Лабораторна робота №6. Робота з проектом «SportShop» (продовження). Створення функціонального модуля для адміністрування магазину.

Мета: В розробленому проекті «SportShop» створити динамічно завантажуваний функціональний модуль Angular, який містив би інструменти адміністрування необхідні для ведення каталогу товарів і обробки замовлень.

Завдання:

I) Створити автономний функціональний модуль admin.module.ts для адміністрування Angular додатку SportShop з відповідною маршрутизацією для відображення компонентів з адміністрування, таких як: введення логіну та пароля адміністратора; редагування наявних продуктів, видалення та створення нових продуктів; виведення усіх заказів з можливістю видаляти існуючі закази.

II) До проекту додати бібліотеку Angular Material для відображення таблиці продуктів та заказів в панелі адміністратора.

III) Розробити меню адміністратора з наступними пунктами: продукти (Products), замовлення (Orders) та вихід (Logout). При переході в меню Products необхідно вивести всі наявні продукти з можливістю їх видалення, редагування та створення нових. Редагувати і створювати новий продукт адміністратор повинен в окремому вікні. При переході в меню Orders відобразити вікно з усіма замовниками та для кожного замовника вивести перелік замовлень. Також передбачити можливість видалення замовлення та варіант, коли треба відобразити тільки виконані замовлення.

IV) Зробити звіт по роботі. Звіт повинен включати: титульний лист, зміст, основну частину, список використаних джерел.

V) Роботу додатку «SportShop», отриманого в результаті виконання завдання, продемонструвати в режимі відеоконференції.

В цій лабораторній роботі до додатку «SportShop» додамо функції адміністрування. Доступ до функцій адміністрування знадобиться відносно невеликій кількості користувачів, і було б неефективно змушувати всіх користувачів завантажувати весь адміністративний код та контент, які їм, швидше за все, не потрібні. Натомість функції адміністрування будуть включені у функціональний модуль, який завантажуватиметься лише у разі потреби. В цій роботі ми підготуємо програму: створимо функціональний модуль, додамо деякий вихідний контент і налаштуємо код програми так, щоб модуль завантажувався динамічно [11].

I) Створення автономного функціонального модуля admin.module.ts для адміністрування Angular додатку «SportShop»

Підготовка додатку

Щоб запустити REST-сумісну веб-службу, відкрийте вікно командного рядка та виконайте наступну команду з папки SportShop:

```
npm run json
```

Відкрийте друге вікно командного рядка та введіть наступну команду з папки SportShop, щоб запустити інструменти розробки та сервер HTTP:

```
ng serve --port 4200 --open
```

Процес створення функціонального модуля для адміністрування стандартний, тільки враховуючи один важливий момент: ніяка інша частина програми не повинна залежати від модуля або класів, що містяться в ньому, так як це порушило б саму концепцію динамічного завантаження модуля і змусило модуль JavaScript завантажувати код адміністрування, навіть якщо він не використовується.

Робота над функціями адміністрування має розпочатися з аутентифікації [11]. Створіть файл auth.component.ts у папці SportShop/src/app/admin і визначте компонент із лістингу 10.1.

Лістинг 10.1. Вміст файлу auth.component.ts в папці SportShop/src/app/admin

```
import { Component } from "@angular/core";
import { NgForm } from "@angular/forms";
import { Router } from "@angular/router";
@Component({
templateUrl: "auth.component.html"
})
export class AuthComponent {
    username?: string;
    password?: string;
    errorMessage?: string;
    constructor(private router: Router) {}
    authenticate(form: NgForm) {
        if (form.valid) {
            // виконати аутентифікацію
            this.router.navigateByUrl("/admin/main");
        } else {
            this.errorMessage = "Form Data Invalid";
        }
    }
}
```

Компонент визначає властивості для імені користувача та пароля, які будуть використовуватися для аутентифікації; властивість errorMessage, яка виводитиме повідомлення для користувача при виникненні проблем; і метод authenticate для виконання процесу аутентифікації (поки що цей метод нічого не робить).

Щоб створити шаблон для компонента, створіть файл auth.component.html у папці SportShop/src/app/admin і додайте до нього контент із лістингу 10.2.

Лістинг 10.2. Ввміст файлу auth.component.html в папці SportShop/src/app/admin

```
<div class="bg-info p-2 text-center text-white">
    <h3>SportShop Admin</h3>
</div>
<div class="bg-danger mt-2 p-2 text-center text-white" *ngIf="errorMessage != null">
    {{errorMessage}}
</div>
<div class="p-2">
    <form novalidate #form="ngForm" (ngSubmit)="authenticate(form)">
        <div class="form-group">
            <label>Name</label>
            <input class="form-control" name="username"
                [(ngModel)]="username" required />
        </div>
        <div class="form-group">
```

```
                <label>Password</label>
                <input class="form-control" type="password" name="password"
                [(ngModel)]="password" required />
            </div>
            <div class="text-center p-2">
                <button class="btn btn-secondary m-1" routerLink="/">Go back</button>
                <button class="btn btn-primary m-1" type="submit">Log In</button>
            </div>
        </form>
    </div>
```

Шаблон містить форму HTML, яка використовує двосторонні вирази прив'язки даних для властивостей компонента. Форма містить кнопку надсилання даних, кнопку для повернення до кореневого URL та елемент div, видимий тільки при виведенні повідомлення про помилку.

Щоб створити тимчасову реалізацію функцій адміністрування, створіть файл admin.component.ts у папці SportShop/src/app/admin та включіть у нього визначення компонента з лістингу 10.3.

Лістинг 10.3. Вміст файлу admin.component.ts в папці SportShop/app/admin

```
import { Component } from "@angular/core";
@Component({
        templateUrl: "admin.component.html"
})
export class AdminComponent {}
```

В даний час компонент не містить жодної функціональності. Щоб визначити шаблон для компонента, створіть файл admin.component.html у папці SportShop/src/app/admin та додайте тимчасовий контент із лістингу 10.4.

Лістинг 10.4. Вміст файлу admin.component.html в папці SportShop/src/app/admin

```
<div class="bg-info p-2 text-white">
        <h3>Placeholder for Admin Features</h3>
</div>
```

Щоб визначити функціональний модуль, створіть файл admin.module.ts у папці SportShop/app/admin та додайте код із лістингу 10.5.

Лістинг 10.5. Вміст файлу admin.module.ts в папці SportShop/src/app/admin

```
import { NgModule } from "@angular/core";
import { CommonModule } from "@angular/common";
import { FormsModule } from "@angular/forms";
import { RouterModule } from "@angular/router";
import { AuthComponent } from "./auth.component";
import { AdminComponent } from "./admin.component";
let routing = RouterModule.forChild([
        { path: "auth", component: AuthComponent },
        { path: "main", component: AdminComponent },
        { path: "**", redirectTo: "auth" }
]);
@NgModule({
        imports: [CommonModule, FormsModule, routing],
        declarations: [AuthComponent, AdminComponent]
})
export class AdminModule { }
```

Головна відмінність при створенні модуля, що динамічно завантажується, полягає в тому, що функціональний модуль повинен бути автономним і в нього повинна бути включена вся інформація, необхідна для Angular, включаючи підтримувані URL для маршрутизації і для відображення компонентів.

Метод RouterModule.forChild використовується для визначення конфігурації маршрутизації функціонального модуля, яка потім включається до властивості imports модуля.

Заборона на експортування класів з модуля, що динамічно завантажується, не поширюється на імпортування. Цей модуль залежить від функціонального модуля моделі, який був доданий у властивість imports модуля, щоб компоненти могли звертатися до репозиторіїв та класів моделі.

Налаштування системи маршрутизації URL

Для керування модулями, що динамічно завантажуються, використовується конфігурація маршрутизації, яка ініціює процес завантаження при переході програми до конкретної URL-адреси. Лістинг 10.6 розширює конфігурацію маршрутизації програми, щоб за URL/admin завантажувався функціональний модуль адміністрування.

Лістинг 10.6. Налаштування динамічно завантажуваного модуля у файлі app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { ShopModule } from "./shop/shop.module";
import { ShopComponent } from "./shop/shop.component";
import { CheckoutComponent } from "./shop/checkout.component";
import { CartDetailComponent } from "./shop/cartDetail.component";
import { RouterModule } from "@angular/router";
import { ShopFirstGuard } from "./shopFirst.guard";
@NgModule({
    declarations: [AppComponent],
    imports: [BrowserModule, ShopModule,
        RouterModule.forRoot([
            {
                path: "shop", component: ShopComponent,
                canActivate: [ShopFirstGuard]
            },
            {
                path: "cart", component: CartDetailComponent,
                canActivate: [ShopFirstGuard]
            },
            {
                path: "checkout", component: CheckoutComponent,
                canActivate: [ShopFirstGuard]
            },
            {
                path: "admin",
                loadChildren: () => import("./admin/admin.module")
                    .then(m => m.AdminModule),
                canActivate: [ShopFirstGuard]
            },
            { path: "**", redirectTo: "/shop" }
        ])],
    providers: [ShopFirstGuard],
```

```
    bootstrap: [AppComponent]
})
export class AppModule { }
```

Новий маршрут повідомляє Angular, що при переході програми на URL/admin слід завантажити функціональний модуль, який визначається класом з ім'ям AdminModule з файлу /app/admin/admin.module.ts. Під час обробки адміністративного модуля Angular вбудовує маршрутну інформацію, що міститься в ньому, в загальний набір маршрутів.

Перехід по URL адміністрування

Залишилось зробити останній підготовчий крок - надати користувачеві можливість перейти на URL /admin, щоб завантажився функціональний модуль адміністрування, а компонент з'явився на екрані. У лістингу 10.7 до шаблону компонента магазину додається кнопка для переходу.

Лістинг 10.7. Додавання кнопки навігації в файл shop.component.html

