Angular Identity Management

Présentation

Angular est développé par Google. C'est est un framework frontend Javascript comme React, VueJS, Ember, ... Il se diffère cependant des autres framework car il faut utiliser TypeScript et non JavaScript. De plus, il permet de générer des Progressive Web Apps distribuables sur n'importe quelle plateforme (Smartphone, …)

TypeScript est une surcouche de JavaScript, il est open source et est développé par Microsoft. Il impose un typage strict ce qui permet de vérifier la cohérence des variables/valeurs. TypeScript est "transpilé" en ECMAScript, il est donc traduit en JS.

Installation

Installation des outils

Pour développer avec Angular, vous devez installer les outils suivants :

- Node.JS : installez la dernière version LTS (Pour Debian/Ubuntu : utilisez plutôt le lien suivant : https://github.com/nodesource/distributions/blob/master/README.md). Vérifiez la version avec la commande `node --version`
- NPM : NPM est un gestionnaire de package. Déjà installé avec la commande précédente mais vous pouvez installer la dernière version de NPM avec la commande `npm install -g npm@latest`. Dans ce document, nous utiliserons Yarn
- Yarn : Yarn est réputé plus stable et plus performant que NPM. Installez la dernière version de Yarn avec la commande `npm install --global yarn@latest`

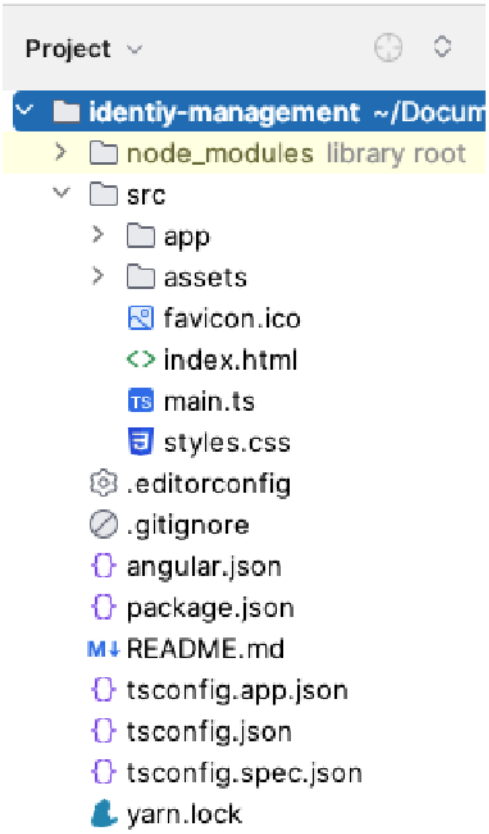
Installation du client Angular

Nous allons utiliser la version LTS 15. Vous pouvez, si vous le voulez installer la version 16).
Le client Angular permet de construire, générer, tester une application Angular (et plus encore). C'est un outil indispensable pour le développeur.
Exécutez la commande `sudo npm install --global @angular/cli` ou `sudo yarn global add @angular/cli`
La dépendance @angular définit le scope associé à l'organisation angular (Google plus précisément).
Le client permet d'utiliser l'outil ng en ligne de commande. Nous verrons cette commande au fil du projet.

Création du projet

Création de la structure

Placez-vous dans le répertoire de votre choix et entrez la commande suivante dans un terminal : `ng new identity-management`
Lors de la création, ajoutez le routage et sélectionnez le style de CSS voulu (CSS dans cet exemple).
La structure est la suivante :



node_modules: modules (dépendances) de l'application
src: répertoire principal du projet
app : contient les fichiers liés à l'objet app (application principale)
asset : contient tous les fichiers images, …

index.html : point d'entrée de l'application
main.ts : appelé par index.html, point d'entrée d'Angular

angular.json : configuration d'Angular
package.json : liste des modules utilisés par l'application

