import {

  Component,

  Input,

  OnChanges,

  OnInit,

  SimpleChanges,

} from "@angular/core";

import { Observable } from "rxjs";

import { Property } from "../property";

@Component({

  selector: "property-list",

  templateUrl: "./property-list.component.html",

  styleUrls: ["./property-list.component.scss"],

})

export class PropertyListComponent  {

  @Input() propertyList!: Observable<Property[]>;

}

<div \*ngIf="propertyList | async as properties">

      <div \*ngFor="let property of properties">

        <h3>{{ property.name }}</h3>

        <p>{{ property.description }}</p>

      </div>

    </div>

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

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Property } from '../property';

import { environment } from 'src/environments/environment';

@Injectable({

  providedIn: 'root',

})

export class PropertyService {

  private apiUrl = `${environment.apiUrl}/properties`;

  constructor(private http: HttpClient) {}

  getProperties(): Observable<Property[]> {

    return this.http.get<Property[]>(this.apiUrl);

  }

  getProperty(id: string): Observable<Property> {

    return this.http.get<Property>(`${this.apiUrl}/${id}`);

  }

  addProperty(property: Property): Observable<any> {

    return this.http.post(this.apiUrl, property);

  }

}

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

import { Property } from './property';

import { PropertyService } from './services/property.service';

import { BehaviorSubject, combineLatest, Observable, of } from 'rxjs';

import { map } from 'rxjs/operators';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

})

export class AppComponent implements OnInit {

  property: Property = this.getDefaultProperty();

  properties$!: Observable<Property[]>;

  filteredProperties$!: Observable<Property[]>;

  private searchTerm$ = new BehaviorSubject<string>('');

  constructor(private propertyService: PropertyService) {}

  ngOnInit(): void {

    this.properties$ = this.propertyService.getProperties();

    this.filteredProperties$ = combineLatest([

      this.properties$,

      this.searchTerm$,

    ]).pipe(

      map(([properties, term]) =>

        properties.filter((p) =>

          p.name.toLowerCase().includes(term.toLowerCase())

        )

      )

    );

  }

  getDefaultProperty(): Property {

    return {

      id: '',

      name: '',

      location: '',

      price: 0,

      rooms: 1,

      type: 'Apartment',

      size: 0,

    };

  }

  addProperty(): void {

    this.propertyService.addProperty(this.property).subscribe(() => {

      this.property = this.getDefaultProperty();

      this.ngOnInit(); // Refresh the list

    });

  }

  searchProperties(event: any): void {

    const term = event.target.value || '';

    this.searchTerm$.next(term);

  }

}

<div>

  <h1>Property Management</h1>

  <input

    type="text"

    placeholder="Search properties"

    (input)="searchProperties($event)"

  />

  <form (ngSubmit)="addProperty()">

    <input [(ngModel)]="property.name" name="name" placeholder="Name" required />

    <input [(ngModel)]="property.location" name="location" placeholder="Location" required />

    <input [(ngModel)]="property.price" name="price" type="number" placeholder="Price" required />

    <input [(ngModel)]="property.rooms" name="rooms" type="number" placeholder="Rooms" required />

    <input [(ngModel)]="property.size" name="size" type="number" placeholder="Size" required />

    <select [(ngModel)]="property.type" name="type">

      <option value="Apartment">Apartment</option>

      <option value="Villa">Villa</option>

    </select>

    <button type="submit">Add Property</button>

  </form>

  <app-property-list [propertyList]="filteredProperties$"></app-property-list>

</div>

export interface Book {

  id: number;

  title: string;

  author: string;

  genre: string;

  publication\_date: string;

  price: number;

}

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

import { Observable, of } from 'rxjs';

@Injectable({

  providedIn: 'root'

})

export class BookService {

  private books: Book[] = [

    { id: 1, title: 'A', author: 'B', genre: 'C', publication\_date: '2020-01-01', price: 10 }

