SEG3502 – Lab 4

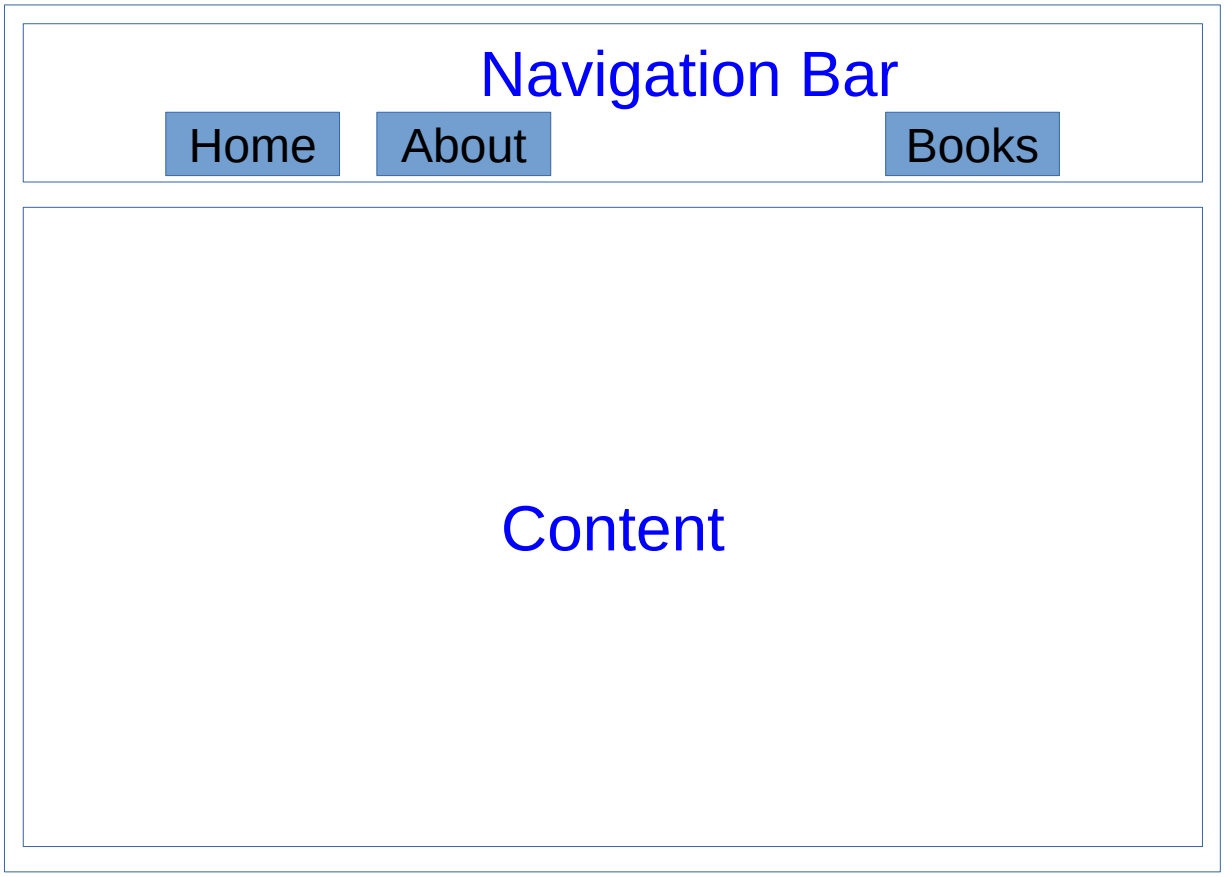
Angular – Routage

L'objectif de ce laboratoire est de démontrer le routage d'Angular. Nous allons construire un exemple d'application qui permet de naviguer entre les différentes «pages» d'une application.

Le code source du laboratoire est disponible dans le référentiel Github https://github.com/stephanesome/angRouting.git.

Librairie

Nous illustrons avec un prototype d'application Bookstore avec la mise en page suivante.



L'application présente une barre de navigation avec des boutons. Cliquer sur chacun de ces boutons affiche un contenu spécifique dans la zone Content.

Configuration de l'application

Créez une nouvelle application angular (`ng new book-store`). Sélectionnez oui pour ajouter un routage et utiliser CSS comme format de feuille de style.

La CLI crée un module AppRoutingModule (`src/app/app-routing.module.ts`) avec l'ajout du routage au projet. Nous spécifierons les routes pour l'application dans ce module.