```
…
<div class="d-grid gap-2">
    <button class="btn btn-outline-primary btn-block" (click)="changeCategory()">
        Home
    </button>
    <button *ngFor="let cat of categories"
        class="btn btn-outline-primary btn-block"
        [class.active]="cat == selectedCategory"
        (click)="changeCategory(cat)">
        {{cat}}
    </button>
    <button class="btn btn-danger btn-block" routerLink="/admin">
        Admin
    </button>
</div>
…
```

Щоб побачити результат, використовуйте засоби розробника F12 у браузері для перегляду запитів, виданих браузером під час завантаження програми. Файли модуля адміністрування не завантажуватимуться, доки користувач не активізує кнопку Admin; у цей момент Angular запитує файли та виводить сторінку введення облікових даних (рис. 10.1).

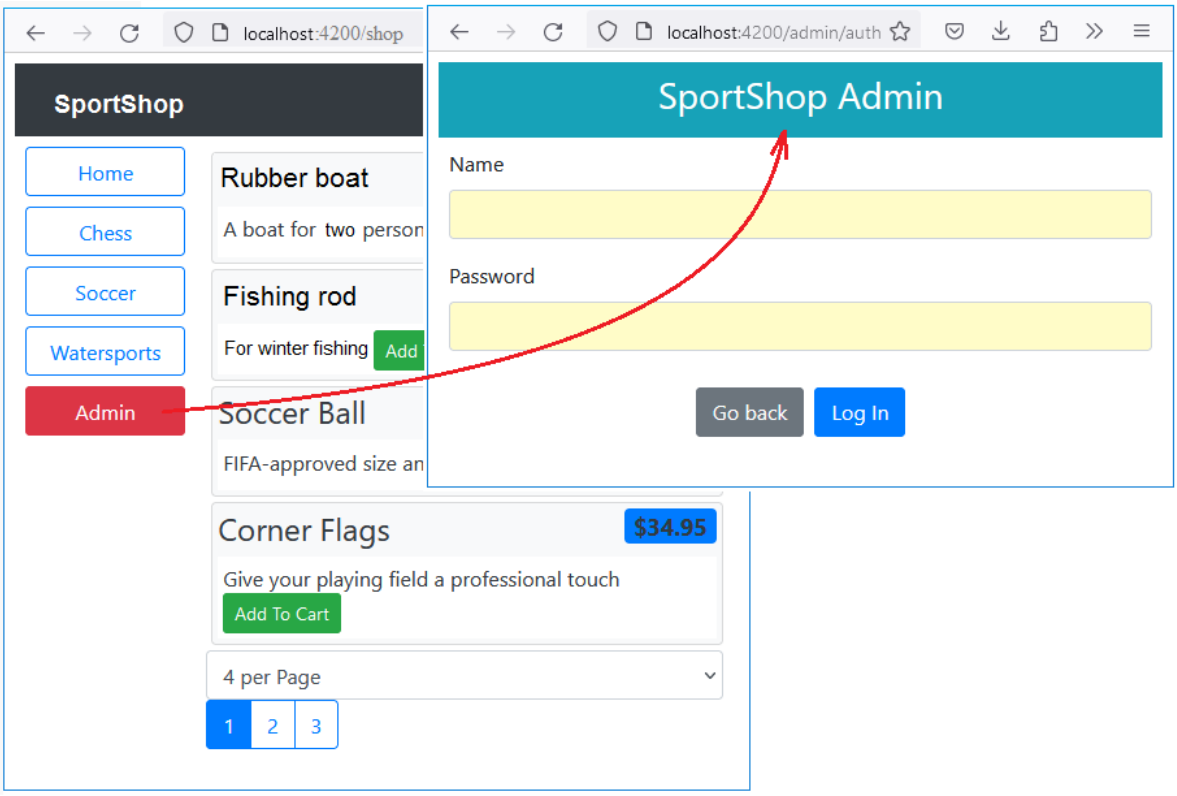

Рис. 10.1. Використання модуля, що динамічно завантажується.

Введіть ім'я та пароль у полях форми. Натисніть кнопку Log In, щоб переглянути тимчасовий контент (рис. 10.2). Якщо хоча б одне з полів залишиться порожнім, виводиться попереджувальне повідомлення.

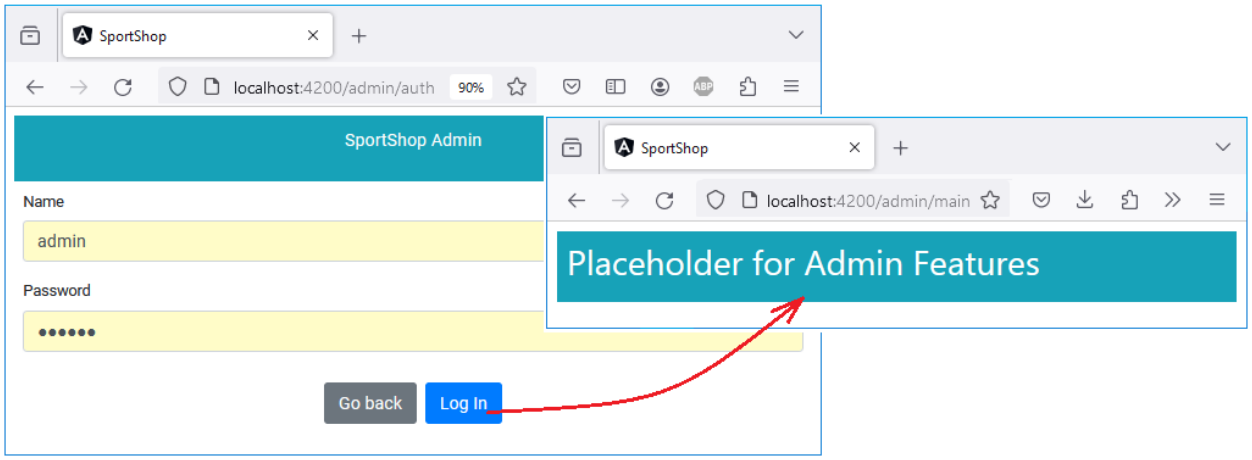

Рис. 10.2. Тимчасова заміна функцій адміністрування

Реалізація аутентифікації

REST-сумісна веб-служба була налаштована так, щоб вона вимагала аутентифікації для запитів, необхідних засобам адміністрування. Нижче модель даних буде розширена для підтримки аутентифікованих запитів HTTP та інтегрована у модуль адміністрування.

Система аутентифікації

Результатом аутентифікації з REST-сумісною веб-службою є веб-маркер JSON (JWT, JSON Web Token), який повертається сервером і повинен включатися в усі наступні запити, тим самим показуючи, що програма дозволяє виконувати захищені операції. Зі

специфікацією JWT можна ознайомитися за адресою https://tools.ietf.org/html/rfc7519, а поки в контексті програми SportShop достатньо знати, що програма Angular може провести аутентифікацію відправкою запиту POST на URL /login, включаючи в тіло запиту об'єкт в форматі JSON з властивостями name і password. У коді аутентифікації існує тільки один набір дійсних облікових даних, наведений в табл. 10.1.

Таблиця 10.1. Облікові дані аутентифікації, які підтримує REST-сумісна веб-служба

Ім’я користувача	Пароль
admin	secret

У реальних проектах облікові дані не повинні жорстко кодуватися, але для нашого навчального проекту SportShop ми відійдемо від цього правила і будемо використовувати саме ці ім'я та пароль в файлі authMiddleware.js. Якщо на URL /login надіслані правильні облікові дані, відповідь від REST-сумісної веб-служби міститиме об'єкт JSON наступного вигляду:

```
{
"success": true,
"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjoiYWRtaW4iLCJleHBpcmVz
SW4iOiIxaCIsImlhdCI6MTQ3ODk1NjI1Mn0.lJaDDrSu-bHBtdWrz0312p_DG5tKypGv6cA
NgOyzlg8"
}
```

Властивість success визначає результат операції аутентифікації, а властивість token містить маркер JWT, який повинен включатися в наступні запити за допомогою заголовка HTTP Authorization в наступному форматі:

```
Authorization: Bearer<eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjoiYWRtaW4iLC
JleHBpcmVzSW4iOiIxaCIsImlhdCI6MTQ3ODk1NjI1Mn0.lJaDDrSu-
bHBtdWrz0312p_DG5tKypGv6cANgOyzlg8>
```

Якщо серверу були надіслані невірні облікові дані, то об'єкт JSON, повернутий у відповіді, міститиме лише властивість success зі значенням false:

```
{
"success": false
}
```

Редагування джерела даних

Більшість роботи буде виконуватися класом REST-сумісної веб-служби, тому що цей клас відповідає за надсилання запитів аутентифікації на URL /login та включення JWT до наступних запитів. У лістингу 10.8 до класу RestDataSource додається аутентифікація та визначається змінна, в яку записується маркер JWT після його отримання у випадку успішного проходження аутентифікації користувачем.

Лістинг 10.8. Включення аутентифікації в файл rest.datasource.ts

```
import { Injectable } from "@angular/core";
import { HttpClient } from "@angular/common/http";
import { map, Observable } from "rxjs";
import { Product } from "./product.model";
import { Order } from "./order.model";
const PROTOCOL = "http";
const PORT = 3500;
@Injectable()
export class RestDataSource {
```

```
    baseUrl: string;
    auth_token?: string;
    constructor(private http: HttpClient) {
        this.baseUrl = `${PROTOCOL}://${location.hostname}:${PORT}/`;
    }
    getProducts(): Observable<Product[]> {
        return this.http.get<Product[]>(this.baseUrl + "products");
    }
    saveOrder(order: Order): Observable<Order> {
        return this.http.post<Order>(this.baseUrl + "orders", order);
    }
    authenticate(user: string, pass: string): Observable<boolean> {
        return this.http.post<any>(this.baseUrl + "login", {
            name: user, password: pass
        }).pipe(map(response => {
            this.auth_token = response.success ? response.token : null;
            return response.success;
        }));
    }
}
```

Метод pipe та функція map надаються пакетом RxJS, і вони дозволяють відповідну реакцію сервера, яка представлена типом Observable<any> перетворити в Observable<bool>, що є результатом методу аутентифікації.

Створення служби аутентифікації

Замість того, щоб відкривати доступ до джерела даних всьому коду програми, ми створимо службу, яка використовуватиметься для виконання аутентифікації та перевірки її проходження. Створіть файл auth.service.ts у папці SportShop/src/app/model та визначте клас з лістингу 10.9.

Лістинг 10.9. Вміст файлу auth.service.ts в папці SportShop/src/app/model

```
import { Injectable } from "@angular/core";
import { Observable } from "rxjs";
import { RestDataSource } from "./rest.datasource";
@Injectable()
export class AuthService {

    constructor(private datasource: RestDataSource) {}

    authenticate(username: string, password: string): Observable<boolean> {
        return this.datasource.authenticate(username, password);
    }

    get authenticated(): boolean {
        return this.datasource.auth_token != null;
    }

    clear() {
        this.datasource.auth_token = undefined;
    }
}
```

Метод authenticate отримує облікові дані користувача та передає їх методу authenticate джерела даних, повертаючи об'єкт Observable, який повертає true, якщо процес аутентифікації завершився успішно, або false у протилежному випадку. Властивість

authenticated, доступна тільки для читання, повертає true, якщо джерело даних отримало маркер аутентифікації. Метод clear видаляє маркер із джерела даних.

Лістинг 10.10 реєструє нову службу у загальному модулі моделі. Він також додає запис providers для класу RestDataSource, який у попередніх лабораторних роботах використовувався лише як заміна для класу StaticDataSource. Оскільки клас AuthService має параметр конструктора RestDataSource, йому потрібен власний запис у модулі.

Лістинг 10.10. Конфігурація служб в файле model.module.ts

```
import { NgModule } from "@angular/core";
import { ProductRepository } from "./product.repository";
import { StaticDataSource } from "./static.datasource";
import { Cart } from "./cart.model";
import { Order } from "./order.model";
import { OrderRepository } from "./order.repository";
import { RestDataSource } from "./rest.datasource";
import { HttpClientModule } from "@angular/common/http";
import { AuthService } from "./auth.service";
@NgModule({
    imports: [HttpClientModule],
    providers: [ProductRepository, StaticDataSource, Cart, Order,
OrderRepository,
        { provide: StaticDataSource, useClass: RestDataSource },
        RestDataSource, AuthService]
})
export class ModelModule { }
```

Включення аутентифікації

На наступному кроці компонент, який отримує облікові дані від користувача, підключається до виконання аутентифікації через нову службу, як показано у лістингу 10.11.

Лістинг 10.11. Включение аутентификации в файле auth.component.ts

```
import { Component } from "@angular/core";
import { NgForm } from "@angular/forms";
import { Router } from "@angular/router";
import { AuthService } from "../model/auth.service";
@Component({
    templateUrl: "auth.component.html"
})
export class AuthComponent {
    username?: string;
    password?: string;
    errorMessage?: string;

    constructor(private router: Router,
        private auth: AuthService) { }

    authenticate(form: NgForm) {
        if (form.valid) {
            this.auth.authenticate(this.username ?? "", this.password ?? "")
                .subscribe(response => {
                    if (response) {
                        this.router.navigateByUrl("/admin/main");
                    }
                    this.errorMessage = "Authentication Failed";
                })
```

```
            } else {
                this.errorMessage = "Form Data Invalid";
            }
        }
    }
```

Щоб програма не допускала прямих переходів до функцій адміністрування, що призвело б до надсилання запитів HTTP без маркера, створіть файл auth.guard.ts у папці SportShop/src/app/admin та визначте захисника маршруту з лістингу 10.12.

Лістинг 10.12. Вміст файлу auth.guard.ts в папці SportShop/app/admin

```
import { Injectable } from "@angular/core";
import { ActivatedRouteSnapshot, RouterStateSnapshot,
Router } from "@angular/router";
import { AuthService } from "../model/auth.service";
@Injectable()
export class AuthGuard {
    constructor(private router: Router, private auth: AuthService) { }
    canActivate(route: ActivatedRouteSnapshot,
        state: RouterStateSnapshot): boolean {

        if (!this.auth.authenticated) {
            this.router.navigateByUrl("/admin/auth");
            return false;
        }
        return true;
    }
}
```

У лістингу 10.13 захисник маршруту застосовується до одного з маршрутів, що визначаються функціональним модулем адміністрування.

Лістинг 10.13. Захист маршрута в файлі admin.module.ts

