREATE Anoada_Nohayla_Tp4_1/src/app/app.component.html (20239 bytes)
CREATE Anoada_Nohayla_Tp4_1/src/app/app.component.spec.ts (987 bytes)
CREATE Anoada_Nohayla_Tp4_1/src/app/app.component.ts (329 bytes)
CREATE Anoada_Nohayla_Tp4_1/src/app/app.component.css (0 bytes)
CREATE Anoada_Nohayla_Tp4_1/src/app/app.config.ts (330 bytes)
CREATE Anoada_Nohayla_Tp4_1/src/app/app.routes.ts (80 bytes)
CREATE Anoada_Nohayla_Tp4_1/src/app/app.config.server.ts (361 bytes)
CREATE Anoada_Nohayla_Tp4_1/src/assets/.gitkeep (0 bytes)
✓ Packages installed successfully.
warning: in the working copy of 'package-lock.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'package.json', LF will be replaced by CRLF the next time Git touches it
      Successfully initialized git.
```

2. LANCER LE SERVEUR DE DÉVELOPPEMENT

```
PS C:\Users\nohayla\Desktop\study\school\Master\S2\Systèmes_Distribués\Tp\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> ng serve
✔ Browser application bundle generation complete.

Initial Chunk Files | Names           | Raw Size
vendor.js           | vendor          | 2.04 MB |
polyfills.js        | polyfills       | 313.74 kB |
styles.css, styles.js | styles          | 208.86 kB |
main.js             | main            | 48.77 kB |
runtime.js          | runtime         | 6.54 kB |

| Initial Total | 2.61 MB

Build at: 2024-05-20T21:13:45.021Z - Hash: 8feadb1b1da1cc0a - Time: 79
608ms

** Angular Live Development Server is listening on localhost:4200, open
in your browser on http://localhost:4200/ **
```

3. INSTALLATION DE BOOTSTRAP

```
PS C:\Users\nohayla\Desktop\study\school\Master\S2\Systèmes_Distribués\Tp\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> npm i bootstrap bootstrap-icons
added 3 packages, removed 1 package, and audited 987 packages in 27s

  styles.css ×
  1  /* You can add global styles to this file, and also
  2   @import "bootstrap-icons/font/bootstrap-icons.css";

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

3. INSTALLATION DE BOOTSTRAP

```
PS C:\Users\nohayla\Desktop\study\school\Master\S2\Systèmes_Distribués\Tp\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> ng g c home
CREATE src/app/home/home.component.html (19 bytes)
CREATE src/app/home/home.component.spec.ts (585 bytes)
CREATE src/app/home/home.component.ts (194 bytes)
CREATE src/app/home/home.component.css (0 bytes)
UPDATE src/app/app.module.ts (467 bytes)
PS C:\Users\nohayla\Desktop\study\school\Master\S2\Systèmes_Distribués\Tp\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> ng g c products
CREATE src/app/products/products.component.html (23 bytes)
CREATE src/app/products/products.component.spec.ts (613 bytes)
CREATE src/app/products/products.component.ts (210 bytes)
CREATE src/app/products/products.component.css (0 bytes)
UPDATE src/app/app.module.ts (557 bytes)
```

```

PS C:\Users\nohayla\Desktop\study\school\Master\S2\Systèmes_Distribués\Tp\
Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> ng g c new-product
CREATE src/app/new-product/new-product.component.html (26 bytes)
CREATE src/app/new-product/new-product.component.spec.ts (628 bytes)
CREATE src/app/new-product/new-product.component.ts (221 bytes)
CREATE src/app/new-product/new-product.component.css (0 bytes)

```

4. BASE DE DONNÉE

```

{
  "users": [
    {
      "id": "user1",
      "password": "MTIzNA==",
      "roles": [
        "USER"
      ],
      "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyMSIsImlhCI6MTUxNjMwOTk3fQ.4Hg4DmZC9WYVtPjLJyfzvKJF0BdOOGqL8DgTzJmzQ"
    },
    {
      "id": "nohayla",
      "password": "MTIzNA==",
      "roles": [
        "USER"
      ],
      "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyMiIsImlhCI6MTUxNjMwOTk3fQ.4Hg4DmZC9WYVtPjLJyfzvKJF0BdOOGqL8DgTzJmzQ"
    },
    {
      "id": "admin",
      "password": "MTIzNA==",
      "roles": [
        "USER",
        "ADMIN"
      ],
      "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZG1pbkiIsImlhCI6MTUxNjMwOTk3fQ.4Hg4DmZC9WYVtPjLJyfzvKJF0BdOOGqL8DgTzJmzQ"
    }
  ],
  "products": [
    {
      "id": "1",
      "name": "Computer",
      "price": 8000,
      "checked": true
    },
    {
      "id": "2",
      "name": "Computer",
      "price": 7000,
      "checked": true
    },
    {
      "id": "3",
      "name": "Smart phone",
      "price": 3000,
      "checked": false
    },
    {
      "id": "5",
      "name": "Smart phone",
      "price": "4500",
      "checked": true
    }
  ]
}

```

Le fichier db.json sert de base de données pour les utilisateurs et les produits. Il est utilisé par le serveur JSON pour stocker et gérer les données.

- **Users:** Contient les informations des utilisateurs, y compris les identifiants, les mots de passe (encodés en Base64), les rôles (USER, ADMIN) et les tokens JWT pour l'authentification.
- **Products:** Contient les détails des produits avec des attributs tels que id, name, price, et checked (indiquant si le produit est coché ou non).

5. LANCER L'APPLICATION

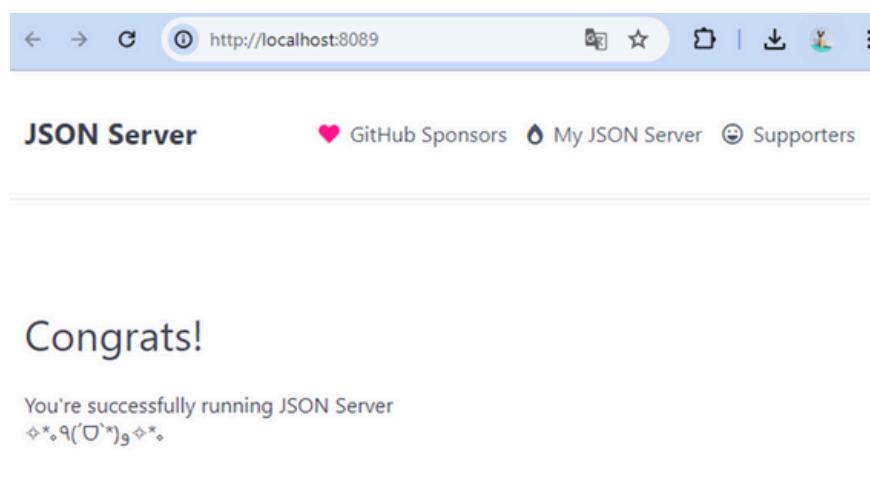
5.1 LANCER LE SERVEUR ANGULAR:

```
\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> ng serve
Browser bundles
main.server.mjs      | main.server          | 428.13 kB |
chunk-VPSODEBW.mjs  | -                  | 2.51 kB |
render-utils.server.mjs | render-utils.server | 423 bytes |

Lazy chunk files      | Names           | Raw size
chunk-OTT6LQ5K.mjs   | xhr2            | 39.10 kB |

Application bundle generation complete. [45.933 seconds]

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



5.2 LANCER LE SERVEUR JSON:

```
\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> json-server -w data/db.json -p 8089

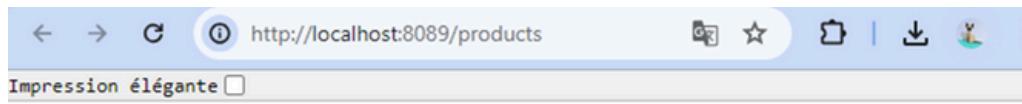
\{^_^}/ hi!

Loading data/db.json
Done

Resources
http://localhost:8089/users
http://localhost:8089/products

Home
http://localhost:8089

Type s + enter at any time to create a snapshot of the database
Watching...
```



The screenshot shows a web browser window with the URL <http://localhost:8089/products> in the address bar. The page content displays a JSON array of product objects.

```
[{"id": "1", "name": "Computer", "price": 8000, "checked": true}, {"id": "2", "name": "Computer", "price": 7000, "checked": true}, {"id": "3", "name": "Smart phone", "price": 3000, "checked": false}, {"id": "4", "name": "Laptop", "price": "12000", "checked": true}, {"id": "5", "name": "Smart phone", "price": "4500", "checked": true}, {"id": "uP8J0zs", "name": "Smart phone", "price": "12000", "checked": true}]
```

```

[{"id": "user1", "password": "MTIzNA==", "roles": ["USER"], "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyMSIsImhdCI6MTUxNjIzOTAyMiwiZXhwIjoxNTE2MjM5MDIyLCJpc3MiOiJodHRwOi8vbG9jYXob3N0Ojg4MDgvYXV0aCIsIm5iZiI6MTUxNjIzOTAyMiwiZmlyc3ROYW1lIjoiTW9oYW1tZWQiLCJsYXN0TmFtZSI6I1lPVVNTRkkilCJ1bwFpbCI6Im1lZEByb2xlcyI6WyJVU0VSIl19.kATxOoGzNi6NOralr_2k2i-ekK2gI0w7G7ty0t031n8"}, {"id": "nohayla", "password": "MTIzNA==", "roles": ["USER"], "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyMiIsImhdCI6MTUxNjIzOTAyMiwiZXhwIjoxNTE2MjM5MDIyLCJpc3MiOiJodHRwOi8vbG9jYXob3N0Ojg4MDgvYXV0aCIsIm5iZiI6MTUxNjIzOTAyMiwiZmlyc3ROYW1lIjoiTW9oYW1tZWQiLCJsYXN0TmFtZSI6I1lPVVNTRkkilCJ1bwFpbCI6Im1lZEByb2xlcyI6WyJVU0VSIl19.RhY-9z5rywbJH0qwGO5PPxQafUx1ZSCnMXJ2y8qy7w"}, {"id": "admin", "password": "MTIzNA==", "roles": ["USER", "ADMIN"], "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZG1pbisImhdCI6MTUxNjIzOTAyMiwiZXhwIjoxNTE2MjM5MDIyLCJpc3MiOiJodHRwOi8vbG9jYXob3N0Ojg4MDgvYXV0aCIsIm5iZiI6MTUxNjIzOTAyMiwiZmlyc3ROYW1lIjoiTW9oYW1tZWQiLCJsYXN0TmFtZSI6I1lPVVNTRkkilCJ1bwFpbCI6Im1lZEByb2xlcyI6WyJVU0VSIlwiQUFRNSU4iXX0.g6LoXcTSM3y6ZxaRBCDJplvJ-KSrtjCNf93JC7SoKrA"}]

```

G. COMPOSANTS

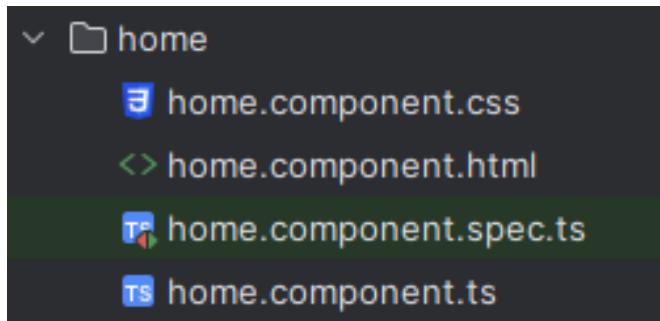
G.1 HOME

- CRÉATION DÉ COMPOSANT

```
\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> ng g c home
CREATE src/app/home/home.component.html (20 bytes)
CREATE src/app/home/home.component.spec.ts (605 bytes)
CREATE src/app/home/home.component.css (0 bytes)
```

Cette commande permet de générer un composant angulaire nommé "home" à l'aide de Angular CLI

la commande génère les fichiers suivants :



- **src/app/home/home.component.ts:**
Fichier TypeScript pour la logique du composant.
- **src/app/home/home.component.html**
Template HTML pour le composant.
- **src/app/home/home.component.css)**
Feuille de style pour le composant.
- **src/app/home/home.component.spec.ts**
Fichier de test unitaire pour le composant.
- **TYPESCRIPT (HOME.COMPONENT.TS)**

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

1+ usages
@Component({
  selector: 'app-home',
  standalone: true,
  imports: [],
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css'
})
export class HomeComponent
```

- **TEMPLATE HTML
(HOME.COMPONENT.HTML)**

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

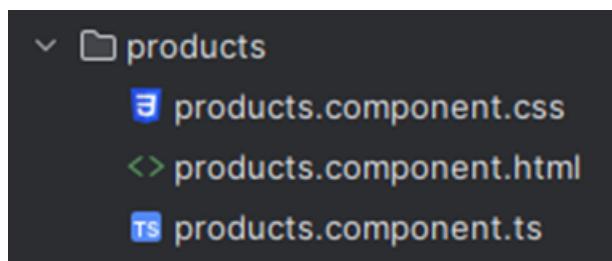
Home Component

6.2 PRODUCTS

- **CRÉATION DE COMPOSANT**

```
\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> ng g c products
CREATE src/app/products/products.component.ts (254 bytes)
CREATE src/app/products/products.component.css (0 bytes)
```

Cette commande vous permet de générer un composant angular nommé "products" à l'aide de Angular CLI
la commande génère les fichiers suivants :



- **src/app/products/products.component.ts:**
Fichier TypeScript pour la logique du composant.
- **src/app/products/products.component.html**
Template HTML pour le composant.
- **src/app/products/products.component.css**
Feuille de style pour le composant.
- **src/app/products/products.component.spec.ts**
Fichier de test unitaire pour le composant.

• TYPESCRIPT (PRODUCTS.COMPONENT.TS)

```
@Component({
  selector: 'app-products',
  standalone: true,
  imports: [
    HttpClientModule,
    NgForOf,
    AsyncPipe,
    FormsModule,
    NgIf,
    NgClass
  ],
  templateUrl: './products.component.html',
  styleUrls: ['./products.component.css'
})
export class ProductsComponent implements OnInit
{
  no usages
  constructor(private productService:ProductService,
              private router: Router,
              public appState: AppStateService)
  {}
  no usages
  ngOnInit(): void
  searchProducts(): void
  {
    /*this.appState.setProductState({
      status : "LOADING"
   });*/
    this.productService.searchProducts(
      this.appState.productsState.keyword,
      this.appState.productsState.currentPage,
      this.appState.productsState.pageSize)
      .subscribe( observerOrNext: {
        next: (resp : HttpResponse<Object> ) : void  => {
          let products : Product[]  = resp.body as Product[];

          let totalProducts: number = parseInt(resp.headers.get('x-total-count'))!;

          // this.appState.productsState.totalProducts=totalProducts;

          let totalPages : number = Math.floor( totalProducts / this.appState.productsState.pageSize);

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

```

handleCheckProduct(product: Product) : void
{
  this.productService.checkProducts(product)
    .subscribe( observerOrNext: {
      next: updatedProduct : Product  => {
        product.checked=!product.checked;
        //this.searchProducts();
      },
      error: err => {
        console.log(err);
      }
    });
}

1+ usages
handleDelete(product: Product) : void
{
  if (confirm("Êtes-vous sûr de vouloir supprimer ce produit? "))
    this.productService.deleteProducts(product).subscribe(
      observerOrNext: {
        next:value => {
          this.searchProducts();
          //this.appState.productsState.products = this.appState.productsState.products.filter((p:any)=>p.id!=product.id);
        }
      }
    );
}

1+ usages
handleGotoPage(page: number) : void {
  this.appState.productsState.currentPage=page;
  this.searchProducts();
}

1+ usages
handleEdit(product: Product) : void {
  this.router.navigateByUrl( url: `/admin/editProduct/${product.id}`);
}
}

```

La classe implémente OnInit pour initialiser les données au chargement du composant.

Constructeur :

Le constructeur injecte les services nécessaires :

- **ProductService** : Pour les opérations sur les produits.
- **Router** : Pour la navigation.
- **AppStateService** : Pour gérer l'état de l'application.

Méthodes :

- **ngOnInit** : Cette méthode est appelée après l'initialisation du composant et déclenche la recherche des produits.

- **searchProducts** : Cette méthode recherche les produits en fonction du mot-clé, de la page actuelle et de la taille de la page. Elle met à jour l'état des produits et la pagination.
- **handleCheckProduct** : Cette méthode gère la vérification d'un produit, en inversant son état "checked".
- **handleDelete** : Cette méthode gère la suppression d'un produit après confirmation de l'utilisateur.
- **handleGotoPage** : Cette méthode gère la navigation vers une page spécifique et relance la recherche de produits.
- **handleEdit** : Cette méthode gère la navigation vers la page d'édition d'un produit.

Le composant **ProductsComponent** fournit une interface utilisateur complète pour gérer une liste de produits. Il permet de rechercher, afficher, vérifier, supprimer, éditer des produits et de naviguer entre les pages de résultats. Le composant utilise des services pour effectuer des opérations CRUD et gérer l'état de l'application.

• TEMPLATE HTML (C.COMPONENT.HTML)

