ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

КВАЛІФІКАЦІЙНА РОБОТА

Тема: «Вебсайт інтернет-магазину комп’ютерних комплектуючих з використанням багатокритеріальних фільтрів та рекомендаційної системи (комплексна робота)»

Ступінь вищої освіти – бакалавр

Спеціальність – 122 «Комп’ютерні науки»

Освітня програма «Комп’ютерні науки»

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав: здобувач 4 курсу
групи КН-21
Роман КОСЕНКО

Керівники: доцент кафедри комп’ютерних наук, кандидат технічних наук
Олександр ПОЛІЩУК
асистент кафедри комп’ютерних наук
Микита ФЕЩЕНКО

Засвідчую, що кваліфікаційна робота оформлена відповідно до ДСТУ 3008:2015 та не містить запозичень з праць інших авторів без відповідних посилань.

Здобувач: _____
(підпис)

м. Київ – 2025 рік

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

ЗАТВЕРДЖУЮ:
завідувач кафедри
комп'ютерних наук
_____Сергій МІЧКІВСЬКИЙ
«____»____20___р

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
Косенко Роман Дмитрович

Тема роботи	Вебсайт інтернет-магазину комп'ютерних комплектуючих з використанням багатокритеріальних фільтрів та рекомендаційної системи (комплексна робота)
Номер та дата наказу про затвердження теми	№121-7 від 24 грудня 2024 року
Коротка постановка завдання	Розробити функціональний вебсайт інтернет-магазину комп'ютерних комплектуючих, який забезпечує зручну навігацію для користувачів, реалізацію багатокритеріальних фільтрів для точного підбору товарів та впровадження рекомендаційної системи для персоналізованих пропозицій.
Посилання на джерела інформації (не більше п'яти найменувань, які рекомендує науковий керівник)	Microsoft. ASP.NET Core Documentation [Електронний ресурс]. – URL: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0 (дата звернення: 21.04.2025). Microsoft. Entity Framework Core [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 21.04.2025).
Вимоги до кваліфікаційної роботи	Кваліфікаційна робота має містити теоретичне, системотехнічне або експериментальне дослідження за темою роботи, яку слід розглядати як складне спеціалізоване завдання або практичну проблему в галузі комп'ютерних наук, яка характеризується комплексністю та невизначеністю умов і потребує застосування теорій і методів інформаційних технологій.

Дата видачі завдання «27» грудня 2024 р.

Керівник Олександр ПОЛІЩУК

Керівник Микита ФЕЩЕНКО

Здобувач освітнього ступеня бакалавра Роман КОСЕНКО

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання	Примітка
Підготовчий етап			
1	Вибір напряму дослідження	02.12.2024 р.	*виконано*
2	Формування теми та призначення керівника	16.12.2024 р.	*виконано*
3	Затвердження теми кваліфікаційної роботи	23.12.2024 р.	*виконано*
4	Затвердження завдання на кваліфікаційну роботу	27.12.2024 р.	*виконано*
Основний етап			
5	Розробка концепції кваліфікаційної роботи	13.01.2025 р.	*виконано*
6	Підбір та вивчення джерел інформації з напряму дослідження. Огляд існуючих аналогів	20.01.2025 р.	*виконано*
7	Затвердження розширеної постановки завдання. Підготовка та подання керівникові розділу 1 кваліфікаційної роботи	10.03.2025 р.	*виконано*
8	Проєктування. Підготовка та подання керівникові розділу 2 кваліфікаційної роботи	24.03.2025 р.	*виконано*
9	Підготовка доповіді для експертизи стану виконання кваліфікаційної роботи (проміжний контроль)	31.03-04.04.2025 р.	*виконано*
10	Реалізація. Підготовка та подання керівникові розділу 3 кваліфікаційної роботи	07.04.2025 р.	*виконано*
11	Підготовка та подання керівнику першого варіанту всієї кваліфікаційної роботи	14.04.2025 р.	*виконано*
12	Доопрацювання кваліфікаційної роботи з урахуванням зауважень керівника та представлення керівникові доопрацьованого варіанту кваліфікаційної роботи	21.04.2025 р.	*виконано*
Завершальний етап			
13	Представлення рукопису для перевірки на плагіат	28.04-04.05.2025 р.	*виконано*
14	Підготовка презентації та доповіді на передзахист	05.05-11.05.2025 р.	*виконано*
15	Передзахист кваліфікаційної роботи	12.05-16.05.2025 р.	*виконано*
16	Доопрацювання роботи за результатами передзахисту	19.05-06.06.2025 р.	*виконано*
17	Експертиза роботи керівником та зовнішнім експертом	09.06-15.06.2025 р.	*виконано*
18	Доопрацювання доповіді та презентації для захисту	09.06-15.06.2025 р.	*виконано*
19	Захист кваліфікаційної роботи	16.06-22.06.2025 р.	*виконано*

Керівник Олександр ПОЛІЩУК

Керівник Микита ФЕЩЕНКО

Здобувач освітнього ступеня бакалавра Роман КОСЕНКО

Косенко Р.Д. Вебсайт інтернет-магазину комп'ютерних комплектуючих з використанням багатокритеріальних фільтрів та рекомендаційної системи (комплексна робота).

Пояснювальна записка кваліфікаційної роботи за спеціальністю 122 – Комп'ютерні науки (освітня програма – Комп'ютерні науки) СО Бакалавр. – ВНЗ «Університет економіки та права «КРОК», Навчально-науковий інститут інформаційних та комунікаційних технологій, кафедра комп'ютерних наук, Київ, 2025.

У роботі досліджено процес розробки серверної частини (backend) вебсайту інтернет-магазину комп'ютерних комплектуючих. Основну увагу приділено реалізації багатокритеріальних фільтрів для точного підбору товарів, рекомендаційної системи для персоналізованих пропозицій.

Ключові слова: backend, вебсайт, інтернет-магазин, комп'ютерні комплектуючі.

Табл. 3. Рис. 21. Бібліограф.: 17 найм.

Kosenko R.D. Website of the online store of computer components using multi-criteria filters and recommendation system (complex work).

Explanatory note of the qualification work in the specialty 122 – Computer Science (educational program – Computer Science) BA Bachelor. – Higher Educational Institution "University of Economics and Law "KROK" Educational and Scientific Institute of Information and Communication Technologies, Department of Computer Science, Kyiv, 2025.

This work investigates the process of developing the backend of the website of an online store of computer components. The main attention is paid to the implementation of multi-criteria filters for accurate selection of goods, and a recommendation system for individual offers.

Keywords: backend, website, online store, computer components.

Tabl. 3. Fig. 21. Bibliography: 17 Items.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

API (Application Programming Interface) – це посередник між програмами, який задає правила «спілкування». Тому в його назві закладено поняття «інтерфейс» – межа між об’єктами. Одній програмі не важливо, як працює інша. Розробники просто користуються інтерфейсом: надсилають запит і отримують зворотну відповідь [1].

Backend або бекенд – це серверна частина застосунку, яка забезпечує взаємодію клієнтської частини з базами даних, бізнес-логікою та іншими зовнішніми системами. [2].

Кешування – це процес збереження даних у кеш (місце, куди комп’ютер тимчасово зберігає дані, що часто використовуються). Правила цього процесу встановлює адміністратор програми або сайту. У налаштуваннях сервера можна вибрати, що саме пристрої відвідувачів повинні зберігати як довго [3].

ВСТУП

Актуальність теми знаходиться на перетині двох динамічних сучасних сфер – електронної комерції та інформаційних технологій. Інтернет-магазини стали невід'ємною частиною цифрової економіки, надаючи користувачам швидкий доступ до широкого асортименту товарів, зручний пошук, персоналізовані рекомендації та онлайн-оплати.

Особливу складність становить сегмент комп'ютерних комплектуючих, де вибір товарів визначається великою кількістю технічних характеристик (тип роз'єму процесору, об'єм пам'яті, частота, TDP тощо). У таких умовах багатокритеріальні фільтри стають необхідним інструментом, що дозволяє користувачам точно підібрати продукцію відповідно до своїх потреб.

Крім того, на сучасному ринку зростає потреба у персоналізованих рішеннях: рекомендаційних системах, збереженні обраного, порівнянні товарів та підтримці багатомовності. Успішна реалізація цих функцій потребує якісної серверної логіки, побудованої за сучасними принципами продуктивності, безпеки та масштабованості.

Зважаючи на це, розробка серверної частини вебсайту інтернет-магазину комп'ютерних комплектуючих є актуальним завданням, що поєднує прикладне програмування, проєктування баз даних, побудову API та інтеграцію з хмарними платформами.

Метою проєкту є створення серверної частини вебсайту інтернет-магазину комп'ютерних комплектуючих, яка забезпечує:

- зручну навігацію та точну фільтрацію товарів за численними параметрами;
- персоналізовану систему рекомендацій;
- функціонал обраного, порівняння, відгуків і замовлень;
- безпечну авторизацію користувачів;
- інтеграцію з базою даних Supabase.

Завдання дослідження:

- сформулювати основні вимоги до функціоналу інтернет-магазину;
- провести аналіз аналогів та визначити їх переваги й недоліки;
- дослідити методи реалізації багатокритеріальних фільтрів і рекомендаційних систем;
- спроєктувати структуру бази даних і архітектуру бекенду;
- реалізувати серверну частину з урахуванням сучасних вимог до зручності, безпеки та масштабованості.

Об'єктом дослідження є серверна частина (backend) вебсайту для інтернет-магазину комп'ютерних комплектуючих.

Предметом дослідження архітектура, функціональність і реалізація серверної логіки в системах електронної комерції, орієнтованих на комп'ютерну техніку.

Практична цінність проєкту полягає у створенні реального програмного рішення - повнофункціонального серверного застосунку для інтернет-магазину комп'ютерних комплектуючих, яке може бути адаптоване або масштабоване для реального комерційного використання.

Проєкт демонструє:

- ефективне застосування сучасного стеку технологій (ASP.NET Core [4], Entity Framework Core [5], Supabase [6], PostgreSQL [7]);
- побудову багатокритеріальної фільтрації товарів з перекладами, одиницями виміру та діапазонами значень;
- реалізацію персоналізованої рекомендаційної системи на основі історії переглядів, сумісності товарів, цінової близькості та популярності;
- функціонал обраного (wishlist), порівняння товарів, пошуку з підтримкою синонімів, рейтингової оцінки та відгуків;
- розмежування доступу користувачів через безпечну авторизацію та зв'язок із базою даних;

- структура замовлень реалізована у форматі прототипу з можливістю подальшого розширення для реального комерційного використання.

Архітектура проєкту дозволяє легко масштабувати окремі компоненти системи при зростанні навантаження. Легко підтримується та розширюється, відповідає вимогам сучасної веброзробки та підходить як для навчальних, так і для прикладних комерційних цілей. Такий підхід може бути основою для побудови справжнього проєкту електронної комерції або MVP продукту.

Структура та обсяг пояснювальної записки. Пояснювальна записка до проєкту складається зі вступу, трьох розділів, висновків, списку посилань (17 найменувань) та 1 додаток. Пояснювальна записка містить 21 рисунок, 3 таблиці. Загальний обсяг пояснювальної записки складає 49 сторінок, основний зміст викладено на 43 сторінках.

РОЗДІЛ 1
ПОСТАНОВКА ЗАВДАННЯ НА РОЗРОБКУ

1.1 Предметна область

Інтернет-магазин – це онлайн-платформа для продажу товарів, яка забезпечує користувачам можливість переглядати асортимент, здійснювати пошук, додавати товари до кошика, оформлювати замовлення та залишати відгуки. Основною складовою такої платформи є серверна частина (бекенд), яка відповідає за обробку запитів, управління базою даних, авторизацію користувачів, бізнес-логіку та взаємодію з клієнтською частиною [8].

У даній роботі як серверна платформа використовується ASP.NET Core. Supabase як хмарна бекенд-інфраструктура з підтримкою PostgreSQL, а також Entity Framework Core для взаємодії з базою даних. Код проєкту доступний у відкритому репозиторії GitHub [9].

