**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.

з дисципліни: «Реактивне програмування»

Студент: Головня Олександр Ростиславович Група: ІП-11 Дата захисту роботи: Викладач: доц. Полупан Юлія Вікторівна Захищено з оцінкою:

Київ, 2024

**Зміст**

[«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО» 1](#_Toc180823024)

[**І) Частина 1: Створення проекту «SportShop»** 5](#_Toc180823025)

[**Підготовка проекту** 5](#_Toc180823026)

[**Створення структури папок** 5](#_Toc180823027)

[**Встановлення додаткових пакетів NPM** 5](#_Toc180823028)

[**Додавання CSS стилів до проекту** 5](#_Toc180823029)

[**Підготовка REST – сумісної веб-служби** 5](#_Toc180823030)

[**Підготовка файлу HTML** 8](#_Toc180823031)

[**Запуск прикладу** 9](#_Toc180823032)

[**Запуск REST-сумісної веб-служби** 9](#_Toc180823033)

[**Підготовка проекту Angular** 10](#_Toc180823034)

[**Оновлення кореневого компонента** 10](#_Toc180823035)

[**Перевірка кореневого модуля** 10](#_Toc180823036)

[**Аналіз файлу початкового завантаження** 11](#_Toc180823037)

[**Початок роботи над моделлю даних** 12](#_Toc180823038)

[**Створення класів моделі** 12](#_Toc180823039)

[**Створення фіктивного джерела даних** 13](#_Toc180823040)

[**Створення репозиторію моделі** 14](#_Toc180823041)

[**Створення функціонального модуля** 15](#_Toc180823042)

[**Створення сховища** 16](#_Toc180823043)

[**Створення компоненту магазину та шаблону** 16](#_Toc180823044)

[**Створення функціонального модуля сховища** 18](#_Toc180823045)

[**Оновлення кореневого компонента та кореневого модуля** 19](#_Toc180823046)

[**Додавання функціональності: докладна інформація про товари. Виведення докладної інформації про товари** 20](#_Toc180823047)

[**Додавання вибору категорій** 22](#_Toc180823048)

[**Посторінне виведення списку товарів** 25](#_Toc180823049)

[**Створення нестандартної директиви** 28](#_Toc180823050)

[**ІІ) Частина 2: Вибір товарів та оформлення замовлень. Підготовка програми** 31](#_Toc180823051)

[**Створення кошика** 31](#_Toc180823052)

[**Створення моделі кошика** 31](#_Toc180823053)

[**Створення компонентів для зведеної інформації кошика** 33](#_Toc180823054)

[**Інтеграція кошика у додаток** 35](#_Toc180823055)

[**Маршрутизація URL** 38](#_Toc180823056)

[**Створення та застосування конфігурації маршрутизації** 40](#_Toc180823057)

[**Навігація у додатку** 41](#_Toc180823058)

[**Захисники маршрутів** 43](#_Toc180823059)

[**Завершення виведення вмісту кошика** 46](#_Toc180823060)

[**Обробка замовлень** 48](#_Toc180823061)

[**Розширення моделі** 49](#_Toc180823062)

[**Оновлення репозиторію та джерела даних** 50](#_Toc180823063)

[**Оновлення функціонального модуля** 51](#_Toc180823064)

[**Отримання інформації про замовлення** 51](#_Toc180823065)

[**Використання REST-сумісної веб-служби** 55](#_Toc180823066)

[**Застосування джерела даних** 56](#_Toc180823067)

[Посилання на додатки: 60](#_Toc180823068)

[**Висновок:** 60](#_Toc180823069)

[**Список використаних джерел:** 61](#_Toc180823070)

**Мета: Мета: Навчитися створювати Angular-додатки, які містять сервіси, pipes, директиви та використовувати бібліотеку RxJS.**

**Завдання:**

**Створити два Angular-додатки під назвою Service1 та Service2.**

**Частина 1:**

Створити проект «SportShop». Встановити та налаштувати засоби розробника, створити кореневі структурні блоки для проекту (модель даних, фіктивне джерело даних, репозиторій моделі, сховище, компоненти магазина та шаблони).

Вивести дані фіктивної моделі даних на головну сторінку магазину, реалізувати розбивку на сторінки та фільтрацію товарів за категоріями. Створити нестандартну директиву для пагінації. Додаток, отриманий в результаті виконання

**Частина 2:**

Розробити додаткову логіку в додатку «SportShop» для вибору

товарів та оформлення замовленнь. Реалізувати підтримку кошика для вибору товарів користувачем та процесу оформлення замовлення. Фіктивне джерело даних у проекті замінити джерелом, що надсилає запити до HTTP REST-сумісної веб-служби. Роботу додатку «SportShop», отриманого в результаті виконання Частини 1 та Частини 2, продемонструвати в режимі відеоконференції.

**Зробити звіт по роботі. Звіт повинен включати: титульний лист, зміст,**

**основну частину, список використаних джерел.**

**Angular-додаток SportShop, отриманий в результаті виконання Частини 1**

**завдання, розгорнути на платформі Firebase у проекті з ім’ям «ПрізвищеГрупаLaba5-1», наприклад «KovalenkoIP01Laba5-1».**

**І) Частина 1: Створення проекту «SportShop»**

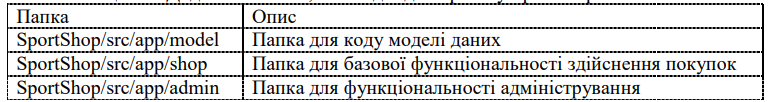
**Підготовка проекту**

Спочатку я підготовив новий проєкт командою:

ng new SportShop --routing false --style css --skip-git --skip-tests

**Створення структури папок**

Структура папок:



**Встановлення додаткових пакетів NPM**

Додаткові пакети потрібні для проекту SportShop на додаток до основних пакетів Angular. Виконав наведені нижче команди, щоб додайти необхідні пакети:   
npm install bootstrap@5.1.3

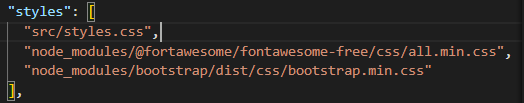
npm install @fortawesome/fontawesome-free@6.0.0

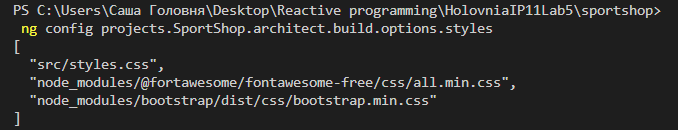
npm install --save-dev [json-server@0.17.0](mailto:json-server@0.17.0)

npm install --save-dev [jsonwebtoken@8.5.1](mailto:jsonwebtoken@8.5.1)

**Додавання CSS стилів до проекту**

Додав стилі до проєкту:

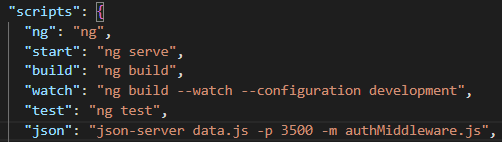




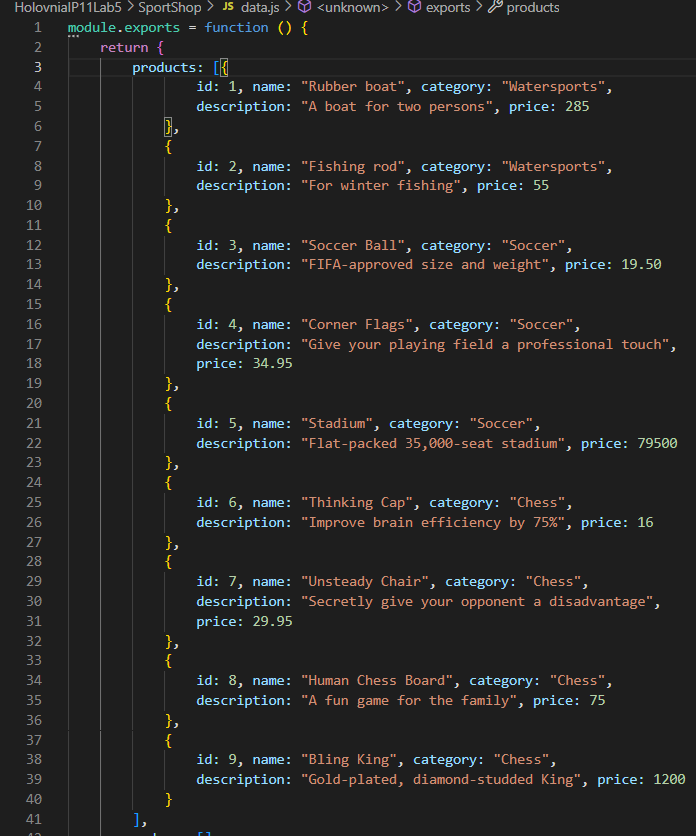
**Підготовка REST – сумісної веб-служби**

Програма SportShop використовуватиме асинхронні HTTP-запити, щоб отримати дані моделі, надані REST-сумісною веб-службою. До проекту було додано пакет jsonserver. Це чудовий пакет для створення веб-сервісів із даних JSON або коду JavaScript.

Додав оператор, показаний на рисунку, до розділу сценаріїв у файлі package.json, щоб пакет json-server можна було запустити з командного рядка.



Щоб забезпечити пакет json-server даними для роботи, додав файл під назвою data.js у папку SportShop і додав код, показаний нижче, який забезпечить доступність тих самих даних кожного разу, коли запускається пакет json-server.



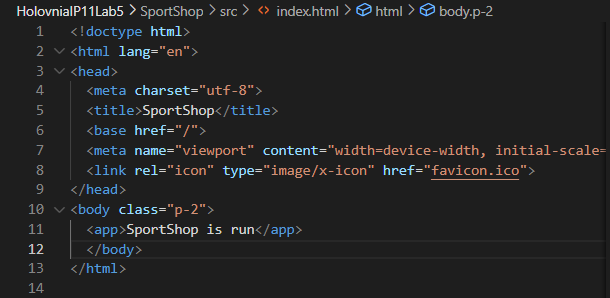
Дані, що зберігаються REST-сумісною веб-службою, необхідно захистити, щоб звичайні користувачі не могли змінювати опис товарів або стан своїх замовлень. Пакет json-server не містить вбудованих засобів аутентифікації, тому було створено файл з ім'ям authMiddleware.js в папці SportShop з наступним кодом:



Цей код перевіряє запити HTTP, надіслані REST-сумісній веб-службі, та реалізує найпростіші засоби безпеки. Це серверний код, що не пов'язаний безпосередньо з розробкою додатків Angular.

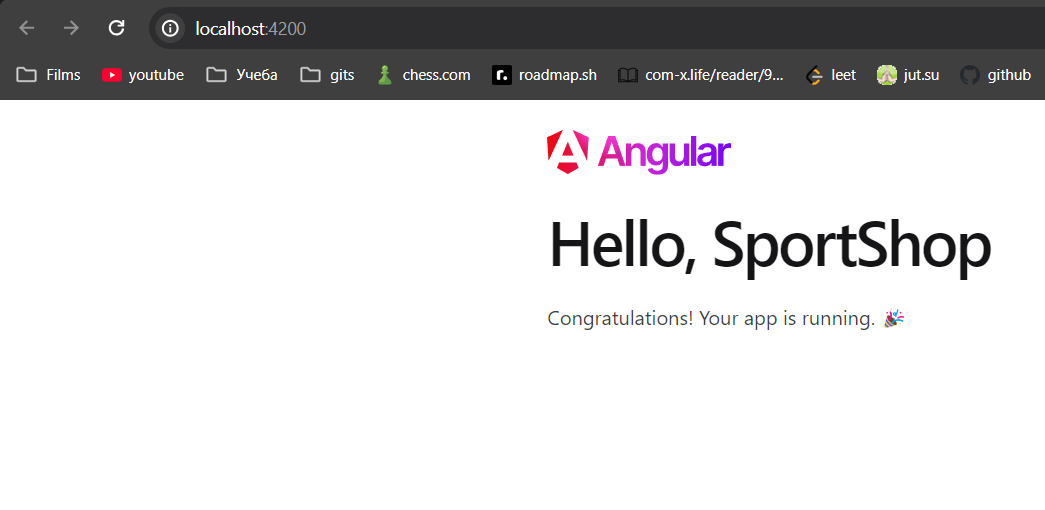
**Підготовка файлу HTML**

Кожна веб-програма Angular має файл HTML, який завантажується браузером і виконує завантаження і запуск програми. Відредагував файл index.html в папці SportShop/src та додав до нього елементи нижче.



Документ HTML містить елемент link для завантаження таблиці стилів Bootstrap та елемент app що резервує місце для функціональності SportShop.

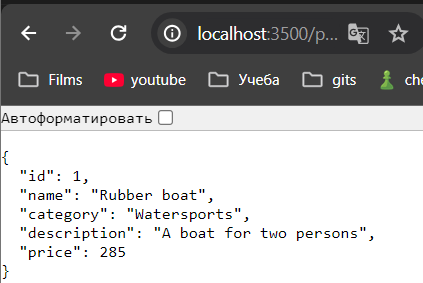
**Запуск прикладу**



**Запуск REST-сумісної веб-служби**

Команда: npm run json

Бачимо результат:

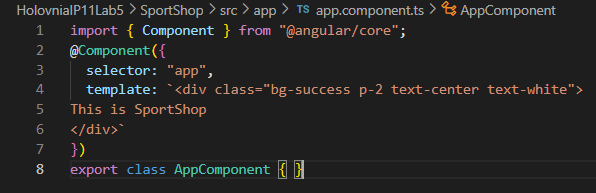


**Підготовка проекту Angular**

У наступних розділах ми закладемо основу, на якій будуватиметься програма SportShop.

**Оновлення кореневого компонента**

Почнемо з кореневого компонента – структурного блоку Angular, який керуватиме елементом app у документі HTML. Додаток може містити кілька компонентів, але серед них завжди є кореневий компонент, що відповідає за відображення контенту верхнього рівня. Відредагував файл app.component.ts в папці SportShop/src/app та включив в нього код нижче:

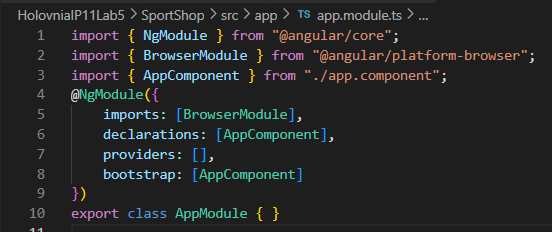


**Перевірка кореневого модуля**

Модулі Angular поділяються на дві категорії: функціональні модулі та кореневий модуль.