  ];

  getBooks(): Observable<Book[]> {

    return of(this.books);

  }

  filterBooks(genre: string): Book[] {

    return this.books.filter(book => book.genre === genre);

  }

}

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

import { CommonModule } from '@angular/common';

import { BookService, Book } from '../book.service';

@Component({

  selector: 'app-book-list',

  templateUrl: './book-list.component.html',

  styleUrls: ['./book-list.component.css'],

  imports: [CommonModule]

})

export class BookListComponent implements OnInit {

  books: Book[] = [];

  filteredBooks: Book[] = [];

  constructor(private bookService: BookService) {}

  ngOnInit(): void {

    this.bookService.getBooks().subscribe(data => {

      this.books = data;

      this.filteredBooks = data;

    });

  }

  onFilter(genre: string): void {

    this.filteredBooks = this.bookService.filterBooks(genre);

  }

}

import { Component, EventEmitter, Output } from '@angular/core';

import { FormsModule } from '@angular/forms';

@Component({

  selector: 'app-book-filter',

  templateUrl: './book-filter.component.html',

  styleUrls: ['./book-filter.component.css'],

  imports: [FormsModule]

})

export class BookFilterComponent {

    genre?: string;

    minPrice?: number;

    maxPrice?: number;

    author?: string;

    publicationDate?: string;

    @Output() filter = new EventEmitter<{

      genre?: string;

      minPrice?: number;

      maxPrice?: number;

      author?: string;

      publicationDate?: string;

    }>();

    applyFilter(): void {

      this.filter.emit({

        genre: this.genre,

        minPrice: this.minPrice,

        maxPrice: this.maxPrice,

        author: this.author,

        publicationDate: this.publicationDate

      });

    }

}

export class AppComponent {

  title = 'My Book App';

}

BOOK-SERVICE.TS

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

import { HttpClient } from '@angular/common/http';

import { Book } from './models/book.model';

import { Observable, map } from 'rxjs';

import { environment } from 'src/environments/environment.development';

@Injectable({

providedIn: 'root',

})

export class BookService {

private apiUrl = environment.apiUrl;

constructor(private http: HttpClient) {}

getBooks(): Observable<Book[]> {

return this.http.get<Book[]>(this.apiUrl);

}

getBookByIsbn(isbn: string): Observable<Book | undefined> {

return this.http

.get<Book[]>(`${this.apiUrl}?isbn=${isbn}`)

.pipe(map((books) => books[0]));

}

addBook(book: Book): Observable<Book> {

return this.http.post<Book>(this.apiUrl, book);

}

}

VIEW-BOOK-COMPONENT.TS

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

import { Book } from 'src/app/models/book.model';

import { BookService } from 'src/app/book.service';

@Component({

selector: 'app-view-book',

templateUrl: './view-book.component.html',

})

export class ViewBookComponent {

isbn: string = '';

book?: Book;

errorMessage: string | null = null;

constructor(private bookService: BookService) {}

fetchBook() {

this.errorMessage = null;

this.bookService.getBookByIsbn(this.isbn).subscribe({

next: (book) => {

if (book) {

this.book = book;

} else {

this.book = undefined;

this.errorMessage = 'Book not found';

}

},

error: () => {

this.book = undefined;

this.errorMessage = 'Error fetching book';

},

});

}

}

ADD-BOOK-COMPONENT.TS

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

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { BookService } from 'src/app/book.service';

import { NewBook } from 'src/app/models/book.model';

@Component({

selector: 'app-add-book',

templateUrl: './add-book.component.html',

})

export class AddBookComponent {

bookForm: FormGroup;

constructor(private fb: FormBuilder, private bookService: BookService) {

this.bookForm = this.fb.group({

isbn: ['', [Validators.required, Validators.pattern(/^97[89]-\d{10}$/)]],

title: ['', Validators.required],

author: ['', Validators.required],

genre: ['', Validators.required],

price: [null, [Validators.required, Validators.min(0)]],

pages: [null, [Validators.required, Validators.min(1)]],

});