Ринок комп’ютерних комплектуючих є одним із динамічних у сфері електронної комерції, через що виникають підвищені вимоги до продуктивності, масштабованості та персоналізації сервісів. Зокрема, користувачам важливо швидко знаходити товари, що відповідають їхнім критеріям (частота, об’єм пам’яті, тип роз’єму тощо), а також отримувати персоналізовані рекомендації та мати змогу зберігати улюблені товари.

Основні виклики, з якими стикається бекенд-розробка подібного інтернет-магазину:

- високе навантаження на сервер через велику кількість товарів, запитів і фільтрацій;
- складність реалізації багатокритеріального пошуку з підтримкою діапазонів, перекладів і синонімів;
- реалізація персоналізованих рекомендацій, зокрема на основі історії переглядів, обраного, сумісності та популярності;
- безпека зберігання персональних даних, авторизації та замовлень;

- підтримка масштабування та кешування для стабільної роботи API.

Для вирішення цих проблем у розробленому проєкті реалізовано:

- оптимізовану структуру бази даних на PostgreSQL з індексацією та нормалізацією;
- серверну логіку багатокритеріальних фільтрів з динамічним формуванням діапазонів і перекладів;
- систему рекомендацій, яка враховує характеристики товарів, поведінку користувача та популярність;
- реалізацію додавання до списику обраного (wishlist), порівняння, пошуку з підтримкою синонімів;
- захищену авторизацію та авторизацію через Supabase Authentication;
- REST API з чіткою структурою запитів, побудоване за принципами Clean Architecture.

Таким чином, бекенд є критичною частиною вебсайту, що забезпечує ефективну, безпечну та масштабовану роботу всієї системи.

1.2 Огляд аналогів

Для аналізу сучасних підходів до архітектури бекенду, продуктивності та функціональних можливостей інтернет-магазинів у сфері комп'ютерних комплектуючих було розглянуто кілька конкурентних платформ. Основну увагу приділено таким аспектам: пошук і фільтрація товарів, рекомендаційні системи, інтеграція з API та юзабіліті.

GameHall – це інтернет-магазин ігрової техніки (рис. 1.1).

Переваги аналога:

- чітке розділення товарів за категоріями;
- зручна навігація, проста структура даних (ID товарів).

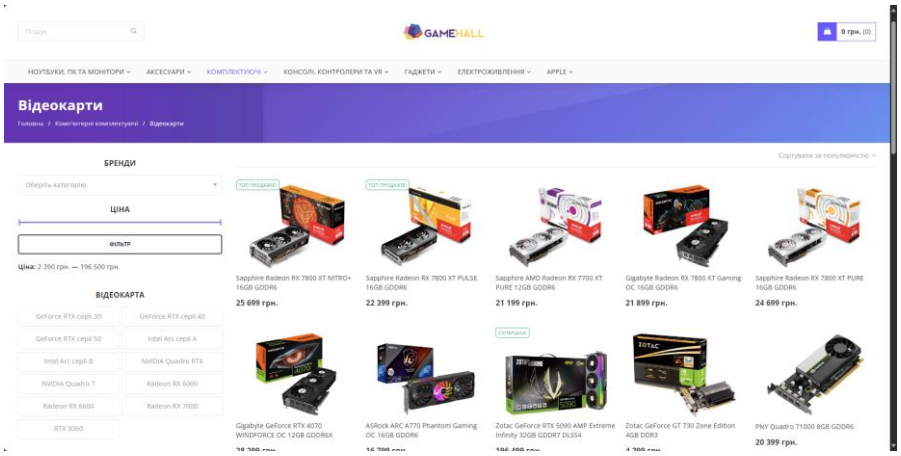


Рисунок 1.1 – Каталог відеокарт на GameHall.

Джерело: [10]

Недоліки аналога:

– повільна обробка фільтрів при великій кількості товарів;

– відсутність автоматизованої обробки замовлень чи повідомлень;

– недостатня персоналізація.

CompX, як і GameHall, здійснює електронну торгівлю ігровим обладнанням (рис. 1.2).

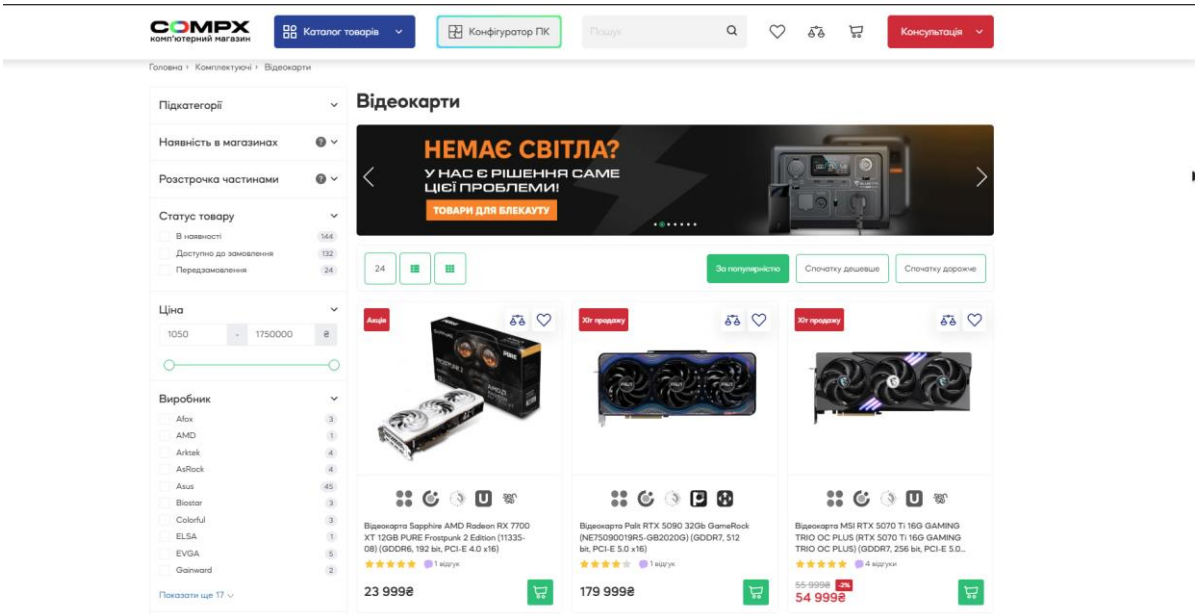


Рисунок 1.2 – Каталог відеокарт на CompX.

Джерело: [11]

Переваги аналога:

– розширена система характеристик (корисна для досвідчених користувачів);
– стабільна робота при високому навантаженні.

Недоліки аналога:

– відсутність персоналізованих рекомендацій;
– застарілий пошук без підтримки синонімів або автозаміни;
– відсутній публічний API для зовнішніх сервісів.

Telemart Інтернет-магазин електроніки (рис. 1.3).

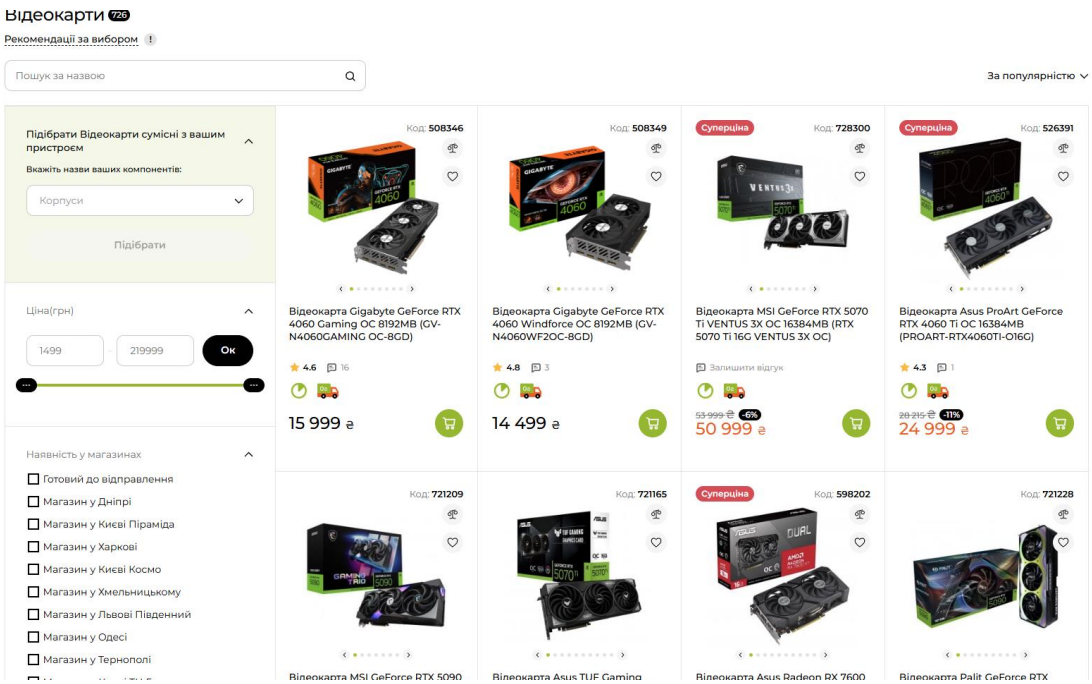


Рисунок 1.3 – Каталог відеокарт на Telemart.

Джерело: [12]

Переваги аналога:

– висока швидкодія завдяки кешуванню;
– багатофункціональні фільтри;
– інтеграція з платіжними системами та службами доставки;

– масштабованість при пікових навантаженнях.

Недоліки аналога:

– відсутність рекомендацій на основі історії переглядів;

– недостатня адаптація пошуку під потреби користувача (немає синонімів, автозаповнення).

1.3 Постановка завдання на кваліфікаційну роботу

Беручи до уваги результати аналізу предметної області та функціональність існуючих аналогів, основне завдання цієї роботи полягає у створенні серверної частини (бекенду) вебсайту інтернет-магазину комп’ютерних комплектуючих.

Система повинна забезпечувати реалізацію ключових функцій, зокрема:

– обробку пошукових запитів з підтримкою синонімів та часткових збігів;

– багатокритеріальну фільтрацію товарів за числовими та текстовими параметрами, включно з діапазонами;

– побудову персоналізованої системи рекомендацій на основі переглядів, цінової близькості, подібних характеристик і сумісності компонентів;

– можливість додавання товарів до обраного (wishlist) та створення списків для порівняння;

– реалізацію рейтингової оцінки товарів та залишення відгуків лише після покупки;

– збереження інформації про замовлення користувача без реалізації платіжної інтеграції (як демонстраційний функціонал);

– безпечну реєстрацію, авторизацію та оновлення облікових даних з використанням Supabase Authentication;

– ефективну структуру бази даних PostgreSQL із використанням ORM Entity Framework Core.

Проєкт не передбачає повноцінної інтеграції з платіжними сервісами, але створена архітектура дозволяє у майбутньому додати таку можливість без значних змін у логіці системи.

У таблиця 1.1 описані до функціональних та нефункціональних вимог, а також у таблиці 1.2 представлено вхідні та вихідні дані.

Таблиця 1.1 – Функціональні та нефункціональні вимоги

Функціональні вимоги	**Нефункціональні вимоги**
Пошук товарів за ключовими словами з підтримкою синонімів	Оптимізація запитів до бази даних для швидкого доступу
Багатокритеріальні фільтри з діапазонами	Висока швидкодія при обробці запитів до серверної частини
Система персоналізованих рекомендацій (перегляди, ціна, сумісність, популярність)	Побудова структури, що дозволяє масштабування проєкту в майбутньому
Додавання товарів до обраного (wishlist) та порівняння	Безпечна авторизація користувачів через supabase authentication
Можливість відгук до товару лише після покупки	Стабільна робота при збільшенні обсягу даних і кількості користувачів
Система реєстрації, авторизації та оновлення облікових даних	Зберігання особистих даних із дотриманням базових принципів безпеки

Таблиця 1.2 – Вхідні та вихідні дані

Вхідні дані	**Вихідні дані**
Категорії товарів, параметри фільтрів та їх переклади	Динамічна багатокритеріальна фільтрація з діапазонами значень і мовною підтримкою
Дані користувача (електронна пошта, ім’я, адреса тощо)	Система авторизації та редагування профілю
Історія переглядів товарів	Персоналізовані рекомендації на основі поведінки
Дії користувача (додавання до обраного, порівняння)	Вивід збережених товарів із характеристиками та зображеннями
Відгуки та рейтинг для авторизованих користувачів	Середній рейтинг товару та відображення відгуків реальних покупців
Дані форми замовлення (ПІБ, адреса, електронна пошта, список товарів)	Збереження інформації про замовлення в базу як демонстраційна функція
Пошукові запити (у тому числі з помилками чи синонімами)	Пошукова видача з урахуванням синонімів і часткових збігів

Висновок до розділу 1

У цьому розділі було проведено комплексне дослідження предметної області електронної комерції в контексті розробки бекенду інтернет-магазину комп’ютерних комплектуючих. Здійснено огляд особливостей цієї галузі, зокрема великого обсягу товарів, складної фільтрації за технічними параметрами та потреби у персоналізації функціоналу.