```
<div class="p-3">
  <div class="card">
    <div class="card-body">
      <div class="card-body">
        {{appState.productsState.keyword}}
        <input type="text" [(ngModel)]="appState.productsState.keyword" >
        <button (click)="searchProducts()" class="btn btn-outline-success ms-1">
          <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>
        <button (click)="handleCheckProduct(product)" class="btn btn-outline-success">
          <i [class] = "product.checked?'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-success">
          <i class="bi bi-pencil"></i>
        </button>
      </td>
    </tr>
  </tbody>
</table>
<ul class="nav nav-pills" *ngIf="appState.productsState.totalPages && appState.productsState.totalPages > 0">
  <li *ngFor="let page of [].constructor(this.appState.productsState.totalPages); let i=index">
    <button (click)="handleGotoPage(page: i+1)" [ngClass]="appState.productsState.currentPage==(i+1)?'btn-success' : 'btn-outline-success'" class="btn m-1">
      {{i+1}}
    </button>
  </li>
</ul>
</div>
</div>
</div>

```

Ce code HTML crée une interface utilisateur pour gérer une liste de produits. Il permet aux utilisateurs de rechercher des produits par mot-clé, d'afficher les produits dans une table avec des options pour les vérifier, les supprimer ou les éditer, et de naviguer entre différentes pages de résultats. La mise en page utilise des classes Bootstrap pour le style et la présentation.

The screenshot shows a user interface for managing a list of products. At the top is a search bar with a magnifying glass icon. Below it is a table with three columns: Name, Price, and Checked. The table contains three rows of data:

Name	Price	Checked
Computer	8000	<input checked="" type="checkbox"/>
Computer	7000	<input checked="" type="checkbox"/>
Smart phone	3000	<input type="checkbox"/>

Each row has three buttons in the last column: a red trash can icon for deletion, a green pencil icon for editing, and a green checkmark icon for marking as checked. At the bottom of the table is a navigation bar with three buttons labeled 1, 2, and 3.

com com

Name	Price	Checked		
Computer	8000	✓		
Computer	7000	✓		

1



com com

Name	Price	Checked		
Computer	8000	✓		
Computer	7000	✓		

1

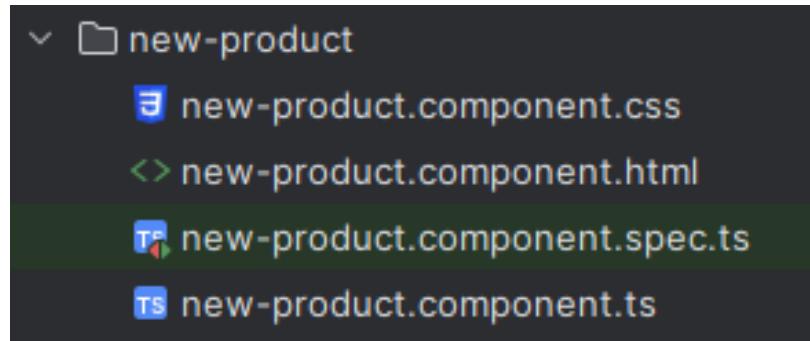
6.3 NEW-PRODUCT

• CRÉATION DE COMPOSANT

```
\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> 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 (648 bytes)
CREATE src/app/new-product/new-product.component.ts (265 bytes)
CREATE src/app/new-product/new-product.component.css (0 bytes)
```

Cette commande vous permet de générer un composant angulaire nommé "new-product" à l'aide de Angular CLI

la commande génère les fichiers suivants :



- **src/app/new-product/new-product.component.ts:**
Fichier TypeScript pour la logique du composant.
- **src/app/new-product/new-product.component.html**
Template HTML pour le composant.
- **src/app/new-product/new-product.component.css)**
Feuille de style pour le composant.
- **src/app/new-product/new-product.component.spec.ts**
Fichier de test unitaire pour le composant.

• TYPESCRIPT (NEW-PRODUCT.COMPONENT.TS)

```
@Component({
  selector: 'app-new-product',
  standalone: true,
  imports: [
    ReactiveFormsModule,
    JsonPipe
  ],
  templateUrl: './new-product.component.html',
  styleUrls: ['./new-product.component.css']
})
export class NewProductComponent implements OnInit
{
  public productForm!:FormGroup;
  no usages
  constructor(private fb:FormBuilder, private productService:ProductService)
  {
  }
  no usages
  ngOnInit(): void
  {
    this.productForm=this.fb.group( controls: {
      name: this.fb.control( formState: '' , validatorOrOpts: [Validators.required]),
      price : this.fb.control( formState: 0),
      checked : this.fb.control( formState: false),
    });
  }
}
```

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

La classe implémente OnInit pour initialiser les données au chargement du composant.

Constructeur

Le constructeur injecte les services nécessaires :

- FormBuilder : Pour créer et gérer les formulaires réactifs.
- ProductService : Pour les opérations sur les produits.

Méthodes :

- ngOnInit : Cette méthode est appelée après l'initialisation du composant et initialise le formulaire du produit.
- saveProduct : Cette méthode est appelée lors de la soumission du formulaire pour sauvegarder le produit.

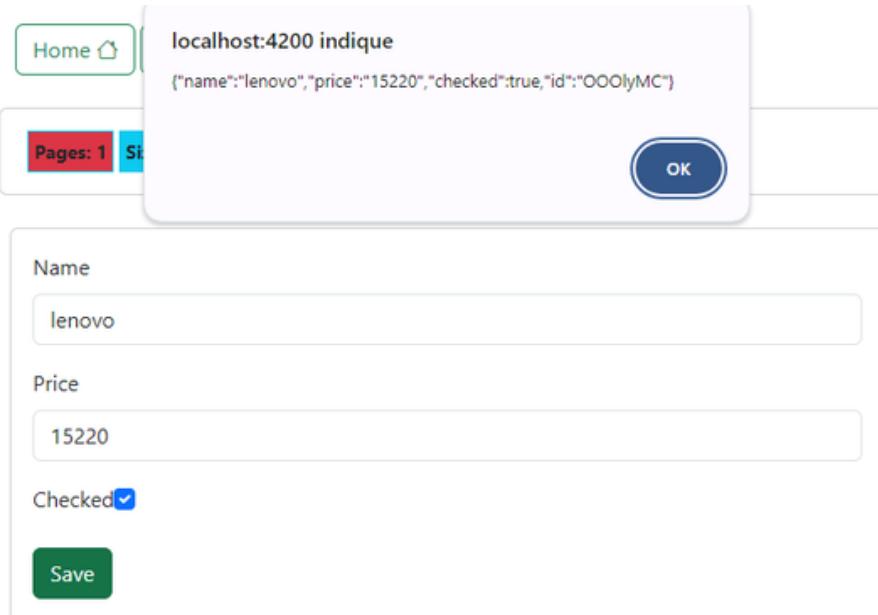
Le NewProductComponent fournit une interface utilisateur pour créer un nouveau produit via un formulaire réactif. Le composant valide les entrées utilisateur et envoie les données du produit au serveur pour les sauvegarder. Le composant utilise les bonnes pratiques d'Angular pour la gestion des formulaires et l'interaction avec les services.

• TEMPLATE HTML (NEW-PRODUCT.COMPONENT.HTML)

```
<div class="p-3">
  <div class="card">
    <div class="card-body">
      <div class="row">
        <div class="col-md-6">
          <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 [disabled]="productForm.invalid" class="btn btn-success">Save</button>
          </form>
        </div>
      </div>
    </div>
  </div>
</div>
```

Ce code HTML fournit une interface utilisateur propre et intuitive pour créer un nouveau produit, avec des champs de formulaire pour le nom, le prix et l'état vérifié, ainsi qu'un bouton de soumission. Les classes Bootstrap et les directives Angular sont utilisées pour la mise en page et la gestion des formulaires.

The screenshot shows a user interface for creating a new product. It consists of a card with three input fields and a save button. The first field is labeled "Name" and contains an empty input field. The second field is labeled "Price" and contains the value "0". The third field is labeled "Checked" and contains a checkbox that is unchecked. Below these fields is a green "Save" button.

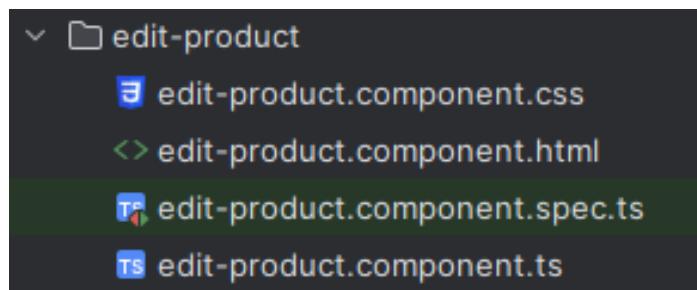


6.4 EDIT-PRODUCT

- CRÉATION DÉ COMPOSANT

```
a_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> ng generate component edit-product
CREATE src/app/edit-product/edit-product.component.html (28 bytes)
CREATE src/app/edit-product/edit-product.component.spec.ts (655 bytes)
CREATE src/app/edit-product/edit-product.component.ts (269 bytes)
CREATE src/app/edit-product/edit-product.component.css (0 bytes)
```

Cette commande vous permet de générer un composant angulaire nommé "edit-product" à l'aide de Angular CLI
la commande génère les fichiers suivants :



- **src/app/edit-product/edit-product.component.ts:**
Fichier TypeScript pour la logique du composant.
- **src/app/edit-product/edit-product.component.html**
Template HTML pour le composant.
- **src/app/edit-product/edit-product.component.css)**
Feuille de style pour le composant.
- **src/app/edit-product/edit-product.component.spec.ts**
Fichier de test unitaire pour le composant.

• TYPESCRIPT (EDIT-PRODUCT.COMPONENT.TS)

```
@Component({
  selector: 'app-edit-product',
  standalone: true,
  imports: [
    ReactiveFormsModule,
    NgIf
  ],
  templateUrl: './edit-product.component.html',
  styleUrls: ['./edit-product.component.css'
})
export class EditProductComponent implements OnInit{
  productId!: number;
  productFormGroup!: FormGroup;
  no usages
  constructor(private activatedRoute: ActivatedRoute,
              private productService: ProductService,
              private fb : FormBuilder) {
  }
  no usages
  ngOnInit(): void {
    this.productId=this.activatedRoute.snapshot.params['id'];
    this.productService.getProductById(this.productId).subscribe( observerOrNext: {
      next: (product : Product ) : void =>{
        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.min( min: 100)]),
          checked : this.fb.control(product.checked),
        });
      },
      error : err => {
        console.log(err);
      }
    });
  }
  1+ usages
  updateProduct(): void {
    let product : Product = this.productFormGroup.value;
    this.productService.updateProduct(product).subscribe( observerOrNext: {
      next : data:Product =>{
        alert(JSON.stringify(data));
      }
    });
  }
}
```

La classe implémente OnInit pour initialiser les données au chargement du composant.

Constructeur

Le constructeur injecte les services nécessaires :

- **ActivatedRoute:** Pour accéder aux paramètres de la route et obtenir l'ID du produit à éditer.
- **ProductService:** Pour les opérations sur les produits.
- **FormBuilder:** Pour créer et gérer les formulaires réactifs.

Méthodes :

- **ngOnInit :** Cette méthode est appelée après l'initialisation du composant et initialise le formulaire du produit en récupérant les données du produit par ID.
- **updateProduct :** Cette méthode est appelée lors de la soumission du formulaire pour mettre à jour le produit.

Le EditProductComponent fournit une interface utilisateur pour modifier un produit existant via un formulaire réactif. Le composant valide les entrées utilisateur et envoie les données mises à jour du produit au serveur pour les sauvegarder. Le composant utilise les bonnes pratiques d'Angular pour la gestion des formulaires et l'interaction avec les services.

• TEMPLATE HTML (EDIT-PRODUCT.COMPONENT.HTML)

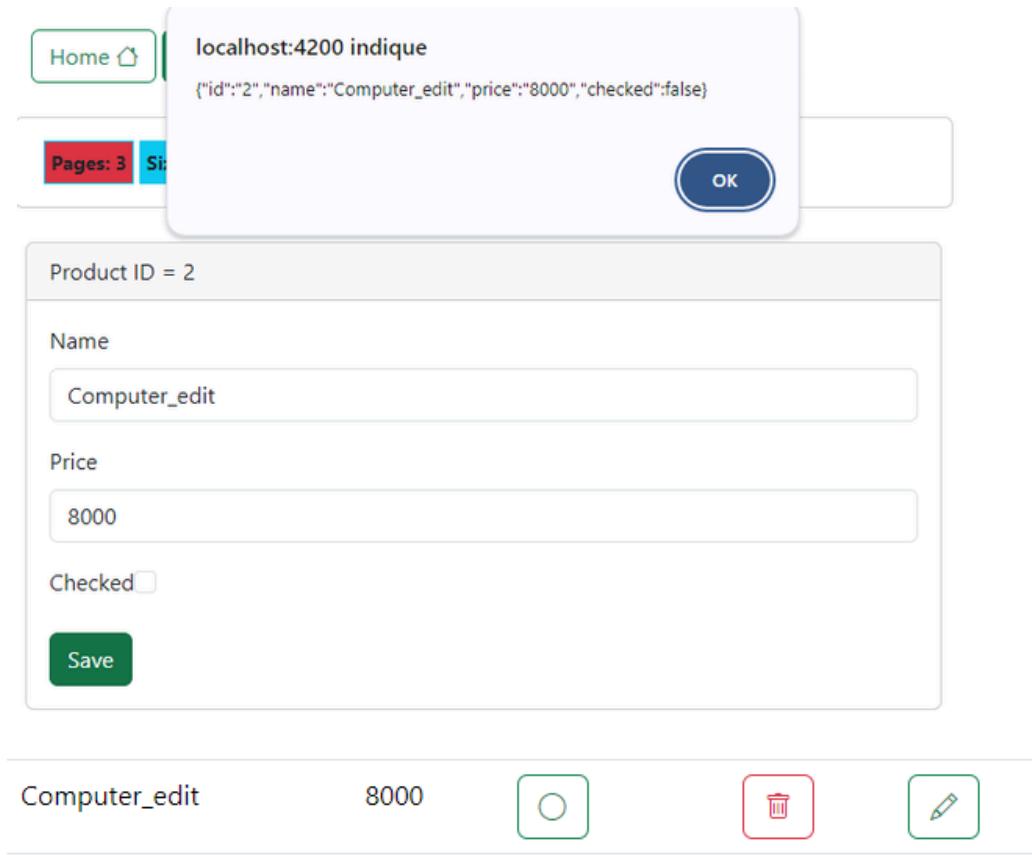
```
<div class="p-3">
  <div class="card">
    <div class="card-header">
      Product ID = {{productId}}
    </div>
    <div class="card-body" *ngIf="productFormGroup">
      <form [formGroup]="productFormGroup" (ngSubmit)="updateProduct()">
        <div class="mb-3">
          <label class="form-label">Name</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>
  </div>
</div>
```



A screenshot of the edit product form. The form fields are pre-filled with the following values:

- Product ID: 2
- Name: Computer
- Price: 7000
- Checked: Yes (indicated by a checked checkbox)

A green "Save" button is at the bottom right of the form.



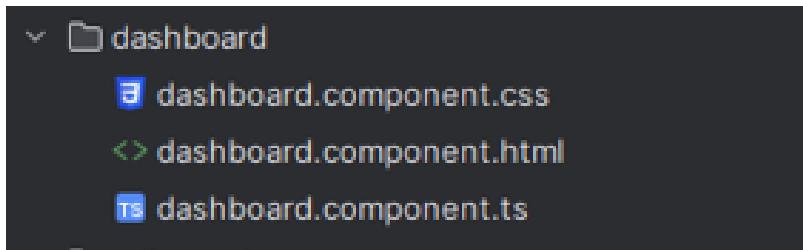
Ce code HTML fournit une interface utilisateur propre et intuitive pour modifier un produit existant, avec des champs de formulaire pour le nom, le prix et l'état vérifié, ainsi qu'un bouton de soumission. Les classes Bootstrap et les directives Angular sont utilisées pour la mise en page et la gestion des formulaires.

6.5 DASHBOARD

• CRÉATION DE COMPOSANT

```
ada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> ng g c dashboard
CREATE src/app/dashboard/dashboard.component.html (25 bytes)
CREATE src/app/dashboard/dashboard.component.spec.ts (640 bytes)
CREATE src/app/dashboard/dashboard.component.ts (258 bytes)
CREATE src/app/dashboard/dashboard.component.css (0 bytes)
```

Cette commande vous permet de générer un composant angulaire nommé "dashboard" à l'aide de Angular CLI
la commande génère les fichiers suivants :



- **src/app/dashboard/dashboard.component.ts:**
Fichier TypeScript pour la logique du composant.
 - **src/app/dashboard/dashboard.component.html**
Template HTML pour le composant.
 - **src/app/dashboard/dashboard.component.css)**
Feuille de style pour le composant.
 - **src/app/dashboard/dashboard.component.spec.ts**
Fichier de test unitaire pour le composant.
- **TYPESCRIPT (DASHBOARD.COMPONENT.TS)**

```
@Component({
  selector: 'app-dashboard',
  standalone: true,
  imports: [],
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css'
})
export class DashboardComponent
{
  no usages
  constructor(public appState:AppStateService)
  {}
  1+ usages
  totalCheckedProducts()
  {
    let checkedProducts = this.appState.productsState.products.filter((p:any)=>p.checked==true);
    return checkedProducts.length;
  }
}
```

Constructeur

Le constructeur Injecte le service AppStateService pour accéder à l'état des produits.

Méthodes :

- **totalCheckedProducts:** Cette méthode filtre les produits pour ne garder que ceux qui sont vérifiés (checked == true), puis il retourne le nombre de produits vérifiés.

• TEMPLATE HTML (DASHBOARD.COMPONENT.HTML)

```
<small>
  <nav class="p-1 m-1 text-white fw-bold">
    <div class="card">
      <div class="card-body">
        <ul class="nav nav-pills">
          <li>
            <div class="border border-info bg-danger p-1 ms-1">
              Pages: {{appState.productsState.totalPages}}
            </div>
          </li>
          <li>
            <div class="border border-info bg-info p-1 ms-1">
              Size: {{appState.productsState.pageSize}}
            </div>
          </li>
          <li>
            <div class="border border-info bg-success p-1 ms-1">
              Total: {{appState.productsState.totalProducts}}
            </div>
          </li>
          <li>
            <div class="border border-info bg-primary p-1 ms-1">
              Checked: {{totalCheckedProducts()}}
            </div>
          </li>
        </ul>
      </div>
    </div>
  </nav>
</small>
```

Pages: 3 Size: 3 Total: 9 Checked: 2

Ce code HTML crée une section de tableau de bord compact pour afficher des informations sur les produits, telles que le nombre total de pages, la taille des pages, le nombre total de produits et le nombre de produits vérifiés. Il utilise Bootstrap pour le style et Angular pour l'interaction dynamique avec les données.

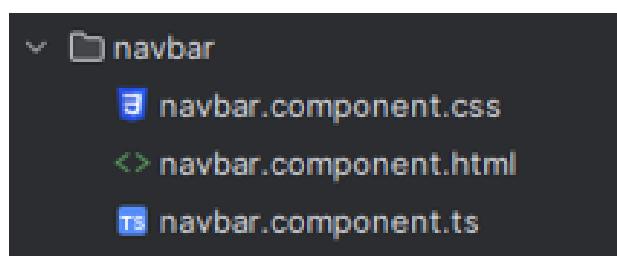
6.6 NAVBAR

• CRÉATION DE COMPOSANT