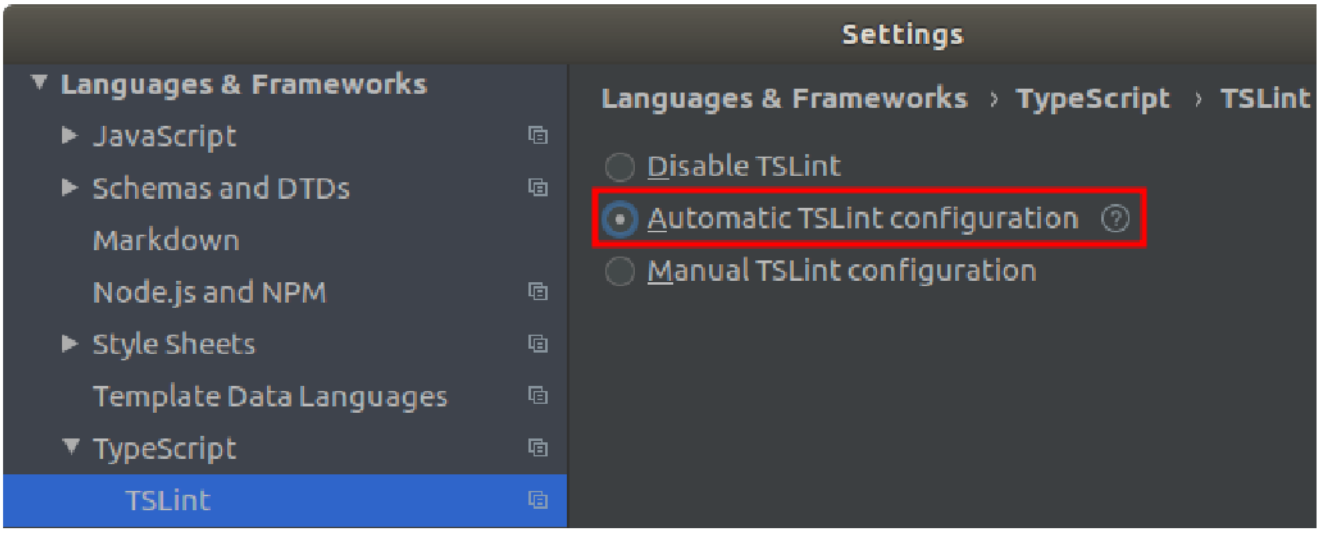
Le répertoire node_modules est créé et mis à jour en fonction des modules listés dans package.json. Il peut donc être supprimer et recréer avec `yarn install` ou mis à jour avec `yarn update`.

Lancement de l'application

Placez-vous dans le répertoire du projet et lancez l'application en ligne de commande : `ng serve --open`.
L'option `--open` lance le navigateur. Par la suite, nous utiliserons WebStorm pour lancer l'application.

Ouverture du projet avec WebStorm

Vous pouvez utiliser plusieurs IDE : IntelliJIDEA, WebStorm, Visual Studio Code, ~~Notepad++~~, … Nous allons, dans ce projet, utiliser WebStorm.
Lancez WebStorm, choisissez "Open" et ouvrez le répertoire de votre application Angular, c'est à dire le répertoire « identity_management ».
Vérifier que TSLint est activé (Settings) :


Settings
Languages & Frameworks
JavaScript
Schemas and DTDs
Markdown
Node.js and NPM
Style Sheets
Template Data Languages
TypeScript
TSLint
Languages & Frameworks › TypeScript › TSLint
Disable TSLint
Automatic TSLint configuration
Manual TSLint configuration

Le composant app

Le fichier index.html

Le fichier src/index.html est le point d'entrée de l'application. Il contient le code suivant :

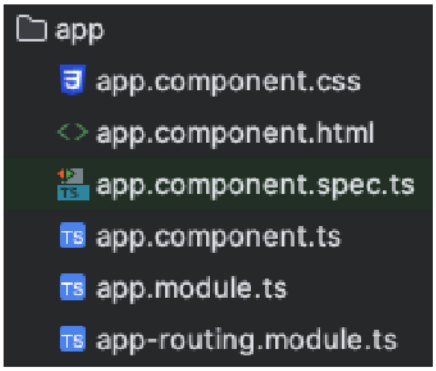
```
<body>
 <app-root></app-root>
</body>
```

La balise app-root indique à Angular d'injecter le composant nommé app-root à la place de cette balise.
Angular regroupe tous les fichiers liés à un composant dans un seul répertoire d'où le répertoire app.

Le composant app est le composant de base qui lancera tous les autres composants (ils seront donc inclus dans le composant app - nested en anglais).

Structure du composant app

La structure de fichier du composant app est la suivante :



app : nom du composant

app.component.html : code html et balise angular
app.component..css : css du composant
app.component.spec.ts : tests e2e

app.component.ts : point d'entrée du composant

app.module.ts : définit les modules utilisés par l'application
app-routing.module.ts : définit les routes de l'application

Les fichiers app.module.ts et app-routing.module.ts sont propres au composant app car c'est le composant racine.
Les fichiers <nom_du_composant>.component.html , <nom_du_composant>.component..css, <nom_du_composant>.component.spec.ts et <nom_du_composant>.component.ts sont commun aux composants.

