НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО

Факультет інформатики та обчислювальної техніки Кафедра інформатики та програмної інженерії

Звіт по лабораторній роботі №5

«Створення проекту «SportShop».

Розробка основних структурних блоків та моделі даних. Вибір товарів та оформлення замовленнь.»

роботи з дисципліни: «Реактивне програмування»

Студент: Мєшков Андрій Ігорович

Група: ІП-15

Дата захисту роботи: 14 «грудня» 2024

Викладач: доц. Полупан Юлія Вікторівна

Захищено з оцінкою: _____

Київ, 2024

ЗМІСТ

Огляд моделі даних Angular-додатку SportShop 3

Огляд фіктивного джерела даних у додатку..... 4

Детальний огляд компонентів магазину та шаблонів..... 7

Огляд існуючих компонентів в додатку. Призначення, використання..... 12

Огляд існуючих сервісів в додатку. Призначення, використання..... 18

Огляд існуючих директив в додатку. Призначення, використання 26

Можливі способи фільтрації товарів у додатку за категоріями 30

Реалізація пагінації на головній сторінці магазину (без директиви та з директивою)..... 33

Огляд моделі кошика у додатку, його призначення 35

Огляд компонентів, що призначені для роботи з кошиком 38

Наявна маршрутизація у додатку 44

Реалізація роботи з замовленнями через джерело данних та репозиторій. 47

REST-сумісні веб-служби: призначення, використання 51

ВИСНОВКИ..... 55

СПИСОК ДЖЕРЕЛ 57

Огляд моделі даних Angular-додатку SportShop

Модель даних Angular-додатку SportShop створена для управління інформацією про товари, які представлені у спортивному інтернет-магазині, а також для обробки замовлень користувачів.

Клас Product забезпечує структуру для збереження інформації про товари, включаючи такі властивості як:

id – унікальний ідентифікатор продукту;

name – назва продукту;

category – категорія, до якої належить продукт;

description – опис продукту;

price – ціна продукту.

Клас розташовується у файлі product.model.ts в папці SportShop/src/app/model і виглядає наступним чином:

```
export class Product {
    constructor(
        public id?: number,
        public name?: string,
        public category?: string,
        public description?: string,
        public price?: number) { }
}
```

Огляд фіктивного джерела даних у додатку

Фіктивне джерело даних (static data source) у додатку SportShop реалізує модель роботи з даними без необхідності з'єднання з реальним сервером. Це зручний інструмент для тестування і розробки функціональності додатку.

Клас StaticDataSource відповідає за симуляцію джерела даних для роботи з товарами та замовленнями. Він визначений як Angular-сервіс із використанням декоратора @Injectable().

Властивість **products** - Містить список продуктів у форматі масиву об'єктів класу Product. У прикладі визначено 15 об'єктів, поділених на три категорії.

Метод **getProducts** - Повертає всі продукти як об'єкт Observable<Product[]>, що спрощує їх асинхронне використання в додатку. Це відповідає підходу RxJS.

Метод **saveOrder** - Використовується для симуляції збереження замовлення. Він приймає об'єкт класу Order і повертає його через Observable<Order>. Для зручності тестування метод додає консольний лог з деталями замовлення.

/src/app/model/static.datasource.ts

```
import { Injectable } from "@angular/core";
import { Product } from "./product.model";
import { Observable, from } from "rxjs";
import { Order } from "./order.model";
@Injectable()
export class StaticDataSource {
    private products: Product[] = [
        new Product(1, "Product 1", "Category 1", "Product 1 (Category 1)", 100),
        new Product(2, "Product 2", "Category 1", "Product 2 (Category 1)", 100),
        new Product(3, "Product 3", "Category 1", "Product 3 (Category 1)", 100),
        new Product(4, "Product 4", "Category 1", "Product 4 (Category 1)", 100),
```

```
        new Product(5, "Product 5", "Category 1", "Product 5 (Category 1)", 100),
        new Product(6, "Product 6", "Category 2", "Product 6 (Category 2)", 100),
        new Product(7, "Product 7", "Category 2", "Product 7 (Category 2)", 100),
        new Product(8, "Product 8", "Category 2", "Product 8 (Category 2)", 100),
        new Product(9, "Product 9", "Category 2", "Product 9 (Category 2)", 100),
        new Product(10, "Product 10", "Category 2", "Product 10 (Category 2)",
100),
        new Product(11, "Product 11", "Category 3", "Product 11 (Category 3)",
100),
        new Product(12, "Product 12", "Category 3", "Product 12 (Category 3)",
100),
        new Product(13, "Product 13", "Category 3", "Product 13 (Category 3)",
100),
        new Product(14, "Product 14", "Category 3", "Product 14 (Category 3)",
100),
        new Product(15, "Product 15", "Category 3", "Product 15 (Category 3)",
100),
    ];
    getProducts(): Observable<Product[]> {
        return from([this.products]);
    }
    saveOrder(order: Order): Observable<Order> {
        console.log(JSON.stringify(order));
        return from([order]);
    }
}
```

Клас ProductRepository є посередником між фіктивним джерелом даних StaticDataSource і компонентами додатку.

Властивість **products** - Зберігає масив продуктів, отриманих із StaticDataSource.

Властивість **categories** - Містить унікальні категорії продуктів. Список формується динамічно при завантаженні даних.

Метод **getProducts** - Фільтрує продукти за категоріями, якщо вона задана. Якщо параметр category не визначено, повертається весь список.

Метод **getProduct** - Знаходить продукт за унікальним id.

Метод **getCategories** - Повертає список категорій, попередньо відсортований.

src/app/model/product.repository.ts

```
import { Injectable } from "@angular/core";
import { Product } from "./product.model";
import { StaticDataSource } from "./static.datasource";
@Injectable()
export class ProductRepository {
    private products: Product[] = [];
    private categories: string[] = [];
    constructor(private dataSource: StaticDataSource) {
        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;
    }
}
```

При ініціалізації репозиторію (ProductRepository) дані завантажуються з StaticDataSource. Категорії формуються з поля category кожного продукту, дублікати виключаються.

Усі методи ProductRepository взаємодіють із підготовленими масивами (products і categories), дозволяючи отримувати, фільтрувати та обробляти дані.

Детальний огляд компонентів магазину та шаблонів

1. Компонент ShopComponent

Компонент ShopComponent реалізує основну функціональність відображення та фільтрації товарів, а також управління кошиком покупок у додатку SportShop.

Фільтрація товарів за категоріями

- Кнопки категорій формуються на основі списку, отриманого через метод getCategories класу ProductRepository.
- Кнопка Home скидає фільтр за категорією, відображаючи всі товари.

Посторінкова пагінація (pagination)

- Компонент дозволяє розділяти товари на сторінки відповідно до кількості товарів, зазначених у властивості productsPerPage.
- Метод get products виконує відбір товарів для відображення на поточній сторінці.

Зміна кількості товарів на сторінку

- Динамічне налаштування кількості товарів на сторінку реалізується методом changePageSize, який дозволяє вибір із заданого списку.

