**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Сумський державний університет**

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ
(підпис)

«  » червня 2024 р.

# КВАЛІФІКАЦІЙНА РОБОТА

## на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна система пошуку комплементарних товарів на основі колаборативної фільтрації»
здобувача групи ІН-01 Горлача Ярослава Андрійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Ярослав ГОРЛАЧ
(підпис)

Керівник,
старший викладач кафедри
комп'ютерних наук,
к.т.н., доцент Борис КУЗІКОВ _____
(підпис)

Суми – 2024

**Сумський державний університет**

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ
(підпис)

# ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

**на здобуття освітнього ступеня бакалавра**

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-01 Горлача Ярослава Андрійовича

1. Тема роботи: «Інформаційна система пошуку комплементарних товарів на основі колаборативної фільтрації»
   затверджую наказом по СумДУ від *«22» квітня 2024 р. № 0414-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 29 травня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
6. Консультанти до проекту (роботи), із зазначенням розділів проекту, що стосується їх

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |

7. Дата видачі завдання «__» _____ 20__ р.

Завдання прийняв на виконання _____ Керівник _____
(підпис) (підпис)

# КАЛЕНДАРНИЙ ПЛАН

| № п/п | Назва етапів кваліфікаційної роботи | Термін виконання | Примітка |
|---|---|---|---|
| *1* | *Аналіз проблематики предметної області, формулювання і постановка завдань дослідження* | | |
| *2* | *Огляд технологій, що застосовуються для рекомендації комплементарних товарів* | | |
| *3* | *Розробка інформаційної системи пошуку комплементарних товарів на основі колаборативної фільтрації* | | |
| *4* | *Аналіз отриманих результатів* | | |
| *5* | *Оформлення пояснювальної записки до кваліфікаційної роботи* | | |

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

# АНОТАЦІЯ

Записка: 55 стр., 32 рис., 1 додаток, 22 використаних джерел.

**Обґрунтування актуальності теми роботи** – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв'язанню важливої практичної задачі пошуку комплементарних товарів на основі колаборативної фільтрації шляхом розробки відповідних методів, моделей та інформаційної технології.

**Об'єкт дослідження** — системи рекомендацій в електронній комерції.

**Мета роботи** — розробка інформаційної системи пошуку комплементарних товарів на основі колаборативної фільтрації.

**Методи дослідження** — методи колаборативної фільтрації.

**Результати** — розроблено інформаційну систему, яка відносно покупок зроблених іншими користувачами, підбирає комплементарні товари, які будуть до вподобання саме для того користувача, що додає їх в кошик. Проведено тестування розробки на реальних даних про вина, а також створених заздалегідь користувачів і кошиків користувачів.

ІНФОРМАЦІЙНА СИСТЕМА, КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ, ПОШУК КОМПЛЕМЕНТАРНИХ ТОВАРІВ, ANGULAR, ASP.NET CORE, REDIS, SQLITE.

# ЗМІСТ

# ВСТУП

**Актуальність.** Сучасний світ майже неможливо уявити собі без веб магазинів. З їх допомогою клієнти можуть отримувати різноманітні товари та послуги з мінімальними зусиллями, маючи при собі лише телефон або комп’ютер зі стабільним доступом до інтернету, а також банківську картку.

Однією з основних переваг веб-магазинів є можливість швидкого доступу до широкого асортименту товарів і зручності їх покупки. Це дозволяє користувачам обирати товари на основі власних потреб та вподобань, що допомагає економити час замовляючи товари за декілька секунд і отримувати ті товари, яких може не бути в звичайних магазинах. Однак, часто користувачам важко самостійно визначити, які товари можуть бути корисними в доповнення до вже обраних або ж вони не здогадуються, що є якісь товари, які б могли бути до вподобання саме їм. Це створює потребу в інструментах, які можуть автоматично підбирати комплементарні товари.

На сайтах, що продають різноманітні товари, користувачі мають можливість додавати товари до кошику. Однак для поліпшення користувацького досвіду недостатньо лише можливості переглядати та обирати товари самостійно, було б дуже зручно мати певний список рекомендацій в самому кошику, який би допоміг знайти для себе щось нове і несподіване, або ж просто корисне. Більш ефективним було б впровадження автоматизованої системи підбору комплементарних товарів, яка б надавала рекомендації на основі аналізу вже обраних товарів користувача, спираючись на виборі інших користувачів. Використання колаборативної фільтрації для цього завдання дозволяє аналізувати поведінку користувачів і їхні покупки, щоб надавати персоналізовані рекомендації щодо комплементарних товарів.

**Об’єкт дослідження:** система рекомендацій в електронній комерції.

**Предмет дослідження:** методи колаборативної фільтрації для генерації рекомендацій комплементарних товарів.

**Гіпотеза.** Використання алгоритмів колаборативної фільтрації у системах рекомендацій електронної комерції значно покращить точність і релевантність пошуку комплементарних товарів, що призведе до підвищення рівня задоволеності користувачів та збільшення конверсії на сайті.

**Новизна.** Інформаційна система реалізує колаборативну фільтрацію для інтернет магазину вин.

**Практичною значимістю** буде те, що застосування розроблюваного веб-додатку зменшить кількість необхідних ресурсів для пошуку комплементарних товарів та скоротить час на прийняття рішень щодо покупок. Використання колаборативної фільтрації для аналізу куплених товарів користувачів та надання рекомендацій щодо комплементарних товарів окремо від основного списку покупок значно покращить користувацький досвід.

**Структура.** Дане робота складається зі вступу, аналітичного огляду, постановки задачі, вибір методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

# 1. АНАЛІТИЧНИЙ ОГЛЯД

## 1.1. Моделі і методи колаборативної фільтрації

Більшість сучасних комерційних систем використовують методи колаборативної фільтрації для рекомендації товарів користувачам. Ці методи базуються на припущенні, що користувачі з подібними вподобаннями будуть обирати схожі товари, а подібні об'єкти використовуються спільно різними користувачами. Схожість між об'єктами та користувачами визначається не за їхнім вмістом, а за історією їхніх взаємодій. Цю історію можна представити у вигляді матриці, де стовпці відповідають об'єктам, рядки — користувачам, а значення вказують на рівень інтересу користувача до об'єкта [1].

Оцінки можна поділити на прямі, такі як рейтинг об'єкта за певною шкалою, та непрямі, наприклад, дані про поведінку користувача: кількість переглядів, час взаємодії з об'єктом тощо. Прямі оцінки більш інформативні, оскільки відображають безпосереднє ставлення користувача до об'єкта, але вимагають активної участі користувачів у процесі збору інформації. Альтернативним підходом є використання непрямих оцінок, які отримують шляхом фіксації поведінки користувачів. Однак непрямі оцінки мають свої особливості, такі як відсутність негативних оцінок, наявність шуму та інша семантика числових значень, що ускладнює їх аналіз методами, розробленими для прямих рейтингів [1].

Методи колаборативної фільтрації можна згрупувати у два загальні класи: методи на основі сусідів (neighborhood) та методи на основі моделей (model-based). Також існують гібридні методи. У фільтрації на основі сусідів або на основі пам'яті, оцінки елементів користувача, що зберігаються безпосередньо в системі, використовуються для прогнозування рейтингів нових об'єктів. Ця фільтрація може бути виконана двома способами, відомими як рекомендація на основі користувачів (user-based) або на основі елементів (item-based). Системи на основі користувачів, такі як GroupLens, оцінюють зацікавленість користувача *u* в об'єкті *i* на основі оцінок цього об'єкта іншими

користувачами, які називаються сусідами, що мають схожі моделі оцінювання. Сусідами користувача *u* зазвичай є користувачі *v*, чиї рейтинги на об'єктах, оцінених як *u* так і *v*, тобто $I_{uv}$, найбільше корелюють з оцінками *u*. Підходи на основі елементів, з іншого боку, передбачають рейтинг користувача *u* для елемента *i* на основі рейтингів *u* для елементів, схожих на *i*. У таких підходах два елементи вважаються схожими, якщо кілька користувачів системи оцінили їх однаково [2].

На відміну від систем на основі сусідства, які використовують збережені рейтинги безпосередньо у прогнозуванні, підходи, що базуються на моделях, використовують ці рейтинги для навчання прогностичної моделі. Загальна ідея полягає в моделюванні взаємодії користувача та об'єкта з факторами, що представляють латентні характеристики користувачів та об'єктів у системі, такі як клас уподобань користувачів та клас категорій об'єктів. Потім ця модель навчається на наявних даних, а згодом використовується для прогнозування оцінок користувачів для нових товарів. Підходи на основі моделей для завдання рекомендації товарів є численними і включають латентний семантичний аналіз, латентний розподіл Діріхле і т.д. [2].

Основними перевагами методів, що базуються на сусідстві, є:

- Простота - методи на основі сусідів інтуїтивно зрозумілі та відносно прості в реалізації. У найпростішій формі лише один параметр (кількість сусідів, що використовуються для прогнозування) потребує налаштування.
- Обґрунтованість - такі методи також надають стисле та інтуїтивно зрозуміле обґрунтування отриманих прогнозів.
- Ефективність - однією з сильних сторін систем, основаних на сусідстві, є їхня ефективність. На відміну від більшості систем, заснованих на моделях, вони не потребують затратних етапів навчання, які необхідно проводити через часті проміжки часу у великих комерційних додатках. Більше того, зберігання цих найближчих сусідів вимагає дуже мало пам'яті, що робить такі

підходи масштабованими для додатків з мільйонами користувачів та об'єктів.

- Стабільність - ще однією корисною властивістю рекомендаційних систем, заснованих на цьому підході, є те, що на них мало впливає постійне додавання користувачів, товарів і оцінок, яке зазвичай спостерігається у великих комерційних додатках. [2]

Колаборативна фільтрація також має недоліки, такі як:

- Розрідженість даних - велика матриця користувацьких елементів часто є розрідженою через велику кількість даних, що може ускладнити процес рекомендацій.
- Проблеми з масштабованістю - при великій кількості елементів і користувачів виникають проблеми з продуктивністю, що вимагає потужних обчислювальних ресурсів.
- Проблеми синонімів - однакові або дуже схожі елементи можуть мати різні назви, що ускладнює їх ідентифікацію.
- Проблема "сірих овець" - користувачі з унікальними вподобаннями можуть не отримати корисних рекомендацій, оскільки система не здатна адекватно врахувати їхні потреби.
- Проблема шилінгових атак - навмисне завищення або заниження оцінок з метою маніпуляції системою рекомендацій, що знижує їхню точність. [3]

### 1.1.1. Прогнозування рейтингу на основі даних користувачів

Методи рекомендацій на основі сусідства передбачають рейтинг $r_{ui}$ користувача $u$ для нового елемента $i$, використовуючи оцінки, виставлені $i$ користувачами, найбільш схожими на $u$, які називаються найближчими сусідами. Припустимо, що для кожного користувача $v \neq u$ ми маємо значення $w_{uv}$, що представляє схожість переваг між $u$ та $v$. $k$ найближчими сусідами (k-nearest-neighbors or $k$-$NN$) $u$, позначеними через $N(u)$, є $k$ користувачів $v$ з найвищою схожістю $w_{uv}$ з $u$. Однак, тільки користувачі, які оцінили позицію $i$,

можуть бути використані для прогнозування $r_{ui}$, тому ми розглядаємо $k$ користувачів, найбільш схожих на $u$, які оцінили $i$. Ми записуємо цю множину сусідів як $N_i(u)$. Рейтинг $r_{ui}$ можна оцінити як середню оцінку, дану $i$ цими сусідами:

$$\hat{r}_{ui} = \frac{1}{|N_i(u)|}\sum_{v \in N_i(u)} r_{vi} \qquad (1.1)$$

Проблема з (1.1) полягає в тому, що вона не враховує той факт, що сусіди можуть мати різний рівень схожості. Поширеним рішенням цієї проблеми є зважування внеску кожного сусіда за його схожістю до $u$. Однак, якщо ці ваги не дорівнюють 1, то прогнозовані рейтинги можуть бути далеко за межами діапазону допустимих значень. Тому прийнято нормалізувати ці ваги таким чином, щоб передбачуваний рейтинг набув вигляду:

$$\hat{r}_{ui} = \frac{\sum_{v \in N_i(u)} w_{uv} r_{vi}}{\sum_{v \in N_i(u)} |w_{uv}|} \qquad (1.2)$$

У знаменнику (1.2) замість $w_{uv}$ використовується $|w_{uv}|$, оскільки від'ємні ваги можуть призвести до того, що рейтинги вийдуть за межі дозволеного діапазону. Також $w_{uv}$ можна замінити на $w_{uv}^{\alpha}$, де $\alpha > 0$ - коефіцієнт підсилення [4]. Коли $\alpha > 1$, як це найчастіше використовується, ще більше значення надається сусідам, які є найближчими до $u$ [2].

### 1.1.2. Класифікація на основі користувачів

Щойно описаний підхід до прогнозування, де прогнозовані рейтинги обчислюються як середньозважене значення рейтингів сусідів, по суті, вирішує проблему регресії (regression). Класифікація на основі сусідів (Neighborhood-based classification) [5], з іншого боку, знаходить найбільш ймовірну оцінку, яку користувач $u$ поставив об'єкту $i$, шляхом голосування найближчих сусідів $u$ за це значення. Голос $v_{ir}$, відданий $k$-$NN$ користувача $u$ за оцінку $r \in S$ ($S$ - набір можливих значень для рейтингу), може бути отриманий як сума ваг подібності сусідів, які дали цю оцінку $i$:

$$v_{ir} = \sum_{v \in N_i(u)} \delta(r_{vi} = r) w_{uv} \qquad (1.3)$$

де $\delta(r_{vi} = r)$ дорівнює 1, якщо $r_{vi} = r$, і 0 в іншому випадку. Після того, як це було обчислено для кожного можливого значення рейтингу, прогнозований рейтинг - це просто значення $r$, для якого $v_{ir}$ є найбільшим [2].

## 1.2. Сучасні системи пошуку товарів за допомогою колаборативної фільтрації.

На сайті Netflix[6] колаборативна фільтрація використовується для надання рекомендацій щодо фільмів та серіалів, які можуть зацікавити користувачів на основі їхньої переглядової історії. Наприклад, якщо користувач переглянув декілька фільмів у жанрі біографії, система може рекомендувати інші популярні фільми жанру біографія, які дивилися схожі користувачі (див. Рисунок 1.1 ).