```
ada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> ng g c navbar
CREATE src/app/navbar/navbar.component.html (22 bytes)
CREATE src/app/navbar/navbar.component.spec.ts (619 bytes)
CREATE src/app/navbar/navbar.component.css (0 bytes)
```

Cette commande vous permet de générer un composant angulaire nommé "navbar" à l'aide de Angular CLI

la commande génère les fichiers suivants :



- **src/app/navbar/navbar.component.ts:**
Fichier TypeScript pour la logique du composant.
- **src/app/navbar/navbar.component.html**
Template HTML pour le composant.
- **src/app/navbar/navbar.component.css)**
Feuille de style pour le composant.
- **src/app/navbar/navbar.component.spec.ts**
Fichier de test unitaire pour le composant.
- **TYPESCRIPT (NAVBAR.COMPONENT.TS)**

```
@Component({
  selector: 'app-navbar',
  standalone: true,
  imports: [
    NgForOf,
    RouterLink,
    NgIf,
    AsyncPipe
  ],
  templateUrl: './navbar.component.html',
  styleUrls: ['./navbar.component.css']
})
```

```

export class NavbarComponent {
  actions : Array<any> = [
    {title : "Home", "route":"/admin/home", icon : "house"},
    {title : "Products", "route":"/admin/products", icon : "search"},
    {title : "New product", "route":"/admin/newProduct", icon : "safe"}
  ]
  currentAction : any;
  public isLoading :boolean=false;
  no usages
  constructor(public appState :AppStateService ,
              public loadingService : LoadingService) {
    /*this.loadingService.isLoading$.subscribe({
      next : (value)=>{
        this.isLoading=value;
      }
    })*/
  }
  1+ usages
  setCurrentAction(action: any) : void  {
    this.currentAction = action;
  }
}

```

Propriétés :

- **actions** : Tableau des actions disponibles dans la barre de navigation, avec leur titre, route et icône.
- **currentAction** : Action actuellement sélectionnée.
- **isLoading** : État de chargement, initialisé à false.

Constructeur :

- Injecte les services AppStateService et LoadingService.

Méthode

- **setCurrentAction** : Met à jour l'action actuellement sélectionnée.

Le NavbarComponent fournit une barre de navigation dynamique et réactive avec des liens vers différentes sections de l'application. Il affiche également un indicateur de chargement lorsque des opérations asynchrones sont en cours. Le composant utilise les bonnes pratiques d'Angular pour la gestion des états et des routes, et Bootstrap pour le style.

- **TEMPLATE HTML
(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>
    <li *ngIf="loadingService.isLoading$ | async">
      <div class="spinner-border text-primary ms-2" role="status">
        </div>
    </li>
  </ul>
</nav>
```



Ce code HTML fournit une barre de navigation dynamique et réactive qui permet de naviguer entre différentes sections de l'application. Il affiche également un indicateur de chargement lorsque des opérations asynchrones sont en cours. Les classes Bootstrap sont utilisées pour le style et la mise en page, et les directives Angular telles que *ngFor et *ngIf sont utilisées pour générer dynamiquement les éléments de la liste et afficher conditionnellement le spinner de chargement.

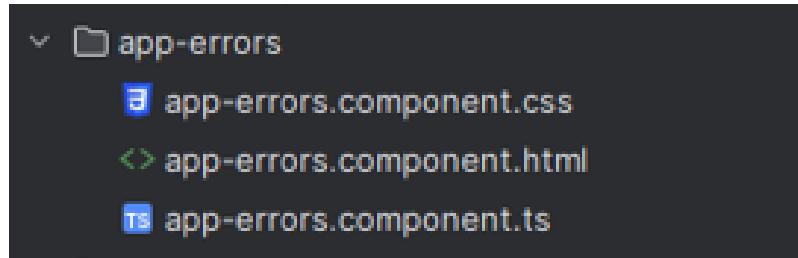
6.7 APP-ERRORS

- **CRÉATION DÉ COMPOSANT**

```
ada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> 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 (641 bytes)
CREATE src/app/app-errors/app-errors.component.ts (261 bytes)
CREATE src/app/app-errors/app-errors.component.css (0 bytes)
```

Cette commande vous permet de générer un composant angulaire nommé "app-errors" à l'aide de Angular CLI

la commande génère les fichiers suivants :



- **src/app/app-errors/app-errors.component.ts:**
Fichier TypeScript pour la logique du composant.
- **src/app/app-errors/app-errors.component.html**
Template HTML pour le composant.
- **src/app/app-errors/app-errors.component.css)**
Feuille de style pour le composant.
- **src/app/app-errors/app-errors.component.spec.ts**
Fichier de test unitaire pour le composant.
- **TYPESCRIPT (APP-ERRORS.COMPONENT.TS)**

```
@Component({
  selector: 'app-app-errors',
  standalone: true,
  imports: [
    JsonPipe,
    NgIf
  ],
  templateUrl: './app-errors.component.html',
  styleUrls: ['./app-errors.component.css'
})
export class AppErrorsComponent
{
  no usages
  constructor(public appState : AppStateService)
  {
  }
}
```

Constructeur :

- Injecte le service AppStateService pour accéder à l'état des produits.

Le AppErrorsComponent fournit une interface utilisateur simple pour afficher les messages d'erreur lorsque l'état des produits est en erreur. Il utilise les bonnes pratiques d'Angular pour la gestion des états et des conditions.

- TEMPLATE HTML
(APP-ERRORS.COMPONENT.HTML)

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

Http failure response for http://localhost:8089/products?
name_like=&_page=1&_limit=3: 0 undefined

Ce code HTML fournit une manière simple et efficace d'afficher les messages d'erreur lorsque l'état des produits est en erreur. Il utilise les bonnes pratiques d'Angular pour la gestion conditionnelle de l'affichage et Bootstrap pour le style des alertes. Lorsque l'état des produits passe à ERROR, le composant affiche le message d'erreur correspondant dans une alerte rouge, offrant ainsi une expérience utilisateur claire et intuitive.

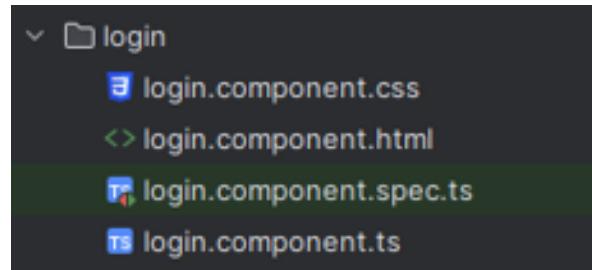
6.8 LOGIN

- CRÉATION DE COMPOSANT

```
\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> ng g c login
CREATE src/app/login/login.component.html (21 bytes)
CREATE src/app/login/login.component.spec.ts (612 bytes)
CREATE src/app/login/login.component.ts (242 bytes)
CREATE src/app/login/login.component.css (0 bytes)
```

Cette commande vous permet de générer un composant angulaire nommé "login" à l'aide de Angular CLI

la commande génère les fichiers suivants :



- **src/app/login/login.component.ts:**
Fichier TypeScript pour la logique du composant.
- **src/app/login/login.component.html**
Template HTML pour le composant.
- **src/app/login/login.component.css)**
Feuille de style pour le composant.
- **src/app/login/login.component.spec.ts**
Fichier de test unitaire pour le composant.
- **TYPESCRIPT (LOGIN.COMPONENT.TS)**

```
@Component({
  selector: 'app-login',
  standalone: true,
  imports: [
    ReactiveFormsModule
  ],
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css'
})
export class LoginComponent implements OnInit
{
  formLogin! : FormGroup;
  no usages
  constructor(private fb :FormBuilder,private router:Router)
  {
  }
  no usages
  ngOnInit() : void
  {
    this.formLogin=this.fb.group( controls: {
      username : this.fb.control( formState: ""),
      password : this.fb.control( formState: "")
    })
  }
}
```

```

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

```

Propriété

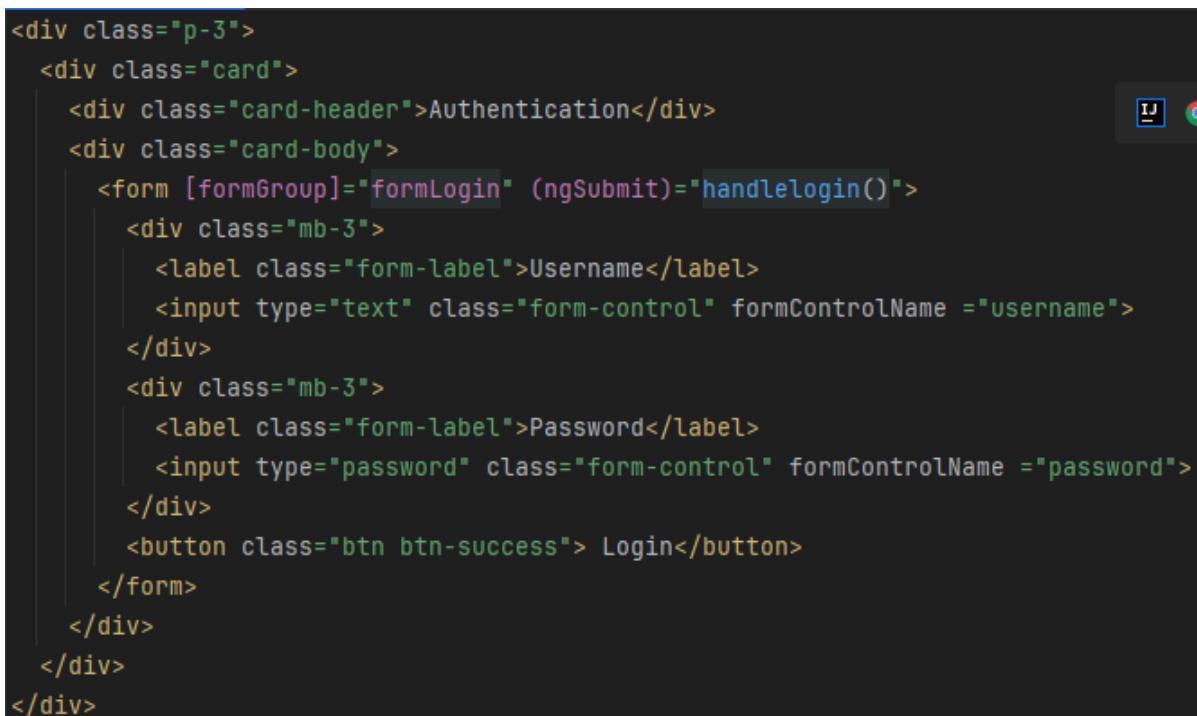
- **formLogin** : Formulaire réactif pour gérer les entrées utilisateur.

Constructeur

- Injecte FormBuilder pour créer le formulaire et Router pour la navigation.

Méthode

- **ngOnInit** : Initialise le formulaire avec des champs pour username et password.
- **handlelogin** : Gère la soumission du formulaire. Si les informations d'identification sont correctes, redirige l'utilisateur vers la page d'administration.
- **TEMPLATE HTML (LOGIN.COMPONENT.HTML)**



```

<div class="p-3">
  <div class="card">
    <div class="card-header">Authentication</div>
    <div class="card-body">
      <form [formGroup]="formLogin" (ngSubmit)="handlelogin()">
        <div class="mb-3">
          <label class="form-label">Username</label>
          <input type="text" class="form-control" formControlName ="username">
        </div>
        <div class="mb-3">
          <label class="form-label">Password</label>
          <input type="password" class="form-control" formControlName ="password">
        </div>
        <button class="btn btn-success"> Login</button>
      </form>
    </div>
  </div>
</div>

```

The screenshot shows a clean, modern login interface. At the top, the title 'Authentication' is displayed in a light gray font. Below it are two input fields: one for 'Username' and one for 'Password', both with placeholder text. A prominent green button labeled 'Login' is centered at the bottom of the form.

Ce code HTML fournit un formulaire de connexion élégant et fonctionnel utilisant Angular Reactive Forms pour la gestion des entrées utilisateur et Bootstrap pour le style. Le formulaire comprend des champs pour le nom d'utilisateur et le mot de passe, et un bouton pour soumettre les informations de connexion. Lorsque l'utilisateur soumet le formulaire, la méthode handlelogin() est appelée pour gérer les informations de connexion.

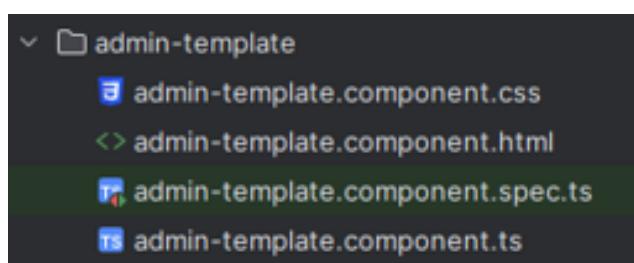
6.9 ADMIN-TEMPLATE

• CRÉATION DE COMPOSANT

```
noada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> 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 (669 bytes)
CREATE src/app/admin-template/admin-template.component.ts (277 bytes)
CREATE src/app/admin-template/admin-template.component.css (0 bytes)
```

Cette commande vous permet de générer un composant angulaire nommé "admin-template" à l'aide de Angular CLI

la commande génère les fichiers suivants :



- **src/app/admin-template/admin-template.component.ts:**
Fichier TypeScript pour la logique du composant.
- **src/app/admin-template/admin-template.component.html**
Template HTML pour le composant.
- **src/app/admin-template/admin-template.component.css)**
Feuille de style pour le composant.
- **src/app/admin-template/admin-template.component.spec.ts**
Fichier de test unitaire pour le composant.

• **TYPESCRIPT (ADMIN-TEMPLATE.COMPONENT.TS)**

```
@Component({
  selector: 'app-admin-template',
  standalone: true,
  imports: [
    AppErrorsComponent,
    DashboardComponent,
    NavbarComponent,
    RouterOutlet
  ],
  templateUrl: './admin-template.component.html',
  styleUrls: ['./admin-template.component.css']
})
export class AdminTemplateComponent
{}
```

• **TEMPLATE HTML (ADMIN-TEMPLATE.COMPONENT.HTML)**

```
<app-navbar></app-navbar>

<app-dashboard></app-dashboard>

<app-app-errors></app-app-errors>

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

Le composant AdminTemplateComponent sert de modèle principal pour l'interface d'administration de l'application. Il inclut d'autres composants pour la barre de navigation, le tableau de bord et les erreurs, ainsi qu'un espace pour le contenu dynamique géré par le routeur Angular.

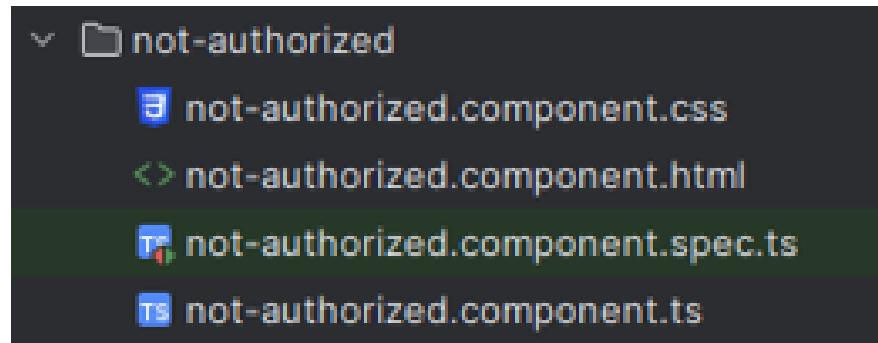
6.10 NOT-AUTHORIZED

- CRÉATION DE COMPOSANT

```
ada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> 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 (669 bytes)
CREATE src/app/not-authorized/not-authorized.component.ts (277 bytes)
CREATE src/app/not-authorized/not-authorized.component.css (0 bytes)
```

Cette commande vous permet de générer un composant angulaire nommé "not-authorized" à l'aide de Angular CLI

la commande génère les fichiers suivants :



- **src/app/not-authorized/not-authorized.component.ts:**
Fichier TypeScript pour la logique du composant.
 - **src/app/not-authorized/not-authorized.component.html**
Template HTML pour le composant.
 - **src/app/not-authorized/not-authorized.component.css)**
Feuille de style pour le composant.
 - **src/app/not-authorized/not-authorized.component.spec.ts**
Fichier de test unitaire pour le composant.
-
- **TYPESCRIPT
(NOT-AUTHORIZED.COMPONENT.TS)**

```
@Component({
  selector: 'app-not-authorized',
  standalone: true,
  imports: [],
  templateUrl: './not-authorized.component.html',
  styleUrls: ['./not-authorized.component.css']
})
export class NotAuthorizedComponent
```

- TEMPLATE HTML
(NOT-AUTHORIZED.COMPONENT.HTML)