```
import { NgModule } from "@angular/core";
import { CommonModule } from "@angular/common";
import { FormsModule } from "@angular/forms";
import { RouterModule } from "@angular/router";
import { AuthComponent } from "./auth.component";
import { AdminComponent } from "./admin.component";
import { AuthGuard } from "./auth.guard";
let routing = RouterModule.forChild([
    { path: "auth", component: AuthComponent },
    { path: "main", component: AdminComponent },
    { path: "main", component: AdminComponent, canActivate: [AuthGuard] },
    { path: "**", redirectTo: "auth" }
]);
@NgModule({
    imports: [CommonModule, FormsModule, routing],
    declarations: [AuthComponent, AdminComponent],
    providers: [AuthGuard]
})
export class AdminModule { }
```

Щоб протестувати систему аутентифікації, клацніть на кнопці Admin, введіть облікові дані та клацніть на кнопці Log In. Якщо запровадити облікові дані з табл. 10.1, то

ви побачите тимчасовий контент функцій адміністрування. Якщо ввести інші облікові дані, з'явиться повідомлення про помилку. Обидва результати представлені на рис. 10.3.

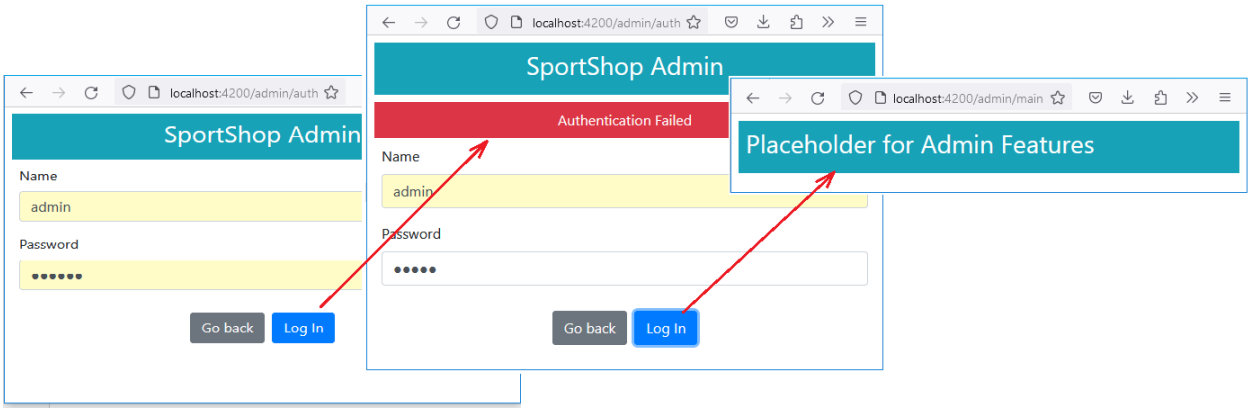

Рис. 10.3. Тестування функцій адміністрування

Розширення джерела даних та репозиторіїв

Після того, як система аутентифікації займе своє місце, наступним кроком має стати розширення джерела даних для надсилання аутентифікованих запитів та надання доступу до цих функцій через класи замовлень та репозиторій товарів.

Лістинг 10.14. Додавання нових операцій в файлі rest.datasource.ts в папці src/app/model

```
import { Injectable } from "@angular/core";
import { HttpClient } from "@angular/common/http";
import { map, Observable } from "rxjs";
import { Product } from "./product.model";
import { Order } from "./order.model";
import { HttpHeaders } from '@angular/common/http';

const PROTOCOL = "http";
const PORT = 3500;
@Injectable()
export class RestDataSource {
    baseUrl: string;
    auth_token?: string;
    constructor(private http: HttpClient) {
        this.baseUrl = `${PROTOCOL}://${location.hostname}:${PORT}/`;
    }
    getProducts(): Observable<Product[]> {
        return this.http.get<Product[]>(this.baseUrl + "products");
    }
    saveOrder(order: Order): Observable<Order> {
        return this.http.post<Order>(this.baseUrl + "orders", order);
    }
    authenticate(user: string, pass: string): Observable<boolean> {
        return this.http.post<any>(this.baseUrl + "login", {
            name: user, password: pass
        }).pipe(map(response => {
            this.auth_token = response.success ? response.token : null;
            return response.success;
        }));
    }
    saveProduct(product: Product): Observable<Product> {
```

```
            return this.http.post<Product>(this.baseUrl + "products",
                product, this.getOptions());
        }
        updateProduct(product: Product): Observable<Product> {
            return this.http.put<Product>(`${this.baseUrl}products/${product.id}`,
                product, this.getOptions());
        }
        deleteProduct(id: number): Observable<Product> {
            return this.http.delete<Product>(`${this.baseUrl}products/${id}`,
                this.getOptions());
        }
        getOrders(): Observable<Order[]> {
            return this.http.get<Order[]>(this.baseUrl + "orders", this.getOptions());
        }
        deleteOrder(id: number): Observable<Order> {
            return this.http.delete<Order>(`${this.baseUrl}orders/${id}`,
                this.getOptions());
        }
        updateOrder(order: Order): Observable<Order> {
            return this.http.put<Order>(`${this.baseUrl}orders/${order.id}`,
                order, this.getOptions());
        }
        private getOptions() {
            return {
                headers: new HttpHeaders({
                    "Authorization": `Bearer<${this.auth_token}>`
                })
            }
        }
    }
```

У лістингу 10.15 до класу репозиторію товарів додаються нові методи для створення, оновлення або видалення товарів. Метод saveProduct відповідає за створення та оновлення товарів; цей підхід добре працює при використанні одного об'єкта, який знаходиться під керуванням компонента. У лістингу 10.15 аргументу конструктора також призначається новий тип RestDataSource.

Лістинг 10.15. Додавання нових операцій у файл product.repository.ts в папці src/app/model Folder

```
import { Injectable } from "@angular/core";
import { Product } from "./product.model";
//import { StaticDataSource } from "./static.datasource";
import { RestDataSource } from "./rest.datasource";
@Injectable()
export class ProductRepository {
    private products: Product[] = [];
    private categories: string[] = [];
    constructor(private dataSource: RestDataSource) {
        dataSource.getProducts().subscribe(data => {
            this.products = data;
            this.categories = data.map(p => p.category ?? "(None)")
                .filter((c, index, array) => array.indexOf(c) == index).sort();
        });
    }
    getProducts(category?: string): Product[] {
        return this.products
```

```
                .filter(p => category == undefined || category == p.category);
        }
        getProduct(id: number): Product | undefined {
            return this.products.find(p => p.id == id);
        }
        getCategories(): string[] {
            return this.categories;
        }
        saveProduct(product: Product) {
            if (product.id == null || product.id == 0) {
                this.dataSource.saveProduct(product)
                    .subscribe(p => this.products.push(p));
            } else {
                this.dataSource.updateProduct(product)
                    .subscribe(p => {
                        this.products.splice(this.products.
                            findIndex(p => p.id == product.id), 1, product);
                    });
            }
        }
        deleteProduct(id: number) {
            this.dataSource.deleteProduct(id).subscribe(p => {
                this.products.splice(this.products.
                    findIndex(p => p.id == id), 1);
            })
        }
    }
```

У лістингу 10.16 відповідні зміни вносяться до репозиторію замовлень, з додаванням методу модифікації та видалення замовлень.

Лістинг 10.16. Додавання нових операцій в файл order.repository.ts в папці src/app/model

```
import { Injectable } from "@angular/core";
import { Observable } from "rxjs";
import { Order } from "./order.model";
//import { StaticDataSource } from "./static.datasource";
import { RestDataSource } from "./rest.datasource";
@Injectable()
export class OrderRepository {
    private orders: Order[] = [];
    private loaded: boolean = false;

    constructor(private dataSource: RestDataSource) { }
    loadOrders() {
        this.loaded = true;
        this.dataSource.getOrders()
            .subscribe(orders => this.orders = orders);
    }
    getOrders(): Order[] {
        if (!this.loaded) {
            this.loadOrders();
        }
        return this.orders;
    }
    saveOrder(order: Order): Observable<Order> {
```

```
        this.loaded = false;
        return this.dataSource.saveOrder(order);
    }
    updateOrder(order: Order) {
        this.dataSource.updateOrder(order).subscribe(order => {
            this.orders.splice(this.orders.
                findIndex(o => o.id == order.id), 1, order);
        });
    }
    deleteOrder(id: number) {
        this.dataSource.deleteOrder(id).subscribe(order => {
            this.orders.splice(this.orders.findIndex(o => id == o.id), 1);
        });
    }
}
```

Репозиторій замовлень визначає метод loadOrders, який отримує замовлення з репозиторію, та гарантує, що запит не буде надісланий REST-сумісній веб-службі до виконання аутентифікації.

II) Встановлення компонентів додаткової бібліотеки Angular Material

Всі можливості, що представляються користувачу для дизайну і якщо вони повинні бути використані в Angular додатку, можуть бути взяті з бібліотеки Bootstrap. Альтернативним варіантом є використання компонентів library, що містять спільні потреби, такі як tables і layouts та фокусуються на нюансах, які є унікальними для вашого проекту. Досвід використання подібних бібліотек дозволяє зробити висновок про те, що вони дають змогу злегкістю масштабувати проекти та керувати ними, але є і недоліки, які заключаються в тому, що дані та код моделі повинні відповідати компонентам бібліотеки і це може бути складно виконати розробникам відповідної бібліотеки.

Далі в роботі будемо використовувати компоненти бібліотеки Angular Material. Існують і інші хороші пакети, доступні для Angular, але Angular Material є найбільш популярним пакетом. Щоб додати Angular Material до проекту, перейдіть в папку зі своїм проектом і виконайте команду.

Лістинг 10.17. Встановлення бібліотеки Angular Material

```
ng add @angular/material@13.0.2 --defaults
```

Додатково може знадобитися виконати наступну команду:

```
ng add @angular/cdk@13.0.2
```

У процесі встановлення виникає кілька запитань. Перше запитання – це просто запит на підтвердження того, що ви хочете встановити пакет:

```
Using package manager: npm
Package information loaded.
The package @angular/material@13.0.2 will be installed and executed.
Would you like to proceed? (Y/n)
```

Для проекту SportShop всі інші параметри виберіть за замовчуванням.

Кожна функція, яку надає Angular Material, визначається у власному модулі, тому визначемо окремий модуль, який будемо використовувати лише для вибору функцій Angular Material, необхідних для проекту. Додайте файл з назвою material.module.ts до папки src/app/admin із вмістом, показаним у лістингу 10.18.

Лістинг 10.18. Вміст файлу material.module.ts у папці src/app/admin

```
import { NgModule } from "@angular/core";
const features: any[] = [];
@NgModule({
        imports: [features],
        exports: [features]
})
export class MaterialFeatures {}
```

Зараз ще не вибрано жодних функцій Angular Material, але ми додамо їх до цього файлу під час роботи з функціями адміністрування. У лістингу 10.19 цей модуль включається у додаток.

Лістинг 10.19. Використання властивостей Angular Material в файлі admin.module.ts папки src/app/admin

```
import { NgModule } from "@angular/core";
import { CommonModule } from "@angular/common";
import { FormsModule } from "@angular/forms";
import { RouterModule } from "@angular/router";
import { AuthComponent } from "./auth.component";
import { AdminComponent } from "./admin.component";
import { AuthGuard } from "./auth.guard";
import { MaterialFeatures } from "./material.module";
let routing = RouterModule.forChild([
    { path: "auth", component: AuthComponent },
    { path: "main", component: AdminComponent },
    { path: "main", component: AdminComponent, canActivate: [AuthGuard] },
    { path: "**", redirectTo: "auth" }
]);
@NgModule({
    imports: [CommonModule, FormsModule, routing, MaterialFeatures],
    declarations: [AuthComponent, AdminComponent],
    providers: [AuthGuard]
})
export class AdminModule { }
```

Збережіть зміни та виконайте команду ng serve у папці SportShop, щоб знову запустити інструменти розробки Angular.

III) Створення структури підсистеми адміністрування

Наразі, система аутентифікації успішно працює, а репозиторії забезпечують весь спектр операцій. Тепер можна перейти до створення структури для відображення функцій адміністрування, яка буде створена на базі існуючої конфігурації маршрутизації URL. В табл. 10.2 перераховані URL-адреси, які будуть підтримуватися, і функціональність, яку кожна з них буде надавати користувачеві.

Таблиця 10.2. URL-адреси функцій адміністрування

URL	Описание
/admin/main/products	Виведення всіх товарів у таблицю з кнопками для редагування та видалення існуючих, а також створення нових товарів
/admin/main/products/create	Пустий редактор для створення нового товару

/admin/main/products/edit/1	Редактор для редагування існуючого товару (з попереднім заповненням полів)
/admin/main/orders	Список усіх замовлень у таблиці з кнопками для поміток замовлення як відправленого та відміни замовлення з його видаленням

Створення компонентів-заповнювачів

Найпростіший спосіб додати функції до проекту Angular — це визначити компоненти, які мають вміст-заповнювач, і побудувати навколо них структуру програми. Коли структура створена, тоді можна повернутися до компонентів і детально реалізувати їхні функції. Для функцій адміністрування почнемо з додавання файлу під назвою productTable.component.ts до папки src/app/admin і визначемо компонент, показаний у лістингу 10.20. Цей компонент відповідатиме за показ списку продуктів разом із кнопками, необхідними для їх редагування та видалення або створення нового продукту.

Лістинг 10.20. Вміст файлу productTable.component.ts в папці src/app/admin