Функціональні модулі використовуються для угруповання взаємопов'язаної функціональності програми, щоб спростити керування програмою. Ми створимо функціональні модулі для всіх основних функціональних областей програми, включаючи модель даних, інтерфейс магазину користувача і інтерфейс адміністрування. Кореневий модуль передає опис програми для Angular. В описі зазначено, які функціональні модулі необхідні для запуску додатку, які нестандартні можливості слід завантажити та як називається кореневий компонент.

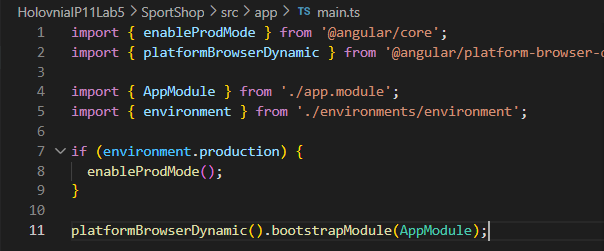
Традиційно файлу кореневого компонента надається ім'я app.module.ts. Перевірив файл із таким ім'ям у папці SportShop/src/app. Цей файл повинен включати код нижче



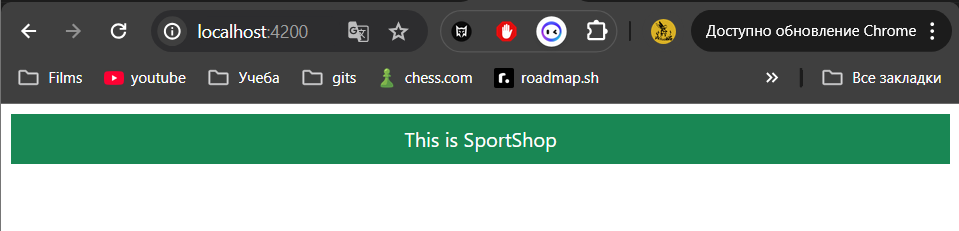
За аналогією з кореневим компонентом клас кореневого модуля не містить код. Річ у тім, що кореневий модуль існує лише для передачі інформації через декоратор @NgModule. Властивість imports наказує Angular завантажити функціональний модуль BrowserModule з усією основною функціональністю Angular, необхідною для вебпрограми. Властивість declarations наказує Angular завантажити кореневий компонент, а властивість bootstrap повідомляє, що кореневим компонентом є клас AppModule. Надалі буде додана інформація до властивостей цього декоратора.

**Аналіз файлу початкового завантаження**

Наступний блок службового коду – файл початкового завантаження, який запускає програму. Файл початкового завантаження використовує браузерну платформу Angular для завантаження кореневого модуля та запуску програми. Створив файл з ім'ям main.ts в папці SportShop/src/app і додав код нижче:



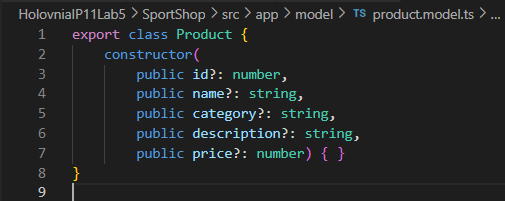
Результат:



**Початок роботи над моделлю даних**

**Створення класів моделі**

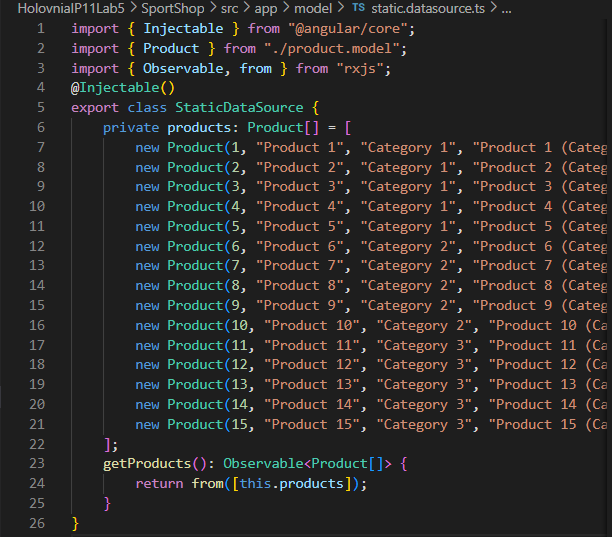
Кожній моделі даних необхідні класи для опису типів даних, що входять до моделі даних. У додатку SportShop це класи з описом товарів, що продаються в інтернетмагазині, та замовлення, отримані від користувачів. Для початку роботи програми SportShop достатньо можливості опису товарів; інші класи моделей будуть створюватися для підтримки розширеної функціональності в міру їхньої реалізації. Створив файл з ім'ям product.model.ts в папці SportShop/src/app/model та включив код нижче:



Клас Product визначає конструктор, який отримує властивості id, name, category, description і price. Ці властивості відповідають структурі даних, які використовуються для заповнення REST-сумісної веб-служби у лістингу 8.5. Знаки запитання (?) за іменами параметрів вказують, що це необов'язкові параметри, які можуть бути опущені під час створення нових об'єктів з використанням классу Product; це може бути зручно при розробці програм.

**Створення фіктивного джерела даних**

Щоб підготувати перехід від фіктивних даних до реальних, ми будемо передавати дані з джерела даних. Решта коду програми не знає, звідки надійшли дані, і перехід на отримання даних із запитів HTTP пройде прозоро. Створив файл static.datasource.ts в папці SportShop/src/app/model та включив визначення класу нижче:



Клас StaticDataSource визначає метод з ім'ям getProducts, що повертає фіктивні дані. Виклик методу getProducts повертає результат Observable - реалізацію Observable для отримання масивів об'єктів Product. Клас Observable надається пакетом Reactive Extensions, який використовується Angular для обробки змін стану проекту.

Об'єкт Observable схожий на об'єкт JavaScriptPromise: він представляє асинхронне завдання, яке в майбутньому має повернути результат. Angular розкриває використання об'єктів Observable для деяких своїх функцій, включаючи роботу із запитами HTTP; саме тому метод getProducts повертає Observable замість повернення даних - простого синхронного або з використанням Promise. Декоратор @Injectable застосовується до классу StaticDataSource. Цей декоратор повідомляє Angular, що цей клас буде використовуватися як служба, що дозволяє іншим класам звертатися до його функціональності через механізм впровадження залежностей.

**Створення репозиторію моделі**

Джерело даних має надати додатку запитувані дані, але звернення до даних зазвичай відбувається через посередника (репозиторій), що відповідає за передачу цих даних окремим структурним блокам програми, щоб подробиці отримання даних залишалися прихованими.

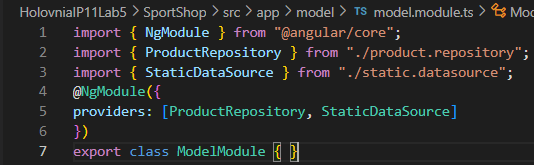
Створив файл product.repository.ts в папці SportShop/src/app/model та визначив в ньому клас нижче:



Коли Angular буде потрібно створити новий екземпляр репозиторію, Angular аналізує клас і бачить, що для виклику конструктора ProductRepository та створення нового об'єкта йому потрібен об'єкт StaticDataSource. Конструктор репозиторію викликає метод getProducts джерела даних, після чого використовує метод subscribe об'єкта Observable, що повертається для отримання цих товарів.

**Створення функціонального модуля**

Зараз визначимо функціональну модель Angular для моделі даних, яка дозволить легко використовувати функціональність моделі даних у будь-якій точці програми. Створив файл з ім'ям model.module.ts в папці SportShop/app/model та визначив клас, наведений нижче:



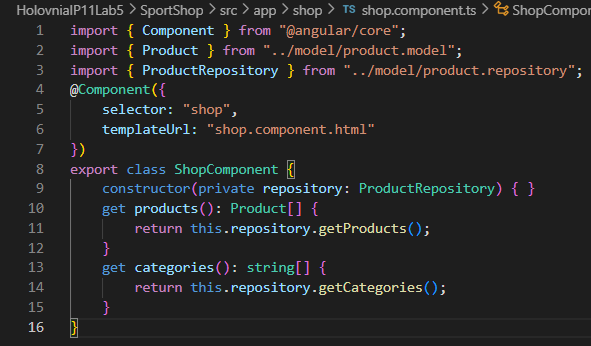
Декоратор @NgModule використовується для створення функціональних модулів, а його властивості повідомляють Angular про те, як повинен використовуватися модуль. В даному випадку модуль містить лише одну властивість providers, яка повідомляє, які класи повинні використовуватися як служби для механізму впровадження залежностей.

**Створення сховища**

Модель даних готова, і ми можемо переходити до побудови функціональності магазину: перегляд списку товарів та оформлення замовлень. У магазині використовуватиметься двостовпцевий макет, список товарів фільтруватиметься за допомогою кнопок категорій, а самі товари виводитимуться в таблиці.

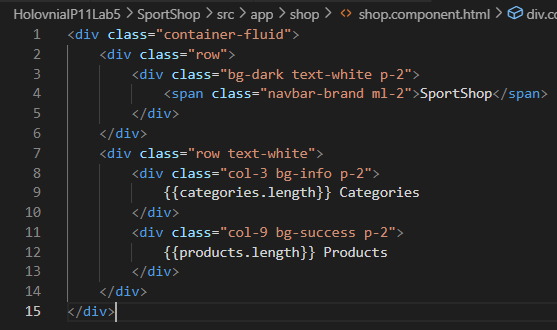
**Створення компоненту магазину та шаблону**

Відправною точкою для функціональності магазину стане новий компонент - клас, що надає дані та логіку для шаблону HTML, що містить прив'язки даних динамічного генерування контенту. Створив файл з ім'ям shop.component.ts в папці SportShop/src/app/shop та додав визначення класу нижче:



До класу ShopComponent застосовується декоратор @Component, який повідомляє Angular, що клас є компонентом. Властивості декоратора вказують Angular, як застосовувати компонент до HTML контенту (з використанням елемента з ім'ям shop) і де знаходиться шаблон компонента (у файлі з ім'ям shop.component.html). Клас ShopComponent надає логіку, яка забезпечує отримання контенту шаблоном. Конструктор класу отримує об'єкт ProductRepository в аргументі, що передається через механізм застосування залежностей. Компонент визначає властивості products і categories, які будуть використовуватися для генерування контенту HTML у шаблоні на підставі даних, одержаних з репозиторію.

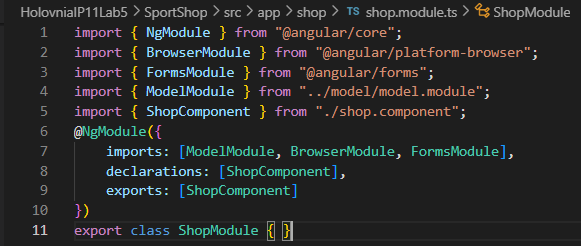
Щоб реалізувати шаблон компонента, створив файл shop.component.html в папці SportShop/src/app/shop та додав контент HTML нижче:



Початкова версія шаблону проста. Більшість елементів надає структуру для макета магазину та застосування деяких CSS-класів Bootstrap. На даний момент використовуються лише дві прив'язки даних Angular, позначені символами{{і}}. Це прив'язки до рядкової інтерполяції; вони наказують Angular обчислити вираз прив'язки і вставити результат в елемент. Вирази у цих прив'язках виводять кількість продуктів та категорій, що надаються компонентом сховища.

**Створення функціонального модуля сховища**

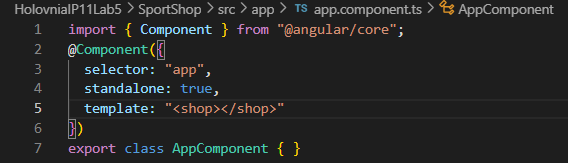
Щоб створити функціональний модуль Angular для функціональності магазину, створив файл з ім'ям shop.module.ts в папці SportShop/src/app/shop та додав код нижче:



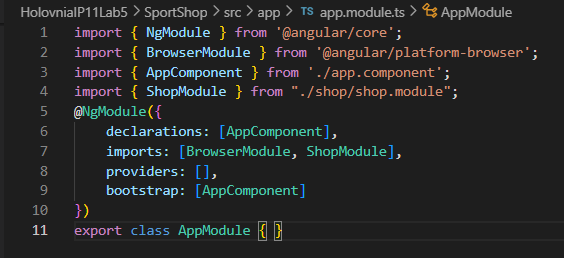
Декоратор @NgModule налаштовує модуль; при цьому властивість imports використовується для передачі Angular інформації про те, що модуль магазину залежить від модуля моделі, а також модулів Browser Module і FormsModule, що містять стандартні функції Angular для веб-застосунків і роботи з елементами форм HTML. Декоратор використовує властивість declarations для передачі Angular інформації про клас ShopComponent, який (як повідомляє властивість exports) може використовуватися в інших частинах програми, це важливо, тому що він буде використовуватися кореневим модулем.

**Оновлення кореневого компонента та кореневого модуля**

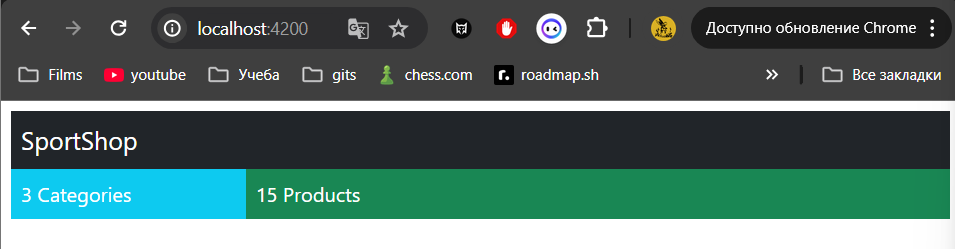
Застосування базової функціональності магазину та моделі вимагає оновлення кореневого модуля програми: він повинен імпортувати два функціональні модулі, а також оновити шаблон кореневого модуля для додавання елемента HTML, до якого буде застосовуватися компонент модуля магазину. Нижче представлені зміни шаблону кореневого компонента.



Елемент shop замінює попередній контент у шаблоні кореневого компонента та відповідає значенню властивості selector декоратора @Component. Нижче показані зміни, які необхідно внести до кореневого модулю, щоб середовище Angular завантажувало функціональний модуль з функціональністю магазину.



