

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Реалізація рекомендаційної системи для
платформи електронної комерції»**

Виконав студент групи ІПЗмп-23-1
ГРІШИН Ярослав Олегович

Керівник роботи:
ЛОКТИКОВА Тамара Миколаївна

Рецензент:
ВАКАЛЮК Тетяна Анатоліївна

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
 ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
 ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
 КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
 програмного забезпечення

Тетяна ВАКАЛЮК

«14» лютого 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу

Здобувач вищої освіти: **ГРІШИН Ярослав Олегович**

Керівник роботи: **ЛОКТИКОВА Тамара Миколаївна**

Тема роботи: **«Реалізація рекомендаційної системи для платформи електронної комерції»**, затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р. № 60/с.**

Вихідні дані для роботи: об'єктом дослідження є технології розробки веб-додатків для створення рекомендаційних систем у сфері електронної комерції, включаючи методи обробки даних про товари, аналіз поведінки користувачів і персоналізацію пропозицій.

Предметом дослідження є моделі, методи та алгоритми розробки веб-додатку для генерації релевантних товарних рекомендацій, їхньої оптимізації та надання персоналізованих пропозицій користувачам з метою підвищення ефективності продажів і задоволеності клієнтів.

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Локтікова Т.М.	14.02.2025р.	14.02.2025р.
2	Локтікова Т.М.	14.02.2025р.	14.02.2025р.
3	Локтікова Т.М.	14.02.2025р.	14.02.2025р.

РЕФЕРАТ

Метою кваліфікаційної роботи є розробка рекомендаційної системи для платформи електронної комерції, спрямованої на підвищення конверсії та задоволеності користувачів за рахунок персоналізації виведення товарів. Об'єктом дослідження є сучасні технології побудови рекомендаційних систем, зокрема методи колаборативної фільтрації, контент-базованого підходу та аналізу поведінкових даних. Предметом дослідження виступають алгоритми обробки даних про перегляди товарів, їх категоріальну приналежність та популярність з метою генерації релевантних рекомендацій.

Теоретична значущість роботи полягає у дослідженні принципів побудови рекомендаційних систем для електронної комерції, аналізі методів оцінки ефективності таких систем та оптимізації їх роботи на основі даних про взаємодію користувачів. Розглянуто підходи до інтеграції зовнішніх сервісів аналітики, таких як Google Analytics, для покращення якості рекомендацій.

Практична цінність роботи полягає у впровадженні функціональної рекомендаційної системи в реальний інтернет-магазин. Система дозволяє автоматично підбирати товари на основі поточного перегляду, категорійної схожості та статистики популярності, що сприяє збільшенню середнього чека та утриманню клієнтів. Додатково реалізовано механізм адаптації рекомендацій на основі зворотного зв'язку від користувачів, що підвищує точність підбору.

Магістерська робота складається з вступу, трьох розділів, висновків, списку використаних джерел (30 позицій) та додатка. Основна частина містить 72 сторінки, загальний обсяг роботи становить 86 сторінок. До тексту включено 4 таблиці та 25 ілюстрацій, які візуалізують ключові аспекти дослідження.

КЛЮЧОВІ СЛОВА: РЕКОМЕНДАЦІЙНА СИСТЕМА, ЕЛЕКТРОННА КОМЕРЦІЯ, ПЕРСОНАЛІЗАЦІЯ, ПОВЕДІНКОВІ ДАНІ, КОНВЕРСІЯ.

ABSTRACT

The purpose of the qualification work is to develop a recommendation system for an e-commerce platform aimed at increasing conversion rates and user satisfaction through personalized product suggestions. The object of research is modern technologies for building recommendation systems, including collaborative filtering, content-based approaches, and behavioral data analysis. The subject of the study focuses on algorithms for processing product view data, category affiliation, and popularity metrics to generate relevant recommendations.

The theoretical significance of the work lies in exploring the principles of recommendation systems for e-commerce, analyzing methods for evaluating their effectiveness, and optimizing their performance based on user interaction data. Approaches to integrating external analytics services, such as Google Analytics, to enhance recommendation quality are also examined.

The practical value of the work is demonstrated through the implementation of a functional recommendation system in a real-world online store. The system automatically selects products based on current views, category similarity, and popularity statistics, contributing to higher average order values and customer retention. Additionally, a feedback-driven adaptation mechanism improves recommendation accuracy.

The master's thesis consists of an introduction, three chapters, conclusions, a list of references (30 items), and appendice. The main text comprises 72 pages, with a total volume of 86 pages. The work includes 4 tables and 25 illustrations to visualize key research aspects.

KEYWORDS: RECOMMENDATION SYSTEM, E-COMMERCE, PERSONALIZATION, BEHAVIORAL DATA, CONVERSION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБГРУНТУВАННЯ АКТУАЛЬНОСТІ ТЕМИ РОБОТИ	9
1.1 Сучасні методи побудови рекомендаційних систем	9
1.2 Аналіз предметної області рекомендаційних систем	12
1.3 Аналіз аналогів програмного продукту	14
1.4 Обґрунтування вибору інструментальних засобів	18
Висновки до першого розділу.....	23
РОЗДІЛ 2. ПРОЄКТУВАННЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ДЛЯ ПЛАТФОРМИ ЕЛЕКТРОННОЇ КОМЕРЦІЇ	24
2.1 Визначення варіантів використання.....	24
2.2 Архітектура системи.....	26
2.3 Алгоритми рекомендацій	29
2.4 Реалізація бази даних.....	35
2.5 Інтеграція з зовнішніми сервісами	42
Висновки до другого розділу	46
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБДОДАТКОМ.....	48
3.1 Створення програмного додатку	48
3.2 Структура інтерфейсу вебдодатку.....	55
3.3 Тестування роботи програмного комплексу	61
Висновки до третього розділу.....	64
ВИСНОВКИ	66
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	69
ДОДАТКИ	72

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface (інтерфейс програмування додатків)

CMS – Content Management System (система управління контентом)

CTR – Click-Through Rate (коефіцієнт клікабельності)

DB – Database (база даних)

GA – Google Analytics (Гугл Аналітика)

HTTP – Hypertext Transfer Protocol (протокол передачі гіпертексту)

JSON – JavaScript Object Notation (формат обміну даними)

ML – Machine Learning (машинне навчання)

NoSQL – Not Only SQL (не реляційна база даних)

TF-IDF – Term Frequency-Inverse Document Frequency (частота терміну – зворотна частота документа)

UI – User Interface (користувацький інтерфейс)

ВСТУП

Актуальність теми. Сучасний світ електронної комерції характеризується стрімким зростанням конкуренції та підвищеними вимогами до персоналізації взаємодії з користувачами. У цих умовах особливої актуальності набуває розробка інтелектуальних рекомендаційних систем, здатних аналізувати поведінку споживачів та пропонувати їм релевантні товарні пропозиції. Тема даної роботи – "Реалізація рекомендаційної системи для платформи електронної комерції" – є надзвичайно важливою, оскільки такі системи безпосередньо впливають на ключові показники ефективності інтернет-магазинів, такі як конверсія, середній чек та утримання клієнтів.

Актуальність дослідження підкріплюється тим, що сучасні покупці стикаються з надлишком інформації та варіантів вибору, що ускладнює процес прийняття рішень. Ефективна рекомендаційна система дозволяє вирішити цю проблему, пропонуючи персоналізований підхід на основі аналізу попередніх переглядів, категорійної приналежності товарів та їхньої популярності серед інших користувачів.

Метою даної роботи є розробка та впровадження веб-орієнтованої рекомендаційної системи, здатної автоматизувати процес підбору товарів для кожного конкретного користувача. Для досягнення цієї мети було поставлено **наступні завдання:**

- провести аналіз існуючих підходів до побудови рекомендаційних систем у електронній комерції;
- дослідити методи обробки поведінкових даних та їх інтеграції з системами аналітики;
- розробити алгоритми підбору рекомендацій на основі категорійної схожості та популярності товарів;
- реалізувати веб-додаток з функцією динамічного оновлення рекомендацій на основі зворотного зв'язку;

Об'єктом дослідження виступають сучасні технології розробки рекомендаційних систем для платформ електронної комерції.

Предметом дослідження є моделі, методи та алгоритми розробки рекомендаційної системи для платформи електронної комерції.

У кваліфікаційній роботі використано такі методи: аналіз даних – для вивчення шаблонів поведінки користувачів та визначення ключових факторів впливу; математичне моделювання – для побудови алгоритмів оцінки схожості товарів та прогнозування їхньої релевантності; веб-розробка – для створення інтерактивного інтерфейсу системи з використанням сучасних технологій Next.js та MongoDB.

Наукова новизна роботи полягає в розробці гібридного підходу до генерації рекомендацій, який поєднує категорійний аналіз, дані про популярність товарів з Google Analytics та механізми адаптації на основі зворотного зв'язку. Особливістю системи є її здатність динамічно оновлювати рекомендації при отриманні сигналів від користувачів, що підвищує точність підбору.

Практичне значення дослідження полягає у створенні функціональної системи, яка може бути інтегрована в існуючі платформи електронної комерції. Реалізований додаток дозволяє значно підвищити ефективність товарних рекомендацій, що веде до зростання ключових показників бізнесу. Додатковою перевагою є модульна архітектура системи, яка дозволяє легко масштабувати її функціонал.

Особистий внесок автора полягає у розробці алгоритму комбінації категорійного підходу з даними про популярність товарів, а також у реалізації механізму адаптації рекомендацій на основі дій користувачів. Всі етапи дослідження, включаючи проектування архітектури, написання коду та тестування, були виконані автором самостійно.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБГРУНТУВАННЯ АКТУАЛЬНОСТІ ТЕМИ РОБОТИ

1.1 Сучасні методи побудови рекомендаційних систем

Сучасний ринок електронної комерції характеризується жорсткою конкуренцією та постійним зростанням очікувань клієнтів щодо персоналізації сервісу. У цих умовах рекомендаційні системи стали невід'ємним інструментом для підвищення конверсії та утримання клієнтів. Вони пройшли значний шлях еволюції - від простих правил "купають разом" до складних алгоритмів машинного навчання, здатних адаптуватися до індивідуальних уподобань кожного користувача.

Основним завданням сучасних рекомендаційних систем є аналіз великих масивів даних про поведінку користувачів, характеристик товарів та контексту взаємодії. Цей аналіз проводиться за допомогою трьох ключових підходів, кожен з яких має свої переваги та обмеження.

Колаборативна фільтрація залишається одним з найпопулярніших методів [7], особливо для великих інтернет-магазинів з численною базою клієнтів. Вона працює з історичними даними про взаємодії користувачів з товарами, використовуючи методи матричної факторизації. Суть цього підходу полягає у розкладанні великої розрідженої матриці рейтингів на добуток двох матриць меншої розмірності - користувацьких та товарних факторів. Це дозволяє виявити приховані зв'язки між користувачами та продуктами, які не очевидні при поверхневому аналізі. Оптимізація параметрів моделі зазвичай здійснюється за допомогою стохастичного градієнтного спуску з L2-регуляризацією, що запобігає перенавчанню та покращує узагальнюючу здатність системи. Важливою перевагою колаборативної фільтрації є її здатність виявляти неочевидні асоціації між товарами, однак вона страждає від проблеми "холодного старту" для нових товарів або користувачів.

Контент-орієнтовані системи пропонують альтернативний підхід, заснований на аналізі властивостей товарів. Вони розглядають такі атрибути як категорія, ціна, технічні параметри, опис та інші характеристики продукту. Для обробки цих даних використовується TF-IDF перетворення [5], яке дозволяє оцінити важливість кожного атрибута в контексті всього каталогу товарів. Обчислення схожості між товарами зазвичай проводиться за допомогою косинусної міри або інших метрик подібності у векторному просторі ознак. Такий підхід особливо ефективний для нових товарів, які ще не мають достатньої кількості взаємодій з користувачами, а також для нішевих категорій продуктів. Однак він обмежений якістю та повнотою метаданих про товари і не враховує соціальні аспекти вибору.

Найбільш просунутим напрямком є застосування методів глибокого навчання, які дозволяють створювати складні моделі, що враховують як поведінкові дані, так і характеристики продуктів. Серед найпопулярніших архітектур можна виділити Wide & Deep та Two-Tower моделі. Wide & Deep архітектура поєднує широку лінійну модель для запам'ятовування часто зустрічаючихся шаблонів із глибокою нейронною мережею для узагальнення. Two-Tower архітектура використовує окремі підмережі для обробки даних про користувачів та товари, що дозволяє ефективно обчислювати схожість між ними у спільному векторному просторі. Ці підходи особливо ефективні для обробки різномірних даних і виявлення складних нелінійних залежностей.

Сучасні рекомендаційні системи все частіше інтегрують дані з зовнішніх джерел, таких як Google Analytics або соціальні мережі. Це дозволяє враховувати не лише внутрішні взаємодії на платформі, але й загальні тенденції споживання, сезонність, популярність товарів у соціальних мережах. Для обробки цих даних використовуються методи частотного аналізу, машинного навчання на основі подій та аналізу часових рядів. Особливу увагу приділяють аналізу послідовностей дій користувачів за допомогою рекурентних нейронних мереж або трансформерів, що дозволяє виявляти часові залежності у поведінці та передбачати наступні дії.

Критично важливою характеристикою сучасних систем є їх здатність до онлайн-навчання. Традиційні підходи, що вимагають періодичного перетренування моделі на всіх даних, стають неефективними для динамічних середовищ електронної комерції. Сучасні алгоритми інкрементального навчання дозволяють адаптувати параметри моделі в реальному часі на основі нових взаємодій без повного перетренування. Це забезпечує оперативну реакцію на зміни у поведінці користувачів, оновлення товарного асортименту та сезонні коливання попиту [24].

Ефективність рекомендаційної системи визначається комплексом факторів, серед яких точність прогнозу, різноманітність пропозицій, контекстуальна релевантність та швидкість відгуку. Сучасні алгоритми дозволяють не лише пропонувати релевантні товари, але й враховувати багатовимірний контекст взаємодії, включаючи пристрій користувача, час доби, історію переглядів та інші ситуаційні фактори. Це значно підвищує якість персоналізації та загальний користувацький досвід.