Аналіз конкурентних рішень (GameHall, CompX, Telemart) дозволив виявити їхні ключові переваги й недоліки - такі як обмежена персоналізація, відсутність підтримки синонімів у пошуку. Це дало змогу чітко визначити напрямки, у яких розроблюване рішення має досягти кращої ефективності.

На основі аналізу сформульовано вимоги до функціоналу системи: реалізація багатокритеріальних фільтрів із діапазонами значень, персоналізована система рекомендацій, можливість порівняння товарів, управління списком обраного, залишення відгуків і рейтингів, а також безпечна реєстрація й авторизація користувачів.

Визначено основні технічні виклики – забезпечення високої продуктивності, ефективної роботи з базою даних, збереження персональних даних, а також готовність архітектури до масштабування в майбутньому.

Отримані результати закладають основу для наступного етапу – проєктування структури системи, бази даних і логіки взаємодії між основними компонентами бекенду вебсайту.

РОЗДІЛ 2
ПРОЄКТУВАННЯ

2.1 Проєктування структури системи

На етапі проєктування структури системи було визначено ключові компоненти бекенду вебсайту інтернет-магазину комп’ютерних комплектуючих, їхню взаємодію та логіку функціонування. Особлива увага приділялась організації серверної частини, оскільки саме вона відповідає за обробку запитів, доступ до бази даних, бізнес-логіку та безпеку.

Проєктування включало розробку функціональної моделі системи, побудову сценаріїв взаємодії між користувачами та системою, а також визначення меж відповідальності кожного модуля.

Архітектура побудована за модульним принципом: кожен контролер відповідає за окремий функціональний блок, що спрощує підтримку, тестування та розширення системи в майбутньому.

Окрім загальної структури, важливим елементом є проєктування бази даних, яка забезпечує ефективне зберігання, зв’язки між сутностями та швидкий доступ до необхідної інформації. У наступному підрозділі буде детально розглянуто логічну модель бази даних та зв’язки між основними об’єктами системи.

Use case діаграма відображає основні сценарії взаємодії користувача з системою (рис. 2.1). Незареєстрований користувач має доступ до перегляду, пошуку та фільтрації товарів. Авторизований користувач отримує розширений функціонал: додавання до обраного та порівняння, оформлення замовлення, перегляд рекомендацій і залишення відгуків.

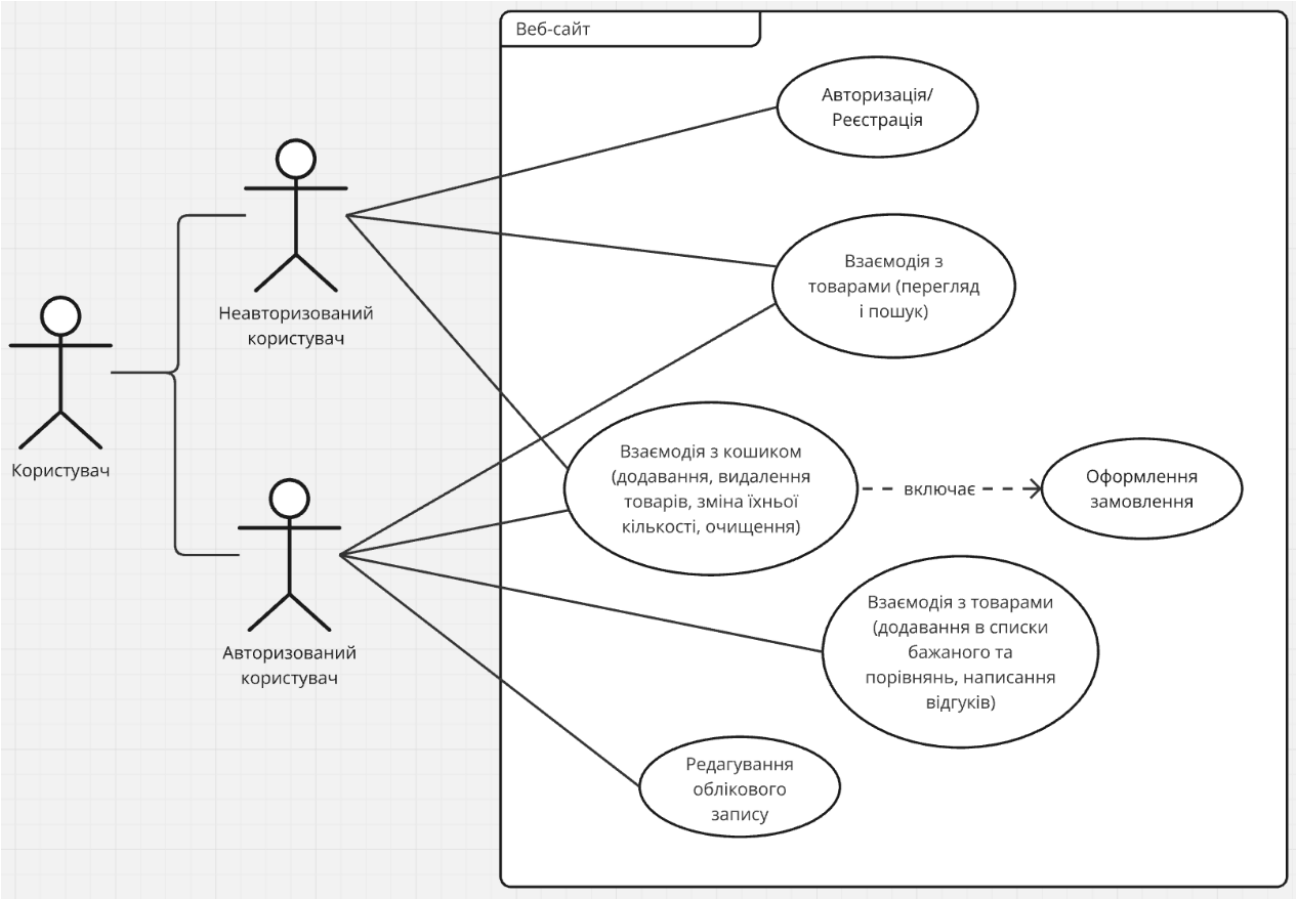


Рисунок 2.1 – Use case діаграма

Джерело: розроблено автором

Діаграма діяльності показує логіку послідовності дій користувача на сайті: від відкриття головної сторінки до оформлення замовлення (рис. 2.2). Включає фільтрацію, перегляд товарів, авторизацію, роботу з обраним, порівнянням, рекомендаціями, та залишенням відгуків після придбання товару.

Діаграма послідовності демонструє повний сценарій дій користувача: пошук і фільтрація товарів, перегляд деталей, додавання до кошика та оформлення замовлення (рис. 2.3). Вона ілюструє взаємодію між інтерфейсом, серверною частиною й базою даних Supabase.