```
<div class="p-3">
  <div class="card">
    <div class="card-body">
      <div class="alert alert-danger">
        You are not authorized
      </div>
    </div>
  </div>
</div>
```



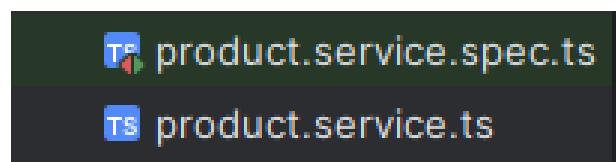
Le `NotAuthorizedComponent` fournit une interface utilisateur simple et efficace pour afficher un message d'alerte lorsque l'utilisateur n'est pas autorisé à accéder à une certaine partie de l'application. Il utilise Bootstrap pour le style et Angular pour la gestion du composant. Le composant est autonome et peut être facilement intégré dans d'autres parties de l'application pour gérer les erreurs d'autorisation.

7. SERVICES

7.1 PRODUCT

```
\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> ng g s services/product
CREATE src/app/services/product.service.spec.ts (378 bytes)
CREATE src/app/services/product.service.ts (145 bytes)
```

Cette commande va créer deux fichiers :



- **product.service.ts** : Contient la définition du service.
- **product.service.spec.ts** : Contient les tests unitaires pour le service.

PRODUCT.SERVICE.TS

```
@Injectable({
  providedIn: 'root'
})
export class ProductService
{
  private host : string="http://localhost:8089";
  no usages
  constructor(private http : HttpClient)
  { }
  1+ usages
  public searchProducts(keyword:string="",page :number=1, size:number=4) : Observable<HttpResponse<Object...>>
  {
    return this.http.get(`url: ${this.host}/products?name_like=${keyword}&_page=${page}&_limit=${size}` , options: {observe:'response'});
  }
  1+ usages
  public checkProducts(product : Product):Observable<Product>
  {
    return this.http.patch<Product>( url: `${this.host}/products/${product.id}` ,
      body: { checked: !product.checked });
  }
  1+ usages
  public deleteProducts(product : Product) : Observable<any>
  {
    return this.http.delete<any>( url: `${this.host}/products/${product.id}`);
  }
  1+ usages
  saveProduct(product: Product):Observable<Product>
  {
    return this.http.post<Product>( url: `${this.host}/products` ,product);
  }
  1+ usages
  getProductById(productId: number): Observable<Product>
  {
    return this.http.get<Product>( url: `${this.host}/products/${productId}`);
  }
  1+ usages
  updateProduct(product: Product): Observable<Product>
  {
    return this.http.put<Product>( url: `${this.host}/products/${product.id}` ,product);
  }
}
```

Propriété host :

- URL de base de l'API.

Constructeur :

- Injecte le service HttpClient pour effectuer des requêtes HTTP.

Méthodes :

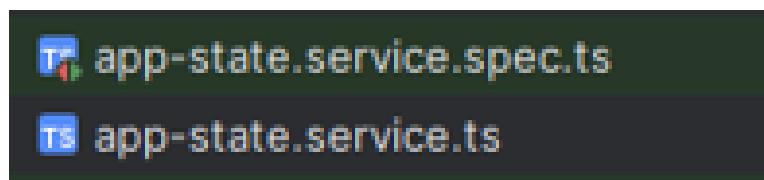
- searchProducts : Rechercher des produits en fonction d'un mot-clé, d'une page et d'une taille de page.
- checkProducts : Vérifier (ou dévérerifier) un produit en inversant son état checked.
- deleteProducts : Supprimer un produit en fonction de son ID.
- saveProduct : Sauvegarder un nouveau produit.
- getProductById : Obtenir un produit en fonction de son ID.
- updateProduct : Mettre à jour un produit existant.

Le ProductService est un service complet pour gérer les produits via une API RESTful. Il permet de rechercher, vérifier, supprimer, sauvegarder, obtenir et mettre à jour des produits. En utilisant HttpClient, il envoie des requêtes HTTP et utilise des observables pour gérer les réponses de manière asynchrone. Ce service est essentiel pour la communication entre le front-end Angular et l'API back-end, assurant une gestion efficace des produits.

7.2 APP-STATE

```
ada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> 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)
```

Cette commande va créer deux fichiers :



- app-state.service.ts : Contient la définition du service.
- app-state.service.spec.ts : Contient les tests unitaires pour le service.

APP-STATE.SERVICE.TS

```
@Injectable({
  providedIn: 'root'
})
export class AppStateService {
  public productsState :any={
    products:[],
    keyword : "",
    totalPages:0,
    pageSize:3,
    currentPage :1,
    totalProducts :0,
    status : "",
    errorMessage : ""
  }
  public authState :any ={
    isAuthenticated : false,
    username : undefined,
    roles : undefined,
    token : undefined
  }
  no usages
  constructor() { }
  1+ usages
  public setProductState(state :any):void {
    this.productsState={...this.productsState, ...state}
  }
  1+ usages
  public setAuthState(state : any) :void{
    this.authState={...this.authState, ...state};
  }
}
```

Propriété

productsState : Objet contenant l'état des produits avec les propriétés suivantes :

- **products** : Tableau des produits.
- **keyword** : Mot-clé de recherche.
- **totalPages** : Nombre total de pages de résultats.
- **pageSize** : Taille de chaque page de résultats.
- **currentPage** : Page actuelle.
- **totalProducts** : Nombre total de produits.
- **status** : Statut de l'état des produits (e.g., "LOADING", "LOADED", "ERROR").
- **errorMessage** : Message d'erreur si une erreur se produit.

Constructeur :

- Le constructeur est vide car aucune initialisation spécifique n'est nécessaire.

Méthode

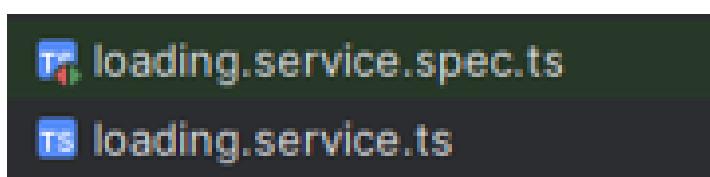
- `setProductState` : Permet de mettre à jour l'état des produits. Elle prend un objet `state` en argument et fusionne ses propriétés avec l'état actuel en utilisant la décomposition d'objet (`{ ...this.productsState, ...state }`).

Le `AppStateService` est un service essentiel pour gérer l'état global de l'application concernant les produits. Il centralise les informations relatives aux produits et fournit une méthode pour mettre à jour cet état. Cela permet à différents composants de l'application de lire et de modifier l'état des produits de manière synchronisée.

7.3 LOADING

```
ada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> ng g s services/loading
CREATE src/app/services/loading.service.spec.ts (378 bytes)
CREATE src/app/services/loading.service.ts (145 bytes)
```

Cette commande va créer deux fichiers :



- `loading.service.ts` : Contient la définition du service.
- `loading.service.spec.ts` : Contient les tests unitaires pour le service.

LOADING.SERVICE.SPEC.TS

```
@Injectable({
  providedIn: 'root'
})
export class LoadingService {
  public isLoading$ : Subject<boolean> = new Subject<boolean>();
no usages
constructor() { }
1+ usages
showLoadingSpinner():void
{
  this.isLoading$.next( value: true);
}
1+ usages
hideLoadingSpinner():void
{
  this.isLoading$.next( value: false);
}
}
```

Propriété

- `isLoading$` : Un `Subject` de type `boolean` utilisé pour diffuser des événements de chargement. Les composants peuvent s'abonner à `isLoading$` pour réagir aux changements de l'état de chargement.

Constructeur

- Le constructeur est vide car aucune initialisation spécifique n'est nécessaire.

Méthodes

- `showLoadingSpinner` : Appelée pour indiquer que le chargement commence. Elle émet `true` via `isLoading$`.
- `hideLoadingSpinner` : Appelée pour indiquer que le chargement est terminé. Elle émet `false` via `isLoading$`.

7.4 AUTH

```
ada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> ng g s auth
CREATE src/app/auth.service.spec.ts (363 bytes)
CREATE src/app/auth.service.ts (142 bytes)
```

Cette commande va créer deux fichiers :

 auth.service.spec.ts

 auth.service.ts

- **auth.service.ts** : Contient la définition du service.
- **auth.service.spec.ts** : Contient les tests unitaires pour le service.

AUTH.SERVICE.SPEC.TS

```
@Injectable({
  providedIn: 'root'
})
export class AuthService {
  no usages
  constructor(private http: HttpClient, private appState: AppStateService)
  {}

  1+ usages
  async login(username: string, password: string): Promise<boolean>
  {
    let user: any = await firstValueFrom(this.http.get(url: "http://localhost:8089/users/" + username));
    console.log(password);
    console.log(user.password);
    console.log(atob(user.password));
    if (password == atob(user.password))
    {
      let decodedJwt: any = jwtDecode(user.token);
      this.appState.setAuthState({
        isAuthenticated: true,
        username: decodedJwt.sub,
        roles: decodedJwt.roles,
        token: user.token
      });
      return Promise.resolve(value: true);
    }
    else
    {
      return Promise.reject(reason: "Bad credentials");
    }
  }
}
```

Propriété

- baseUrl : URL de base de l'API.

Constructeur

- Injecte les services HttpClient et AppStateService.

Méthode

- login : Gère l'authentification des utilisateurs.

Le AuthService est un service complet pour gérer l'authentification des utilisateurs dans une application Angular. Il effectue des requêtes HTTP pour récupérer les informations de l'utilisateur, vérifie le mot de passe, décode les tokens JWT, et met à jour l'état d'authentification de l'application. En utilisant HttpClient, firstValueFrom, et jwtDecode, le service offre une gestion efficace et sécurisée de l'authentification des utilisateurs.

8. INTERCEPTEURS

APP-HTTP

Cette commande crée deux fichiers

```
ada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> ng g interceptor app-http
CREATE src/app/app-http.interceptor.spec.ts (506 bytes)
CREATE src/app/app-http.interceptor.ts (157 bytes)
```

- app-http.interceptor.ts : Contient la définition de l'intercepteur.
- app-http.interceptor.spec.ts : Contient les tests unitaires pour l'intercepteur.

APP-HTTP.INTERCEPTOR.TS

```
@Injectable()
export class AppHttpInterceptor implements HttpInterceptor
{
  no usages
  constructor(private appState : AppStateService,
             private loadingService:LoadingService) {}
  no usages
  intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>>
  {
```

```

    /*this.appState.setProductState({
      status :"LOADING"
    })*/
    this.loadingService.showLoadingSpinner();
    let req : HttpRequest<?> = request.clone( update: {
      headers : request.headers.set("Authorization","Bearer JWT")
    });
    return next.handle(req).pipe(
      finalize( callback: () : void =>{
        /*this.appState.setProductState({
          status : ""
        })*/
        this.loadingService.hideLoadingSpinner();
      })
    );
  }
}

```

L'intercepteur `AppHttpInterceptor` permet d'ajouter des en-têtes, de gérer des tokens d'authentification, et de gérer l'état de chargement de l'application. Après avoir généré l'intercepteur avec Angular CLI, vous pouvez le configurer dans votre module principal pour qu'il soit utilisé globalement dans votre application. Cela centralise et simplifie la gestion des requêtes HTTP et de l'état de l'application.

9. GARDES

9.1 AUTHENTICATION

Cette commande crée deux fichiers

```

ada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> 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)

```

- `authentication.guard.ts` : Contient la définition du garde de route.
- `authentication.guard.spec.ts` : Contient les tests unitaires pour le garde de route.

AUTHENTICATION.GUARD.TS

```
@Injectable({
  providedIn: 'root'
})
export class AuthenticationGuard {
  no usages
  constructor(private appState : AppStateService, private router : Router)
  {}
  no usages
  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree
  {
    if (this.appState.authState.isAuthenticated)
    {
      return true;
    }
    else
    {
      this.router.navigateByUrl({ url: "/login"});
      return false;
    }
  }
}
```

Le garde de route AuthenticationGuard vérifie si un utilisateur est authentifié avant de lui permettre d'accéder à certaines routes. En utilisant AppStateService pour vérifier l'état d'authentification et Router pour la redirection, ce garde assure que seules les personnes authentifiées peuvent accéder aux parties protégées de votre application. Après avoir généré le garde avec Angular CLI.

9.2 AUTHORIZATION

Cette commande crée deux fichiers

```
ada_Nohayla_Tp4\Anoada_Nohayla_Tp4_1> 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)
```

- **authorization.guard.ts** : Contient la définition du garde de route.
- **authorization.guard.spec.ts** : Contient les tests unitaires pour le garde de route.

AUTHORIZATION.GUARD.TS

```
@Injectable({
  providedIn: 'root'
})
export class AuthorizationGuard {
  no usages
  constructor(private appState : AppStateService, private router : Router ) {
  }
  no usages
  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree
  {
    if(this.appState.authState.roles.includes(route.data['requiredRoles']))
    {
      return true;
    }
    else
    {
      this.router.navigateByUrl(url: "/admin/notAuthorized")
      return false;
    }
  }
}
```

Le garde de route AuthorizationGuard vérifie si un utilisateur possède les rôles requis avant de lui permettre d'accéder à certaines routes protégées de l'application. En utilisant AppStateService pour vérifier les rôles d'autorisation et Router pour la redirection, ce garde assure que seules les personnes autorisées peuvent accéder aux parties protégées de votre application. Après avoir généré le garde avec Angular CL.

PARTIE 2 :

A: DÉVELOPPER ET TESTER LA PARTIE BACKEND AVEC SPRING.

1. CRÉER LES ENTITÉS JPA

1.1. PAYMENTSTATUS

L'énumération PaymentStatus représente les différents états qu'un paiement.

```
public enum PaymentStatus
{
    2 usages
    CREATED, VALIDATED, REJECTED
}
```

1.2. PAYMENTTYPE

L'énumération `PaymentType` représente les différentes méthodes de paiement qu'un étudiant peut utiliser.

```
public enum PaymentType
{
    no usages
    CASH, CHECK, TRANSFER, DEPOSIT
}
```

1.3. PAYMENT

L'entité `Payment` représente une transaction de paiement effectuée par un étudiant. Chaque enregistrement de paiement contient des détails tels que la date, le montant, le type, le statut et un fichier associé.

```
@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;
}
```

1.4. STUDENT

L'entité Student représente un étudiant dans le système. Chaque étudiant a un identifiant unique et d'autres attributs comme son nom, son programme et sa photo.

```
@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;  
}
```

2. CRÉER LES INTERFACES JPAREPOSITORY BASÉES SUR SPRING DATA

2.1. PAYMENTREPOSITORY

L'interface PaymentRepository fournit des méthodes pour effectuer des opérations de recherche sur l'entité Payment. Elle étend JpaRepository avec l'entité Payment et le type de clé primaire Long.

```
public interface PaymentRepository extends JpaRepository<Payment, Long>  
{  
    1 usage  ↗ nohayla  
    List<Payment> findByStudentCode(String code);  
    1 usage  ↗ nohayla  
    List<Payment> findByStatus(PaymentStatus status);  
    1 usage  ↗ nohayla  
    List<Payment> findByType(PaymentType type);  
    1 usage  ↗ nohayla  
    List<Payment> findByStudentProgramId(String programId);  
}
```

2.2. STUDENTREPOSITORY

L'interface `StudentRepository` fournit des méthodes pour effectuer des opérations de recherche sur l'entité `Student`. Elle étend `JpaRepository` avec l'entité `Student` et le type de clé primaire `String`.

```
public interface StudentRepository extends JpaRepository<Student, String>
{
    2 usages  ↳ nohayla
        Student findByCode(String code);
    1 usage  ↳ nohayla
        List<Student> findByProgramId(String programId);
}
```

3 . GÉNÉRER DES DONNÉES ALÉATOIRES

La classe `AnoadaNohaylaTp42Application` est la classe principale de l'application Spring Boot. Elle contient la méthode `main` qui lance l'application et une méthode `CommandLineRunner` qui initialise les données dans la base de données.