Окремо варто відзначити проблеми, з якими стикаються розробники рекомендаційних систем. До них належать проблема "фільтраційної бульбашки", коли система починає пропонувати занадто схожі товари, обмежуючи вибір користувача; вимоги до обчислювальних ресурсів для обробки великих обсягів даних у реальному часі; необхідність балансу між персоналізацією та новизною рекомендацій. Вирішення цих проблем вимагає використання спеціалізованих алгоритмів, таких як bandit-алгоритми для дослідження-використання, метрики diversity для оцінки різноманітності та розподілених обчислювальних систем для обробки даних.

Таким чином, сучасні рекомендаційні системи в електронній комерції представляють собою складні інтелектуальні системи, що поєднують передові методи машинного навчання з глибоким розумінням специфіки бізнесу та поведінки споживачів. Їх розвиток продовжується у напрямку підвищення точності, швидкодії та здатності адаптуватися до індивідуальних потреб кожного користувача в реальному часі.

1.2 Аналіз предметної області рекомендаційних систем

Основна цінність рекомендаційних систем полягає в їх здатності вирішувати складні завдання електронної комерції. Вони ефективно поділяються з проблемою інформаційного перевантаження, фільтруючи величезні каталоги товарів і пропонуючи кожному користувачеві найбільш релевантні позиції. Завдяки персоналізованим підходам вони значно збільшують глибину перегляду каталогу, стимулюючи покупців до знайомства з більшою кількістю товарів.

Розвиток технологій останніх років значно розширив доступність рекомендаційних систем. Якщо раніше вони були привілеєм великих маркетплейсів, то сьогодні навіть невеликі онлайн-бізнеси можуть собі дозволити впровадження складних алгоритмів машинного навчання завдяки хмарним рішенням та готовим API сервісам. Однак створення дійсно ефективної системи залишається складною задачею, яка вимагає глибокого розуміння як технічних аспектів, так і специфіки конкретного бізнесу.

Для оцінки ефективності роботи рекомендаційних систем використовується комплекс метрик. Click-Through Rate відображає клікабельність рекомендацій, Conversion Rate показує відсоток покупок, здійснених через рекомендації, а Average Order Value демонструє вплив системи на середній чек. Не менш важливими є показники Diversity та Serendipity, які відображають різноманітність пропонованих товарів та наявність неочікуваних, але корисних рекомендацій [4].

Особливу увагу приділяють вирішенню проблеми "холодного старту", коли система стикається з нестачею даних про нового користувача або новий товар. Для подолання цієї ситуації використовуються гібридні підходи, що поєднують аналіз атрибутів товарів з базовими поведінковими сигналами. Наприклад, для нового товару можуть враховуватися його категорія, бренд та цінова категорія, а для нового користувача - демографічні дані або технічні характеристики пристрою.

Важливим аспектом сучасних рекомендаційних систем є підтримання оптимального балансу між персоналізацією та дослідженням. Надмірна персоналізація може призвести до так званого ефекту "фільтраційної бульбашки", коли користувач постійно отримує пропозиції лише схожих товарів, що обмежує його досвід. Для запобігання цьому використовуються спеціальні алгоритми, які вводять елемент випадковості або новизни в рекомендації, розширюючи таким чином спектр пропонованих товарів.

Технологічна реалізація сучасних рекомендаційних систем базується на потужних інструментах обробки даних. Для роботи з великими обсягами інформації застосовуються такі рішення як Apache Spark та Hadoop. Машинне навчання реалізується через фреймворки TensorFlow та PyTorch, а мікросервісна архітектура будується з використанням Docker та Kubernetes. Для ефективного пошуку схожих товарів використовуються спеціалізовані векторні бази даних, такі як Pinecone чи Milvus, а аналіз даних у реальному часі здійснюється за допомогою Apache Kafka та Flink [26].

Цільова аудиторія рекомендаційних систем відрізняється різноманітністю потреб і очікувань. Звичайні покупці цінують зрозумілість і релевантність пропозицій, маркетологи потребують інструментів для налаштування бізнес-правил, а технічні фахівці зосереджені на питаннях інтеграції та масштабованості системи. Задоволення всіх цих вимог одночасно становить значний виклик для розробників.

Сучасна рекомендаційна система в електронній комерції являє собою складний технологічний комплекс, який поєднує передові методи обробки даних, глибоке розуміння психології споживачів та специфіку бізнес-процесів конкретного онлайн-магазину. Її ефективна реалізація вимагає комплексного підходу, що охоплює аспекти data science, програмної інженерії та дизайну користувацького досвіду. Майбутній розвиток цієї галузі пов'язаний з інтеграцією генеративних AI моделей, вдосконаленням контекстного аналізу та розвитком мультиканальних рекомендацій, що працюють узгоджено на всіх точках дотику з клієнтом.

1.3 Аналіз аналогів програмного продукту

Аналіз існуючих рекомендаційних систем для електронної комерції є критично важливим етапом у процесі проектування власного рішення. Дослідження ринкових пропозицій дозволяє виявити найефективніші практики, зрозуміти очікування кінцевих користувачів та сформувані конкурентні переваги нового продукту. Сьогодні на ринку присутні численні рішення для персоналізації покупок, кожне з яких має свої особливості реалізації та цільову аудиторію.

Amazon Personalize є одним з найпопулярніших рішень для електронної комерції, яке пропонує комплексний підхід до генерації рекомендацій [10]. Ця система, розроблена на основі технологій Amazon Web Services, використовує передові алгоритми машинного навчання для аналізу поведінки користувачів. Головною перевагою цього рішення є його масштабованість та інтеграція з іншими сервісами AWS, що дозволяє обробляти величезні обсяги даних у реальному часі. Система автоматично адаптує рекомендації на основі останніх взаємодій користувача, що забезпечує високу релевантність пропозицій. Однак основним недоліком Amazon Personalize є його висока вартість та складність налаштування, що робить його недоступним для малих та середніх бізнесів.

На рис. 1.1 зображено головну сторінку програмного продукту AWS.

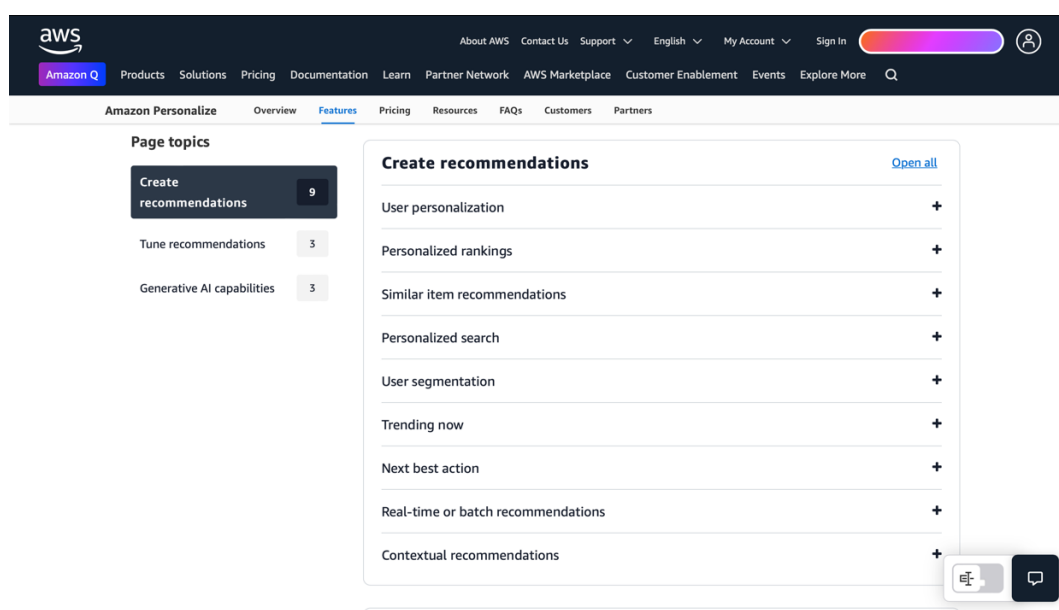


Рисунок 1.1 – Головна сторінка Amazon Personalize

Algolia Recommend пропонує більш спеціалізований підхід до генерації рекомендацій, орієнтований на пошукові запити та поведінку користувачів [12]. Ця система відзначається високою швидкістю та можливістю тонкого налаштування правил рекомендацій. Algolia дозволяє комбінувати різні типи рекомендацій, такі як "купають разом", "схожі товари" або "персоналізовані підбірки", забезпечуючи гнучкість у роботі. Особливістю цього рішення є його здатність інтегруватися з існуючими системами електронної комерції через API, що значно спрощує процес впровадження. Недоліком можна вважати обмежену можливість аналізу довгострокової історії покупок, оскільки система орієнтована на поточну сесію користувача.

На рис. 1.2 зображено інтерфейс програмного продукту Algolia.

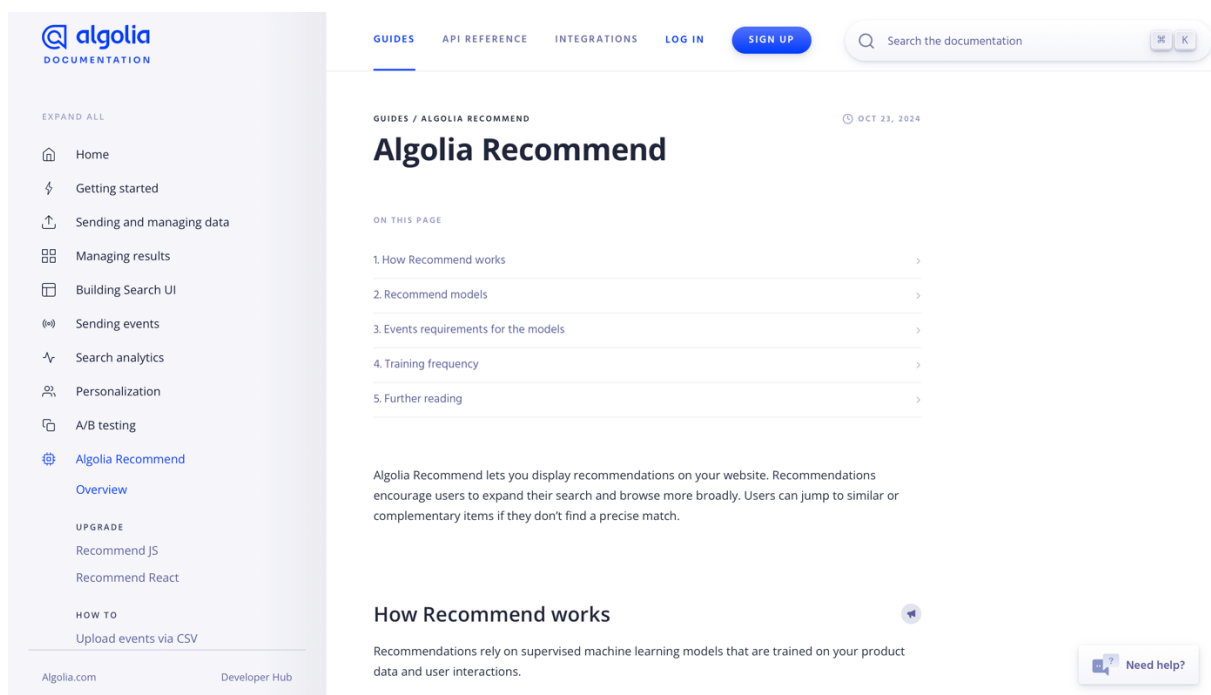


Рисунок 1.2 – Інтерфейс Algolia Recommend

Resombee є спеціалізованим рішенням для середнього бізнесу, яке поєднує в собі методи колаборативної фільтрації та контент-базованого аналізу [11]. Ця система відзначається простотою інтеграції та наявністю готових шаблонів для різних сценаріїв використання. Resombee пропонує розширені функції A/B тестування, що дозволяє оцінювати ефективність різних алгоритмів рекомендацій. Система також надає детальну аналітику по роботі рекомендацій, що є важливим інструментом для оптимізації продажів. Серед недоліків можна

відзначити обмежену підтримку мов, що ускладнює роботу з неангломовними аудиторіями.

На рис. 1.3 зображено панель управління програмного продукту Recombee.

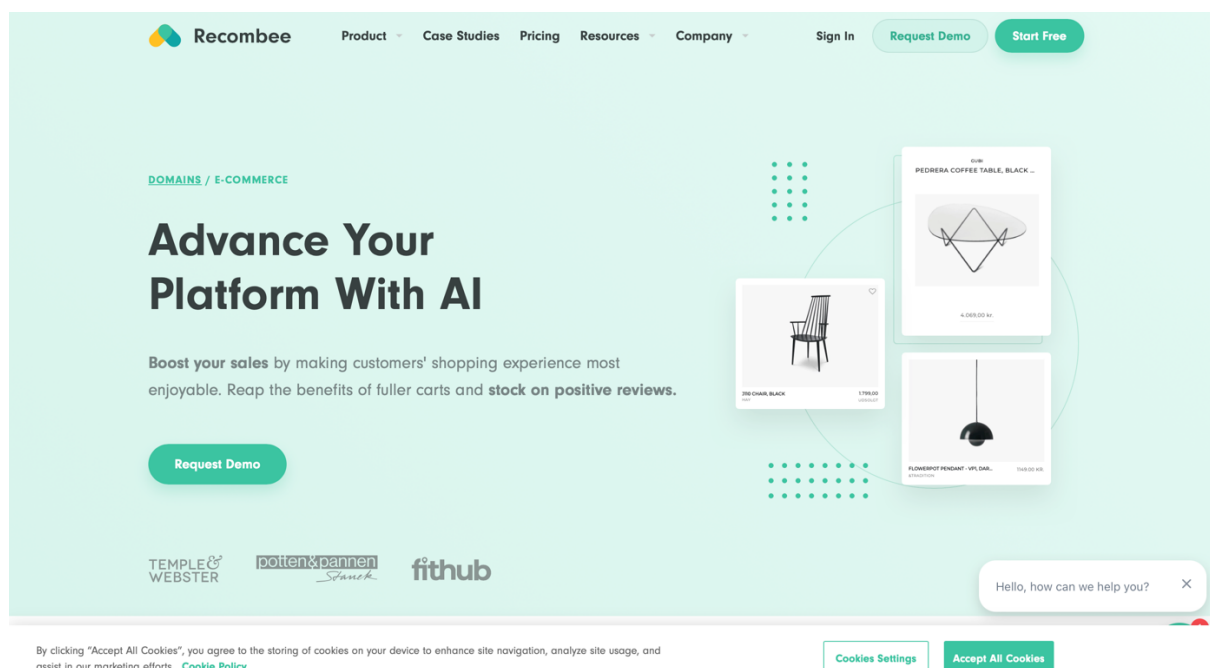


Рисунок 1.3 – Панель управління Recombee

Сучасні рекомендаційні системи демонструють різний підхід до архітектурних рішень. Більшість комерційних продуктів, такі як Adobe Target або Dynamic Yield, використовують мікросервісну архітектуру, що дозволяє їм масштабувати окремі компоненти системи незалежно. Такі рішення зазвичай працюють у хмарних середовищах і пропонують API для інтеграції з існуючими платформами електронної комерції. Водночас, відкриті рішення, такі як Apache Mahout або TensorFlow Recommenders, дозволяють створювати власні системи з нуля, але вимагають значно більших технічних ресурсів для підтримки та розвитку.