}

onSubmit() {

if (this.bookForm.valid) {

const newBook: NewBook = this.bookForm.value;

this.bookService.addBook(newBook).subscribe({

next: () => {

alert('Book added successfully!');

this.bookForm.reset();

},

error: () => {

alert('Failed to add book.');

},

});

}

}

}

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

import { Employee } from '../models/employee.model';

@Injectable({

providedIn: 'root'

})

export class EmployeeService {

private employees: Employee[] = [];

constructor() {}

getEmployees(): Employee[] {

return [...this.employees]; // return a copy to avoid external mutation

}

addEmployee(employee: Employee): void {

this.employees.push(employee);

}

}

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

import { Router } from '@angular/router';

@Component({

selector: 'app-product-filter',

template: `

<form (ngSubmit)="applyFilters()">

<input [(ngModel)]="category" name="category" placeholder="Category" />

<input [(ngModel)]="price" name="price" type="number" placeholder="Price" />

<input [(ngModel)]="rating" name="rating" type="number" step="0.1" placeholder="Rating" />

<button type="submit">Apply Filters</button>

</form>

`

})

export class ProductFilterComponent {

category: string = '';

price?: number;

rating?: number;

constructor(private router: Router) {}

applyFilters(): void {

const queryParams: any = {};

if (this.category) queryParams.category = this.category;

if (this.price !== undefined) queryParams.price = this.price;

if (this.rating !== undefined) queryParams.rating = this.rating;

this.router.navigate(['/products'], { queryParams });

}

}

export interface Book {

  id?: number;

  title: string;

  author: string;

  genre: string;

}

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

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Book } from './book';

import { environment } from 'src/environments/environment';

@Injectable({

  providedIn: 'root'

})

export class BookService {

  private apiUrl = `${environment.apiUrl}/books`;

  constructor(private http: HttpClient) {}

  getBooks(): Observable<Book[]> {

    return this.http.get<Book[]>(this.apiUrl);

  }

  addBook(book: Book): Observable<Book> {

    return this.http.post<Book>(this.apiUrl, book);

  }

 updateBook(book: Book): Observable<Book> {

  return this.http.put<Book>(`${this.apiUrl}/${book.id}`, book);

}

deleteBook(id: number): Observable<void> {

  return this.http.delete<void>(`${this.apiUrl}/${id}`);

}

}

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

import { FormBuilder, FormGroup } from '@angular/forms';

import { BookService } from '../book.service';

import { Book } from '../book';

@Component({

  selector: 'app-book',

  templateUrl: './book.component.html',

  styleUrls: ['./book.component.css']

})

export class BookComponent implements OnInit {

  books: Book[] = [];

  bookForm: FormGroup;

  isEditMode = false;

  currentBookId: number | null = null;

  constructor(private fb: FormBuilder, private bookService: BookService) {

    this.bookForm = this.fb.group({

      title: [null],

      author: [null],

      genre: [null]

    });

  }

  ngOnInit(): void {

    this.loadBooks();

  }

  loadBooks(): void {

    this.bookService.getBooks().subscribe((data: Book[]) => {

      this.books = data;

    });

  }

  addOrUpdateBook(): void {

    const formValue = this.bookForm.value;

    if (this.isEditMode && this.currentBookId !== null) {

      const updatedBook: Book = {

        id: this.currentBookId,

        ...formValue

      };

      this.bookService.updateBook(updatedBook).subscribe(() => {

        this.loadBooks();

        this.resetForm();

      });

    } else {

      this.bookService.addBook(formValue).subscribe(() => {

        this.loadBooks();

        this.resetForm();

      });

    }

  }

  editBook(book: Book): void {

    this.isEditMode = true;

    this.currentBookId = book.id!;

    this.bookForm.setValue({

      title: book.title,

      author: book.author,

      genre: book.genre

    });