Exécutez la commande `ng add @ng-bootstrap/ng-bootstrap` dans le dossier racine du projet pour ajouter Bootstrap au projet.

Routes

Ajoutez des routes au tableau Routes dans le module de routage App Routing comme suit.

```
1.  const routes: Routes = [
2.    {path: 'home', component: HomeComponent},
3.    {path: 'about', component: AboutComponent},
4.    {path: 'contact', component: ContactComponent},
5.    {path: 'books', component: BooksComponent},
6.    {path: '', redirectTo: 'home', pathMatch: 'full'},
7.    {path: '**', component: HomeComponent}];
```

Nous spécifions des routes vers

- HomeComponent avec le chemin *home*,
- AboutComponent avec le chemin *about*,
- ContactComponent avec le chemin *contact* et
- BooksComponent avec le chemin *books*.

De plus, une requête adressée à l'URL de base est redirigée vers *home* (ligne 6) ainsi que toutes les autres requêtes (ligne 7).
Générez les composants (ng generate component home, ng generate component about, ng generate component contact, ng generate component books).

Définissez les modèles HTML de HomeComponent, AboutComponent, ContactComponent en vous référant au code source sur Github. Assurez-vous de copier les images jpeg dans `src/assets`.

Routing Outlet

Modifiez le modèle HTML du composant App comme suit.

```
1.  <div class="container">
2.    <div class="nav-link">
3.      <a [routerLink]="['/home']"> Home </a>
4.      <a [routerLink]="['/about']"> About Us </a>
5.      <a [routerLink]="['/contact']"> Contact Us </a>
6.      <a [routerLink]="['/books']"> Books </a>
7.    </div>
8.  </div>
9.
10. <div class="container-fluid" >
11.   <div class="row">
12.     <div class="col">
```

```
13.       <img src="../assets/gobooks.jpg" height="150" width="175"/>
14.     </div>
15.     <div class="col-6">
16.       <router-outlet></router-outlet>
17.     </div>
18.     <div class="col">
19.       <img src="../assets/1books2-med.jpg" height="150" width="175"/>
20.     </div>
21.   </div>
22. </div>
```

Le modèle définit la barre de navigation (lignes 3 à 9) à l'aide de l'attribut routerLink pour associer des routes à des balises d'ancrage de liens.
Le router-outlet à la ligne 16, spécifie où une route est rendue lorsqu'elle est sélectionnée.

Style CSS

Modifiez le fichier de style du composant App (`src/app/app.component.css`) comme suit.

```
1.  a:link, a:visited {
2.    background-color: #0000b3;
3.    color: white;
4.    padding: 14px 25px;
5.    text-align: center;
6.    text-decoration: none;
7.    display: inline-block;
8.  }
9.
10. a:hover, a:active {
11.   background-color: #e60000;
12. }
```

Composant Books

Supposons que nous souhaitons afficher un formulaire de recherche lorsqu'un utilisateur accède à l'URL `/books` en cliquant sur le bouton Books. Le formulaire invite l'utilisateur à entrer un identifiant de livre et affiche les détails de ce livre lorsqu'il est trouvé.
Nous devons naviguer vers une route spécifique au livre sélectionné. Cela ne peut pas être fait de manière statique car il y a potentiellement un grand nombre de livres et plus de livres peuvent être ajoutés à la collection. Le routage d'angular offre la possibilité de créer des routes paramétrées pour ce scénario.

Route imbriquée

Nous devons d'abord spécifier la route des books comme une route imbriquée. Mettez à jour les routes dans le module de routage d'application comme suit.

```
1.  const booksRoutes: Routes = [
2.    {path: ':id', component: BookComponent}
```

```
3.  ];
4.
5.  const routes: Routes = [
6.    {path: 'home', component: HomeComponent},
7.    {path: 'about', component: AboutComponent},
8.    {path: 'contact', component: ContactComponent},
9.    {path: 'books', component: BooksComponent,
10.     children: booksRoutes
11.   },
12.   {path: '', redirectTo: 'home', pathMatch: 'full'},
13.   {path: '**', component: HomeComponent}
14. ];
```

Ensuite, nous définissons les sous-routes (booksRoutes) comme des routes paramétrées (lignes 1-3). L'identifiant de l'itinéraire est un paramètre correspondant à un identifiant de livre ajouté au chemin de l'itinéraire parent (`/books`). Ces chemins rendent le composant Book.