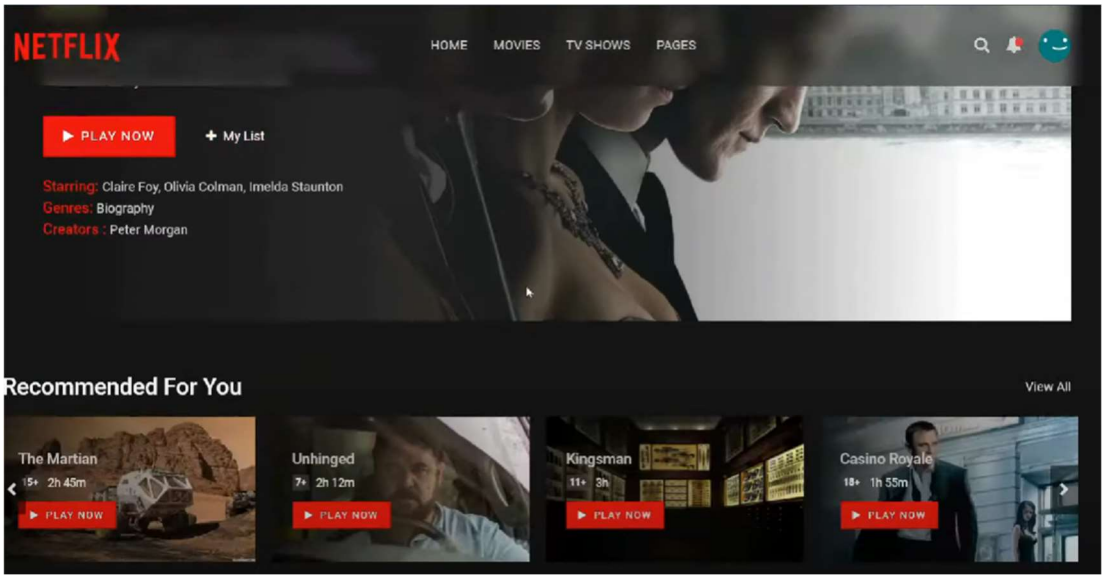


Рисунок 1.1 - Рекомендації на сторінці біографічного фільму на сайті Netflix[6].

На сайті eBay[7] колаборативна фільтрація допомагає користувачам знаходити комплементарні товари до того товару, який ще не був доданий до кошику. Наприклад коли обираємо кросівки нам будуть запропоновані схожі кросівки, або ті, які б могли нам підійти (див. Рисунок 1.2).

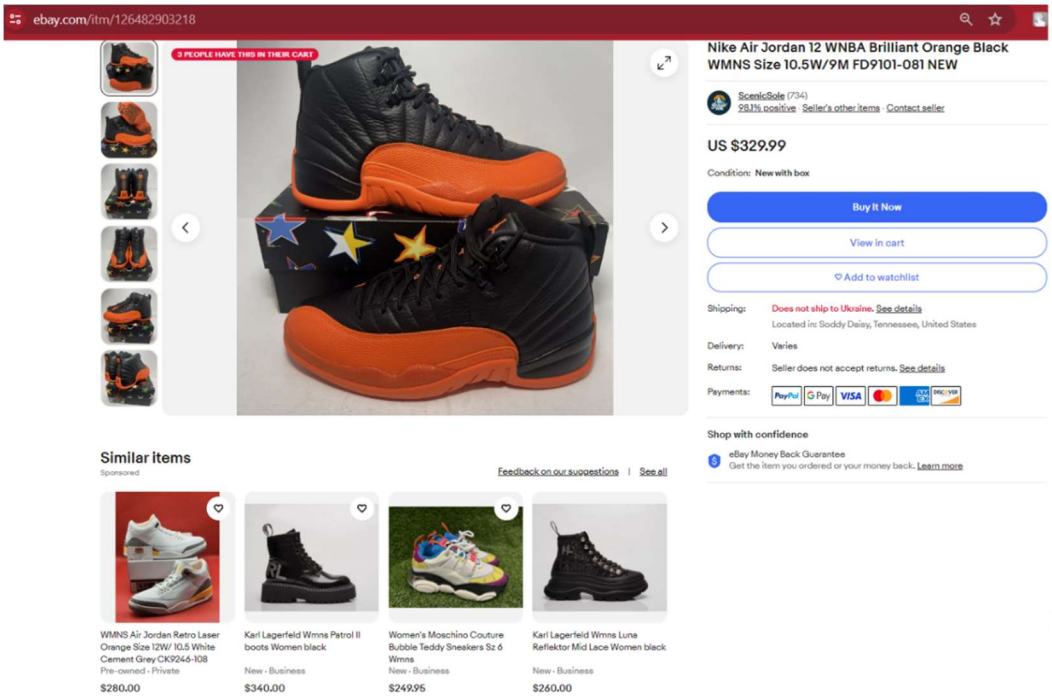


Рисунок 1.2 – Сторінка товару кросівок з комплементарними товарами на сайті eBay[7].

На сайті Rozetka[8] колаборативна фільтрація використовується для рекомендацій товарів, які доповнюють ті, що переглядає або купує користувач. Наприклад, якщо користувач переглядає телефон (див. Рисунок 1.3), система може рекомендувати чохли, захисне скло, зарядні пристрої та інші аксесуари (див. Рисунок 1.4).

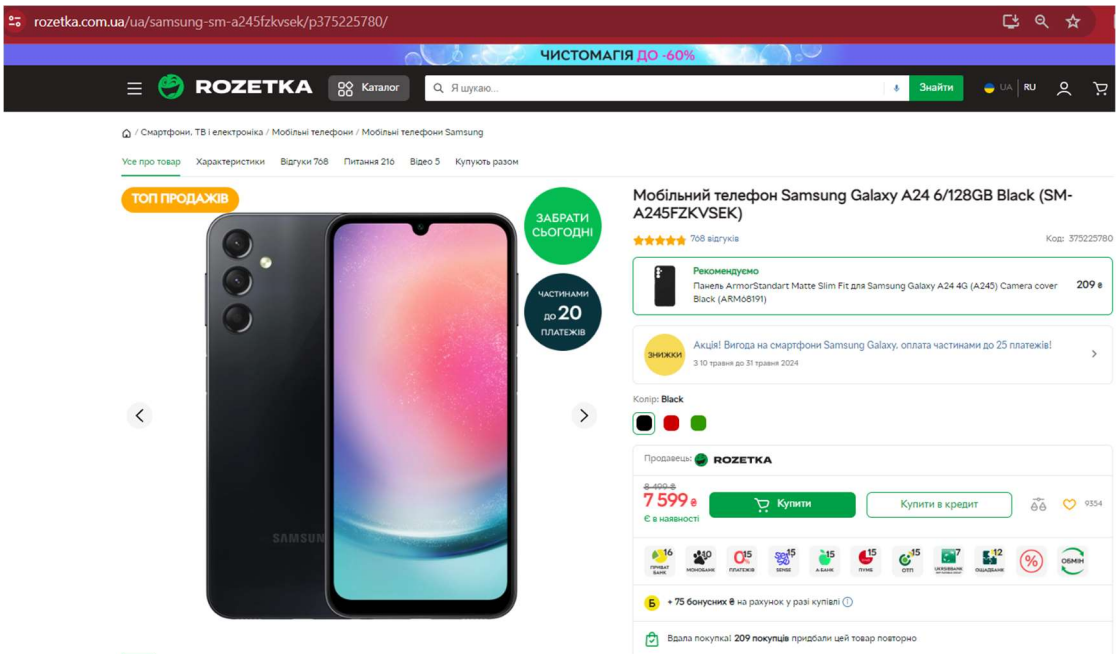


Рисунок 1.3 – Сторінка товару мобільного телефону на сайті Rozetka[8].

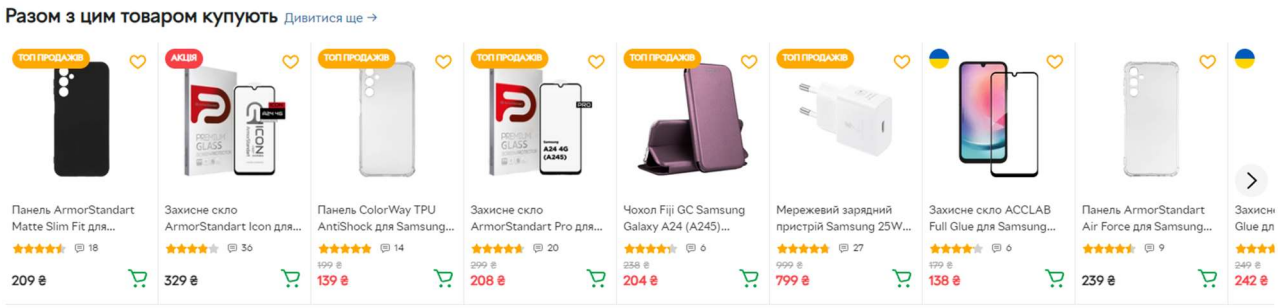


Рисунок 1.4 – Блок "Разом з цим товаром купують" на сторінці товару мобільного телефону на сайті Rozetka[8]

На Amazon[9] колаборативна фільтрація допомагає надавати рекомендації, засновані на переглядах і покупках інших користувачів. Наприклад, після покупки книги та телефону, користувачам можуть рекомендувати інші книги того ж автора або в тому ж жанрі, а також певні аксесуари до телефону, або ж телефони інших фірм (див. Рисунок 1.5.).

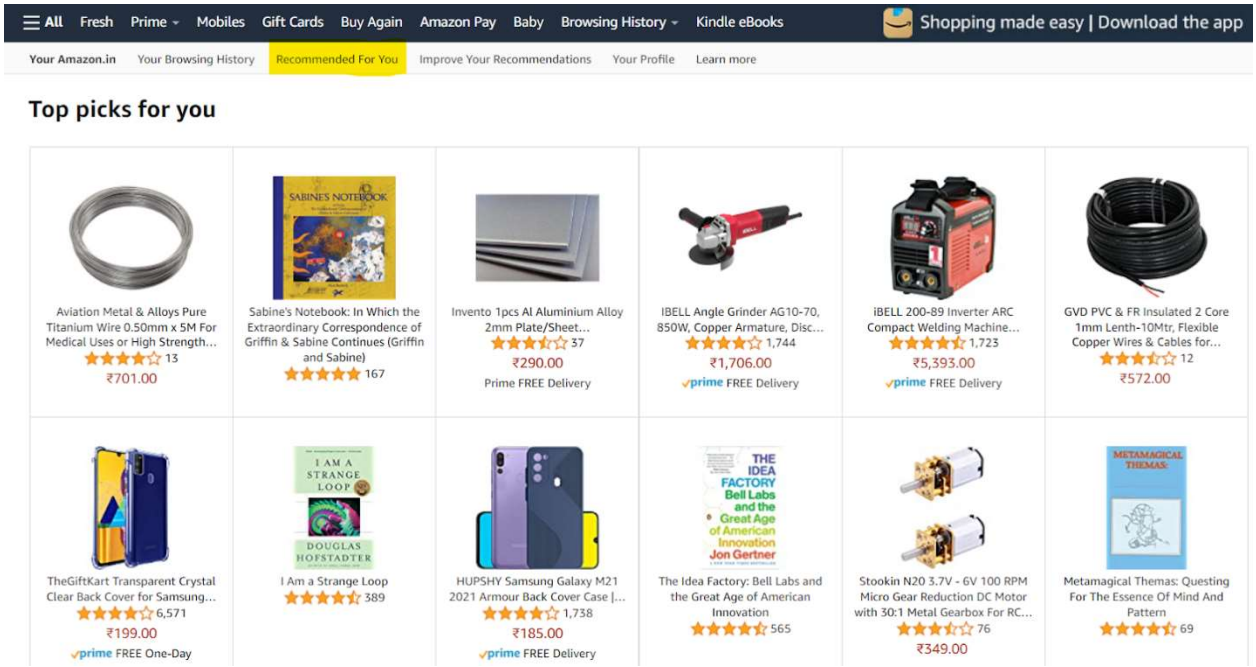


Рисунок 1.5. – Сторінка "Top picks for you" на сайті Amazon[9]

## 1.3. Single page app та їх особливості

Для того, щоб продемонструвати роботу інформаційної системи, нами було створено власний веб магазин. Так як для написання клієнтської частини було обрано фреймворк Angular, який націлений на розробку веб додатків типу Single Page Application (SPA) [10], розглянемо плюси і мінуси SPA, а також

альтернативи, щоб зрозуміти, чим SPA кращий тип саме для написання веб магазину.

Single Page Application (SPA) - це тип веб-додатка, який відомий своєю властивістю завантажувати лише одну сторінку HTML під час першого завантаження, а після цього взаємодія з користувачем відбувається без перезавантаження сторінки. В SPA весь вміст та функціональність завантажуються асинхронно за допомогою AJAX-запитів, а зміна URL-адреси в браузері використовується для створення історії перегляду та для закладання закладок.

Основними альтернативами до SPA є Multi-Page Application (MPA) та Server-Side Rendering (SSR). У MPA для кожної сторінки створюється окрема HTML-сторінка, і перехід між сторінками відбувається шляхом перезавантаження всієї сторінки. У SSR весь вміст створюється на стороні сервера та відправляється клієнту як HTML сторінка. Після завантаження клієнтський код виконується для підсилення інтерактивності.

Основним плюсом SPA є те, що весь вміст знаходиться на одній сторінці HTML, яку не треба перезавантажувати, після чого перехід між сторінками здійснюється шляхом маршрутизації, що також дає змогу зберігати історію перегляду сторінок. В цьому плані SPA кращий за MPA, бо для кожної сторінки в MPA потрібна окрема HTML-сторінка і при переході між ними, треба її перезавантажувати. Для нашого випадку, швидка відповідь на дії користувача, а також можливість маршрутизації були найважливішими при виборі типу веб додатку.

Основними ж недоліками SPA є пошукова оптимізація (SEO) та проблеми з безпекою. Так як при використанні SPA пошукові системи, окрім Google, не здатні обробляти JavaScript, на якому написаний сайт, без додаткової оптимізації вони не готові до просування в пошукових системах. Також SPA може бути більш вразливими до деяких видів атак, таких як атаки на перехресне сайтове підтвердження (XSS), якщо не вжито відповідних заходів безпеки. Ці проблеми вирішує SSR, оскільки логіка додатка

виконується на стороні сервера, він може бути більш безпечним підходом, а також оскільки контент формується на стороні сервера, пошукові роботи легше аналізують сторінки, що може покращити видимість веб додатку в пошукових результатах. В нашому випадку ми використовуємо нашу систему локально, тому як пошукова оптимізація так і проблеми з безпекою обходять нас стороною.

## 1.4. Постановка задачі

Метою роботи є розробка програмного забезпечення для веб-магазину, що використовує метод колаборативної фільтрації для надання рекомендацій комплементарних товарів користувачам. Дане дослідження виконується в рамках кваліфікаційної роботи бакалавра за спеціальністю 122 «Комп'ютерні науки».

Розроблений веб-додаток повинен включати наступний функціонал:

- Можливість реєстрації за допомогою логіну та паролю.
- Авторизація в систему за допомогою облікових даних користувача.
- Додавання та видалення товарів з кошика.
- Можливість зміни кількості товарів в кошику.
- Можливість переглядати комплементарні товари до товарів у кошику.
- Можливість сортувати товари за ціною і назвою.
- Можливість фільтрувати товари за сортом і брендом.
- Можливість шукати товари по назві.
- Пагінація сторінок.
- Можливість додавати товар до кошика.

# 2. ВИБІР МЕТОДУ ВИРІШЕННЯ ЗАДАЧІ

## 2.1. Інформаційна модель додатку

Для написання веб додатку було використано трирівневу клієнт-серверну архітектуру. Вона поділяється на три рівні:

- Шар клієнта - є найвищим рівнем додатку з інтерфейсом користувача, головною функцією якого, є передавання задач і результатів зрозумілих користувачеві. В цьому шарі не повинна міститись бізнес-логіка та зберігатись критично важливі дані. Також він не повинен взаємодіяти з шаром бази даних безпосередньо, а лише через бізнес-логіку.
- Шар бізнес-логіки – цей шар координує програму, обробляє команди, виконує логічні рішення і обчислення, виконує розрахунки. Також він переміщується і обробляє данні між двома навколишніми шарами.
- Шар даних – на цьому рівні зберігається інформація і дістається з бази даних. Інформація надсилається у шар логіки для обробки і потім повертається до користувача.