Важливим аспектом сучасних рекомендаційних систем є їх здатність обробляти різні типи даних. Кращі рішення на ринку, такі як Salesforce Commerce Cloud, поєднують аналіз структурованих даних (історія покупок, переглядів) з обробкою неструктурованого контенту (відгуки, зображення товарів). Це дозволяє формувати більш точні та релевантні рекомендації, що враховують не лише поведінкові фактори, а й контекст взаємодії з товаром.

Провівши порівняльний аналіз існуючих рішень, можна виділити кілька ключових напрямів для розвитку власної системи. Перш за все, це інтеграція різних типів алгоритмів рекомендацій (колаборативна фільтрація, контент-базований підхід, глибоке навчання) в єдину гібридну модель. По-друге, важливим аспектом є розробка зручного інтерфейсу для налаштування правил рекомендацій, що дозволить маркетологам без глибоких технічних знань керувати роботою системи. По-третє, особливу увагу варто приділити аналітиці роботи системи, щоб забезпечити можливість постійного вдосконалення алгоритмів на основі реальних даних про ефективність рекомендацій.

Проведемо порівняльний аналіз цих рішень з планованим власним продуктом у табличній формі (табл. 1.1) [12].

Таблиця 1.1

Порівняльна характеристика рекомендаційних систем

Критерій оцінки	Amazon Personalize	Algolia Recommend	Recombee	Власне рішення
Складність інтеграції	Висока	Низька	Низька	Середня
Гнучкість налаштувань	Висока	Низька	Висока	Висока
Вартість реалізації	Висока	Низька	Середня	Середня
Якість рекомендацій	Висока	Середня	Висока	Висока
Можливість кастомізації	Обмежена	Дуже обмежена	Висока	Висока
Підтримка мультимовності	Обмежена	Обмежена	Обмежена	Повноцінна
Локалізація для українського ринку	Відсутня	Відсутня	Відсутня	Повноцінна
Можливість роботи з різними CMS	Обмежена	Відсутня	Обмежена	Повноцінна

1.4 Обґрунтування вибору інструментальних засобів

Технологічний стек сучасних рекомендаційних систем зазвичай включає мови програмування Python або Java для реалізації алгоритмів машинного навчання, фреймворки для обробки великих даних (Apache Spark, Hadoop), а також спеціалізовані бібліотеки для роботи з векторними представленнями (Faiss, Annoy). Для зберігання даних використовуються як традиційні реляційні бази даних (PostgreSQL, MySQL), так і спеціалізовані NoSQL рішення (MongoDB, Cassandra), що дозволяють ефективно працювати з великими обсягами неструктурованих даних [13].

Архітектура сучасних рекомендаційних систем зазвичай будується на принципах поділу на окремі модулі: збір даних, попередня обробка, тренування моделей, генерація рекомендацій та їх доставка до клієнтських додатків. Такий підхід дозволяє масштабувати окремі компоненти системи незалежно та забезпечує високу доступність сервісу навіть при значних навантаженнях.

На рис. 1.4 зображено типову архітектуру рекомендаційної системи.

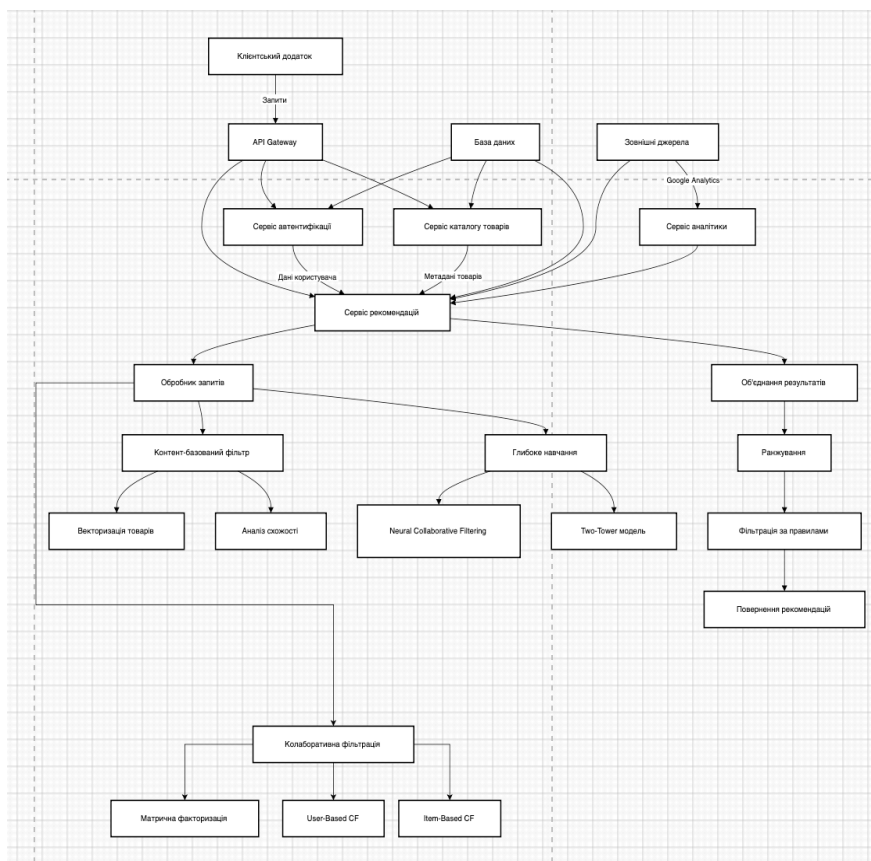


Рисунок 1.4 – Типова архітектура рекомендаційної системи

Важливим аспектом при проектуванні власної системи є забезпечення підтримки різних сценаріїв використання. Наприклад, система повинна вміти генерувати як персоналізовані рекомендації для окремих користувачів, так і загальні підбірки товарів для груп користувачів зі схожими характеристиками. Крім того, важливо враховувати тимчасові фактори, такі як сезонність або актуальні тренди, що можуть впливати на релевантність рекомендацій.

Проведений аналіз дозволив виявити кілька ключових проблем, з якими стикаються сучасні рекомендаційні системи. Перша проблема пов'язана з якістю даних - часто системи отримують неповну або зашумлену інформацію про поведінку користувачів, що негативно впливає на точність рекомендацій. Друга проблема стосується "фільтраційної бульбашки", коли система починає пропонувати занадто схожі товари, обмежуючи вибір користувача. Третя проблема - це складність оцінки ефективності роботи системи, оскільки традиційні метрики (наприклад, CTR) не завжди адекватно відображають реальний вплив на бізнес-показники.

Для подолання цих проблем у власній системі планується реалізувати кілька ключових механізмів. По-перше, це механізм очищення та збагачення вхідних даних, який дозволить підвищити якість інформації, що надходить до алгоритмів рекомендацій. По-друге, система буде включати спеціальні алгоритми балансування між персоналізацією та дослідженням, що запобігатиме ефекту "фільтраційної бульбашки". По-третє, для оцінки ефективності буде використовуватися комплекс метрик, що враховують як безпосередній вплив на продажі, так і довгостроковий ефект у вигляді підвищення лояльності клієнтів.

Серед недоліків монолітної архітектури варто виділити її схильність до зростання розмірів проєкту, що може створювати труднощі з управлінням та розширенням системи. Також існує потенційна залежність між компонентами, яка може негативно впливати на стійкість та надійність усієї системи. Крім того, оновлення окремих компонентів є складним завданням, оскільки зміни можуть впливати на роботу всієї архітектури.

На рис. 1.5 зображено спрощену архітектуру проєкту рекомендаційної системи.

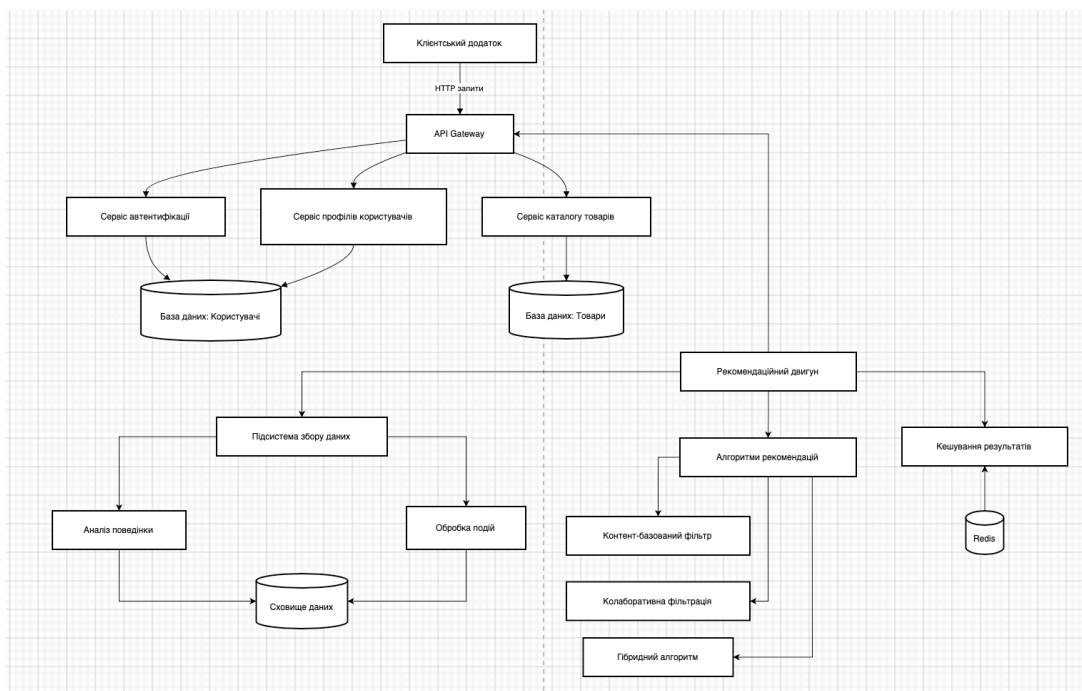


Рисунок 1.5 – Спрощена архітектура проєкту

Важливим аспектом при проектуванні власної системи є забезпечення підтримки різних сценаріїв використання. Наприклад, система повинна вміти генерувати як персоналізовані рекомендації для окремих користувачів, так і загальні підбірки товарів для груп користувачів зі схожими характеристиками. Крім того, важливо враховувати тимчасові фактори, такі як сезонність або актуальні тренди, що можуть впливати на релевантність рекомендацій.

Проведений аналіз дозволив виявити кілька ключових проблем, з якими стикаються сучасні рекомендаційні системи. Перша проблема пов'язана з якістю даних - часто системи отримують неповну або зашумлену інформацію про поведінку користувачів, що негативно впливає на точність рекомендацій. Друга проблема стосується "фільтраційної бульбашки", коли система починає пропонувати занадто схожі товари, обмежуючи вибір користувача. Третя проблема - це складність оцінки ефективності роботи системи, оскільки традиційні метрики (наприклад, CTR) не завжди адекватно відображають реальний вплив на бізнес-показники.

Особливу увагу при проектуванні власної системи варто приділити питанням конфіденційності даних. Сучасні користувачі дедалі більше турбуються про захист своєї персональної інформації, тому важливо забезпечити прозорість роботи алгоритмів та надати користувачам контроль над тим, які дані використовуються для формування рекомендацій. Це може бути реалізовано через систему налаштувань конфіденційності та можливість видалення історії взаємодій.

Проведений аналіз аналогічних рішень дозволив сформувати чітке уявлення про сучасний стан технологій рекомендаційних систем у електронній комерції. На основі цього аналізу було вирішено орієнтуватися на розробку гібридної системи, яка поєднує переваги різних підходів до генерації рекомендацій, при цьому забезпечуючи простоту інтеграції та ефективне використання обчислювальних ресурсів. Важливим аспектом буде також розробка зручних інструментів для аналітики та налаштування системи, що дозволить адаптувати її до специфічних потреб конкретного бізнесу.

Стек технологій, використаний для розробки рекомендаційної системи електронної комерції, включає сучасні інструменти та фреймворки, які забезпечують ефективну роботу всіх компонентів системи.

Основою клієнтської частини додатку є Next.js - React-фреймворк, який дозволяє створювати гібридні додатки з підтримкою статичної генерації, серверного рендерингу та клієнтського рендерингу. Next.js забезпечує швидке завантаження сторінок та оптимізовану роботу з пошуковими системами. Для побудови інтерфейсу використовуються React-компоненти, що дозволяє створювати інтерактивні та динамічні елементи інтерфейсу.

Серверна частина також реалізована за допомогою Next.js API routes, що дає можливість обробляти бізнес-логіку без необхідності використання додаткового серверного фреймворку. Це спрощує архітектуру додатку та зменшує кількість залежностей.

Як основну базу даних обрано MongoDB - документоорієнтовану NoSQL базу даних, яка ідеально підходить для роботи з товарними каталогами. Для

взаємодії з MongoDB використовується Mongoose - ODM бібліотека, яка забезпечує просту роботу з моделями даних та включає вбудовані механізми валідації.

Для відображення рекомендацій у вигляді каруселі товарів застосовано бібліотеку Nuka Carousel, яка надає гнучкі налаштування адаптивного відображення товарів на різних пристроях. Бібліотека підтримує кастомізацію елементів керування та плавні анімації переходів..

Робочим середовищем розробки обрано WebStorm - потужне IDE для JavaScript, яке надає розширені можливості для налагодження, рефакторингу та роботи з кодом. WebStorm особливо ефективний для роботи з Next.js проектами.

Для забезпечення якості коду використовується ESLint - інструмент статичного аналізу коду, який допомагає дотримуватись єдиного стилю кодування та виявляти потенційні проблеми. Налаштування ESLint враховують специфіку Next.js проекту.