Nous avons ajouté une propriété children à la route books (ligne 10). Cela indique une route imbriqué par rapport au composant Books.

Créez le composant Book (ng generate component books/book) et importez-le dans le module de routage d'application App Routing.

Modèle HTML du composant Books

Modifiez le modèle HTML du composant Books comme suit.

```
1.  <div>
2.    <h2>Search our Book Collection</h2>
3.    <div>
4.      <label for="id">Book Id:</label>
5.      <input type="text" #bookQuery id="id">
6.      <button (click)="submit(bookQuery.value)">Search</button>
7.    </div>
8.    <div class="container">
9.      <router-outlet></router-outlet>
10.   </div>
11. </div>
```

Le modèle HTML du composant Books présente une entrée pour entrer un identifiant pour le livre recherché, à la ligne 5. Le bouton Search à la ligne 6, soumet l'identifiant saisi par l'utilisateur. La ligne 9 spécifie une balise router-outlet, où la navigation vers le livre recherché est rendue.

Classe de composants Books

Modifiez la classe du composant Books comme suit.

```
1.  import { Component, OnInit } from '@angular/core';
2.  import {ActivatedRoute, Router, Routes} from '@angular/router';
3.
4.  @Component({
```

```
5.    selector: 'app-books',
6.    templateUrl: './books.component.html',
7.    styleUrls: ['./books.component.css']
8.  })
9.  export class BooksComponent implements OnInit {
10.   constructor(private router: Router, private route: ActivatedRoute) { }
11.
12.   ngOnInit(): void {
13.   }
14.
15.   submit(value: string) {
16.     this.router.navigate(['./', value], {relativeTo: this.route});
17.   }
18. }
```

Le routeur (router) et la route activée (activated route) sont injectés au composant à la ligne 10. La route activée est un observable qui renvoie les paramètres de la route courante. La navigation vers l'itinéraire sélectionné est effectuée par programmation en utilisant la fonction de navigation du routeur à la ligne 16 de la fonction submit.

Modèle Book

Les informations relatives aux livres sont capturées comme une instance de la classe Book. Générez la classe (ng generate class books/model/book) et modifiez `src/app/books/model/book.ts` comme suit.

```
1.  export class Book {
2.    constructor(
3.      public id: number,
4.      public category: string,
5.      public title: string,
6.      public cost: number,
7.      public authors?: Author[],
8.      public year?: number,
9.      public description?: string
10.   ) {}
11. }
12.
13. export class Author {
14.   constructor(
15.     public firstName: string,
16.     public lastName: string
17.   ){}
18. }
```

Chaque livre est associé à une collection d'instances d'auteurs de la classe Author.

Service Book

Un service est utilisé pour fournir un accès à la collection de livres. Générez le service (ng generate service books/service/books) et modifiez `src/app/books/service/books.service.ts` comme suit.

```
1.  import { Injectable } from '@angular/core';
2.  import {Book} from '../model/book';
3.
4.  @Injectable({
5.    providedIn: 'root'
6.  })
7.  export class BooksService {
8.    private books = [
9.      new Book(1001, 'Tech', 'Introduction to Angular', 50.25, [new Author('Bob', 'T')], 2017),
10.     new Book(1002, 'Tech', 'Angular Advanced Concepts', 125.95, [new Author('Zorb', 'Tar')], 2019),
11.     new Book(1003, 'Kids', 'A Fantastic Story', 12.25,
12.           [new Author('Jane', 'C'), new Author('Tala', 'Tolo')], 2009),
13.     new Book(1004, 'Cook', 'The Best Shawarmas', 18.99, [new Author('Chef', 'Z')], 1978),
14.     new Book(1005, 'Tech', 'Angular Demystified', 210.50, [new Author('Zorb', 'Tar')], 2020)
15.   ];
16.
17.   constructor() {}
18.
19.   public getBook(id: string): Book {
20.     // tslint:disable-next-line:radix
21.     return <Book>this.books.find(book => book.id === Number.parseInt(id));
22.   }
23. }
```

Le service code en dur une collection d'instances Book et fournit une fonction qui renvoie un livre en fonction de son identifiant. Dans une application déployée, ce service interagirait avec un serveur pour obtenir les livres.