Le component app

app est un composant au sens angular. Pour mieux découvrir un composant, voyons le code du fichier app.components.ts :

```
import { Component } from '@angular/core';
@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.scss']
})
export class AppComponent {
 title = 'identity-management';
}
```

Explications :

- import { Component } from '@angular/core' : Importation du composant “Component” depuis le module “@angular/core”. Ce composant est le composant de base d’angular;
- @Component: annotation (decorator component en angular). permet de définir le comportement de la classe “AppComponent”:
    - selector: 'app-root' : Nom de la balise HTML. Lorsque angular rencontrera la balise <app-root></app-root> (voir fichier src/index.html), il remplacera cette balise par le composant AppComponent
    - templateUrl: './app.component.html' : fichier template contenant le code html et les balises angular qui seront affichées
    - styleUrls: ['./app.component.scss'] : tableau contenant la liste des feuilles de style utilisées

Pour l’instant il n'y a que très peu de code. Par la suite un composant définira sa logique de code dans ce fichier.

Le module app

app est aussi un module au sens angular. Un module permet de définir le contexte de l'application.

Le module pour l’application

Le code de app.module.ts est le suivant :

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

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

Explications :

- @NgModule : decorator pour définir un module
    - **declarations** : définit les composants, directives et pipes qui appartiennent à ce module
    - **imports** : modules dont les classes sont utilisées par ce module.
    - **providers** : définit les services existants (qui seront injectables dans l'application)
    - **bootstrap** : définit le composant racine

Le module pour le routage

Ce module permet de définir le routage de l'application. Le code app-routing.module.ts est le suivant :

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

```
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [];

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

Explications :

- @NgModule : decorator pour définir un module
  imports : modules dont les classes sont utilisées par ce module. Le routage sera disponible pour tout le module.
  exports : définit les modules qui seront utilisés lors de l'importation de ce module.

Les composants

Un composant est une classe permettant d'avoir un rendu HTML, il est constitué de 3 parties :

- Fichier TypeScript :
  - décrit le comportement du composant
- Fichier HTML :
  - contient le template
  - utilise des variables et/ou méthodes du composant par interpolation, ex: {{ var }}
- Fichier CSS (uniquement à ce composant)

Pour créer un composant ("component"), il faut exécuter la commande suivante : `ng generate component <nom_du_composant>`.

Les composants de base

Afin d'intégrer Material UI, nous allons créer les composants pour lister les utilisateurs et pour les routes non trouvées.
Pour créer un composant ("component"), il faut exécuter la commande suivante : `ng generate component <nom_du_composant>`.
Dans un terminal (à la racine du projet), exécutez les commandes suivantes :

- `ng generate component ldap-list`
- `ng generate component page-not-found`

Nous les implémenterons après avoir intégré Material UI.

Les routes

La navigation se fait par "routing" :

- Avec le module Router
- D'après des chemins

```
const routes: Routes = [
  { path: 'first-component', component: FirstComponent },
  { path: 'second-component', component: SecondComponent },
  { path: '',   redirectTo: '/first-component', pathMatch: 'full' },
  { path: '**', component: PageNotFoundComponent },
];
```

- Avec le composant router-outlet (permet d'inclure un template)

Les routes de base

Nous allons modifier le module AppRoutingModule (src/app/app-routing.module.ts).
Les routes pour nos composants seront les suivantes :

```
const routes: Routes = [
  { path: 'users/list', component: LdapListComponent },
  { path: '**', component: PageNotFoundComponent }
];

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

Material UI

Présentation

Material UI est un framework CSS créé par Google : https://material.angular.io/, c'est une bibliothèque de composants UI.

Installation

Exécutez la commande suivante (à la racine du projet) : `ng add @angular/material`
Lors de la création :

- Choisissez le thème voulu (Indigo/Pink par exemple)
- Appliquer la typographie au projet
- Ajouter ou non les animations (nous n'en ferons pas dans ce projet)

Ajout des modules de Material

Nous allons générer le module qui importera les librairies de Material.
Nous allons inclure les modules Material nécessaires à l'application dans ce module puis les exporter pour pouvoir les utiliser dans l'application.