```
import { Component } from "@angular/core";
@Component({
    template: `
        <h3 style="padding-top: 30px">
            Product Table Placeholder
        </h3>
    `
})
export class ProductTableComponent {}
```

Додамо файл під назвою productEditor.component.ts у папку src/app/admin. Його ми будемо використовувати для визначення компонента, показаного в лістингу 10.21, який використовуватиметься, щоб дозволити користувачеві вводити деталі, необхідні для створення або редагування товару.

Лістинг 10.21. Вміст файлу productEditor.component.ts в папці src/app/admin

```
import { Component } from "@angular/core";
@Component({
    template: `<h3 style="padding-top: 30px">
            Product Editor Placeholder
        </h3>`
})
export class ProductEditorComponent { }
```

Щоб створити компонент, який буде відповідати за керування замовленнями клієнтів, додамо у проект файл під назвою orderTable.component.ts до папки src/app/admin з кодом, показаним у лістингу 10.22.

Лістинг 10.22. Вміст файлу orderTable.component.ts File в папці src/app/admin Folder

```
import { Component } from "@angular/core";
@Component({
    template: `<h3 style="padding-top: 30px">
            Order Table Placeholder
        </h3>`
})
export class OrderTableComponent { }
```

Підготовка загального вмісту та фунціонального модуля

Компоненти, створені раніше, відповідають за певні функції. Щоб об’єднати ці функції та дозволити користувачеві переходити між ними, потрібно змінити шаблон компонента заповнювача, який використовувався для демонстрації результату успішної спроби аутентифікації. Замінимо вміст заповнювача на елементи, показані в лістингу 10.23.

Лістинг 10.23. Заміна вмісту у файлі admin.component.html папки src/app/admin

```
<mat-toolbar color="primary">
    <div>
        <button mat-icon-button *ngIf="sidenav.mode === 'over'"
                (click)="sidenav.toggle()">
            <mat-icon *ngIf="!sidenav.opened">menu</mat-icon>
            <mat-icon *ngIf="sidenav.opened">close</mat-icon>
        </button>
        SportShop Administration
    </div>
</mat-toolbar>
<mat-sidenav-container>
    <mat-sidenav #sidenav="matSidenav" class="mat-elevation-z8">
        <div>
            <button mat-button class="menu-button"
            routerLink="/admin/main/products"
            routerLinkActive="mat-accent"
            (click)="sidenav.close()">
                <mat-icon>shopping_cart</mat-icon>
                <span>Products</span>
            </button>
        </div>
        <div>
            <button mat-button class="menu-button"
            routerLink="/admin/main/orders"
            routerLinkActive="mat-accent"
            (click)="sidenav.close()">
                <mat-icon>local_shipping</mat-icon>
                <span>Orders</span>
            </button>
        </div>

        <mat-divider></mat-divider>

        <button mat-button class="menu-button logout" (click)="logout()">
            <mat-icon>logout</mat-icon>
                <span>Logout</span>
        </button>
    </mat-sidenav>
    <mat-sidenav-content>
        <div class="content">
            <router-outlet></router-outlet>
        </div>
    </mat-sidenav-content>
</mat-sidenav-container>
```

Коли вперше починається робота з бібліотекою компонентів, то може знадобитися деякий час, щоб зрозуміти, як компоненти застосовуються. Цей шаблон використовує панель інструментів Angular Material, застосовану за допомогою елемента mattoolbar, і компонент sidenav, який застосовується через елементи mat-sidenav-container, mat-sidenav і

mat-sidenav-content. Бокова навігація — це панель, що згортається, і містить навігаційний вміст, який дозволяє користувачеві вибирати різні функції адміністрування.

Цей шаблон також містить елемент a router-outlet, який використовуватиметься для відображення компонентів із попереднього шаблону-заповнювача. Панель sidenav містить кнопки, до яких застосовано директиву mat-button, яка форматує кнопки відповідно до решти теми Angular Material. Ці кнопки налаштовано за допомогою атрибутів routerLink, націлених на елемент router-outlet, і стилізовано за допомогою атрибута routerLinkAttribute, щоб вказати, яку функцію було вибрано. Лістинг 10.24 додає залежності від функцій Angular Material, які використовуються в цьому шаблоні.

Лістинг 10.24. Додавання функцій у файл material.module.ts у папці src/app/admin

```
import { NgModule } from "@angular/core";
import { MatToolbarModule } from "@angular/material/toolbar";
import { MatSidenavModule } from "@angular/material/sidenav";
import { MatIconModule } from '@angular/material/icon';
import { MatDividerModule } from '@angular/material/divider';
import { MatButtonModule } from "@angular/material/button";
const features: any[] = [MatToolbarModule,MatSidenavModule,MatIconModule,
MatDividerModule, MatButtonModule];
@NgModule({
        imports: [features],
        exports: [features]
})
export class MaterialFeatures {}
```

Одним із недоліків пакета Angular Material є те, що він вимагає застосування стилів CSS для точного налаштування макета компонента. Лістинг 10.25 визначає стилі, необхідні для розміщення компонентів, які використовуються в лістингу 10.23.

Лістинг 10.25. Визначення стилів у файлі styles.css папки src

```
html, body { height: 100%}
body { margin: 0; font-family: Roboto, "Helvetica Neue", sans-serif; }
mat-toolbar span { flex: 1 1 auto; }
.menu-button { width: 100%; font-size: 1rem; }
.menu-button .mat-icon { margin-right: 10px; }
.menu-button span { flex: 1 1 auto; }
mat-sidenav { margin: 16px; width: 175px; border-right: none;
    border-radius: 4px; padding: 4px;
}
mat-sidenav .mat-divider { margin-top: 20px; margin-bottom: 5px; }
mat-sidenav-container { height: calc(100vh - 60px); }
mat-sidenav .mat-button-wrapper {
    display: flex; width: 100%; justify-content: baseline; align-content: center;
}
mat-sidenav .mat-button-wrapper mat-icon { margin-top: 5px; }
mat-sidenav .mat-button-wrapper span { text-align: start; }
```

Ці стилі може бути складно визначати, тому найкориснішим підходом є використання інструментів розробника браузера F12, щоб визначити, які елементи як повинні виглядати. Панель бокової навігації, визначена в лістингу 10.23, містить кнопку виходу з прив’язкою подій, яка націлена на метод під назвою logout. Лістинг 10.26 додає цей метод до компонента, який використовує службу аутентифікації для видалення маркера носія аутентифікації та переміщує програму до URL-адреси за замовчуванням.

Лістинг 10.26. Реалізація методу виходу з системи у файлі admin.component.ts папки src/app/

```
import { Component } from "@angular/core";
import { Router } from "@angular/router";
import { AuthService } from "../model/auth.service";
@Component({
    templateUrl: "admin.component.html"
})
export class AdminComponent {
    constructor(private auth: AuthService, private router: Router) { }
    logout() {
        this.auth.clear();
        this.router.navigateByUrl("/");
    }
}
```

Лістинг 10.27 активує компоненти-заповнювачі, які використовуватимуться для кожної функції адміністрування, і розширює конфігурацію маршрутизації URL-адрес для реалізації URL-адрес із Таблиці 10.2.

Лістинг 10.27. Налаштування функціонального модуля у файлі admin.module.ts папки src/app/admin

```
import { NgModule } from "@angular/core";
import { CommonModule } from "@angular/common";
import { FormsModule } from "@angular/forms";
import { RouterModule } from "@angular/router";
import { AuthComponent } from "./auth.component";
import { AdminComponent } from "./admin.component";
import { AuthGuard } from "./auth.guard";
import { MaterialFeatures } from "./material.module";
import { ProductTableComponent } from "./productTable.component";
import { ProductEditorComponent } from "./productEditor.component";
import { OrderTableComponent } from "./orderTable.component";
let routing = RouterModule.forChild([
    { path: "auth", component: AuthComponent },
    // { path: "main", component: AdminComponent },
    // { path: "main", component: AdminComponent, canActivate: [AuthGuard] },
    {
        path: "main", component: AdminComponent, canActivate: [AuthGuard],
        children: [
            { path: "products/:mode/:id", component: ProductEditorComponent },
            { path: "products/:mode", component: ProductEditorComponent },
            { path: "products", component: ProductTableComponent },
            { path: "orders", component: OrderTableComponent },
            { path: "**", redirectTo: "products" }
        ]
    },
    { path: "**", redirectTo: "auth" }
]);
@NgModule({
    imports: [CommonModule, FormsModule, routing, MaterialFeatures],
    declarations: [AuthComponent, AdminComponent, ProductTableComponent,
        ProductEditorComponent, OrderTableComponent],
    providers: [AuthGuard]
})
export class AdminModule { }
```

Індивідуальні маршрути можна розширити за допомогою властивості children, яка використовується для визначення маршрутів, спрямованих на вкладений елемент router-outlet. Як ви побачите, компоненти можуть отримати деталі активного маршруту з Angular, щоб вони змогли адаптувати свою поведінку. Маршрути можуть включати параметри маршруту, такі як :mode або :id, які відповідають будь-якому сегменту URL-адреси та які можна використовувати для надання інформації компонентам для зміни їх поведінки.

Коли всі зміни буде збережено, натисніть кнопку «Адміністратор» і пройдіть аутентифікацію як адміністратор із секретним паролем. Ви побачите новий макет, як показано на рис. 10.4. Натисніть кнопку в лівій частині панелі інструментів, щоб відкрити навігаційну панель, і натисніть кнопку «Замовлення», щоб змінити вибраний компонент, або кнопку «Вийти», щоб вийти з області адміністрування.

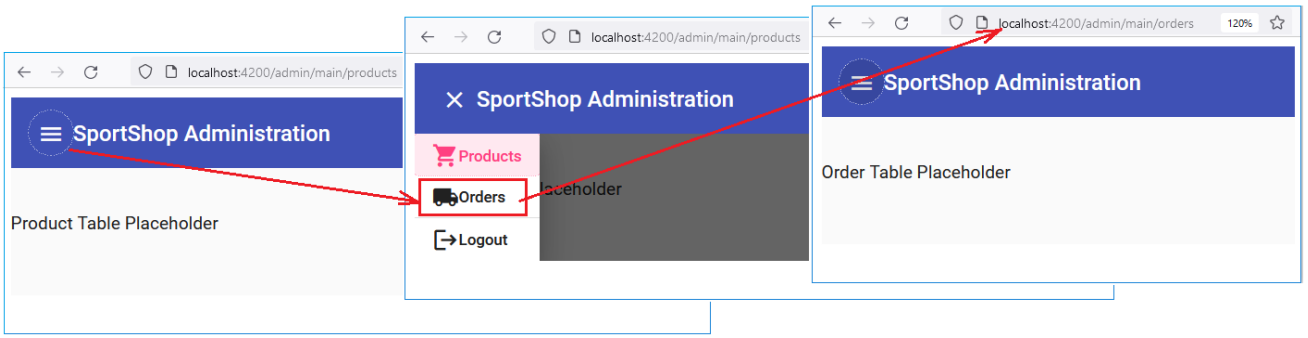

Рис. 10.4. Структура макета адміністрації

Реалізація функції таблиці продуктів

Початкова функція адміністрування, представлена користувачеві, буде таблицею продуктів із можливістю створення нового продукту та видалення чи редагування існуючого. Лістинг 10.28 додає до програми компонент таблиці Angular Material.

Лістинг 10.28. Додавання функцій у файл material.module.ts папки src/app/admin