Система інтегрується з Google Analytics API для отримання даних про популярність товарів. Це дозволяє враховувати актуальні тенденції при формуванні рекомендацій. Обмін даними з Google Analytics здійснюється через спеціальні серверні ендпоінти.

Для керування станом додатку використовується вбудований механізм React - хуки та контекст. Це дозволяє ефективно керувати даними без застосування додаткових бібліотек стану.

Всі компоненти системи розроблені на чистому JavaScript без використання TypeScript, що спрощує підтримку коду для команд, які не працювали з типізованим JavaScript. Синтаксис коду відповідає стандартам ES6+.

Для стилізації компонентів використовується CSS Modules, що забезпечує локальну область видимості стилів та запобігає конфліктам імен класів. Це дозволяє створювати модульні та повторно використововані компоненти.

Система розгортається на Vercel - хмарній платформі, спеціально оптимізований для Next.js додатків. Vercel забезпечує автоматичне масштабування, CDN для статичних ресурсів та безперервне розгортання з Git.

Висновки до першого розділу

Проведений аналіз сучасних підходів до побудови рекомендаційних систем для електронної комерції дозволив виявити ключові тенденції та визначити оптимальні рішення для власної реалізації. Дослідження показало, що сучасні рекомендаційні системи базуються на комбінації кількох методів, зокрема колаборативної фільтрації, контент-базованого аналізу та алгоритмів машинного навчання.

Аналіз предметної області підтвердив важливість персоналізації у сучасній електронній комерції, де рекомендаційні системи стали невід'ємним інструментом для підвищення конверсії та утримання клієнтів. Особливу увагу приділено проблемі "холодного старту" та необхідності балансу між персоналізацією та різноманіттям рекомендацій.

Порівняльний аналіз існуючих аналогів дозволив виявити як сильні сторони комерційних рішень, так і їх обмеження, що стосуються складності інтеграції, вартості та гнучкості налаштувань. Це обґрунтувало вибір власного рішення на базі Next.js, яке поєднує простоту розробки з достатньою потужністю для реалізації основних сценаріїв рекомендацій.

Стек технологій (Next.js, MongoDB, Mongoose, Nuka Carousel) був обраний з урахуванням специфіки проекту та дозволяє ефективно вирішувати поставлені завдання. Використання чистого JavaScript без TypeScript спрощує підтримку системи, а інтеграція з Google Analytics забезпечує врахування актуальних даних про популярність товарів.

Проведене дослідження підтвердило доцільність обраного підходу до архітектури системи, який поєднує серверний рендеринг Next.js з гнучкістю MongoDB.

РОЗДІЛ 2. ПРОЄКТУВАННЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ДЛЯ ПЛАТФОРМИ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

2.1 Визначення варіантів використання

За ціллю візуалізації взаємозв'язків між компонентами системи, була створена діаграма варіантів використання (рис. 2.1).

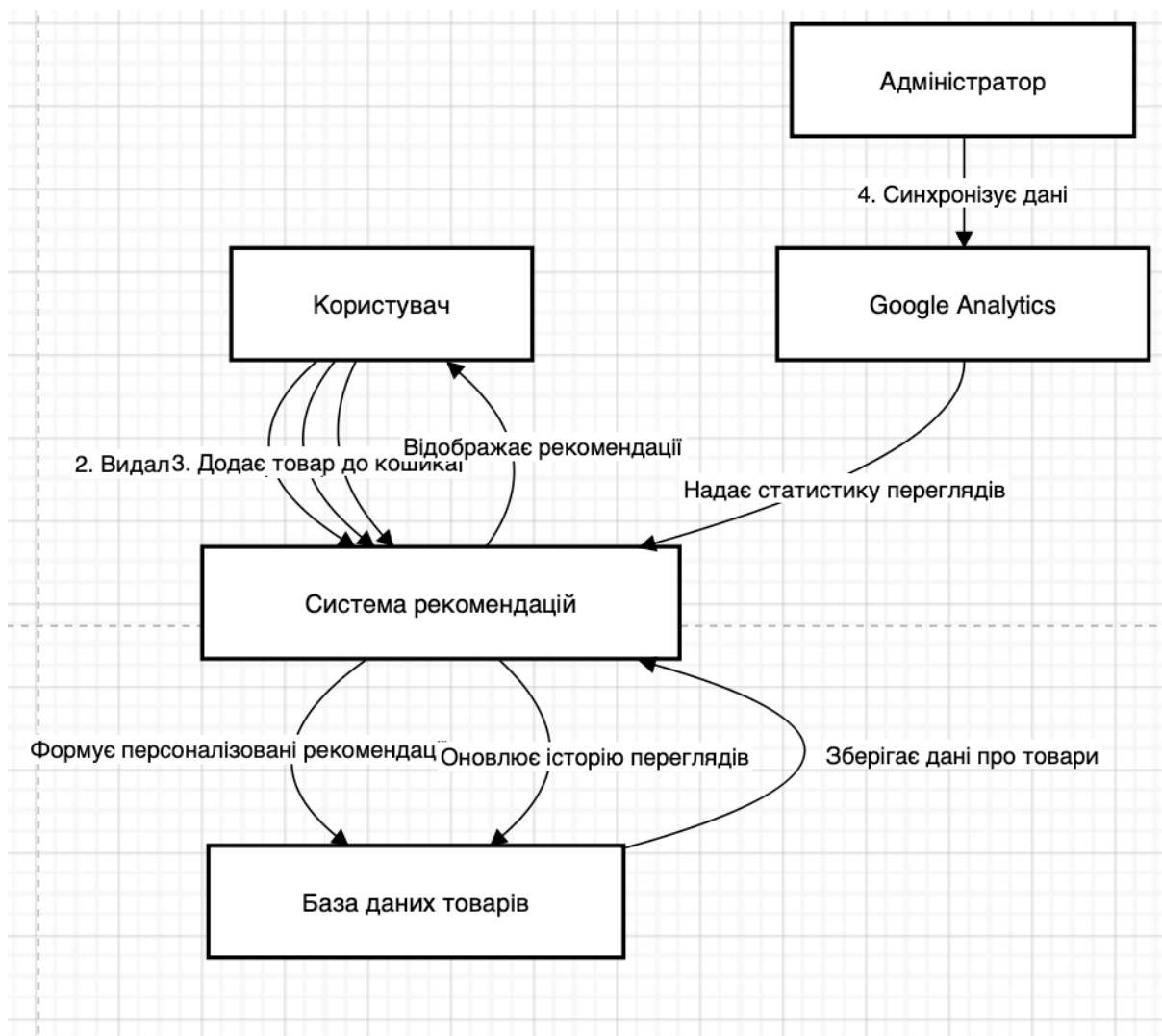


Рисунок 2.1 – Діаграма варіантів використання

Рекомендаційна система для платформи електронної комерції передбачає кілька ключових варіантів використання, які забезпечують її ефективну роботу та корисність для кінцевих користувачів. Кожен варіант використання детально

продуманий та реалізований з урахуванням специфіки роботи інтернет-магазину та потреб його відвідувачів.

Основним варіантом використання є перегляд персоналізованих рекомендацій товарів. Коли користувач відвідує сторінку товару або головну сторінку магазину, система автоматично аналізує його поточну активність та історичні дані про перегляди і покупки. На основі цієї інформації формується блок рекомендацій, який відображає товари, що потенційно можуть зацікавити користувача. Цей процес відбувається в реальному часі, що забезпечує актуальність пропозицій. Система враховує різні фактори, включаючи категорію поточного товару, його характеристики, популярність серед інших покупців та інші параметри [1].

Важливим варіантом використання є управління рекомендаціями через механізм видалення непотрібних пропозицій. Користувач має можливість видалити товар з блоку рекомендацій, якщо він його не цікавить. При цьому система негайно реагує на таку дію - видалений товар прибирається з подальших рекомендацій, а на його місце система підбирає новий варіант, який більше відповідає вподобанням користувача. Цей механізм дозволяє постійно вдосконалювати якість рекомендацій на основі безпосередньої зворотного зв'язку від користувача.

Для адміністратора системи передбачений окремий варіант використання, пов'язаний з оновленням даних про популярність товарів. Через спеціальний інтерфейс адміністратор може ініціювати синхронізацію з Google Analytics, під час якої система отримує актуальні дані про перегляди товарів на сайті. Ця інформація використовується для коригування алгоритмів рекомендацій, що дозволяє враховувати загальні тенденції популярності товарів серед усіх відвідувачів магазину.

Діаграма взаємодії компонентів системи наочно демонструє, як відбувається обмін даними між різними частинами системи. Клієнтський інтерфейс взаємодіє з серверною частиною через API, передаючи інформацію про дії користувача та отримуючи оновлені рекомендації. Серверна частина, у

свою чергу, звертається до бази даних для отримання інформації про товари та до зовнішніх сервісів для отримання додаткових даних. Вся ця взаємодія відбувається в режимі реального часу, що забезпечує оперативне оновлення рекомендацій у відповідь на дії користувача.

Сценарії використання розроблені з урахуванням різних типів користувачів. Для звичайних покупців передбачені механізми отримання персоналізованих рекомендацій та можливість їх корекції. Для нових відвідувачів, про яких ще немає достатньої інформації, система використовує узагальнені дані про популярність товарів серед інших користувачів.

Вибір саме таких варіантів використання обґрунтований аналізом потреб сучасних електронних магазинів та їх клієнтів. Персоналізація рекомендацій дозволяє підвищити конверсію та середній чек замовлень. Можливість управління рекомендаціями через механізм видалення непотрібних товарів забезпечує більш точний підбір пропозицій і підвищує задоволеність користувачів. Інтеграція з Google Analytics дає змогу враховувати загальні тенденції популярності товарів, що особливо важливо для нових відвідувачів, про яких ще немає достатньої інформації. Усі ці фактори разом забезпечують ефективну роботу рекомендаційної системи та її корисність як для магазину, так і для його клієнтів.

2.2 Архітектура системи

Архітектура рекомендаційної системи для платформи електронної комерції була розроблена з урахуванням сучасних вимог до продуктивності, масштабованості та надійності. Система побудована за принципом розділення на клієнтську та серверну частини, що дозволяє ефективно розподіляти навантаження та забезпечує гнучкість у розробці та підтримці.

Клієнтська частина системи реалізована на базі фреймворку Next.js, що дозволило забезпечити швидкий рендеринг сторінок та оптимізовану роботу з інтерфейсом. Основною складовою клієнтської частини є компонент

рекомендацій, який відображає карусель товарів. Для реалізації каруселі використана бібліотека `nuka-carousel`, що забезпечує адаптивність під різні розміри екранів та плавну анімацію переходів. Взаємодія користувача з рекомендаціями реалізована через систему подій, які відправляються на сервер при діях користувача (наприклад, видалення рекомендації або додавання товару до кошика) [13].

На рис. 2.2 зображено секцію з рекомендаціями.

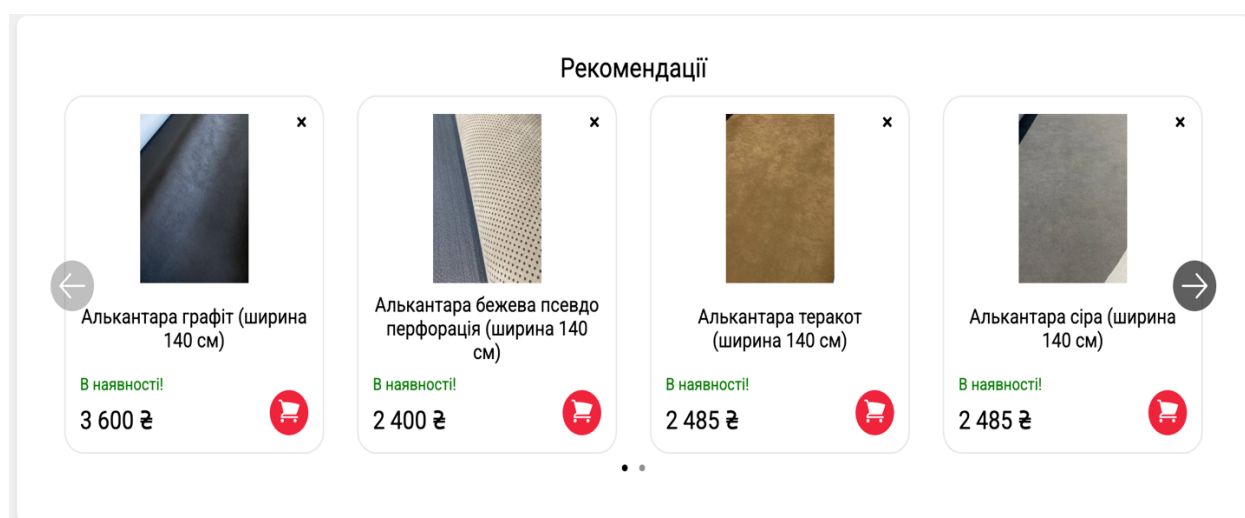


Рисунок 2.2 – Карусель рекомендацій

Серверна частина системи побудована на основі API маршрутів `Next.js`, що дозволило інтегрувати її з існуючою інфраструктурою електронного магазину. Ядром серверної частини є модуль обробки рекомендацій, який включає кілька взаємопов'язаних компонентів. Основним є API ендпоінт `/api/recommendations`, який відповідає за формування списку рекомендованих товарів. Цей ендпоінт виконує складні запити до бази даних, комбінуючи різні критерії відбору товарів для забезпечення максимальної релевантності рекомендацій.

Схема взаємодії компонентів системи демонструє чіткий потік даних між різними рівнями архітектури. Клієнтський інтерфейс відправляє запити до API, які обробляються серверною частиною. Серверна логіка взаємодіє з базою даних для отримання інформації про товари та з зовнішніми сервісами (такими як `Google Analytics`) для отримання додаткових даних про популярність товарів.

Результати обробки повертаються клієнту у форматі JSON, що дозволяє швидко оновлювати інтерфейс без повного перезавантаження сторінки.

Вибір технологічного стеку був обґрунтований низкою факторів. Next.js був обраний як основний фреймворк через його здатність поєднувати переваги React для клієнтської частини та Node.js для серверної логіки. MongoDB [14] була вибрана як основна база даних через її гнучкість у роботі з напівструктурованими даними та хорошу продуктивність при операціях читання, що критично важливо для рекомендаційних систем. Для інтеграції з Google Analytics був використаний офіційний API Google, що забезпечило надійний механізм отримання актуальних даних про перегляди товарів [15].