Додавання товарів до кошика

- Метод addProductToCart додає обраний товар у кошик і перенаправляє користувача на сторінку кошика.

2. HTML-шаблон shop.component.html

Шаблон компонента включає інтерфейс для вибору категорій, відображення товарів, налаштування пагінації та додавання товарів до кошика.

Основні секції шаблону:

1. Шапка сторінки Містить назву магазину та кнопку для переходу в кошик.
2. Меню категорій Відображає список доступних категорій. Обраній категорії додається клас active для візуальної індикації.
3. Список товарів Товари поточної категорії та сторінки відображаються у вигляді карток. Для кожного товару виводяться:
    - Назва
    - Ціна (форматована за допомогою currency пайпа)
    - Опис
    - Кнопка для додавання у кошик.
4. Налаштування пагінації
    - Кнопки переходу між сторінками формуються за допомогою директиви *counter.
    - Доступна зміна кількості товарів, що відображаються на сторінці, через випадаючий список.

3. Ключові аспекти реалізації

Декларативний стиль програмування Шаблони компонента реалізують декларативну логіку Angular для:

- Формування списків за допомогою *ngFor.
- Динамічного застосування стилів та класів.

RxJS та асинхронність - Дані з репозиторію продуктів завантажуються та обробляються асинхронно, що забезпечує ефективність роботи з фіктивним джерелом даних.

Модульність та повторне використання - Весь функціонал розподілений між класом ShopComponent і відповідними сервісами, що полегшує подальший розвиток додатку.

src/app/shop/shop.component.ts

```
import { Component } from "@angular/core";
import { Product } from "../model/product.model";
import { ProductRepository } from "../model/product.repository";
import { Cart } from "../model/cart.model";
import { Router } from "@angular/router";
@Component({
    selector: "shop",
    templateUrl: "./shop.component.html"
})
export class ShopComponent {
    selectedCategory: string | undefined;
    productsPerPage = 4;
    selectedPage = 1;
    constructor(private repository: ProductRepository,
        private cart: Cart,
        private router: Router) { }
    get products(): Product[] {
        let pageIndex = (this.selectedPage - 1) * this.productsPerPage
        return this.repository.getProducts(this.selectedCategory)
            .slice(pageIndex, pageIndex + this.productsPerPage);
    }
    get categories(): string[] {
        return this.repository.getCategories();
    }
    changeCategory(newCategory?: string) {
        this.selectedCategory = newCategory;
    }
    changePage(newPage: number) {
        this.selectedPage = newPage;
    }
    changePageSize(newSize: number) {
        this.productsPerPage = Number(newSize);
        this.changePage(1);
    }
    get pageCount(): number {
        return Math.ceil(this.repository
            .getProducts(this.selectedCategory).length /
            this.productsPerPage)
    }
    addProductToCart(product: Product) {
        this.cart.addLine(product);
        this.router.navigateByUrl("/cart");
    }
}
```

src/app/shop/shop.component.html

```
<div class="container-fluid">
    <div class="row">
```

```
        <div class="bg-dark text-white p-2">
            <span class="navbar-brand ml-2">
                SportShop
                <cart-summary></cart-summary>
            </span>
        </div>
    </div>
    <div class="row text-white">
        <div class="col-3 p-2">
            <div class="d-grid gap-2">
                <button class="btn btn-outline-primary"
                    (click)="changeCategory()">
                    Home
                </button>
                <button *ngFor="let cat of categories"
                    class="btn btn-outline-primary"
                    [class.active]="cat == selectedCategory"
                    (click)="changeCategory(cat)">
                    {{cat}}
                </button>
            </div>
        </div>
        <div class="col-9 p-2 text-dark">
            <div *ngFor="let product of products" class="card m-1 p-1 bg-light ">
                <h4>
                    {{product.name}}
                    <span class="badge rounded-pill bg-primary"
style="float:right">
                        {{ product.price | currency:"USD":"symbol":"2.2-2" }}
                    </span>
                </h4>
                <div class="card-text bg-white p-1">
                    {{product.description}}
                    <button class="btn btn-success btn-sm float-end"
                        (click)="addProductToCart(product)">
                        Add To Cart
                    </button>
                </div>
            </div>
            <div class="form-inline float-start mr-1">
                <select class="form-control" [value]="productsPerPage"
                    (change)="changePageSize($any($event).target.value)">
                    <option value="3">3 per Page</option>
                    <option value="4">4 per Page</option>
                    <option value="6">6 per Page</option>
                    <option value="8">8 per Page</option>
                </select>
            </div>
            <div class="btn-group float-end">
                <button *counter="let page of pageCount"
                    (click)="changePage(page)"
```

```
                    class="btn btn-outline-primary"
                    [class.active]="page == selectedPage">
                    {{page}}
                </button>
            </div>
        </div>
    </div>
</div>
```

Огляд існуючих компонентів в додатку. Призначення, використання

Додаток SportShop включає низку Angular-компонентів, кожен із яких відповідає за окремий функціонал магазину. Ось перелік основних компонентів із описом їхнього призначення та використання.

1. AppComponent

Призначення: Кореневий компонент додатка. Використовується для організації маршрутизації та відображення основних підкомпонентів.

Особливості:

- Використовує директиву <router-outlet> для динамічного відображення компонентів відповідно до поточного маршруту.
- Завантажується як головний компонент додатка.

src/app/app.component.ts

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

@Component({
  selector: 'app',
  template: "<router-outlet></router-outlet>",
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'SportShop';
}
```

2. ShopComponent

Призначення: Основний компонент магазину, який забезпечує користувачів можливістю переглядати товари, фільтрувати їх за категоріями, виконувати посторінкову пагінацію та додавати товари до кошика.

Особливості:

- Відображає список категорій і товарів.
- Реалізує посторінкову навігацію та керує розміром сторінки.
- Забезпечує інтерактивну взаємодію для вибору категорій і додавання товарів до кошика.

src/app/shop/shop.component.ts

```
import { Component } from "@angular/core";
import { Product } from "../model/product.model";
import { ProductRepository } from "../model/product.repository";
import { Cart } from "../model/cart.model";
import { Router } from "@angular/router";
@Component({
    selector: "shop",
    templateUrl: "./shop.component.html"
})
export class ShopComponent {
    selectedCategory: string | undefined;
    productsPerPage = 4;
    selectedPage = 1;
    constructor(private repository: ProductRepository,
        private cart: Cart,
        private router: Router) { }
    get products(): Product[] {
        let pageIndex = (this.selectedPage - 1) * this.productsPerPage
        return this.repository.getProducts(this.selectedCategory)
            .slice(pageIndex, pageIndex + this.productsPerPage);
    }
    get categories(): string[] {
        return this.repository.getCategories();
    }
    changeCategory(newCategory?: string) {
        this.selectedCategory = newCategory;
    }
    changePage(newPage: number) {
        this.selectedPage = newPage;
    }
    changePageSize(newSize: number) {
        this.productsPerPage = Number(newSize);
        this.changePage(1);
    }
    get pageCount(): number {
        return Math.ceil(this.repository
            .getProducts(this.selectedCategory).length /
            this.productsPerPage)
```