Composant Book

Modèle HTML

Modifiez le modèle HTML du composant Book (`src/books/book/book.component.html`) comme suit.

```
1.  <div *ngIf="!selectedBook">
2.    <h2>Sorry can't find the requested book...</h2>
3.  </div>
4.  <div *ngIf="selectedBook">
5.    <h2>Here are details of the Book {{selectedBook.id}}</h2>
6.    <div class="row">
```

```
7.      <div class="col-xs-3">Category:</div>
8.      <div class="col-xs-9">{{ selectedBook.category }}</div>
9.    </div>
10.   <div class="row">
11.     <div class="col-xs-3">Title:</div>
12.     <div class="col-xs-9">{{ selectedBook.title }}</div>
13.   </div>
14.   <div class="row">
15.     <div class="col-xs-3">Cost:</div>
16.     <div class="col-xs-9">{{ selectedBook.cost }}</div>
17.   </div>
18.   <div class="row" [hidden]="!selectedBook.authors">
19.     <div class="col-xs-3">Author:</div>
20.     <div class="col-xs-9"><span [innerHTML]="selectedBook.authors | authornames"></span></div>
21.   </div>
22.   <div class="row" [hidden]="!selectedBook.year">
23.     <div class="col-xs-3">Year:</div>
24.     <div class="col-xs-9">{{ selectedBook.year }}</div>
25.   </div>
26.   <div class="row" [hidden]="!selectedBook.description">
27.     <div class="col-xs-3">Description:</div>
28.     <div class="col-xs-9">{{ selectedBook.description }}</div>
29.   </div>
30. </div>
```

Pipe nom d'Author

Le modèle HTML du composant Book utilise un pipe à la ligne 20 pour afficher correctement les noms des auteurs. Générez le pipe avec la commande `ng generate pipe pipes/authornames` et éditez `src/app/pipes/authornames.pipe.ts` comme suit.

```
1.  import { Pipe, PipeTransform } from '@angular/core';
2.  import {Author} from '../books/model/book';
3.
4.  @Pipe({
5.    name: 'authornames'
6.  })
7.  export class AuthornamesPipe implements PipeTransform {
8.
9.    transform(value: Author[]): string {
10.     if (value == null) return '';
11.     return value.map((author) => `${author.firstName}, ${author.lastName}`).join(' <b>and</b> ');
12.   }
13.
14. }
```

Un pipe est créé en tant que classe décorée avec ***@Pipe*** qui implémente la fonction transform de l'interface PipeTransform. La fonction transform à la ligne 11, retourne une chaîne qui concatène chacun des firstName et lastName de l'auteur et joint avec *and*.

Class de Composant

Modifiez la classe de composants de Book (`src/books/book/book.component.ts`) comme suit.

```
1.  import { Component, OnInit } from '@angular/core';
2.  import {ActivatedRoute} from '@angular/router';
3.  import {Book} from '../model/book';
4.  import {BooksService} from '../service/books.service';
5.
6.  @Component({
7.    selector: 'app-book',
8.    templateUrl: './book.component.html',
9.    styleUrls: ['./book.component.css']
10. })
11. export class BookComponent implements OnInit {
12.   selectedBook!: Book;
13.
14.   constructor(private route: ActivatedRoute, private booksService: BooksService) {
15.   }
16.
17.   ngOnInit(): void {
18.     this.route.params.subscribe(params => {
19.       const id = params.id;
20.       this.selectedBook = this.booksService.getBook(id);
21.     });
22.   }
23.
24. }
```

La route activée (activated route) et le service booksService sont injectés à la ligne 14. Le composant s'abonne en tant qu'observateur à la route activé (ligne 18) afin de récupérer le paramètre d'itinéraire, l'identifiant du livre interrogé, puis le service de livres est utilisé pour rechercher un livre avec le identifiant (ligne 20).

Route authentifiée

Nous allons à présent ajouter une page d'administration pour permettre l'ajout de livres. Cette page doit cependant être accessible uniquement aux utilisateurs authentifiés. Par conséquent, nous utiliserons un route guard pour contrôler l'accès à la route de la page d'administration.