```
@SpringBootApplication
public class AnoadaNohaylaTp42Application
{

    ↳ nohayla
    public static void main(String[] args) { SpringApplication.run(AnoadaNohaylaTp42Application.class, args); }

    @Bean
    CommandLineRunner commandLineRunner(StudentRepository studentRepository, PaymentRepository paymentRepository)
    {
        return args -> {
            studentRepository.save(Student.builder().id(UUID.randomUUID().toString())
                .code("112233").firstName("Mohamed").programId("SDIA").build());
            studentRepository.save(Student.builder().id(UUID.randomUUID().toString())
                .code("112244").firstName("Imane").programId("GLSID").build());
            studentRepository.save(Student.builder().id(UUID.randomUUID().toString())
                .code("112255").firstName("Alae").programId("IAAO").build());
            studentRepository.save(Student.builder().id(UUID.randomUUID().toString())
                .code("112266").firstName("Najat").programId("GLSID").build());

            PaymentType[] paymentTypes = PaymentType.values();
            Random random=new Random();
            studentRepository.findAll().forEach(st->{
                for (int i = 0; i <10 ; i++)
                {
                    int index = random.nextInt(paymentTypes.length);
                    Payment payment = Payment.builder()
                        .amount(1000+(int)(Math.random()*20000))
                        .type(paymentTypes[index])
                        .status(PaymentStatus.CREATED)
                        .date(LocalDate.now())
                        .student(st)
                        .build();
                    paymentRepository.save(payment);
                }
            });
        };
    }
}
```

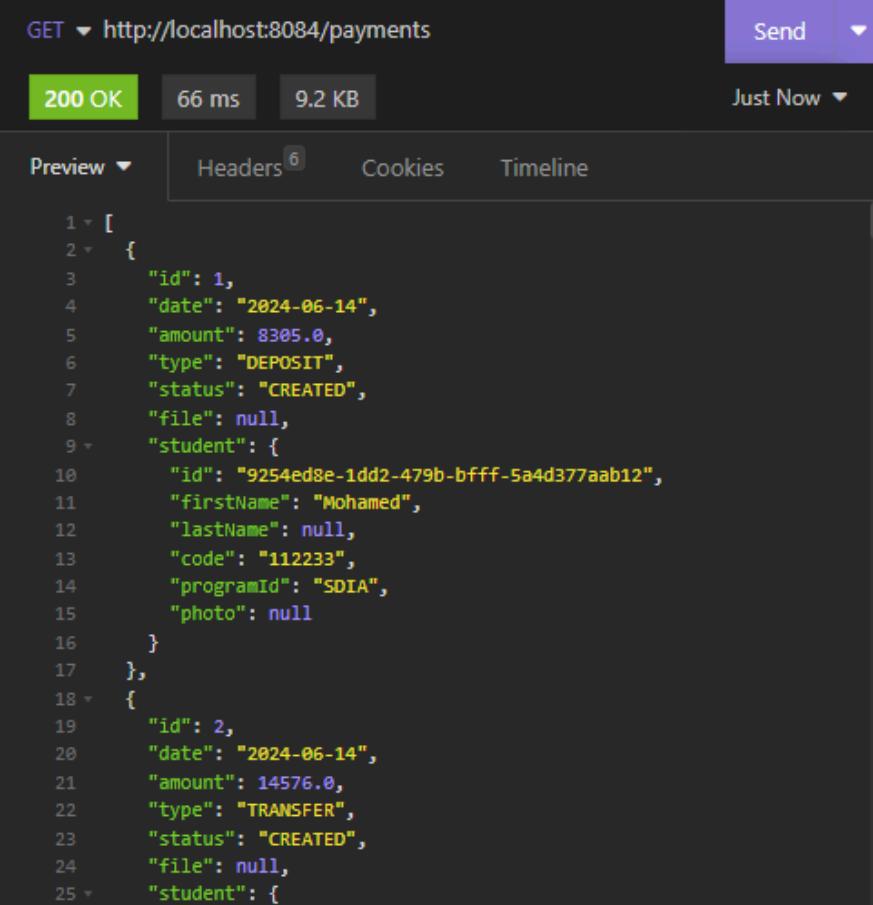
4. CRÉER UNE WEB SERVICE RESTFUL ET TESTER DE BACKEND EN UTILISANT UN CLIENT REST ET AVEC SWAGGER UI

4.1. CONSULTER TOUS LES PAYEMENTS

• CODE

```
@GetMapping(path="/payments")
public List<Payment> allPayments()
{
    return paymentRepository.findAll();
}
```

• TESTE AVEC UN CLIENT REST



GET ▾ http://localhost:8084/payments Send ▾

200 OK 66 ms 9.2 KB Just Now ▾

Preview ▾ Headers 6 Cookies Timeline

```
1 [ 
2 {
3     "id": 1,
4     "date": "2024-06-14",
5     "amount": 8305.0,
6     "type": "DEPOSIT",
7     "status": "CREATED",
8     "file": null,
9     "student": {
10         "id": "9254ed8e-1dd2-479b-bfff-5a4d377aab12",
11         "firstName": "Mohamed",
12         "lastName": null,
13         "code": "112233",
14         "programId": "SDIA",
15         "photo": null
16     }
17 },
18 {
19     "id": 2,
20     "date": "2024-06-14",
21     "amount": 14576.0,
22     "type": "TRANSFER",
23     "status": "CREATED",
24     "file": null,
25     "student": {
```

• TESTE AVEC SWAGGER UI

The screenshot shows the Swagger UI interface for a `GET /payments` endpoint. The top navigation bar shows the method `GET` and the path `/payments`. Below the path, there's a section for **Parameters** which says "No parameters". There are two buttons at the bottom: **Execute** (highlighted in blue) and **Clear**. The **Responses** section contains a **Curl** command:

```
curl -X 'GET' \
  'http://localhost:8084/payments' \
  -H 'accept: */*'
```

Below the curl command is the **Request URL**: `http://localhost:8084/payments`. The **Server response** section shows a **Code** dropdown set to `200` and a **Details** link. The response body is displayed as a JSON array:

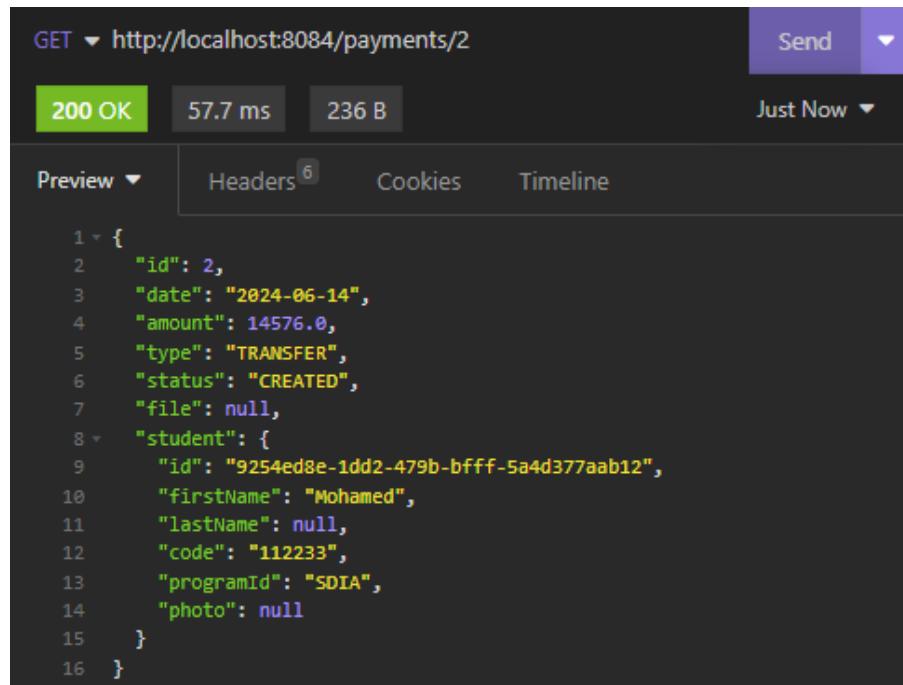
```
[  
  {  
    "id": 1,  
    "date": "2024-06-14",  
    "amount": 8385,  
    "type": "DEPOSIT",  
    "status": "CREATED",  
    "file": null,  
    "student": {  
      "id": "9254ed8e-1dd2-479b-bfff-5a4d377aab12",  
      "firstName": "Mohamed",  
      "lastName": null,  
      "code": "112233",  
      "programId": "SDIA",  
      "photo": null  
    },  
  }  
]
```

4.2. CONSULTER UN PAYEMENT SACHANT SON ID

• CODE

```
@GetMapping(path="/payments/{id}")  
public Payment getPaymentById(@PathVariable Long id)  
{  
    return paymentRepository.findById(id).get();  
}
```

• TESTE AVEC UN CLIENT REST



GET ▾ http://localhost:8084/payments/2

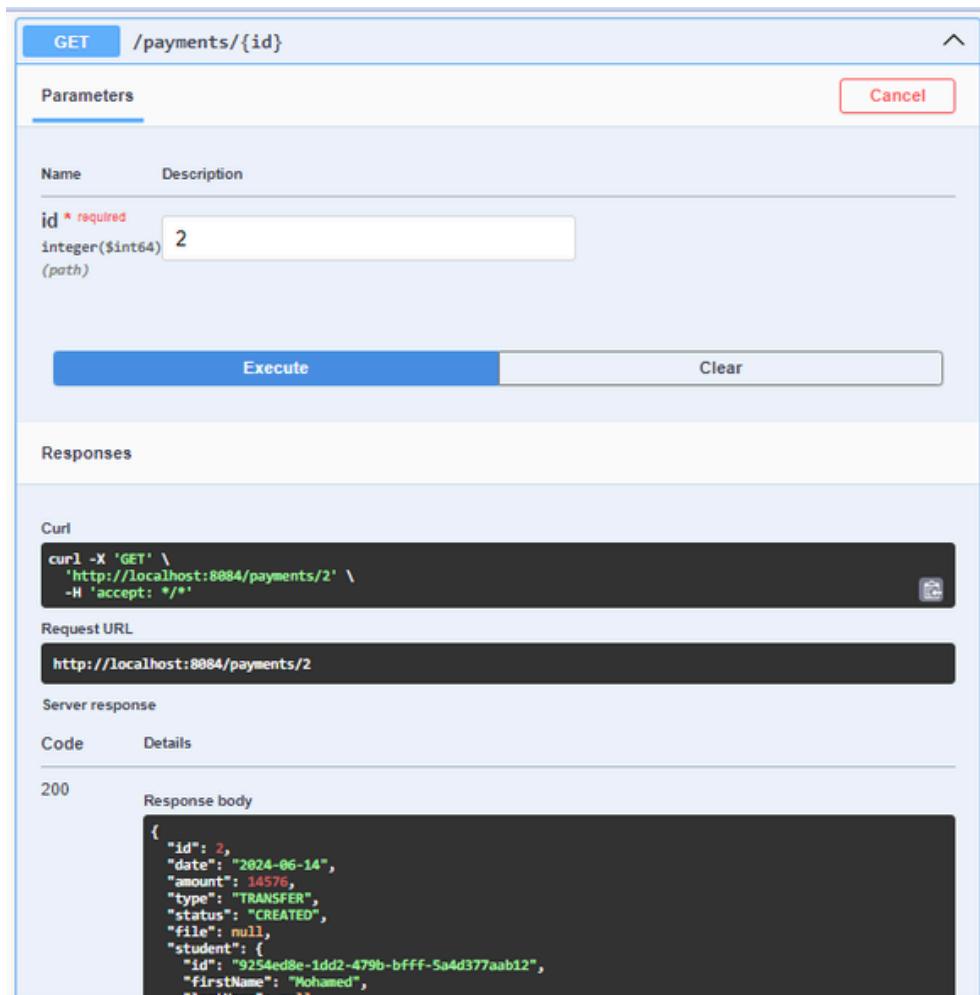
Send ▾

200 OK 57.7 ms 236 B Just Now ▾

Preview ▾ Headers 6 Cookies Timeline

```
1 {  
2   "id": 2,  
3   "date": "2024-06-14",  
4   "amount": 14576.0,  
5   "type": "TRANSFER",  
6   "status": "CREATED",  
7   "file": null,  
8   "student": {  
9     "id": "9254ed8e-1dd2-479b-bfff-5a4d377aab12",  
10    "firstName": "Mohamed",  
11    "lastName": null,  
12    "code": "112233",  
13    "programId": "SDIA",  
14    "photo": null  
15  }  
16 }
```

• TESTE AVEC SWAGGER UI



GET /payments/{id}

Parameters

Name	Description
id * required	integer(\$int64) (path)

Cancel

Execute Clear

Responses

Curl

```
curl -X 'GET' \  
  'http://localhost:8084/payments/2' \  
  -H 'accept: */*'
```

Request URL

```
http://localhost:8084/payments/2
```

Server response

Code Details

200 Response body

```
{  
  "id": 2,  
  "date": "2024-06-14",  
  "amount": 14576,  
  "type": "TRANSFER",  
  "status": "CREATED",  
  "file": null,  
  "student": {  
    "id": "9254ed8e-1dd2-479b-bfff-5a4d377aab12",  
    "firstName": "Mohamed",  
    "lastName": null,  
    "code": "112233",  
    "programId": "SDIA",  
    "photo": null  
  }  
}
```

4.3. CONSULTER LES PAYEMENTS D'UN TYPE DONNÉ

• CODE

```
@GetMapping("/payments/byType")
public List<Payment> paymentsByType(@RequestParam PaymentType type)
{
    return paymentRepository.findByType(type);
}
```

• TESTE AVEC UN CLIENT REST

The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: http://localhost:8084/payments/byType?type=DEPOSIT
- Status: 200 OK
- Time: 49.3 ms
- Size: 2.1 KB
- Last Updated: Just Now
- Preview tab is selected, showing a JSON response with two payment objects:

```
1 [ 
2   {
3     "id": 2,
4     "date": "2024-06-15",
5     "amount": 13805.0,
6     "type": "DEPOSIT",
7     "status": "CREATED",
8     "file": null,
9     "student": {
10       "id": "538d7900-204e-4cba-8809-3c546fd4ba2a",
11       "firstName": "Mohamed",
12       "lastName": null,
13       "code": "112233",
14       "programId": "SDIA",
15       "photo": null
16     }
17   },
18   {
19     "id": 5,
20     "date": "2024-06-15",
21     "amount": 18638.0,
22     "type": "DEPOSIT",
23     "status": "CREATED",
24     "file": null,
25     "student": {
26       "id": "538d7900-204e-4cba-8809-3c546fd4ba2a",
27       "firstName": "Mohamed",
28       "lastName": null,
29       "code": "112233"
29 }
```

• TESTE AVEC SWAGGER UI

The screenshot shows the Swagger UI interface for a `GET /payments/byType` endpoint. In the 'Parameters' section, there is a single parameter named `type` with a dropdown value set to `DEPOSIT`. Below the parameters are two buttons: `Execute` and `Clear`. The 'Responses' section contains a 'Curl' block with a command to run a `curl` request, a 'Request URL' block with the URL `http://localhost:8084/payments/byType?type=DEPOSIT`, and a 'Server response' section showing a `200` status code. The response body is a JSON array containing one element:

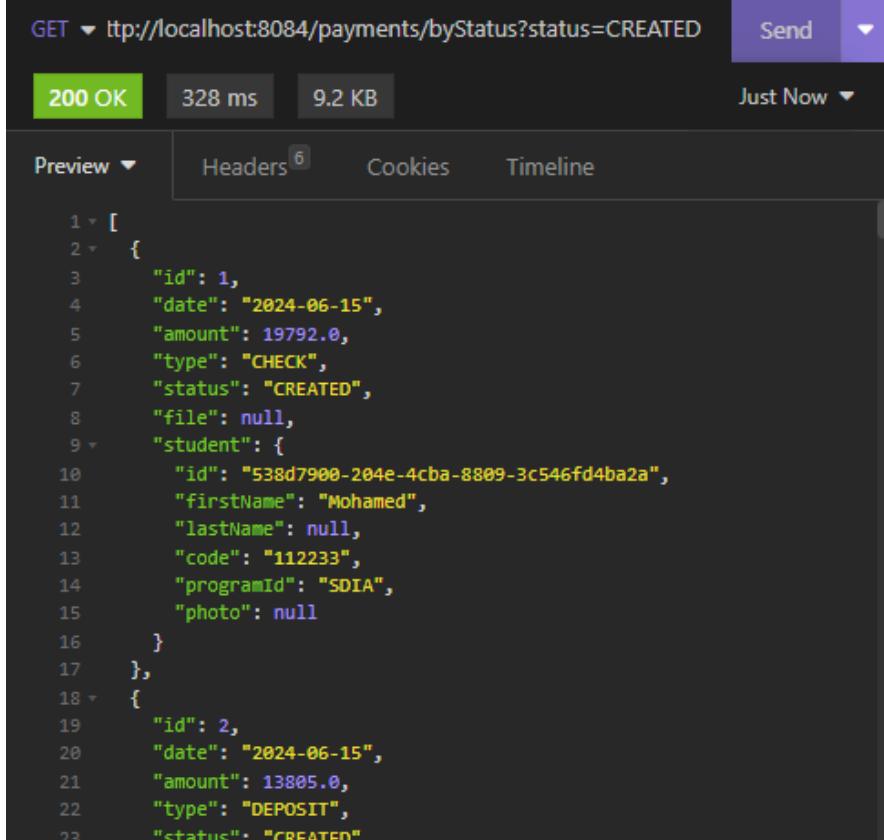
```
[{"id": 2, "date": "2024-06-15", "amount": 13805, "type": "DEPOSIT", "status": "CREATED", "file": null, "student": {"id": "538d7900-204e-4cba-8889-3c546fd4ba2a", "firstName": "Mohamed"}}
```

4.4. CONSULTER LES PAYEMENTS D'UN STATUS DONNÉ

• CODE

```
@GetMapping("/payments/byStatus")
public List<Payment> paymentsByStatus(@RequestParam PaymentStatus status)
{
    return paymentRepository.findByStatus(status);
}
```

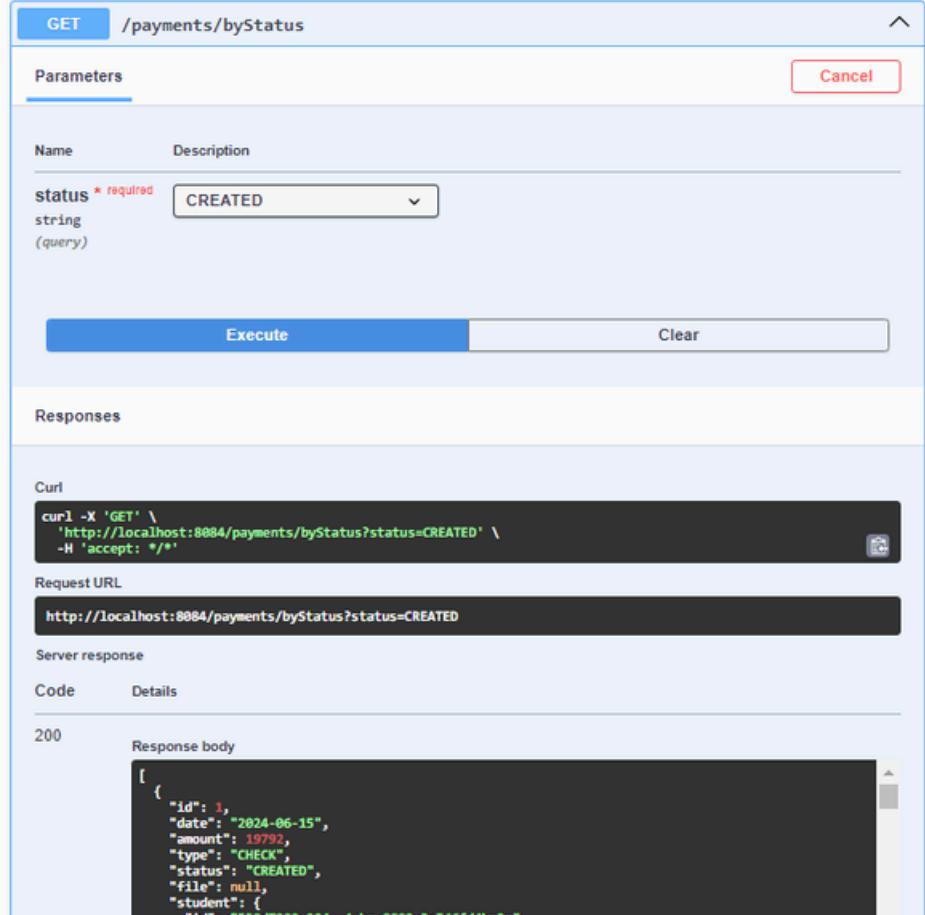
• TESTE AVEC UN CLIENT REST



A screenshot of a REST client interface. At the top, a header bar shows "GET ▾ http://localhost:8084/payments/byStatus?status=CREATED" and a "Send" button. Below the header, a status bar indicates "200 OK", "328 ms", and "9.2 KB". To the right, a timestamp says "Just Now". Underneath, there are tabs for "Preview", "Headers 6", "Cookies", and "Timeline". The "Preview" tab displays a JSON response with two payment objects. The first payment has an id of 1, a date of "2024-06-15", an amount of 19792.0, a type of "CHECK", a status of "CREATED", and a student object with an id of "538d7900-204e-4cba-8809-3c546fd4ba2a", a first name of "Mohamed", and a code of "112233". The second payment has an id of 2, a date of "2024-06-15", an amount of 13805.0, a type of "DEPOSIT", and a status of "CREATED".