Принцип роботи трирівневої архітектури взаємодії клієнт-сервер полягає в тому, що на відміну від дворівневої архітектури, де на серверному рівні обробка запиту відбувається на одному сервері без звернення до інших серверів у процесі цієї обробки,  ми розбили серверну частину на 2 окремі частини: шар бізнес-логіки та шар даних [11] (див. Рисунок 2.1Рисунок 2.1Рисунок 2.1Рисунок 2.1Рисунок 2.1Рисунок 2.1Рисунок 2.1Рисунок 2.1Рисунок 2.1Рисунок 2.1Рисунок 2.1Рисунок 2.1).

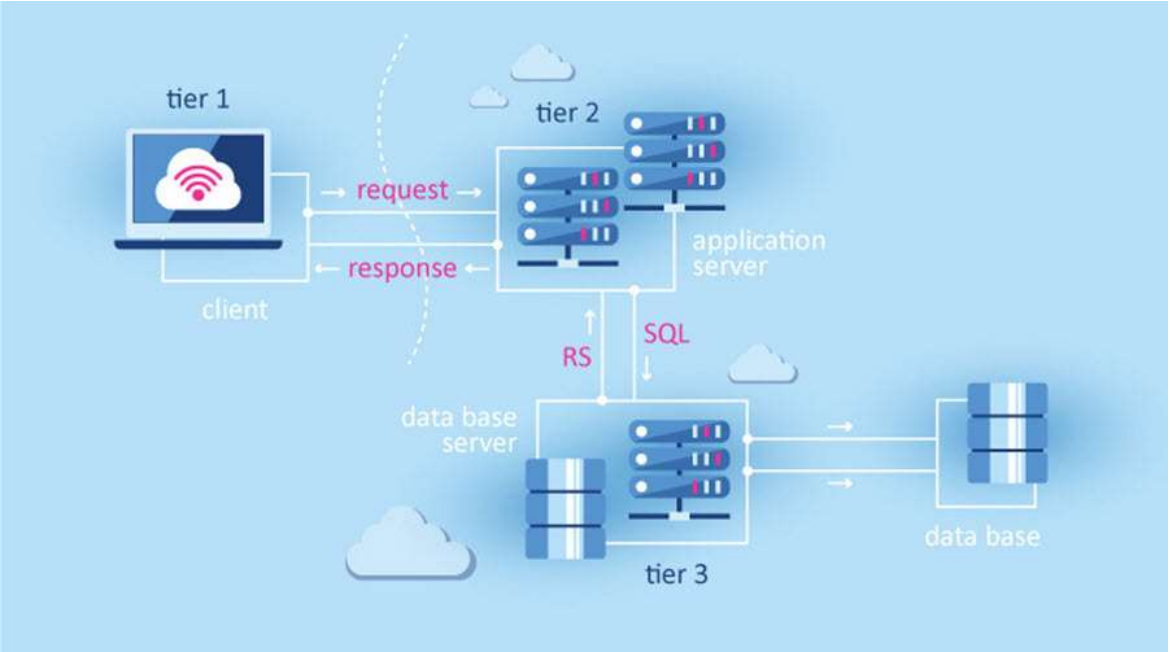


Рисунок 2.1 - Принцип роботи трирівневої архітектури [12].

Зроблено це було тому, що: по-перше, бази даних почали вимагати багато пам'яті та процесорного часу на обробку даних і їх почали виносити на окремі сервери, а по-друге, бази даних стали розумними, бо у них виникла власна бізнес-логіка.

Трирівнева архітектура була обрана для нашого веб-додатку, оскільки вона забезпечує чітке розділення логіки презентації, бізнес-логіки та доступу до даних. Це особливо важливо для демонстрації роботи колаборативної фільтрації для пошуку комплементарних товарів, оскільки, по-перше, дозволяє легше масштабувати кожен компонент окремо. Наприклад, якщо система рекомендацій потребуватиме більше обчислювальних ресурсів, можна масштабувати рівень бізнес-логіки, не зачіпаючи інші рівні. А по-друге, зміни в одному рівні не впливають безпосередньо на інші рівні. Це спрощує підтримку та оновлення системи. Наприклад, зміну алгоритму колаборативної фільтрації можна здійснити на рівні бізнес-логіки, не змінюючи при цьому рівень презентації чи рівень доступу до даних.

Далі наведемо DFD, на якій буде показано чим саме відрізняється взаємодія користувача, якому пропонуються комплементарні товари відносно тих товарів, які лежать у нього в кошику, на основі колаборативної фільтрації

та кошиків з придбаними товарами від взаємодії користувачів на інших веб магазинах, де рекомендації реалізовані по іншому принципу, або ж взагалі відсутні (див. Рисунок 2.2 ).

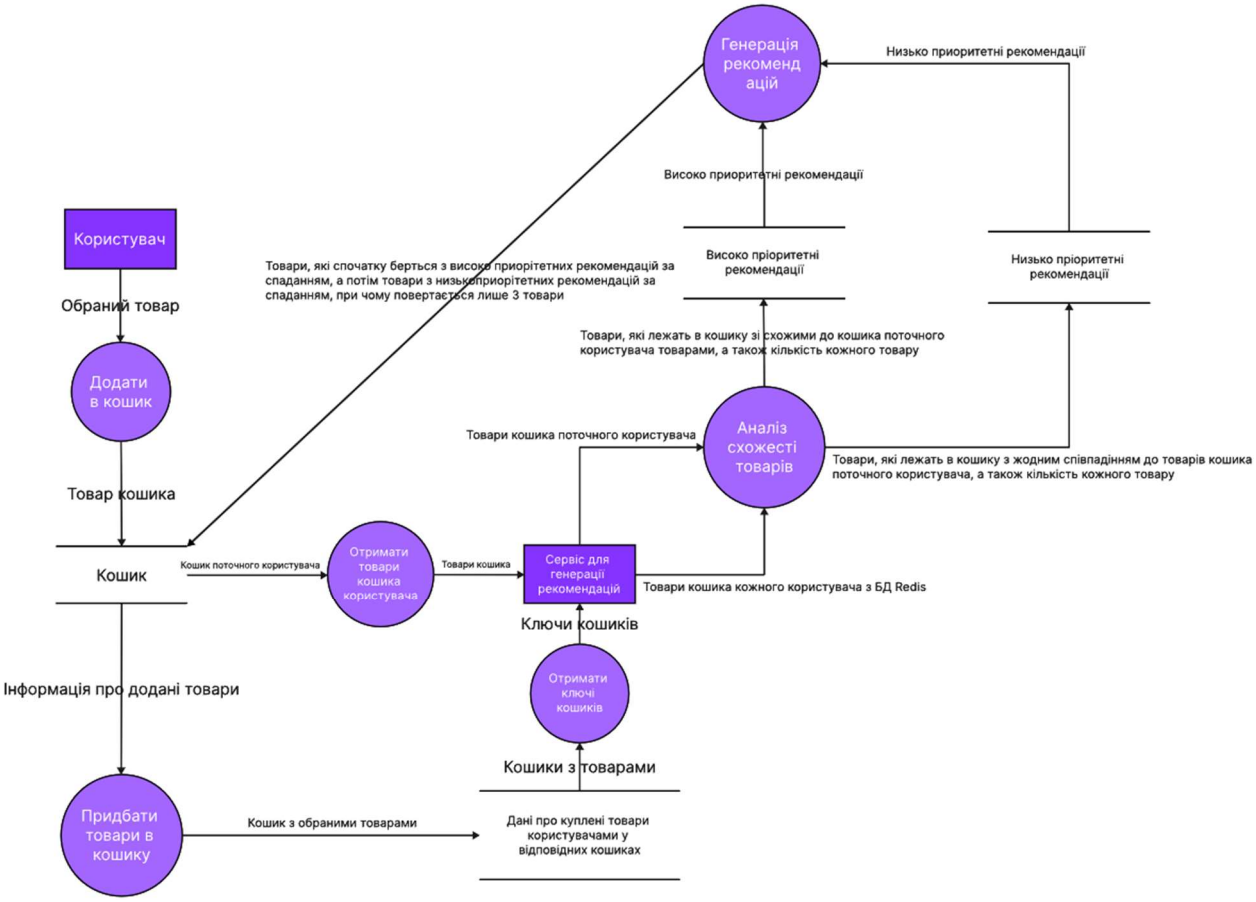


Рисунок 2.2 – DFD пошуку комплементарних товарів на основі колаборативної фільтрації.

Для написання серверної частини веб додатку було використано фреймворк ASP.NET Core[13]. ASP.NET Core - це відкрите програмне забезпечення, яке використовується для створення сучасних веб-додатків. Його основними перевагами є продуктивність, масштабованість, велика кількість розширень, а також підтримка мови програмування C#. Його аналогами є такі фреймворки як Ruby on Rails, написаний на Ruby, Django, написаний на Python, та Node.js, написаний на JavaScript. Всі ці фреймворки використовуються для написання веб додатків, але в нашому випадку було обрано саме ASP.NET Core тому, що по-перше, при написанні програм використовується мова програмування C#, яка має дуже гарні бібліотеки і

фреймворки, що допомагають при роботі з даними та базами даних, такі як LINQ і Entity Framework Core, і по-друге, C# має за замовчуванням функціонал асинхронності, що допомагає виконувати операції паралельно з іншими операціями, без блокування виконання основного коду програми. Саме ці властивості допоможуть нам написати веб додаток, який буде виконувати весь функціонал бізнес логіки, а також роботу з базою даних дуже швидко, при цьому не витрачаючи час на написання зайвого коду для підтримки функціоналу, якого немає в інших фреймворках, або ж які реалізують його гірше ніж ASP.NET Core.

Як СУБД для нашого проекту було обрано SQLite[14]. SQLite - це легка та вбудовувана реляційна база даних, яка використовується для збереження, опрацювання та управління даними. Через те, що вона не використовує парадигму клієнт-сервер, тобто рушій SQLite не є окремим процесом, з яким взаємодіє застосунок, і її можна легко вбудовувати, для нашого проекту вона підходить більше ніж будь-яка інша СУБД. Хоча і існують дуже гарні альтернативи, такі як MySQL та PostgreSQL, вони в основному використовуються для обробки великих обсягів інформації та запитів, а також їх треба окремо вбудовувати в програму, що в нашому випадку є зайвим. Також при використанні ASP.NET Core разом з Entity Framework Core, ми маємо доступ до функціоналу потрібного для роботи саме з SQLite.

Для написання клієнтської частини веб додатку було використано фреймворк Angular[15]. Angular – це фреймворк для розробки веб-додатків, оснований на TypeScript, який дозволяє створювати динамічні та інтерактивні користувацькі інтерфейси. Аналогами до Angular є такий популярний фреймворк як Vue.js та бібліотека React. Всі ці засоби виконують одну й ту саму функцію, але в кожного з них є свої переваги та недоліки. В порівнянні з React, Angular являє собою комплексний інструмент, який включає в себе багато засобів та функцій за замовчуванням, а також структура проекту складається з окремих логічних компонентів, з якими дуже легко взаємодіяти, налагоджувати маршрутизацію, тощо. React зі свого боку зазвичай потребує

вибору додаткових бібліотек і інструментів для забезпечення такої самої функціональності, а також хоч він і надає більше свободи в організації проекту, але це також може означати більше роботи для розробників у сенсі вибору правильних інструментів та архітектури. Vue.js в свою чергу славиться своєю швидкістю розробки та легкістю вивчення, чого не можна сказати про хоч і гарно задокументований, але все таки комплексний Angular.

Не зважаючи на те, що наш веб додаток є досить невеличким проектом, через те що Angular надає більше інструментів для розробки, яких в деяких випадках немає ні в Vue.js ні в React, а також запропоновану за замовченням архітектуру проекту, яку в інших випадках треба вигадувати самому, кращим варіантом для написання клієнтської частини буде використання Angular.

## 2.2. Структура бази даних

Перед початком розробки слід визначити які дані ми будемо зберігати. Так як на нашому веб додатку користувачі можуть переглядати вина певних типів та брендів, то в нашій БД ми повинні зберігати дані про користувачів, типи та бренди вин, самі вина, а також кошики користувачів, що вже купили певні товари для подальшого їх використання для пошуку комплементарних товарів на основі колаборативної фільтрації. Для цього створимо дві окремі бази даних для вин та їх брендів і сортів, а також для користувачів та їх адрес. Це зроблено тому, що спосіб, у який ми дозволимо користувачам авторизуватися на сайті – JWT[16], що означає, що спочатку їм має бути виданий токен і тільки потім вони зможуть його використовувати для авторизації. Так як ми використовуємо для серверної частини СУБД SQLite разом з фреймворком ASP.NET Core, ми будемо зберігати всі потрібні нам дані у форматі json, і за допомогою Entity Framework Core, зможемо створювати міграції наших таблиць з усіма потрібними атрибутами, їх ключами, обмеженнями та типами. Тому, щоб додати функціонал для видачі токенів авторизації та більш легкого створення таблиць користувачів треба використовувати готову систему ідентифікації ASP.NET Identity, для роботи з

якою буде краще створити окрему БД і відповідно класи, які допоможуть діставати звідти дані. Для демонстрації ієрархії таблиць в базі даних товарів, накреслимо відповідну ER-діаграму (див. Рисунок 2.3).

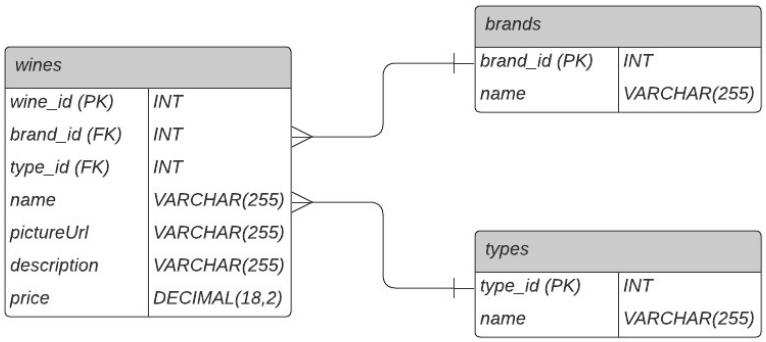


Рисунок 2.3 – ERD для БД вин.

Для демонстрації ієрархії таблиць в базі даних користувачів, накреслимо відповідну ER-діаграму (див. Рисунок 2.4 ).

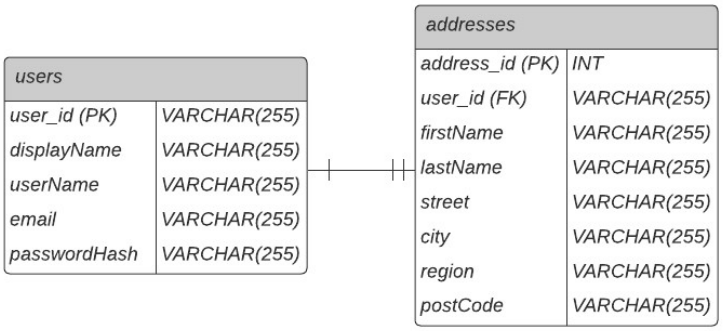


Рисунок 2.4 – ERD для БД користувачів.