При проектуванні архітектури виникло кілька технічних проблем, які були успішно вирішені. Однією з основних проблем була необхідність обробки великих обсягів даних у реальному часі для формування персоналізованих рекомендацій. Це було вирішено шляхом впровадження кешування проміжних результатів та оптимізації запитів до бази даних. Іншою проблемою була необхідність забезпечення узгодженості даних між власною базою даних системи та зовнішніми джерелами (Google Analytics). Для вирішення цієї проблеми був розроблений спеціальний механізм синхронізації, який періодично оновлює дані та вирішує конфлікти між різними джерелами.

Важливим аспектом архітектури стала реалізація механізму обробки дій користувача, таких як видалення рекомендацій. При такій дії система не просто приховує товар для конкретного користувача, але й враховує цю інформацію при подальшому формуванні рекомендацій, що значно підвищує їх якість. Це було досягнуто за рахунок розробки спеціальної таблиці в базі даних для зберігання інформації про відхилені користувачами товари.

Для візуалізації архітектури системи була створена детальна діаграма компонентів, яка наочно відображає всі основні елементи системи та їх взаємозв'язки. На діаграмі можна побачити три основні рівні системи: клієнтський інтерфейс, серверну логіку та зовнішні сервіси, а також потоки даних між ними.

На рис. 2.3 зображено детальну діаграму компонентів.

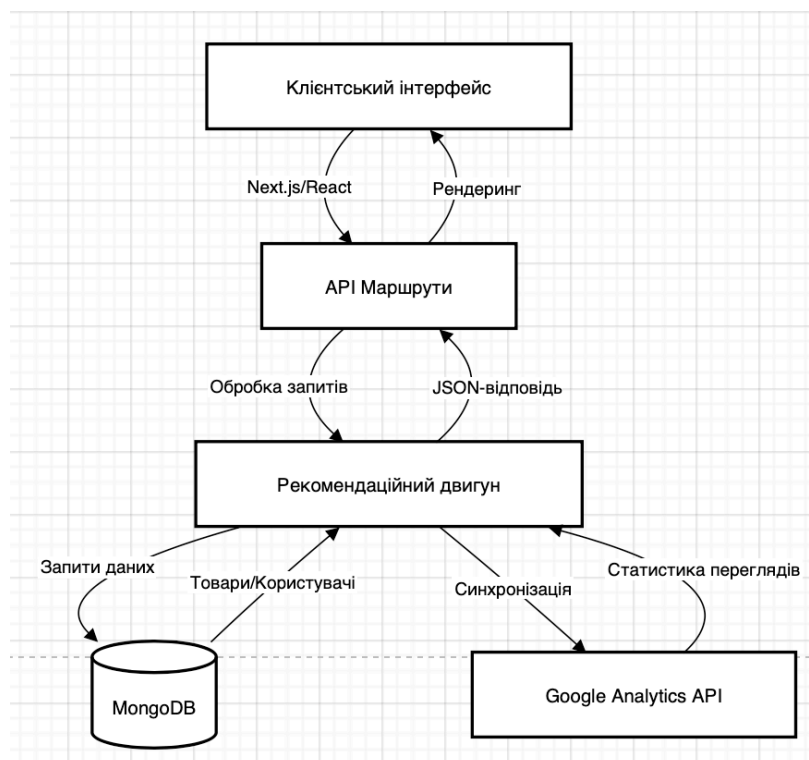


Рисунок 2.3 – Діаграма компонентів

Продуктивність системи була оптимізована за рахунок реалізації механізму пагінації при завантаженні рекомендацій, що дозволило уникнути надмірного навантаження на клієнтську частину при роботі з великими списками товарів. Додатково був реалізований механізм попереднього завантаження даних для рекомендацій, що значно покращило відчуття швидкості роботи системи для кінцевого користувача.

Інтеграція з існуючою платформою електронної комерції була забезпечена через розробку спеціального API шара, який виконує роль посередника між рекомендаційною системою та основним функціоналом магазину. Це дозволило мінімізувати зміни в існуючому коді магазину при впровадженні нової функціональності.

2.3 Алгоритми рекомендацій

Розробка рекомендаційної системи для платформи електронної комерції потребує ретельного підходу до вибору алгоритмів, які забезпечують

релевантність і корисність пропонованих товарів для користувачів. У даному проєкті алгоритми відбору товарів побудовані на основі аналізу категорійної приналежності товарів, їхньої доступності та популярності, отриманої з даних Google Analytics. Оскільки система не передбачає авторизації користувачів і не зберігає історію їхніх переглядів, основний акцент зроблено на контекстному підході, де ключову роль відіграє інформація про поточний товар, який переглядає відвідувач, а також загальні дані про попит на товари, отримані з аналітики. Такий підхід дозволяє адаптувати рекомендації до потреб широкої аудиторії, не покладаючись на персоналізовані дані.

Алгоритм відбору товарів розпочинається з аналізу поточного товару, для якого необхідно сформулювати рекомендації. Основним джерелом даних слугує база MongoDB, де зберігається інформація про товари, зокрема їхні ідентифікатори, назви, ціни, категорії, доступність та зображення. Першим етапом алгоритму є пошук товарів, що належать до тієї ж категорії, що й поточний продукт. Це забезпечує логічну пов'язаність рекомендацій із контекстом перегляду, адже користувач, який цікавиться певним товаром, найімовірніше зацікавиться схожими продуктами з тієї ж категорії. У коді це реалізовано через запит до бази даних із фільтром, який виключає поточний товар і враховує лише доступні продукти, сортуючи їх за датою створення у зворотному порядку, щоб віддавати перевагу новим товарам. Такий підхід дозволяє системі пропонувати актуальні продукти, які можуть бути більш привабливими для покупців через їхню новизну [2].

Однак категорійний підхід має свої обмеження, особливо якщо в обраній категорії недостатньо товарів для формування повноцінного набору рекомендацій. Щоб уникнути ситуацій, коли користувачу відображається замало пропозицій, алгоритм передбачає механізм доповнення. Якщо кількість товарів у поточній категорії менша за встановлений мінімум, система звертається до інших категорій і підтягує додаткові продукти, які також є доступними. У цьому випадку пріоритет зберігається за датою створення, щоб користувач бачив нові товари, незалежно від категорії. У разі, якщо навіть після такого розширення

кількість рекомендацій залишається недостатньою, алгоритм переходить до резервного сценарію, обираючи будь-які доступні товари з бази даних, знову ж таки з сортуванням за новизною. Таким чином, система гарантує, що користувач завжди отримує певну кількість рекомендацій, навіть за умов обмеженого асортименту в конкретній категорії.

Комбінування різних підходів у системі рекомендацій відіграє ключову роль у забезпеченні її гнучкості та адаптивності. Основний метод комбінування полягає у послідовному застосуванні трьох рівнів відбору: спочатку товари з тієї ж категорії, потім із різних категорій, і нарешті будь-які доступні продукти. Цей підхід дозволяє балансувати між точністю рекомендацій, коли пріоритет віддається схожим товарам, і повнотою, коли система заповнює прогалини за рахунок ширшого асортименту. Додатковим елементом комбінування є інтеграція даних про популярність товарів, отриманих із Google Analytics. Хоча основний відбір базується на категоріях і новизні, популярність враховується на етапі заміни рекомендацій, коли користувач видаляє запропонований товар. У цьому випадку система звертається до моделі Recommendation у базі даних, де зберігаються дані про перегляди товарів, і обирає продукт із високим показником популярності, якщо в категорії більше немає відповідних варіантів. Така комбінація контекстного відбору та популярності дозволяє системі залишатися релевантною навіть без персоналізованих даних.

Обробка та аналіз даних із Google Analytics є важливим компонентом системи, що забезпечує її зв'язок із реальними показниками попиту. У проєкті для цього створено спеціальний API-ендпоінт у рамках адмін-панелі, який періодично оновлює інформацію про перегляди товарів. Процес починається з автентифікації через Google API із використанням ключа доступу, після чого система формує запит до Google Analytics Data API. Запит охоплює дані за останній тиждень і включає метрику переглядів сторінок (screenPageViews) із фільтром за шляхом сторінки, що починається з `"/product/"`, щоб отримати інформацію лише про сторінки товарів. Отримані дані обробляються для витягнення ідентифікаторів продуктів із URL-адрес, а також відповідних їм

кількості переглядів. Ця інформація потім записується в базу даних у модель Recommendation, де для кожного товару оновлюється поле переглядів і дата останнього оновлення. Якщо товар раніше не мав запису в цій моделі, створюється новий запис. Такий підхід дозволяє системі накопичувати статистику популярності товарів, яка згодом використовується для заміни видалених рекомендацій.

На рис. 2.4 зображено головну сторінку Google Analytics з даними про активність користувачів.

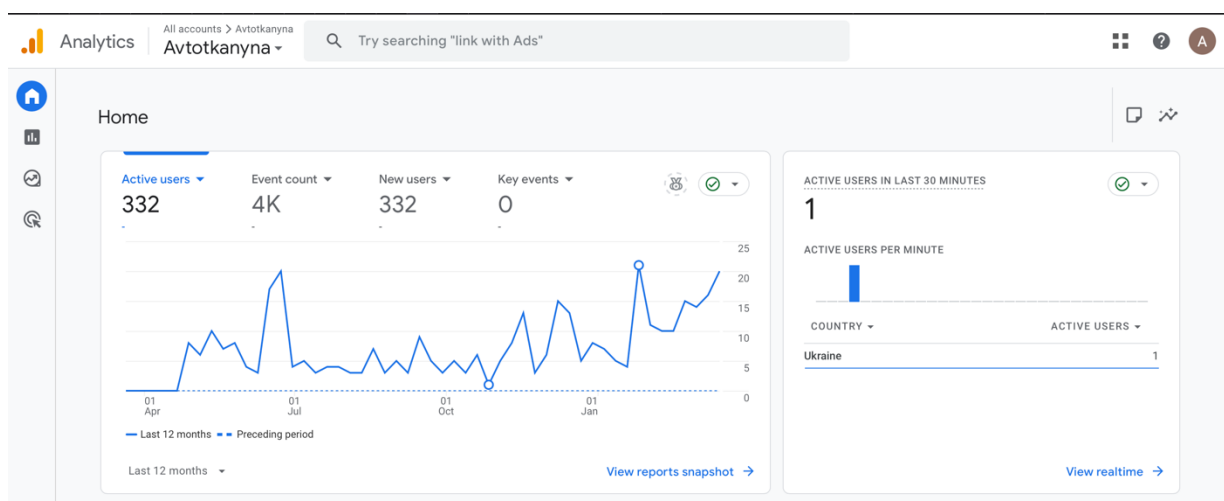


Рисунок 2.4 – Google Analytics із даними переглядів

Для реалізації цього процесу використано бібліотеку `googleapis`, яка забезпечує зручну взаємодію з аналітичними сервісами Google. Обробка даних відбувається асинхронно, що дозволяє уникнути перевантаження сервера під час оновлення. У разі відсутності даних від Google Analytics або помилок при обробці система повертає відповідні повідомлення, що дозволяє адміністратору вчасно реагувати на проблеми. Отримана статистика переглядів не впливає на початковий відбір рекомендацій, але відіграє важливу роль у динамічному оновленні списку, коли користувач видаляє певний товар із каруселі. У такому випадку алгоритм заміни звертається до даних популярності, щоб підібрати продукт, який має високий попит серед відвідувачів сайту, тим самим підвищуючи ймовірність зацікавленості користувача.

Критерії оцінки якості рекомендацій у даній системі базуються на кількох аспектах, які враховують як технічну ефективність, так і користувацький досвід.

Першим критерієм є релевантність рекомендацій, яка визначається за тим, наскільки запропоновані товари відповідають контексту перегляду, тобто належать до тієї ж категорії, що й поточний продукт. Оскільки система не має доступу до персональних даних користувача, релевантність оцінюється через логічну зв'язаність категорій і доступність товарів. Другим критерієм є повнота, тобто здатність системи завжди надавати достатню кількість рекомендацій, навіть якщо асортимент у певній категорії обмежений. Це досягається завдяки механізму доповнення з інших категорій і резервному відбору доступних товарів.

На рис. 2.5 зображена діаграма, що показує послідовність відбору товарів для рекомендацій із трьома рівнями.

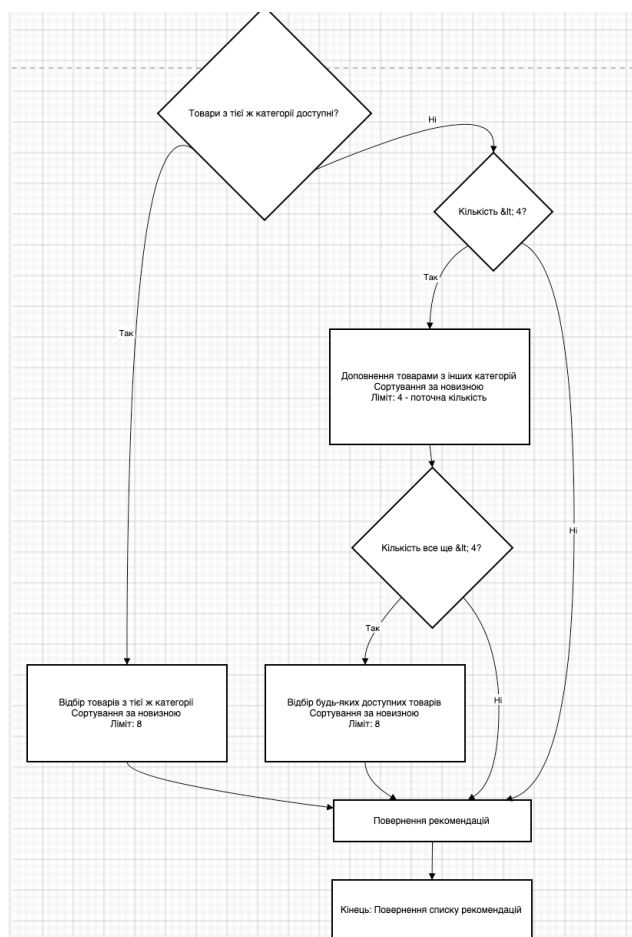


Рисунок 2.5 – Діаграма послідовності відбору

Третім критерієм є адаптивність, яка проявляється в можливості системи реагувати на дії користувача, зокрема на видалення рекомендацій. У цьому контексті якість оцінюється за швидкістю заміни видаленого товару та

доречністю нового вибору, де враховується популярність із Google Analytics. Четвертим критерієм є швидкодія системи, адже відображення рекомендацій має відбуватися без значних затримок, щоб не погіршувати користувацький досвід. У проєкті це забезпечено завдяки асинхронним запитам до API та оптимізованій роботі з базою даних MongoDB. Нарешті, п'ятим критерієм є візуальна привабливість і зручність інтерфейсу, адже рекомендації подаються через адаптивну карусель, яка коректно відображається на пристроях із різною шириною екрана.