```
[{"id": 1, "date": "2024-06-15", "amount": 19792.0, "type": "CHECK", "status": "CREATED", "file": null, "student": {"id": "538d7900-204e-4cba-8809-3c546fd4ba2a", "firstName": "Mohamed", "lastName": null, "code": "112233", "programId": "SDIA", "photo": null}}, {"id": 2, "date": "2024-06-15", "amount": 13805.0, "type": "DEPOSIT", "status": "CREATED"}]
```

• TESTE AVEC SWAGGER UI



A screenshot of the Swagger UI interface for the "/payments/byStatus" endpoint. At the top, it shows a "GET" method and the URL "/payments/byStatus". Below this, there's a "Parameters" section with a "status" parameter set to "CREATED". There are "Execute" and "Clear" buttons. Under the "Responses" section, there's a "Curl" section with a command: "curl -X 'GET' \ 'http://localhost:8084/payments/byStatus?status=CREATED' \ -H 'accept: */*'". Below that is a "Request URL" input field containing "http://localhost:8084/payments/byStatus?status=CREATED". In the "Server response" section, there's a "Code" dropdown set to "200" and a "Details" tab. The "Response body" pane shows the same JSON response as the REST client, listing two payments with their respective details.

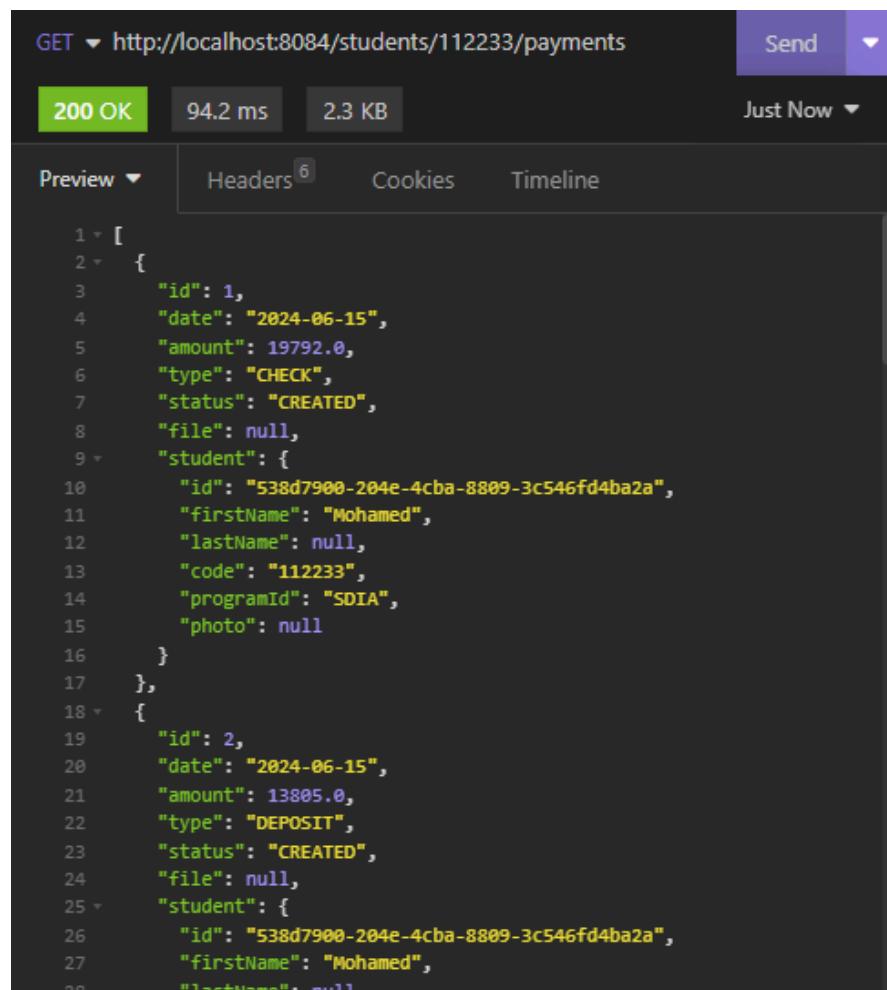
```
[{"id": 1, "date": "2024-06-15", "amount": 19792.0, "type": "CHECK", "status": "CREATED", "file": null, "student": {"id": "538d7900-204e-4cba-8809-3c546fd4ba2a", "firstName": "Mohamed", "lastName": null, "code": "112233", "programId": "SDIA", "photo": null}}, {"id": 2, "date": "2024-06-15", "amount": 13805.0, "type": "DEPOSIT", "status": "CREATED"}]
```

4.5. CONSULTER LES PAYEMENTS D'UN ÉTUDIANT DONNÉ

• CODE

```
@GetMapping("/students/{code}/payments")
public List<Payment> paymentsByStudent(@PathVariable String code)
{
    return paymentRepository.findByStudentCode(code);
}
```

• TESTE AVEC UN CLIENT REST



GET ▾ http://localhost:8084/students/112233/payments

Send ▾

200 OK 94.2 ms 2.3 KB Just Now ▾

Preview ▾ Headers 6 Cookies Timeline

```
1 [ 
2 {
3     "id": 1,
4     "date": "2024-06-15",
5     "amount": 19792.0,
6     "type": "CHECK",
7     "status": "CREATED",
8     "file": null,
9     "student": {
10        "id": "538d7900-204e-4cba-8809-3c546fd4ba2a",
11        "firstName": "Mohamed",
12        "lastName": null,
13        "code": "112233",
14        "programId": "SDIA",
15        "photo": null
16    }
17 },
18 {
19     "id": 2,
20     "date": "2024-06-15",
21     "amount": 13805.0,
22     "type": "DEPOSIT",
23     "status": "CREATED",
24     "file": null,
25     "student": {
26        "id": "538d7900-204e-4cba-8809-3c546fd4ba2a",
27        "firstName": "Mohamed",
28        "lastName": null
29    }
30 }
```

• TESTE AVEC SWAGGER UI

The screenshot shows the Swagger UI interface for testing a REST API. At the top, it displays a GET request to the endpoint `/students/{code}/payments`. In the 'Parameters' section, there is a single required parameter named `code` with the value `112233`. Below the parameters are two buttons: 'Execute' (in blue) and 'Clear'. The 'Responses' section contains a 'Curl' command and a 'Request URL'. The 'Server response' section shows a successful 200 status code. The response body is a JSON array containing one payment object:

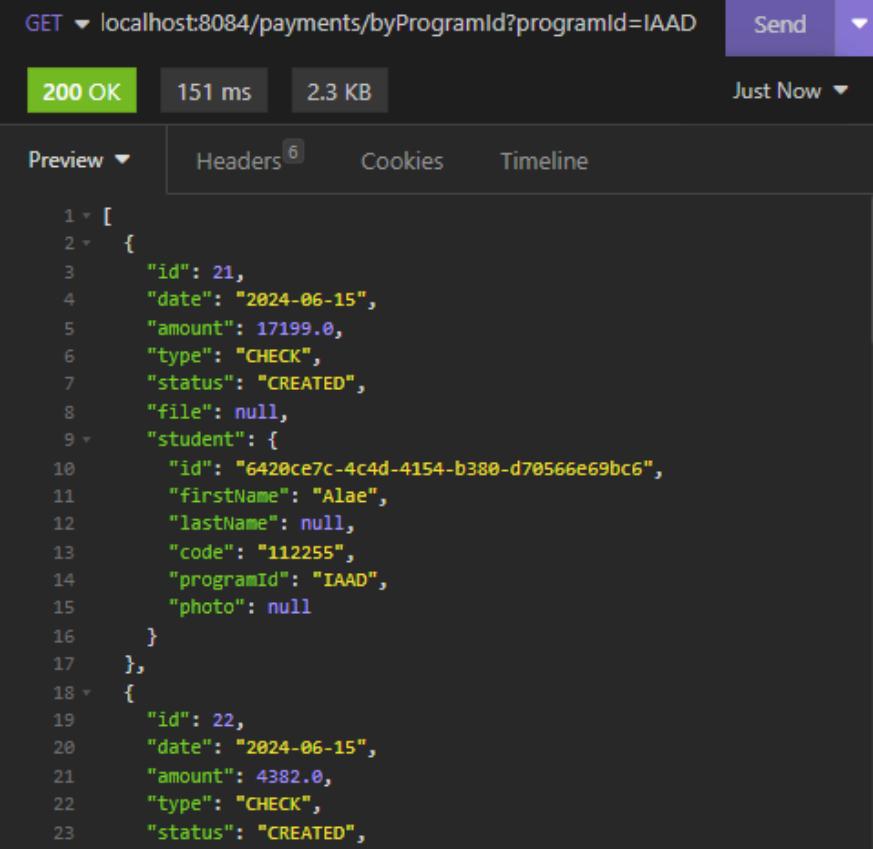
```
[{"id": 1, "date": "2024-06-15", "amount": 19792, "type": "CHECK", "status": "CREATED", "file": null, "student": {"id": "538d7900-204e-4cba-8809-3c546fd4ba2a", "name": "John Doe", "surname": "Doe", "age": 22, "program": "Bachelor's Degree in Computer Science", "status": "Active", "lastPayment": {"id": 1, "date": "2024-06-15", "amount": 19792, "type": "CHECK", "status": "CREATED", "file": null}}]}
```

4.6. CONSULTER LES PAYEMENTS D'UNE FILIÈRE DONNÉE

• CODE

```
@GetMapping("/payments/byProgramId")
public List<Payment> getPaymentsByProgramId(@RequestParam String programId)
{
    return paymentRepository.findByStudentProgramId(programId);
}
```

• TESTE AVEC UN CLIENT REST



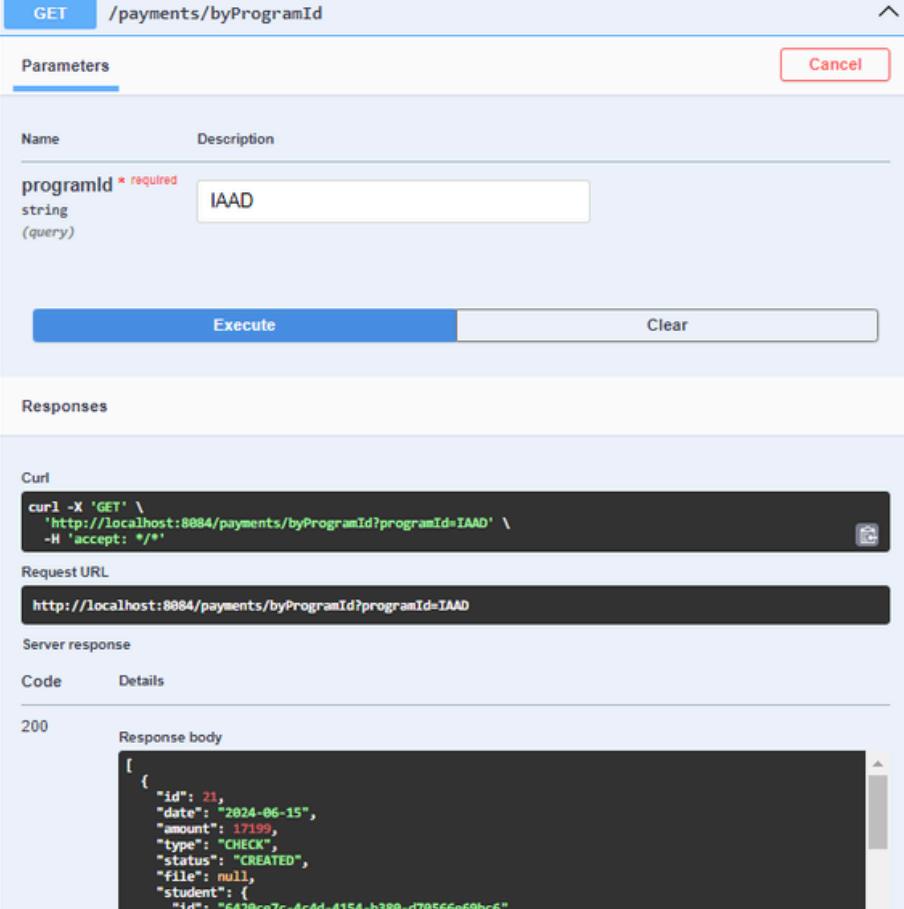
GET ▾ localhost:8084/payments/byProgramId?programId=IAAD Send ▾

200 OK 151 ms 2.3 KB Just Now ▾

Preview ▾ Headers 6 Cookies Timeline

```
1 [  
2 {  
3   "id": 21,  
4   "date": "2024-06-15",  
5   "amount": 17199.0,  
6   "type": "CHECK",  
7   "status": "CREATED",  
8   "file": null,  
9   "student": {  
10     "id": "6420ce7c-4c4d-4154-b380-d70566e69bc6",  
11     "firstName": "Alae",  
12     "lastName": null,  
13     "code": "112255",  
14     "programId": "IAAD",  
15     "photo": null  
16   },  
17 },  
18 {  
19   "id": 22,  
20   "date": "2024-06-15",  
21   "amount": 4382.0,  
22   "type": "CHECK",  
23   "status": "CREATED",  
24 }
```

• TESTE AVEC SWAGGER UI



GET /payments/byProgramId

Parameters

Name	Description
programId * required	IAAD string (query)

Responses

Curl

```
curl -X 'GET' \  
  'http://localhost:8084/payments/byProgramId?programId=IAAD' \  
  -H 'accept: */*'
```

Request URL

```
http://localhost:8084/payments/byProgramId?programId=IAAD
```

Server response

Code Details

200 Response body

```
[  
 {  
   "id": 21,  
   "date": "2024-06-15",  
   "amount": 17199.0,  
   "type": "CHECK",  
   "status": "CREATED",  
   "file": null,  
   "student": {  
     "id": "6420ce7c-4c4d-4154-b380-d70566e69bc6",  
     "firstName": "Alae",  
     "lastName": null,  
     "code": "112255",  
     "programId": "IAAD",  
     "photo": null  
   }  
 },  
 {  
   "id": 22,  
   "date": "2024-06-15",  
   "amount": 4382.0,  
   "type": "CHECK",  
   "status": "CREATED",  
   "file": null,  
   "student": {  
     "id": "6420ce7c-4c4d-4154-b380-d70566e69bc6",  
     "firstName": "Alae",  
     "lastName": null,  
     "code": "112255",  
     "programId": "IAAD",  
     "photo": null  
   }  
 }
```

4.7. CONSULTER TOUS LES ÉTUDIANTS

- CODE

```
@GetMapping(path = "/students")
public List<Student> allStudents()
{
    return studentRepository.findAll();
}
```

- TESTÉ AVEC UN CLIENT REST

The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: http://localhost:8084/students
- Status: 200 OK
- Time: 77 ms
- Size: 524 B
- Last Updated: Just Now

The response body is displayed in a code editor-like format with line numbers:

```
1 [
2 {
3     "id": "538d7900-204e-4cba-8809-3c546fd4ba2a",
4     "firstName": "Mohamed",
5     "lastName": null,
6     "code": "112233",
7     "programId": "SDIA",
8     "photo": null
9 },
10 {
11     "id": "131aecaf-746f-4350-b020-506bf6bb4b86",
12     "firstName": "Imane",
13     "lastName": null,
14     "code": "112244",
15     "programId": "GLSID",
16     "photo": null
17 },
18 {
19     "id": "6420ce7c-4c4d-4154-b380-d70566e69bc6",
20     "firstName": "Alae",
21     "lastName": null,
22     "code": "112255",
23     "programId": "IAAD",
24     "photo": null
25 },
26 {
```

• TESTE AVEC SWAGGER UI

The screenshot shows the Swagger UI for a `GET /students` endpoint. The **Parameters** section indicates "No parameters". Below it are **Execute** and **Clear** buttons. The **Responses** section includes a **Curl** command and a **Request URL** input field set to `http://localhost:8084/students`. Under **Server response**, a **Code** dropdown is set to `200`, and the **Details** tab is selected, showing a JSON response body:

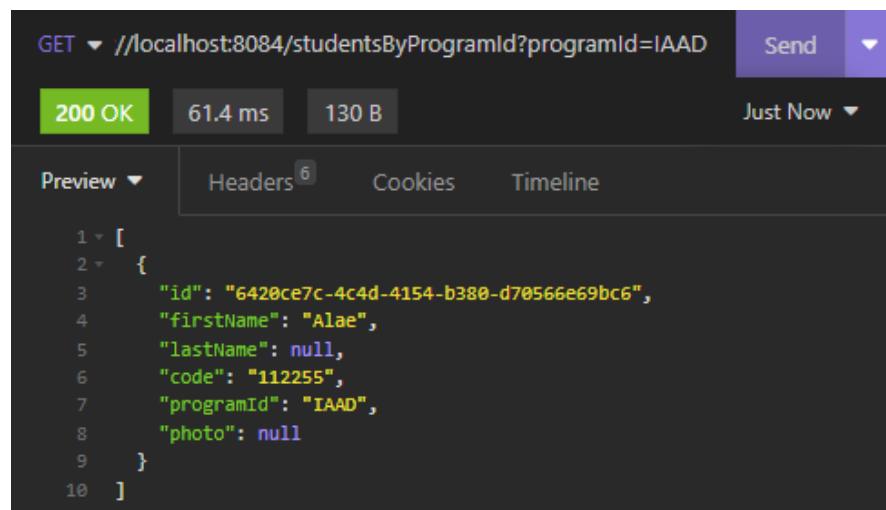
```
[  
  {  
    "id": "538d7900-284e-4cba-8809-3c546fd4ba2a",  
    "firstName": "Mohamed",  
    "lastName": null,  
    "code": "112233",  
    "programId": "SDIA",  
    "photo": null  
  },  
  {  
    "id": "131aeca1-746f-4350-b020-506bf6bb4b86",  
    "firstName": "Imane",  
    "lastName": null,  
    "code": "112244",  
    "programId": "GLSID",  
    "photo": null  
  }]
```

4.8. CONSULTER LES ÉTUDIANTS D'UNE FILIÈRE DONNÉE

• CODE