Залишається лише місце, де ми будемо зберігати кошики користувачів, які містять товари, що вже були куплені. Кошик це здебільшого client side state, і ми повинні вибрати відповідне місце для зберігання вмісту кошика. Для цього більше всього нам підійде Redis (Remote Dictionary Service) [17], що являє собою зберігання даних в пам'яті на нашому сервері. Його основне призначення це кешинг. Він дуже швидкий, є постійним місцем зберігання, і працює як структура даних типу ключ-значення. Оскільки Redis - це окрема база даних, нам потрібно було створити окремий репозиторій і контролер для

роботи з нею, оскільки за допомогою Entity Framework до неї немає доступу, ми робимо це напряму. Ми дали ім'я кожному кошику у вигляді рядка і серіалізували наш кошик як JSON-структуру і представили її у вигляді рядка для зберігання в Redis. І хоч здебільшого основні зміни в кошику трапляються на клієнтському боці, дані йтимуть на наш сервер і ми зможемо в разі чого додавати відповідний функціонал. Для демонстрації ієрархії таблиць в базі даних кошиків користувачів, накреслимо відповідну ER-діаграму (див. Рисунок 2.5).

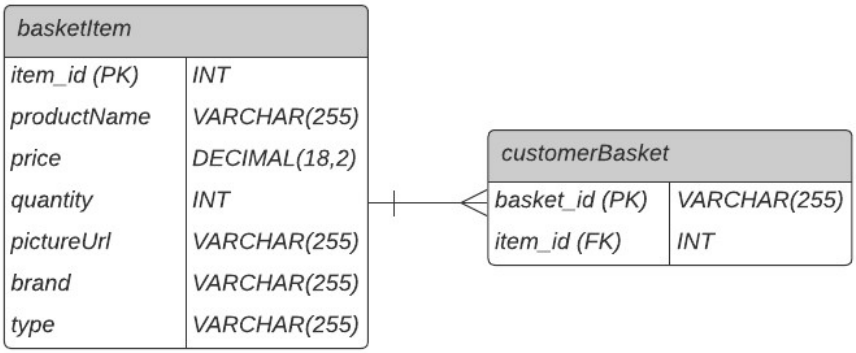


Рисунок 2.5 – ERD для бази даних кошиків користувачів.

## 2.3. Архітектура серверної частини проекту

Архітектура серверної частини проекту складається з 3-ох проектів, а саме API, Infrastructure та Core (див. Рисунок 2.6 ). Кожен з них відповідає за певні задачі всередині глобального проекту Diploma webstore, що являє собою папку нашого solution.

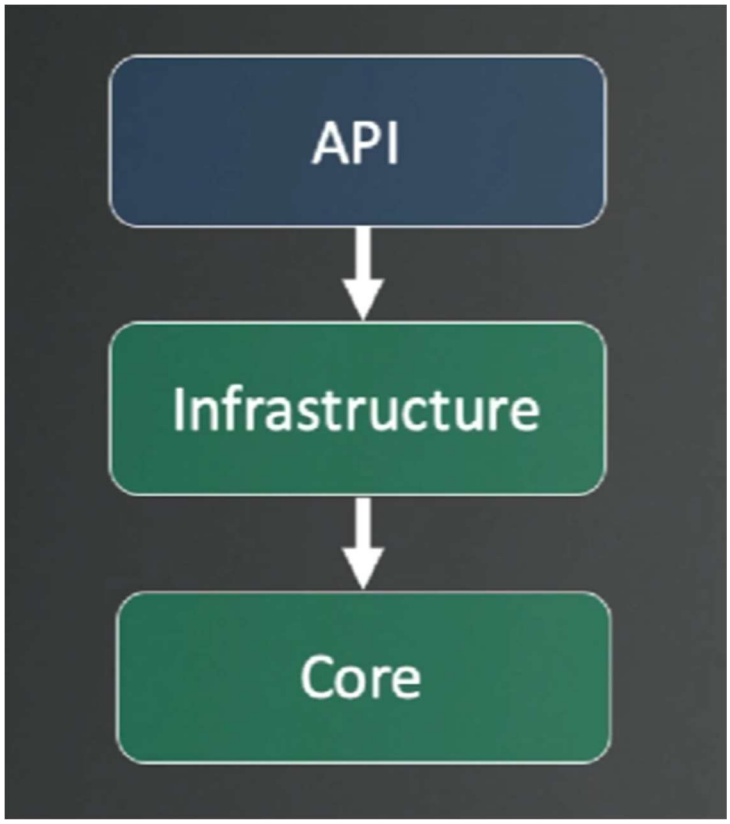


Рисунок 2.6 – Архітектура серверної частини проекту.

1. API являє собою окремий проект зі стартовим додатком, базами даних і допоміжними елементами, які додають конфігурації і корисні пакети в проект, відстежують та обробляють помилки. В ньому також прописується логіка, що відповідає за перенаправлення будь яких запитів, які надходять до контролерів.
2. Infrastructure являє собою окремий проект, який містить репозиторії, сервіси та DbContext. Також він відповідає за відправку запитів до бази даних.
3. Core являє собою окремий проект, який містить сутності, інтерфейси та специфікації.

## 2.4. Архітектура клієнтської частини проекту

Архітектура клієнтської частини проекту являє собою структуру файлів і папок рекомендовану для написання Angular проекту [1818]. Залежно від того, як ми використовуємо модулі в нашому проекті, ми можемо класифікувати наші модулі на наступні чотири категорії: кореневий (app),

функціональний (feature), спільний (shared) та основний (core) (див. Рисунок 2.7).

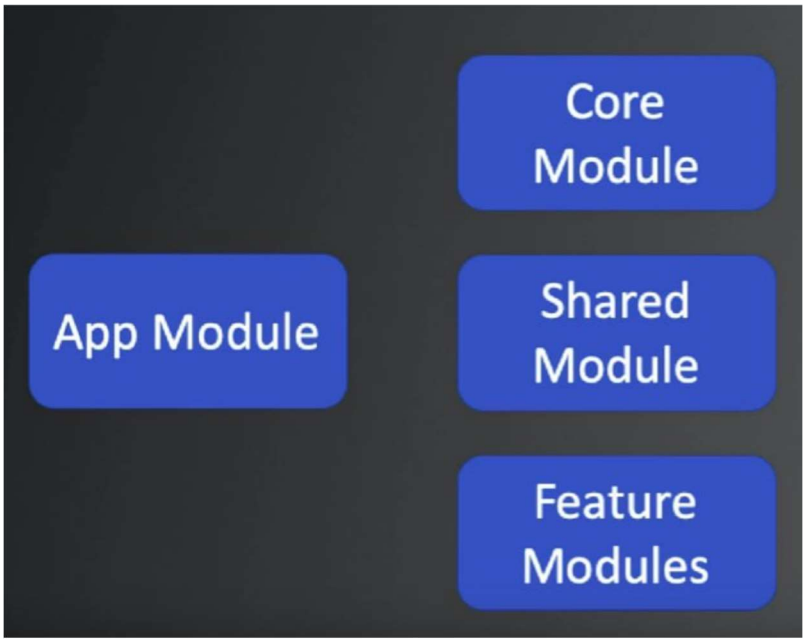


Рисунок 2.7 - Архітектура клієнтської частини проекту.

1. Angular вимагає, щоб під час запуску програми завантажувався один модуль. Ми називаємо його кореневим модулем або AppModule. Кореневий модуль завантажує кореневий компонент і всі інші модулі.
2. Сервіси, які спільно використовуються в додатку, стають частиною основного модулю або CoreModule. Зазвичай служби мають бути Singleton[19], тобто існувати має лише один екземпляр служби. Надання його в CoreModule гарантує, що сервіси залишатимуться Singleton. Основний модуль повинен бути імпортований тільки в кореневому модулі.
3. Існує багато компонентів, моделей і директив, якими ми можемо поділитися з різними модулями. Усі ці компоненти слід розмістити у спільному модулі або SharedModule. Спільний модуль оголошує компоненти, моделі і директиви за допомогою метаданих декларацій і експортує їх за допомогою метаданих експорту.

4. Функціональний модуль або FeatureModule реалізує певну функцію програми. Всі компоненти та директиви, які реалізують цю функцію, стають частиною модуля.

# 3. ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

## 3.1. Опис файлової системи серверної частини проекту

В цьому розділі розглянемо папки і файли основних трьох проектів серверної частини проекту, а саме API, Infrastructure та Core. Почнемо з основних елементів проекту API (див. Рисунок 3.1).

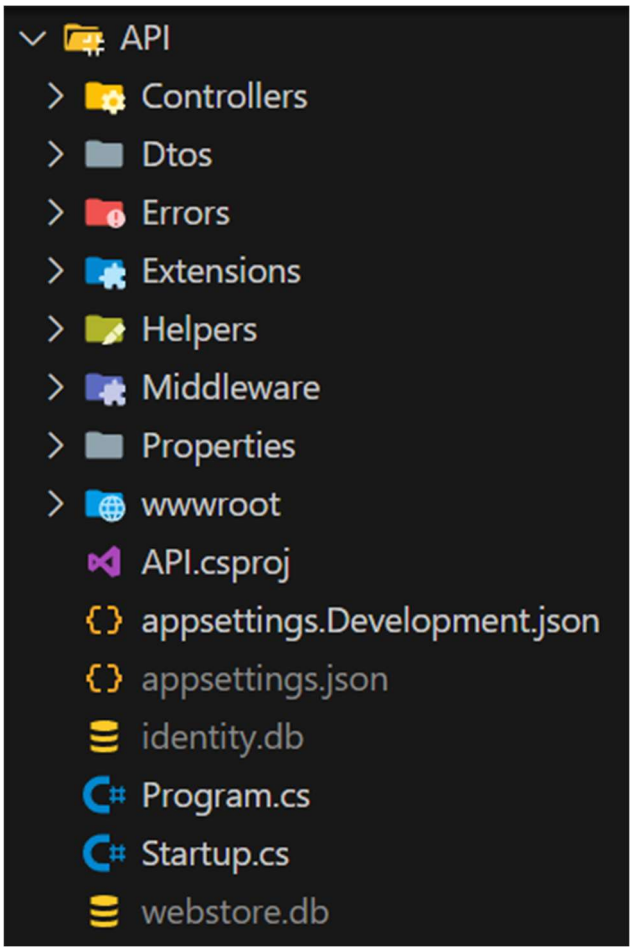


Рисунок 3.1 – Елементи API проекту.

API включає в себе:

- Controllers – папка, що містить контролери, до яких робляться запити з клієнтської частини додатку до серверної. Вони виконують дану їм задачу на серверній частині додатку, наприклад отримають дані про продукти і згруповують їх у потрібний нам формат, та повертають певні дані назад на клієнтську частину.

- DTO (Data Transfer Object) [20] - об’єкт, що використовується для передачі даних з одного місця програми в інше. Ми їх використовуємо для передачі даних з серверного рівня додатку на клієнтський рівень.
- Errors - папка, в якій знаходяться класи, що дають нам змогу повертати не відгук на помилку за замовчуванням, яка створюється методом у контролері (наприклад NotFound()), а наш власний відгук (наприклад NotFound(new ApiResponse(404))). Це було зроблено для того, щоб виводити стабільний відгук про помилки від нашого Api.
- Extensions – папка, в яку винесено написані нами сервіси і конфігурації для того, щоб спростити читабельність класу Startup, а також винести написані нами сервіси і конфігурації.
- Helpers – папка, що містить допоміжні класи для написання програми, наприклад клас для пагінації.
- Middleware - проміжне програмне забезпечення, яке в нашому випадку обробляє помилку типу 500 і повертає її у потрібному нам вигляді.
- wwwroot – папка, в якій зберігаються зображення для наших товарів.
- API.csproj – файл, який містить пакети і залежності потрібні для правильного функціонування API проекту. До важливих залежностей відноситься Infrastructure.csproj, який ми включили для того, щоб наприклад підключити сервіси до нашого основного проекту.

Далі розглянемо основні елементи проекту Infrastructure (див. Рисунок 3.2 ).

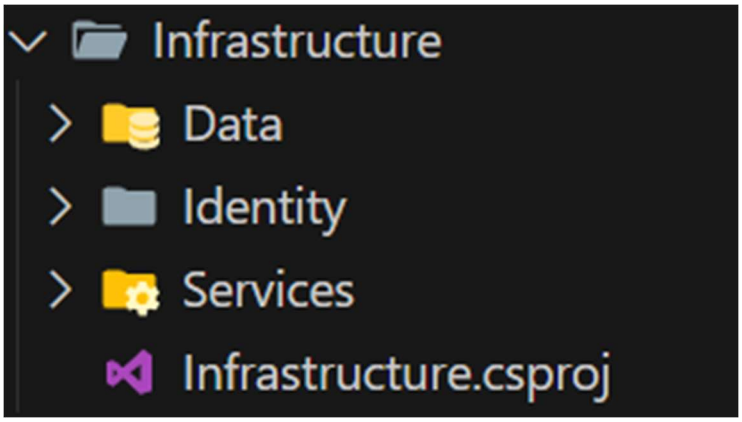


Рисунок 3.2 - Елементи Infrastructure проекту.

Infrastructure включає в себе:

- Data – папка, що містить папки Config та SeedData, а також папку Migration, що містить міграції для StoreContext. Також вона містить репозиторії та сам StoreContext.
- Config – папка, що містить конфігурацію залежностей і полів для сутності товару, які використовуються для створення коректної таблиці товарів в базі даних.
- SeedData – папка, що містить початкові дані для заповнення основних таблиць бази даних.
- Identity – папка, що містить міграції для AppIdentityDbContext, такі як таблиця користувачів та адрес в папці Migration. Також вона містить сам AppIdentityDbContext та дані про користувача для заповнення в базі даних таблиці користувачів.
- Services – папка, що містить різноманітні сервіси, які ми створили власноруч.
- Infrastructure.csproj – файл, який містить пакети і залежності потрібні для правильного функціонування Infrastructure проекту. До важливих залежностей відноситься Core.csproj, який ми включили для того, щоб наприклад мати доступ до сутностей в Infrastructure та Api проектах.

Під кінець розглянемо основні елементи проекту Core (див. Рисунок 3.3).

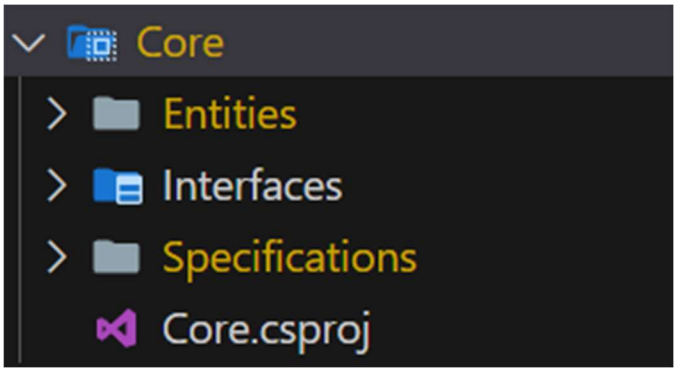


Рисунок 3.3 - Елементи Core проекту.