Routes

Modifiez le tableau de routes dans le module AppRouting (`src/app/app-routing.module.ts`) comme suit.

```
1.  const routes: Routes = [
2.    {path: 'home', component: HomeComponent},
3.    {path: 'about', component: AboutComponent},
4.    {path: 'contact', component: ContactComponent},
5.    { path: 'login', component: LoginComponent },
6.    {
7.      path: 'admin',
8.      component: AdminComponent,
9.      canActivate: [ LoggedInGuard ]
10.   },
11.   {path: 'books', component: BooksComponent,
12.     children: booksRoutes
13.   },
14.   {path: '', redirectTo: 'home', pathMatch: 'full'},
15.   {path: '**', component: HomeComponent}
16. ];
```

Nous avons ajouté un chemin *login* qui rend LoginComponent (ligne 5) et un chemin *admin* qui rend AdminComponent. La propriété canActivate du chemin *admin* est définie comme un tableau avec une garde.
Générez le composant Login (ng generate component login), le composant Admin (ng generate component admin) et le LoggedIn Guard (ng generate guard logged-in) avec la méthode CanActivate. Assurez-vous d'importer les composants dans le module AppRouting.

Modifiez le modèle HTML du composant App comme suit.

```
1.  <div class="container">
2.    <p>Main Page</p>
3.    <div class="nav-link">
4.      <a [routerLink]="['/home']"> Home </a>
5.      <a [routerLink]="['/about']"> About Us </a>
6.      <a [routerLink]="['/contact']"> Contact Us </a>
7.      <a [routerLink]="['/books']"> Books </a>
8.      <a [routerLink]="['/login']"> Login </a>
9.      <a [routerLink]="['/admin']"> Admin </a>
10.   </div>
11. </div>
12.
13. <div class="container-fluid" >
14.   <div class="row">
15.     <div class="col">
16.       <img src="../assets/gobooks.jpg" height="150" width="175"/>
17.     </div>
18.     <div class="col-6">
19.       <router-outlet></router-outlet>
20.     </div>
21.     <div class="col">
22.       <img src="../assets/1books2-med.jpg" height="150" width="175"/>
```

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

Nous avons ajouté des liens pour les chemins *admin* et *login*.

Service Authentication

Nous allons créer un service pour encapsuler la vérification de l'authentification des utilisateurs. Cette première version est basique mais avoir un service séparé aidera si/lorsque nous décidons d'utiliser une approche d'authentification plus sophistiquée car les changements seront contenus.

Générer un service Authentication (`ng generate service authentication`). Modifiez la classe AuthenticationService (`src/app/authentication.service.ts`) comme suit.

```
import { Injectable } from '@angular/core';
import {Router} from '@angular/router';
import {noop} from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class AuthenticationService {
  redirectUrl: string | null | undefined;

  constructor(private router: Router) {}

  login(user: string, password: string): boolean {
    // hard coded for now
    if (user === 'admin' && password === 'password') {
      sessionStorage.setItem('username', user);
      if (this.redirectUrl) { this.router.navigate([this.redirectUrl]).then(noop); }
      this.redirectUrl = null;
      return true;
    }
    return false;
  }

  logout(): any {
    sessionStorage.removeItem('username');
  }

  getUser(): any {
    return sessionStorage.getItem('username');
  }

  isLoggedIn(): boolean {
    return this.getUser() !== null;
  }
}
```

AuthenticationService est utilisé pour la connexion, la déconnexion et pour vérifier qu'un utilisateur est authentifié. La fonction login aux lignes 13-22 (que nous utiliserons dans le composant Login) prend un id et un mot de passe (password) d'utilisateur et vérifie les informations d'identification de l'utilisateur. Des informations d'identification valides codées en dur sont utilisées ici. Dans une application de production, cela se ferait typiquement par une recherche d'utilisateurs stockés sur un serveur. Si les informations d'identification saisies sont valides, nous stockons l'utilisateur dans le sessionStorage du navigateur.

Nous voulons être amenés ici chaque fois qu'un utilisateur accède à un chemin protégé. AuthenticationService «se souvient» de l'URL dans la propriété redirectUrl et utilise le routeur (router) (injecté à la ligne 11) pour rediriger vers cette URL.

La fonction de déconnexion logout (lignes 24-26) efface simplement sessionStorage tandis que la fonction isLoggedIn (lignes 32-34) vérifie la présence d'un utilisateur authentifié dans sessionStorage à l'aide de la fonction getUser (lignes 28-30). isLoggedIn retourne true si un utilisateur authentifié est trouvé dans sessionStorage, false dans le cas contraire.