```
    }
    addProductToCart(product: Product) {
        this.cart.addLine(product);
        this.router.navigateByUrl("/cart");
    }
}
```

3. CartSummaryComponent

Призначення: Надає міні-огляд вмісту кошика, включаючи кількість доданих товарів і загальну вартість. Використовується, зокрема, у шапці магазину.

Особливості:

- Відображається як підкомпонент у межах інших компонентів (наприклад, у ShopComponent).

src/app/shop/cartSummary.component.ts

```
import { Component } from "@angular/core";
import { Cart } from "../model/cart.model";
@Component({
    selector: "cart-summary",
    templateUrl: "./cartSummary.component.html"
})
export class CartSummaryComponent {
    constructor(public cart: Cart) { }
}
```

4. CartDetailComponent

Призначення: Надає повний огляд кошика покупок. Дозволяє користувачам переглядати вибрані товари, змінювати їх кількість і видаляти товари.

Особливості:

- Забезпечує користувачів деталізованим оглядом їхніх вибраних покупок перед оформленням замовлення.

src/app/shop/cartDetail.component.ts

```
import { Component } from "@angular/core";
import { Cart } from "../model/cart.model";
@Component({
    templateUrl: "./cartDetail.component.html"
})
export class CartDetailComponent {
    constructor(public cart: Cart) {}
}
```

5. CheckoutComponent

Призначення: Реалізує форму для оформлення замовлення. Дозволяє вводити дані користувача та надсилати замовлення до сервісу.

Особливості:

- Використовує Angular Forms для роботи з формою.
- Інтерактивно обробляє помилки вводу та перевіряє валідність форми.
- Надсилає дані замовлення через OrderRepository.
- Показує повідомлення після успішного відправлення замовлення.

src/app/shop/checkout.component.ts

```
import { Component } from "@angular/core";
import { NgForm } from "@angular/forms";
import { OrderRepository } from "../model/order.repository";
import { Order } from "../model/order.model";
@Component({
    templateUrl: "./checkout.component.html",
    styleUrls: ["checkout.component.css"]
})
export class CheckoutComponent {
```

```
    orderSent: boolean = false;
    submitted: boolean = false;
    constructor(public repository: OrderRepository,
    public order: Order) {}
    submitOrder(form: NgForm) {
        this.submitted = true;
        if (form.valid) {
            this.repository.saveOrder(this.order).subscribe(order => {
            this.order.clear();
            this.orderSent = true;
            this.submitted = false;
            });
        }
    }
}
```

Загальні характеристики компонентів

Модульність і повторне використання: Кожен компонент виконує чітко визначене завдання, що забезпечує легкість підтримки та розширення додатку.

Зв'язок через сервіси: Компоненти обмінюються даними за допомогою сервісів (ProductRepository, Cart, OrderRepository), дотримуючись принципу залежності через ін’єкцію.

Використання маршрутизації:

- Кореневий компонент (AppComponent) управляє відображенням компонентів на основі маршрутів.
- Інші компоненти, такі як ShopComponent, CartDetailComponent, і CheckoutComponent, відображаються залежно від маршруту.

Фокус на UX: Компоненти реалізують зручний інтерфейс для взаємодії з магазином (категорії, кошик, фільтрація, пагінація тощо).

Використання компонентів

AppComponent: кореневий компонент, що відповідає за завантаження підкомпонентів через маршрути.

ShopComponent: використовується для виводу товарів, роботи з категоріями й додавання у кошик.

CartSummaryComponent: міні-презентація кошика, відображається у шапці магазину.

CartDetailComponent: доступ до деталей кошика перед оформленням замовлення.

CheckoutComponent: форма оформлення замовлення для завершення покупки.

Огляд існуючих сервісів в додатку. Призначення, використання

Додаток SportShop включає кілька сервісів, які забезпечують різноманітний функціонал, зокрема управління даними, взаємодію з API, логіку кошика і замовлень, а також маршрутизацію.

1. Сервіс Cart

Призначення: Відповідає за управління вмістом кошика користувача, включаючи додавання, видалення і зміну кількості товарів, а також обчислення підсумкових даних (кількість товарів, загальна ціна).

Особливості:

- Містить масив CartLine[], який представляє окремі лінії (товар + кількість).
- Обчислює загальну вартість (cartPrice) і загальну кількість товарів (itemCount).
- Методи:
    - addLine: додає товар до кошика (новий або змінює кількість існуючого).
    - updateQuantity: оновлює кількість товару.
    - removeLine: видаляє товар із кошика.
    - clear: очищає кошик.
    - recalculate: внутрішній метод для оновлення загальних підсумків.

Використання:

- Використовується в компонентах для роботи з кошиком, зокрема CartDetailComponent і CartSummaryComponent.

```
import { Injectable } from "@angular/core";
```

```
import { Product } from "./product.model";
@Injectable()
export class Cart {
    public lines: CartLine[] = [];
    public itemCount: number = 0;
    public cartPrice: number = 0;
    addLine(product: Product, quantity: number = 1) {
        let line = this.lines.find(line => line.product.id == product.id);
        if (line != undefined) {
            line.quantity += quantity;
        } else {
            this.lines.push(new CartLine(product, quantity));
        }
        this.recalculate();
    }
    updateQuantity(product: Product, quantity: number) {
        let line = this.lines.find(line => line.product.id == product.id);
        if (line != undefined) {
            line.quantity = Number(quantity);
        }
        this.recalculate();
    }
    removeLine(id: number) {
        let index = this.lines.findIndex(line => line.product.id == id);
        this.lines.splice(index, 1);
        this.recalculate();
    }
    clear() {
        this.lines = [];
        this.itemCount = 0;
        this.cartPrice = 0;
    }
    private recalculate() {
        this.itemCount = 0;
        this.cartPrice = 0;
        this.lines.forEach(l => {
            this.itemCount += l.quantity;
            this.cartPrice += l.lineTotal;
        })
    }
}

export class CartLine {
    constructor(public product: Product,
    public quantity: number) {}
    get lineTotal() {
        return this.quantity * (this.product.price ?? 0);
    }
}
```

2. Сервіс Order

Призначення: Забезпечує модель і основну логіку для оформлення замовлення, пов’язану із введенням даних користувача і взаємодією із кошиком.

Особливості:

- Має властивості для збереження даних про доставку (name, address, city, тощо).
- Зберігає стан замовлення через властивість shipped.
- Методи:
    - clear: очищує дані про замовлення і одночасно очищує кошик.

Використання:

- Використовується у CheckoutComponent для керування процесом оформлення замовлення.