  }

  deleteBook(id: number): void {

    this.bookService.deleteBook(id).subscribe(() => {

      this.loadBooks();

    });

  }

  resetForm(): void {

    this.bookForm.reset();

    this.isEditMode = false;

    this.currentBookId = null;

  }

}

export interface Book {

  id?: number;

  title: string;

  author: string;

  genre: string;

}

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

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Book } from './book';

import { environment } from 'src/environments/environment';

@Injectable({

  providedIn: 'root'

})

export class BookService {

  private apiUrl = `${environment.apiUrl}/books`;

  constructor(private http: HttpClient) {}

  getBooks(): Observable<Book[]> {

    return this.http.get<Book[]>(this.apiUrl);

  }

  addBook(book: Book): Observable<Book> {

    return this.http.post<Book>(this.apiUrl, book);

  }

  updateBook(book: Book): Observable<Book> {

    return this.http.put<Book>(`${this.apiUrl}/${book.id}`, book);

  }

  deleteBook(id: number): Observable<void> {

    return this.http.delete<void>(`${this.apiUrl}/${id}`);

  }

}

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

import { FormBuilder, FormGroup } from '@angular/forms';

import { BookService } from '../book.service';

import { Book } from '../book';

@Component({

  selector: 'app-book',

  templateUrl: './book.component.html'

})

export class BookComponent implements OnInit {

  books: Book[] = [];

  bookForm: FormGroup;

  isEditMode = false;

  currentBookId: number | null = null;

  constructor(private fb: FormBuilder, private bookService: BookService) {

    this.bookForm = this.fb.group({

      title: [null],

      author: [null],

      genre: [null]

    });

  }

  ngOnInit(): void {

    this.loadBooks();

  }

  loadBooks(): void {

    const books$ = this.bookService.getBooks();

    if (books$) {

      books$.subscribe((books) => {

        this.books = books;

      });

    }

  }

  addOrUpdateBook(): void {

    const bookData = this.bookForm.value;

    if (this.isEditMode && this.currentBookId !== null) {

      const updatedBook: Book = { id: this.currentBookId, ...bookData };

      this.bookService.updateBook(updatedBook).subscribe(() => {

        this.loadBooks();

        this.resetForm();

      });

    } else {

      this.bookService.addBook(bookData).subscribe(() => {

        this.loadBooks();

        this.resetForm();

      });

    }

  }

  editBook(book: Book): void {

    this.isEditMode = true;

    this.currentBookId = book.id!;

    this.bookForm.setValue({

      title: book.title,

      author: book.author,

      genre: book.genre

    });

  }

  deleteBook(id: number): void {

    this.bookService.deleteBook(id).subscribe(() => {

      this.loadBooks();

    });

  }

  resetForm(): void {

    this.bookForm.reset();

    this.isEditMode = false;

    this.currentBookId = null;

  }

}

<form [formGroup]="bookForm" (ngSubmit)="addOrUpdateBook()">

    <input formControlName="title" placeholder="Title" />

    <input formControlName="author" placeholder="Author" />

    <input formControlName="genre" placeholder="Genre" />

    <button type="submit">{{ isEditMode ? 'Update' : 'Add' }} Book</button>

    <button type="button" (click)="resetForm()">Reset</button>

  </form>

  <ul>

    <li \*ngFor="let book of books">

      {{ book.title }} by {{ book.author }} ({{ book.genre }})

      <button (click)="editBook(book)">Edit</button>

      <button (click)="deleteBook(book.id!)">Delete</button>

    </li>

  </ul>

export interface Item {

    id: string;

    name: string;

    description: string;

    price: number;

  }

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

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Item } from './item';

@Injectable({

  providedIn: 'root'

})

export class ItemService {

  private baseUrl = '/api/items';

  constructor(private http: HttpClient) {}

  getItems(search: string, sortBy: string): Observable<Item[]> {

    return this.http.get<Item[]>(`${this.baseUrl}?search=${search}&sortBy=${sortBy}`);