Core включає в себе:

- Entities – папка, в якій було створено різноманітні сутності, такі як продукт, кошик, користувач, адреса тощо.
- Interfaces – папка, в якій було створено інтерфейси для репозиторіїв і сервісів.
- Specifications – папка, що містить специфікації для товарів, які допомагають нам отримувати дані про товари з бази даних в потрібному нам форматі.
- Core.csproj - файл, який містить пакети і залежності потрібні для правильного функціонування Core проекту.

## 3.2. Опис файлової системи клієнтської частини проекту

В цьому розділі розглянемо папки і файли чотирьох основних модулів, а саме кореневого (app), функціонального (feature), спільного (shared) та основного (core). Розпочнемо з кореневого модуля, який знаходиться в папці app разом з іншими папками модулів, а також з компонентами AppModule (див. Рисунок 3.4 ).

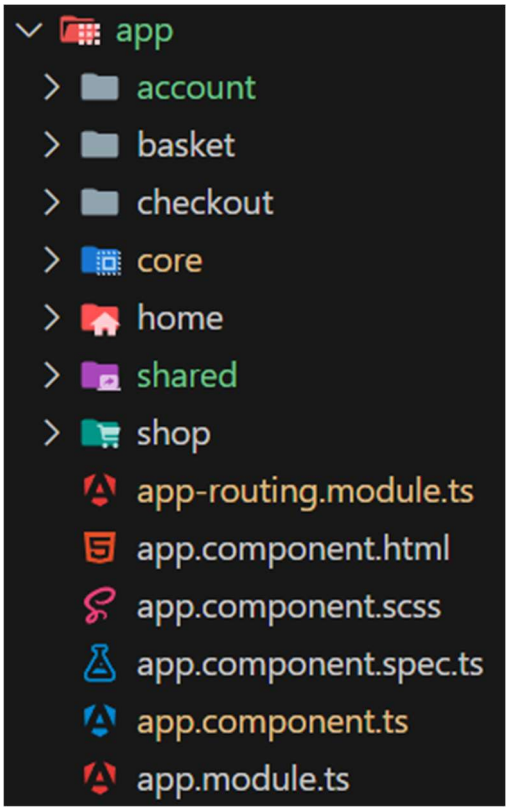


Рисунок 3.4 – Елементи папки app.

Основний же модуль знаходиться в папці core разом з компонентом навігаційної панелі, компонентами помилок та їх тестування, компонентом заголовку розділу та перехоплювачами (див. Рисунок 3.5).

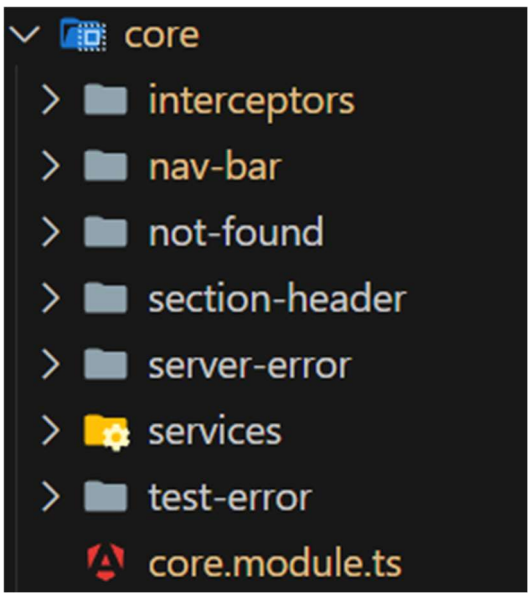


Рисунок 3.5 - Елементи папки core.

Далі йде спільний модуль, який знаходиться в папці shared разом з моделями, наприклад товар, бренд та кошик, а також такими компонентами

як: меню пагінації, заголовок пагінації та текстове введення (див. Рисунок 3.6 ).

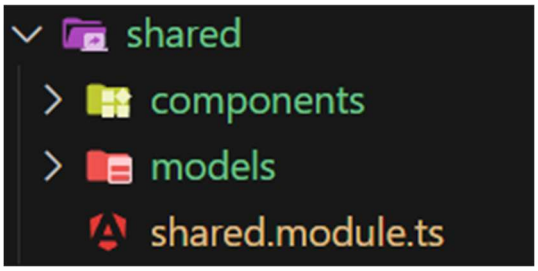


Рисунок 3.6 - Елементи папки shared.

Також маємо наступні функціональні модулі:

- ShopModule – модуль, що знаходиться в папці shop і містить компонент для основної сторінки магазину, а також додаткові компоненти для окремого товару на сторінці магазину, а також для власної сторінки кожного товару (див. Рисунок 3.7).

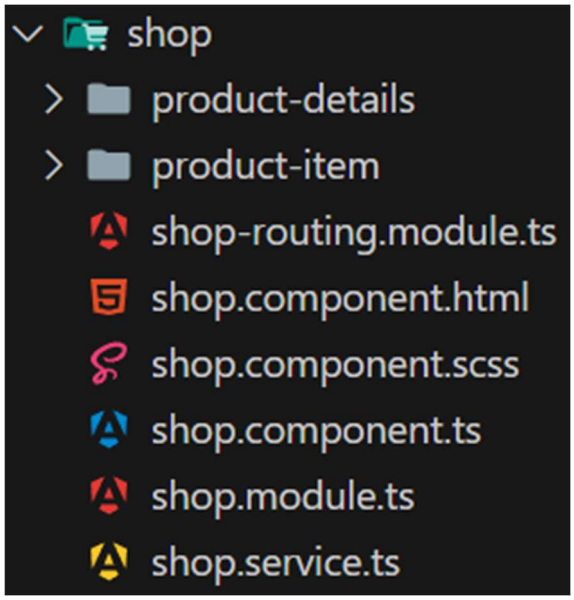


Рисунок 3.7 - Елементи папки shop.

- HomeModule – модуль, що знаходиться в папці home і містить компонент для домашньої сторінки магазину (див. Рисунок 3.8 ).

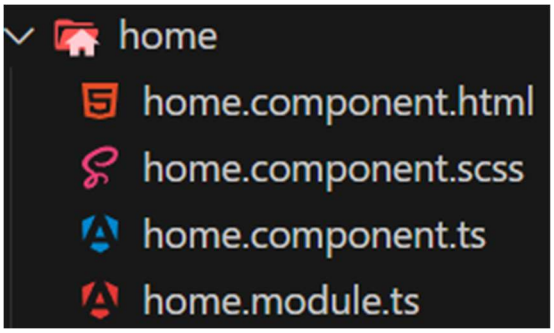


Рисунок 3.8 - Елементи папки home.

- BasketModule – модуль, що знаходиться в папці basket і містить компонент для сторінки кошика магазину (див. Рисунок 3.9 ).

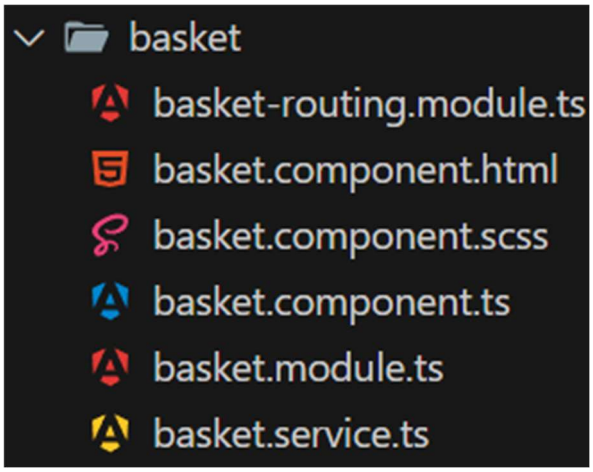


Рисунок 3.9 - Елементи папки basket.

- AccountModule - модуль, що знаходиться в папці account і містить компонентb для сторінки логіну і реєстрації (див. Рисунок 3.10 ).

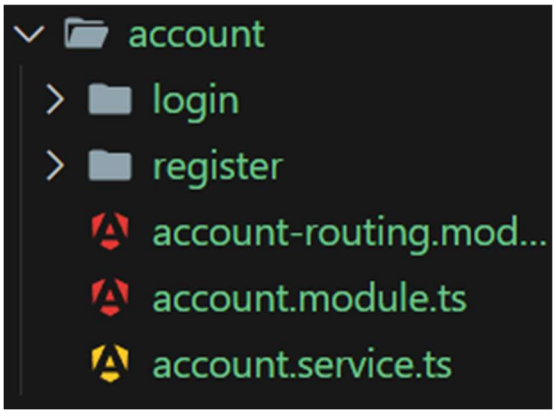


Рисунок 3.10 - Елементи папки account.

### 3.3. Формування вхідних даних

Для того щоб провести пошук комплементарних товарів на основі колаборативної фільтрації було заповнено базу даних товарами з інформацією про вина їх бренди та типи, а також заповнено базу даних кошиків користувачів. Спочатку було написано стартові дані для баз даних вин, брендів та типів (див. Рисунок 3.11).

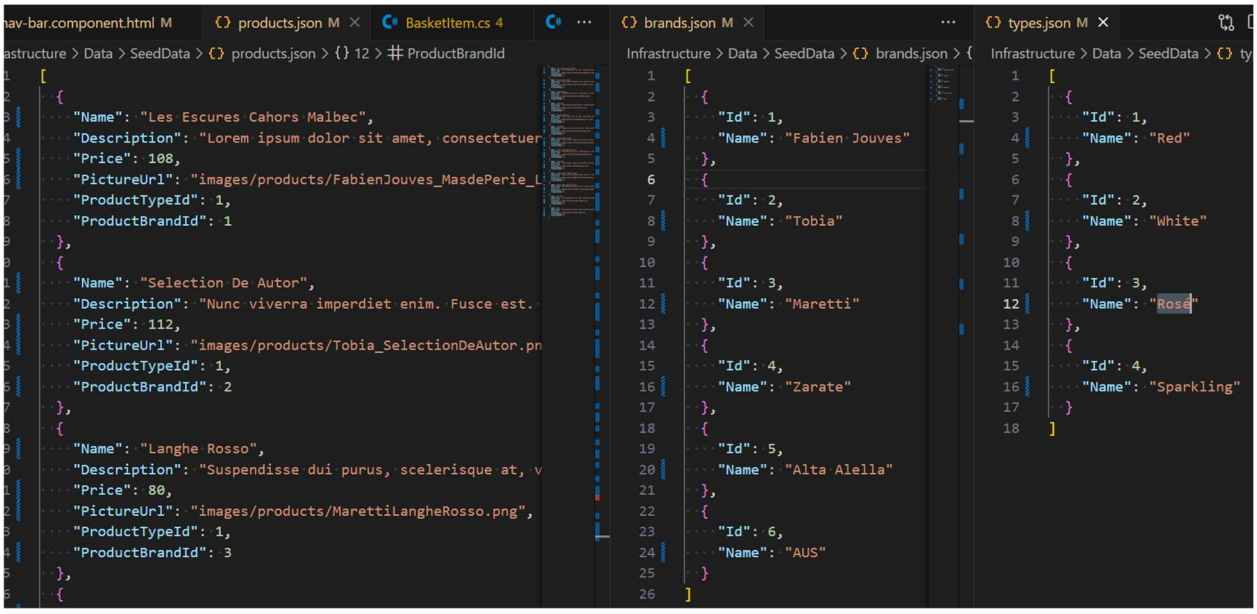


Рисунок 3.11 – Стартові дані для таблиць вин, типів та брендів.

У нашому випадку всі допоміжні додатки на кшталт Redis можуть запускатися окремо від нашого основного додатка, і у випадку з Redis, його неможливо завантажити без зайвих проблем на операційній системі Windows. Тому, використовуючи Docker[21], ми змогли легко встановити його на нашу машину для розробки та використати його з метою зберігання, а також перегляду вмісту Redis. Для цього спочатку потрібно було встановити Docker Desktop і надалі перед тим як запускати нашу програму, потрібно запускати Docker Desktop. Для налаштування Redis, у файлі docker-compose.yml ми прописали сервіси, які нам потрібні, а саме Redis і Redis-commander (див. Рисунок 3.12 ). Redis-commander дає нам змогу переглядати дані, які зберігаються в Redis, на окремому порті.

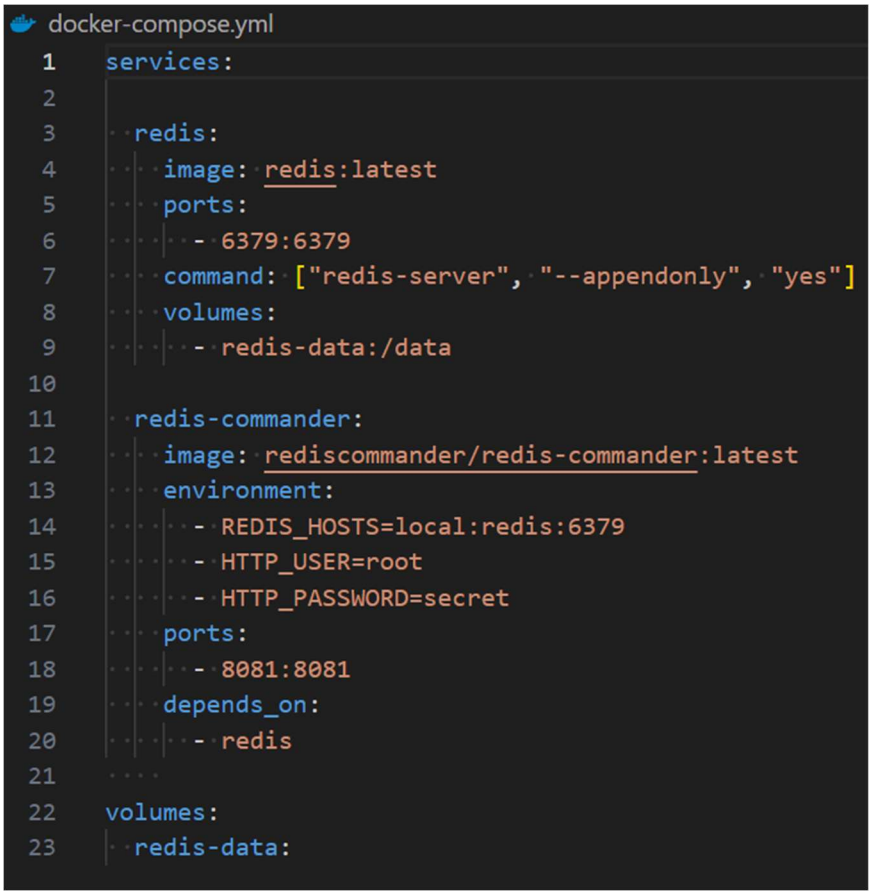


Рисунок 3.12 – Налаштування в файлі docker-compose.yml .

Далі було заповнено базу даних Redis даними про товари з кошиків заздалегідь створених для подальшого тестування колаборативної фільтрації, які починаються зі слова basket, за допомогою Post запитів до BasketController з використанням програми Postman[22]. Кошик з набором символів є кошиком, який створився про додаванні першого товару авторизованим користувачем на сайті. Для перегляду даних занесених в Redis, ми можемо перейти на сайт Redis-commander за вказаним нами портом і ввести пароль та ім'я з файлу docker-compose.yml (див. Рисунок 3.13 ).