```
@GetMapping(path = "/studentsByProgramId")  
public List<Student> getStudentsByProgramId(@RequestParam String programId)  
{  
    return studentRepository.findByProgramId(programId);  
}
```

• TESTE AVEC UN CLIENT REST



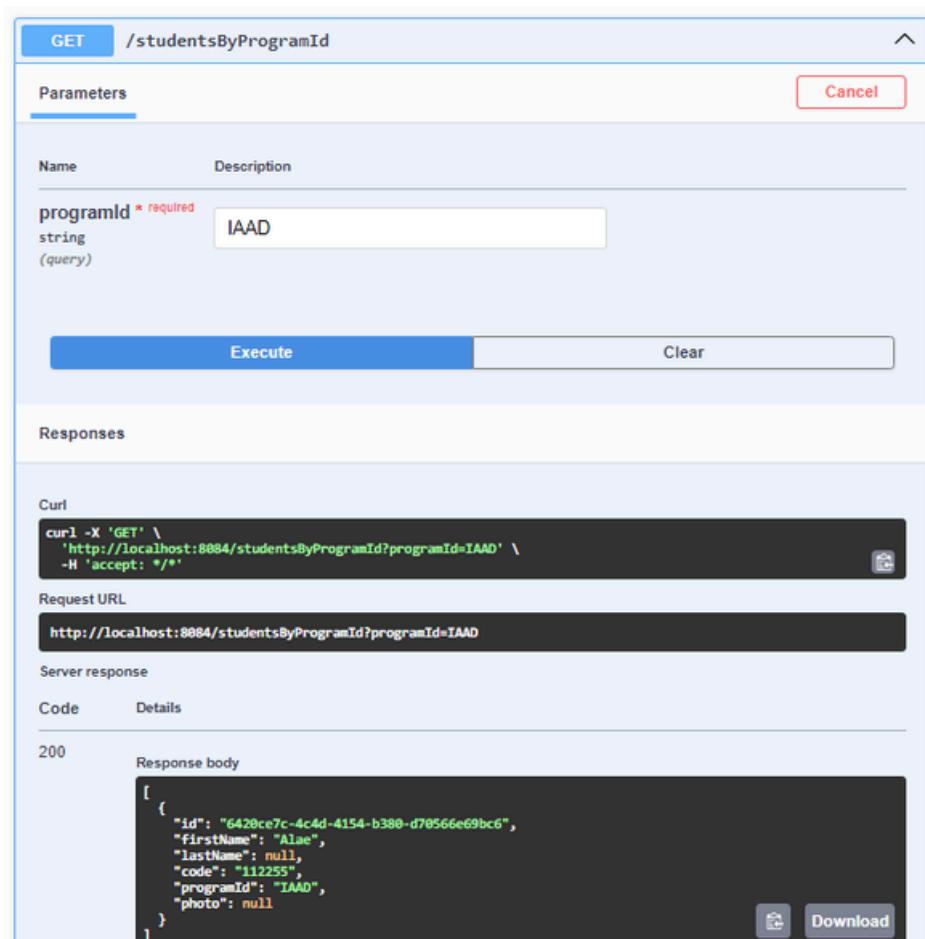
GET ▾ //localhost:8084/studentsByProgramId?programId=IAAD Send ▾

200 OK 61.4 ms 130 B Just Now ▾

Preview ▾ Headers 6 Cookies Timeline

```
1 > [  
2 >   {  
3 >     "id": "6420ce7c-4c4d-4154-b380-d70566e69bc6",  
4 >     "firstName": "Alae",  
5 >     "lastName": null,  
6 >     "code": "112255",  
7 >     "programId": "IAAD",  
8 >     "photo": null  
9 >   }  
10 > ]
```

• TESTE AVEC SWAGGER UI



GET /studentsByProgramId

Parameters

Name Description

programId * required
string
(query) IAAD

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8084/studentsByProgramId?programId=IAAD' \
-H 'accept: */*' 
```

Request URL

```
http://localhost:8084/studentsByProgramId?programId=IAAD
```

Server response

Code Details

200 Response body

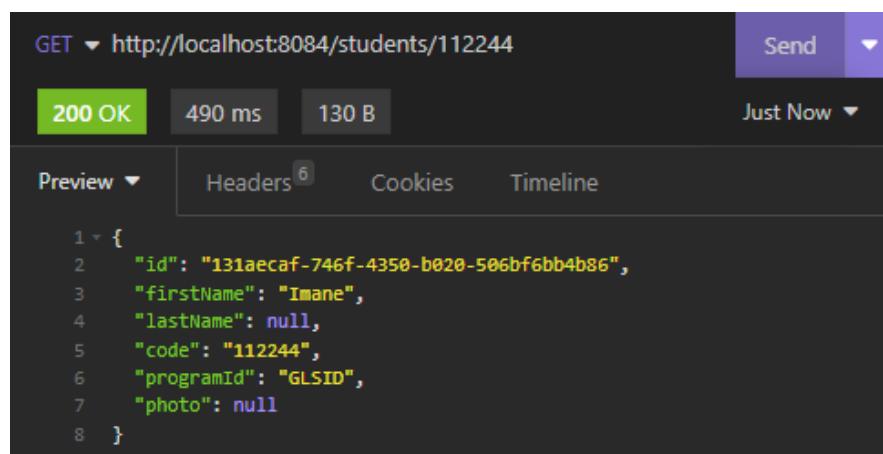
```
[  
  {  
    "id": "6420ce7c-4c4d-4154-b380-d70566e69bc6",  
    "firstName": "Alae",  
    "lastName": null,  
    "code": "112255",  
    "programId": "IAAD",  
    "photo": null  
  }]
```

4.9. CONSULTER UN ÉTUDIANT SACHANT SON CODE

• CODE

```
@GetMapping(path = "/students/{code}")
public Student getStudentByCode(@PathVariable String code)
{
    return studentRepository.findByCode(code);
}
```

• TESTE AVEC UN CLIENT REST



GET ▾ http://localhost:8084/students/112244

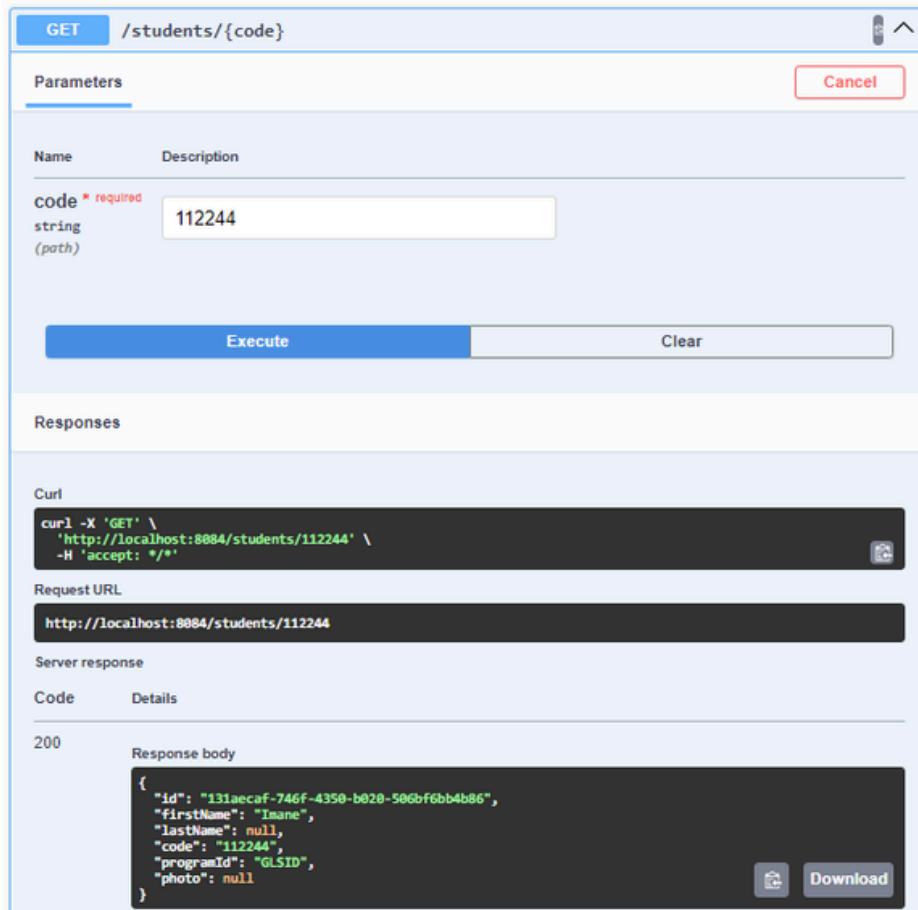
Send ▾

200 OK 490 ms 130 B Just Now ▾

Preview ▾ Headers 6 Cookies Timeline

```
1 ▾ {
  2   "id": "131aecaf-746f-4350-b020-506bf6bb4b86",
  3   "firstName": "Imane",
  4   "lastName": null,
  5   "code": "112244",
  6   "programId": "GLSID",
  7   "photo": null
  8 }
```

• TESTE AVEC SWAGGER UI



GET /students/{code}

Parameters

Name	Description
code * required	string (path)

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8084/students/112244' \
-H 'accept: */*'
```

Request URL

http://localhost:8084/students/112244

Server response

Code Details

200 Response body

```
{
  "id": "131aecaf-746f-4350-b020-506bf6bb4b86",
  "firstName": "Imane",
  "lastName": null,
  "code": "112244",
  "programId": "GLSID",
  "photo": null
}
```

Download

4.10. EFFECTUER UN NOUVEAU PAYEMENT AVEC LES DONNÉES ET LE RÉÇU DE PAYEMENT AU FORMAT PDF

• CODE

```
@PostMapping(path=@"/payments", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
public Payment savePayment(@RequestParam MultipartFile file, LocalDate date, double amount,
                           PaymentType type, String studentCode) throws IOException {
    Path folderPath = Paths.get(System.getProperty("user.home"), ...more: "master-iaad","payments");
    if(!Files.exists(folderPath)){
        Files.createDirectories(folderPath);
    }
    String fileName = UUID.randomUUID().toString();
    Path filePath = Paths.get(System.getProperty("user.home"), ...more: "master-iaad","payments",fileName+".pdf");
    Files.copy(file.getInputStream(), filePath);
    Student student = studentRepository.findByCode(studentCode);
    Payment payment = Payment.builder()
        .date(date)
        .type(type)
        .student(student)
        .amount(amount)
        .file(filePath.toUri().toString())
        .status(PaymentStatus.CREATED)
        .build();
    return paymentRepository.save(payment);
}
```

• TESTE AVEC SWAGGER UI

POST /payments

Cancel Reset

Name	Description
amount * required number(\$double) (query)	5200
type * required string (query)	TRANSFER
date * required string(\$date) (query)	2024-06-15
studentCode * required string (query)	112233

Request body multipart/form-data

file * required
string(\$binary)
Choisir un fichier EMAIL_FILTERIN..._PROJECT.ipynb

Execute Clear

4.11. METTRE À JOUR LE STATUS D'UN PAYEMENT

• CODE

```
@PutMapping("/payments/{id}/updateStatus")
public Payment updatePaymentStatus(@RequestParam PaymentStatus status, @PathVariable Long id) {
    Payment payment=paymentRepository.findById(id).get();
    payment.setStatus(status);
    return paymentRepository.save(payment);
}
```

• TESTE AVEC UN CLIENT REST

The screenshot shows a REST client interface with the following details:

- Method: PUT
- Endpoint: `lhost:8084/payments/2/updateStatus?status=VALIDATED`
- Status: 200 OK
- Time: 72.6 ms
- Size: 237 B
- Last Updated: Just Now
- Preview tab selected, showing the response body:

```
1  {
2      "id": 2,
3      "date": "2024-06-15",
4      "amount": 13805.0,
5      "type": "DEPOSIT",
6      "status": "VALIDATED",
7      "file": null,
8      "student": {
9          "id": "538d7900-204e-4cba-8809-3c546fd4ba2a",
10         "firstName": "Mohamed",
11         "lastName": null,
12         "code": "112233",
13         "programId": "SDIA",
14         "photo": null
15     }
16 }
```

• TESTE AVEC SWAGGER UI

The screenshot shows the Swagger UI for a PUT request to `/payments/{paymentId}/updateStatus`. The 'Parameters' section displays two parameters: `status` (required, string, query, value: VALIDATED) and `paymentId` (required, integer(\$int64), path, value: 2). Below the parameters are 'Execute' and 'Clear' buttons. The 'Responses' section includes a 'Curl' command, a 'Request URL' (http://localhost:8084/payments/2/updateStatus?status=VALIDATED), and a 'Server response' block. The 'Server response' block shows a 200 status with a JSON response body:

```
{
  "id": 2,
  "date": "2024-06-15",
  "amount": 13885,
  "tvoe": "DEPOSIT".
}
```

4.12. CONSULTER LE RÉÇU D'UN PAYEMENT (FICHIER PDF)

```
@GetMapping(path = "/paymentFile/{paymentId}", produces = MediaType.APPLICATION_PDF_VALUE)
public byte[] getPaymentFile(@PathVariable Long paymentId) throws IOException {
    Payment payment = paymentRepository.findById(paymentId).get();
    return Files.readAllBytes(Path.of(URI.create(payment.getFile())));
}
```

5. FAIRE UN REFACTORING DU CODE EN UTILISANT LA COUCHE SERVICE, LES DTOS ET LES MAPPERS

5.1. PAYMENTSERVICE

```
@Service @Transactional
public class PaymentService
{
    5 usages
    private PaymentRepository paymentRepository;
    2 usages
    private StudentRepository studentRepository;
    ± nohayla *
    public PaymentService(PaymentRepository paymentRepository,
                          StudentRepository studentRepository)
    {
        this.paymentRepository = paymentRepository;
        this.studentRepository = studentRepository;
    }
    1 usage ± nohayla *
    public Payment savePayment(MultipartFile file, double amount, PaymentType type,
                               LocalDate date, String studentCode) throws IOException
    {
        Path folderPath = Paths.get(System.getProperty("user.home"), ...more: "iaad", "payments");
        if(!Files.exists(folderPath)){
            Files.createDirectories(folderPath);
        }
        String fileName = UUID.randomUUID().toString();
        Path filePath = Paths.get(System.getProperty("user.home"), ...more: "iaad", "payments",fileName+".pdf");
        Files.copy(file.getInputStream(), filePath);
        Student student = studentRepository.findByCode(studentCode);
        Payment payment=Payment.builder()
            .type(type)
            .status(PaymentStatus.CREATED)
            .date(date)
            .student(student)
            .amount(amount)
            .file(filePath.toUri().toString())
            .build();
        return paymentRepository.save(payment);
    }
    1 usage ± nohayla *
    public byte[] getPaymentFile(Long id) throws IOException
    {
        Payment payment = paymentRepository.findById(id).get();
        return Files.readAllBytes(Path.of(URI.create(payment.getFile())));
    }

    1 usage ± nohayla *
    public Payment updatePaymentStatus(PaymentStatus status, Long paymentId)
    {
        Payment payment = paymentRepository.findById(paymentId).get();
        payment.setStatus(status);
        return paymentRepository.save(payment);
    }
}
```

5.2 PAYMENTDTO

```
@PutMapping(path="/payments/{paymentId}/updateStatus")
public Payment updatePaymentStatus(@RequestParam PaymentStatus status, @PathVariable Long paymentId){
    return paymentService.updatePaymentStatus(status,paymentId);
}
@PostMapping(path="/payments", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
public Payment savePayment(@RequestParam MultipartFile file, double amount, PaymentType type,
                           LocalDate date, String studentCode) throws IOException {
    return paymentService.savePayment(file,amount,type,date,studentCode);
}
@GetMapping(path="/payments/{id}/file",produces = MediaType.APPLICATION_PDF_VALUE)
public byte[] getPaymentFile(@PathVariable Long id) throws IOException {
    return paymentService.getPaymentFile(id);
}
```

5.3 PAYMENTDTO

```
@NoArgsConstructor @AllArgsConstructor @Getter @Setter @ToString @Builder
public class PaymentDTO {
    private Long id;
    private LocalDate date;
    private double amount;
    private PaymentType type;
    private PaymentStatus status;
}
```

B : DÉVELOPPER LA PARTIE FRONTEND EN UTILISANT ANGULAR AVEC ANGULAR MATERIAL POUR LA PARTIE DESIGN

1. CRÉER UN PROJET ANGULAR