  }

  getItemById(id: string): Observable<Item> {

    return this.http.get<Item>(`${this.baseUrl}/${id}`);

  }

  createItem(item: Item): Observable<any> {

    return this.http.post(`${this.baseUrl}`, item);

  }

  updateItem(item: Item): Observable<any> {

    return this.http.put(`${this.baseUrl}/${item.id}`, item);

  }

  deleteItem(id: string): Observable<any> {

  return this.http.delete(`${this.baseUrl}/${id}`);

  }

}

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

import { Item } from '../item';

import { ItemService } from '../item.service';

import { ActivatedRoute, Router } from '@angular/router';

@Component({

  selector: 'app-item-form',

  templateUrl: './item-form.component.html'

})

export class ItemFormComponent implements OnInit {

  item: Item = { id: '', name: '', description: '', price: 0 };

  isEdit = false;

  validationErrors: any = {};

  constructor(

    private itemService: ItemService,

    private route: ActivatedRoute,

    private router: Router

  ) {}

  ngOnInit(): void {

    const id = this.route.snapshot.paramMap.get('id');

    if (id) {

      this.isEdit = true;

      this.itemService.getItemById(id).subscribe(item => {

        this.item = item;

      });

    }

  }

  validate(): boolean {

    this.validationErrors = {};

    if (!this.item.name.trim()) {

      this.validationErrors.name = 'Name is required.';

    }

    if (!this.item.description.trim()) {

      this.validationErrors.description = 'Description is required.';

    }

    if (this.item.price <= 0) {

      this.validationErrors.price = 'Price must be greater than 0.';

    }

    return Object.keys(this.validationErrors).length === 0;

  }

  saveItem(): void {

    if (!this.validate()) return;

    const action = this.isEdit

      ? this.itemService.updateItem(this.item)

      : this.itemService.createItem(this.item);

    action.subscribe(() => {

      this.router.navigate(['/items']);

    });

  }

}

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

import { Item } from '../item';

import { ItemService } from '../item.service';

import { Router } from '@angular/router';

@Component({

  selector: 'app-item-list',

  templateUrl: './item-list.component.html'

})

export class ItemListComponent implements OnInit {

  items: Item[] = [];

  sortBy: keyof Item = 'id';

  constructor(private itemService: ItemService, private router: Router) {}

  ngOnInit(): void {

    this.getItems();

  }

  getItems(): void {

    this.itemService.getItems('', this.sortBy).subscribe(items => {

      this.items = items;

      this.sortItems();

    });

  }

  editItem(item: Item): void {

    this.router.navigate(['/form', { id: item.id, hideId: true }]);

  }

  deleteItem(item: Item): void {

    if (confirm('Are you sure you want to delete this item?')) {

    this.itemService.deleteItem(item.id).subscribe(() => {

        this.getItems();

      });

    }

  }

  changeSort(field: keyof Item): void {

    this.sortBy = field;

    this.sortItems();

  }

  private sortItems(): void {

    this.items.sort((a, b) => {

      const valA = a[this.sortBy];

      const valB = b[this.sortBy];

      if (typeof valA === 'number' && typeof valB === 'number') {

        return valA - valB;

      }

      return String(valA).localeCompare(String(valB));

    });

  }

}

export interface Student {

  id?: string;

  name: string;

  course: string;

  grade: number;

}

// student.service.ts

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

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Student } from './student';

@Injectable({

  providedIn: 'root'

})

export class StudentService {

  private baseUrl = '/api/students'; // Adjust as needed

  constructor(private http: HttpClient) {}

  getStudents(filter: string, sortBy: string): Observable<Student[]> {

    return this.http.get<Student[]>(`${this.baseUrl}?filter=${filter}&sortBy=${sortBy}`);

  }

  getStudentById(id: string): Observable<Student> {

    return this.http.get<Student>(`${this.baseUrl}/${id}`);