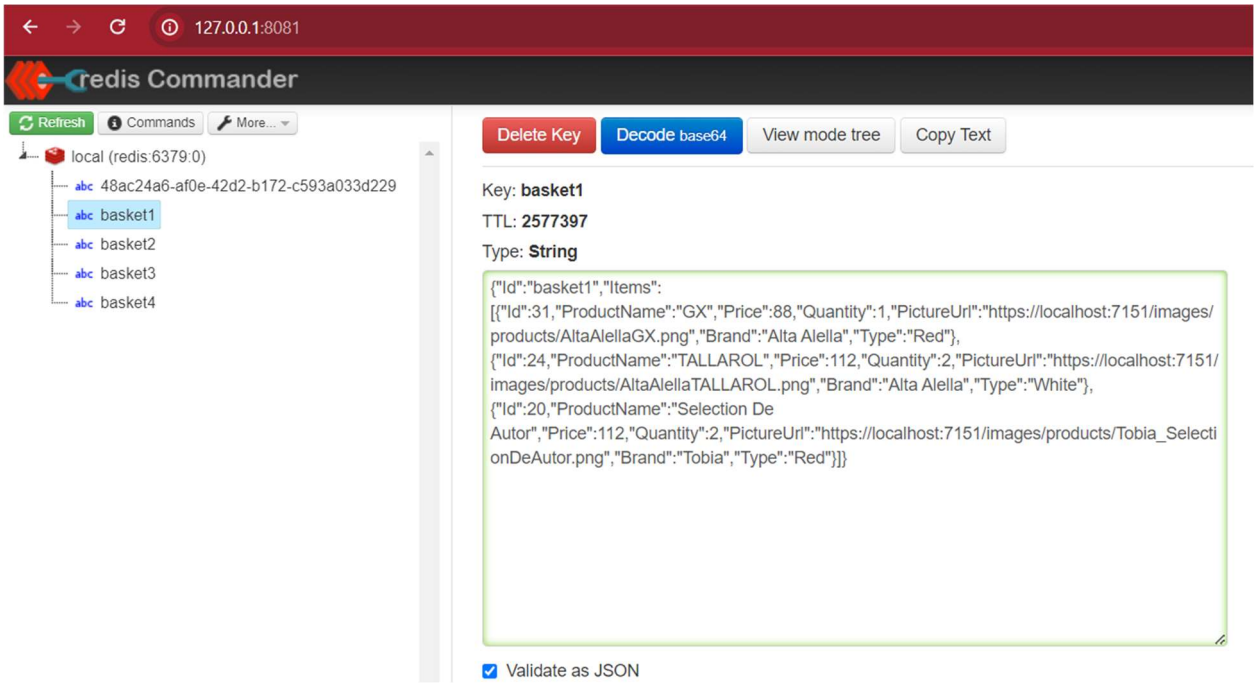


Рисунок 3.13 – Інформація про кошики користувачів на сайті Redis-commander.

## 3.4. Опис програмної реалізації

Для того щоб знайти комплементарні товари на основі колаборативної фільтрації, на серверній частині в класі BasketRepository, що відповідає за отримання даних з кошику користувача, їх оновлення та видалення кошиків, було додано два методи, а саме FindComplementaryItemAsync(string basketId) та GetAllBasketKeysAsync() (див. Додаток).

Спочатку отримуємо дані кошика користувача з бази даних Redis за basketId, десеріалізуємо JSON-дані в об'єкт CustomerBasket та зберігаємо товари кошика в userItems. Далі Якщо кошик користувача порожній, повертаємо порожній список. За допомогою методу GetAllBasketKeysAsync() отримуємо ключі кошиків користувачів, що вже придбали певні товари, з Redis та створюємо словники для високо та низько пріоритетних рекомендацій. Далі в циклі перебираються всі ключі кошиків та отримуються дані кошиків. Якщо хоч якісь товари співпадають кошику співпадають з кошиком користувача, то ті товари з цього кошику, що не повторюються, потрапляють в високо пріоритетні рекомендації, інакше, якщо співпадінь з кошиком користувача немає, ті товари, що не повторюються, потрапляють в

низко пріоритетні рекомендації. Якщо товар вже в рекомендаціях, збільшує його кількість. Потім створюється HashSet для імен високопріоритетних товарів. Фільтруються низько пріоритетні рекомендації, щоб уникнути дублювання рекомендацій з високо пріоритетними. Під кінець високо пріоритетні рекомендації об'єднуються з відфільтрованими низько пріоритетними, сортуються за кількістю у порядку спадання та беруться 3 найкращі рекомендації, після чого повертаємо їх як кінцеві рекомендації для користувача.

В контролері BasketController, в якому знаходяться кінцеві пункти до методів отримання, оновлення та видалення даних з кошику, прописуємо кінцевий пункт типу HttpGet і ключовим словом «recomendation», при запиту до якого, буде визиватись метод FindComplementaryItemAsync() в який передаємо id кошику користувача:

```
[HttpGet("recomendation")]
public async Task<ActionResult<List<BasketItem>>>
GetRecomendationsByBasketId(string id)
{
    var recomendations = await
_basketRepository.FindComplementaryItemAsync(id);

    return Ok(recomendations);
}
```

На клієнтській частині проекту в сервісі BasketService, який потрібен для отримання певних даних про кошик користувача за допомогою запитів до кінцевих пунктів класу BasketController, прописуємо запит до кінцевого пункту GetRecomendationsByBasketId(), щоб отримати рекомендаційні товари:

```
getRecomendations(basket: IBasket) {
    return this.http.get<IBasketItem[]>(this.baseUrl +
'basket/recomendation?id=' + basket.id)
}
```

В компоненті BasketComponent, де прописуються методи, які використовуються при натиску певних кнопок на html сторінці кошику або інших змін на сторінці, прописуємо при ініціалізації компонента встановлення this.basket$ як Observable, який стежить за станом кошика та встановлення this.recomendations$ як Observable, який залежить від this.basket$. Коли this.basket$ змінюється, то switchMap() отримує нове значення кошика і якщо

кошик існує, викликається getRecomendations() для отримання рекомендацій на основі поточного кошика. Якщо ж кошик не існує, повертається порожній масив:

```
ngOnInit() {
    this.basket$ = this.basketService.basket$;
    this.recomendations$ = this.basket$.pipe(
        switchMap(basket => {
            if (basket) {
                return this.basketService.getRecomendations(basket);
            }
            return of([]);
        })
    )as Observable<IBasketItem[]>;
}
```

Останнє про що треба сказати, це шаблон basket.component.html компоненту BasketComponent, в якому прописується код сторінки кошику. Виводимо рекомендації перед кошиком користувача наступним чином:

```
<div *ngIf="recomendations$ | async as recomendations">
    <div class="row mt-5">
        <div class="col-12">
            <h3>Recommended Products</h3>
            <ul class="list-group">
                <li class="list-group-item d-flex justify-content-between
align-items-center" *ngFor="let item of recomendations">
                    <div class="d-flex align-items-center">
                        <img src="{{ item.pictureUrl }}" alt="{{
item.productName }}" class="img-fluid" style="max-height: 50px;">
                        <div class="ml-3 d-inline-block align-middle">
                            <h5 class="mb-0">
                                <a routerLink="/shop/{{item.id}}"
                                class="text-dark">
                                    {{item.productName}}
                                </a>
                            </h5>
                            <span class="text-muted font-weight-normal
                            font-italic d-block">
                                Type: {{item.type}}
                            </span>
                        </div>
                    </div>
                    <div class="d-flex align-items-center">
                        <strong class="mr-3">{{ item.price | currency
}}</strong>
                        <button class="btn btn-outline-primary btn-lg">Add
to Cart</button>
                    </div>
                </li>
            </ul>
        </div>
    </div>
</div>
```

## 3.5. Аналіз результатів

Користувач має можливість реєструватись та авторизуватись на сайті. Різниця між авторизованим і не авторизованим користувачем полягає в тому, що не зареєстрований користувач не зможе придбати товари, а авторизований зможе. В іншому весь функціонал сайту буде доступний обом типам користувачів.

Для реєстрації на сайт користувач повинен натиснути кнопку sign up розташовану справа зверху та заповнити всі необхідні поля (див. Рисунок 3.14 ), і якщо пошта введена коректно та не зайнята, а також введений правильний формат паролю, користувача перекине на основну сторінку магазину, а справа зверху він зможе побачити своє ім’я, яке він прописав при реєстрації.

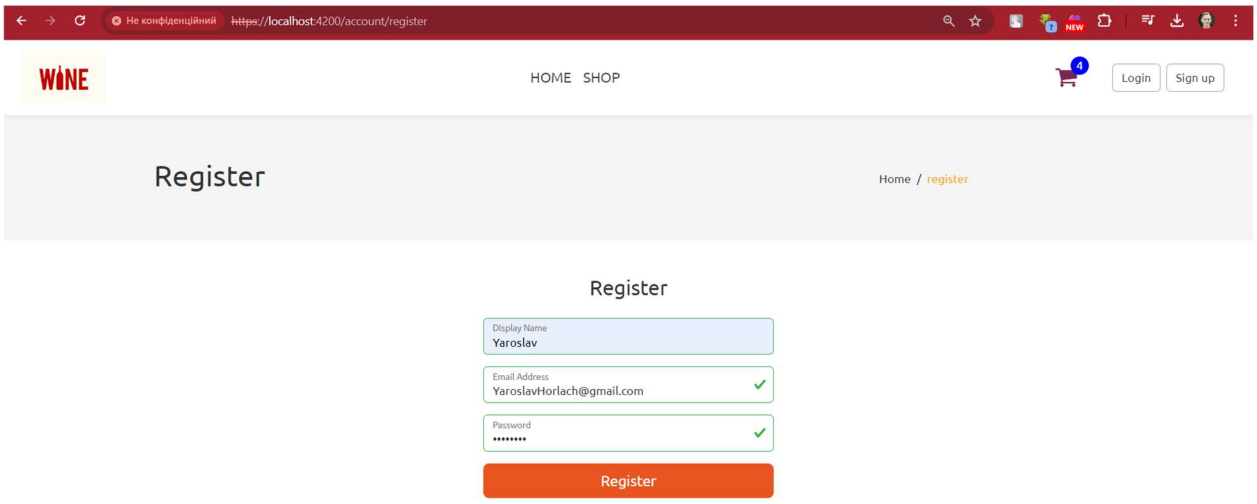


Рисунок 3.14 – Сторінка реєстрації.

Що стосується авторизації, то якщо користувач був попередньо зареєстрований, то він зможе після виходу зі свого акаунту натиснути на кнопку login розташовану справа зверху, заповнити поля з поштою та паролем і натиснути на кнопку авторизуватись (див. Рисунок 3.15 Рисунок 3.14 ), після чого його так само перекине на основну сторінку магазину.

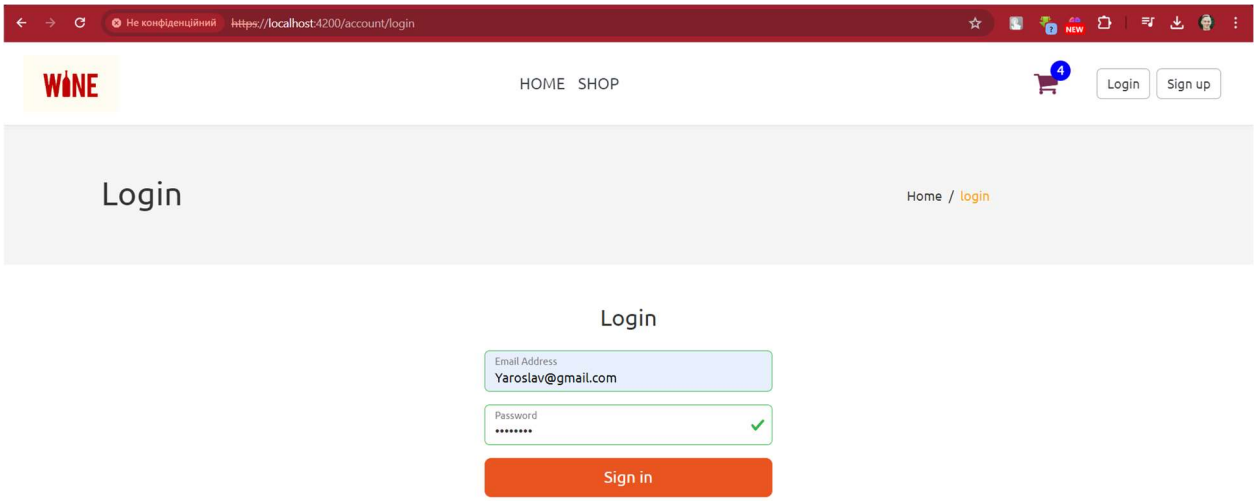


Рисунок 3.15 – Сторінка авторизації.

На основній сторінці магазину (див. Рисунок 3.16 Рисунок 3.15 ), на яку можна перейти натиснувши посередині зверху на пункт Shop, користувач може фільтрувати товари за брендом та типом, сортувати товари по алфавіту та ціною, а також шукати за певною назвою потрібний товар. Крім цього, якщо товарів більше шести, то на сторінці знизу можна буде побачити меню пагінації по сторінкам, для того щоб можна було продивитись всі товари. Крім перегляду товарів, якщо навести на один з них мишкою то з’явиться можливість або одразу додати товар до кошику, або ж перейти на сторінку товару.

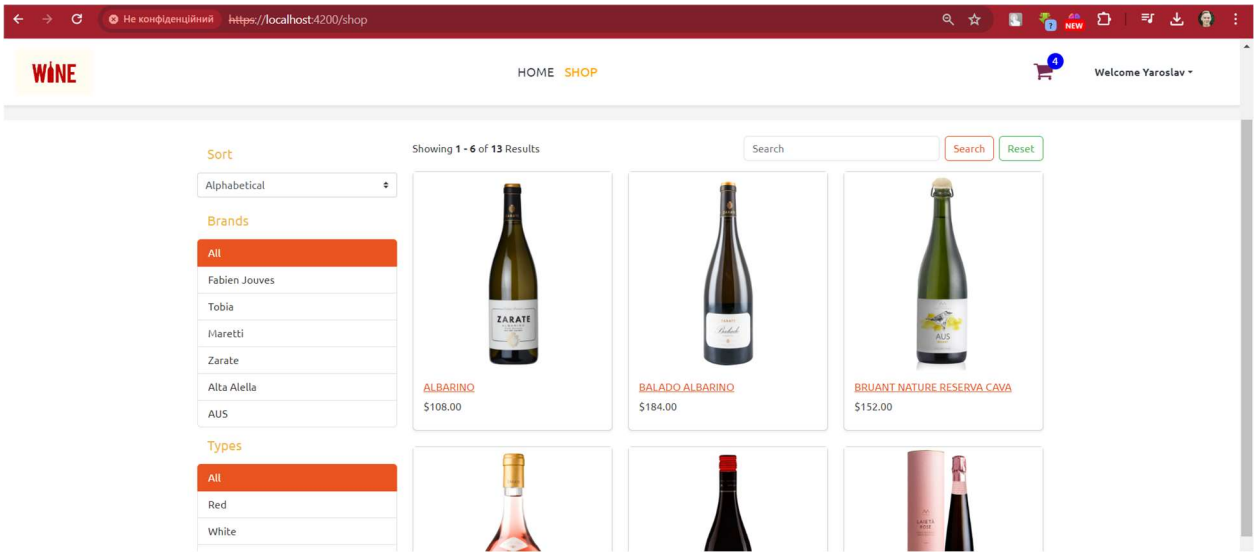


Рисунок 3.16 – Основна сторінка магазину.