Garde Logged In

Nous pouvons maintenant compléter la Garde LoggedIn. Modifiez la classe LoggedInGuard (`src/app/logged-in.guard.ts`) comme suit.

```
1.  import { Injectable } from '@angular/core';
2.  import {CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree, Router} from
    '@angular/router';
3.  import { Observable } from 'rxjs';
4.  import {AuthenticationService} from './authentication.service';
5.
6.  @Injectable({
7.    providedIn: 'root'
8.  })
9.  export class LoggedInGuard implements CanActivate {
10.   constructor(private authService: AuthenticationService, private router: Router) {}
11.
12.   canActivate(
13.     next: ActivatedRouteSnapshot,
14.     state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean
    | UrlTree {
15.     // return  this.authService.isLoggedIn();
16.     if (this.authService.isLoggedIn()) {return true; }
17.     this.authService.redirectUrl = state.url;
18.     this.router.navigate(['./login']);
19.     return false;
20.   }
21. }
```

La garde est un *injectable* qui implémente la fonction canActivate (lignes 12-20) de l'interface CanActivate. La fonction retourne true si une route protégée peut être activée, false dans le cas contraire. À la ligne 16, le service Authentication, injecté à la ligne 10, est utilisé pour vérifier que l'utilisateur de la session en cours est authentifié. Sinon, l'URL actuelle est stockée en tant que redirectUrl dans le service Authentication et le routeur, injecté à la ligne 10, redirige vers le chemin *login*.

Composant Login

Le composant Login obtient les informations d'identification d'un utilisateur et les vérifie à l'aide du service Authentication.

Modèle HTML du composant Login

Modifier le modèle HTML du composant Login (`src/app/login/login.component.html`) comme suit.

```
1.  <div class="alert alert-danger" role="alert" *ngIf="message">
2.    {{ message }}
3.  </div>
4.
5.  <div class="container" *ngIf="!isLoggedIn">
6.    <div>
7.      User Name : <input type="text" name="username" [(ngModel)]="username">
8.      Password : <input type="password" name="password" [(ngModel)]="password">
9.    </div>
10.   <button (click)=checkLogin() class="btn btn-success">
11.     Login
12.   </button>
13. </div>
14. <div class="well" *ngIf="isLoggedIn">
15.   <p>Logged in as <b>{{ loggedUser }}</b></p>
16.   <button (click)=logout() class="btn btn-danger">
17.     Log out
18.   </button>
19. </div>
```

Le composant présente des champs de saisie pour le nom d'utilisateur et le mot de passe (lignes 7 à 8) et les lie aux propriétés de la classe de composant. Le bouton Login à la ligne 10 déclenche alors la vérification des informations d'identification lorsque on clic dessus en appelant la fonction checkLogin.
Si un utilisateur est connecté, un autre bouton Log out s'affiche (ligne 16) et un clic dessus déclenche la déconnexion de l'utilisateur.

Classe de Composant Login

Modifiez la classe du composant Login (`src/app/login/login.component.ts`) comme suit.

```
1.  import { Component, OnInit } from '@angular/core';
2.  import {AuthenticationService} from '../authentication.service';
```

```
3.  import {Router} from '@angular/router';
4.
5.  @Component({
6.    selector: 'app-login',
7.    templateUrl: './login.component.html',
8.    styleUrls: ['./login.component.css']
9.  })
10. export class LoginComponent {
11.   username = '';
12.   password = '';
13.   message!: string;
14.
15.   constructor(private router: Router,
16.               private loginService: AuthenticationService) { }
17.
18.   get isLoggedIn(): boolean {
19.     return this.loginService.isLoggedIn();
20.   }
21.
22.   get loggedUser(): string {
23.     return this.loginService.getUser();
24.   }
25.
26.   checkLogin(): boolean {
27.     this.message = '';
28.     if (!this.loginService.login(this.username, this.password)) {
29.       this.message = 'Invalid Login';
30.       setTimeout(() => {
31.         this.message = '';
32.       }, 2500);
33.       return false;
34.     }
35.     return true;
36.   }
37.
38.   logout(): boolean {
39.     this.loginService.logout();
40.     return true;
41.   }
42. }
```

Le routeur (Router) et le service Authentication sont injectés dans le constructeur (lignes 15-16). Dans la fonction checkLogin, nous utilisons le service Authentication pour vérifier les informations d'identification saisies (ligne 28). Si la connexion de l'utilisateur est refusée, nous définissons un message et utilisons un minuteur pour qu'il s'affiche pendant une durée donnée (lignes 28-33).

Composant Admin

Nous offrons maintenant la possibilité d'ajouter plus de livres à la librairie dans le composant Admin. Nous utilisons également des formulaires réactifs et dynamiques et la validation de formulaire.