  }

  createStudent(student: Student): Observable<Student> {

    return this.http.post<Student>(this.baseUrl, student);

  }

  updateStudent(student: Student): Observable<Student> {

    return this.http.put<Student>(`${this.baseUrl}/${student.id}`, student);

  }

  deleteStudent(id: string): Observable<any> {

    return this.http.delete(`${this.baseUrl}/${id}`);

  }

}

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

import { CommonModule } from '@angular/common'; // Import CommonModule

import { StudentService } from '../student.service';

import { Student } from '../student';

import { ActivatedRoute, Router } from '@angular/router';

import { FormBuilder, FormGroup, FormsModule, Validators } from '@angular/forms';

@Component({

  selector: 'app-student-form',

  templateUrl: './student-form.component.html',

  styleUrls: ['./student-form.component.css'],

  imports: [FormsModule, CommonModule] // Add CommonModule to imports

})

export class StudentFormComponent implements OnInit {

  studentForm: FormGroup;

  isEdit = false;

  student: Student = { name: '', course: '', grade: 0 };

  constructor(

    private fb: FormBuilder,

    private studentService: StudentService,

    private route: ActivatedRoute,

    private router: Router

  ) {

    this.studentForm = this.fb.group({

      name: ['', Validators.required],

      course: [''],

      grade: [0, [Validators.required, Validators.min(0), Validators.max(100)]]

    });

  }

  ngOnInit(): void {

    const id = this.route.snapshot.paramMap.get('get')?.toString() || this.route.snapshot.paramMap.get('id');

    if (id) {

      this.isEdit = true;

      this.studentService.getStudentById(id).subscribe((data: Student) => {

        this.student = data;

        this.studentForm.patchValue(data);

      });

    }

  }

saveStudent(): void {

  // Ensure form is patched with latest student values before saving

  if (this.isEdit && this.student) {

    this.studentForm.patchValue(this.student);

  }

  const formValue = this.studentForm.value;

  const studentData: Student = {

    id: this.student.id,

    name: formValue.name,

    course: formValue.course,

    grade: formValue.grade

  };

  if (this.isEdit) {

    this.studentService.updateStudent(studentData).subscribe(() => {

      this.router.navigate(['/students']);

    });

  } else {

    this.studentService.createStudent(studentData).subscribe(() => {

      this.router.navigate(['/students']);

    });

  }

}

}

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

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';

import { StudentService } from '../student.service';

import { Student } from '../student';

import { Router } from '@angular/router';

@Component({

  selector: 'app-student-list',

  templateUrl: './student-list.component.html',

  styleUrls: ['./student-list.component.css'],

  imports: [CommonModule, FormsModule]

})

export class StudentListComponent implements OnInit {

  students: Student[] = [];

  sortBy: keyof Student = 'id';

  constructor(

    private studentService: StudentService,

    private router: Router

  ) {}

  ngOnInit(): void {

    this.getStudents();

  }

  getStudents(): void {

    this.studentService.getStudents('', this.sortBy).subscribe((data: Student[]) => {

      this.students = data;

    });

  }

  editStudent(student: Student): void {

    this.router.navigate(['/form', student.id]);

  }

  deleteStudent(student: Student): void {

    if (confirm('Are you sure you want to delete this student?')) {

      this.studentService.deleteStudent(student.id!).subscribe(() => {

        this.getStudents();

      });

    }

  }

  changeSort(field: keyof Student): void {

    this.sortBy = field;

    this.students.sort((a, b) => {

      const valA = a[field];

      const valB = b[field];

      if (typeof valA === 'string' && typeof valB === 'string') {

        return valB.localeCompare(valA);

      }

      return (valB as number) - (valA as number);

    });

  }

}

export interface Employee {

  id?: string;

  name: string;

  department: string;

  salary: number;

}

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

import { HttpClient, HttpParams } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Employee } from './employee';

import { environment } from '../environments/environment';