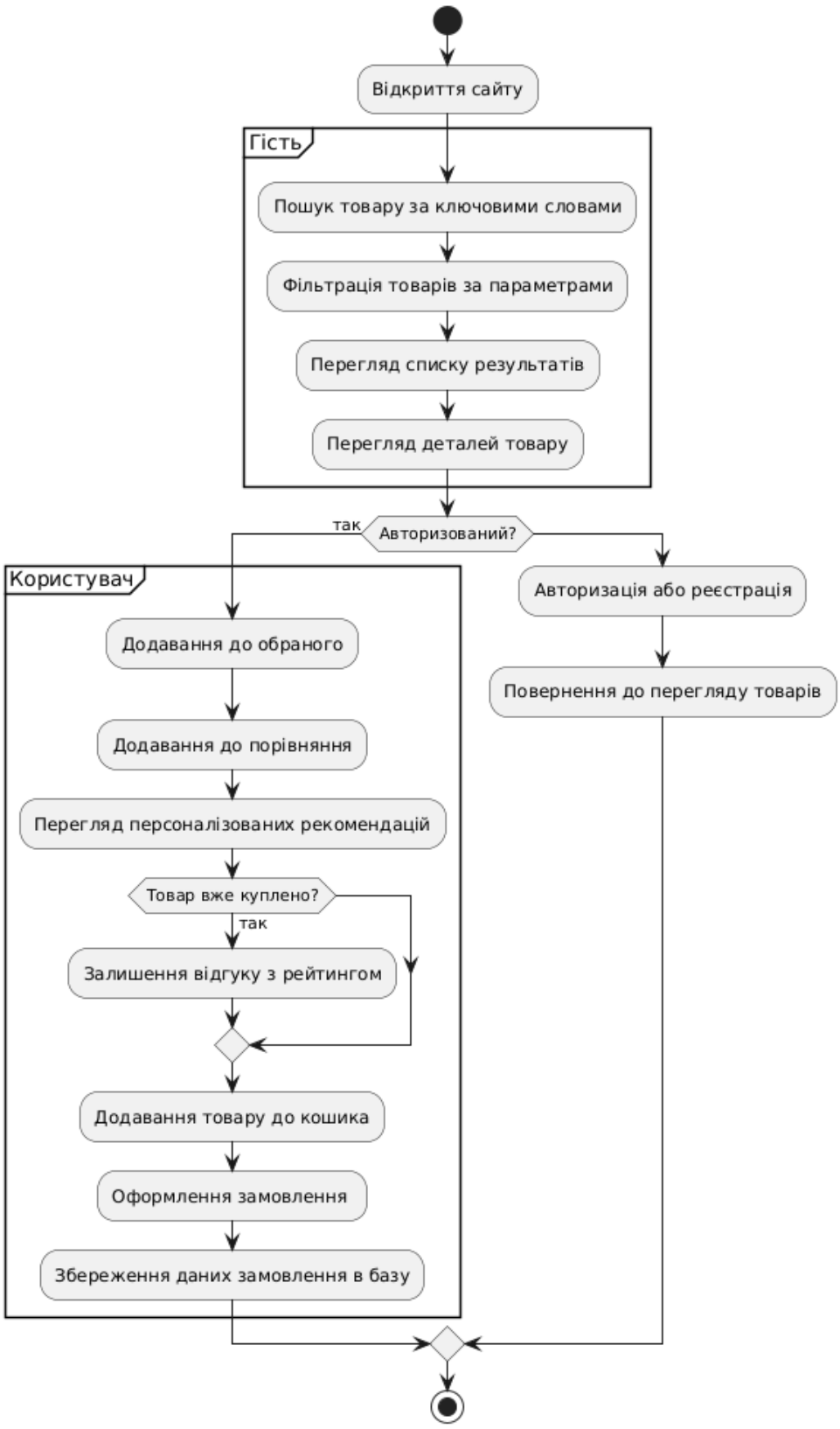


Рисунок 2.2 – Діаграма діяльності

Джерело: розроблено автором

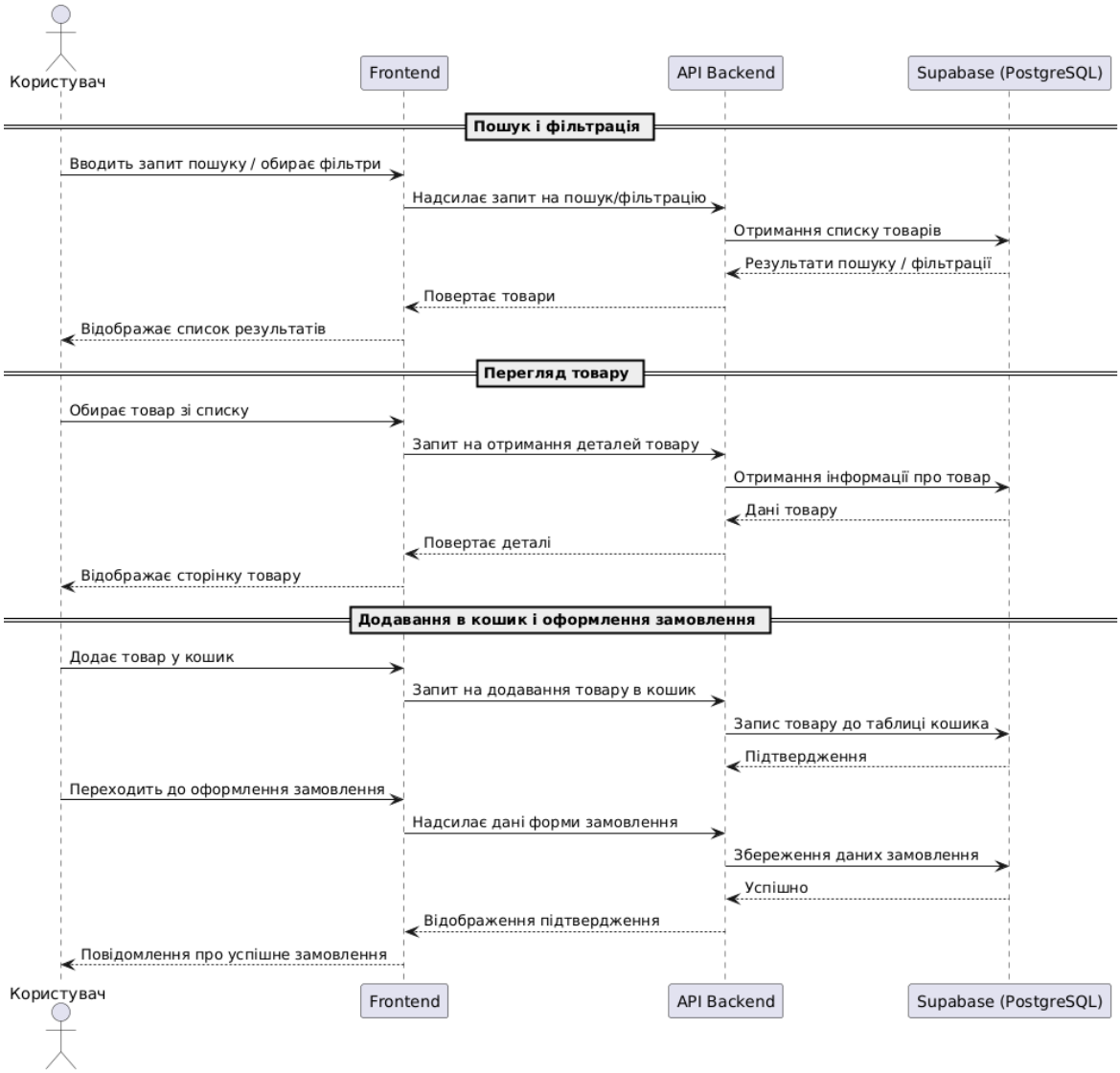


Рисунок 2.3 – Діаграма послідовності

Джерело: розроблено автором

Діаграма компонентів ілюструє архітектуру взаємодії між користувачем, візуальною складовою сайту, бекендом і базою даних (рис. 2.4). Користувач взаємодіє з інтерфейсом, який надсилає запити до серверної частини ASP.NET API. Бекенд обробляє фільтрацію товарів, рекомендації, замовлення та оновлення кошика використовуючи логіку, побудовану відповідно до REST-принципів [13], інтеграцію з Supabase SDK [14], та взаємодію з базою PostgreSQL. Всі дані зберігаються в базі даних PostgreSQL. Такий поділ забезпечує модульність, масштабованість і зручність підтримки системи.

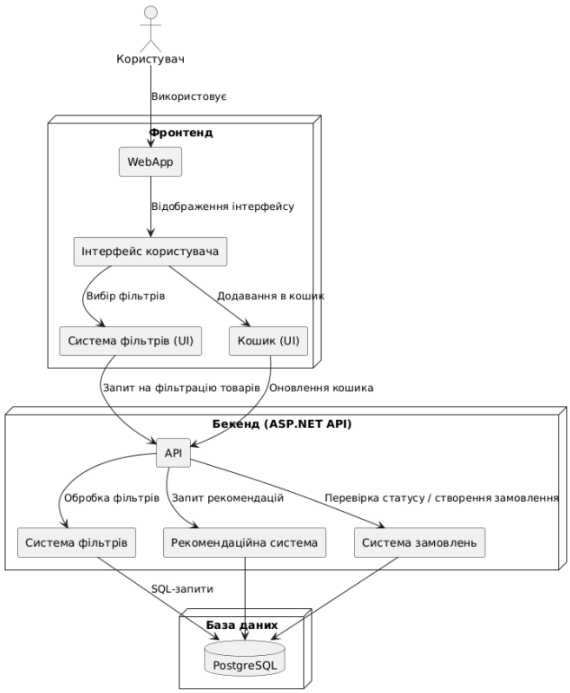


Рисунок 2.4 – Діаграма компонентів

Джерело: розроблено автором

2.2 Моделювання даних і архітектури

Моделювання даних є ключовим етапом проєктування серверної частини, оскільки визначає структуру бази даних, взаємозв’язки між сутностями та ефективність обробки інформації. У цьому підрозділі розглянуто логічну модель бази (ER-діаграму), діаграми класів та об'єктів, а також основні принципи організації даних.

Структура бази даних проєкту реалізована на основі PostgreSQL із чіткою нормалізацією таблиць. Основні сутності включають:

- користувачі (`user_data`) – зберігають дані зареєстрованих користувачів (електронна пошта, ПІБ, телефон, місто, адреса);
- товари (`products`) – описують асортимент, включаючи характеристики;
- параметри товарів (`product_parameters`, `product_parameters_int`) - зберігають технічні характеристики з підтримкою числових значень;

- категорії (`categories`) – об’єднують товари за типами (відеокарти, процесори тощо);
- замовлення (`orders`) – містять інформацію про придбані товари та контактні дані;
- обране (`wishlist`) – дає змогу зберігати товари для подальшого перегляду;
- порівняння (`comparison`) – дозволяє користувачам зіставляти характеристики декількох товарів;
- переглянуті товари (`viewed_products`) – основа для рекомендацій;
- відгуки (`product_reviews`) – містять оцінку від 1 до 5 та коментарі, доступні лише після покупки.

Всі сутності пов’язані між собою через унікальні ключі (зокрема `product_id` та `user_email`) для забезпечення цілісності та швидкого доступу до інформації. Особливу увагу приділено оптимізації запитів: використовуються індексація, типізація параметрів, перевірка унікальності та підтримка пагінації.

Дані моделювання було візуалізовано у вигляді ER-діаграми (рис. 2.5), а також діаграм класів і об’єктів, що детально описують логіку системи на рівні програмної реалізації.

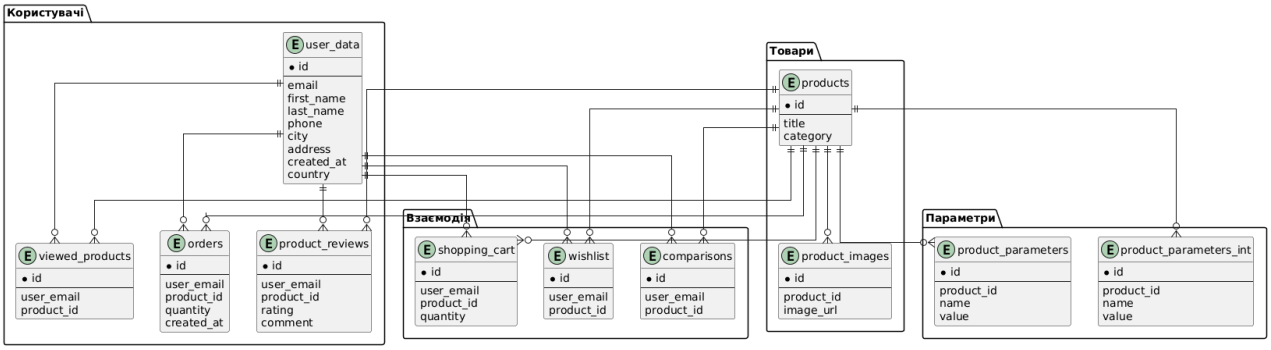


Рисунок 2.5 – Діаграма відносин сутностей (ERD)

Джерело: розроблено автором

Діаграма класів (рис. 2.6) демонструє об’єктну модель системи, що реалізована у проєкті. Вона охоплює сутності бази даних (Product, UserData, Order тощо) та логічні сервіси (SupabaseConnector, RecommendationService). Зв’язки між класами відповідають реальній структурі взаємодії в коді: RecommendationService працює через SupabaseConnector, який виконує CRUD-операції з таблицями.

Кожен клас містить ключові атрибути та методи, що реалізують бізнес-логіку. Така структура забезпечує модульність, повторне використання коду та легке розширення функціоналу.

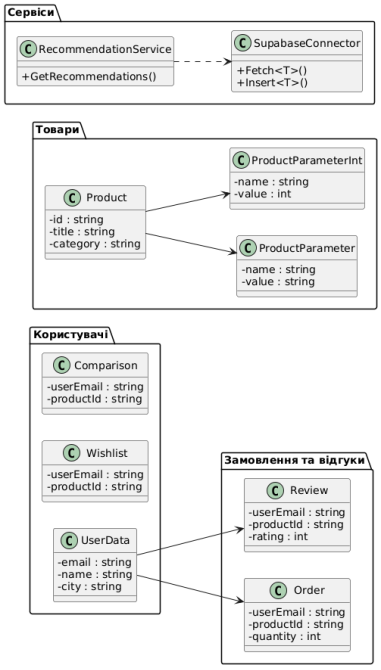


Рисунок 2.6 – Діаграма класів

Джерело: розроблено автором

На діаграмі об’єктів (рис. 2.7) ілюструється замовлення з кількома товарами, що зберігається у Supabase. Користувач ivan@example.com оформив одне замовлення, до якого входять відеокарта та процесор. Наступного дня він залишив відгук на один з товарів. Діаграма базується на структурі таблиць orders, products, product_reviews та user_data.

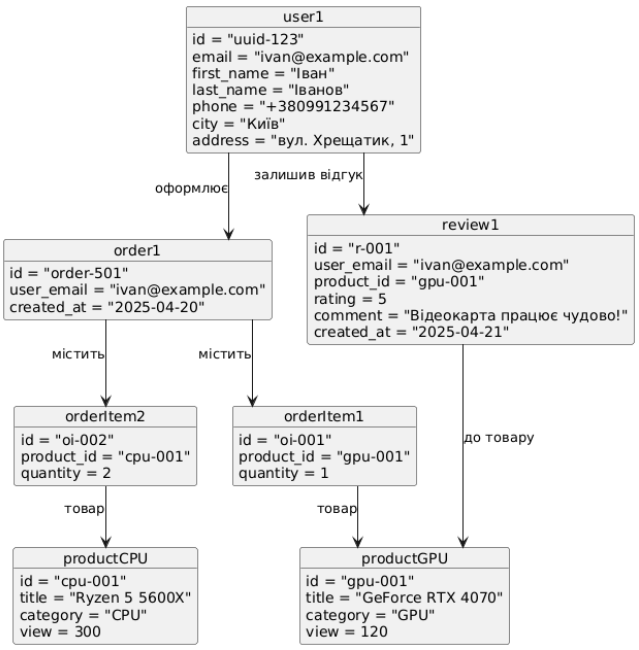


Рисунок 2.7 – Діаграма об’єктів

Джерело: розроблено автором

Діаграма комунікацій (рис. 2.8) описує типовий сценарій пошуку товарів через систему фільтрації. Користувач встановлює параметри (наприклад, бренд, ціна, модель), сайт передає запит до API, який звертається до бази даних та повертає список відповідних товарів.