На рис. 2.6 – 2.7 зображена адаптивність секції рекомендацій на різних пристроях.

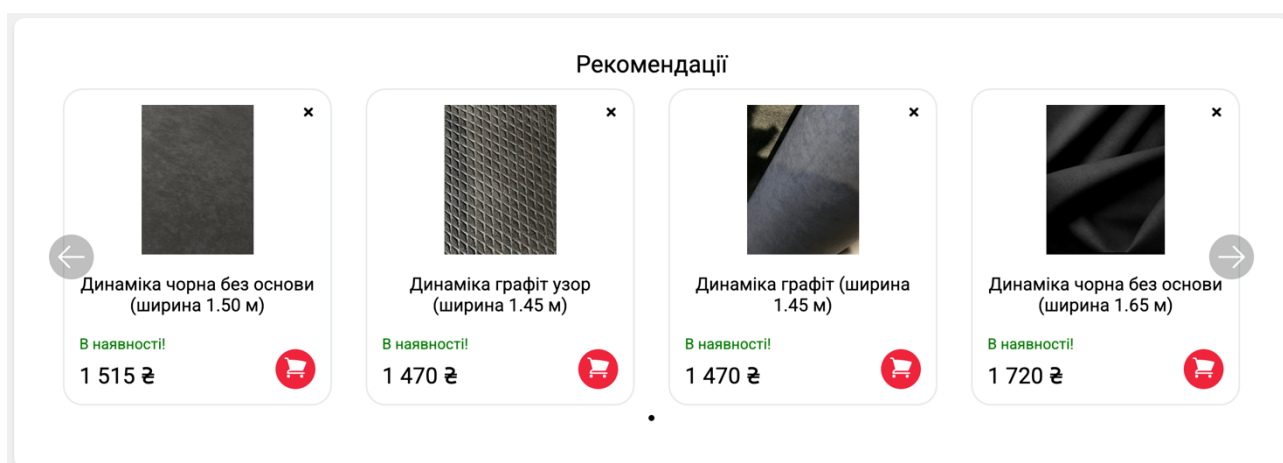


Рисунок 2.6 – Адаптивність каруселі на ПК

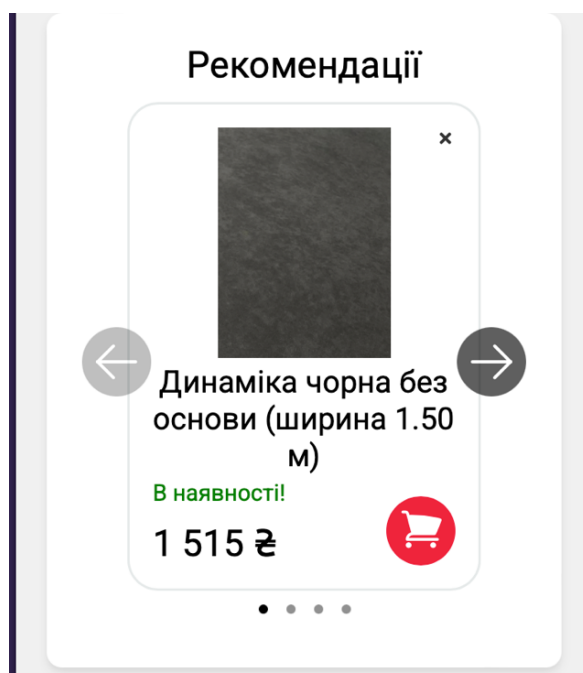


Рисунок 2.7 – Адаптивність каруселі на телефоні

Для оцінки цих критеріїв можна використовувати як кількісні, так і якісні методи. Наприклад, релевантність і повноту можна виміряти через відсоток рекомендацій, що належать до тієї ж категорії, або через середню кількість запропонованих товарів за один запит. Адаптивність і швидкодія піддаються аналізу через час відгуку API та затримку при заміні товарів. Візуальну складову можна оцінити за відгуками тестових користувачів або аналізом взаємодії з каруселлю через Google Analytics, якщо додати відповідні події. У майбутньому для підвищення якості рекомендацій можна розглянути інтеграцію додаткових джерел даних, наприклад, аналізу трендів у продажах, однак на поточному етапі система ефективно виконує свої функції в межах заданого контексту.

Таким чином, алгоритми рекомендацій у цьому проєкті поєднують контекстний підхід із використанням категорій, гнучке доповнення асортименту та аналіз популярності через Google Analytics. Вони забезпечують релевантність, повноту й адаптивність пропозицій, враховуючи обмеження відсутності персоналізації, і створюють міцну основу для подальшого розвитку системи.

2.4 Реалізація бази даних

Розробка бази даних для рекомендаційної системи платформи електронної комерції є ключовим етапом, який визначає ефективність роботи всієї системи. У даному проєкті використано MongoDB — документо-орієнтовану базу даних, яка забезпечує гнучкість у роботі з неструктурованими даними та дозволяє легко масштабувати систему в майбутньому. Оскільки проєкт не передбачає авторизації користувачів і не зберігає історію їхніх переглядів, структура бази даних зосереджена на підтримці рекомендацій, базованих на категоріях товарів і їхній популярності, отриманій із Google Analytics. Основними сутностями бази даних є моделі **Product** і **Recommendation**, які разом формують основу для зберігання інформації про товари та їхню релевантність у контексті рекомендацій. Додатково використовується модель **Category** для групування товарів за категоріями.

Модель **Product** слугує центральним елементом бази даних, оскільки вона містить усю інформацію про товари, доступні на платформі електронної комерції. Ця модель включає набір полів, які описують ключові характеристики продукту. Поле **title** є обов'язковим і зберігає назву товару, що відображається на сайті. Поле **description** дозволяє додавати детальний опис, хоча воно не є обов'язковим і може бути відсутнім для деяких товарів. Для SEO-оптимізації передбачені поля **metaTitle** і **metaDescription**, які використовуються для метаданих сторінок товарів. Поле **code** призначене для зберігання унікального коду продукту, якщо такий застосовується в системі. Ціна товару зберігається в полі **price**, яке також є обов'язковим, адже без цього система не зможе коректно відобразити товар у рекомендаціях чи кошику. Зображення продуктів зберігаються у вигляді масиву рядків у полі **images**, де кожен елемент масиву є URL-адресою зображення. Категорія товару пов'язана через поле **category**, яке посилається на ідентифікатор об'єкта в колекції **Category** за допомогою типу **ObjectId**. Додаткові властивості товару, такі як розмір чи колір, зберігаються в полі **properties** як об'єкт, що забезпечує гнучкість у роботі з різними типами продуктів. Поле **available** вказує на статус доступності товару (тип **Boolean**), наприклад, чи можна додати товар до кошика. Нарешті, поля **createdAt** і **updatedAt** автоматично додають дати створення та оновлення запису, що полегшує відстеження актуальності даних.

Структура моделі **Product** представлена в табл. 2.1.

Таблиця 2.1

Структура моделі Product

Поле	Тип даних	Опис
id	ObjectId	Унікальний ідентифікатор продукту
title	String	Назва продукту (обов'язкове поле)
description	String	Опис продукту (необов'язкове поле)
metaTitle	String	Мета-заголовок для SEO (необов'язкове поле)
metaDescription	String	Мета-опис для SEO (необов'язкове поле)

code	String	Код продукту (необов'язкове поле)
price	Number	Ціна продукту (обов'язкове поле)
images	String[]	Масив посилань на зображення продукту
category	ObjectId	Ідентифікатор категорії (зовнішній ключ до таблиці Category)
properties	Object	Властивості продукту (об'єкт із додатковими характеристиками)
available	String	Доступність продукту
createdAt	Date	Дата створення продукту
updatedAt	Date	Дата останнього оновлення продукту

Модель **Recommendation** доповнює **Product**, зберігаючи інформацію, пов'язану з рекомендаційною системою. Основним полем цієї моделі є **productId**, яке посилається на ідентифікатор товару з колекції **Product** і є обов'язковим, адже без цього зв'язку рекомендація втрачає сенс. Поле **categoryId** також використовує тип **ObjectId** для зв'язку з колекцією **Category**, хоча воно не є обов'язковим і може бути відсутнім, якщо рекомендація не прив'язана до конкретної категорії. Поле **views** зберігає кількість переглядів товару, отриману з Google Analytics, і має значення за замовчуванням 0, якщо дані ще не оновлені. Поле **lastUpdated** фіксує дату останнього оновлення даних про перегляди, що дозволяє відстежувати актуальність статистики. Ця модель відіграє важливу роль у процесі заміни рекомендацій, коли користувач видаляє запропонований товар, адже система звертається до поля **views** для вибору популярного продукту.

Структура моделі **Recommendation** представлена в табл. 2.2.

Таблиця 2.2

Структура моделі Recommendation

Поле	Тип даних	Опис
id	ObjectId	Унікальний ідентифікатор рекомендації
productId	ObjectId	Ідентифікатор продукту, до якого відноситься рекомендація (зовнішній ключ до таблиці Product)
categoryId	ObjectId	Ідентифікатор категорії продукту (зовнішній ключ до таблиці Category)
views	Number	Кількість переглядів рекомендації
lastUpdated	Date	Дата останнього оновлення рекомендації

Модель **Category** використовується для групування товарів за категоріями. Вона містить поле **name**, яке є обов'язковим і зберігає назву категорії, а також поле **description** для додаткового опису (необов'язкове). Для SEO-оптимізації передбачені поля **metaTitle** і **metaDescription**. Поле **orderCategory** визначає порядок відображення категорії на сайті, а **images** зберігає масив URL-адрес зображень, пов'язаних із категорією. Поле **properties** дозволяє зберігати додаткові характеристики категорії у вигляді об'єкта.

Структура моделі **Category** представлена в табл. 2.3.

Таблиця 2.3

Структура моделі Category

Поле	Тип даних	Опис
id	ObjectId	Унікальний ідентифікатор категорії
name	String	Назва категорії (обов'язкове поле)
description	String	Опис категорії (необов'язкове поле)
metaTitle	String	Мета-заголовок для SEO (необов'язкове поле)
metaDescription	String	Мета-опис для SEO (необов'язкове поле)
orderCategory	Number	Порядок відображення категорії
images	String[]	Масив посилань на зображення категорії

properties	Object	Властивості категорії (об'єкт із додатковими характеристиками)
------------	--------	--

Оптимізація запитів до бази даних є критично важливою для забезпечення швидкодії системи, особливо враховуючи, що рекомендації формуються в реальному часі під час перегляду сторінки товару. У коді проєкту використано асинхронні запити через бібліотеку **Mongoose**, яка забезпечує зручну роботу з **MongoDB** у середовищі **Node.js**. Наприклад, у файлі **/api/recommendations.js** запит для відбору товарів із тієї ж категорії реалізовано через метод **find** із фільтром за полем **category**, умовою виключення поточного товару через **\$ne** і обмеженням кількості результатів за допомогою **limit**. Сортування за полем **createdAt** у зворотному порядку (-1) дозволяє швидко отримувати найновіші товари, що зменшує час обробки запиту. Для підвищення продуктивності в таких запитах можна додати індекси на поля **category** і **createdAt** у колекції **Product**, хоча в поточній реалізації це не вказано явно. Індксація прискорила б пошук і сортування, особливо при зростанні обсягу даних. У разі доповнення рекомендацій товарами з інших категорій або будь-якими доступними продуктами також застосовуються аналогічні методи фільтрації та сортування, що забезпечує консистентність підходу.

Ще одним аспектом оптимізації є обробка запитів на заміну рекомендацій, яка реалізується в **/api/replacerecommendations.js**. Тут система використовує оператор **\$nin** для виключення вже показаних товарів, поточних рекомендацій і видалених ідентифікаторів із вибірки. Це дозволяє уникнути дублювання і гарантує, що користувач отримує нові пропозиції. Якщо в категорії немає відповідних товарів, система звертається до моделі **Recommendation** і сортує записи за полем **views** у порядку спадання, щоб обрати популярний продукт. Такий підхід оптимізує вибір за рахунок попередньо накопичених даних, зменшуючи кількість складних обчислень у реальному часі. Для забезпечення швидкості обробки запитів у цьому сценарії доцільно додати індекс на

поле **viewsy** колекції **Recommendation**, що прискорить сортування за популярністю.

Схеми взаємозв'язків між сутностями в базі даних побудовані навколо зв'язків через ідентифікатори типу **ObjectId**. Модель **Product** пов'язана з колекцією **Category** через поле **category**, що дозволяє групувати товари за категоріями і швидко отримувати релевантні продукти для рекомендацій. Модель **Recommendation** також має зв'язок із **Category** через поле **categoryId**, хоча цей зв'язок є опціональним і використовується лише в окремих випадках, наприклад, для заміни рекомендацій у межах певної категорії. Основний зв'язок у **Recommendation** — це поле **productId**, яке посилається на запис у колекції **Product**. Така структура забезпечує гнучкість: інформація про товари зберігається централізовано в **Product**, а **Recommendation** виступає як допоміжна модель для накопичення статистики популярності. Ця схема дозволяє легко розширювати базу даних у майбутньому, наприклад, додаванням нових полів чи моделей, без значних змін у логіці запитів.

На рис. 2.8 зображена діаграма із відображенням зв'язків між колекціями **Product**, **Recommendation** і **Category**.