Classe de Composant Admin

Modifiez la classe du composant Admin (`src/app/admin/admin.component.ts`) comme suit.

```
1.  import { Component, OnInit } from '@angular/core';
2.  import {AbstractControl, FormArray, FormBuilder, FormControl, Validators} from '@angular/forms';
3.  import {Author, Book} from '../books/model/book';
4.  import {BooksService} from '../books/service/books.service';
5.
6.  function categoryValidator(control: FormControl<string>): { [s: string]: boolean } | null {
7.    const validCategories = ['Kids', 'Tech', 'Cook'];
8.    if (!validCategories.includes(control.value)) {
9.      return {invalidCategory: true};
10.   }
11.   return null;
12. }
13.
14. @Component({
15.   selector: 'app-admin',
16.   templateUrl: './admin.component.html',
17.   styleUrls: ['./admin.component.css']
18. })
19. export class AdminComponent implements OnInit {
20.   bookForm = this.builder.group({
21.     id: ['', [Validators.required, Validators.pattern('[1-9]\\d{3}')]],
22.     category: ['', [Validators.required, categoryValidator]],
23.     title: ['', Validators.required],
24.     cost: ['', [Validators.required, Validators.pattern('\\d+(\\.\\d{1,2})?')] ],
25.     authors: this.builder.array([]),
26.     year: [''],
27.     description: ['']
28.   });
29.
30.   get id(): AbstractControl<string> {return <AbstractControl>this.bookForm.get('id'); }
31.   get category(): AbstractControl<string> {return <AbstractControl>this.bookForm.get('category'); }
32.   get title(): AbstractControl<string> {return <AbstractControl>this.bookForm.get('title'); }
33.   get cost(): AbstractControl<string> {return <AbstractControl>this.bookForm.get('cost'); }
34.   get authors(): FormArray {
35.     return this.bookForm.get('authors') as FormArray;
36.   }
37.
38.   constructor(private builder: FormBuilder,
39.               private booksService: BooksService) { }
40.
41.   ngOnInit(): void {
```

```
42.   }
43.
44.   onSubmit(): void {
45.     const book =  new Book(Number(this.bookForm.value.id),
46.       <string>this.bookForm.value.category,
47.       <string>this.bookForm.value.title,
48.       Number(this.bookForm.value.cost),
49.       <Author[]>this.bookForm.value.authors,
50.       Number(this.bookForm.value.year),
51.       <string>this.bookForm.value.description);
52.     this.booksService.addBook(book);
53.     this.bookForm.reset();
54.     this.authors.clear();
55.   }
56.
57.   addAuthor(): void {
58.     this.authors.push(
59.       this.builder.group({
60.         firstName: [''],
61.         lastName: ['']
62.       })
63.     );
64.   }
65.
66.   removeAuthor(i: number): void {
67.     this.authors.removeAt(i);
68.   }
69. }
```

Le composant Admin utilise un formulaire réactif (Reactive Form). Assurez-vous d'importer FormsModule et ReactiveFormsModule dans AppModule. Ces modules doivent être spécifiés dans le tableau des imports du décorateur @NgModule.

Le formulaire bookForm, est défini aux lignes 20-28 à l'aide du service FormBuilder injecté dans le constructeur (ligne 38). Le builder crée un groupe de formulaire (*form group*) dans lequel les champs de formulaires (*form controls*) sont déclarés en tant que paires valeurs propriété. Chaque champ de formulaire peut être associé à une liste de validateurs. Par exemple, les validateurs de l'identifiant sont ***Validators.required*** et ***Validators.pattern('[1-9]\\d{3}')***. Le premier validateur échoue si une valeur n'est pas fournie pour le contrôle tandis que le second assure que la valeur fournie satisfait une expression régulière. categoryValidator utilisé pour *category* à la ligne 22, est un validateur personnalisé défini aux lignes 6-12. Il vérifie que la valeur fournie se trouve dans une liste de catégories valides.

Le contrôle *authors* est un FormArray qui permet la création dynamique de contrôles de formulaires. Nous utilisons ceci pour qu'un nombre variable de noms d'auteurs puisse être capturé. La fonction

addAuthor aux lignes 57-64, ajoute un auteur tandis que la fonction removeAuthor aux lignes 66-68 supprime un auteur du *form array*.

Nous spécifions des getters aux lignes 30-36 pour faire référence aux contrôles de formulaire plus facilement dans le modèle HTML.