```
import { Injectable } from "@angular/core";
import { Cart } from "./cart.model";
@Injectable()
export class Order {
    public id?: number;
    public name?: string;
    public address?: string;
    public city?: string;
    public state?: string;
    public zip?: string;
    public country?: string;
    public shipped: boolean = false;
    constructor(public cart: Cart) {}
    clear() {
        this.id = undefined;
        this.name = this.address = this.city = undefined;
        this.state = this.zip = this.country = undefined;
        this.shipped = false;
        this.cart.clear();
    }
}
```

3. Сервіс OrderRepository

Призначення: Забезпечує взаємодію із джерелами даних для отримання і збереження замовлень.

Особливості:

- Зберігає локальну копію замовлень у масиві orders.
- Використовує сервіс StaticDataSource для збереження замовлень.
- Методи:
  - getOrders: повертає масив збережених замовлень.
  - saveOrder: надсилає нове замовлення в джерело даних.

Використання:

- Застосовується у CheckoutComponent для збереження замовлення під час оформлення покупки.

```
import { Injectable } from "@angular/core";
import { Observable } from "rxjs";
import { Order } from "./order.model";
import { StaticDataSource } from "./static.datasource";
@Injectable()
export class OrderRepository {
    private orders: Order[] = [];
    constructor(private dataSource: StaticDataSource) {}
    getOrders(): Order[] {
        return this.orders;
    }
    saveOrder(order: Order): Observable<Order> {
        return this.dataSource.saveOrder(order);
    }
}
```

4. Сервіс StaticDataSource

Призначення: Фіктивне джерело даних, яке імітує взаємодію з сервером. Використовується для тестування і розробки.

Особливості:

- Визначає статичний масив товарів (products).
- Методи:
  - getProducts: повертає масив продуктів як спостережуваний об'єкт Observable<Product[]>.
  - saveOrder: імітує збереження замовлення і повертає його як Observable<Order>.

Використання:

- Застосовується у ProductRepository і OrderRepository як джерело даних.

```
import { Injectable } from "@angular/core";
import { Product } from "./product.model";
import { Observable, from } from "rxjs";
import { Order } from "./order.model";
@Injectable()
export class StaticDataSource {
    private products: Product[] = [
        new Product(1, "Product 1", "Category 1", "Product 1 (Category 1)", 100),
        new Product(2, "Product 2", "Category 1", "Product 2 (Category 1)", 100),
        new Product(3, "Product 3", "Category 1", "Product 3 (Category 1)", 100),
        new Product(4, "Product 4", "Category 1", "Product 4 (Category 1)", 100),
        new Product(5, "Product 5", "Category 1", "Product 5 (Category 1)", 100),
        new Product(6, "Product 6", "Category 2", "Product 6 (Category 2)", 100),
        new Product(7, "Product 7", "Category 2", "Product 7 (Category 2)", 100),
        new Product(8, "Product 8", "Category 2", "Product 8 (Category 2)", 100),
        new Product(9, "Product 9", "Category 2", "Product 9 (Category 2)", 100),
        new Product(10, "Product 10", "Category 2", "Product 10 (Category 2)",
100),
        new Product(11, "Product 11", "Category 3", "Product 11 (Category 3)",
100),
```

```
        new Product(12, "Product 12", "Category 3", "Product 12 (Category 3)",
100),
        new Product(13, "Product 13", "Category 3", "Product 13 (Category 3)",
100),
        new Product(14, "Product 14", "Category 3", "Product 14 (Category 3)",
100),
        new Product(15, "Product 15", "Category 3", "Product 15 (Category 3)",
100),
    ];
    getProducts(): Observable<Product[]> {
        return from([this.products]);
    }
    saveOrder(order: Order): Observable<Order> {
        console.log(JSON.stringify(order));
        return from([order]);
    }
}
```

5. Сервіс RestDataSource

Призначення: Реалізує взаємодію з RESTful API для отримання товарів і збереження замовлень.

Особливості:

- Визначає базову URL для роботи з API.
- Методи:
    - getProducts: отримує список товарів з API.
    - saveOrder: зберігає замовлення на сервері.

Використання:

- Може замінити StaticDataSource для роботи з реальним сервером під час розгортання додатка.

```
import { Injectable } from "@angular/core";
import { HttpClient } from "@angular/common/http";
import { 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;
    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);
    }
}
```

6. Сервіс ShopFirstGuard

Призначення: Реалізує логіку маршрутизації, яка забезпечує, щоб перша навігація в додатку завжди вела до магазину.

Особливості:

- Відстежує першу навігацію за допомогою властивості firstNavigation.
- Метод canActivate: перенаправляє користувача на головну сторінку магазину, якщо перший маршрут в додатку відрізняється від ShopComponent.

Використання:

- Використовується як гвардія для маршруту, який веде до магазину.

```
import { Injectable } from "@angular/core";
import { ActivatedRouteSnapshot, RouterStateSnapshot, Router } from
"@angular/router";
import { ShopComponent } from "./shop/shop.component";
@Injectable()
export class ShopFirstGuard {
```

```
    private firstNavigation = true;
    constructor(private router: Router) { }
    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean
{
        if (this.firstNavigation) {
            this.firstNavigation = false;
            if (route.component != ShopComponent) {
                this.router.navigateByUrl("/");
                return false;
            }
        }
        return true;
    }
}
```

Огляд існуючих директив в додатку. Призначення, використання

В додатку використовується директива **CounterDirective**, яка забезпечує динамічне рендеринг шаблонів на основі числового значення.

Призначення Директива створює визначену кількість вбудованих представлень (елементів DOM) на основі числового значення, переданого їй через властивість counterOf.

Особливості

1. **Декоратор @Directive:** Директива визначена за допомогою декоратора Angular @Directive, що забезпечує її інтеграцію в DOM через селектор [counterOf].
2. **Властивості:**
    - @Input("counterOf") counter: числове значення, яке визначає кількість елементів, що будуть створені.
    - container (типу ViewContainerRef): контейнер, у який будуть додаватися вбудовані представлення.
    - template (типу TemplateRef<Object>): шаблон, який створюватиме нові представлення.
3. **Методи:**
    - ngOnChanges: Викликається щоразу при зміні значення вхідної властивості counterOf. Директива:
        - Очищує контейнер від попередніх представлень (this.container.clear()).
        - Циклічно додає нові представлення у контейнер із переданим контекстом (CounterDirectiveContext).
4. **Контекст шаблону:**

- Додає контекст CounterDirectiveContext, який доступний у шаблоні через змінну $implicit.

Використання

1. **Шаблонний синтаксис:**

```
<div *counterOf="5; let idx">
  Item {{ idx }}
</div>
```

2. **Результат рендерингу:**

У контейнер буде динамічно додано 5 елементів div з відповідними індексами (1, 2, 3, 4, 5).

3. **Логіка:**

- Коли вказується counterOf="5", метод ngOnChanges викликається з новим значенням.
- Контейнер очищується.
- У шаблон послідовно створюються нові представлення, де індекс (починаючи з 1) доступний через $implicit.

Переваги та можливості

1. **Динамічний контент:** Директива дозволяє динамічно створювати вміст у залежності від вхідного числа.
2. **Повторення шаблонів:** Простий і декларативний спосіб рендерингу набору однакових елементів.

3. **Контекст індексу:** Значення $implicit (змінна контексту) дозволяє використовувати поточний індекс елементів, що особливо корисно для нумерації або прив’язки даних.

Типові сценарії використання

1. **Генерація списків:** Використовується для створення повторюваних блоків в шаблоні на основі заданої кількості.
2. **Робота із статичною кількістю елементів:** Може замінити використання статичного масиву з директивою *ngFor в певних випадках, спрощуючи структуру коду.
3. **Гнучке управління рендерингом:** Універсальний підхід для створення динамічного UI залежно від числового значення (наприклад, кількість кроків форми, кількість показаних результатів тощо).

src/app/shop/counter.directive.ts