Exécutez la commande suivante dans un terminal (à la racine du projet) : `ng generate module app-material --flat --module=app`
Explications :

- `--flat` : ne crée pas de sous dossier mais créer le fichier à la racine
- `--module=app` inclut ce module dans les import du module de l'application app

Remarque : Le module AppMaterialModule est automatiquement ajouté au Module AppModule.

Le code sera le suivant :

```
/* MATERIAL import omis */
@NgModule({
  exports: [
    BrowserAnimationsModule,
    LayoutModule,
    MatToolbarModule,
    MatButtonModule,
    MatCheckboxModule,
    MatSidenavModule,
    MatIconModule,
    MatListModule,
    MatTableModule,
    MatPaginatorModule,
  ]
})
export class AppMaterialModule { }
```

Nous ajouterons des modules Material dans ce module au fur et à mesure du développement de l'application.

Ajout de la barre de navigation

Nous allons utiliser les schematic pour ajouter la navigation. La navigation sera de type side-bar sur la gauche, ouverture/fermeture dynamique d'après la dimension de la fenêtre.

Exécutez la commande suivante : `ng generate @angular/material:navigation navbar --module=app` où navbar est le nom du composant et app est le module pour lequel nous voulons ajouter ce composant.

Le code de "navbar.component.hml" est à modifier avec le code suivant :

```
<div>
  <mat-toolbar color="primary">
    <button
      type="button"
      aria-label="Toggle sidenav"
      mat-icon-button
      (click)="drawer.toggle()"
      *ngIf="isHandset$ | async">
      <mat-icon aria-label="Side nav toggle icon">menu</mat-icon>
    </button>
    <span>Gestion ds utilisateurs</span>

    <!-- This fills the remaining space of the current row -->
    <span class="fill-remaining-space"></span>

    <button mat-button (click)="logout()">
      <mat-icon>exit_to_app</mat-icon>
      <span>Déconnexion</span>
    </button>
  </mat-toolbar>
```

```
  <mat-sidenav-container class="sidenav-container">
    <mat-sidenav #drawer class="sidenav" fixedInViewport
                 [fixedTopGap]="64"
      [attr.role]="(isHandset$ | async) ? 'dialog' : 'navigation'"
      [mode]="(isHandset$ | async) ? 'over' : 'side'"
      [opened]="(isHandset$ | async) === false">
      <mat-toolbar>Menu</mat-toolbar>
      <mat-nav-list>
        <h2 matSubheader>Utilisateurs</h2>
        <a mat-list-item
           routerLink="/users/dashboard"
           routerLinkActive="list-item-active"
        >
          <mat-icon>dashboard</mat-icon>
          Dashboard
        </a>
        <a mat-list-item
           routerLink="/users/list"
           routerLinkActive="list-item-active"
        >
          <mat-icon>list</mat-icon>
          Utilisateurs
        </a>

        <mat-divider></mat-divider>

        <h2 matSubheader>Autre sous-menu</h2>
        <a mat-list-item
        >
          <mat-icon>save_alt</mat-icon>
          Un autre lien
        </a>
      </mat-nav-list>
    </mat-sidenav>
    <mat-sidenav-content>
      <!-- Add Content Here -->
      <div class="main_content">
        <router-outlet></router-outlet>
      </div>
    </mat-sidenav-content>
  </mat-sidenav-container>

</div>
```

Explication :
La balise <router-outlet></router-outlet> permet d'afficher le contenu de la page géré par le routage.

Modification de app.component.html ; il faut modifier le html pour afficher notre composant navbar :
`<app-navbar></app-navbar>`
Le fichier contient cette unique ligne !

Ajouter le code suivant à "navbar.component.css" :

```
.sidenav {
  width: 200px;
}

.main_content {
 margin: 15px 15px 15px 15px;
}

.fill-remaining-space {
 /* This fills the remaining space, by using flexbox.
    Every toolbar row uses a flexbox row layout. */
 flex: 1 1 auto;
}
```

Et enfin le code de navbar.component.ts :

```ts
@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styleUrls: ['./navbar.component.scss']
})
export class NavbarComponent {

  isHandset$: Observable<boolean>
      = this.breakpointObserver.observe(Breakpoints.Handset)
      .pipe(
          map(result => result.matches),
          shareReplay()
      );

  constructor(private breakpointObserver: BreakpointObserver) {}

  logout(): void  {
    // Plus tard ...
  }
}
```

Affichage des utilisateurs (composant ldap-list)

Nous voulons afficher la liste des utilisateurs LDAP. Pour ce faire nous allons utiliser un service pour obtenir les données puis les afficher. Mais dans un premier temps nous allons utiliser des fausses données.
Définition Wikipédia : "*En programmation orientée objet, les mocks (simulacres ou mock object) sont des objets simulés qui reproduisent le comportement d'objets réels de manière contrôlée. Un programmeur crée un mock dans le but de tester le comportement d'autres objets, réels, mais liés à un objet inaccessible ou non implémenté. Ce dernier est alors remplacé par un mock.*"

Le CSS

Le css est pour l'instant très simple :

```
table {
  width: 90%;
}
```

Le modèle

Le composant va utiliser un objet de la classe UserLdap qui contient les attributs d'un utilisateur.