La fonction onSubmit (lignes 44-55) est le gestionnaire de soumission du formulaire. Il crée une instance de la classe du modèle Book à partir des valeurs des contrôles du formulaire et appelle le service Books (injecté à la ligne 39), pour ajouter la nouvelle instance à la collection de livres.

Ajoutez la fonction addBook à la classe BookService (`src/app/books/service/books.service.ts`).

```
1.  public addBook(b: Book): void {
2.      this.books.push(b);
3.  }
```

La fonction ajoute un livre à la collection de livres.

Modèle HTML du composant Admin

Modifier le modèle HTML du composant Admin (`src/app/admin/admin.component.html`) comme suit.

```
1.  <div class="container">
2.    <h1>Book Form</h1>
3.    <button (click)="addAuthor()">Add Author</button>
4.    <form [formGroup]="bookForm" (ngSubmit)="onSubmit()">
5.      <div class="form-group">
6.        <label for="id">Id:</label>
7.        <input type="text" class="form-control" id="id"  formControlName="id" required>
8.        <div [hidden]="id.pristine || id.valid"
9.             class="alert alert-danger">
10.         Id must be 4 digits long starting with a number different from 0.
11.       </div>
12.     </div>
13.     <div class="form-group">
14.       <label for="category">Category:</label>
15.       <input type="text" class="form-control" id="category"  formControlName="category">
16.       <div [hidden]="category.pristine || category.valid"
17.            class="alert alert-danger">
18.         Category is required to be <b>Kids</b>, <b>Tech</b> or <b>Cook</b>
19.       </div>
20.     </div>
21.     <div class="form-group">
22.       <label for="title">Title:</label>
23.       <input type="text" class="form-control" id="title" required formControlName="title">
24.       <div [hidden]="title.pristine || title.valid"
25.            class="alert alert-danger">
26.         Title is required.
```

```
      </div>
    </div>
    <div class="form-group">
      <label for="cost">Cost:</label>
      <input type="text" class="form-control" id="cost" required  pattern="\d+(.\d{1,2})?"
  formControlName="cost">
      <div [hidden]="cost.pristine || cost.valid"
          class="alert alert-danger">
        Cost should be a number with two optional decimals
      </div>
    </div>
    <div class="form-group">
      <div formArrayName="authors">
        <div *ngFor="let _ of authors.controls; let i=index">
          <ng-container [formGroupName]="i">
            <label>
              Author First Name:
              <input formControlName="firstName" type="text">
            </label>
            <label>
              Author Last Name:
              <input formControlName="lastName" type="text">
            </label>
            <button class="btn btn-dark" (click)="removeAuthor(i)">X</button>
          </ng-container>
        </div>
      </div>
    </div>
    <div class="form-group">
      <label for="year">Year:</label>
      <input type="text" class="form-control" id="year" formControlName="year">
    </div>
    <div class="form-group">
      <label for="description">Description:</label>
      <textarea cols="40" class="form-control" id="description"
  formControlName="description"></textarea>
    </div>
    <button type="submit" class="btn btn-success" [disabled]="bookForm.invalid">Submit</button>
  </form>
</div>
```

Le modèle HTML présente le formulaire défini dans la classe du composant. Les erreurs de validation sont affichées dans un *div* qui est masqué lorsqu'un contrôle de formulaire est valide (*valid*) ou vierge (*pristine* - sa valeur n'a pas encore été modifiée).

Exercise

L'exercice suivant est le livrable pour le laboratoire. Complétez et enregistrez le code dans Github Classroom avant la date limite. Seul le travail de ce exercice sera évalué.

Implémentez une application Angular à l'aide d'un formulaire réactif. L'application présentera un formulaire dans lequel les utilisateurs peuvent soumettre les informations suivantes:

- Prénom
- Nom de famille
- Numéro de téléphone
- Email

Une validation doit être effectuée pour vérifier que les valeurs saisies satisfont aux contraintes suivantes.

- Le prénom et le nom sont obligatoires.
- Le numéro de téléphone, lorsqu'il est fourni, doit comporter 10 chiffres avec les premier et quatrième chiffres différents de 0.
- L'e-mail doit être conforme à un modèle d'e-mail valide (utilisez Angular Validators.email ()).

L'application affichera les données d’utilisateur saisies dans une table lorsque des informations valides seront soumises. Des messages d’erreurs appropriés seront affichés au cas où les entrées sont invalides.