```
import { Directive, ViewContainerRef, TemplateRef, Input, SimpleChanges} from
"@angular/core";
@Directive({
    selector: "[counterOf]"
})
export class CounterDirective {
    constructor(private container: ViewContainerRef,
    private template: TemplateRef<Object>) {}
    @Input("counterOf")
    counter: number = 0;
    ngOnChanges(changes: SimpleChanges) {
        this.container.clear();
        for (let i = 0; i < this.counter; i++) {
            this.container.createEmbeddedView(this.template,
            new CounterDirectiveContext(i + 1));
        }
    }
}
class CounterDirectiveContext {
```

```
    constructor(public $implicit: any) { }
}
```

Можливі способи фільтрації товарів у додатку за категоріями

У додатку використовується простий і ефективний механізм фільтрації товарів за категоріями, реалізований у компоненті **ShopComponent**.

Основні підходи до фільтрації:

1. Вибір категорії через кнопку

- **Призначення:** Дозволяє користувачеві обирати категорію товарів, натискаючи кнопку з назвою категорії.
- **Реалізація в шаблоні:**

```
<button *ngFor="let cat of categories"
    class="btn btn-outline-primary"
    [class.active]="cat == selectedCategory"
    (click)="changeCategory(cat)">
    {{cat}}
</button>
```

  - Використовується цикл *ngFor для генерації кнопок для кожної категорії.
  - Динамічне застосування класу active для обраної категорії.
  - Метод changeCategory(cat) оновлює вибрану категорію.

2. Повернення до "Home"

- **Призначення:** Користувач може скинути фільтр, повернувшись до перегляду всіх товарів.
- **Реалізація в шаблоні:**

```
<button class="btn btn-outline-primary"
    (click)="changeCategory()">
    Home
</button>
```

- Виклик методу changeCategory() з параметром undefined, щоб відобразити всі товари.

3. Динамічне визначення товарів за категоріями

- **Призначення:** Фільтрація на основі категорії та підготовка даних для відображення.
- **Реалізація в класі:**

```
get products(): Product[] {
    let pageIndex = (this.selectedPage - 1) * this.productsPerPage
    return this.repository.getProducts(this.selectedCategory)
        .slice(pageIndex, pageIndex + this.productsPerPage);
}
```

- Метод getProducts(category) у ProductRepository повертає товари, фільтровані за вибраною категорією.
- У методі обробляється пагінація — відображаються лише товари для поточної сторінки.

Додаткові можливості для покращення фільтрації:

1. Комбінована фільтрація

- Реалізація додаткових критеріїв фільтрації, таких як **ціна**, **бренд** чи **рейтинг**.
- Можливе використання селекторів (<select>), чекбоксів (<input type="checkbox">) або слайдерів.

2. Сортування товарів

- Можливість змінювати порядок відображення (наприклад, за зростанням/спаданням ціни).

```
<select (change)="changeSortOrder($event)">
    <option value="asc">Price: Low to High</option>
    <option value="desc">Price: High to Low</option>
</select>
```

3. Пошук всередині категорії

- Реалізація пошукового поля для пошуку товарів у вибраній категорії.

Переваги підходу

1. **Простота реалізації:** Нинішній механізм з використанням кнопок і методу changeCategory легко зрозумілий і достатньо ефективний для невеликого асортименту.
2. **Гнучкість:**
    - Вибір категорії можливий у будь-який момент.
    - Легке розширення для додавання нових критеріїв.
3. **Інтуїтивний інтерфейс:** Користувачам зрозуміло, як обирати або скидувати категорію.

Поточний механізм забезпечує базову, але функціональну фільтрацію товарів за категоріями. Для масштабування функціоналу можна додати додаткові методи фільтрації, такі як комбіновані критерії або пошук, що зробить роботу з товарами ще зручнішою.

Реалізація пагінації на головній сторінці магазину (без директиви та з директивою)

Реалізація без директиви:

1. **Посторінне виведення**:
    - Використання масиву чисел для кнопок навігації: pageNumbers, що генерується через метод Array(...).fill(...).map(...).
    - Кількість сторінок розраховується на основі загальної кількості товарів та заданої кількості товарів на сторінці.
2. **Динамічний вибір кількості товарів на сторінці**:
    - Елемент <select> дозволяє користувачеві змінювати розмір сторінки (кількість товарів на ній).
    - Перерахунок списку виконується в методі changePageSize().
3. **Категорії товарів**:
    - Фільтрація товарів за вибраною категорією через зміну значення selectedCategory.

Реалізація з директивою:

1. **Створення структурної директиви**:
    - **CounterDirective** полегшує роботу з генерацією послідовних чисел.
    - Вона викликає метод createEmbeddedView стільки разів, скільки визначено у вхідному значенні counter.
2. **Використання директиви в шаблоні**:
    - Замість масиву pageNumbers, використовується новостворена директива:

```
<button *counter="let page of pageCount" ...>{{page}}</button>
```

- Спрощення логіки та краща масштабованість завдяки гнучкості директив.

3. **Адаптація компонента**:
   - Додано pageCount, що напряму передає кількість сторінок до директиви.
   - Забрано метод pageNumbers.

Висновки:

- **Переваги підходу з директивою**:
  - Зменшення логіки у компоненті.
  - Використання спеціалізованих рішень робить шаблон чистішим і зрозумілішим.
- **Використання в інших проектах**:
  - Настроювані директиви, як CounterDirective, можна повторно використовувати у схожих завданнях.

Огляд моделі кошика у додатку, його призначення

Модель кошика (Cart) є важливим компонентом інтернет-магазину, що забезпечує управління товарами, які користувач додає для подальшої покупки. Основні цілі моделі кошика:

- Зберігати інформацію про товари, додані до кошика.
- Обчислювати загальну кількість товарів і їхню загальну вартість.
- Забезпечувати механізми для оновлення, додавання, видалення і очищення товарів у кошику.

Призначення класу Cart

Модель Cart відповідає за збереження даних кошика та виконання операцій над ними. Основні методи та їхнє призначення:

1. **addLine(product: Product, quantity: number)**
    - Додає новий товар у кошик або оновлює кількість товару, якщо він уже є у кошику.
    - Використовує пошук по id товару, щоб уникнути дублювання товарів.
2. **updateQuantity(product: Product, quantity: number)**
    - Оновлює кількість конкретного товару в кошику.
    - Використовується для змін обсягу замовлення товару вже після його додавання.
3. **removeLine(id: number)**
    - Видаляє обраний товар із кошика за його id.
4. **clear()**
    - Очищає кошик, видаляючи всі товари.
5. **recalculate()**

- Перераховує загальну кількість товарів (itemCount) і загальну вартість кошика (cartPrice).
- Викликається після кожної операції, що впливає на дані у кошику.

Призначення класу CartLine

Модель CartLine представляє окрему позицію (товар) у кошику:

- **Конструктор**:
  - product — товар, пов'язаний із даною позицією.
  - quantity — кількість цього товару в кошику.
- **Геттер lineTotal**:
  - Обчислює загальну вартість позиції в кошику на основі кількості (quantity) та ціни товару (product.price).

Особливості та переваги реалізації

1. **Гнучкість**:
   - Можливість легко додавати або змінювати товар у кошику.
   - Підрахунок даних у реальному часі через автоматичне перерахування.
2. **Централізоване управління**:
   - Усі операції щодо даних кошика зосереджені в одному місці, що полегшує підтримку та розширення функціоналу.
3. **Чистота коду**:
   - Використання окремого класу CartLine робить модель легкою для розуміння і зручною у використанні.
4. **Просте інтегрування**:

- Модель кошика може бути використана в компонентах для відображення товарів, загальної ціни або контролю товарів у кошику.

Огляд компонентів, що призначені для роботи з кошиком

Реалізація інтерактивних компонентів для роботи з кошиком покликана забезпечити зручний та інтуїтивний інтерфейс користувача. У цьому коді представлені два основні компоненти: **CartDetailComponent** і **CartSummaryComponent**, кожен із яких має конкретне призначення.

Компоненти

1. CartDetailComponent

- **Призначення**: Використовується для повного перегляду товарів у кошику, оновлення їхньої кількості, видалення товарів, а також переходу до оформлення замовлення.
- **Особливості**:
    - **Інтерфейс**: Відображає товари у вигляді таблиці, де вказані:
        - Назва продукту.
        - Кількість.
        - Ціна за одиницю.
        - Загальна вартість товару (subtotal).
    - Забезпечує оновлення кількості товарів через текстові поля введення.
    - Реалізує функцію видалення товарів кнопкою **Remove**.
    - Підраховує та відображає загальну вартість товарів у кошику.
    - Два кнопки дії:
        - **Continue Shopping**: повернення до магазину.
        - **Checkout**: перехід до оформлення замовлення (активна, якщо у кошику є товари).

src/app/shop/cartDetail.component.ts

```
import { Component } from "@angular/core";
import { Cart } from "../model/cart.model";
@Component({
    templateUrl: "./cartDetail.component.html"
})
export class CartDetailComponent {
    constructor(public cart: Cart) {}
}
```

src/app/shop/cartDetail.component.html

```
<div class="container-fluid">
    <div class="row">
        <div class="bg-dark text-white p-2">
            <span class="navbar-brand ml-2">SportShop</span>
        </div>
    </div>
    <div class="row">
        <div class="col mt-2">
            <h2 class="text-center">Your Cart</h2>
            <table class="table table-bordered table-striped p-2">
                <thead>
                    <tr>
                        <th>Quantity</th>
                        <th>Product</th>
                        <th class="text-end">Price</th>
                        <th class="text-end">Subtotal</th>
                    </tr>
                </thead>
                <tbody>
                    <tr *ngIf="cart.lines.length == 0">
                        <td colspan="4" class="text-center">
                        Your cart is empty
                        </td>
                    </tr>
                    <tr *ngFor="let line of cart.lines">
                        <td>
                            <input type="number" class="form-control-sm"
                                style="width:5em" [value]="line.quantity"
                                (change)="cart.updateQuantity(line.product,
                                $any($event).target.value)" />
                        </td>
                        <td>{{line.product.name}}</td>
                            <td class="text-end">
                            {{line.product.price | currency:"USD":"symbol":"2.2-
2"}}
                        </td>
                        <td class="text-end">
```

```
                        {{(line.lineTotal) | currency:"USD":"symbol":"2.2-2" }}
                    </td>
                    <td class="text-center">
                        <button class="btn btn-sm btn-danger"
(click)="cart.removeLine(line.product.id ?? 0)">
                            Remove
                        </button>
                    </td>
                </tr>
            </tbody>
            <tfoot>
            <tr>
            <td colspan="3" class="text-end">Total:</td>
            <td class="text-end">
            {{cart.cartPrice | currency:"USD":"symbol":"2.2-2"}}
            </td>
            </tr>
            </tfoot>
        </table>
    </div>
</div>
<div class="row">
    <div class="col">
        <div class="text-center">
            <button class="btn btn-primary m-1" routerLink="/shop">
                Continue Shopping
            </button>
            <button class="btn btn-secondary m-1" routerLink="/checkout"
[disabled]="cart.lines.length == 0">
                Checkout
            </button>
        </div>
    </div>
</div>
</div>
```

2. CartSummaryComponent

- **Призначення**: Використовується для короткого резюме стану кошика. Зазвичай відображається у верхньому правому куті або в навігаційному меню, щоб нагадувати користувачу про стан кошика.
- **Особливості**:
    - **Інтерфейс**: Відображає:
        - Загальну кількість товарів у кошику (cart.itemCount).
        - Загальну вартість товарів (cart.cartPrice).

- Забезпечує швидкий доступ до деталей кошика кнопкою з іконкою **cart**.

src/app/shop/cartSummary.component.ts

```
import { Component } from "@angular/core";
import { Cart } from "../model/cart.model";
@Component({
    selector: "cart-summary",
    templateUrl: "./cartSummary.component.html"
})
export class CartSummaryComponent {
    constructor(public cart: Cart) { }
}
```

src/app/shop/cartSummary.component.html