```
import { NgModule } from "@angular/core";
import { MatToolbarModule } from "@angular/material/toolbar";
import { MatSidenavModule } from "@angular/material/sidenav";
import { MatIconModule } from '@angular/material/icon';
import { MatDividerModule } from '@angular/material/divider';
import { MatButtonModule } from "@angular/material/button";
import { MatTableModule } from "@angular/material/table";
const features: any[] = [MatToolbarModule,MatSidenavModule,MatIconModule,
        MatDividerModule, MatButtonModule, MatTableModule];
@NgModule({
        imports: [features],
        exports: [features]
})
export class MaterialFeatures {}
```

Щоб надати шаблон, який визначає таблицю, додамо файл під назвою productTable.component.html у папку src/app/admin і додамо розмітку, показану в лістингу 10.29.

Лістинг 10.29. Вміст файлу productTable.component.html папки src/app/admin

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

```
        <mat-text-column name="id"></mat-text-column>
        <mat-text-column name="name"></mat-text-column>
        <mat-text-column name="category"></mat-text-column>
        <ng-container matColumnDef="price">
            <th mat-header-cell *matHeaderCellDef>Price</th>
            <td mat-cell *matCellDef="let item"> {{item.price | currency:"USD"}} </td>
        </ng-container>
        <ng-container matColumnDef="buttons">
            <th mat-header-cell *matHeaderCellDef></th>
            <td mat-cell *matCellDef="let p">
                <button mat-flat-button color="accent"
                    (click)="deleteProduct(p.id)">
                    Delete
                </button>
                <button mat-flat-button color="warn"
                    [routerLink]="['/admin/main/products/edit', p.id]">
                    Edit
                </button>
        </ng-container>
        <tr mat-header-row *matHeaderRowDef="colsAndRows"></tr>
        <tr mat-row *matRowDef="let row; columns: colsAndRows"></tr>
    </table>
    <button mat-flat-button color="primary" routerLink="/admin/main/products/create">
        Create New Product
    </button>
```

Таблиця спирається на функції, надані компонентом таблиці Angular Material, який забезпечує хорошу основу для додаткових функцій. Таблиця визначає стовпці, які відображають деталі продуктів, і кожен рядок містить кнопку «Delete», яка викликає метод компонента під назвою «Delete», і кнопку «Edit», яка переміщує користувача до URL-адреси, націленої на компонент редактора. Компонент редактора також має кнопку «Create New Product», хоча використовується інша URL-адреса. Знову ж таки, спеціальні стилі CSS потрібні для точного налаштування макета таблиці, як показано в лістингу 10.30.

Лістинг 10.30. Визначення стилів у файлі styles.css папки src

```
html, body { height: 100%}
body { margin: 0; font-family: Roboto, "Helvetica Neue", sans-serif; }
mat-toolbar span { flex: 1 1 auto; }
.menu-button { width: 100%; font-size: 1rem; }
.menu-button .mat-icon { margin-right: 10px; }
.menu-button span { flex: 1 1 auto; }
mat-sidenav { margin: 16px; width: 175px; border-right: none;
    border-radius: 4px; padding: 4px;
}
mat-sidenav .mat-divider { margin-top: 20px; margin-bottom: 5px; }
mat-sidenav-container { height: calc(100vh - 60px); }
mat-sidenav .mat-button-wrapper {
    display: flex; width: 100%; justify-content: baseline; align-content: center;
}
mat-sidenav .mat-button-wrapper mat-icon { margin-top: 5px; }
mat-sidenav .mat-button-wrapper span { text-align: start; }

table[mat-table] { width: 100%; table-layout: auto; }
table[mat-table] button { margin-left: 5px;}
table[mat-table] th.mat-header-cell { font-size: large; font-weight: bold;}
```

```
table[mat-table] .mat-column-name { width: 25%; }
table[mat-table] .mat-column-buttons { width: 30%; }
table[mat-table] + button[mat-flat-button] { margin-top: 10px;}
```

Як було сказано раніше, використання пакету Angular Material означає, що розробник повинен адаптувати свою програму або дані до очікувань пакета. Angular Material для повного використання переваг таблиць вимагає використання класу джерела даних, що може викликати додаткові складнощі, коли дані отримуються через запит HTTP. Наразі розглянемо підхід, який полягає у використанні функцій, які надає Angular для виявлення та обробки оновлення. Лістинг 10.31 видаляє вміст заповнювача з компонента таблиці продуктів і додає логіку, необхідну для реалізації цієї функції.

Лістинг 10.31. Додавання функцій у файл productTable.component.ts папки src/app/admin

```
import { Component, IterableDiffer, IterableDiffers } from "@angular/core";
import { MatTableDataSource } from "@angular/material/table";
import { Product } from "../model/product.model";
import { ProductRepository } from "../model/product.repository";
@Component({
    templateUrl: "productTable.component.html"
})
export class ProductTableComponent {
    colsAndRows: string[] = ['id', 'name', 'category', 'price', 'buttons'];
    dataSource = new MatTableDataSource<Product>(this.repository.getProducts());
    differ: IterableDiffer<Product>;
    constructor(private repository: ProductRepository, differs: IterableDiffers) {
        this.differ = differs.find(this.repository.getProducts()).create();
    }
    ngDoCheck() {
        let changes = this.differ?.diff(this.repository.getProducts());
        if (changes != null) {
            this.dataSource.data = this.repository.getProducts();
        }
    }
    deleteProduct(id: number) {
        this.repository.deleteProduct(id);
    }
}
```

Властивість colsAndRows використовується для визначення стовпців, які відображаються в таблиці. В додатку обрано усі стовпці, які були визначені, але цю функцію можна використовувати для програмної зміни структури таблиці. Клас MatTableDataSource<Product> з'єднує дані в додатку з таблицею. Об'єкт джерела даних створюється з даними в сховищі, але це не допоможе, якщо компонент відображається до того, як програма отримає дані з сервера. Angular має ефективну систему виявлення змін, яку він використовує, щоб гарантувати, що оновлення обробляються з мінімумом роботи, а метод ngDoCheck дозволяє підключитися до цієї системи та перевірити, чи дані в репозиторії були змінені, використовуючи методи життєвого циклу компоненту. Якщо є зміни в даних, то оновлюється джерело даних, що призводить до оновлення таблиці. Якщо зберегти зміни та увійти до функцій адміністрування, то можна буде побачите таблицю, показану на рис. 10.5.

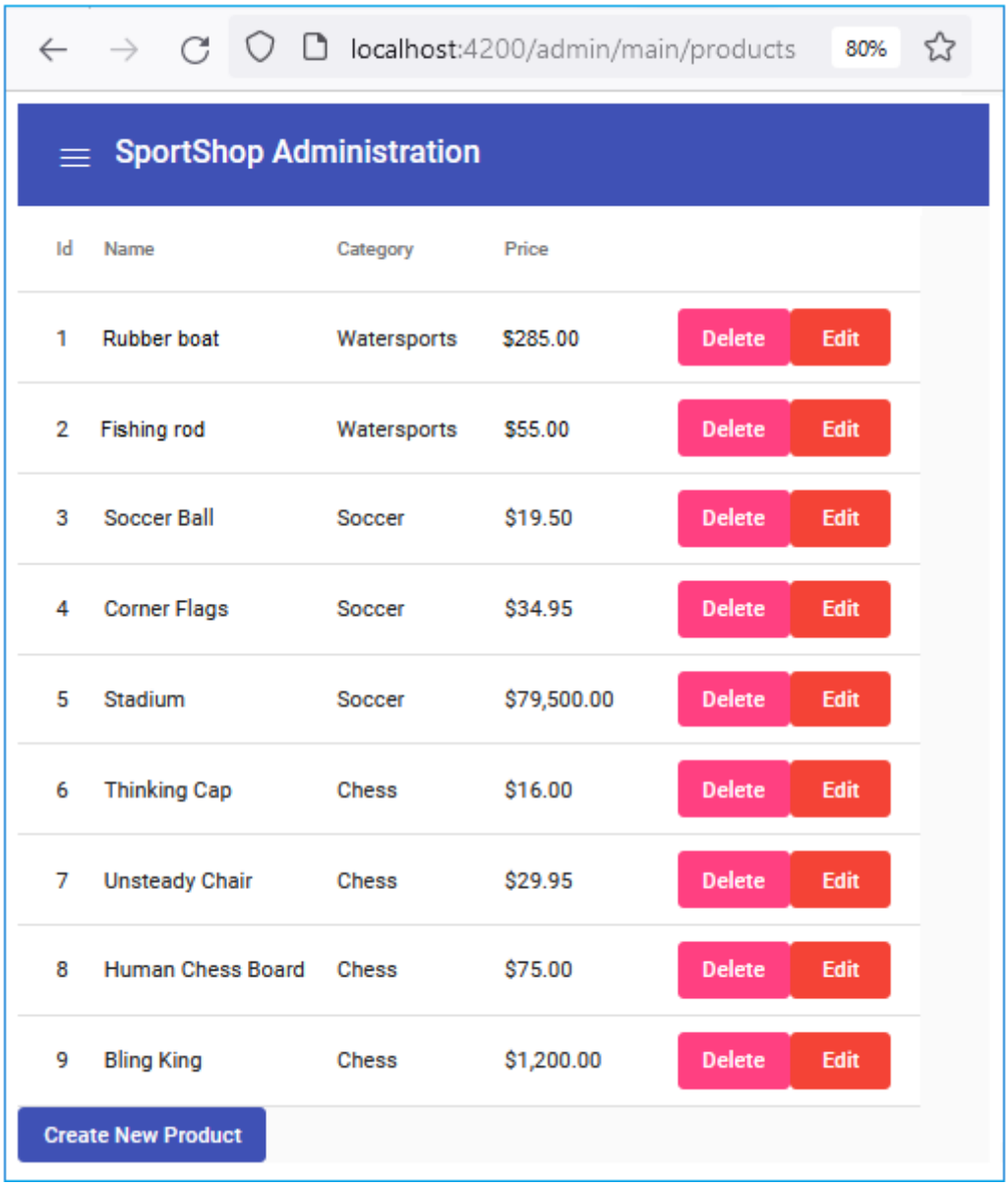

Рис. 10.5. Відображення таблиці товарів.

Використання таблиці товарів

Забезпечення роботи бібліотеки Angular Material може вимагати зусиль, але після завершення початкової роботи стає відносно просто скористатися перевагами додаткових функцій, які надаються. У випадку таблиці Angular Material включає додаткові функції фільтрації, сортування та розбиття даних на сторінки. Продемонструємо роботу функції розбивки на сторінки. Першим кроком є додавання функції розбиття на сторінки до програми, як показано в лістингу 10.32.

Лістинг 10.32. Додавання функції у файл material.module.ts папки src/app/admin

```
import { NgModule } from "@angular/core";
import { MatToolbarModule } from "@angular/material/toolbar";
import { MatSidenavModule } from "@angular/material/sidenav";
import { MatIconModule } from '@angular/material/icon';
import { MatDividerModule } from '@angular/material/divider';
import { MatButtonModule } from "@angular/material/button";
import { MatTableModule } from "@angular/material/table";
import { MatPaginatorModule } from "@angular/material/paginator";
```

```
const features: any[] = [MatToolbarModule,MatSidenavModule,MatIconModule,
        MatDividerModule, MatButtonModule, MatTableModule,
MatPaginatorModule];
@NgModule({
        imports: [features],
        exports: [features]
})
export class MaterialFeatures {}
```

Наступним кроком є додавання пагінатора до компонента, який відображає таблицю, як показано в лістингу 10.33.

Лістинг 10.33. Додавання розбиття на сторінки у файлі productTable.component.html папки src/app/admin

```
<table mat-table [dataSource]="dataSource">
    <mat-text-column name="id"></mat-text-column>
    <mat-text-column name="name"></mat-text-column>
    <mat-text-column name="category"></mat-text-column>
    <ng-container matColumnDef="price">
        <th mat-header-cell *matHeaderCellDef>Price</th>
        <td mat-cell *matCellDef="let item"> {{item.price | currency:"USD"}} </td>
    </ng-container>
    <ng-container matColumnDef="buttons">
        <th mat-header-cell *matHeaderCellDef></th>
        <td mat-cell *matCellDef="let p">
            <button mat-flat-button color="accent"
                (click)="deleteProduct(p.id)">
                Delete
            </button>
            <button mat-flat-button color="warn"
                [routerLink]="['/admin/main/products/edit', p.id]">
                Edit
            </button>
    </ng-container>
    <tr mat-header-row *matHeaderRowDef="colsAndRows"></tr>
    <tr mat-row *matRowDef="let row; columns: colsAndRows"></tr>
</table>
<div class="bottom-box">
  <button mat-flat-button color="primary"
routerLink="/admin/main/products/create">
        Create New Product
  </button>
  <mat-paginator [pageSize]="5" [pageSizeOptions]="[3, 5, 10]">
  </mat-paginator>
</div>
```

Пагінатор має бути пов’язаний із джерелом даних, яке використовується таблицею, яка встраюється в компонент, як показано в лістингу 10.34.

Лістинг 10.34. Підключення Paginator у файлі productTable.component.ts папки src/app/admin