@Injectable({

  providedIn: 'root'

})

export class EmployeeService {

 private apiUrl = `${environment.apiUrl}/employees`;

  constructor(private http: HttpClient) {}

  getEmployees(): Observable<Employee[]> {

    return this.http.get<Employee[]>(this.apiUrl);

  }

  getEmployeeById(id: string): Observable<any> {

    return this.http.get(`${this.apiUrl}/${id}`);

  }

  createEmployee(employee: Employee): Observable<any> {

    return this.http.post(this.apiUrl, employee);

  }

  updateEmployee(employee: Employee): Observable<any> {

    return this.http.put(`${this.apiUrl}/${employee.id}`, employee);

  }

  deleteEmployee(id: string): Observable<any> {

    return this.http.delete(`${this.apiUrl}/${id}`);

  }

}

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

import { EmployeeService } from '../employee.service';

import { Employee } from '../employee';

import { ActivatedRoute, Router } from '@angular/router';

import { NgForm } from '@angular/forms';

@Component({

  selector: 'app-employee-form',

  templateUrl: './employee-form.component.html',

  styleUrls: ['./employee-form.component.css'],

})

export class EmployeeFormComponent implements OnInit {

  employee = {

    id: '',

    name: '',

    department: '',

    salary: 0

  };

  isEdit = false;

  constructor(

    private employeeService: EmployeeService,

    private route: ActivatedRoute,

    private router: Router

  ) {}

ngOnInit(): void {

  const id = this.route.snapshot.paramMap.get('id');

  // Defensive logic: only treat as edit if explicitly marked

  if (this.isEdit || (id && id !== '')) {

    this.isEdit = true;

    this.employeeService.getEmployeeById(id!).subscribe(emp => {

      this.employee = emp;

    });

  }

}

saveEmployee(form: NgForm): void {

  if (form.invalid) return;

  if (this.isEdit) {

    this.employeeService.updateEmployee(this.employee).subscribe(() => {

      this.router.navigate(['/employees']);

    });

  } else {

    this.employeeService.createEmployee(this.employee).subscribe(() => {

      this.router.navigate(['/employees']);

    });

  }

}

}

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

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';

import { EmployeeService } from '../employee.service';

import { Employee } from '../employee';

import { Router } from '@angular/router';

@Component({

  selector: 'app-employee-list',

  templateUrl: './employee-list.component.html',

})

export class EmployeeListComponent implements OnInit {

  employees: any[] = [];

  sortBy: string = 'name';

  sortOrder: 'asc' | 'desc' = 'asc';

  constructor(

    private employeeService: EmployeeService,

    private router: Router

  ) {}

  ngOnInit(): void {

    this.getEmployees();

  }

  getEmployees(): void {

    this.employeeService.getEmployees().subscribe(data => {

      this.employees = [...data]; // clone to avoid mutation

      this.sortEmployees();

    });

  }

  editEmployee(employee: any): void {

    this.router.navigate(['/form', employee.id]);

  }

  deleteEmployee(id: string): void {

    this.employeeService.deleteEmployee(id).subscribe(() => {

      this.getEmployees();

    });

  }

  changeSort(column: string): void {

    if (this.sortBy === column) {

      this.sortOrder = this.sortOrder === 'asc' ? 'desc' : 'asc';

    } else {

      this.sortBy = column;

      this.sortOrder = 'asc';

    }

    this.sortEmployees();

  }

  private sortEmployees(): void {

    this.employees.sort((a, b) => {

      const valA = a[this.sortBy];

      const valB = b[this.sortBy];

      if (typeof valA === 'string') {

        return this.sortOrder === 'asc'

          ? valA.localeCompare(valB)

          : valB.localeCompare(valA);

      }

      if (typeof valA === 'number') {

        return this.sortOrder === 'asc' ? valA - valB : valB - valA;

      }

      return 0;

    });

  }

}