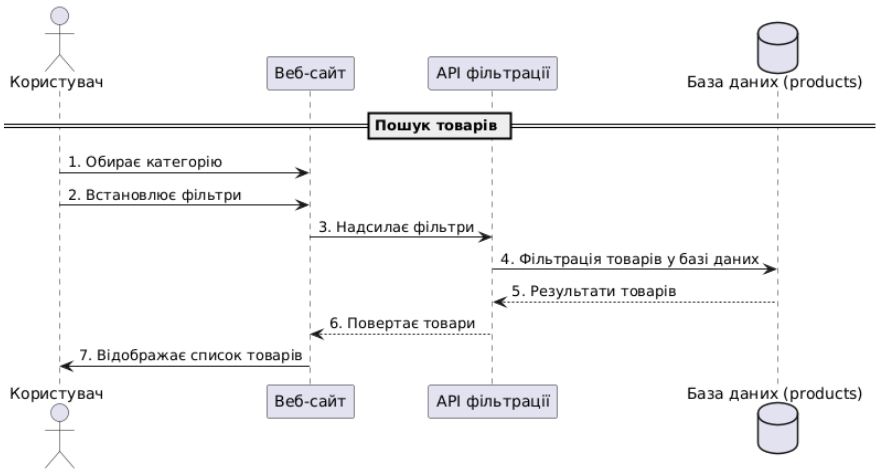


Рисунок 2.8 – Діаграма комунікації

Джерело: розроблено автором

Висновок до розділу 2

У цьому розділі було здійснено комплексне проєктування структури вебсайту інтернет-магазину комп'ютерних комплектуючих, що охоплює архітектуру системи, моделювання даних та опис ключових сценаріїв взаємодії користувачів із функціоналом сайту.

Зокрема, у розділі представлено:

- use case діаграму, що ілюструє основні сценарії використання системи;
- діаграму діяльності, яка демонструє логіку поведінки користувача під час пошуку та взаємодії з сайтом;
- діаграму послідовності, що описує етапи оформлення замовлення та фільтрації товарів;
- діаграму компонентів, яка відображає модулі бекенду та зв'язки між ними;
- діаграму класів, що моделює структуру основних об'єктів і сервісів у системі;
- ER-діаграму, яка деталізує реляційну модель бази даних Supabase;
- діаграму об'єктів, що показує реальні приклади зв'язку між екземплярами сутностей;
- діаграму комунікації, що деталізує обмін повідомленнями між модулями системи в рамках одного сценарію.

Усі діаграми побудовані відповідно до реальної структури проєкту, з урахуванням специфіки бекенд-архітектури та способу зберігання даних у Supabase. На основі проєктування сформовано узгоджену модель, яка надалі використовується як основа для реалізації бекенду, створення запитів до бази даних, розробки API та впровадження логіки обробки замовлень, фільтрації, обраного, порівняння, рекомендацій і відгуків.

Таким чином, результати цього етапу забезпечують відповідність архітектури функціональним вимогам, логічну цілісність системи та готовність до ефективного впровадження у рамках наступного етапу – розробки.

Усі UML-діаграми, подані у цьому розділі, були створені за допомогою онлайн-інструменту PlantUML [15], що дозволяє текстовим способом генерувати діаграми відповідно до стандарту UML.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Загальна архітектура реалізації

Серверна частина вебсайту реалізована на базі ASP.NET Core – сучасного фреймворку для створення масштабованих вебзастосунків. Архітектура побудована за принципами REST і Clean Architecture [16], що забезпечує розділення відповідальності між контролерами, сервісами, моделями та зовнішніми джерелами даних.

Для зберігання та обробки інформації використовується хмарна платформа Supabase, яка забезпечує доступ до реляційної бази даних PostgreSQL, а також авторизацію користувачів через вбудований модуль Supabase Auth. Обмін даними між сервером і клієнтом здійснюється у форматі JSON через HTTP-запити.

Основні компоненти системи:

- модулі авторизації та керування користувачами, реалізовані через Supabase SDK та API;
- контролери ASP.NET Core, що відповідають за фільтрацію товарів, генерацію рекомендацій, обране, порівняння, оформлення замовлень та обробку відгуків;
- сервіси взаємодії з базою даних через окремий клас SupabaseConnector, який інкапсулює запити до Supabase;
- моделі даних, що відображають структуру таблиць у базі та використовуються в обробці запитів;
- допоміжні класи, зокрема LocalizationHelper – для генерації перекладів назв характеристик та параметрів товарів.

Таким чином, архітектура проєкту (рис. 3.1) забезпечує високу гнучкість і модульність, дозволяє легко додавати нові функції, а також гарантує масштабованість і простоту підтримки в подальшому.

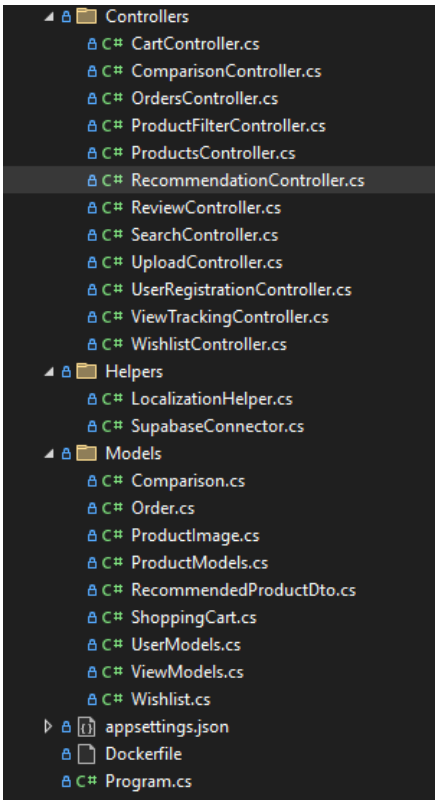


Рисунок 3.1 – Структура проєкту у Visual Studio

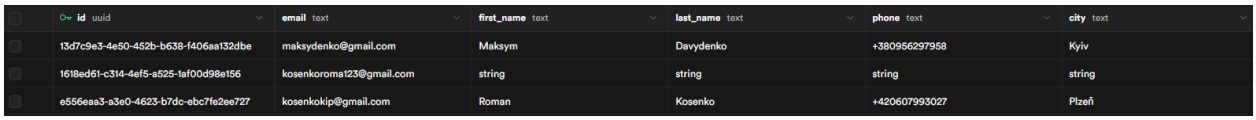
Джерело: розроблено автором

3.2 Опис основних функцій

У цьому підрозділі розглянуто реалізацію основних функціональних модулів системи: реєстрації та авторизації користувача, фільтрації товарів, системи рекомендацій, обраного, порівняння, відгуків та оформлення замовлень. Кожен функціонал реалізовано у вигляді окремого контролера ASP.NET Core, який взаємодіє з базою даних Supabase через універсальний клас SupabaseConnector.

Реєстрація та авторизація. Функціонал реєстрації реалізовано у контролері UserRegistrationController.cs. Після створення облікового запису у Supabase Auth (рис. 3.2), система зберігає додаткові дані користувача (ім’я, прізвище, телефон, місто, адреса) до таблиці `user_data` у базі даних (рис. 3.3).

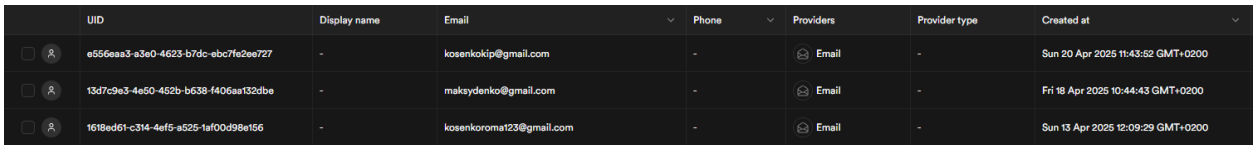
Ідентифікація користувача в системі базується на електронній пошті, що використовується як ключ у багатьох таблицях (wishlist, orders, reviews).



id uuid	email text	first_name text	last_name text	phone text	city text
13d7c9e3-4e50-452b-b638-f406aa132dbe	maksydenko@gmail.com	Maksym	Davydenko	+380956297958	Kyiv
1618ed61-c314-4ef5-a525-1af00d98e156	kosenkoroma123@gmail.com	string	string	string	string
e556eaa3-a3e0-4623-b7dc-ebc7fe2ee727	kosenkokip@gmail.com	Roman	Kosenko	+420607993027	Plzeň

Рисунок 3.2 – Список зареєстрованих користувачів у Supabase Auth

Джерело: розроблено автором



UID	Display name	Email	Phone	Providers	Provider type	Created at
e556eaa3-a3e0-4623-b7dc-ebc7fe2ee727	-	kosenkokip@gmail.com	-	Email	-	Sun 20 Apr 2025 11:43:52 GMT+0200
13d7c9e3-4e50-452b-b638-f406aa132dbe	-	maksydenko@gmail.com	-	Email	-	Fri 18 Apr 2025 10:44:43 GMT+0200
1618ed61-c314-4ef5-a525-1af00d98e156	-	kosenkoroma123@gmail.com	-	Email	-	Sun 13 Apr 2025 12:09:29 GMT+0200

Рисунок 3.3 – Таблиця user_data з розширеними даними користувачів

Джерело: розроблено автором

На рисунках 3.2 та 3.3 представлено процес зберігання даних користувачів у Supabase.

На рисунку 3.2 зображено список зареєстрованих користувачів у модулі Supabase Auth. Кожен користувач має унікальний ідентифікатор (UID), електронну адресу, провайдера авторизації та дати створення і останнього входу. Supabase автоматично обробляє реєстрацію та авторизацію через власний сервіс.

На рисунку 3.3 показано таблицю `user_data`, яка зберігає розширені дані про користувачів: ім’я, прізвище, номер телефону та місто. Ця інформація додається через окремий запит після створення облікового запису та пов'язується з UID користувача.

Такий підхід дозволяє розділяти базову авторизацію та профільні дані користувача для гнучкого керування акаунтами та їх атрибутами.

Система рекомендацій. Рекомендації реалізовано у RecommendationController.cs. Користувач отримує персоналізовані рекомендації на основі чотирьох підходів:

- схожі товари за характеристиками;
- товари з близькою ціною;
- товари, сумісні за категорією (наприклад, відеокарта + блок живлення);
- популярні товари, які переглядали інші користувачі.

Контролер взаємодіє з таблицею `viewed_products`, у якій зберігається історія переглядів, і динамічно формує запит до таблиць `products` та `product_parameters`. Усі рекомендації реалізовано через окремі методи з підтримкою пагінації.

Схожі товари за характеристиками. Для реалізації рекомендацій за схожими характеристиками система порівнює параметри товарів однієї категорії (таблиця 3.1, рис. 3.4). Чим більше збігів – тим вищий відсоток, і тим вище позиція товару в списку рекомендацій. Основна ідея полягає в тому, щоб рекомендувати користувачу ті товари, які мають найбільшу кількість спільних характеристик з товаром, який він переглядає.

У процесі аналізу система:

- отримує усі доступні характеристики базового товару (приклад наведений у таблиці 3.1 з процесором Intel Core i5-12400F);
- порівнює значення кожного параметра з аналогічними параметрами інших товарів у тій самій категорії;
- визначає кількість повних збігів за параметрами;
- обчислює відсоток схожості як відношення кількості збігів до загальної кількості характеристик.

Результати цього аналізу впливають на ранжування товарів у списку рекомендацій: чим більше параметрів співпадає – тим вище товар розташовується у списку.

Таблиця 3.1 – Приклад підбору рекомендацій за схожістю товарів

Назва параметра	Початковий товар	Перша позиція у списку рекомендації	Остання позиція у списку рекомендації
brand	Intel	Intel	AMD
chipset	Intel Core i5-12400F	Intel Core i5-13400F	AMD Ryzen 7 5700G
socket	LGA1700	LGA1700	AM4
integrated_graphics	No	No	Yes
cooler_included	Yes	Yes	Yes
memory_support	DDR4-3200, DDR5-4800	DDR4-3200, DDR5-4800	DDR4-3200
release_year	2022	2023	2021
warranty	3 years	3 years	3 years
base_clock	2500	2500	3800
boost_clock	4400	4600	4600
cores	6	10	8
threads	12	16	16
tdp	65	65	65
l3_cache	18	20	16
price	5999	6499	6599

Відповідно до таблиці 3.1, у процесора Intel Core i5-12400F існує 15 характеристик. Система у першій позиції списку рекомендації видала процесор Intel Core i5-13400. Відсоток схожості для першої позиції обраховується наступним чином: $\%\ \text{схожості} = \frac{8}{15} \times\ 100\% = \mathbf{53},(\mathbf{3})\%$, де 8 – це кількість схожих характеристик Intel Core i5-13400F з Intel Core i5-12400F,

а саме за параметрами: brand, socket, integrated_graphics, cooler_included, memory_support, tdp.