```
import { Component, IterableDiffer, IterableDiffers, ViewChild } from
"@angular/core";
import { MatTableDataSource } from "@angular/material/table";
import { Product } from "../model/product.model";
import { ProductRepository } from "../model/product.repository";
```

```
import { MatPaginator } from "@angular/material/paginator";
@Component({
    templateUrl: "productTable.component.html"
})
export class ProductTableComponent {
    colsAndRows: string[] = ['id', 'name', 'category', 'price', 'buttons'];
    dataSource = new MatTableDataSource<Product>(this.repository.getProducts());
    differ: IterableDiffer<Product>;

    @ViewChild(MatPaginator)
    paginator? : MatPaginator

    constructor(private repository: ProductRepository, differs: IterableDiffers) {
        this.differ = differs.find(this.repository.getProducts()).create();
    }
    ngDoCheck() {
        let changes = this.differ?.diff(this.repository.getProducts());
        if (changes != null) {
            this.dataSource.data = this.repository.getProducts();
        }
    }
    ngAfterViewInit() {
        if (this.paginator) {
            this.dataSource.paginator = this.paginator;
        }
    }
    deleteProduct(id: number) {
        this.repository.deleteProduct(id);
    }
}
```

Декоратор ViewChild використовується для запиту вмісту шаблону компонента і використовується тут для пошуку компонента пагінатора. Метод ngAfterViewInit викликається після того, як Angular завершить обробку шаблону, а до цього часу компонент пагінатора створюється та пов'язується з джерелом даних. І, звичайно, деякі додаткові стилі CSS потрібні для керування макетом, як показано в лістингу 10.35.

Лістинг 10.35. Визначення стилів у файлі styles.css у папці src

```
html, body { height: 100%}
body { margin: 0; font-family: Roboto, "Helvetica Neue", sans-serif; }
mat-toolbar span { flex: 1 1 auto; }
.menu-button { width: 100%; font-size: 1rem; }
.menu-button .mat-icon { margin-right: 10px; }
.menu-button span { flex: 1 1 auto; }
mat-sidenav { margin: 16px; width: 175px; border-right: none;
    border-radius: 4px; padding: 4px;
}
mat-sidenav .mat-divider { margin-top: 20px; margin-bottom: 5px; }
mat-sidenav-container { height: calc(100vh - 60px); }
mat-sidenav .mat-button-wrapper {
    display: flex; width: 100%; justify-content: baseline; align-content: center;
}
mat-sidenav .mat-button-wrapper mat-icon { margin-top: 5px; }
mat-sidenav .mat-button-wrapper span { text-align: start; }
table[mat-table] { width: 100%; table-layout: auto; }
table[mat-table] button { margin-left: 5px;}
```

```
table[mat-table] th.mat-header-cell { font-size: large; font-weight: bold;}
table[mat-table] .mat-column-name { width: 25%; }
table[mat-table] .mat-column-buttons { width: 30%; }
/* table[mat-table] + button[mat-flat-button] { margin-top: 10px;} */

.bottom-box { background-color: white; padding-bottom: 20px;}
.bottom-box > button[mat-flat-button] { margin-top: 10px;}
.bottom-box mat-paginator { float: right; font-size: 14px; }
```

Після збереження змін можна побачити, що початкова робота з адаптації програми для роботи в моделі, очікуваній Angular Material, виконана і тепер можна використовувати вбудовану підтримку розбиття на сторінки, як показано на рис. 10.6.

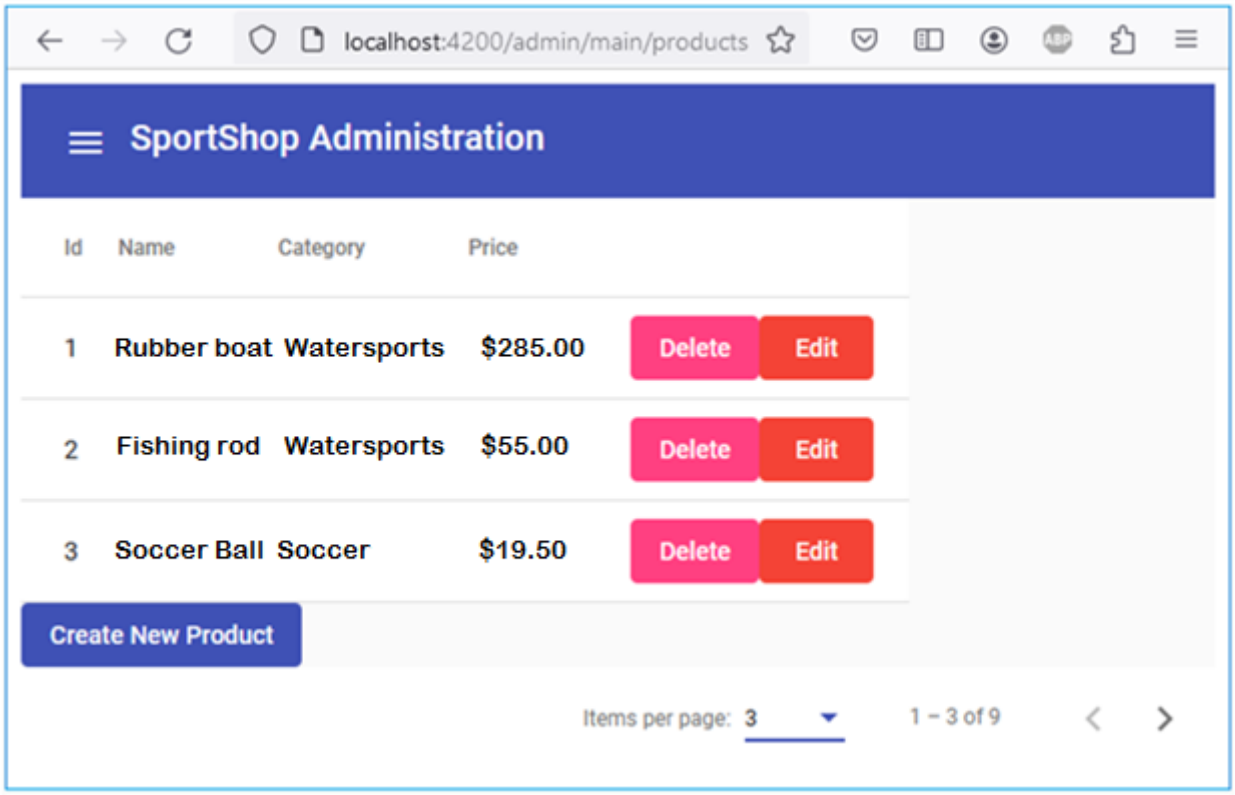

Рис. 10.6. Використання пагінатора в таблиці Angular Material

Впровадження редактора продуктів

Компоненти можуть отримувати інформацію про поточну URL-адресу маршрутизації та відповідним чином адаптувати свою поведінку. Компонент редактора має використовувати цю функцію, щоб розрізняти запити на створення нового компонента та редагування існуючого. Лістинг 10.36 додає функціональні можливості до компонента редактора, необхідні для створення або редагування продуктів.

Лістинг 10.36. Додавання функціональних можливостей у файл productEditor.component.ts папки src/app/admin

```
import { Component } from "@angular/core";
import { Router, ActivatedRoute } from "@angular/router";
import { Product } from "../model/product.model";
import { ProductRepository } from "../model/product.repository";
@Component({
    templateUrl: "productEditor.component.html"
})
export class ProductEditorComponent {
```

```
    editing: boolean = false;
    product: Product = new Product();
    constructor(private repository: ProductRepository,
            private router: Router,
            activeRoute: ActivatedRoute) {

        this.editing = activeRoute.snapshot.params["mode"] == "edit";
        if (this.editing) {
            Object.assign(this.product,
                repository.getProduct(activeRoute.snapshot.params["id"]));
        }
    }

    save() {
        this.repository.saveProduct(this.product);
        this.router.navigateByUrl("/admin/main/products");
    }
}
```

Angular надає об'єкт ActivatedRoute як аргумент конструктора, коли він створює новий екземпляр класу компонента, і цей об'єкт можна використовувати для перевірки активованого маршруту. У цьому випадку компонент визначає, чи слід редагувати чи створювати продукт, і, якщо редагує, отримує поточні деталі зі сховища. Існує також метод збереження, який використовує репозиторій для збереження змін, внесених користувачем. Форма HTML використовуватиметься для того, щоб користувач міг редагувати продукти. Пакет Angular Material забезпечує підтримку полів форми. Лістинг 10.37 додає ці функції до програми SportShop.

Лістинг 10.37. Додавання функції у файл material.module.ts папки src/app/admin

```
import { NgModule } from "@angular/core";
import { MatToolbarModule } from "@angular/material/toolbar";
import { MatSidenavModule } from "@angular/material/sidenav";
import { MatIconModule } from '@angular/material/icon';
import { MatDividerModule } from '@angular/material/divider';
import { MatButtonModule } from "@angular/material/button";
import { MatTableModule } from "@angular/material/table";
import { MatPaginatorModule } from "@angular/material/paginator";
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatInputModule } from '@angular/material/input';

const features: any[] = [MatToolbarModule,MatSidenavModule,MatIconModule,
        MatDividerModule, MatButtonModule, MatTableModule,
MatPaginatorModule,
        MatFormFieldModule, MatInputModule];
@NgModule({
        imports: [features],
        exports: [features]
})
export class MaterialFeatures {}
```

Щоб визначити шаблон із формою, додайте файл під назвою productEditor.component.html у папку src/app/admin і додайте розмітку, показану в лістингу 10.38.

Лістинг 10.38. Вміст файлу productEditor.component.html папки src/app/admin

```
<h3 class="heading">{{editing ? "Edit" : "Create"}} Product</h3>
<form (ngSubmit)="save()">
    <mat-form-field *ngIf="editing">
        <span>ID</span>
        <input  class="form-control" name="id" [(ngModel)]="product.id" disabled />
    </mat-form-field>
    <mat-form-field>
        <span>Name</span>
        <input  class="form-control" name="name" [(ngModel)]="product.name" />
    </mat-form-field>
    <mat-form-field>
        <span>Category</span>
        <input class="form-control" name="category" [(ngModel)]="product.category" />
    </mat-form-field>
    <mat-form-field>
        <span>Description</span>
        <input class="form-control" name="description"
[(ngModel)]="product.description"/>
    </mat-form-field>
    <mat-form-field>
        <span>Price</span>
        <input class="form-control" name="price" [(ngModel)]="product.price" />
    </mat-form-field>
    <div>
        <button  type="submit" mat-flat-button color="primary">
            {{editing ? "Save" : "Create"}}
        </button>
        <button type="reset" mat-stroked-button routerLink="/admin/main/products">
            Cancel
        </button>
    </div>
</form>
```

Шаблон містить форму з полями для властивостей, визначених класом моделі Product. Поле для властивості id відображається лише під час редагування наявного продукту та неактивне, оскільки значення не можна змінити. У лістингу 10.39 визначено стилі, необхідні для компонування форми.

Лістинг 10.39. Визначення стилів у файлі styles.css папки src

```
html, body { height: 100%}
body { margin: 0; font-family: Roboto, "Helvetica Neue", sans-serif; }
mat-toolbar span { flex: 1 1 auto; }
.menu-button { width: 100%; font-size: 1rem; }
.menu-button .mat-icon { margin-right: 10px; }
.menu-button span { flex: 1 1 auto; }
mat-sidenav { margin: 16px; width: 175px; border-right: none;
    border-radius: 4px; padding: 4px;
}
mat-sidenav .mat-divider { margin-top: 20px; margin-bottom: 5px; }
mat-sidenav-container { height: calc(100vh - 60px); }
mat-sidenav .mat-button-wrapper {
    display: flex; width: 100%; justify-content: baseline; align-content: center;
}
mat-sidenav .mat-button-wrapper mat-icon { margin-top: 5px; }
mat-sidenav .mat-button-wrapper span { text-align: start; }
table[mat-table] { width: 100%; table-layout: auto; }
```