На сторінці товару можна побачити зображення та назву товару, його ціну, вибрати скільки одиниць товару користувач хоче додати до кошику, а також опис певного товару (див. Рисунок 3.17).

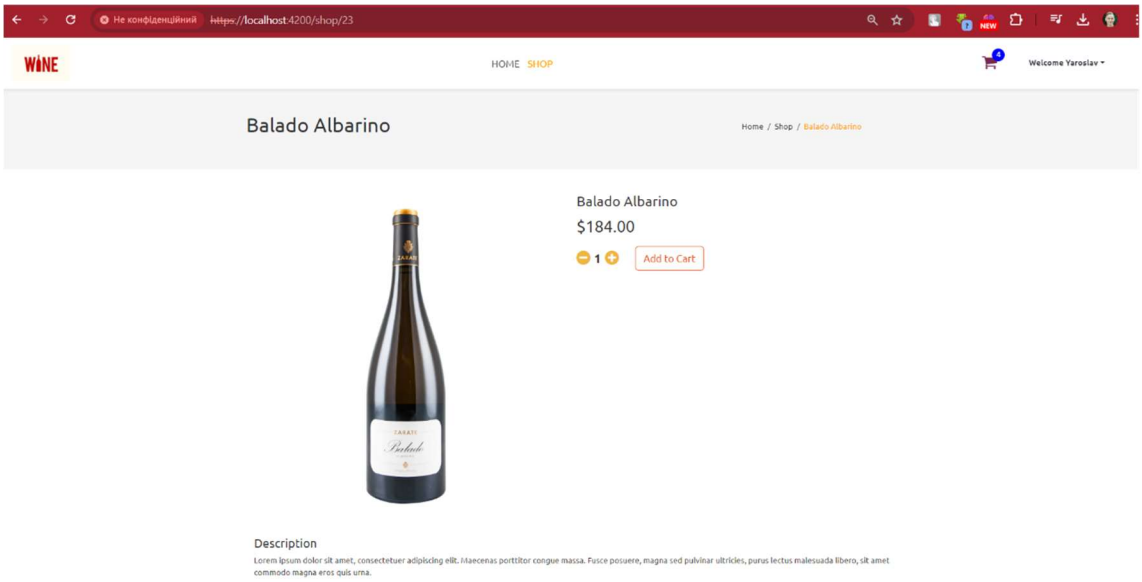


Рисунок 3.17 – Сторінка товару.

Остання важлива сторінка магазину це кошик. Його можна знайти справа зверху і він виглядає як кошик з числом товарів у ньому. Натиснувши на нього, можна побачити ті товари, які були додані до кошику, видалити їх з нього, або зменшити та збільшити їх кількість. І що найголовніше, саме тут відображаються комплементарні товари знайдені за допомогою колаборативної фільтрації (див. Рисунок 3.18 ). Як ми бачимо відображаються рекомендації відносно тих даних, які були занесені в Redis. Тут же можна додати одразу товар в кошик, або перейти на сторінку товару і додати його в кошик там.

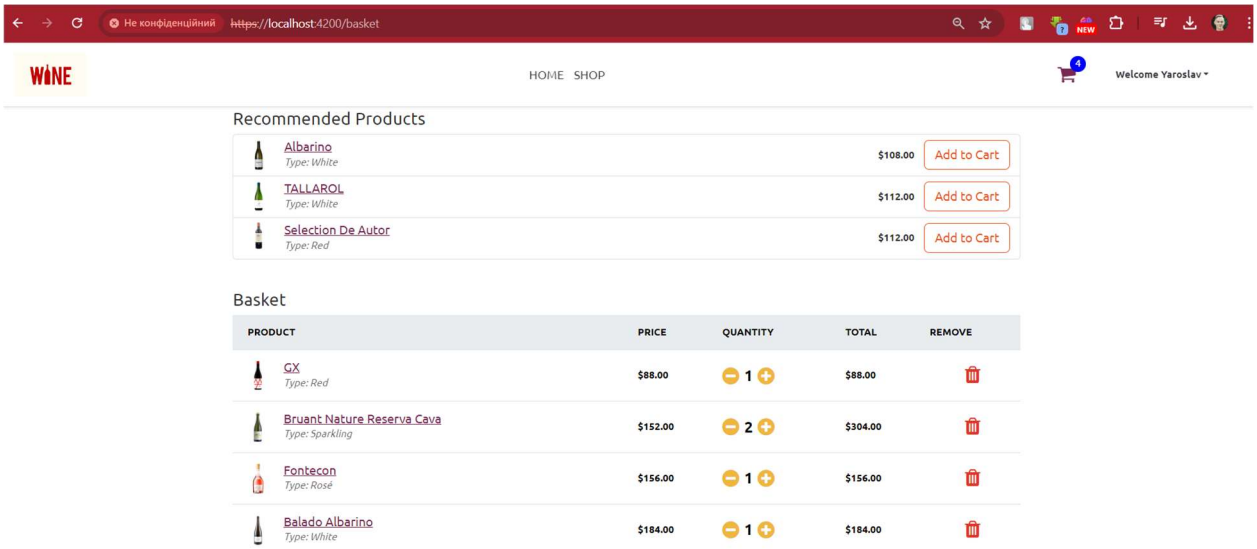


Рисунок 3.18 – Сторінка кошику з підбором комплементарних товарів.

Якщо додати наприклад вино «Albarino» в кошик, то зможемо побачити інший набор рекомендацій при якому спочатку будуть відображатись товари, які є в кошиках разом з товарами поточного користувача і які відсортовані по кількості куплених одиниць. Коли ж всі товари, які були разом з товарами поточного користувача закінчяться, то будуть відображатись відсортовані по кількості куплених одиниць товари в тих кошиках, де не було товарів поточного користувача (див. Рисунок 3.19 ).

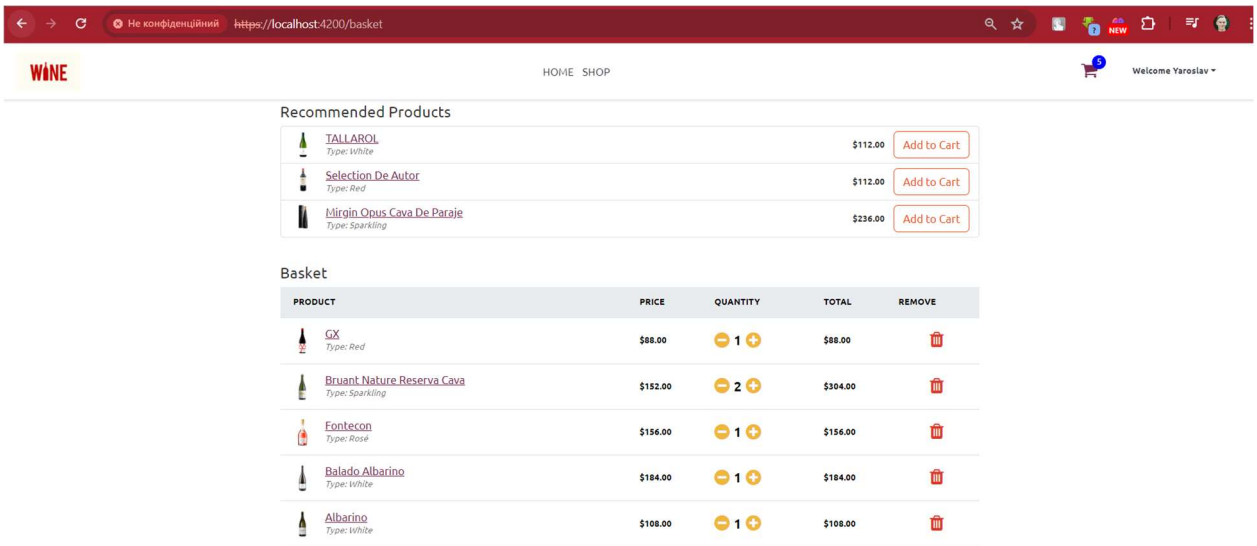


Рисунок 3.19 – Комплементарні товари після додавання одного з рекомендованих товарів («Albarino»).

Нижче ж можна ознайомитись з інформацією про загальну суму товарів в кошику та натиснувши на кнопку Proceed to checkout додати зміст кошику в Redis (див. Рисунок 3.20 ).

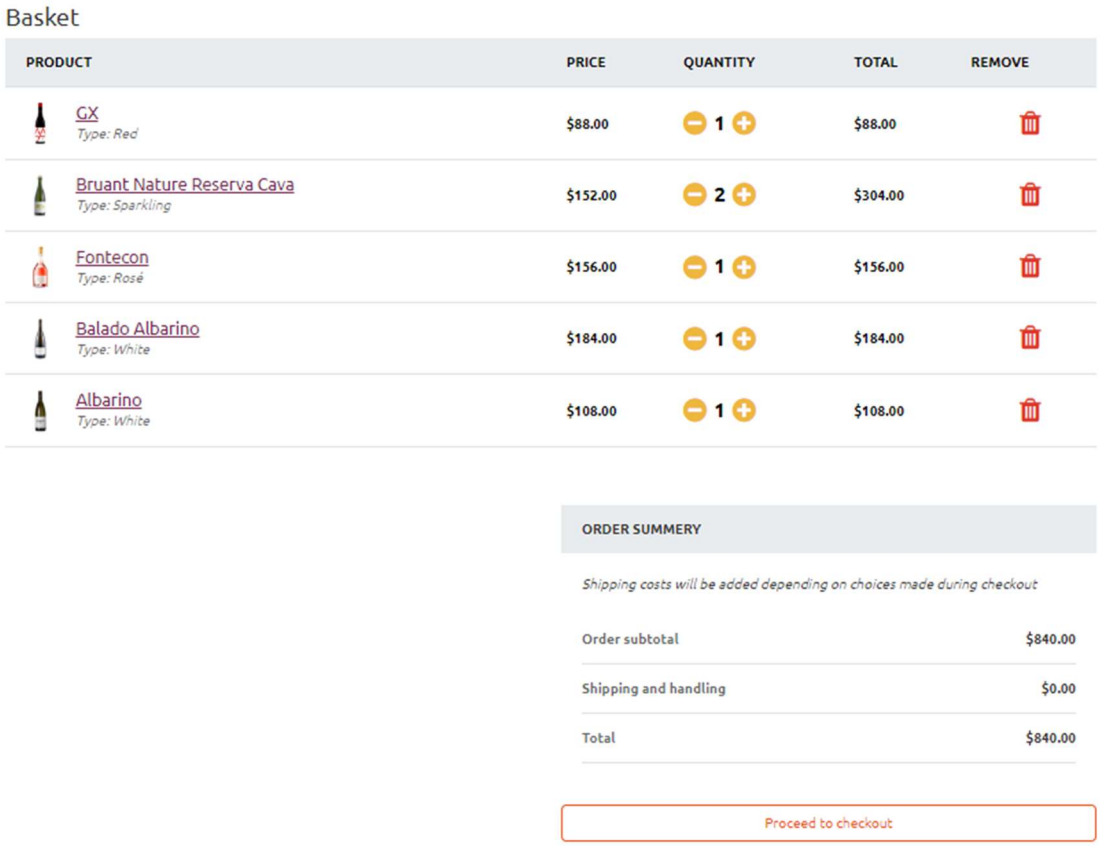


Рисунок 3.20 – Прорахунок загальної суми товарів та кнопка додавання кошику в Redis.

# ВИСНОВКИ

В результаті виконання кваліфікаційної роботи створено інформаційну систему для пошуку комплементарних товарів на основі колаборативної фільтрації.

У ході виконання кваліфікаційної роботи було виконано такі завдання:

1. Проведено огляд літератури та досліджено сучасні підходи до колаборативної фільтрації, включаючи методи на основі сусідів, моделі та гібридні методи. Розглянуті підходи:
    - Методи на основі сусідів (neighborhood-based), які включають рекомендації на основі користувачів (user-based) та на основі елементів (item-based). У першому випадку, інтерес користувача до об'єкта оцінюється на основі оцінок інших користувачів з подібними вподобаннями. У другому випадку, рейтинг користувача для об'єкта передбачається на основі його рейтингів для схожих об'єктів.
    - Методи на основі моделей (model-based), що використовують збережені рейтинги для навчання прогностичних моделей. Ці моделі враховують латентні характеристики користувачів та об'єктів, такі як класи уподобань і категорії об'єктів, і використовуються для прогнозування оцінок для нових товарів.
    - Гібридні методи, які поєднують елементи обох підходів для підвищення точності і релевантності рекомендацій.
2. Виконано аналіз предметної області та існуючих веб-додатків, які використовують колаборативну фільтрацію для підбору комплементарних товарів.
3. Визначено та обрано інструменти для розробки веб-додатку виходячи з їх переваг та недоліків. В нашому випадку найбільше відповіадє вимогам фреймворк ASP.NET Core для написання серверної частини

додатку, фреймворк Angular для клієнтської частини, СУБД SQLite для зберігання даних про вина, бренди та типи, а також користувачів та in-memory БД Redis, для збереження даних про товари в кошиках користувачів.

4. Розроблено архітектуру додатку та визначено спосіб його розміщення.
5. Реалізовано модель інформаційної системи для надання рекомендацій комплементарних товарів до товарів з кошику поточного користувача.
6. Реалізовано основний функціонал сайту, а саме додано можливість: реєстрації та авторизації за допомогою логіну та паролю, додавання та видалення товарів з кошика, зміни кількості товарів в кошику, сортування товарів за ціною і назвою, фільтрація товарів за сортом і брендом, пошук товарів по назві, пагінація сторінок, додавання товару до кошика та перегляд основних сторінок сайту.
7. Проведено тестування додатку і було визначено, що розроблена колаборативна фільтрація для пошуку комплементарних товарів працює так, як і очікувалось.

Застосування розробленої інформаційної системи зменшує кількість необхідних ресурсів для пошуку комплементарних товарів та скорочує час на прийняття рішень щодо покупок. Використання колаборативної фільтрації для аналізу куплених товарів користувачами та надання рекомендацій щодо комплементарних товарів окремо від основного списку покупок значно покращує користувацький досвід. Це сприяє збільшенню клієнтської бази та підвищенню прибутків, оскільки користувачі отримують персоналізовані та релевантні рекомендації без необхідності довгих пошуків.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Савицький А. Й., Попович Д. В. Методи колаборативної фільтрації для непрямих рейтингів. URL: https://ela.kpi.ua/server/api/core/bitstreams/b9c2eea7-9bbb-482e-8cd5-b02d924ab808/content (дата звернення: 13.05.2024).
2. Recommender systems handbook / ed. by F. Ricci et al. Boston, MA : Springer US, 2011. URL: https://www.cse.iitk.ac.in/users/nsrivast/HCC/Recommender_systems_handbook.pdf (дата звернення: 13.05.2024).
3. Чорненький М. А. Інформаційна система підбору пісень на основі вподобань користувача. SumDU Repository: Home. URL: https://essuir.sumdu.edu.ua/bitstream-download/123456789/90627/1/Chornenkiy_mag_rob.pdf (дата звернення: 13.05.2024).
4. Breese, J.S., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: Proc. of the 14th Annual Conf. on Uncertainty in Artificial Intelligence, pp. 43–52. Morgan Kaufmann (1998). URL: https://www.researchgate.net/publication/235357340_Empirical_Analysis_of_Predictive_Algorithm_for_Collaborative_Filtering (дата звернення: 13.05.2024).
5. Nandanwar S., Murty N. M. Structural neighborhood based classification of nodes in a network. *SIGKDD*. URL: https://www.kdd.org/kdd2016/papers/files/Paper_679.pdf (дата звернення: 13.05.2024).
6. Netflix – Serien online ansehen, Filme online ansehen. Netflix - Watch TV Shows Online, Watch Movies Online. URL: https://www.netflix.com/ua/ (дата звернення: 12.07.2024).