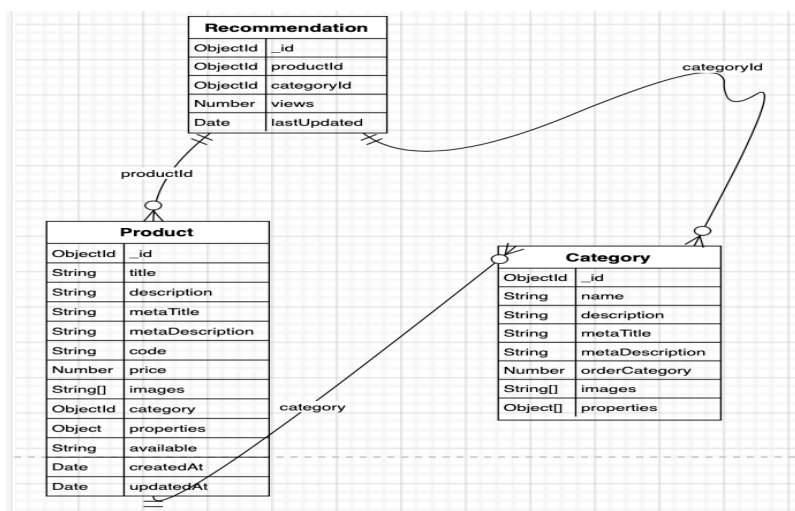


Рисунок 2.8 – Діаграма зв'язків

Забезпечення продуктивності бази даних у проєкті досягається кількома способами. По-перше, використання асинхронного підходу в **Mongoose** дозволяє обробляти запити паралельно, що зменшує затримки при одночасному зверненні кількох користувачів до системи. По-друге, структура моделей мінімізує

дублювання даних: основна інформація про товари зберігається в **Product**, а **Recommendation** містить лише статистику, пов'язану з рекомендаціями. Це знижує обсяг пам'яті, необхідний для зберігання, і спрощує оновлення даних. По-третє, логіка запитів побудована таким чином, щоб уникати надмірної вкладеності чи складних об'єднань, які могли б уповільнити роботу системи. Наприклад, у `/api/recommendations.js` відбір товарів відбувається поетапно, із чітким розподілом на три рівні (товари з тієї ж категорії, з інших категорій, будь-які доступні), що дозволяє оптимізувати використання ресурсів бази даних залежно від доступного асортименту.

Додатковим фактором продуктивності є інтеграція з Google Analytics, яка відбувається через окремий ендпоінт у адмін-панелі (`/api/updateRecommendations.js`). Оновлення даних про перегляди виконується періодично, а не в реальному часі, що зменшує навантаження на базу під час роботи сайту. Отримані дані записуються в **Recommendation** за допомогою операції `findOneAndUpdate` із параметром `upsert`, що дозволяє створювати нові записи або оновлювати існуючі одним запитом. Такий підхід мінімізує кількість операцій із базою і забезпечує консистентність даних.

Для подальшого підвищення продуктивності можна розглянути кешування результатів популярних запитів, наприклад, за допомогою Redis, особливо якщо певні товари чи категорії часто відображаються в рекомендаціях. У поточній реалізації цього не передбачено, але структура бази даних і логіка запитів створюють міцну основу для таких удосконалень. Також варто звернути увагу на моніторинг продуктивності MongoDB, наприклад, через аналіз часу виконання запитів, щоб своєчасно виявляти вузькі місця при зростанні обсягу даних чи кількості користувачів.

Таким чином, реалізація бази даних у проєкті поєднує простоту структури з ефективністю запитів, враховуючи специфіку рекомендаційної системи без персоналізації. Моделі **Product**, **Recommendation** і **Category** забезпечують необхідну функціональність, а оптимізація запитів і чіткі взаємозв'язки між сутностями гарантують швидкодію та масштабованість системи.

2.5 Інтеграція з зовнішніми сервісами

Інтеграція з зовнішніми сервісами у рекомендаційній системі для платформи електронної комерції відіграє важливу роль у забезпеченні актуальності та релевантності пропонованих товарів. У даному проєкті основним зовнішнім сервісом є Google Analytics, який використовується для збору даних про популярність товарів на основі переглядів сторінок. Оскільки система не передбачає авторизації користувачів і не зберігає історію їхніх переглядів, інтеграція з Google Analytics стає ключовим механізмом для аналізу поведінки відвідувачів сайту та оновлення рекомендацій. Крім того, архітектура проєкту дозволяє розглядати можливості підключення інших сервісів у майбутньому, що може розширити функціональність системи. Обробка даних із зовнішніх джерел потребує ретельної нормалізації, щоб забезпечити їхню сумісність із внутрішньою базою даних MongoDB.

Інтеграція з Google Analytics реалізована через спеціальний API-ендпоінт у рамках адмін-панелі, який розташований у файлі `/api/updateRecommendations.js`. Цей процес починається з налаштування автентифікації через Google API. Для цього використано бібліотеку `googleapis`, яка забезпечує зручну взаємодію з сервісами Google. Автентифікація здійснюється за допомогою JSON-файлу з ключем доступу, що зберігається у проєкті під назвою `auto-fabric-brd-55b3ebafbd1a.json`. Цей файл містить `credentials` для доступу до Google Analytics Data API із правами читання, визначеними в обсязі `https://www.googleapis.com/auth/analytics.readonly`. Після успішної автентифікації система формує запит до API для отримання даних про перегляди сторінок за останній тиждень. Період аналізу визначається динамічно: початкова дата встановлюється як поточна дата мінус сім днів, а кінцева — як поточна дата. Таке часове вікно дозволяє отримувати актуальну статистику, яка відображає недавню активність користувачів на сайті.

Запит до Google Analytics включає кілька ключових параметрів. Метрика `screenPageViews` використовується для підрахунку кількості переглядів

сторінок, а розмірність `pagePath` дозволяє отримати URL-адреси сторінок, які відвідували користувачі. Щоб ізолювати дані, пов'язані з товарами, застосовується фільтр, який включає лише сторінки, що починаються з `"/product/"`. Це відповідає структурі URL на сайті, де кожна сторінка товару має формат `/product/[productId]`. Результати сортуються за спаданням кількості переглядів, щоб виділити найбільш популярні товари. Отримана відповідь від API є об'єктом JSON, який містить масив рядків із шляхами сторінок і відповідними значеннями переглядів. Ці дані потребують подальшої обробки, щоб витягти ідентифікатори товарів і привести їх до формату, сумісного з базою даних проєкту.

На рис. 2.9 зображена Google Analytics із даними про перегляди сторінок товарів.

☒	Total	100% of total	100% of total	Avg 0%	Avg 0%	100% of total
☒ 1	Автомобільна тканина. Купити в Україні Avtotkanyna	152 (23.03%)	36 (55.38%)	4.22	52s	290 (27.18%)
☒ 2	Динаміка графіт псевдо перфорація (ширина 1.45 м) Avtotkanyna	43 (6.52%)	4 (6.15%)	10.75	2m 15s	72 (6.75%)
☒ 3	Каталог Avtotkanyna	38 (5.76%)	14 (21.54%)	2.71	42s	45 (4.22%)
☒ 4	Кошик Avtotkanyna	24 (3.64%)	2 (3.08%)	12.00	4m 49s	32 (3%)
☒ 5	Купити алькантара Avtotkanyna	24 (3.64%)	9 (13.85%)	2.67	38s	31 (2.91%)
☐ 6	Купити динаміка (dinamica) Avtotkanyna	22 (3.33%)	12 (18.46%)	1.83	28s	35 (3.28%)
☐ 7	Шкіряний чорний без паралона Skoda (ширина 1,65 м) Avtotkanyna	22 (3.33%)	2 (3.08%)	11.00	6m 40s	42 (3.94%)
☐ 8	Алькантара узор Audi (ширина 140 см) Avtotkanyna	21 (3.18%)	5 (7.69%)	4.20	59s	36 (3.37%)
☐ 9	Купити тканина для сидінь Avtotkanyna	19 (2.88%)	11 (16.92%)	1.73	43s	26 (2.44%)
☐ 10	Алькантара сіра (ширина 140 см) Avtotkanyna	18 (2.73%)	1 (1.54%)	18.00	2m 38s	27 (2.53%)

Рисунок 2.9 – Дані про перегляди сторінок на сайті

Обробка даних із Google Analytics відбувається в кілька етапів. Спочатку система перевіряє наявність рядків у відповіді API. Якщо даних немає, повертається повідомлення про відсутність статистики, що дозволяє адміністратору зрозуміти, що оновлення рекомендацій не відбулося через брак інформації. У разі наявності даних кожен рядок аналізується для витягнення ідентифікатора товару з URL-шляху. Це реалізовано за допомогою регулярного виразу, який шукає 24-символьний MongoDB ObjectId у форматі `/product/[id]`. Якщо ідентифікатор успішно витягнуто, створюється об'єкт із

полями `productId` і `views`, де кількість переглядів конвертується з рядка в число. Після цього система перевіряє, чи існує товар із таким ідентифікатором у колекції `Product`. Якщо товар знайдено, дані про перегляди записуються в модель `Recommendation` через операцію `findOneAndUpdate` із параметром `upsert`. Ця операція або оновлює існуючий запис, установлюючи нові значення для `views` і `lastUpdated`, або створює новий запис, якщо такого ще не було. Такий підхід забезпечує ефективне оновлення бази даних без зайвих запитів.

Можливості розширення для підключення інших зовнішніх сервісів у проєкті є перспективним напрямком розвитку. Поточна архітектура, яка базується на асинхронних API-ендпоінтах і модульній структурі коду, дозволяє відносно легко інтегрувати додаткові джерела даних. Наприклад, можна підключити сервіси аналізу продажів, такі як внутрішня CRM-система або платіжні платформи (Stripe, PayPal), щоб враховувати не лише перегляди, а й фактичні покупки при формуванні рекомендацій. Для цього знадобиться створити новий API-ендпоінт, аналогічний `/api/updateRecommendations.js`, який отримуватиме дані про продажі, нормалізуватиме їх і оновлюватиме поле в моделі `Recommendation`, наприклад, `purchases`. Іншим варіантом є інтеграція з сервісами аналізу трендів, такими як Google Trends, для визначення популярних категорій товарів у певний період часу. Це могло б доповнити логіку відбору рекомендацій, дозволяючи системі пропонувати товари, які відповідають сезонним або ринковим тенденціям. Структура бази даних із моделями `Product` і `Recommendation` уже готова до такого розширення, адже додавання нових полів чи зв'язків не порушить існуючу логіку.

Ще одним потенційним напрямком є підключення сервісів машинного навчання, таких як TensorFlow.js або зовнішні API (наприклад, AWS SageMaker), для аналізу патернів у поведінці користувачів на основі агрегованих даних. Хоча персоналізація відсутня через брак авторизації, агрегована статистика з Google Analytics чи інших джерел могла б слугувати основою для прогнозування популярності товарів. Реалізація такого підходу потребувала б розширення

моделі Recommendation додатковими полями, наприклад, для зберігання прогнозованих рейтингів, а також інтеграції з відповідними API чи бібліотеками. Поточний код уже демонструє гнучкість у роботі з асинхронними запитами, що полегшить додавання таких функцій у майбутньому.

На рис. 2.10 зображено діаграму, що показує можливу інтеграцію з додатковими сервісами, наприклад, CRM і Google Trends.

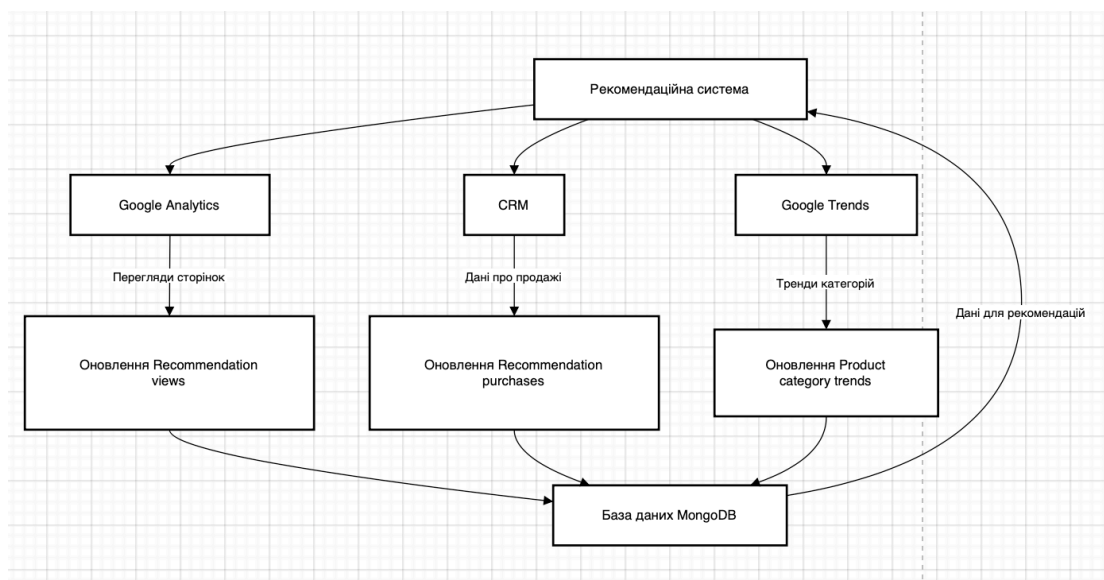


Рисунок 2.10 – Інтеграція з додатковими сервісами

Обробка та нормалізація даних із зовнішніх джерел є критично важливою для забезпечення їхньої сумісності з внутрішньою логікою системи. У випадку Google Analytics нормалізація включає кілька кроків. По-перше, витягнення ідентифікаторів товарів із URL-шляхів потребує чіткого форматування даних, щоб уникнути помилок при зіставленні з базою Product. Регулярний вираз, використаний у коді, гарантує, що лише валідні MongoDB ObjectId потрапляють у подальшу обробку. По-друге, числові значення переглядів конвертуються з рядків у числа, щоб коректно записати їх у поле views. По-третє, система обробляє можливі помилки, такі як відсутність даних чи проблеми з автентифікацією, повертаючи відповідні повідомлення через API. Це дозволяє адміністратору відстежувати стан інтеграції та вчасно реагувати на збої.

Для майбутніх інтеграцій із іншими сервісами нормалізація може бути складнішою. Наприклад, дані про продажі з CRM можуть надходити у вигляді

JSON із різними форматами цін чи ідентифікаторів товарів. У такому випадку знадобиться додаткова логіка для приведення цих даних до стандарту проєкту, наприклад, конвертація валют чи зіставлення ідентифікаторів із внутрішніми кодами товарів. Аналогічно, дані з Google Trends можуть потребувати агрегації за категоріями, щоб корелювати їх із полем `category` у моделі `Product`. У поточній реалізації обробка помилок і логування через `console.log` уже забезпечують базовий рівень контролю, але для масштабніших інтеграцій доцільно додати систему сповіщень або детальніший лог-файл.