```
<div class="float-end">
    <small class="fs-6">
        Your cart:
        <span *ngIf="cart.itemCount > 0">
            {{ cart.itemCount }} item(s)
            {{ cart.cartPrice | currency:"USD":"symbol":"2.2-2" }}
        </span>
        <span *ngIf="cart.itemCount == 0">
            (empty)
        </span>
    </small>
    <button class="btn btn-sm bg-dark text-white" [disabled]="cart.itemCount == 0"
routerLink="/cart">
        <i class="fa fa-shopping-cart"></i>
    </button>
</div>
```

HTML-структура

Деталі реалізації

1. **Основна таблиця у CartDetailComponent**:
    - Створює зручну таблицю для перегляду товарів.
    - Рядки таблиці динамічно генеруються через *ngFor, що обробляє дані з об’єкта cart.lines.
    - Містить:

        - Поле для зміни кількості товару (обробник подій cart.updateQuantity).
        - Кнопку видалення товару (обробник подій cart.removeLine).
    - Загальна сума відображається у нижньому рядку таблиці (tfoot).
2. **Резюме у CartSummaryComponent**:
    - Динамічно відображає:
        - Стан кошика: порожній чи наповнений.
        - Кількість товарів і їх загальну вартість.
    - Кнопка для переходу до деталей кошика (routerLink="/cart").
3. **Логічні дії через Angular Binding**:
    - [value] — Встановлення значення для введення кількості.
    - (change) — Обробка подій зміни значення кількості.
    - [disabled] — Контроль стану кнопок залежно від даних кошика.

Переваги реалізації

1. **Мінімалістичний та зручний UI**:
    - Інтерактивність забезпечена прив’язками Angular.
    - Чітко розділений функціонал між компонентами.
2. **Гнучкість і динамічність**:
    - Актуальний стан кошика динамічно синхронізується з моделлю Cart.
3. **Масштабованість**:
    - Компоненти легко розширюються для нових функцій (наприклад, застосування знижок).

Компоненти **CartDetailComponent** та **CartSummaryComponent** чудово реалізують базові функції для управління кошиком і забезпечують користувачу зручний досвід взаємодії.

Наявна маршрутизація у додатку

Маршрутизація в Angular-додатку визначає навігацію між компонентами, що дозволяє користувачу переміщуватись по сторінках. Наведена конфігурація маршрутизації вказує на структуру додатку, реалізуючи декілька маршрутів для забезпечення доступу до основних розділів.

Головні аспекти маршрутизації

1. **Використання RouterModule**:
    - Маршрутизація налаштована за допомогою RouterModule.forRoot, що реєструє маршрути на кореневому рівні програми.
2. **Основні маршрути**: У додатку налаштовано три основних маршрути:
    - /shop — компонент магазину (ShopComponent).
    - /cart — компонент кошика (CartDetailComponent).
    - /checkout — компонент оформлення замовлення (CheckoutComponent).
3. **Захист маршрутів**:
    - Кожен маршрут захищений за допомогою **Angular Guard** — ShopFirstGuard, що визначає, чи може користувач отримати доступ до певного маршруту.
4. **Маршрут за замовчуванням**:
    - Маршрут із шаблоном ** перенаправляє користувача на /shop, якщо введений URL не відповідає жодному з визначених маршрутів.

Опис ключових елементів

1. Компоненти

- **ShopComponent**:
  - Основний компонент для відображення магазину, каталогу товарів або іншого пов’язаного контенту.
  - Підключений до шляху /shop.
- **CartDetailComponent**:
  - Компонент для відображення деталей кошика.
  - Підключений до шляху /cart.
- **CheckoutComponent**:
  - Компонент для оформлення замовлення.
  - Підключений до шляху /checkout.

2. Встановлення маршруту за замовчуванням

- path: "**":
  - Якщо користувач вводить URL, що не відповідає жодному із заданих маршрутів, виконується перенаправлення на /shop.
  - Забезпечує відсутність помилок 404 у межах додатку.

Призначення кожного маршруту

Шлях	**Компонент**	**Призначення**
/shop	ShopComponent	Відображає головну сторінку магазину.
/cart	CartDetailComponent	Дає змогу переглядати й редагувати вміст кошика.
/checkout	CheckoutComponent	Оформлення замовлення.

Шлях	Компонент	Призначення
**	Редирект на /shop	Перенаправляє на головну сторінку магазину в разі помилок.

Реалізація роботи з замовленнями через джерело даних та репозиторій

У даному випадку представлена реалізація роботи з замовленнями у додатку, використовуючи шаблони проектування **Repository** та **Observable** для роботи з замовленнями через джерело даних.

Ключові компоненти цієї реалізації:

1. **Модель замовлення** (Order):
    - Модель визначає структуру замовлення, яке містить всі необхідні поля: ім'я, адреса, місто, інші контактні дані та статус відправлення.
    - Також міститься посилання на кошик (cart), тому при створенні замовлення користувач пов'язує його з товарами в кошику.
    - Важливе поле - shipped, яке відображає, чи було замовлення відправлене.
2. **Репозиторій замовлень** (OrderRepository):
    - Репозиторій відповідає за отримання та збереження замовлень. Він використовує **StaticDataSource** (джерело даних), яке зазвичай працює з HTTP-запитами або ж зберігає дані в локальній пам'яті/файлах.
    - У методі getOrders() повертається масив збережених замовлень.
    - У методі saveOrder() використовується джерело даних (StaticDataSource) для збереження нових замовлень через **Observable**.
3. **Компонент оформлення замовлення** (CheckoutComponent):
    - Цей компонент містить логіку для оформлення замовлення та відправки даних через форму.

- Використовує Angular форму (NgForm) для валідації даних замовлення.
- При відправці форми (метод submitOrder) перевіряється її валідність. Якщо форма заповнена правильно, відправляється запит через репозиторій на збереження замовлення через метод saveOrder.

Детальний опис кожного з компонентів

1. Модель Order

Модель замовлення Order забезпечує зберігання інформації, необхідної для оформлення замовлення:

```
import { Injectable } from "@angular/core";
import { Cart } from "./cart.model";
@Injectable()
export class Order {
    public id?: number;
    public name?: string;
    public address?: string;
    public city?: string;
    public state?: string;
    public zip?: string;
    public country?: string;
    public shipped: boolean = false;
    constructor(public cart: Cart) {}
    clear() {
        this.id = undefined;
        this.name = this.address = this.city = undefined;
        this.state = this.zip = this.country = undefined;
        this.shipped = false;
        this.cart.clear();
    }
}
```

- При створенні замовлення передається об'єкт кошика (cart), що дозволяє миттєво створити замовлення, яке містить товари.

- Метод clear() скидає дані замовлення, повертаючи об'єкт у початковий стан, й очищує кошик.

2. Репозиторій OrderRepository

Репозиторій здійснює доступ до даних замовлень, взаємодіє з джерелом даних через StaticDataSource:

```
import { Injectable } from "@angular/core";
import { Observable } from "rxjs";
import { Order } from "./order.model";
import { StaticDataSource } from "./static.datasource";
@Injectable()
export class OrderRepository {
    private orders: Order[] = [];
    constructor(private dataSource: StaticDataSource) {}
    getOrders(): Order[] {
        return this.orders;
    }
    saveOrder(order: Order): Observable<Order> {
        return this.dataSource.saveOrder(order);
    }
}
```

- getOrders(): Метод для отримання всіх замовлень, зберігає замовлення в локальному масиві orders.
- saveOrder(order): Метод для збереження нового замовлення, через джерело даних StaticDataSource.

3. Компонент для оформлення замовлення CheckoutComponent

Компонент обробляє всю логіку пов'язану з оформленням замовлення:

```
import { Component } from "@angular/core";
import { NgForm } from "@angular/forms";
import { OrderRepository } from "../model/order.repository";
import { Order } from "../model/order.model";
@Component({
    templateUrl: "./checkout.component.html",
```