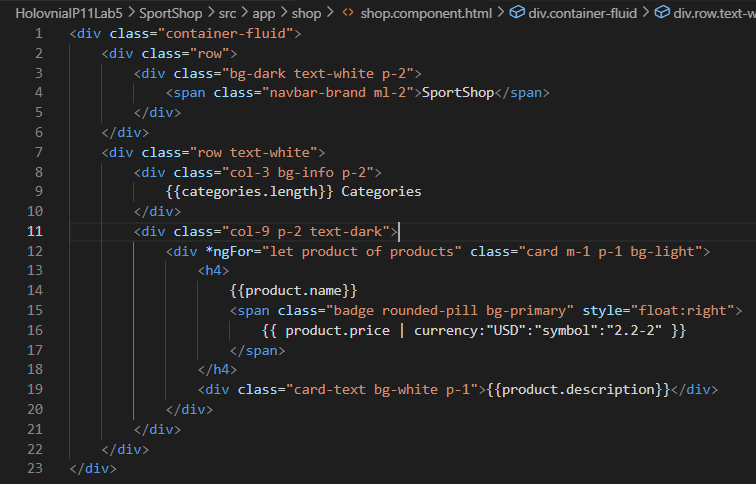
Angular матиме всю інформацію, необхідну для завантаження програми та відображення контенту з модуля магазину.



Всі створені структурні блоки спільно працюють для відображення контенту, який показує, скільки в магазині товарів і на скільки категорій вони діляться.

**Додавання функціональності: докладна інформація про товари. Виведення докладної інформації про товари**

Очевидна відправна точка для роботи над магазином — виведення докладної інформації про товари, щоб користувач бачив, що йому пропонують. Нижче у шаблон компонента магазину додаються елементи HTML з прив'язками даних, що генерують контент для кожного товару, що надається компонентом.



Більшість елементів керує макетом та зовнішнім виглядом контенту. Найважливіша зміна – додавання виразу прив'язки даних Angular.



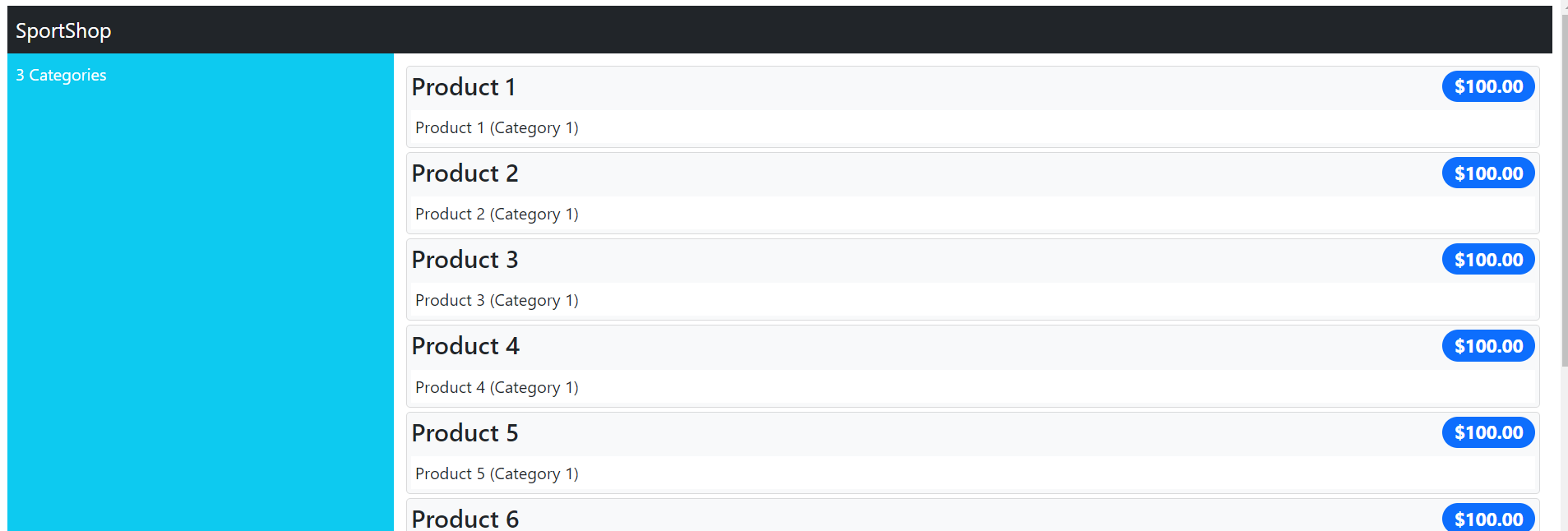
Перед вами приклад директиви, що трансформує елемент HTML, до якого вона застосовується. Ця конкретна директива ngFor перетворює елемент div дублюючи його для кожного об'єкта, що повертається властивістю products компонента. Angular включає низку вбудованих директив для вирішення більшості типових завдань. При дублюванні елемента div поточний об'єкт надається змінною з ім'ям product, що дозволяє легко посилатися на нього з інших прив'язок даних - як у наступному прикладі, де значення властивості description поточного товару вставляється як контент елемента div:



Не всі дані моделі даних проекту можуть виводитися безпосередньо для користувача. Angular включає механізм каналів (pipes), що використовуються класами для перетворення або підготовки значень для прив'язування даних. Angular містить кілька вбудованих каналів, включаючи канал currency, що форматує числові значення у грошовому форматі:

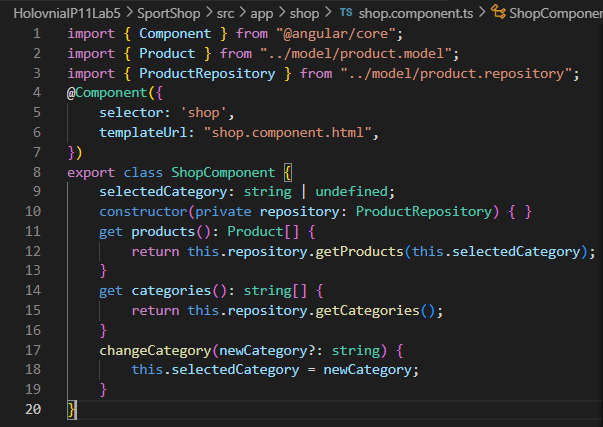


Вираз у цій прив'язці наказує Angular відформатувати властивість price поточного продукту з використанням каналу currency за правилами форматування грошових величин, прийнятими в США. Збережіть зміни у шаблоні. Список товарів з моделі даних виводиться у вигляді довгого списку.

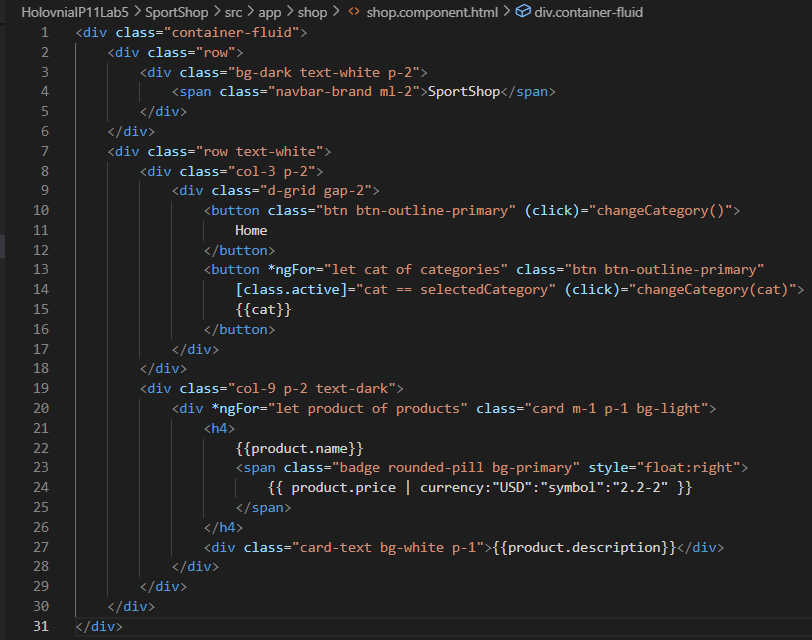


**Додавання вибору категорій**

Щоб додати підтримку фільтрації списку товарів за категоріями, необхідно підготувати компонент магазину. Він повинен стежити за тим, яка категорія була обрана користувачем, та змінювати механізм вибірки даних для використання обраної категорії

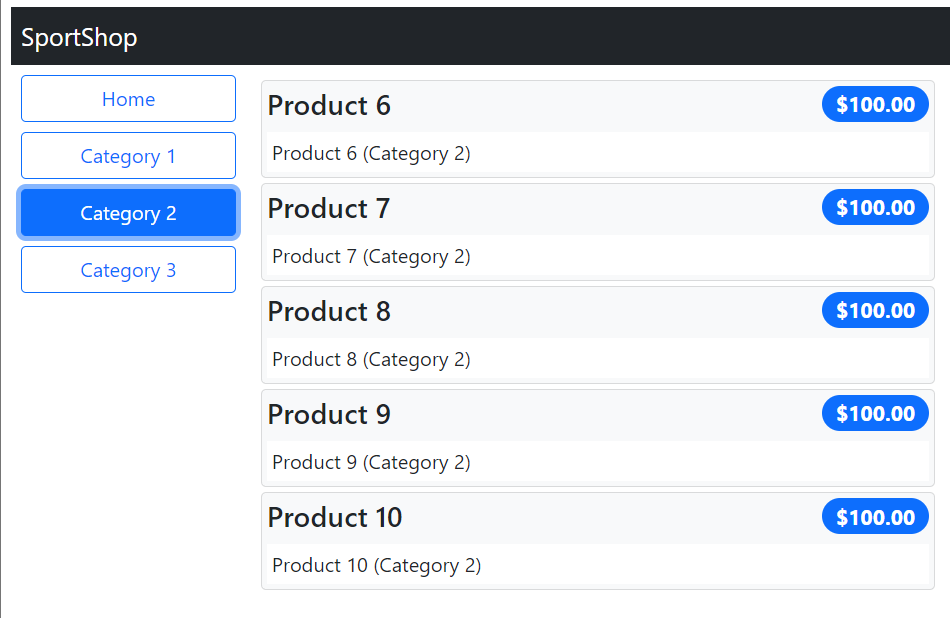


Властивості selectedCategory надається обрана користувачем категорія (null- Усі категорії); ця властивість використовується в методі updateData як аргумент методу getProducts так що фільтрація делегується джерелу даних. Метод change Category об'єднує ці значення у методі, який може викликатись при виборі категорії користувачем. Нижчн представлені відповідні зміни шаблону компонента. Шаблон повинен відображати набір кнопок для зміни вибраної категорії та показувати, яка категорія обрана зараз.



У шаблоні з'явилися два нових елементи button. Перша – кнопка Home - має прив'язку події, яка викликає метод changeCategory компонента при натисканні на кнопці. Метод не отримує аргументу, що рівнозначне призначенню категорії null та вибору всіх товарів. Прив'язка ngFor застосовується до іншого елементу button з виразом, який повторює елемент для кожного значення в масиві, що повертається властивістю categories компонента. Кнопці також призначено прив'язка події click, вираз якої викликає метод changeCategory для вибору поточної категорії; це призведе до фільтрації списку товарів, що виводяться для користувача. Також є прив'язка class яка додає елемент button до активного класу, коли категорія, пов'язана з кнопкою, збігається з обраною категорією.

Таким чином, забезпечується візуальний зворотний зв'язок для користувача при фільтрації за категоріями



**Посторінне виведення списку товарів**

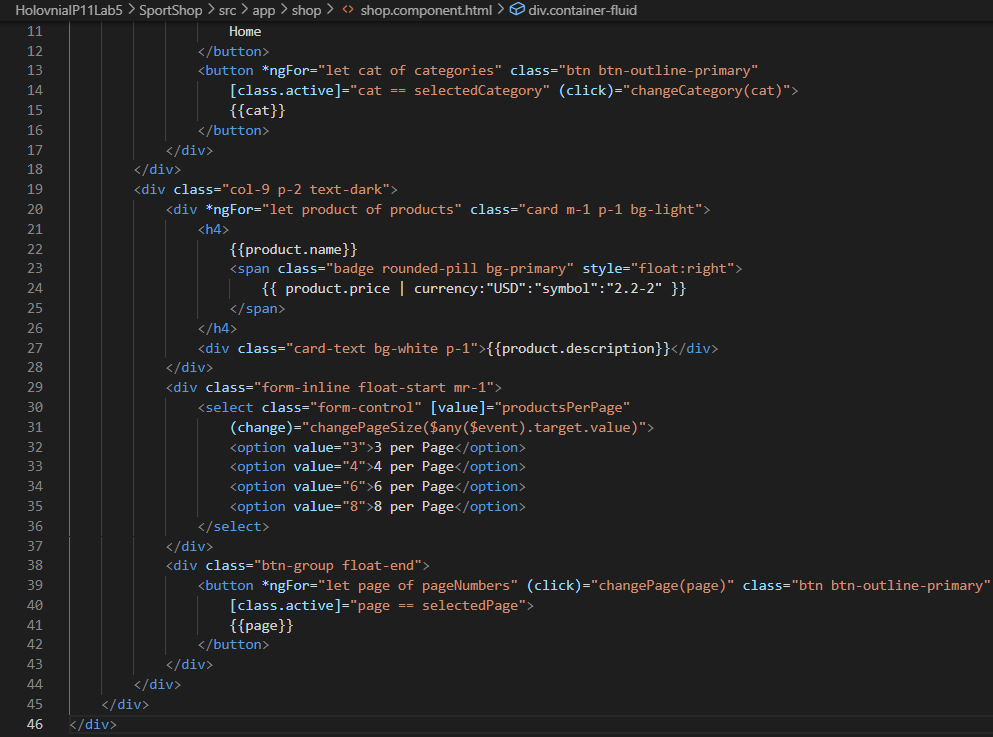
Фільтрування продуктів за категоріями спрощує роботу зі списком товарів, але в більш типовому рішенні список розбивається на менші фрагменти, і кожен фрагмент виводиться на окремій сторінці з навігаційними кнопками для переміщення між сторінками. Нижче у компонент магазину вносяться зміни, щоб у ньому зберігалася поточна сторінка і кількість елементів на сторінці.