Також система видала останню позиції у списку рекомендації – AMD Ryzen 7 5700G. Відсоток схожості для цієї позиції обраховується наступним чином: $\%\ \text{схожості} = \frac{3}{15} \times\ 100\% = \mathbf{20}\%$, де 3 – це кількість схожих характеристик (warranty, cooler_included, tdp).

Таким чином, Intel Core i5-13400F отримає більш високу позицію у списку рекомендацій, оскільки він є більш релевантним вибору користувача за технічними характеристиками. Під час обчислення схожості враховуються як текстові параметри (бренд, сокет), так і числові параметри (частота, кількість ядер).

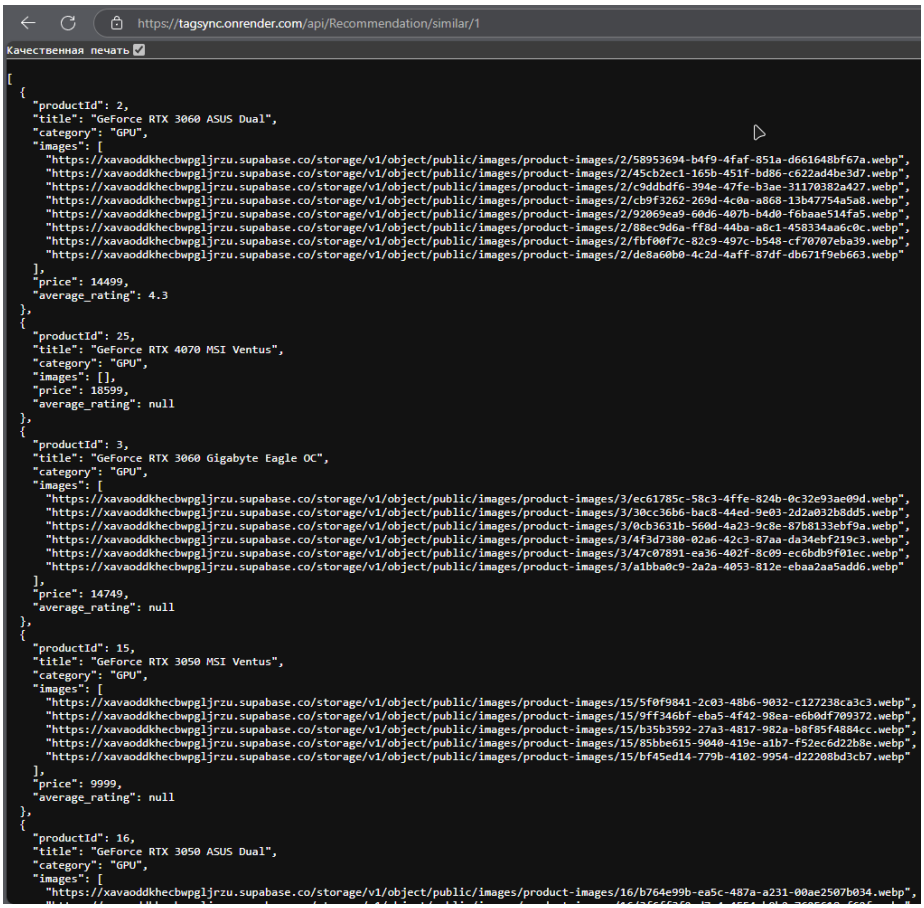


Рисунок 3.4 – Результат API-запиту на отримання схожих товарів

Джерело: розроблено автором

Сумісні товари. Завдяки рекомендаціям за сумісною категорією, до прикладу процесору та материнська плата, система допомагає користувачеві швидко й безпомилково підбирати повністю сумісні компоненти.

Розглянемо приклад з вибором материнської плати під процесор. У багатьох випадках вибір окремої комплектуючої потребує рекомендацій щодо сумісних товарів іншої категорії. Процесор має параметр «socket» – тип роз'єму, який фізично визначає, які материнські плати можна з ним використовувати.

Як працює алгоритм:

1. При отриманні запиту на сумісні товари (наприклад, /api/recommendation/compatible/{productId}), система визначає категорію вибраного товару.
2. Якщо вибрано процесор (CPU), зчитується його параметр socket.
3. Система шукає у категорії Motherboard плати, у яких параметр socket повністю збігається з сокетом процесора.
4. Формується список сумісних материнських плат, відсортований за рейтингом або популярністю.

Іншими прикладами сумісності, можуть бути:

- материнська плата та оперативна пам’ять, тобто підтримка певного типу пам'яті у плати(DDR4, DDR5);
- сумісність корпусу з довжиною відеокарти (max_gpu_length);
- сумісність корпусу з форм-фактором блоку живлення (ATX, SFX).

На рисунку 3.5 наведено приклад відповіді API на запит рекомендацій сумісних товарів. У цьому випадку використовується запит `GET /api/recommendation/compatible/59`, який повертає список материнських плат, що сумісні з процесором, ідентифікатор якого становить `productId = 59`.

У відповіді представлено товари категорії «Motherboard», підібрані за сумісністю роз’єму, чипсета або інших характеристик.

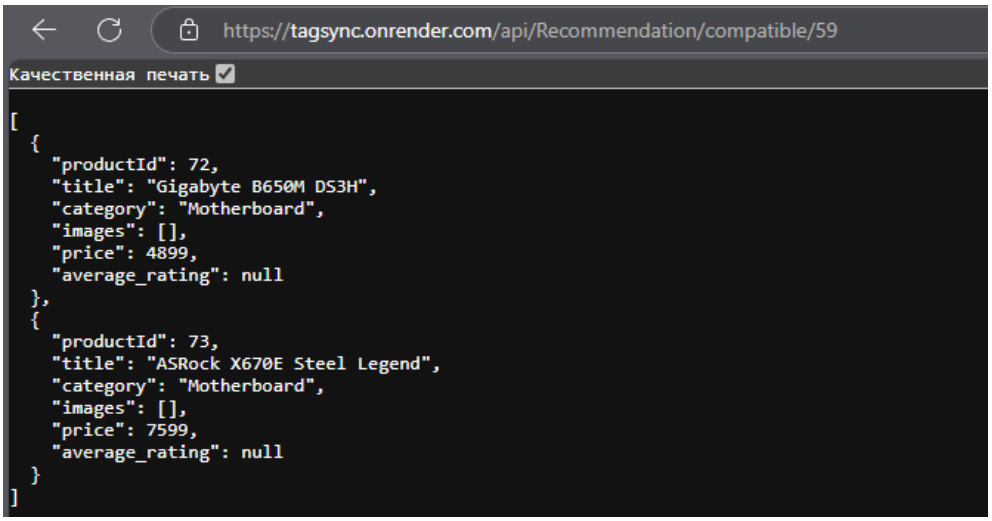


Рисунок 3.5 – Результат API-запиту на отримання сумісних товарів

Джерело: розроблено автором

Механізм рекомендацій за популярними товарами. Основою для таких рекомендацій є статистика переглядів усіх товарів користувачами сайту (рис. 3.6).

Система фіксує кожен перегляд товару та накопичує цю інформацію у базі даних. Далі за допомогою спеціального запиту визначається, які товари мають найбільшу кількість переглядів за певний період.

Такий підхід дозволяє демонструвати товари, які зараз викликають найбільший інтерес у користувачі.

id int4	title text	category text	views int4
5	GeForce RTX 4060 ASUS Dual	GPU	49
1	GeForce RTX 3060 MSI	GPU	23
6	Radeon RX 6700 XT Sapphire Pulse	GPU	16
2	GeForce RTX 3060 ASUS Dual	GPU	15

Рисунок 3.6 – Популярні товари по переглядам

Джерело: розроблено автором

Фільтрація товарів. Багатокритеріальну фільтрацію реалізовано у ProductFilterController.cs. Запити фільтрації надсилаються у форматі JSON із зазначенням категорії (рис. 3.7), параметрів і діапазонів. Контролер обробляє запит, витягує відповідні товари з таблиці `products`, а також пов'язані характеристики з `product_parameters` та `product_parameters_int`.

Користувач має змогу фільтрувати товари за декількома критеріями одночасно, зокрема:

– текстові параметри (наприклад, бренд, чіп);

– числові параметри (наприклад, обсяг пам'яті, частота, TDP).

Функція підтримує пагінацію, сортування та мовну локалізацію назв параметрів через клас LocalizationHelper.

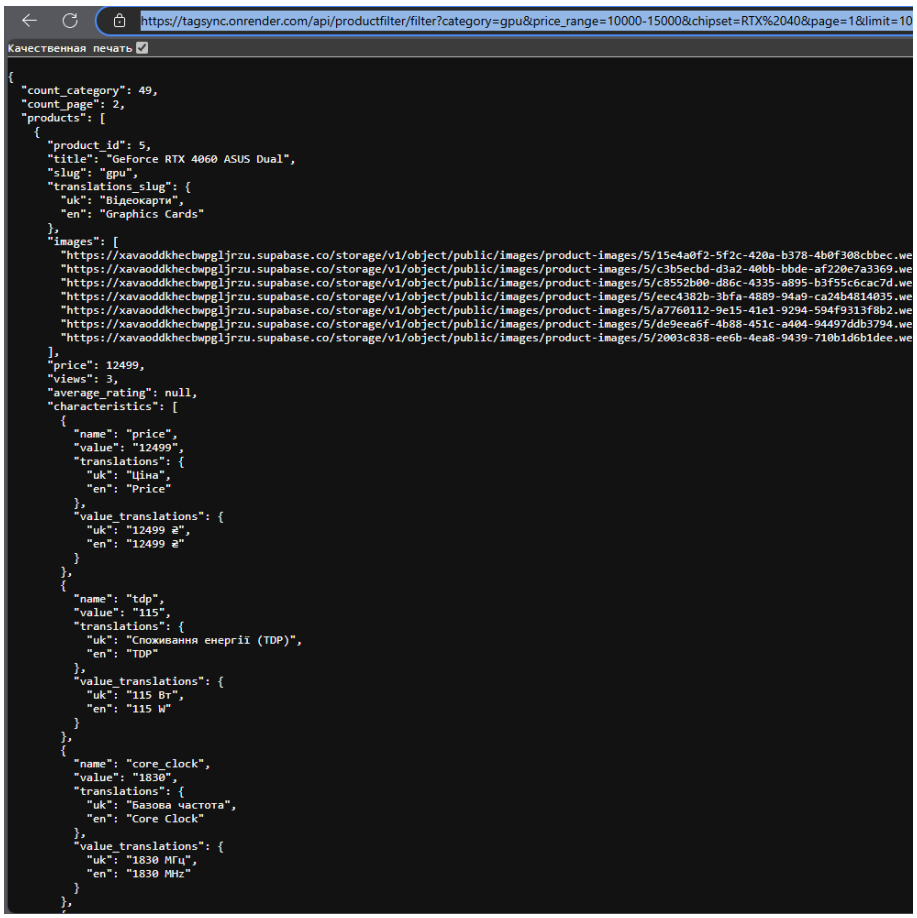


Рисунок 3.7 – Приклад відповіді API на запит фільтрації товарів

Джерело: розроблено автором

Порівняння товарів. Функція порівняння реалізована у ComparisonController.cs аналогічно до обраного. Користувач може додати декілька товарів до списку для порівняння характеристик.

Зберігання реалізовано через таблицю `comparison`, у якій зберігаються пари `user_email` та `product_id`. Запит на отримання порівнюваних товарів повертає повну інформацію про кожен товар, включно з характеристиками, перекладами назв параметрів, діапазонами значень та зображеннями.

Цей функціонал дозволяє користувачам детально порівнювати декілька моделей за всіма характеристиками одночасно.

Оформлення замовлень. Оформлення замовлення реалізовано в OrdersController.cs. Користувач заповнює контактні дані, вибирає товар та кількість, після чого замовлення зберігається до таблиці `orders`.