```
    styleUrls: ["checkout.component.css"]
})
export class CheckoutComponent {
    orderSent: boolean = false;
    submitted: boolean = false;
    constructor(public repository: OrderRepository,
    public order: Order) {}
    submitOrder(form: NgForm) {
        this.submitted = true;
        if (form.valid) {
            this.repository.saveOrder(this.order).subscribe(order => {
            this.order.clear();
            this.orderSent = true;
            this.submitted = false;
            });
        }
    }
}
```

- **Форма замовлення**: Компонент використовує NgForm, який дозволяє підтвердити, що дані у формі є валідними.
- **Метод submitOrder()**:
    - Спочатку поміщає дані з форми на статус submitted (значення true).
    - Якщо форма валідна, то відправляє запит на збереження даних замовлення за допомогою репозиторію (repository.saveOrder(this.order)).
    - Після збереження замовлення через Observable відновлюється початковий стан (this.order.clear()), і замовлення відмічається як надіслане (orderSent = true).

REST-сумісні веб-служби: призначення, використання

REST (Representational State Transfer) — це архітектурний стиль для створення веб-служб, який використовує стандартні HTTP методи для доступу та маніпулювання ресурсами. REST-служби зазвичай використовують запити HTTP для взаємодії між клієнтським і серверним додатками.

У цьому контексті REST-сумісні веб-служби допомагають додатку взаємодіяти з сервером для отримання продуктів і збереження замовлень через HTTP-запити. Нижче детально розглянемо реалізацію таких служб у коді.

Основні компоненти REST-служби

1. **RestDataSource - джерело даних для роботи з HTTP**

- У класі RestDataSource реалізовано методи для роботи з API за допомогою **HTTP-клієнта** Angular (HttpClient).
- При побудові URL для запитів до серверу використовуються статичні протокол і порт:

```
const PROTOCOL = "http";
const PORT = 3500;
```

URL формується як:

```
this.baseUrl = `${PROTOCOL}://${location.hostname}:${PORT}/`;
```

2. **Методи getProducts і saveOrder**

- **getProducts()**: Цей метод виконує HTTP-запит методом GET для отримання списку продуктів зі сервера.

Запит отримує список продуктів у вигляді масиву об'єктів типу Product.

- **saveOrder(order: Order)**: Цей метод відправляє замовлення на сервер через метод POST.

Відправка здійснюється за допомогою HTTP-методу POST та містить передану модель замовлення (order).

Імітація серверного джерела даних (Mock Server) для тестування

У частині коду є налаштування для імітації роботи серверної частини за допомогою Node.js. Даний код демонструє, як можна імітувати просту серверну логіку для роботи з продуктами та замовленнями:

- **Список продуктів**: Це статична структура даних, що імітує базу даних з товарними позиціями (продуктами).
- **Список замовлень**: Пустий масив orders для зберігання замовлень, які створюватимуться за допомогою клієнтського додатка.

Цей код використовується у середовищі тестування або в розробці з метою імітації роботи сервера, наприклад, за допомогою json-server, який дозволяє швидко створювати фейковий API на основі цього файлу.

Призначення REST-сумісних веб-служб

1. **Інтерфейс взаємодії між клієнтським та серверним додатком**:
   - REST дозволяє клієнту (Angular-додатку) запитувати дані з серверу (у даному випадку — продукти) та надсилати дані (наприклад, замовлення).
2. **Уніфікація та простота у використанні**:

- REST-веб-служби забезпечують простоту використання завдяки використанню стандартних HTTP-методів: GET (для отримання даних), POST (для створення нових даних), PUT (для оновлення даних), DELETE (для видалення даних).

3. **Асинхронна робота**:

- Завдяки використанню **Observable** в Angular, усі запити виконуються асинхронно, що дозволяє користувачеві працювати з додатком без очікування на отримання відповіді від сервера.

4. **Масштабованість**:

- REST-сумісні веб-служби забезпечують легке масштабування завдяки стандартам HTTP, що робить їх ідеальними для використання в великих, дистрибуційних системах.

Використання REST API у даному коді

1. **Отримання продуктів**:

- В RestDataSource за допомогою методу getProducts додаток надсилає запит на сервер і отримує список доступних продуктів.
  - Запит: GET http://localhost:3500/products
  - Сервер повертає масив продуктів, які можуть бути відображені на сторінці.

2. **Збереження замовлення**:

- В RestDataSource за допомогою методу saveOrder додаток відправляє дані замовлення на сервер.
  - Запит: POST http://localhost:3500/orders
  - Система зберігає нове замовлення в списку та, можливо, повертає його згенерований ID або інші дані.

src/app/model/rest.datasource.ts

```
import { Injectable } from "@angular/core";
import { HttpClient } from "@angular/common/http";
import { 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;
    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);
    }
}
```

ВИСНОВКИ

У ході виконання лабораторної роботи було створено проект «SportShop», який включає основні структурні блоки для розробки інтерфейсу онлайн-магазину. У рамках виконання лабораторної роботи було здобуто практичні навички з розробки на Angular, інтеграції REST API, роботи з сервісами, pipes та директивами, а також використання бібліотеки RxJS.

1. Розробка основних структурних блоків та моделей даних

У першу чергу було створено основні компоненти додатку, серед яких: моделі даних (для продуктів та замовлень), репозиторії для отримання й збереження даних, сервіси для роботи з моделями даних і запитами, а також компоненти для взаємодії з користувачем (наприклад, для вибору товарів, кошика та оформлення замовлення).

- **Моделі даних** дозволили чітко визначити структуру продуктів, замовлень та елементів кошика. В результаті чого, з’явилася можливість організувати інформацію, що зберігається на клієнтській стороні, а також у репозиторії.
- **Реалізація фіктивного джерела даних** у вигляді статичного файлу забезпечила швидкий старт роботи, а також дозволила адаптувати код для можливості використання справжнього API на основі REST-принципів, що буде актуально в подальшій частині роботи.

2. Вибір товарів та оформлення замовлень

Розробка основних функцій для вибору товарів, додавання їх до кошика, редагування кількості і оформлення замовлень була логічним продовженням першої частини роботи. Користувач отримав змогу виконати наступні операції:

- **Додавання товарів до кошика**. Для кожного товару було передбачено можливість вибору кількості, що дозволяє користувачу персоналізувати свій вибір.
- **Оформлення замовлення** за допомогою збереження даних на сервері через REST API дозволило симулювати реальний процес замовлення продуктів із магазину, а також їх подальше оброблення.

3. Інтеграція з HTTP REST-сумісною веб-службою

У другій частині лабораторної роботи було замінено фіктивне джерело даних на REST API за допомогою сервісу RestDataSource. Це дозволило інтегрувати веб-службу для отримання продуктів і відправки замовлень на сервер, забезпечуючи асинхронну роботу через HTTP запити. Усі операції взаємодії з сервером виконуються через використання RxJS та HttpClient.

4. Функціональність і зовнішній вигляд

Додаток включає ряд функцій, які забезпечують зручний досвід для кінцевих користувачів:

- **Пагінація** на головній сторінці магазину, що дозволяє зручно переглядати список товарів, розбитих на сторінки.
- **Фільтрація товарів** за категоріями, що дозволяє користувачам швидко знаходити потрібні продукти за інтересами.

У рамках розробки було створено **нестандартну директиву для пагінації**, яка дозволила організувати зручний та елегантний спосіб навігації між товарами.

СПИСОК ДЖЕРЕЛ

1. Документація Angular. URL: https://v17.angular.io/docs
2. Документація Nodejs. URL: https://nodejs.org/docs/latest/api/
3. Документація Firebase. URL: https://firebase.google.com/docs?hl=en