```
table[mat-table] button { margin-left: 5px;}
table[mat-table] th.mat-header-cell { font-size: large; font-weight: bold;}
table[mat-table] .mat-column-name { width: 25%; }
table[mat-table] .mat-column-buttons { width: 30%; }
/* table[mat-table] + button[mat-flat-button] { margin-top: 10px;} */
.bottom-box { background-color: white; padding-bottom: 20px;}
.bottom-box > button[mat-flat-button] { margin-top: 10px;}
.bottom-box mat-paginator { float: right; font-size: 14px; }
mat-form-field { width: 100%;}
mat-form-field:first-child { margin-top: 20px;}
form button[mat-flat-button] { margin-top: 10px; margin-right: 10px;}
h3.heading { margin-top: 20px; }
```

Щоб побачити, як працює компонент, пройдіть аутентифікацію для доступу до функцій адміністратора та натисніть кнопку «Create New Product», яка з'явиться під таблицею продуктів. Заповніть форму, натисніть кнопку «Create», і новий продукт буде надіслано до веб-служби RESTful, де йому буде призначено властивість ID і відображено в таблиці продуктів, як показано на рис. 10.7. Може знадобитись перезапустити команду ng serve, якщо з'явиться помилка після збереження цих змін.

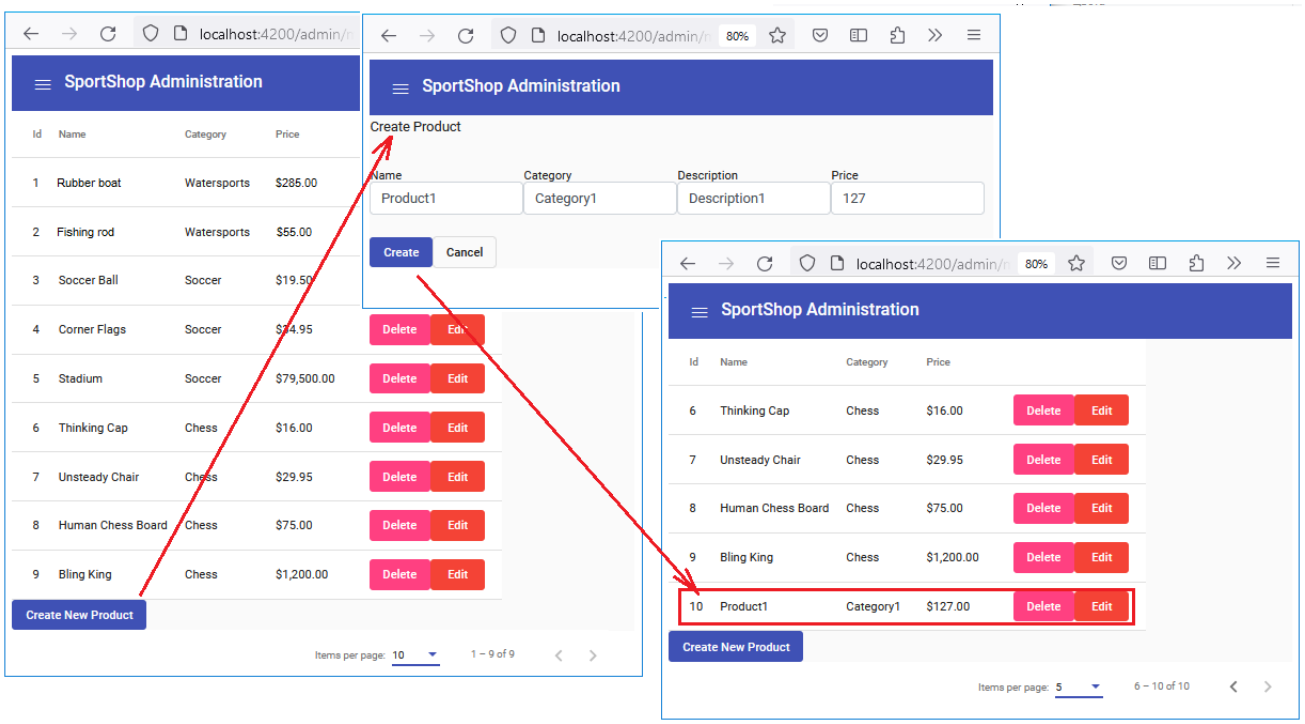

Рис. 10.7. Створення нового продукту

Процес редагування працює подібним чином. Натисніть одну з кнопок "Edit", щоб переглянути поточні деталі та мати змогу редагувати продукт за допомогою полів форми та натисніть кнопку «Save», щоб зберегти зміни, як показано на рис. 10.8.

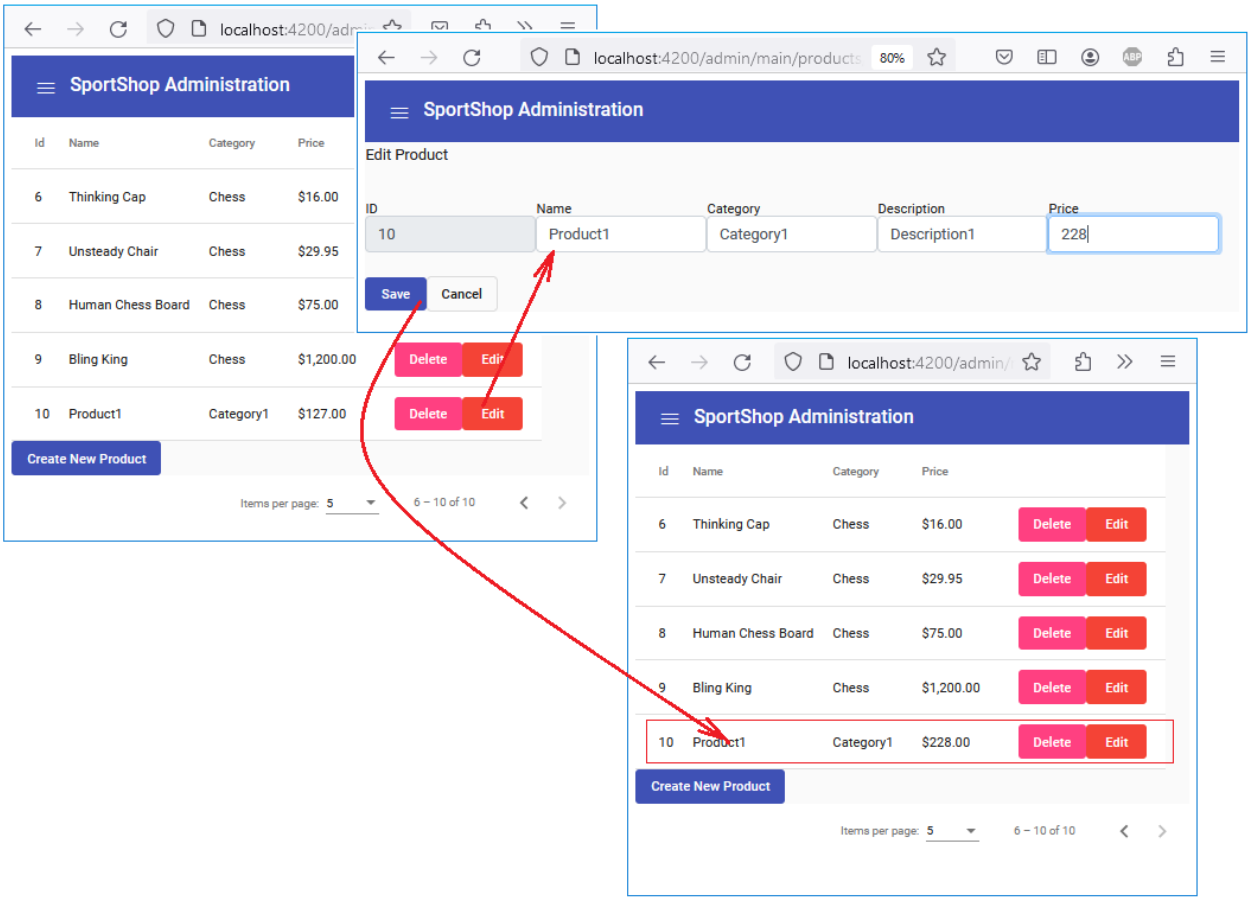

Рис. 10.8. Редагування існуючого продукту

Реалізація функцій таблиці замовлень

Функція керування замовленнями проста. Для цього потрібна таблиця зі списком набору замовлень разом із кнопками, які встановлюють властивість відправлення або повністю видаляють замовлення. Таблиця буде відображена з прапорцем, який міститиме відправлені замовлення в таблиці. Лістинг 10.40 додає функцію прапорця Angular Material до проекту SportShop.

Лістинг 10.40. Додавання функції у файл material.module.ts у папці src/app/admin

```
import { NgModule } from "@angular/core";
import { MatToolbarModule } from "@angular/material/toolbar";
import { MatSidenavModule } from "@angular/material/sidenav";
import { MatIconModule } from '@angular/material/icon';
import { MatDividerModule } from '@angular/material/divider';
import { MatButtonModule } from "@angular/material/button";
import { MatTableModule } from "@angular/material/table";
import { MatPaginatorModule } from "@angular/material/paginator";
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatInputModule } from '@angular/material/input';
import { MatCheckboxModule } from '@angular/material/checkbox';

const features: any[] = [MatToolbarModule,MatSidenavModule,MatIconModule,
        MatDividerModule,  MatButtonModule,  MatTableModule,
MatPaginatorModule,
        MatFormFieldModule, MatInputModule, MatCheckboxModule];

@NgModule({
        imports: [features],
```

```
        exports: [features]
    })
    export class MaterialFeatures {}
```

Щоб створити шаблон, додамо файл під назвою orderTable.component.html до папки src/app/admin із вмістом, показаним у лістингу 10.41.

Лістинг 10.41. Вміст файлу orderTable.component.html папки src/app/admin

```
<mat-checkbox [(ngModel)]="includeShipped">Display Shipped Orders</mat-checkbox>
<table class="orders" mat-table [dataSource]="dataSource">
    <mat-text-column name="name"></mat-text-column>
    <mat-text-column name="zip"></mat-text-column>
    <ng-container matColumnDef="cart_p">
        <th mat-header-cell *matHeaderCellDef></th>
        <td mat-cell *matCellDef="let order">
            <table mat-table [dataSource]="order.cart.lines">
                <ng-container matColumnDef="p">
                    <th mat-header-cell *matHeaderCellDef>Product</th>
                    <td mat-cell *matCellDef="let line">{{ line.product.name }}</td>
                </ng-container>
                <tr mat-header-row *matHeaderRowDef="['p']"></tr>
                <tr mat-row *matRowDef="let row; columns: ['p']"></tr>
            </table>
        </td>
    </ng-container>
<ng-container matColumnDef="cart_q">
    <th mat-header-cell *matHeaderCellDef></th>
    <td mat-cell *matCellDef="let order">
        <table mat-table [dataSource]="order.cart.lines">
            <ng-container matColumnDef="q">
                <th mat-header-cell *matHeaderCellDef>Quantity</th>
                <td mat-cell *matCellDef="let line">{{ line.quantity }}</td>
            </ng-container>
            <tr mat-header-row *matHeaderRowDef="['q']"></tr>
            <tr mat-row *matRowDef="let row; columns: ['q']"></tr>
        </table>
    </td>
</ng-container>
<ng-container matColumnDef="buttons">
    <th mat-header-cell *matHeaderCellDef>Actions</th>
    <td mat-cell *matCellDef="let o">
        <button mat-flat-button color="primary" (click)="toggleShipped(o)">
            {{ o.shipped ? "Unship" : "Ship" }}
        </button>
        <button mat-flat-button color="warn" (click)="delete(o.id)">
            Delete
        </button>
</ng-container>
<tr mat-header-row *matHeaderRowDef="colsAndRows"></tr>
<tr mat-row *matRowDef="let row; columns: colsAndRows"></tr>
<tr class="mat-row" *matNoDataRow>
    <td class="mat-cell no-data" colspan="4">No orders to display</td>
</tr>
</table>
```

Цей шаблон містить таблицю всередині таблиці, що дозволяє створювати змінну кількість рядків для кожного замовлення, що відображає вибір продукту клієнта. Існує також прапорець, який встановлює властивість під назвою includeShipped, яка визначена в лістингу 10.42, а також інші функції, необхідні для підтримки шаблону.

Лістинг 10.42. Додавання функцій у файл orderTable.component.ts папки src/app/admin