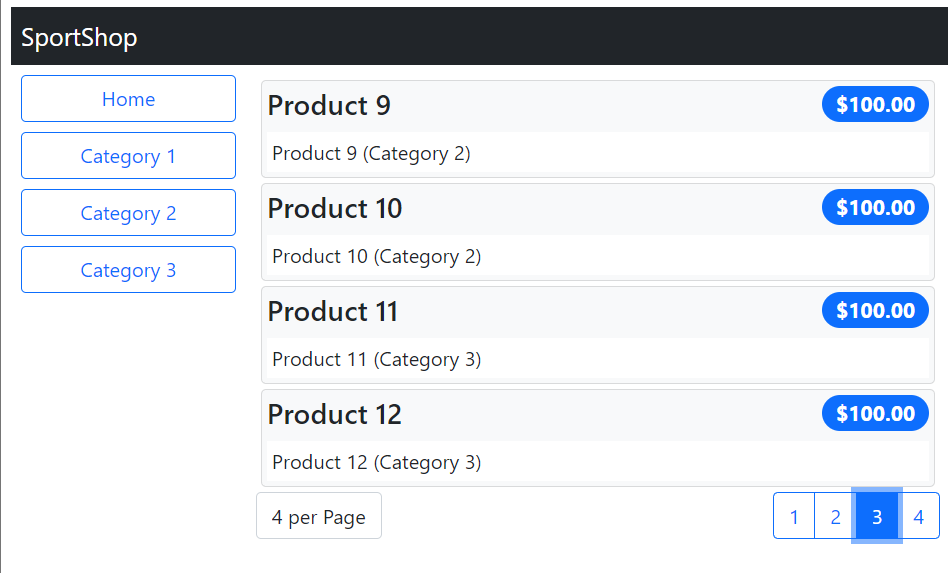
У цьому лістингу реалізовано дві нові можливості: отримання сторінки з інформацією про товари та зміну розміру сторінок (зі зміною кількості товарів, що відображаються на кожній сторінці). Тут є одна дивина, на яку компоненту доводиться використовувати обхідне рішення. Вбудована директива ngFor, що надається Angular, дозволяє генерувати контент тільки для об'єктів із масиву або колекції (без використання лічильника). Так як нам потрібно згенерувати пронумеровані кнопки навігації між сторінками, доводиться створювати масив із потрібними числами:



Ця команда створює новий масив, заповнює його значенням 0, а потім за допомогою методу map генерує новий масив із числовою послідовністю. Таке рішення досить добре працює у реалізації сторінкового виводу, але виглядає досить незграбно; у наступному розділі буде продемонстровано більш вдале рішення. Нижче наведено зміни у шаблоні компонента магазину, необхідні для реалізації сторінкового виводу.

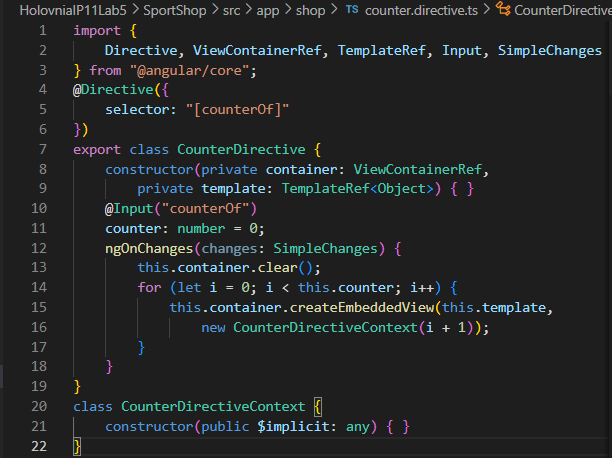


До розмітки додається елемент select, що дозволяє змінювати розмір сторінки, та набір кнопок для переходу між сторінками товарів. Нові елементи містять прив'язки даних, що пов'язують їх із властивостями та методами, що надаються компонентом. В результаті ми отримуємо список товарів, з яким зручніше працювати

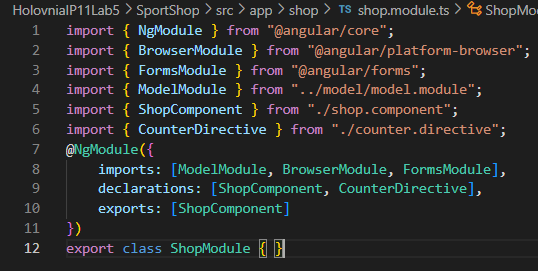


**Створення нестандартної директиви**

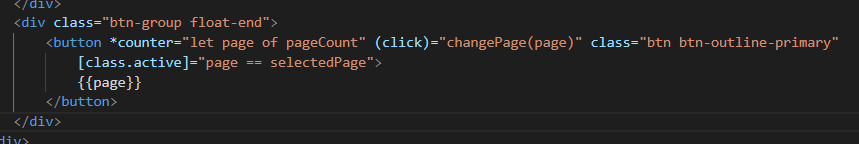
У цьому розділі ми створимо нестандартну директиву, щоб нам не доводилося генерувати масив заповнений числами для створення кнопок навігації. Angular надає хороший набір вбудованих директив, але розробник може відносно просто створювати власні директиви для вирішення завдань, притаманних його додатку, або для підтримки можливостей, які відсутні у вбудованих директивах. Створіть файл counter.directive.ts в папці SportShop/app/shop та використайте його для визначення класу



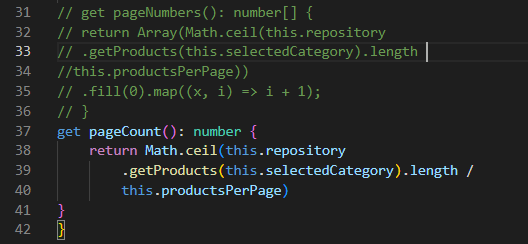
Це приклад структурної директиви. Такі директиви застосовуються до елементів через властивість counter та використовують спеціальні засоби Angular для багаторазового створення контенту (за аналогією із вбудованою директивою ngFor). У цьому випадку, замість того, щоб повертати кожен об'єкт у колекції, нестандартна директива повертає серію чисел, які можуть використовуватися для створення навігаційних кнопок між сторінками. Щоб використати директиву, її необхідно додати до властивості declarations функціонального модуля, як показано нижче:



Після того, як директива була зареєстрована, вона може використовуватися в шаблоні компонента магазину для заміни директиви ngFor, як показано нижче:



Нова прив'язка даних залежить від настроювання нестандартної директиви з використанням властивості pageCount. Нижче масив чисел замінюється простим значенням number, що представляє результат виразу.



В додатку SportShop зовні нічого не змінилося, але продемонстрував використання нестандартної директиви для демонстрації того, як вбудовані функції Angular можуть розширюватись спеціалізованим кодом, адаптованим для потреб конкретного проекту. В цій лабораторній роботі ми розпочали роботу над проектом SportShop.

Ми «заклали фундамент» для проекту: установили та налаштували засоби розробника, створили кореневі структурні блоки для застосування та початку роботи над функціональними модулями. Вивели дані фіктивної моделі даних, реалізували розбивку на сторінки та фільтрацію товарів за категоріями. Також створили нестандартну директиву для демонстрації того, як вбудовані функції Angular можуть розширюватись спеціалізованим кодом.

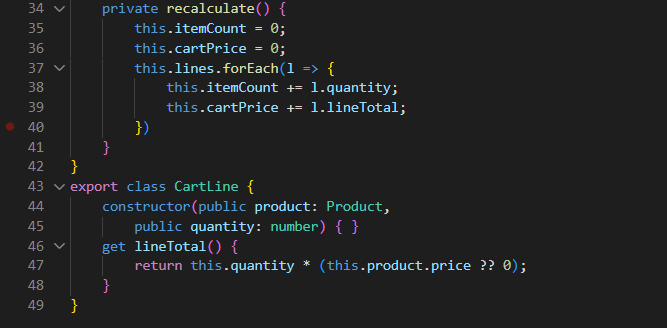
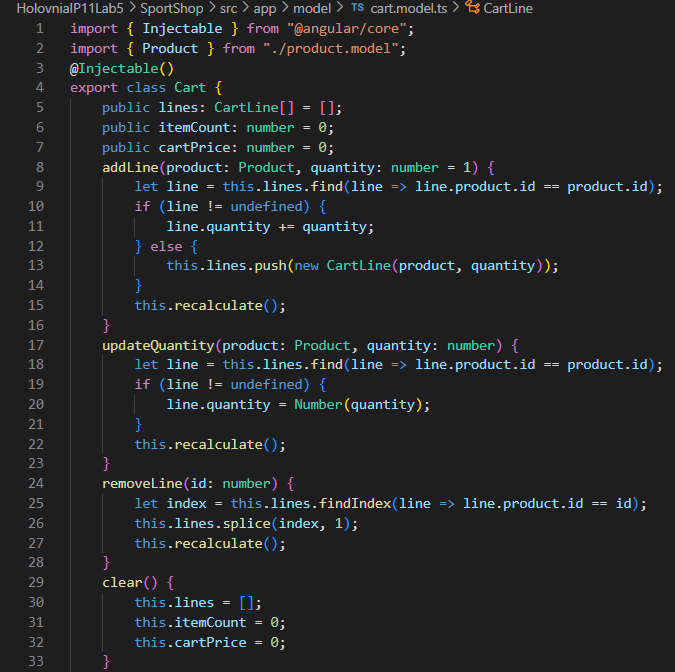
**ІІ) Частина 2: Вибір товарів та оформлення замовлень. Підготовка програми**

**Створення кошика**

Користувачеві знадобиться кошик, в якому розміщуються продукти для подальшого оформлення замовлення. У цій роботі додамо функціональність кошика в додаток і інтегруємо його в магазин, щоб користувач міг вибирати товари, що його цікавлять.

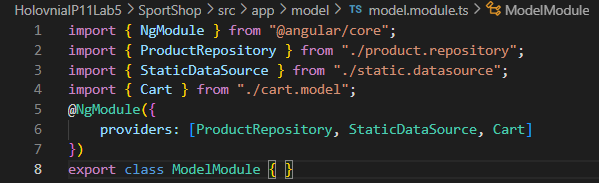
**Створення моделі кошика**

Вихідною точкою для функціональності кошика стане новий клас моделі, який використовуватиметься для збирання товарів, обраних користувачем. Створимо файл з ім'ям cart.model.ts в папці SportShop/src/app/model та включимо у нього визначення класу нижче:



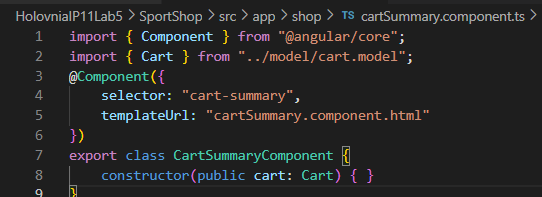
Вибрані продукти є масивом об'єктів CartLine, кожен з яких містить об'єкт Product та кількість одиниць товару. В класі Cart зберігається загальна кількість обраних товарів та їх загальна вартість, яка відображатиметься у процесі покупки. У всьому додатку повинен використовуватися лише один об'єкт Cart, який гарантує, що будь-яка частина програми зможе отримати інформацію про товари, обрані користувачем. Для цього ми оформимо Cart у вигляді глобальної служби; це означає, що Angular буде відповідати за створення екземпляра класу Cart і використовувати його, коли потрібно створити компонент з аргументом конструктора Cart.

Це ще один приклад використання механізму впровадження залежностей Angular, який може використовуватися для спільного доступу до об'єктів у додатку. Декоратор @Injectable, який застосовується до класу Cart у лістингу означає, що клас буде використовуватися як служба. (Строго кажучи, декоратор @Injectable обов'язковий лише за наявності у класі власних аргументів конструктора; проте його краще застосовувати завжди, тому що він сигналізує, що клас призначений для використання як служба.) реєструє клас Cart як службу у властивості providers функціонального модуля моделі.



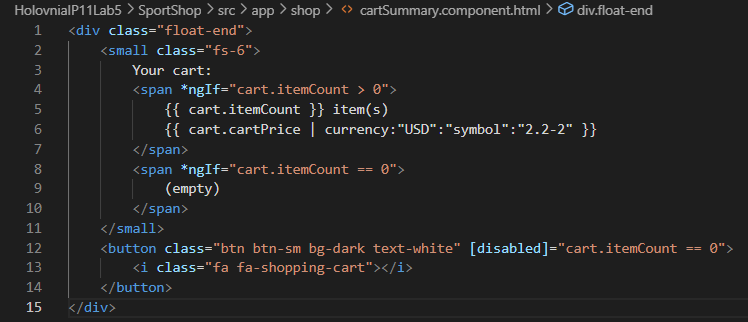
**Створення компонентів для зведеної інформації кошика**

Компоненти є основними структурними блоками у додатках Angular, тому що вони дозволяють легко створювати ізольовані блоки коду та контенту. Програма SportShop виводить зведену інформацію про вибрані товари в заголовку сторінки; для реалізації цієї функціональності треба створити компонент. Створив файл з ім'ям cartSummary.component.ts в папці SportShop/src/app/shop та визначив в ньому компонент

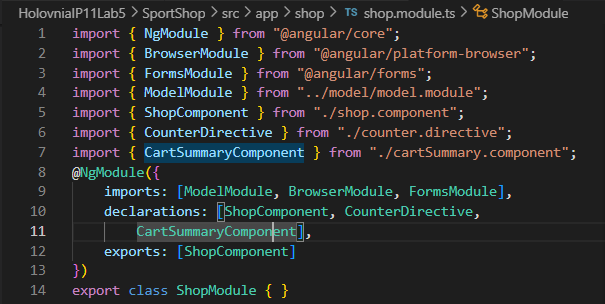


Коли потрібно створити екземпляр цього компонента, середовище Angular має надати об'єкт Cart як аргумент конструктора з використанням служби, налаштованої раніше (додаванням класу Cart у властивість providers функціонального модуля). У варіанті поведінки за замовчуванням один об'єкт Cart буде створено та використано у додатку, хоча доступні різні варіанти поведінки служб.