Créez un répertoire "models" sous le répertoire « src/app » puis un fichier nommé "user-ldap.ts". Il contient le code suivant :

```
export interface UserLdap {
  login: string;
  nom: string;
  prenom: string;
  nomComplet: string;
  motDePasse: string | null;
  mail: string;
  role: string;
  employeNumero: number;
  employeNiveau: number;
  dateEmbauche: string;
  publisherId: number;
  active: boolean;
}
```

Les fausses données

Vous devez créer un fichier “ldap-mock-data.ts” dans le répertoire “model” et ajouter les fausses données (ici un seul élément est représenté) :

```ts
import {UserLdap} from './user-ldap';

export const LDAP_USERS: UserLdap[] = [
  {
    login: 'test.v1',
    nom: 'V1',
    prenom: 'Test',
    nomComplet: 'V1 Test',
    motDePasse: null,
    mail: 'test.v1@epsi.fr',
    role: 'ROLE_USER',
    employeNumero: 1234,
    employeNiveau: 120,
    dateEmbauche: '2020-01-01',
    publisherId: 1,
    active: true,
  },
  {
    login: 'test.v2',
    nom: 'V2',
    prenom: 'Test',
    nomComplet: 'V2 Test',
    motDePasse: null,
    mail: 'test.v2@epsi.fr',
    role: 'ROLE_USER',
    employeNumero: 2234,
    employeNiveau: 220,
    dateEmbauche: '2020-02-02',
    publisherId: 2,
    active: true,
  },
];
```

Le composant

Le composant va utiliser une table Material avec une pagination (voir Table). Le code sera le suivant (il faut bien sûr ajouter les imports) :

```
@Component({
  selector: 'app-ldap-list',
  templateUrl: './ldap-list.component.html',
  styleUrls: ['./ldap-list.component.css']
})
export class LdapListComponent implements OnInit {
  displayedColumns: string[] = ['nomComplet', 'mail', 'employeNumero'];
  dataSource : MatTableDataSource<UserLdap, M...  = new MatTableDataSource<UserLdap>( initialData: []);

  @ViewChild(MatPaginator, {static: true}) paginator: MatPaginator | null;

  no usages
  constructor() {
    this.paginator = null;
  }

  no usages
  ngOnInit(): void {
    this.dataSource.paginator = this.paginator;
    this.getUsers();
  }

  1 usage
  private getUsers() : void {
    this.dataSource.data = LDAP_USERS
  }
}
```

Le décorateur @ViewChild(MatPaginator, …} paginator: MatPaginator | null (Angular @ViewChild: In-Depth Explanation (All Features Covered)) permet d'injecter la référence mat-paginator dans l'attribut paginator. (Angular traduit le composant MatPaginator avec la balise mat-paginator). Autrement dit l'attribut paginator de la classe LdapListComponent est lié à la balise HTML mat-paginator après la création de la vue.

La méthode ngOnInit permet de lier le composant mais il n'est pas initialisé. Le composant sera disponible après la création de la vue. Pour s'en convaincre (optionnel), vous pouvez modifier le code de la classe ainsi (ensuite remettez le code précédent) :

```
export class LdapListComponent implements OnInit, AfterViewInit {
  /* Le code précédent */
  ngOnInit(): void {
    console.log('Values on ngOnInit():');
    this.dataSource.paginator = this.paginator;
    console.log("Mat Paginator:", this.paginator);
  }
  ngAfterViewInit(): void  {
    console.log('Values on ngAfterViewInit():');
    console.log("Mat Paginator:", this.paginator);
  }
}
```

Ouvrez la console pour voir l'état du paginator.