```
import { Component, IterableDiffer, IterableDiffers } from "@angular/core";
import { MatTableDataSource } from "@angular/material/table";
import { Order } from "../model/order.model";
import { OrderRepository } from "../model/order.repository";
@Component({
    templateUrl: "orderTable.component.html"
})
export class OrderTableComponent {
    colsAndRows: string[] = ['name', 'zip','cart_p','cart_q', 'buttons'];
    dataSource = new MatTableDataSource<Order>(this.repository.getOrders());
    differ: IterableDiffer<Order>;
    constructor(private repository: OrderRepository, differs: IterableDiffers) {
        this.differ = differs.find(this.repository.getOrders()).create();
        this.dataSource.filter = "true";
        this.dataSource.filterPredicate = (order, include) => {
            return !order.shipped || include.toString() == "true"
        };
    }
    get includeShipped(): boolean {
        return this.dataSource.filter == "true";
    }
    set includeShipped(include: boolean) {
        this.dataSource.filter = include.toString()
    }
    toggleShipped(order: Order) {
        order.shipped = !order.shipped;
        this.repository.updateOrder(order);
    }
    delete(id: number) {
        this.repository.deleteOrder(id);
    }
    ngDoCheck() {
        let changes = this.differ?.diff(this.repository.getOrders());
        if (changes != null) {
            this.dataSource.data = this.repository.getOrders();
        }
    }
}
```

Те, як дані фільтруються в цьому прикладі, є гарним прикладом адаптації до того, як працює бібліотека компонентів. Підтримка фільтрації даних, надана таблицею Angular Material, призначена для пошуку рядків у таблиці та оновлення відфільтрованих даних лише тоді, коли вказано новий пошуковий рядок. Функція фільтрації рядків в таблиці застосовується, прив’язуючи зміни до прапорця, щоб дані в таблиці змінювались в залежності від того, виконане замовлення чи ні. Останнім кроком є визначення додаткового CSS для керування макетом таблиці та її вмістом, як показано в лістингу 10.43.

Лістинг 10.43. Визначення стилів у файлі styles.css папки src

```
html, body { height: 100%}
body { margin: 0; font-family: Roboto, "Helvetica Neue", sans-serif; }
mat-toolbar span { flex: 1 1 auto; }
.menu-button { width: 100%; font-size: 1rem; }
.menu-button .mat-icon { margin-right: 10px; }
.menu-button span { flex: 1 1 auto; }
mat-sidenav { margin: 16px; width: 175px; border-right: none;
    border-radius: 4px; padding: 4px;
}
mat-sidenav .mat-divider { margin-top: 20px; margin-bottom: 5px; }
mat-sidenav-container { height: calc(100vh - 60px); }
mat-sidenav .mat-button-wrapper {
    display: flex; width: 100%; justify-content: baseline; align-content: center;
}
mat-sidenav .mat-button-wrapper mat-icon { margin-top: 5px; }
mat-sidenav .mat-button-wrapper span { text-align: start; }
table[mat-table] { width: 100%; table-layout: auto; }
table[mat-table] button { margin-left: 5px;}
table[mat-table] th.mat-header-cell { font-size: large; font-weight: bold;}
table[mat-table] .mat-column-name { width: 25%; }
table[mat-table] .mat-column-buttons { width: 30%; }
/* table[mat-table] + button[mat-flat-button] { margin-top: 10px;} */
.bottom-box { background-color: white; padding-bottom: 20px;}
.bottom-box > button[mat-flat-button] { margin-top: 10px;}
.bottom-box mat-paginator { float: right; font-size: 14px; }
mat-form-field { width: 100%;}
mat-form-field:first-child { margin-top: 20px;}
form button[mat-flat-button] { margin-top: 10px; margin-right: 10px;}
h3.heading { margin-top: 20px; }

mat-checkbox { margin: 10px; font-size: large;}
td.no-data { font-size: large;}
table.orders tbody, table.orders thead { vertical-align: top }
table.orders td { padding-top: 10px;}
table.orders table th:first-of-type, table.orders table td:first-of-type {
    margin: 0; padding: 0;
}
```

При цьому треба пам’ятати, що дані, представлені веб-сервісом RESTful, перезавантажуються щоразу, коли починається процес, а це означає, що доведеться використовувати кошик для покупок і перевіряти його, щоб створювати замовлення. Після цього можна буде перевіряти та керувати замовленнями за допомогою розділу «Orders» інструмента адміністрування, як показано на рис. 10.9.

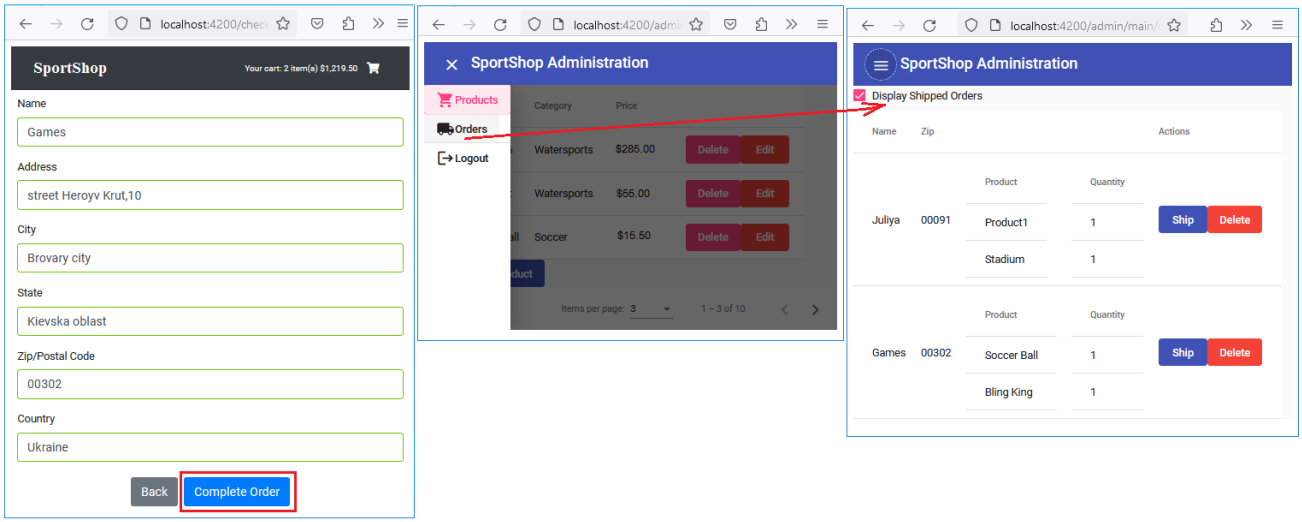

Рис. 10.9. Управління замовленнями

У цій лабораторній роботі було створено динамічно завантажуваний функціональний модуль Angular, який містить інструменти адміністрування необхідні для ведення каталогу товарів і обробки замовлень.

Завдання для самостійного виконання

Розробити додаткову логіку в компоненті orderTable.component.ts, щоб можна було фільтрувати замовлення по першій літері імені замовника (див. рис. 10.10).

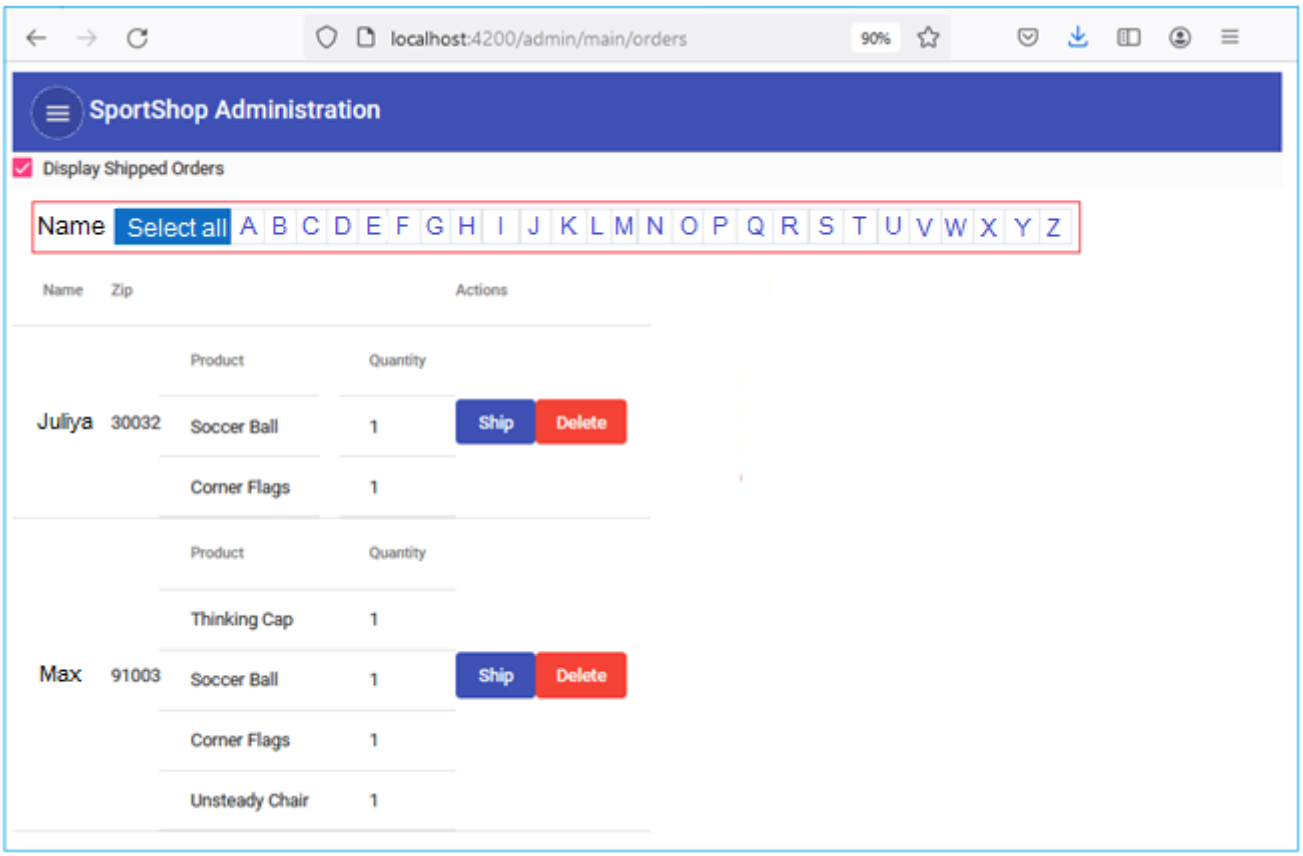

Рис. 10.10.

IV) Завдання для звіту по роботі

Основна частина звіту повинна мати не менше 15 сторінок тексту з рисунками (шрифт Times New Roman, 14, полуторний інтервал). Титульний аркуш приводиться у додатку 1. Основна частина звіту повинна містити наступні розділи:

a) опис модуля admin.module.ts з відповідною маршрутизацією для відображення компонентів з адміністрування;

b) специфікація JWT: основні поняття;

c) опис основних методів джерела даних, що використовуються при роботі адміністратора магазину «SportShop»;

d) опис основних методів репозиторія продуктів, що використовуються при роботі адміністратора магазину «SportShop»;

e) бібліотека Angular Material: основні поняття, призначення. Опис використовуваних в додатку функцій бібліотеки;

g) реалізація меню. Опис функцій таблиці товарів та таблиці замовлень магазину «SportShop».

V) Роботу отриманого додатку продемонструвати в режимі відеоконференції.

ДОДАТОК 1

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт по лабораторній роботі №____

_____

назва лабораторної роботи

з дисципліни: «Реактивне програмування»

Студент: _____

Група: _____

Дата захисту роботи: _____

Викладач: доц. Полупан Юлія Вікторівна

Захищено з оцінкою: _____

Київ, 2024