Основні поля: `user_email`, `product_id`, `quantity`, `full_name`, `phone`, `city`, `address`. Дані відправляються через POST-запит у форматі JSON. На даному етапі система не інтегрована з платіжними сервісами, тому функціонал замовлення реалізовано як демонстраційний.

Після збереження замовлення користувач має змогу залишити відгук лише на ті товари, які він замовив.

Список обраного (wishlist). Функція обраного реалізована у WishlistController.cs. Користувач має змогу додати будь-який товар до списку обраного. Для цього використовується таблиця `wishlist`, яка містить пари `user_email` та `product_id`.

Контролер повертає повні дані товарів із параметрами, зображеннями та локалізованими назвами характеристик.

Пошук товарів із підтримкою синонімів. Пошук реалізовано у SearchController.cs. Користувач може здійснювати пошук товарів за назвою або ключовими словами. На рисунку 3.8 наведено клас SynonymMap, у якому зберігаються відповідності між синонімами, що використовуються для покращення пошуку товарів (наприклад, «гпу» → «gpu», «материнка» → «motherboard»).

Контролер аналізує запит, перетворює його на базову категорію або ключ, після чого виконує SQL-запит до таблиці `products`. У результаті користувач отримує список релевантних товарів навіть у разі неформального запиту.

```
public static readonly Dictionary<string, string[]> SynonymMap = new()
{
    ["gpu"] = new[] {
    "graphics card", "video card", "gpu card", "відеокарта", "видюха", "відіюха", "відео карта",
    "гп", "гпу", "гпу карта", "графіка", "графічна карта", "графічний адаптер"},
    ["geforce"] = new[] {"gf", "джифорс", "джифорсе", "nvidia geforce", "nvidia"},
    ["amd"] = new[] {"radeon", "amd radeon", "амд", "радеон"},
    ["cpu"] = new[] {"processor", "процесор", "цп", "cpu chip", "central processor", "центральний процесор"},
    ["intel"] = new[] {"інтел", "intel core", "intel cpu", "intel processor"},
    ["motherboard"] = new[] {"mobo", "материнка", "материнська плата", "mother board", "mothercard", "системна плата"},
    ["ram"] = new[] {"ram", "ram memory", "оперативка", "озу", "оперативна память", "память"},
    ["ssd"] = new[] {"ssd", "накопичувач", "solid state drive", "диск", "диск ssd", "твердотільний диск"},
    ["hdd"] = new[] {"hdd", "жорсткий диск", "жорсткий", "хард", "hard disk", "hard drive"},
    ["cooler"] = new[] {"cooler", "вентилятор", "охолодження", "кулер", "fan", "cpu fan", "процесорний кулер"},
    ["case"] = new[] {"корпус", "оболочка", "кейс", "computer case", "системник", "системний блок", "комп'ютерний корпус"},
    ["psu"] = new[] {"power supply", "живлення", "пс", "псу", "живлення комп'ютера"}
};
```

Рисунок 3.8 – Фрагмент коду класу SynonymMap.cs

Джерело: розроблено автором

Відгуки та рейтинг. Система відгуків реалізована у ReviewController.cs. Користувач має змогу залишити один відгук на кожен придбаний товар, який містить текстовий коментар та рейтинг (від 1 до 5 балів).

Дані зберігаються до таблиці `product_reviews` з такими полями: `user_email`, `product_id`, `rating`, `comment`, `first_name`, `last_name`, `created_at`.

Контролер також повертає всі відгуки для конкретного товару, а також середній рейтинг, обчислений на основі всіх оцінок.

Щоб показати користувачам актуальний середній рейтинг товару, система автоматично розраховує його при отриманні інформації про товар.

Алгоритм розрахунку простий:

1. Для обраного товару система вибирає усі записи з таблиці product_reviews, де product_id = X.
2. З усіх цих записів вибираються значення поля rating.

3. Обчислюється середнє арифметичне цих значень: сума всіх оцінок ділиться на кількість відгуків.
4. Результат округлюється до одного знаку після коми для зручності відображення.

У коді це реалізовано таким чином:

```
avg_rating = g.Average(r => r.rating)
average_rating = (float)Math.Round(r.avg_rating, 1)
```

Таким чином, якщо товар має 5 відгуків з оцінками: 5, 4, 4, 5, 3, то:

$$\text{Середній рейтинг} = \frac{5+4+4+5+3}{5} = 4{,}3$$

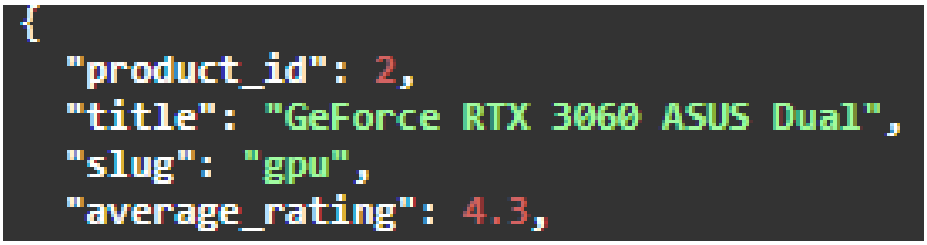


Рисунок 3.9 – Рейтинг товару у JSON

Джерело: розроблено автором

3.3 Взаємодія з Supabase

У проєкті використано хмарну платформу Supabase як основне середовище для зберігання даних і керування авторизацією. Supabase надає повноцінну backend-інфраструктуру, засновану на реляційній базі даних PostgreSQL, і дозволяє взаємодіяти з нею через REST API, JavaScript SDK або SQL-запити.

Уся взаємодія з Supabase реалізована через спеціалізований клас SupabaseConnector.cs. Цей клас інкапсулює логіку надсилання запитів (GET, POST, PUT, DELETE), авторизацію, фільтрацію за умовами, роботу з JSON-відповідями та обробку помилок. Кожен запит будується через HTTP-клієнт, а авторизація відбувається за допомогою `service_role` ключа, який дозволяє повний доступ до бази.

У рамках проєкту використовуються такі основні таблиці Supabase:

- `products` – основна таблиця з товарами;
- `product_parameters`, `product_parameters_int` – характеристики товарів;
- `user_data` – дані користувачів;
- `wishlist`, `comparison`, `shopping_cart` – дії користувачів;
- `orders` – замовлення;
- `product_reviews` – відгуки;
- `viewed_products` – переглянуті товари;
- `product_images` – масиви зображень до кожного товару.

Кожен запит до Supabase супроводжується логічною фільтрацією – наприклад, для пошуку сумісних товарів використовуються складні SQL-умови (WHERE, BETWEEN, LIKE), а також пагінація (LIMIT, OFFSET).

Платформа Supabase також використовується для авторизації користувачів через Supabase Auth. Після реєстрації створюється обліковий запис користувача, а його унікальна електронна пошта використовується як зв’язувальний ключ у багатьох таблицях.

Завдяки Supabase вдалося реалізувати централізовану базу (рис. 3.10), що підтримує одночасно авторизацію, зберігання, фільтрацію, історію дій користувача та аналітику переглядів без необхідності самостійної побудови серверної частини бази даних.

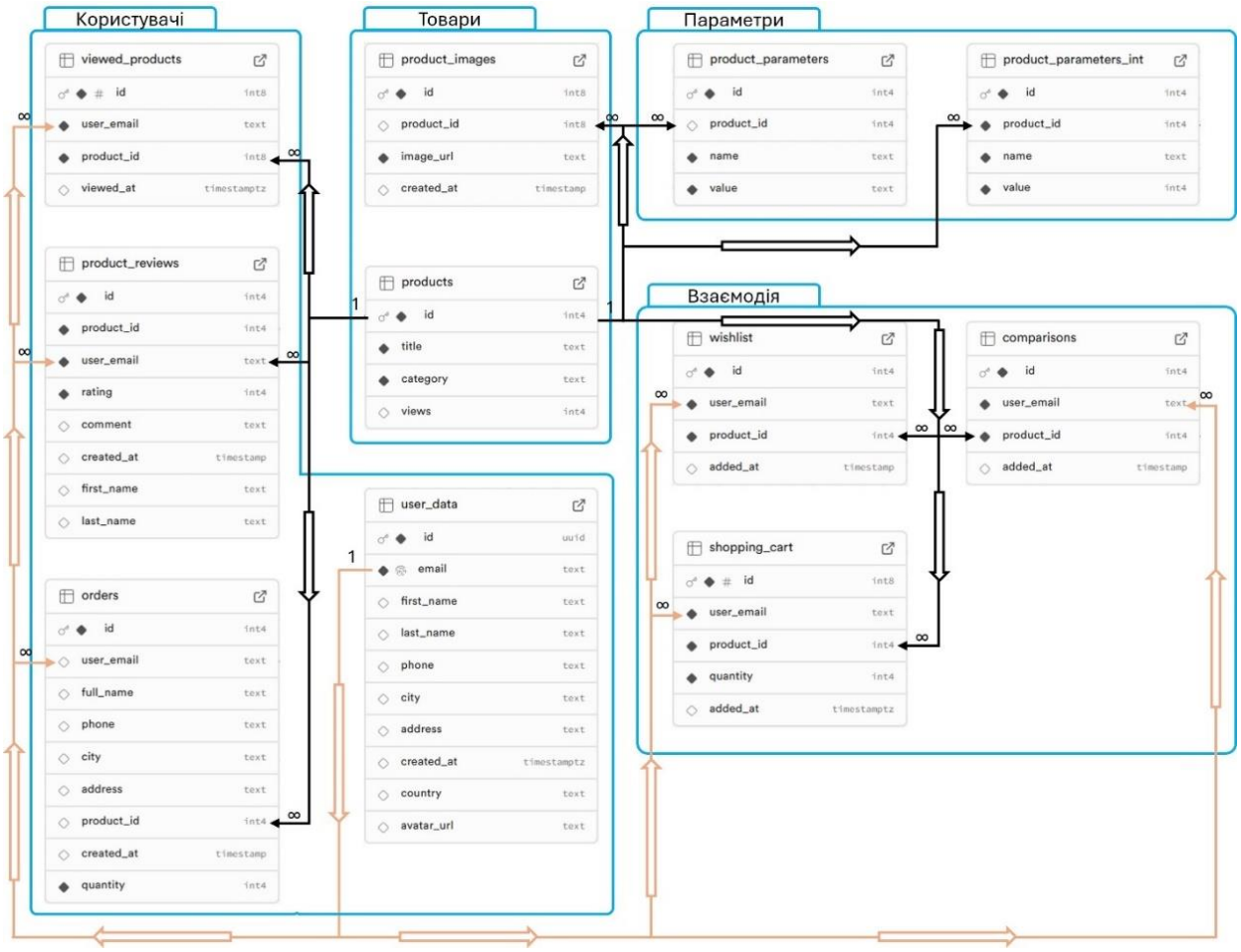


Рисунок 3.10 – Схема бази даних проєкту

Джерело: розроблено автором

3.4 Тестування функціональності системи

Після реалізації основних модулів було проведено тестування працездатності бекенд-системи шляхом ручного тестування через інструменти Postman та вбудовану систему тестування Supabase.

Тестувалися такі основні сценарії:

- успішна реєстрація та вхід користувача через Supabase Auth;
- виконання пошуку товарів із використанням синонімів;
- фільтрація товарів за кількома критеріями одночасно;
- додавання товарів до списку обраного та порівняння;
- формування замовлення із зазначенням контактних даних;
- збереження та виведення відгуків для придбаних товарів;

– правильна робота рекомендаційної системи за різними підходами.

За результатами тестування усі функції працюють коректно відповідно до сформульованих функціональних вимог. Система стабільно обробляє запити та взаємодіє з базою даних Supabase без помилок.

Висновок до розділу 3

У даному розділі було реалізовано функціональний прототип серверної частини вебсайту інтернет-магазину комп’ютерних комплектуючих. Розробка проведена на базі фреймворку ASP.NET Core з використанням хмарної платформи Supabase як середовища для зберігання даних і авторизації.

Здійснено впровадження ключових функцій системи, зокрема: авторизації та реєстрації користувачів, багатокритеріальної фільтрації товарів, персоналізованої системи рекомендацій, системи обраного, порівняння товарів, відгуків з рейтингами, оформлення замовлень і пошуку з підтримкою синонімів.