```
PS D:\study\school\Master\S2\Systèmes_Distribués\Tp\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_2\frontend-angular> ng new frontend-ang-app
--directory=./ --no-stan
dalone
? 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)? Yes
CREATE angular.json (3151 bytes)
CREATE package.json (1293 bytes)
CREATE README.md (1095 bytes)
CREATE tsconfig.json (889 bytes)
CREATE .editorconfig (290 bytes)
CREATE .gitignore (629 bytes)
CREATE .vscode/extensions.json (134 bytes)
CREATE .vscode/launch.json (490 bytes)
CREATE .vscode/tasks.json (980 bytes)
CREATE src/app/app-routing.module.ts (255 bytes)
CREATE src/app/app.module.ts (467 bytes)
CREATE src/app/app.component.html (20239 bytes)
CREATE src/app/app.component.spec.ts (1118 bytes)
CREATE src/app/app.component.ts (227 bytes)
CREATE src/app/app.component.css (0 bytes)
CREATE src/main.ts (221 bytes)
CREATE tsconfig.app.json (342 bytes)
CREATE tsconfig.spec.json (287 bytes)
CREATE src/favicon.ico (15086 bytes)
```

2. COMPOSANTS

2.1 ADMIN-TEMPLATE

• CRÉATION DE COMPOSANT

```
a_Nohayla_Tp4_2\frontend-angular> ng g c admin-template
CREATE src/app/admin-template/admin-template.component.html (30 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 (697 bytes)
```

• TYPESCRIPT

```
@Component({
  selector: 'app-admin-template',
  templateUrl: './admin-template.component.html',
  styleUrls: ['./admin-template.component.css'
})
export class AdminTemplateComponent {
  constructor(public authService : AuthService) {

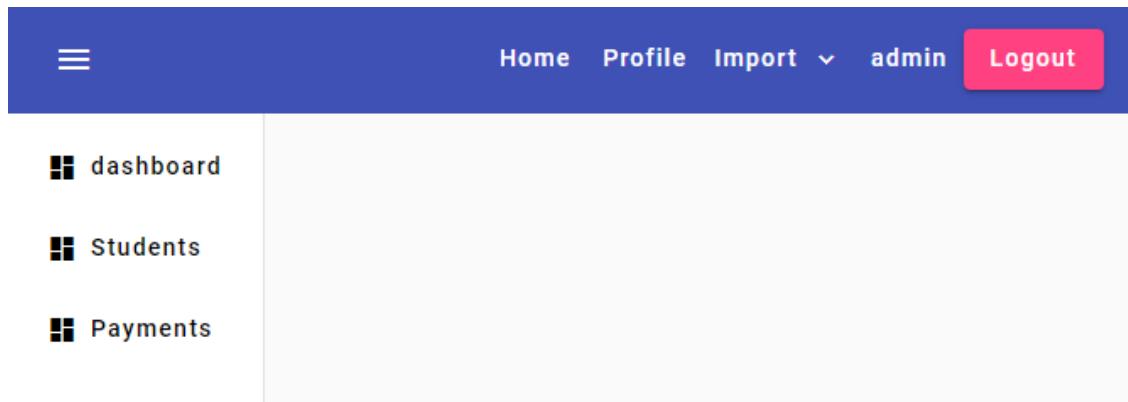
}

  logout() {
    this.authService.logout()
  }
}
```

• TEMPLATE HTML

```
<mat-toolbar color="primary">
  <button mat-icon-button (click)="myDrawer.toggle()">
    <mat-icon>menu</mat-icon>
  </button>
  <span style="..."></span>
  <button mat-button routerLink="/admin/home">Home</button>
  <button mat-button routerLink="/admin/profile" >Profile</button>
  <button *ngIf="authService.roles.includes('ADMIN')" mat-button [matMenuTriggerFor]="importMenu">

    <mat-icon iconPositionEnd>keyboard_arrow_down</mat-icon>
    Import
  </button>
  <mat-menu #importMenu>
    <button mat-menu-item routerLink="/admin/loadStudents">Load Students</button>
    <button mat-menu-item routerLink="/admin/loadPayments">Load Payments</button>
  </mat-menu>
  <button mat-button *ngIf="authService.isAuthenticated">
    {{authService.username}}
  </button>
  <button mat-raised-button color="accent" (click)="logout()" >
    Logout
  </button>
</mat-toolbar>
<mat-drawer-container>
```



dashboard

Students

Payments

2.2 LOGIN

• CRÉATION DE COMPOSANT

```
PS D:\study\school\Master\S2\Systèmes_Distribués\Tp\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_2\frontend-angular> ng g c login
CREATE src/app/login/login.component.html (21 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 (777 bytes)
```

• TYPESCRIPT

```
export class LoginComponent {  
}
```

• TEMPLATE HTML

```
<p>login works!</p>
```

2.3 STUDENTS

• CRÉATION DE COMPOSANT

```
PS D:\study\school\Master\S2\Systèmes_Distribués\Tp\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_2\frontend-angular> ng g c students
CREATE src/app/students/students.component.html (24 bytes)
CREATE src/app/students/students.component.ts (217 bytes)
CREATE src/app/students/students.component.css (0 bytes)
UPDATE src/app/app.module.ts (869 bytes)
```

• TYPESCRIPT

```
export class StudentsComponent implements OnInit, AfterViewInit {
  public students: any[] = [];
  public dataSource: MatTableDataSource<any>;
  public displayedColumns: string[] = ['id', 'firstName', 'lastName', 'code', 'programId', 'photo'];

  @ViewChild(MatPaginator) paginator!: MatPaginator;
  @ViewChild(MatSort) sort!: MatSort;

  no usages
  constructor(private http: HttpClient) {
    this.dataSource = new MatTableDataSource(this.students);
  }

  no usages
  ngOnInit(): void {
    this.http.get<any[]>(`url: "http://localhost:8084/students`)
      .subscribe(observerOrNext: {
        next: data: any[] => {
          this.students = data;
          this.dataSource.data = this.students;
        },
        error: err => {
          console.error(err);
        }
      });
  }
}
```

• TEMPLATE HTML

```
<div class="container">
  <mat-card>
    <mat-card-header>
      <mat-card-title>
        Students
      </mat-card-title>
    </mat-card-header>
    <mat-divider></mat-divider>
    <mat-card-content>
      <table matSort mat-table [dataSource]="dataSource" class="mat-elevation-z1">
        <!-- ID Column -->
        <ng-container matColumnDef="id">
          <th mat-header-cell *matHeaderCellDef mat-sort-header> ID </th>
          <td mat-cell *matCellDef="let element"> {{element.id}} </td>
        </ng-container>
        <!-- First Name Column -->
        <ng-container matColumnDef="firstName">
          <th mat-header-cell *matHeaderCellDef mat-sort-header> First Name </th>
          <td mat-cell *matCellDef="let element"> {{element.firstName}} </td>
        </ng-container>
        <!-- Last Name Column -->
        <ng-container matColumnDef="lastName">
          <th mat-header-cell *matHeaderCellDef mat-sort-header> Last Name </th>
          <td mat-cell *matCellDef="let element"> {{element.lastName}} </td>
        </ng-container>
      </table>
    </mat-card-content>
  </mat-card>
</div>
```

Students					
	ID	First Name	Last Name	Code	Program ID
538d7900-204e-4cba-8809-3c546fd4ba2a		Mohamed		112233	SDIA
131aecaf-746f-4350-b020-506bf6bb4b86		Imane		112244	GLSID
6420ce7c-4c4d-4154-b380-d70566e69bc6		Alae		112255	IAAD
82fc6c7e-cb05-4f2b-835b-187851ec5251		Najat		112266	GLSID

Items per page: 5

1 - 4 of 4 |< < > >|

2.4 DASHBOARD

- CRÉATION DE COMPOSANT

```
PS D:\study\school\Master\S2\Systèmes_Distribués\Tp\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_2\frontend-angular> ng g c dashboard
CREATE src/app/dashboard/dashboard.component.html (25 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 (965 bytes)
```

- TYPESCRIPT

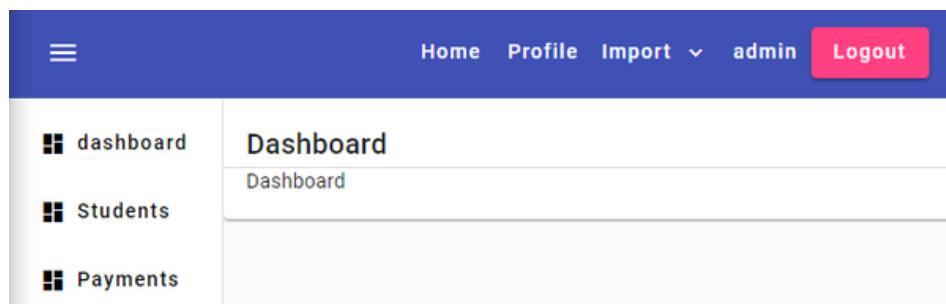
```
export class DashboardComponent {
```

```
}
```

• TEMPLATE HTML

```
<div class ="container">
  <mat-card>
    <mat-card-header>
      <mat-card-title>
        Dashboard
      </mat-card-title>
    </mat-card-header>
    <mat-divider></mat-divider>
    <mat-card-content>
      Dashboard
    </mat-card-content>
  </mat-card>

</div>
```



2.5 PAYMENTS

• CRÉATION DE COMPOSANT

```
a_Nohayla_Tp4_2\frontend-angular> ng g c payments
CREATE src/app/payments/payments.component.html (24 bytes)
CREATE src/app/payments/payments.component.ts (217 bytes)
CREATE src/app/payments/payments.component.css (0 bytes)
UPDATE src/app/app.module.ts (1057 bytes)
```

• TYPESCRIPT

```
export class PaymentsComponent implements OnInit{
  public payments : any;
  public dataSource : any;
  public displayedColumns :string[] = ['id','date','amount','type','status', 'firstName'];

  @ViewChild(MatPaginator) paginator!: MatPaginator ;
  @ViewChild(MatSort) sort! : MatSort;
  no usages
  constructor(private http: HttpClient) {
  }
  no usages
  ngOnInit(): void {
    this.http.get( url: "http://localhost:8084/payments")
      .subscribe( observerOrNext: {
        next : data : Object => {
          this.payments = data;
          this.dataSource = new MatTableDataSource(this.payments);
          this.dataSource.paginator = this.paginator;
          this.dataSource.sort = this.sort;
        },
        error : err => {
          console.log(err);
        }
      })
  }
}
```

• TEMPLATE HTML

```
<div class ="container">
  <mat-card>
    <mat-card-header>
      <mat-card-title>
        Payments
      </mat-card-title>
    </mat-card-header>
    <mat-divider></mat-divider>
    <mat-card-content>
      <table matSort mat-table [dataSource]="dataSource" class="mat-elevation-z1">
        <!-- ID Column -->
        <ng-container matColumnDef="id">
          <th mat-header-cell *matHeaderCellDef mat-sort-header > ID </th>
          <td mat-cell *matCellDef="let element"> {{element.id}} </td>
        </ng-container>
        <!-- Date Column -->
        <ng-container matColumnDef="date">
          <th mat-header-cell *matHeaderCellDef mat-sort-header> Date </th>
          <td mat-cell *matCellDef="let element"> {{element.date}} </td>
        </ng-container>
        <!-- Amount Column -->
        <ng-container matColumnDef="amount">
          <th mat-header-cell *matHeaderCellDef mat-sort-header> Amount </th>
          <td mat-cell *matCellDef="let element"> {{element.amount}} </td>
        </ng-container>
      </table>
    </mat-card-content>
  </mat-card>
</div>
```

		Payments					
		ID	Date	Amount	Type	Status	Student
		1	2024-06-15	19792	CHECK	CREATED	Mohamed
		2	2024-06-15	13805	DEPOSIT	VALIDATED	Mohamed
		3	2024-06-15	10423	CHECK	CREATED	Mohamed
		4	2024-06-15	8971	CHECK	CREATED	Mohamed
		5	2024-06-15	18638	DEPOSIT	CREATED	Mohamed

Items per page: 5 | < < > >|

2.6 HOME

- CRÉATION DE COMPOSANT

```
PS D:\study\school\Master\S2\Systèmes_Distribués\Tp\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_2\frontend-angular> ng g c home
CREATE src/app/home/home.component.html (20 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 (1133 bytes)
```

- TYPESCRIPT

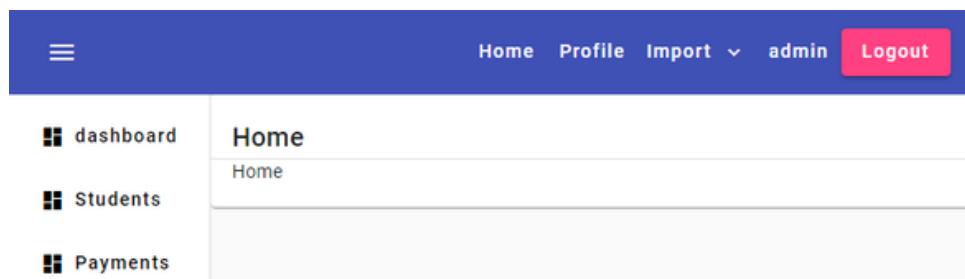
```
export class HomeComponent {
```

```
}
```

• TEMPLATE HTML

```
<div class ="container">
  <mat-card>
    <mat-card-header>
      <mat-card-title>
        Home
      </mat-card-title>
    </mat-card-header>
    <mat-divider></mat-divider>
    <mat-card-content>
      Home
    </mat-card-content>
  </mat-card>

</div>
```



2.7 PROFILE

• CRÉATION DE COMPOSANT

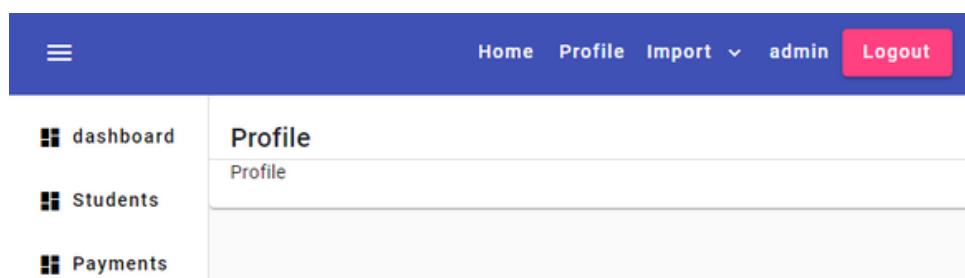
```
a_Nohayla_Tp4_2\frontend-angular> ng g c profile
CREATE src/app/profile/profile.component.html (23 bytes)
CREATE src/app/profile/profile.component.spec.ts (631 bytes)
CREATE src/app/profile/profile.component.ts (213 bytes)
CREATE src/app/profile/profile.component.css (0 bytes)
UPDATE src/app/app.module.ts (1221 bytes)
```

• TYPESCRIPT

```
export class ProfileComponent {  
}  
}
```

• TEMPLATE HTML

```
<div class ="container">  
  <mat-card>  
    <mat-card-header>  
      <mat-card-title>  
        Profile  
      </mat-card-title>  
    </mat-card-header>  
    <mat-divider></mat-divider>  
    <mat-card-content>  
      Profile  
    </mat-card-content>  
  </mat-card>  
  
</div>
```



2.8 LOAD-STUDENTS

• CRÉATION DE COMPOSANT

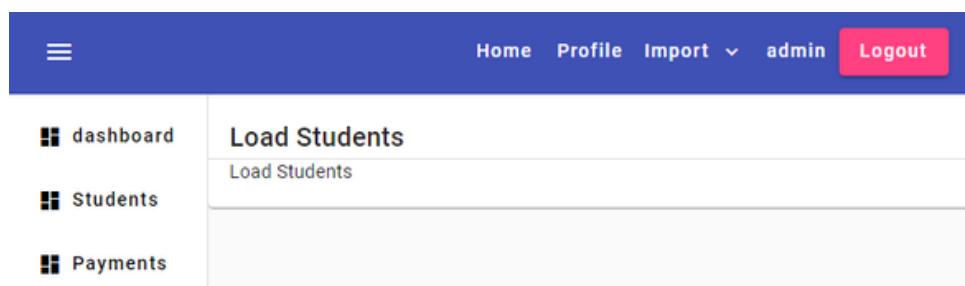
```
PS D:\study\school\Master\S2\Systèmes_Distribués\Tp\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_2\frontend-angular> ng g c load-students  
CREATE src/app/load-students/load-students.component.html (29 bytes)  
CREATE src/app/load-students/load-students.component.spec.ts (667 bytes)  
CREATE src/app/load-students/load-students.component.ts (236 bytes)  
CREATE src/app/load-students/load-students.component.css (0 bytes)  
UPDATE src/app/app.module.ts (1331 bytes)
```

- TYPESCRIPT

```
export class LoadStudentsComponent {  
}  
}
```

- TEMPLATE HTML

```
<div class ="container">  
  <mat-card>  
    <mat-card-header>  
      <mat-card-title>  
        Load Students  
      </mat-card-title>  
    </mat-card-header>  
    <mat-divider></mat-divider>  
    <mat-card-content>  
      Load Students  
    </mat-card-content>  
  </mat-card>  
  
</div>
```



2.9 LOAD-PAYMENTS

• CRÉATION DE COMPOSANT

```
PS D:\study\school\Master\S2\Systèmes_Distribués\Tp\Anoada_Nohayla_Tp4\Anoada_Nohayla_Tp4_2\frontend-angular> ng g c load-payments
CREATE src/app/load-payments/load-payments.component.html (29 bytes)
CREATE src/app/load-payments/load-payments.component.spec.ts (667 bytes)
CREATE src/app/load-payments/load-payments.component.ts (236 bytes)
CREATE src/app/load-payments/load-payments.component.css (0 bytes)
UPDATE src/app/app.module.ts (1441 bytes)
```

• TYPESCRIPT

```
export class LoadPaymentsComponent {  
}
```

• TEMPLATE HTML

```
<div class ="container">  
  <mat-card>  
    <mat-card-header>  
      <mat-card-title>  
        Load Payments  
      </mat-card-title>  
    </mat-card-header>  
    <mat-divider></mat-divider>  
    <mat-card-content>  
      Load Payments  
    </mat-card-content>  
  </mat-card>  
  
</div>
```