Щоб надати компоненту шаблон, створив файл HTML з ім'ям cartSummary.component.html в одній папці з файлом класу компонента та додав розмітку



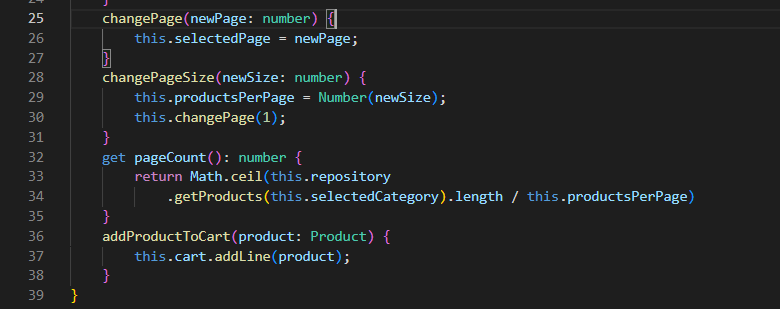
Шаблон використовує об'єкт Cart, наданий компонентом, для виведення кількості товарів у кошику та їх загальної вартості. Також передбачена кнопка для запуску процесу оформлення замовлення, який буде додано до програми пізніше. Код нижче реєструє новий компонент у функціональному модулі магазину, щоб підготуватися до його використання пізніше.



**Інтеграція кошика у додаток**

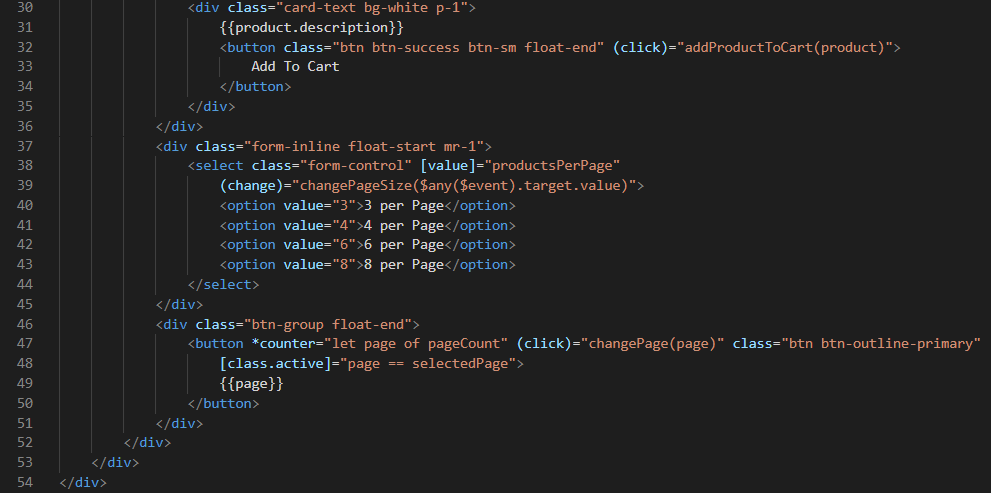
Shop компонент відіграє ключову роль в інтеграції кошика та його віджету у додаток. Код нижче оновлюється компонент Shop: до нього додається конструктор з параметром Cart, а також визначається метод додавання товару в кошик.



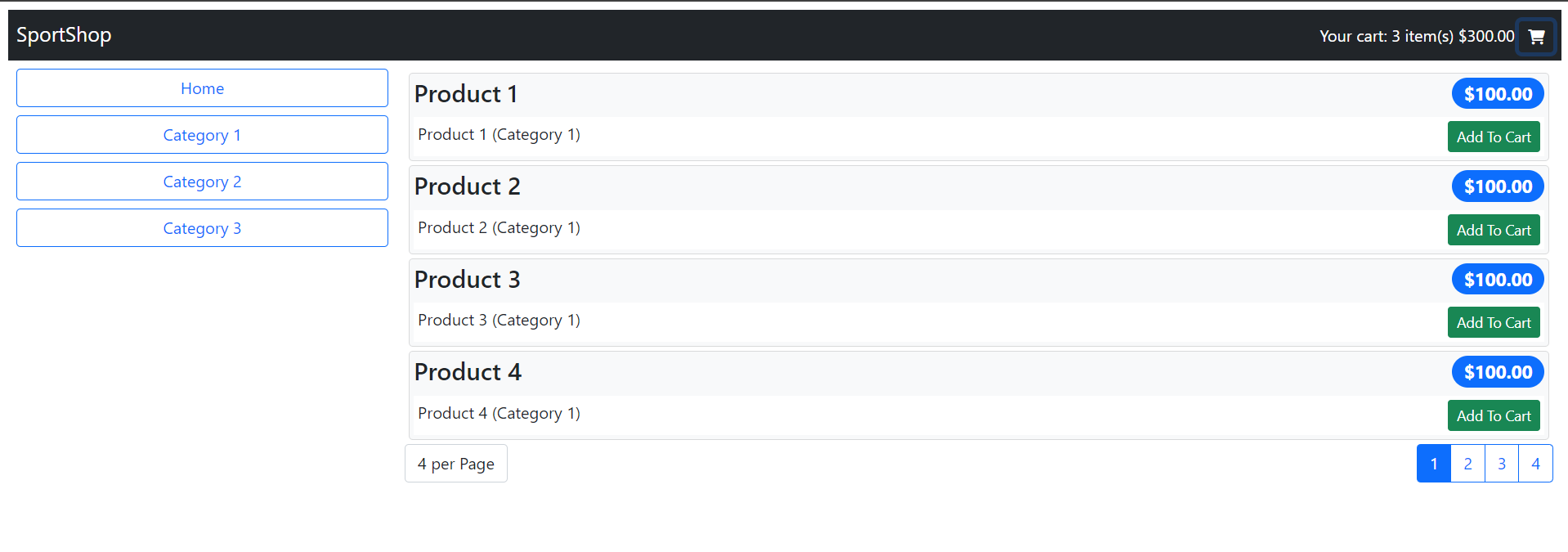


Щоб завершити інтеграцію кошика до компоненту магазину, далі додається елемент, який застосовує компонент зі зведеною інформацією кошика до шаблону компонента магазину та додає в опис кожного товару кнопку з прив'язкою події для виклику методу addProductToCart.





У результаті для кожного товару створюється кнопка додавання до корзини. Повноцінна підтримка кошика ще не реалізована, але наслідки кожного додавання товару відображаються у зведенні у верхній частині сторінки.

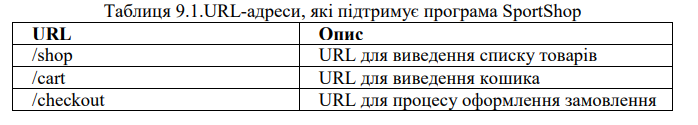


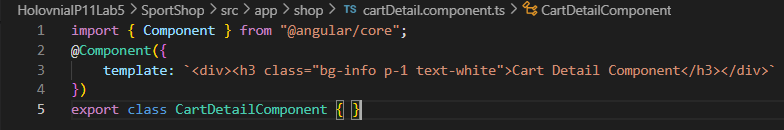
Зверніть увагу: при натисканні однієї з кнопок Add To Cart вміст компонента зведення змінюється автоматично. Це стало можливим завдяки тому, що один об'єкт Cart спільно використовується двома компонентами і зміни, що вносяться одним компонентом, відображаються при обчисленні виразів прив'язок даних в іншому компоненті.

**Маршрутизація URL**

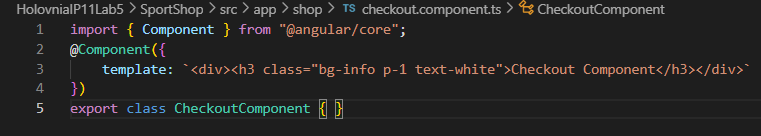
Багато програм відображають різний контент у різні моменти часу. У програмі SportShop при натисканні однієї з кнопок Add To Cart користувач повинен побачити докладний опис обраних товарів, а також отримати можливість запустити процес оформлення замовлення. Angular підтримує механізм маршрутизації URL, який використовує поточну URLадресу в браузері для вибору компонентів, що відображаються для користувача [46, 47, 48]. Цей механізм спрощує створення додатків, в яких компоненти не мають жорсткого зчеплення та легко змінюються без необхідності внесення змін до інших місць.

Маршрутизація URL також дозволяє легко змінити шлях, яким користувач взаємодіє з додатком. У програмі SportShop додамо підтримку трьох різних URL-адрес з табл. 9.1.

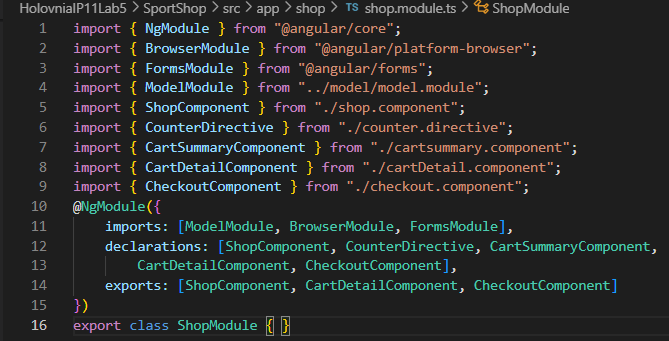




Потім створимо файл checkout.component.ts в папці SportShop/src/app/shop і додамо визначення компонента:



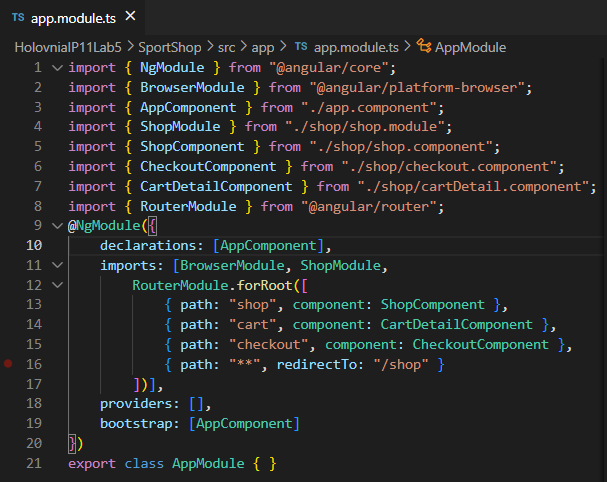
Компонент побудований за тією ж схемою, що і компонент кошика: він виводить тимчасове повідомлення, яке показує, який компонент відображається. У коді компоненти реєструються у функціональному модулі shop і включаються у властивість exports, щоб вони могли використовуватись в інших місцях програми.



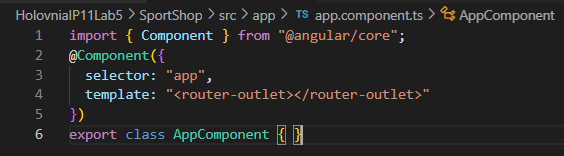
**Створення та застосування конфігурації маршрутизації**

Тепер, коли ми маємо набір компонентів, на наступному кроці створюється конфігурація маршрутизації, яка описує відповідності між URL і компонентами. Кожна відповідність між URL і компонентом називається маршрутом URL, або просто маршрутом (route).

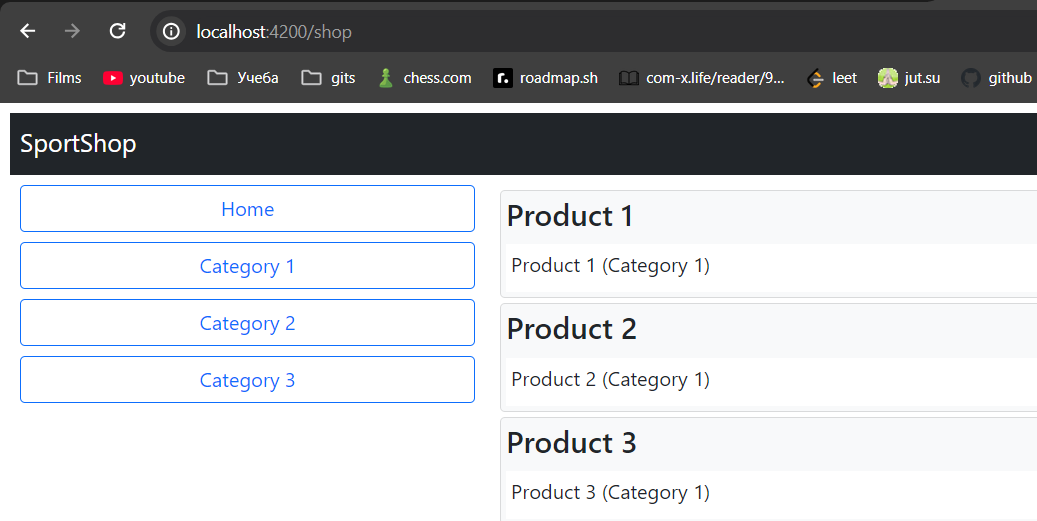
При створенні більш складних конфігурацій маршрутизації, маршрути визначаються в окремому файлі, але в цьому проекті використаємо інше рішення - визначення маршрутів у декораторі @NgModule кореневого модуля програми (лістинг 9.11). Механізм маршрутизації Angular вимагає присутності в документі HTML елемента base, що визначає базову URL-адресу, до якої застосовуються маршрути. Цей елемент був доданий раніше, коли ми створювали проект SportShop. Якщо елемент пропущено, Angular повідомить про помилку та не зможе застосувати маршрути.



Методу RouterModule.forRoot передається набір маршрутів, кожен із яких пов'язує URL з компонентом. Перші три маршрути у лістингу відповідають URL. Останній маршрут є універсальним — він перенаправляє будь-яку іншу URL на /shop, що відображає ShopComponent. При використанні механізму маршрутизації Angular шукає елемент router-outlet, що визначає місце для пошуку компонента, що відповідає поточному URL. У лістингу 9.12 елемент shop шаблону кореневого компонента замінюється елементом router-outlet.



Angular застосовує конфігурацію маршрутизації, коли ви зберігаєте зміни, а браузер перезавантажує HTML документ. Контент, що відображається у вікні браузера, не змінився, але в адресному рядку браузера видно, що конфігурація маршрутизації успішно застосована.

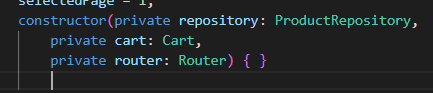


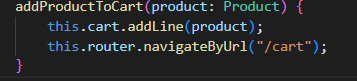
**Навігація у додатку**

Коли конфігурація маршрутизації буде налаштована, можна переходити до підтримки навігації між компонентами зміною URL-адреси у браузері. Механізм маршрутизації URL залежить від JavaScript API, що надається браузером; це означає, що користувач не може просто ввести цільову URL-адресу в адресному рядку браузера. Натомість навігація повинна виконуватися програмою — або з використанням коду JavaScript у компоненті (або іншому структурному блоці), або з додаванням атрибутів до елементів HTML у шаблоні.