Le HTML

Le HTML sera le suivant :

```
<div class="mat-elevation-z8">
  <table mat-table [dataSource]="dataSource">

    <!-- Nom complet Column -->
    <ng-container matColumnDef="nomComplet">
      <th mat-header-cell *matHeaderCellDef> Nom complet </th>
      <td mat-cell *matCellDef="let element"> {{element.nomComplet}} </td>
    </ng-container>

    <!-- Mail Column -->
    <ng-container matColumnDef="mail">
      <th mat-header-cell *matHeaderCellDef> Mail </th>
      <td mat-cell *matCellDef="let element"> {{element.mail}} </td>
    </ng-container>

    <!-- employeNumero Column -->
    <ng-container matColumnDef="employeNumero">
      <th mat-header-cell *matHeaderCellDef> Numero </th>
      <td mat-cell *matCellDef="let element"> {{element.employeNumero}} </td>
    </ng-container>

    <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
    <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
  </table>

  <mat-paginator
          [pageSizeOptions]="[25, 50]"
          showFirstLastButtons>

  </mat-paginator>
</div>
```

Vous l'aurez compris, le but du document n'est pas d'apprendre Material UI (vaste sujet) mais d'essayer de maîtriser Angular.

Testez ! http://localhost:4200

Filtrage des utilisateurs

Nous allons filtrer les utilisateurs d'après le nom complet et l'attribut "active".
Attention : Vous devez importer MatFormFieldModule, MatSlideToggleModule et MatInputModule dans le module dédié à Material.

Ajout dans le HTML

Avant la balise table, nous ajoutons un champ texte de saisie et un bouton on/off :

```
...
<div class="toolbar-filter">
  <mat-form-field style="width: 40%">
    <mat-label>Nom complet</mat-label>
    <input matInput (keyup)="applyFilter($event)"
      placeholder="Entrez le début d'un nom"/>
  </mat-form-field>
    
  <mat-slide-toggle [checked]="unactiveSelected"
                    (change)="unactiveChanged($event)">
    Utilisateur déactivé uniquement
  </mat-slide-toggle>
</div>

<table mat-table [dataSource]="dataSource">
  ...
```

Ajout dans le CSS

Un peu de CSS:

```
.toolbar-filter {
  display: flex;
  align-content: center;
  align-items: center;
  margin: 12px;
  width: 90%;
}
```

Ajout dans le composant

Pour le filtrage d'après le début du nom complet, nous allons utiliser la méthode “filterPredicate” et l’attribut “filter” du “datasource” :

```
ngOnInit(): void {
  this.dataSource.paginator = this.paginator;
  this.dataSource.filterPredicate = (data: UserLdap, filter: string)
      => this.filterPredicate(data, filter);

  this.getUsers();
}
filterPredicate(data, filter): boolean {
  return  !filter || data.nomComplet.toLowerCase().startsWith(filter);
}
applyFilter($event: KeyboardEvent): void  {
  const filterValue = ($event.target as HTMLInputElement).value;
  this.dataSource.filter = filterValue.trim().toLowerCase();
}
...
```

Pour le filtrage de “Utilisateur désactivé uniquement”, nous allons utiliser un attribut et faire le filtrage dans la récupération des utilisateurs :

```
...
2 usages
private getUsers() : void {
  if (this.unactiveSelected) {
    this.dataSource.data = LDAP_USERS.filter( user : U
      !user.active
    })
  } else {
    this.dataSource.data = LDAP_USERS;
  }
}

...
1 usage
unactiveChanged($event: MatSlideToggleChange) : void
  this.unactiveSelected = $event.checked;
  this.getUsers();
}
```

Liens

https://angular.io/start

https://blog.angular.io/

https://blog.angular-university.io/

https://guide-angular.wishtack.io/

https://openclassrooms.com/fr/courses/4668271-developpez-des-applications-web-avec-angular

https://www.typescriptlang.org/docs/home.html

https://blog.soat.fr/

https://www.positronx.io/

Parent Child Two way binding
https://medium.com/@preethi.s/angular-custom-two-way-data-binding-3e618309d6c7

Organisation par module
Mise en place de la sécurité
https://angular.io/guide/route

Login:
https://loiane.com/2017/08/angular-hide-navbar-login-page/

Dialog:
https://blog.angular-university.io/angular-material-dialog/

Angular Material
https://material.angular.io/

https://medium.com/@ismapro/first-steps-with-angular-7-with-angular-cli-and-angular-material-d69f55d8ac51

https://www.positronx.io/create-angular-material-8-custom-theme/

https://akveo.github.io/ngx-admin/

https://auth0.com/blog/creating-beautiful-apps-with-angular-material/