Кожна функція реалізована через окремий контролер ASP.NET Core із логічним розділенням відповідальності. Для взаємодії з базою даних PostgreSQL, яка розгорнута на платформі Supabase, використовується власний сервіс SupabaseConnector. Уся інформація зберігається в структурованих таблицях, що відповідають сутностям системи: товари, користувачі, кошик, перегляди, замовлення тощо.

Таким чином, розроблена реалізація відповідає усім функціональним вимогам, визначеним у першому розділі, забезпечує ефективну та масштабовану серверну логіку та є функціональним прототипом серверної частини сучасного онлайн-магазину.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було реалізовано серверну частину (backend) вебсайту інтернет-магазину комп'ютерних комплектуючих, яка відповідає сучасним вимогам до ефективності, масштабованості та зручності використання.

На етапі аналізу предметної області було досліджено особливості ринку комп'ютерних комплектуючих, а також проаналізовано існуючі аналоги інтернет-магазинів, що дозволило виявити типові функціональні обмеження та сформулювати вимоги до власної системи.

Під час проєктування було розроблено архітектуру майбутнього застосунку, виконано моделювання структури бази даних та взаємодії між компонентами системи. Для цього було створено use case діаграму, діаграми класів, послідовності, компонентів, об'єктів, ER-діаграму та інші.

У ході реалізації проєкту було створено REST API на основі ASP.NET Core, а також реалізовано збереження та обробку даних за допомогою платформи Supabase. Основний функціонал включає багатокритеріальну фільтрацію товарів, рекомендаційну систему, систему обраного, порівняння, відгуки з рейтингами, оформлення замовлень та пошук з підтримкою синонімів. Усі дані зберігаються у структурованій реляційній базі даних PostgreSQL, а авторизація користувачів здійснюється через Supabase Auth.

Розроблений бекенд є функціональним прототипом серверної частини інтернет-магазину з можливістю подальшого масштабування та інтеграції з візуальною частиною сайту. Робота має практичну цінність як приклад побудови сучасного рішення щодо електронної комерції з урахуванням актуальних потреб користувачів та вимог до архітектури, безпеки та продуктивності.

Протягом навчання проводились дослідження, зокрема була проведена апробація дослідження на Міжнародній V Науковій конференції «Сучасний менеджмент організації: витоки, реалії та перспективи розвитку» [17].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Блог хостингової компанії HOSTiQ.ua. Що таке API: простими словами про складне [Електронний ресурс]. – URL:https://hostiq.ua/blog/ukr/what-is-api/ (дата звернення: 21.04.2025).

2. Brainlab. Back-end розробка [Електронний ресурс]. – URL:https://brainlab.com.ua/uk/blog-uk/back-end-rozrobka (дата звернення: 21.04.2025).

3. HOSTiQ.ua Wiki. Що таке кеш і навіщо він потрібний [Електронний ресурс]. – URL: https://hostiq.ua/wiki/ukr/cache/ (дата звернення: 21.04.2025).

4. Microsoft. ASP.NET Core Documentation [Електронний ресурс]. – URL:https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0 (дата звернення: 21.04.2025).

5. Microsoft. Entity Framework Core [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 21.04.2025).

6. Supabase Documentation [Електронний ресурс]. URL: https://supabase.com/docs (дата звернення: 21.04.2025).

7. PostgreSQL Documentation [Електронний ресурс]. URL: https://www.postgresql.org/docs/ (дата звернення: 10.04.2025).

8. Fondy [Електронний ресурс]. URL: https://fondy.ua/uk/knowledge/online-store-creation (дата звернення: 10.03.2025).

9. GitHub – Acid330/tagsync: Репозиторій проекту інтернет-магазину комп'ютерних комплектуючих [Електронний ресурс]. URL: https://github.com/Acid330/tagsync (дата звернення: 21.04.2025).

10. GameHall [Електронний ресурс]. URL: https://gamehall.com.ua (дата звернення: 10.03.2025).

11. CompX [Електронний ресурс]. URL: https://compx.ua (дата звернення: 10.03.2025).

12. Telemart [Електронний ресурс]. URL: https://telemart.ua/ua/ (дата звернення: 10.03.2025).

13. Fielding, Roy. Architectural Styles and the Design of Network-based Software Architectures [Електронний ресурс]. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата звернення: 21.04.2025).

14. Supabase SDK Documentation [Електронний ресурс]. URL: https://supabase.com/docs/reference/javascript/introduction (дата звернення: 21.04.2025).

15. PlantUML – Open-source tool for generating UML diagrams from plain text [Електронний ресурс]. URL: https://plantuml.com/ (дата звернення: 21.04.2025).

16. Uncle Bob (Robert C. Martin). The Clean Architecture [Електронний ресурс]. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (дата звернення: 22.04.2025).

17. Фещенко М.І., Давиденко М.О., Косенко С.М. АНАЛІЗ «КРАНЧУ» ЯК ЯВИЩА В ІНДУСТРІЇ ВІДЕОІГОР // «КРОК» Конференції, Сучасний менеджмент організації: витоки, реалії та перспективи розвитку 2025 – URL: https://conf.krok.edu.ua/MMO/MMO-2025/paper/view/2856 (дата звернення: 23.04.2025).

ДОДАТОК А
КОД СИСТЕМИ РЕКОМЕНДАЦІЙ

```
using Microsoft.AspNetCore.Mvc;
using tagsync.Helpers;
using tagsync.Models;

namespace tagsync.Controllers;

[ApiController]
[Route("api/productfilter")]
public class ProductFilterController : ControllerBase
{
    [HttpGet("filter")]
    public async Task<IActionResult> FilterProducts()
    {
        var queryParams = HttpContext.Request.Query;
        var filters = new Dictionary<string, HashSet<string>>();
        var rangeFilters = new Dictionary<string, List<(int from, int to)>>();

        int page = queryParams.TryGetValue("page", out var pgVal) && int.TryParse(pgVal, out int pg) ? Math.Max(pg, 1) : 1;
        int limit = queryParams.TryGetValue("limit", out var limVal) && int.TryParse(limVal, out int lim) ? Math.Max(lim, 1) : 10;
        string sortBy = queryParams.TryGetValue("sort_by", out var sb) ? sb.ToString().ToLower() : "id";
        string sortOrder = queryParams.TryGetValue("sort_order", out var so) ? so.ToString().ToLower() : "asc";
        string category = queryParams.TryGetValue("category", out var catVal) ? catVal.ToString().ToLower() : "";

        foreach (var param in queryParams)
        {
            var key = param.Key.ToLower();
            var values = param.Value.ToString().Split(',', StringSplitOptions.RemoveEmptyEntries | StringSplitOptions.TrimEntries);

            if (key.EndsWith("_range"))
            {
                var name = key.Replace("_range", "");
                rangeFilters[name] = new();
                foreach (var range in values)
                {
                    var bounds = range.Split('-');
                    if (bounds.Length == 2 && int.TryParse(bounds[0], out int from) && int.TryParse(bounds[1], out int to))
                        rangeFilters[name].Add((from, to));
                }
            }
            else if (key is not ("sort_by" or "sort_order" or "page" or "limit" or "category"))
            {
                filters[key] = values.Select(v => v.ToLower()).ToHashSet();
            }
        }

        var allParams = await SupabaseConnector.Client.From<ProductParameter>().Get();
```

```
        var allParamsInt = await SupabaseConnector.Client.From<ProductParameterInt>().Get();
        var allProducts = await SupabaseConnector.Client.From<Product>().Get();
        var allReviews = await SupabaseConnector.Client.From<ProductReview>().Get();
        var productImages = await SupabaseConnector.Client.From<ProductImage>().Get();

        var filteredProducts = allProducts.Models
            .Where(p => p.Category?.ToLower() == category)
            .ToList();

        var matchedProductIds = filteredProducts
            .Where(p =>
            {
                var productParams = allParams.Models.Where(m => m.product_id == p.Id).ToList();
                foreach (var filter in filters)
                {
                    var match = productParams.Any(pp =>
                        pp.Name.Equals(filter.Key, StringComparison.OrdinalIgnoreCase) &&
                        filter.Value.Any(val => pp.Value.ToLower().Contains(val)));
                    if (!match) return false;
                }
                return true;
            })
            .Select(p => p.Id)
            .ToHashSet();

        var numericMatchedIds = new HashSet<int>();
        foreach (var product in filteredProducts)
        {
            int id = product.Id;
            bool include = true;

            foreach (var rangeFilter in rangeFilters)
            {
                var param = allParamsInt.Models.FirstOrDefault(p => p.product_id == id && p.Name.ToLower() == rangeFilter.Key);
                if (param == null || !int.TryParse(param.Value.ToString(), out int val))
                {
                    include = false;
                    break;
                }

                bool inAnyRange = rangeFilter.Value.Any(r => val >= r.from && val <= r.to);
                if (!inAnyRange)
                {
                    include = false;
                    break;
                }
            }

            if (include)
                numericMatchedIds.Add(id);
        }
```

```
        if (rangeFilters.Count > 0)
            matchedProductIds = matchedProductIds.Intersect(numericMatchedIds).ToHashSet();

        var filtered = filteredProducts
            .Where(p => matchedProductIds.Contains(p.Id))
            .ToList();

        var results = filtered
            .Select(p => new
            {
                product = p,
                price = allParamsInt.Models.FirstOrDefault(x => x.product_id == p.Id && x.Name == "price")?.Value
            });

        results = sortBy switch
        {
            "price" => sortOrder == "desc" ? results.OrderByDescending(x => x.price) : results.OrderBy(x => x.price),
            "title" => sortOrder == "desc" ? results.OrderByDescending(x => x.product.Title) : results.OrderBy(x => x.product.Title),
            "views" => sortOrder == "desc" ? results.OrderByDescending(x => x.product.Views) : results.OrderBy(x => x.product.Views),
            _ => sortOrder == "desc" ? results.OrderByDescending(x => x.product.Id) : results.OrderBy(x => x.product.Id)
        };

        int totalCount = results.Count();

        results = results.Skip((page - 1) * limit).Take(limit);

        var response = results.Select(r =>
        {
            var ratings = allReviews.Models
                .Where(rvw => rvw.product_id == r.product.Id)
                .Select(rvw => rvw.average_rating)
                .ToList();

            float? averageRating = ratings.Count == 0
                ? null
                : (float)Math.Round(ratings.Average(), 1);

            return new
            {
                product_id = r.product.Id,
                title = r.product.Title,
                slug = r.product.Category?.ToLower(),
                translations_slug = LocalizationHelper.CategoryTranslations.TryGetValue(r.product.Category?.ToLower() ?? "", out var slugTr) ? slugTr : null,
                images = productImages.Models
                    .Where(img => img.product_id == r.product.Id)
                    .Select(img => img.ImageUrl)
                    .ToList(),
                price = r.price,
                views = r.product.Views,
                average_rating = averageRating,
                characteristics = allParamsInt.Models
```

```
                        .Where(param => param.product_id == r.product.Id)
                        .Select(param =>
                        {
                            var name = param.Name.ToLower();
                            var value = param.Value.ToString();
                            var translations = LocalizationHelper.ParameterTranslations.TryGetValue(name, out var tr) ? tr : null;

                            var dict = new Dictionary<string, object?>
                            {
                            { "name", param.Name },
                            { "value", value },
                            { "translations", translations }
                            };

                            if (LocalizationHelper.ValueSuffixes.TryGetValue(name, out var suffix))
                            {
                                dict["value_translations"] = new Dictionary<string, string>
                                {
                                { "uk", $"{value} {suffix.uk}" },
                                { "en", $"{value} {suffix.en}" }
                                };
                            }

                            return dict;
                        })
                        .Concat(
                            allParams.Models
                                .Where(param => param.product_id == r.product.Id)
                                .Select(param =>
                                {
                                    var name = param.Name.ToLower();
                                    var translations = LocalizationHelper.ParameterTranslations.TryGetValue(name, out var tr) ? tr : null;
                                    return new Dictionary<string, object?>
                                    {
                                    { "name", param.Name },
                                    { "value", param.Value },
                                    { "translations", translations }
                                    };
                                })
                        ).ToList()
                };
            });

            return Ok(new
            {
                count_category = filteredProducts.Count,
                count_page = response.Count(),
                products = response
            });
        }
    }
```