Коли користувач клацає на одній із кнопок Add To Cart, повинен відображатись компонент з інформацією кошика; це означає, що програма має перейти за URL-адресою /cart. У коді нижче навігація додається у метод компонента, який викликається при натисканні кнопки користувачем.

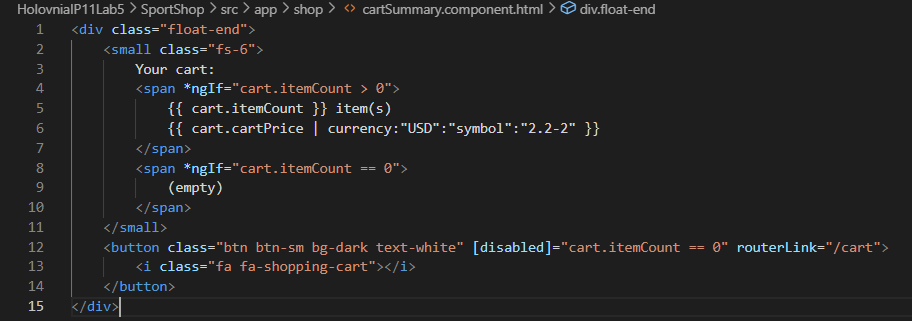




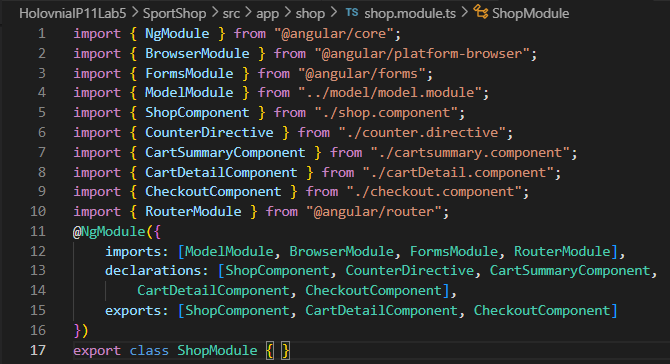


Конструктор отримує параметр Router, що надається Angular через механізм впровадження залежностей при створенні нового екземпляра компонента. У методі addProductToCart метод Router.navigateByUrl використовується для переходу через URL /cart.

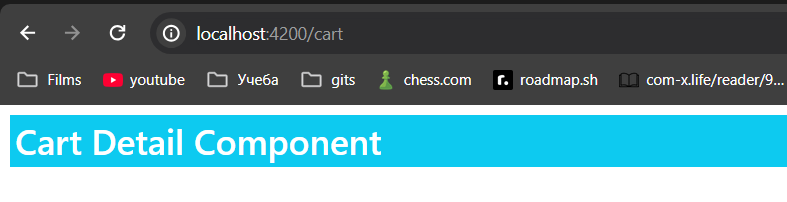
Навігація також може здійснюватися додаванням атрибутуrouterLinkелементи в шаблоні. У коді атрибут routerLink застосовується до кнопки у шаблоні компонента зведеної інформації кошика.



Як значення атрибуту routerLink вказується URL-адреса, за якою має переходити додаток при натисканні на кнопці. Ця кнопка блокується при порожньому кошику, так що перехід буде відбуватися тільки після додавання товару в кошик користувачем. Для додавання підтримки атрибуту routerLink необхідно імпортувати модуль RouterModule у функціональний модуль, як показано нижче:



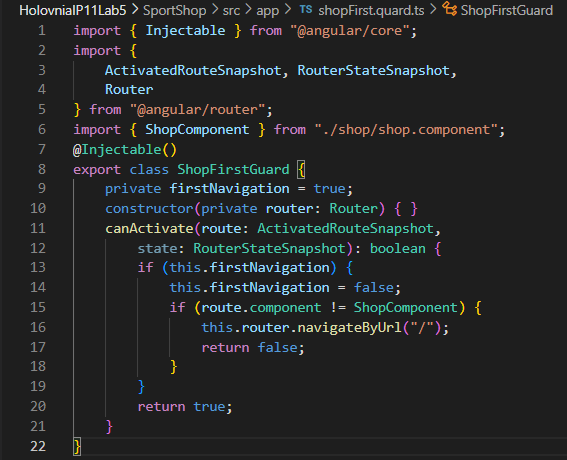
Щоб побачити, як навігація працює, збережемо зміни у файлах, а після того як браузер перезавантажить документ HTML, клацнимо на одній з кнопок Add To Cart. Браузер переходить за URL-адресою /cart



**Захисники маршрутів**

Пам'ятайте, що навігація може виконуватись лише програмою. Якщо змінити URLадресу прямо в адресному рядку браузера, то браузер запросить введену URL-адресу у веб-сервера. Сервер розробки Angular, що відповідає на запити HTTP, відповідає на будь- який запит, що не відповідає файлу, повертаючи вміст index.html. Зазвичай така поведінка зручна, тому що вона запобігає помилці HTTP при натисканні на кнопці оновлення в браузері. Але така поведінка також може створити проблеми, якщо програма очікує, що користувач буде переходити в додатку певним шляхом. Наприклад, якщо клацнути на одній із кнопок Add To Cart, а потім клацнути на кнопці оновлення у браузері, то сервер HTTP поверне вміст файлу index.html, а Angular негайно перейде до компоненту вмісту кошика і пропустить частину програми, що дозволяє користувачеві вибирати продукти. У деяких додатках можливість починати з різних URL-адрес має сенс, а для інших випадків Angular підтримує захисників маршрутів (route guards), що використовуються для керування системою маршрутизації.

Щоб програма не могла починати з URL /cart або /order, додамо файл shopFirst.guard.ts у папку SportShop/app та визначимо клас:

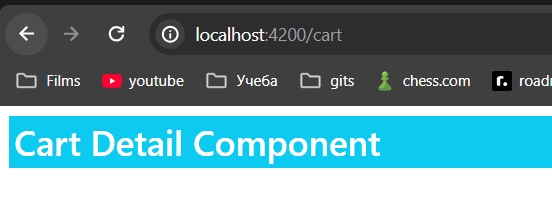


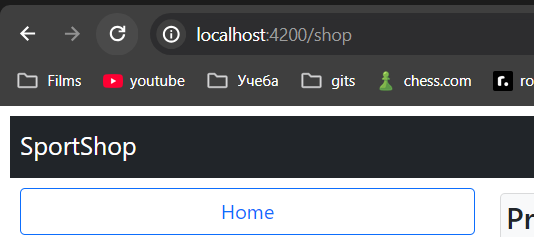
Існують різні способи захисту маршрутів; цей різновид захисника запобігає активізації маршруту. Він реалізується класом, що визначає метод canActivate. Реалізація методу використовує об'єкти контексту, що надаються Angular; по ним він перевіряє, чи є цільовим компонентом ShopComponent. Якщо метод canActivate викликається вперше і використовуватись повинен інший компонент, то метод Router.navigateByUrl використовується для переходу до кореневої URL-адреси. У лістингу застосовується декоратор @Injectable тому, що захисники маршрутів є службами.

Захисник реєструється як служба за допомогою якості providers кореневого модуля та захищає кожен маршрут за допомогою властивості canActivate.



Якщо оновити браузер після клацання на одній із кнопок Add To Cart, побачимо, що браузер автоматично перенаправляється у безпечний стан

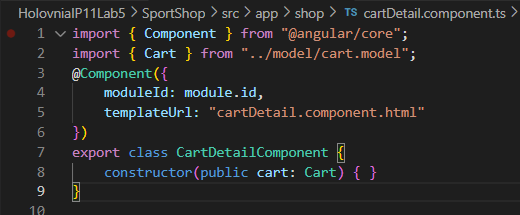




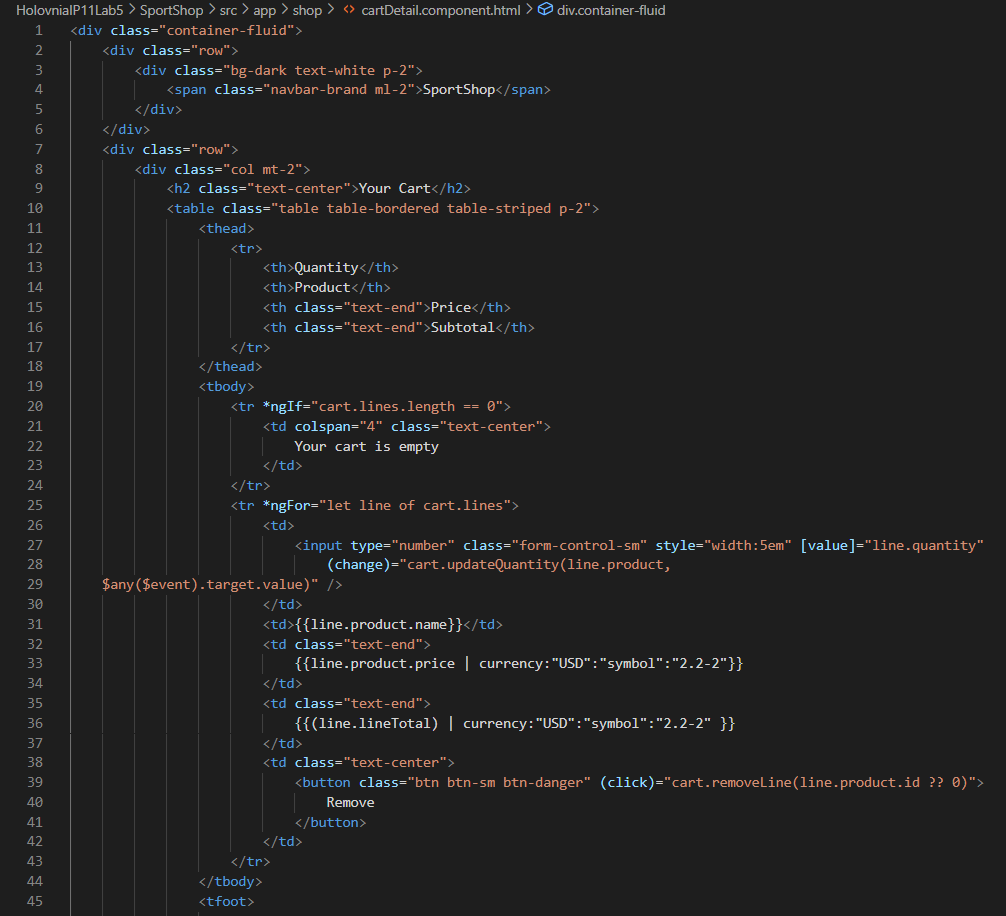
**Завершення виведення вмісту кошика**

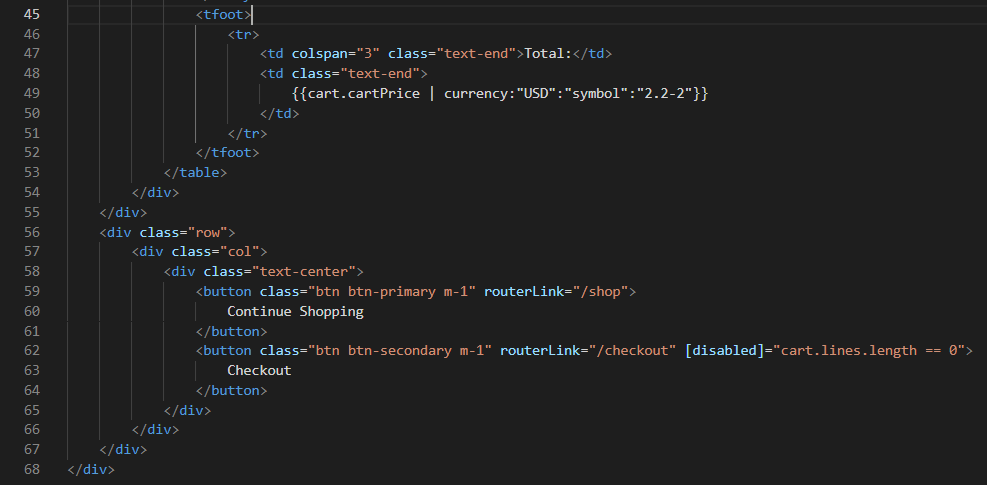
Природний процес розробки програм Angular переходить від підготовки інфраструктури (наприклад, маршрутизації URL) до функцій, видимих користувачеві. Тепер, коли в програмі реалізована підтримка навігації, настав час відобразити подання з докладним вмістом кошику.

У лістингу нижче вбудований шаблон виключається з компонента кошика, замість нього призначається зовнішній шаблон з того ж каталогу, а у конструктор додається параметр Cart, який буде доступний у шаблоні через властивість з ім'ям cart.



Щоб завершити функціональність кошика, створимо файл HTML з ім'ям cartDetail.component.html в папці SportShop/src/app/shop та додамо контент:



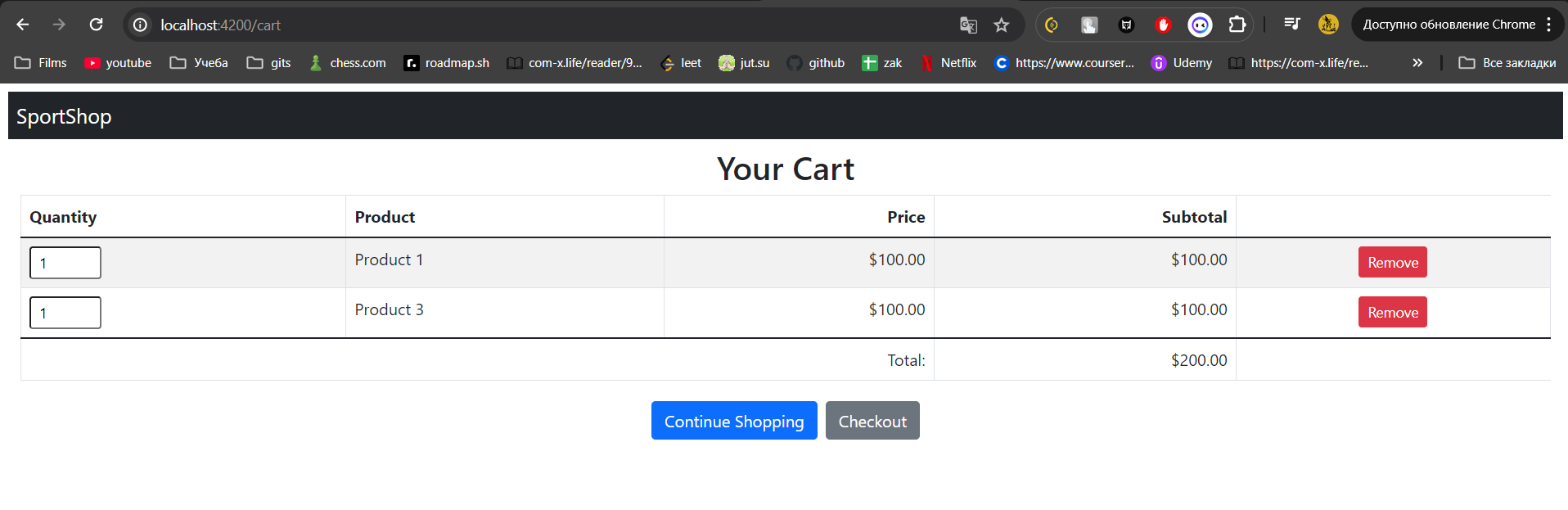


Шаблон виводить таблицю із товарами, обраними користувачем. Для кожного товару створюється елемент input, який може використовуватися для зміни кількості одиниць, та кнопка Remove для видалення товару із кошика. Також створюються дві навігаційні кнопки для повернення до списку товарів та переходу до оформлення замовлення.