7. Nike air jordan 12 WNBA brilliant orange black. *eBay*. URL: https://www.ebay.com/itm/126482903218 (дата звернення: 12.05.2024).
8. Мобільний телефон samsung galaxy A24 6/128GB black. *Rozetka*. URL: https://rozetka.com.ua/ua/samsung-sm-a245fzkvsek/p375225780/ (дата звернення: 12.05.2024).
9. Amazon. *Amazon*. URL: https://www.amazon.com/ (дата звернення: 12.07.2024).
10. Goossaert E. What Are Single-Page Applications (SPA)?. *Medium*. URL: https://medium.com/@egoossaert/what-are-single-page-applications-spa-addeaf6717cc (дата звернення: 11.05.2024).
11. Клієнт-серверна архітектура. JavaRush. URL: https://javarush.com/ua/quests/lectures/ua.questservlets.level14.lecture00 (дата звернення: 11.07.2024).
12. [Принцип роботи трирівневої архітектури]. URL: https://training.qatestlab.com/wp-content/uploads/2020/04/333.jpg (дата звернення: 10.07.2024).
13. ASP.NET core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0 (дата звернення: 09.07.2024).
14. About SQLite. *SQLite About Page*. URL: https://www.sqlite.org/about.html (date of access: 08.05.2024).
15. Angular. Angular. URL: https://angular.io/ (дата звернення: 18.07.2024).
16. JWT.IO - JSON web tokens introduction. *JSON Web Tokens - jwt.io*. URL: https://jwt.io/introduction (дата звернення: 09.05.2024).
17. Redis - the real-time data platform. *Redis*. URL: https://redis.io/ (дата звернення: 08.05.2024).

18. Angular folder structure best practices. TekTutorialsHub. URL: https://www.tektutorialshub.com/angular/angular-folder-structure-best-practices/ (дата звернення: 05.05.2024).
19. Singleton. *Refactoring and Design Patterns*. URL: https://refactoring.guru/design-patterns/singleton (дата звернення: 18.05.2024).
20. Про проблему DTO та шляхи її вирішення. DOU. URL: https://dou.ua/forums/topic/43912/ (дата звернення: 05.05.2024).
21. Docker: accelerated container application development. *Docker*. URL: https://www.docker.com/ (дата звернення: 07.05.2024).
22. Postman home page. *Postman*. URL: https://www.postman.com/ (date of access: 11.05.2024).

# ДОДАТОК

# РЕАЛІЗАЦІЯ КОЛАБОРАТИВНОЇ ФІЛЬТРАЦІЇ

```
// IBasketRepository.cs – інтерфейс для BasketRepository.
using Core.Entities;

namespace Core.Interfaces
{
    public interface IBasketRepository
    {
        Task<CustomerBasket> GetBasketAsync(string basketId);
        Task<CustomerBasket> UpdateBasketAsync(CustomerBasket basket);
        Task<List<BasketItem>> FindComplementaryItemAsync(string basketId);
        Task<bool> DeleteBasketAsync(string basketId);
    }
}


// BasketRepository.cs – клас, що відповідає за отримання даних з кошику
користувача, їх оновлення та видалення кошиків а також який містить пошук
комплементарних товарів на основі колаборативної фільтрації.
using System.Text.Json;
using Core.Entities;
using Core.Interfaces;
using StackExchange.Redis;

namespace Infrastructure.Data
{
    public class BasketRepository : IBasketRepository
    {
        private readonly IDatabase _database;
        public BasketRepository(IConnectionMultiplexer redis)
        {
            _database = redis.GetDatabase();
        }

        public async Task<bool> DeleteBasketAsync(string basketId)
        {
            return await _database.KeyDeleteAsync(basketId);
        }

        public async Task<List<BasketItem>> FindComplementaryItemAsync(string
basketId)
        {
            var data = await _database.StringGetAsync(basketId);
            var userBasket =
JsonSerializer.Deserialize<CustomerBasket>(data);
            var userItems = userBasket.Items;

            if (userItems == null || !userItems.Any())
            {
                return new List<BasketItem>();
            }

            var allBasketKeys = await GetAllBasketKeysAsync();
```

```
            var highPriorityRecommendations = new Dictionary<string,
BasketItem>();
            var lowPriorityRecommendations = new Dictionary<string,
BasketItem>();

            foreach (var key in allBasketKeys)
            {
                if (key == basketId)
                {
                    continue;
                }

                var basketData = await _database.StringGetAsync(key);
                if (!basketData.IsNullOrEmpty)
                {
                    var basket =
JsonSerializer.Deserialize<CustomerBasket>(basketData);
                    bool hasUserItem = basket.Items.Any(item =>
userItems.Any(ui => ui.ProductName == item.ProductName));

                    foreach (var item in basket.Items)
                    {
                        if (userItems.Any(ui => ui.ProductName ==
item.ProductName))
                        {
                            continue;
                        }

                        var recommendations = hasUserItem ?
highPriorityRecommendations : lowPriorityRecommendations;

                        if (recommendations.ContainsKey(item.ProductName))
                        {
                            recommendations[item.ProductName].Quantity +=
item.Quantity;
                        }
                        else
                        {
                            recommendations[item.ProductName] = new
BasketItem
                            {
                                Id = item.Id,
                                ProductName = item.ProductName,
                                Price = item.Price,
                                Quantity = item.Quantity,
                                PictureUrl = item.PictureUrl,
                                Brand = item.Brand,
                                Type = item.Type
                            };
                        }
                    }
                }
            }

            var highPriorityProductNames = new
HashSet<string>(highPriorityRecommendations.Keys);
            var filteredLowPriorityRecommendations =
lowPriorityRecommendations
```

```
                .Where(kvp => !highPriorityProductNames.Contains(kvp.Key))
                .Select(kvp => kvp.Value);

            var finalRecommendations = highPriorityRecommendations.Values
                .OrderByDescending(item => item.Quantity)

.Concat(filteredLowPriorityRecommendations.OrderByDescending(item =>
item.Quantity))
                .Take(3)
                .ToList();

            return finalRecommendations;
        }

        private async Task<List<string>> GetAllBasketKeysAsync()
        {
            var server =
_database.Multiplexer.GetServer(_database.Multiplexer.GetEndPoints().First())
;
            var keys = new List<string>();

            await foreach (var key in server.KeysAsync(pattern: "basket*"))
            {
                keys.Add(key);
            }

            return keys;
        }

        public async Task<CustomerBasket> GetBasketAsync(string basketId)
        {
            var data = await _database.StringGetAsync(basketId);

            return data.IsNullOrEmpty ? null :
JsonSerializer.Deserialize<CustomerBasket>(data);
        }

        public async Task<CustomerBasket> UpdateBasketAsync(CustomerBasket
basket)
        {
            var created = await _database.StringSetAsync(basket.Id,
                JsonSerializer.Serialize(basket), TimeSpan.FromDays(30));

            if (!created) return null;

            return await GetBasketAsync(basket.Id);
        }
    }
}


// BaseApiController.cs - Базовий для всіх контролер.
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
```

```
    public class BaseApiController : ControllerBase
    {
    }
}


// BasketController.cs - контролер, в якому знаходяться кінцеві пункти до методів отримання, оновлення та видалення даних з кошику, а також отримання рекомендацій.
using API.Dtos;
using AutoMapper;
using Core.Entities;
using Core.Interfaces;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers
{
    public class BasketController : BaseApiController
    {
        private readonly IBasketRepository _basketRepository;
        private readonly IMapper _mapper;
        public BasketController(IBasketRepository basketRepository, IMapper
mapper)
        {
            _mapper = mapper;
            _basketRepository = basketRepository;
        }

        [HttpGet]
        public async Task<ActionResult<CustomerBasket>> GetBasketById(string
id)
        {
            var basket = await _basketRepository.GetBasketAsync(id);

            return Ok(basket ?? new CustomerBasket(id));
        }

        [HttpGet("recomendation")]
        public async Task<ActionResult<List<BasketItem>>>
GetRecomendationsByBasketId(string id)
        {
            var recomendations = await
_basketRepository.FindComplementaryItemAsync(id);

            return Ok(recomendations);
        }

        [HttpPost]
        public async Task<ActionResult<CustomerBasket>>
UpdateBasket(CustomerBasketDto basket)
        {
            var customerBasket = _mapper.Map<CustomerBasketDto,
CustomerBasket>(basket);
            var updatedBasket = await
_basketRepository.UpdateBasketAsync(customerBasket);

            return Ok(updatedBasket);
        }
```

```
        [HttpDelete]
        public async Task DeleteBasketAsync(string id)
        {
            await _basketRepository.DeleteBasketAsync(id);
        }
    }
}
```

```
// basket.service.ts - сервіс, який потрібен для отримання певних даних про кошик користувача за допомогою запитів до кінцевих пунктів класу BasketController та рекомендацій товарів, а також в якому прописані методи взаємодії з кошиком та товарами.
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs';
import { map } from 'rxjs/operators';
import { Basket, IBasket, IBasketItem, IBasketTotals } from
'../shared/models/basket';
import { HttpClient } from '@angular/common/http';
import { IProduct } from '../shared/models/product';

@Injectable({
    providedIn: 'root'
})
export class BasketService {
    baseUrl = 'https://localhost:7151/api/';
    private basketSource = new BehaviorSubject<IBasket>(null);
    basket$ = this.basketSource.asObservable();
    private basketTotalSource = new BehaviorSubject<IBasketTotals>(null);
    basketTotal$ = this.basketTotalSource.asObservable();

    constructor(private http: HttpClient) { }

    getBasket(id: string) {
        return this.http.get(this.baseUrl + 'basket?id=' + id)
            .pipe(
                map((basket: IBasket) => {
                    this.basketSource.next(basket);
                    this.calculateTotals();
                })
            );
    }

    getRecomendations(basket: IBasket) {
        return this.http.get<IBasketItem[]>(this.baseUrl +
'basket/recomendation?id=' + basket.id)
    }

    setBasket(basket: IBasket) {
        return this.http.post(this.baseUrl + 'basket',
basket).subscribe((response: IBasket) => {
            this.basketSource.next(response);
            this.calculateTotals();
        }, error => {
            console.log(error);
        });
    }
```

```
    deleteBasket(basket: IBasket) {
        return this.http.delete(this.baseUrl + 'basket?id=' +
basket.id).subscribe(() => {
            this.basketSource.next(null);
            this.basketTotalSource.next(null);
            localStorage.removeItem('basket_id');
        }, error => {
            console.log(error);
        });
    }

    getCurrentBasketValue() {
        return this.basketSource.value;
    }

    addItemToBasket(item: IProduct, quantity = 1) {
        const itemToAdd: IBasketItem = this.mapProductItemToBasketItem(item,
quantity);
        const basket = this.getCurrentBasketValue() ?? this.createBasket();
        basket.items = this.addOrUpdateItem(basket.items, itemToAdd,
quantity);
        this.setBasket(basket);
    }

    incrementItemQuantity(item: IBasketItem) {
        const basket = this.getCurrentBasketValue();
        const foundItemIndex = basket.items.findIndex(x => x.id === item.id);
        basket.items[foundItemIndex].quantity++;
        this.setBasket(basket);
    }

    decrementItemQuantity(item: IBasketItem) {
        const basket = this.getCurrentBasketValue();
        const foundItemIndex = basket.items.findIndex(x => x.id === item.id);

        if (basket.items[foundItemIndex].quantity > 1) {
            basket.items[foundItemIndex].quantity--;
            this.setBasket(basket);
        } else {
            this.removeItemFromBasket(item);
        }
    }

    removeItemFromBasket(item: IBasketItem) {
        const basket = this.getCurrentBasketValue();
        if (basket.items.some(x => x.id === item.id)) {
            basket.items = basket.items.filter(i => i.id !== item.id)

            if (basket.items.length > 0) {
                this.setBasket(basket);
            } else {
                this.deleteBasket(basket);
            }
        }
    }

    private calculateTotals() {
        const basket = this.getCurrentBasketValue();
```

```
        const shipping = 0;
        const subtotal = basket.items.reduce((sum, item) => (item.price *
item.quantity) + sum, 0);
        const total = subtotal + shipping;
        this.basketTotalSource.next({shipping, total, subtotal});
    }

    private addOrUpdateItem(items: IBasketItem[], itemToAdd: IBasketItem,
quantity: number):
    IBasketItem[] {
        const index = items.findIndex(i => i.id === itemToAdd.id);

        if (index === -1) {
            itemToAdd.quantity = quantity;
            items.push(itemToAdd);
        } else {
            items[index].quantity += quantity;
        }

        return items;
    }

    private createBasket(): IBasket {
        const basket = new Basket();
        localStorage.setItem('basket_id', basket.id);

        return basket;
    }

    private mapProductItemToBasketItem(item: IProduct, quantity: number):
IBasketItem {
        return {
            id: item.id,
            productName: item.name,
            price: item.price,
            pictureUrl: item.pictureUrl,
            quantity,
            brand: item.productBrand,
            type: item.productType
        };
    }
}

// basket.component.ts – компонент кошику, в якому прописано методи, які
використовуються при натиску певних кнопок на html сторінці кошику або інших
змін на сторінці, а також прописана ініціалізація компонента і зміна
рекомендацій відносно змін в кошику.
import { Component, OnInit } from '@angular/core';
import { IBasket, IBasketItem } from '../shared/models/basket';
import { Observable, of, switchMap } from 'rxjs';
import { BasketService } from './basket.service';

@Component({
  selector: 'app-basket',
  templateUrl: './basket.component.html',
  styleUrls: ['./basket.component.scss']
})
export class BasketComponent implements OnInit {
```

```
    basket$: Observable<IBasket>;
    recomendations$: Observable<IBasketItem[]>;

    constructor(private basketService: BasketService) {}

    ngOnInit() {
        this.basket$ = this.basketService.basket$;
        this.recomendations$ = this.basket$.pipe(
            switchMap(basket => {
                if (basket) {
                    return this.basketService.getRecomendations(basket);
                }
                return of([]);
            })
        )as Observable<IBasketItem[]>;
    }

    removeBasketItem(item: IBasketItem) {
        this.basketService.removeItemFromBasket(item);
    }

    incrementItemQuantity(item: IBasketItem) {
        this.basketService.incrementItemQuantity(item);
    }

    decrementItemQuantity(item: IBasketItem) {
        this.basketService.decrementItemQuantity(item);
    }
}
```