Поєднання прив'язок даних Angular та загального об'єкта Cart означає, що будь-які зміни, що вносяться в кошик, призводять до негайного перерахунку ціни, а якщо ви натиснете на кнопці Continue Shopping, зміни відображаються в компоненті зі зведеною інформацією кошика, що відображається над списком товарів

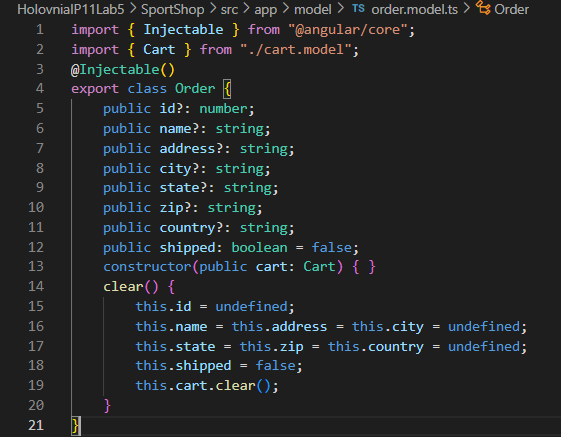
**Обробка замовлень**

Процес обробки замовлень від клієнтів – найважливіший аспект інтернет-магазину. Далі у додатку буде реалізовано підтримку введення додаткової інформації користувачем та оформлення замовлення. Для простоти ми не будемо займатися подробицями взаємодії з платіжними системами, які зазвичай є службовими підсистемами, що не належать до додатків Angular.



**Розширення моделі**

Для опису замовлень від користувачів, створимо файл із ім'ям order.model.ts в папці SportShop/src/app/model і додамо код нижче:

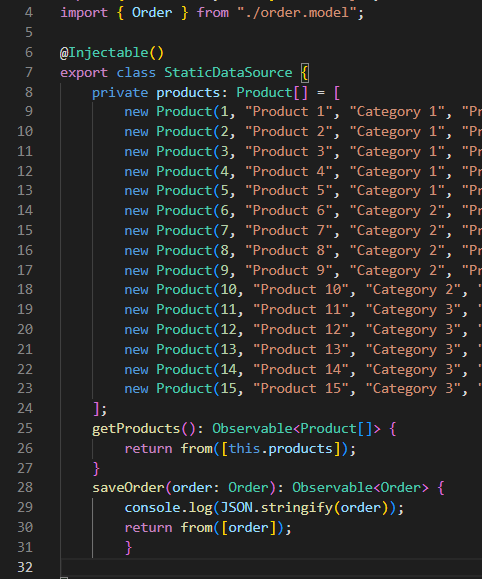


Клас Order також буде оформлено у вигляді служби; це означає, що він існуватиме лише в одному екземплярі, який спільно використовуватиметься в межах програми. Коли середовище Angular створює об'єкт Order, воно виявляє параметр конструктора Cart і надає один і той самий об'єкт Cart.

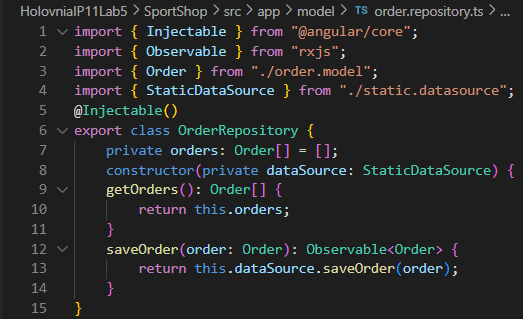
**Оновлення репозиторію та джерела даних**

Для обробки замовлень у додатку необхідно розширити репозиторій та джерело даних, щоб вони могли отримувати об'єкти Order. У коді нижче до джерела даних додається метод отримання замовлення. Оскільки джерело даних поки що залишається фіктивним, метод просто створює на підставі замовлення рядок JSON і виводить його на консоль JavaScript.

У наступному розділі ми створимо джерело даних, що використовує запити HTTP для взаємодії з REST-сумісними веб-службами.

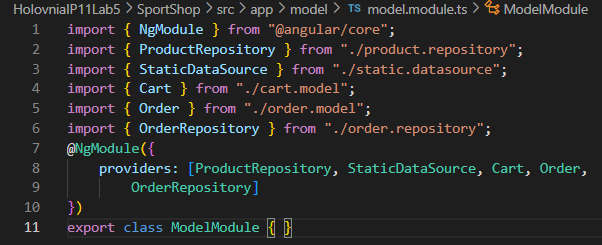


На даний момент репозиторій містить лише один метод, але його функціональність буде розширена далі, коли ми займемося створенням засобів адміністрування. Використовувати різні репозиторії для різних типів моделі у додатку необов'язково. Але рекомендується робити так, тому що один клас, який використовується для декількох типів моделей, стає надто складним і незручним у супроводі.



**Оновлення функціонального модуля**

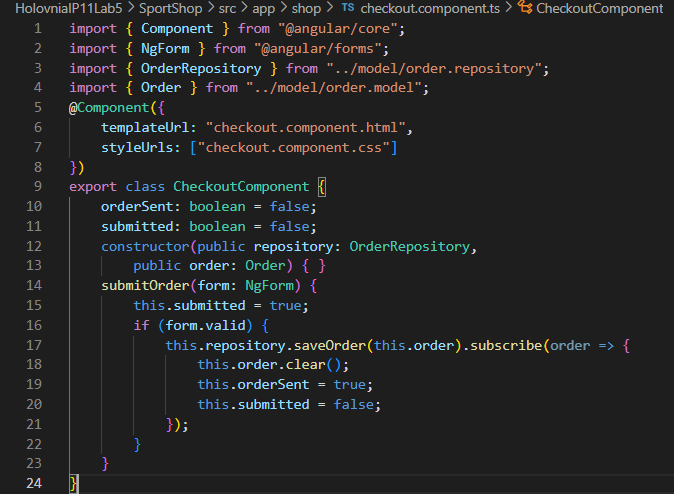
У коді нижче клас Order і новий репозиторій реєструються як служби за допомогою якості providers у функціональному модулі моделі.



**Отримання інформації про замовлення**

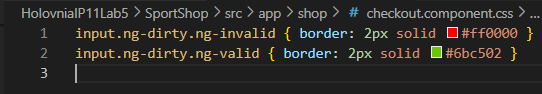
Наступним кроком має стати отримання від користувача додаткової інформації, необхідної для завершення замовлення. Angular включає вбудовані директиви для роботи з формами HTML та перевірки їхнього вмісту.

У коді нижче відбувається підготовка компонента оформлення замовлення, перемикання на зовнішній шаблон, отримання об'єкта Order у параметрі конструктора та реалізація додаткової підтримки роботи шаблону.

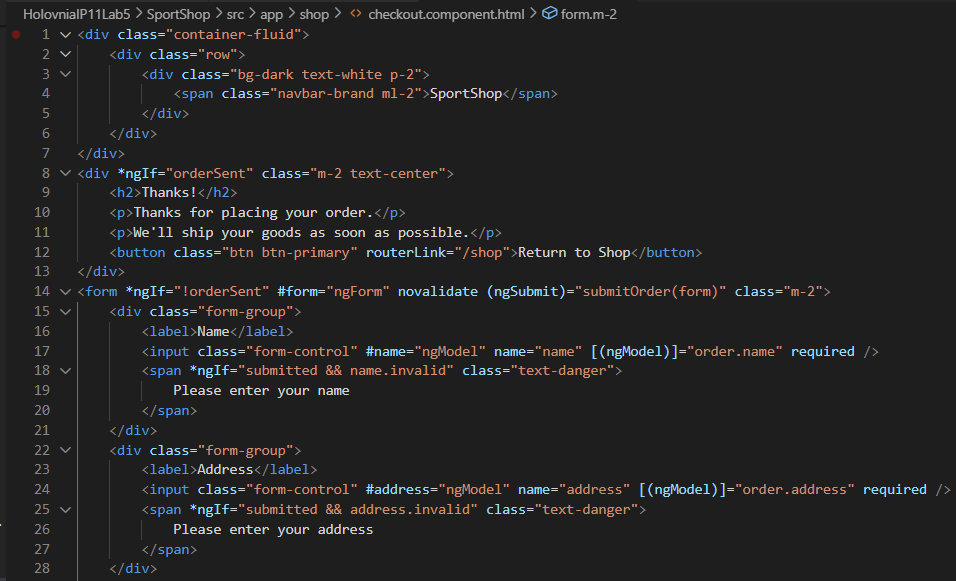


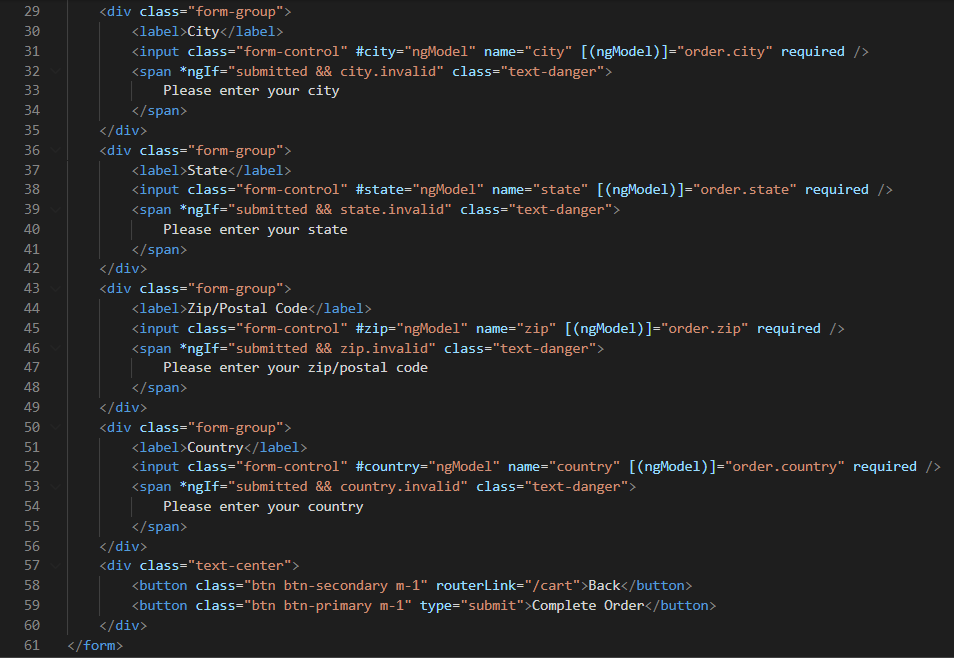
Метод submitOrder буде викликатись при відправленні даних формою, яка представляється об'єктом NgForm. Якщо дані, що містяться у формі, проходять перевірку, то об'єкт Order буде переданий методу saveOrder репозиторія, а дані в кошику та в замовленні скидаються. Властивість styleUrls декоратора @Component використовується для завдання однієї чи кількох стильових таблиць CSS, які мають застосовуватися до контенту шаблону компонента.

Щоб надати зворотній зв'язок перевірки даних для значень, введених користувачем в елементах форми HTML, створимо файл з ім'ям checkout.component.css в папці SportShop/app/shop та визначимо стилі:



Angular додає елементи до класів ng-dirty, ng-validіng-valid для визначення статусу перевірки. Стилі додають зелену рамку до елементів input, що містить перевірені дані, та червону рамку - до недійсних елементів. Останній фрагмент мозаїки — шаблон компонента, який надає користувачеві поля форми, необхідні для заповнення властивостей об'єкта Order

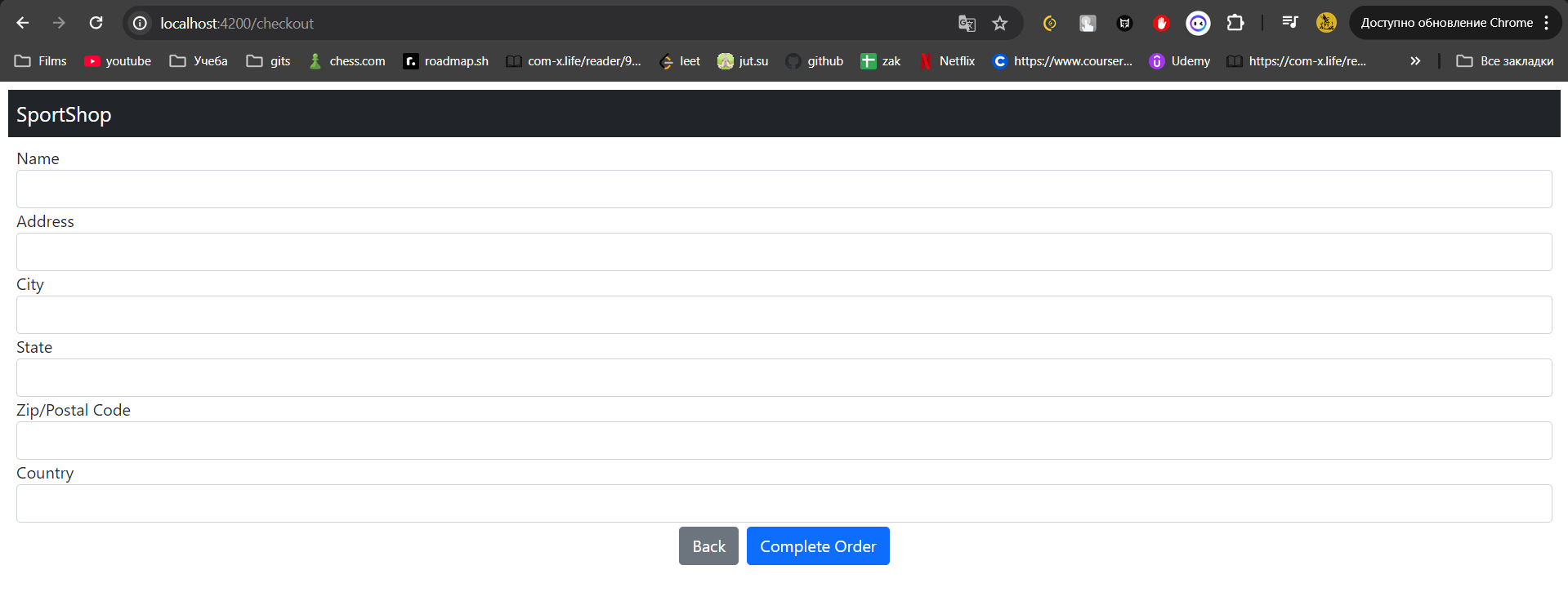


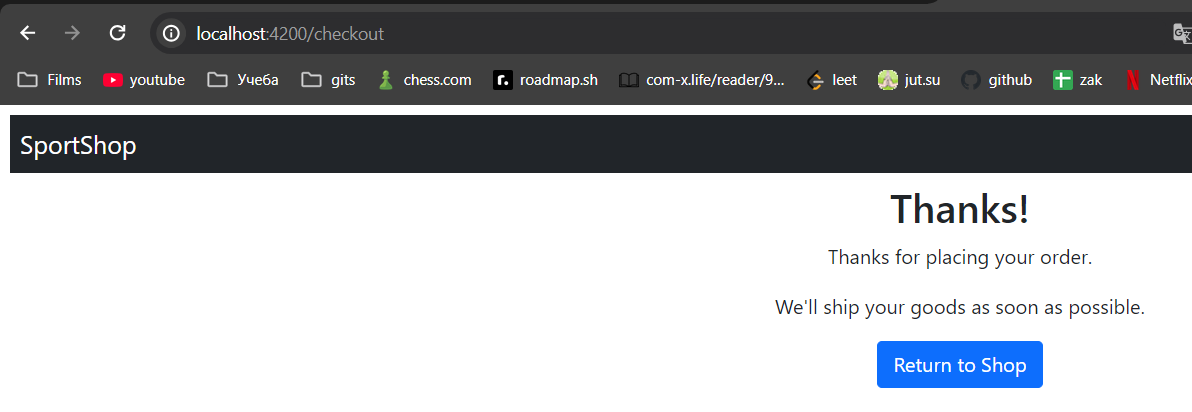


Елементи форми input у цьому шаблоні використовують засоби Angular, щоб переконатися, що користувач ввів значення для кожного поля, і надають візуальний зворотний зв'язок, якщо користувач клацнув на кнопці Complete Order без заповнення форми. Одна частина цього зворотного зв'язку забезпечується застосуванням стилів, а інша – елементами span, які залишаються прихованими, поки користувач не спробує надіслати недійсну форму. Перевірка наявності обов'язкових значень - лише один із способів перевірки полів форми в Angular. Розробник також може легко реалізувати власну, нестандартну перевірку даних.

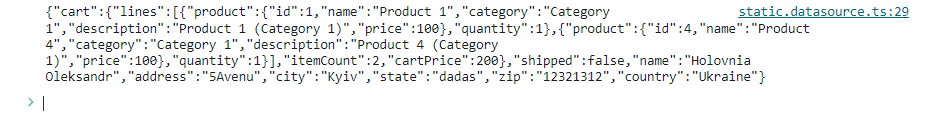
Щоб побачити, як працює цей процес, почнемо зі списку товарів та клацнимо на одній із кнопок Add To Cart, щоб додати товар у кошик. Далі на кнопку Checkout; на екрані з'являється форма HTML. Спробуємо натиснути на кнопку Complete Order, не вводячи текст у жодному полі, і отримаємо повідомлення про помилку перевірки даних.

Заповнимо форму та клацнимо на кнопку Complete Order; з'явиться підтверджуюче повідомлення





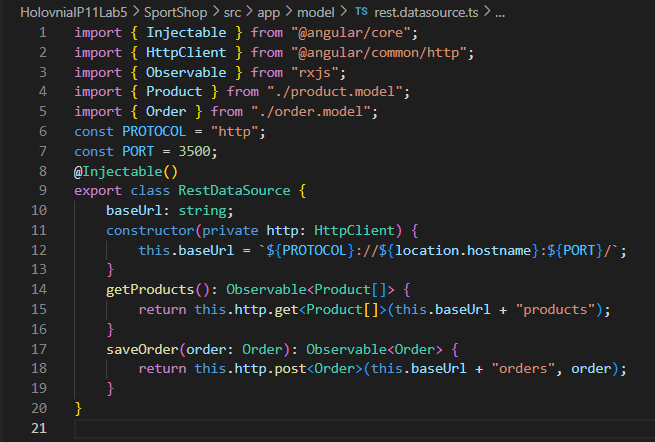
На консолі JavaScript у браузері виводиться подання замовлення у форматі JSON:



**Використання REST-сумісної веб-служби**

Тепер, коли базова функціональність SportShop підготовлена, настав час замінити фіктивне джерело даних іншим, яке отримує дані з REST-сумісної веб-служби, яка була створена під час підготовки проекту раніше.

Щоб створити джерело даних, створимо файл rest.datasource.ts в папці SportShop/src/app/model та додамо наступний код:

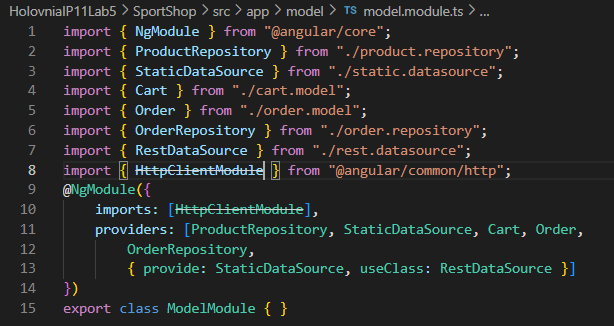


Angular надає вбудовану службу HttpClient, яка використовується для створення запитів HTTP. Конструктор RestDataSource отримує службу Http та використовує глобальний об'єкт location, наданий браузером, для визначення URL-адреси для надсилання запитів; він відповідає порту 3500 на тому хості, з якого було завантажено додаток. Методи, що визначаються класом RestDataSource, відповідають методам, що визначаються статичним джерелом даних, та реалізуються викликом методу sendRequest, який використовує службу HttpClient.

При отриманні даних через HTTP може статися так, що мережевий затор або підвищене навантаження на сервер затримають обробку запиту і користувач буде дивитися на програму, яка не отримала даних. Щоб такого не було треба відповідним чином настроїти систему маршрутизації для запобігання таким проблемам.

**Застосування джерела даних**

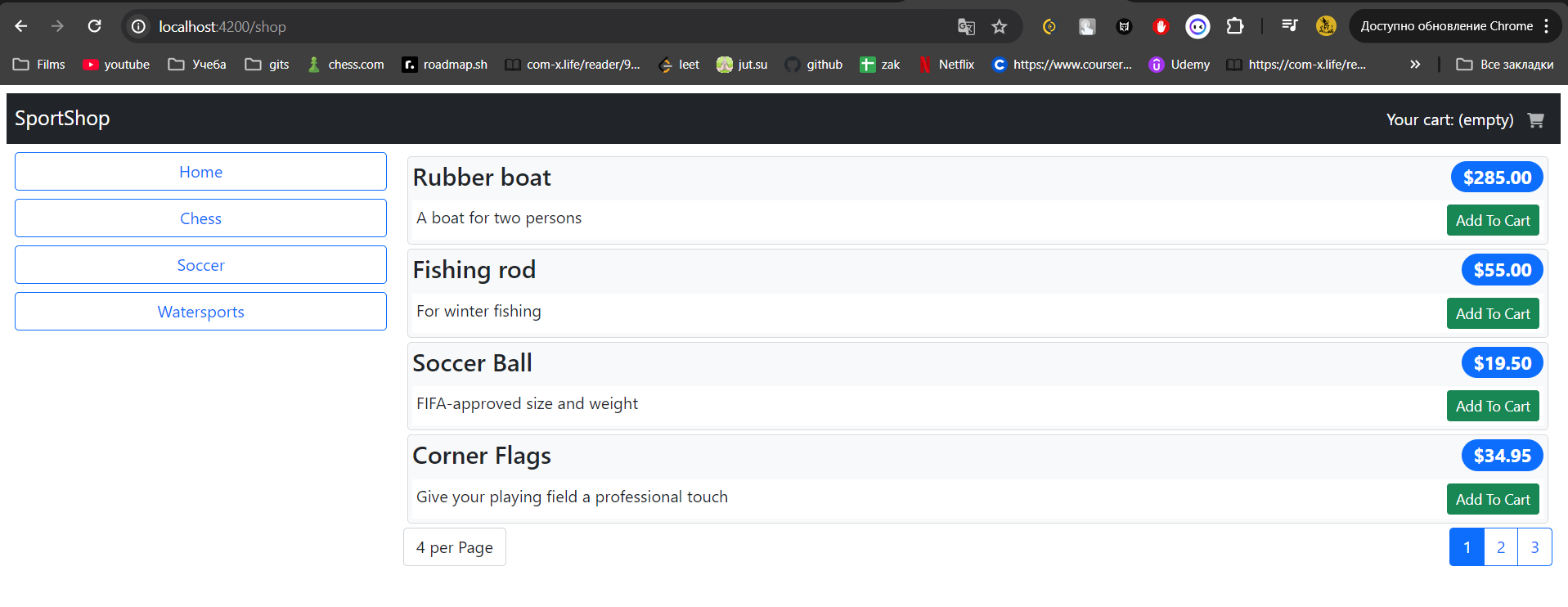
Тепер застосуємо REST-сумісне джерело даних. Для цього перебудуємо додаток, щоб перехід з фіктивних даних на дані REST здійснювався змінами лише в одному файлі. У коді нижче поведінка джерела даних змінюється у функціональному модулі моделі.

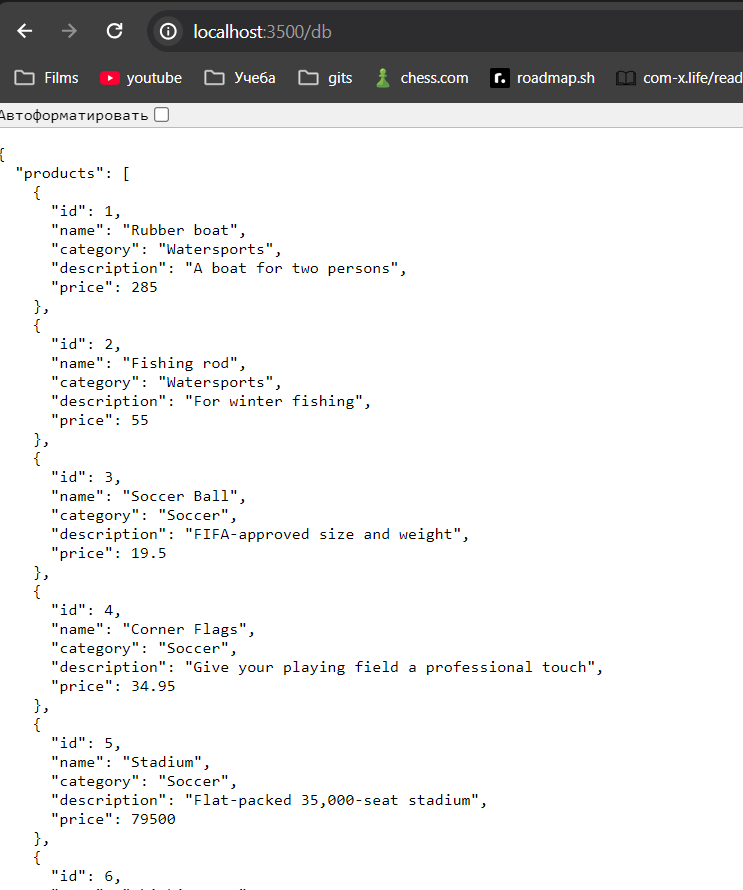


Властивість imports використовується для оголошення залежності від функціонального модуля HttpModule, який надає службу Http. Зміна у властивості providers повідомляє Angular, що коли потрібно створити екземпляр класу з параметром конструктора StaticDataSource замість нього слід використовувати RestDataSource.

Так як обидва об'єкти визначають однакові методи, завдяки динамічній системі типів JavaScript заміна повинна пройти гладко. Після того, як усі зміни будуть збережені, а браузер перезавантажить програму, фіктивні дані замінюються даними, отриманими через HTTP. Якщо пройти процедуру вибору товарів та оформлення замовлення, ви зможете переконатися, що джерело даних записало замовлення до веб-служби.

Перейдемо за наступною URL-адресою:





В цій роботі ми продовжили розширювати функціональність програми SportShop: до програми була додана підтримка кошика для вибору товарів користувачем та процесу оформлення замовлення, що завершує процес покупки. Також у цій частині роботи фіктивне джерело даних було замінено джерелом, що надсилає запити HTTP RESTсумісній веб-службі.

**Посилання на додатки:**

<https://holovniaip11sportshop1.web.app/>

**Висновок:**

Під час виконання комп’ютерного практикуму я навчився створювати Angular-додатки, які містять сервіси, pipes, директиви та використовують бібліотеку RxJS. Створив проект, що складається з двох частин:  
Встановив та налаштував засоби розробника, створив кореневі структурні блоки для проекту (модель даних, фіктивне джерело даних, репозиторій моделі, сховище, компоненти магазина та шаблони). Вивів дані фіктивної моделі даних на головну сторінку магазину, реалізував розбивку на сторінки та фільтрацію товарів за категоріями. Створив нестандартну директиву для пагінації.

Додаток, отриманий в результаті виконання Частини 1 розгорнутий на платформі Firebase.

Далі розробив додаткову логіку в додатку «SportShop» для виборутоварів та оформлення замовленнь. Реалізував підтримку кошика для вибору товарів користувачем та процесу оформлення замовлення. Фіктивне джерело даних у проекті замінив джерелом, що надсилає запити до HTTP REST-сумісної веб-служби.