Таким чином, інтеграція з Google Analytics у проєкті забезпечує систему актуальними даними про популярність товарів, які використовуються для динамічного оновлення рекомендацій. Гнучка архітектура дозволяє розширювати функціональність шляхом підключення інших сервісів, таких як CRM чи платформи аналізу трендів, а ретельна обробка та нормалізація даних гарантують їхню сумісність із внутрішньою базою. Ці аспекти створюють міцну основу для подальшого розвитку рекомендаційної системи.

Висновки до другого розділу

Розробка рекомендаційної системи для платформи електронної комерції стала важливим етапом у створенні інструменту, який полегшує взаємодію користувачів із сайтом і підвищує їхню зацікавленість в асортименті. Через відсутність авторизації та історії переглядів було обрано контекстний підхід, який базується на категоріях товарів і їхній популярності з Google Analytics. Це забезпечило релевантність рекомендацій без збору персональних даних, відповідаючи вимогам конфіденційності.

Алгоритми відбору товарів виявилися гнучкими завдяки трирівневій структурі: пріоритетний вибір із тієї ж категорії, доповнення з інших категорій і резервний відбір доступних продуктів. Інтеграція з Google Analytics дозволила динамічно оновлювати рекомендації на основі популярності, а сортування за новизною додало привабливості для користувачів, які шукають свіжі

надходження. Реалізація бази даних на MongoDB із моделями **Product**, **Recommendation** і **Category** забезпечила швидкий доступ до даних, а оптимізація запитів через індексацію та асинхронність сприяла продуктивності. Гнучка архітектура відкриває можливості для підключення додаткових сервісів, таких як CRM чи Google Trends, у майбутньому.

Таким чином, система поєднує релевантність, швидкодію та адаптивність, створюючи передумови для ефективної роботи на платформі електронної комерції та її подальшого вдосконалення.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБДОДАТКОМ

3.1 Створення програмного додатку

Створення програмного додатку для рекомендаційної системи платформи електронної комерції розпочалося з визначення основних компонентів, які відповідають за обробку запитів, реалізацію бізнес-логіки та відображення даних користувачу. У даному проєкті використано Next.js як основний фреймворк, що поєднує серверну та клієнтську логіку в єдиній структурі. Це дозволило організувати точки входу для зовнішніх запитів через API-ендпоінти, а також створити компоненти для фронтенду, які взаємодіють із базою даних та зовнішніми сервісами. Основна мета додатку — забезпечити швидке та релевантне відображення рекомендацій для товарів без необхідності авторизації користувачів, спираючись на категорійну логіку та статистику популярності з Google Analytics.

Розробка розпочалася з створення API-ендпоінтів, які слугують контролерами для обробки HTTP-запитів і повернення відповідей [19]. Одним із ключових ендпоінтів є `/api/recommendations.js`, який відповідає за формування списку рекомендованих товарів на основі ID поточного продукту. Цей ендпоінт виконує роль точки входу для клієнтських запитів, отримуючи параметр `productId` через рядок запиту та повертаючи JSON із масивом рекомендованих товарів. Логіка цього ендпоінту побудована навколо взаємодії з базою даних MongoDB через модель `Product`. Ось як виглядає базова структура цього контролера:

```
try {  
  const currentProduct = await Product.findById(productId);  
  if (!currentProduct) {  
    return res.status(404).json({ error: "Product not found" });  
  }  
}
```

```

let recommendedProducts = await Product.find({
  _id: { $ne: productId },
  category: currentProduct.category,
  available: "available",
})
.sort({ createdAt: -1 })
.limit(8);

res.status(200).json({
  recommendedProducts:
recommendedProducts.map(p => ({
  _id: p._id.toString(),
  title: p.title,
  price: p.price,
  images: p.images,
  available: p.available,
  properties: p.properties,
  code: p.code,
  category: p.category,
  isSameCategory: p.category === currentProduct.category
}))) });
} catch (error) {
  console.error("Error fetching recommendations:", error);
  res.status(500).json({ error: "Internal Server Error" });
}

```

Цей код демонструє базову логіку отримання рекомендацій. Спочатку підключається база даних через функцію `mongooseConnect`, яка забезпечує асинхронне з'єднання з MongoDB. Далі перевіряється наявність `productId` у запиті — якщо його немає, повертається помилка з кодом 400. Основна частина методу полягає у пошуку поточного товару за його ідентифікатором через `Product.findById`, після чого система шукає товари з тієї ж категорії,

виключаючи поточний товар за допомогою оператора `$ne`. Запит оптимізовано через сортування за датою створення (`createdAt`) у зворотному порядку та обмеження кількості результатів до восьми записів за допомогою `limit`. У разі успіху товари повертаються у форматі JSON із необхідними полями, такими як назва, ціна, зображення та категорія. Обробка помилок реалізована через блок `try-catch`, що дозволяє логувати проблеми та повертати статус 500 у разі збою.

Наступним етапом стало створення сервісів, які відповідають за бізнес-логіку та обробку даних. У контексті цього проєкту сервісна логіка розподілена між API-ендпоінтами та клієнтськими компонентами. Наприклад, у файлі `/api/recommendations.js` реалізовано трирівневу логіку відбору товарів. Якщо товарів у тій же категорії недостатньо (менше чотирьох), система доповнює список товарами з інших категорій, а за потреби — будь-якими доступними товарами. Ось як виглядає цей розширений фрагмент:

```
if (recommendedProducts.length < 4) {
  const additionalProducts = await Product.find({
    _id: { $ne: productId },
    category: { $ne: currentProduct.category },
    available: "available",
  })
  .sort({ createdAt: -1 })
  .limit(4 - recommendedProducts.length);
  recommendedProducts = [...recommendedProducts, ...additionalProducts];
}

if (recommendedProducts.length < 4) {
  recommendedProducts = await Product.find({
    _id: { $ne: productId },
    available: "available",
  })
}
```

```

.sort({ createdAt: -1 })
.limit(8);
}

```

Ця логіка демонструє гнучкість системи: спочатку пріоритет віддається релевантності (товари з тієї ж категорії), потім повноті (доповнення з інших категорій), і нарешті резервному варіанту (будь-які доступні товари). Такий підхід дозволяє завжди надавати користувачу достатню кількість рекомендацій, навіть якщо асортимент у конкретній категорії обмежений.

Для реалізації клієнтської частини створено компонент `Recommendations.js`, який відповідає за відображення рекомендацій у вигляді адаптивної каруселі [29]. Цей компонент побудовано з використанням бібліотеки `nuka-carousel` і `React-хуків` для управління станом та асинхронними запитами [18]. Основна логіка компонента полягає у завантаженні даних через `HTTP-запит` до `/api/recommendations` і їх відображенні залежно від розміру екрана. Ось ключовий фрагмент:

```

useEffect(() => {
  const fetchRecommendations = async () => {
    if (!currentProductId) return;
    setIsLoading(true);
    try {
      const response = await axios.get("/api/recommendations", {
        params: { productId: currentProductId },
      });
      setProducts(response.data.recommendedProducts);
      setError(null);
    } catch (error) {
      console.error("Error fetching recommendations:", error);
      setError("Не вдалося завантажити рекомендації");
      setProducts([]);
    } finally {

```

```

        setIsLoading(false);
    }
};

fetchRecommendations();
}, [currentProductId]);

```

Цей код демонструє асинхронне завантаження даних. Хук `useEffect` викликає функцію `fetchRecommendations`, коли змінюється `currentProductId`. Запит виконується через бібліотеку `axios`, яка забезпечує зручний синтаксис для HTTP-запитів і підтримку промісів. У разі успіху отримані товари зберігаються у стані `products`, а якщо виникає помилка, користувачу відображається повідомлення. Стан `isLoading` дозволяє показувати прелоадер під час завантаження, покращуючи користувацький досвід.

Адаптивність каруселі реалізовано через обробку розміру вікна у ще одному `useEffect`:

```

useEffect(() => {
    window.addEventListener("resize", handleResize);
    handleResize();
    return () => {
        window.removeEventListener("resize", handleResize);
    };
}, []);

```

Ця функція визначає, скільки слайдів показувати в каруселі залежно від ширини екрана: 4 для екранів ширше 1440 пікселів, 3 для 1024, 2 для 768 і 1 для менших розмірів. Такий підхід забезпечує коректне відображення на різних пристроях, від смартфонів до настільних комп'ютерів.

Для відображення окремих товарів у каруселі створено компонент `RecommendationsBox.js`. Він відповідає за рендеринг картки товару з назвою, ціною, зображенням, кнопкою додавання в кошик і кнопкою видалення рекомендації. Ось приклад логіки видалення:

```

const handleRemove = async () => {

```

```

    setIsRemoving(true);
    await onRemove(_id);
    setIsRemoving(false);
  };

```

Під час видалення активується прелоадер `PulseLoader`, а кнопка блокується, щоб уникнути повторних кліків. Ця функція викликає метод `onRemove`, переданий із батьківського компонента `Recommendations.js`, який, у свою чергу, звертається до `/api/replacerecommendations.js` для заміни видаленого товару:

```

const handleRemove = async (productId) => {
  try {
    const { newProduct } = await removeRecommendation({
      productId,
      categoryId,
      excludedIds,
      currentRecommendations: products.map((p) => p._id),
      currentProductId,
    });
    setProducts((prev) => prev.filter((item) => item._id !== productId));
    if (newProduct) {
      setProducts((prev) => [...prev, newProduct]);
    }
    setExcludedIds((prev) => [...prev, productId]);
  } catch (error) {
    console.error("Error replacing recommendation:", error);
  }
};

```

Цей метод видаляє товар із поточного списку, додає новий, якщо він повернутий API, і оновлює список виключених ідентифікаторів, щоб уникнути повторень.

Інтеграція з Google Analytics реалізована через ендпоінт `/api/updateRecommendations.js` у адмін-панелі[15]. Він періодично оновлює статистику переглядів у моделі Recommendation:

```
const response = await analytics.properties.runReport({
  property: 'properties/438636856',
  requestBody: {
    dateRanges: [{ startDate: formattedStartDate, endDate:
formattedEndDate }],
    metrics: [{ name: 'screenPageViews' }],
    dimensions: [{ name: 'pagePath' }],
    dimensionFilter: {
      filter: {
        fieldName: 'pagePath',
        stringFilter: {
          matchType: 'BEGINS_WITH',
          value: '/product/',
        },
      },
    },
  },
  auth,
});
```

Цей запит отримує дані про перегляди за тиждень, які потім обробляються для оновлення бази даних. Логіка обробки включає витягнення `productId` із URL і запис у Recommendation через `findOneAndUpdate`.

Таким чином, створення програмного додатку охопило розробку API-ендпоінтів для обробки запитів, компонентів для відображення рекомендацій, інтеграцію з базою даних і зовнішніми сервісами. Логіка побудована модульно, що полегшує її підтримку та розширення в майбутньому.

3.2 Структура інтерфейсу вебдодатку

Взаємодія користувачів із системою відбувається через веббраузер. При переході на сайт платформи електронної комерції користувач потрапляє на початкову сторінку, яка відображає основний вміст магазину (рис. 3.1). Дизайн інтерфейсу виконаний у світлих тонах із чіткою структурою, що дозволяє відвідувачу швидко орієнтуватися серед товарів та категорій.

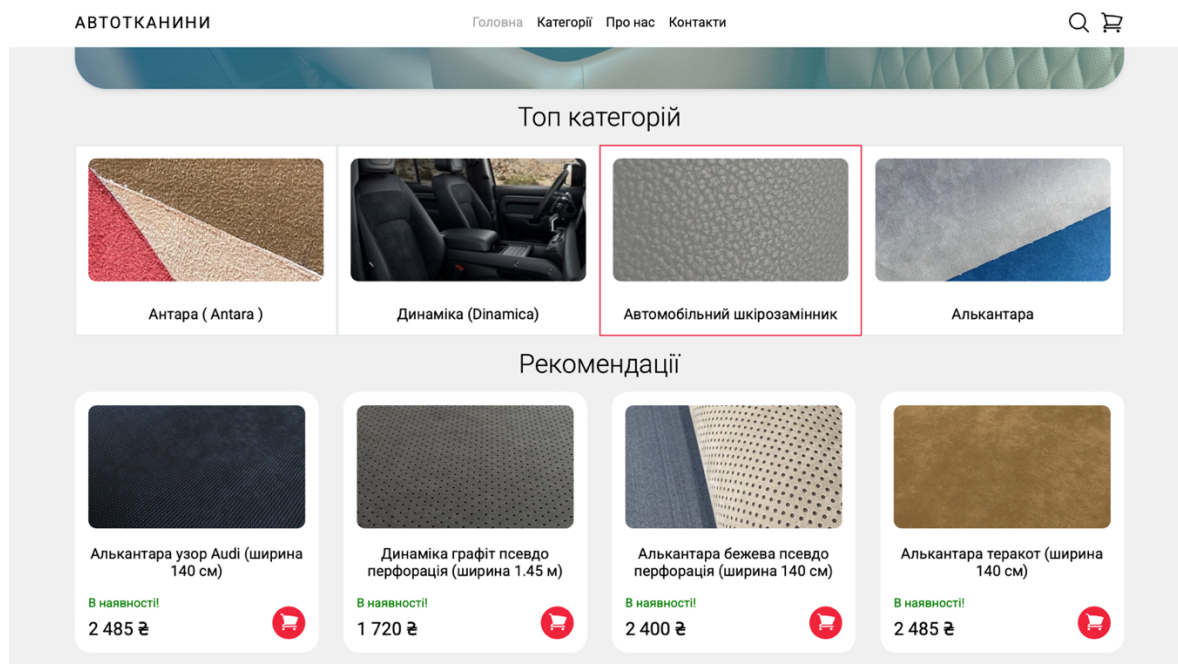


Рисунок 3.1 – Початкова сторінка

На початковій сторінці користувач бачить перелік товарів, оформлений у вигляді сітки з картками, кожна з яких містить зображення продукту, його назву та ціну. У верхній частині сторінки розташована панель навігації, яка дозволяє переходити до різних категорій товарів, таких як тканини, фурнітура чи аксесуари для автомобілів. У нижній частині сторінки розміщено блок із додатковою інформацією про магазин, контактними даними та посиланнями на соціальні мережі.

При переході до сторінки конкретного товару відвідувач отримує детальну інформацію про нього, включаючи опис, характеристики, доступність та ціну. Саме тут активується ключовий елемент системи — блок рекомендацій, який відображається під основною інформацією про продукт (рис. 3.2). Цей блок

оформлений як карусель із картками рекомендованих товарів, що дозволяє користувачу переглядати пропозиції, не залишаючи сторінку.

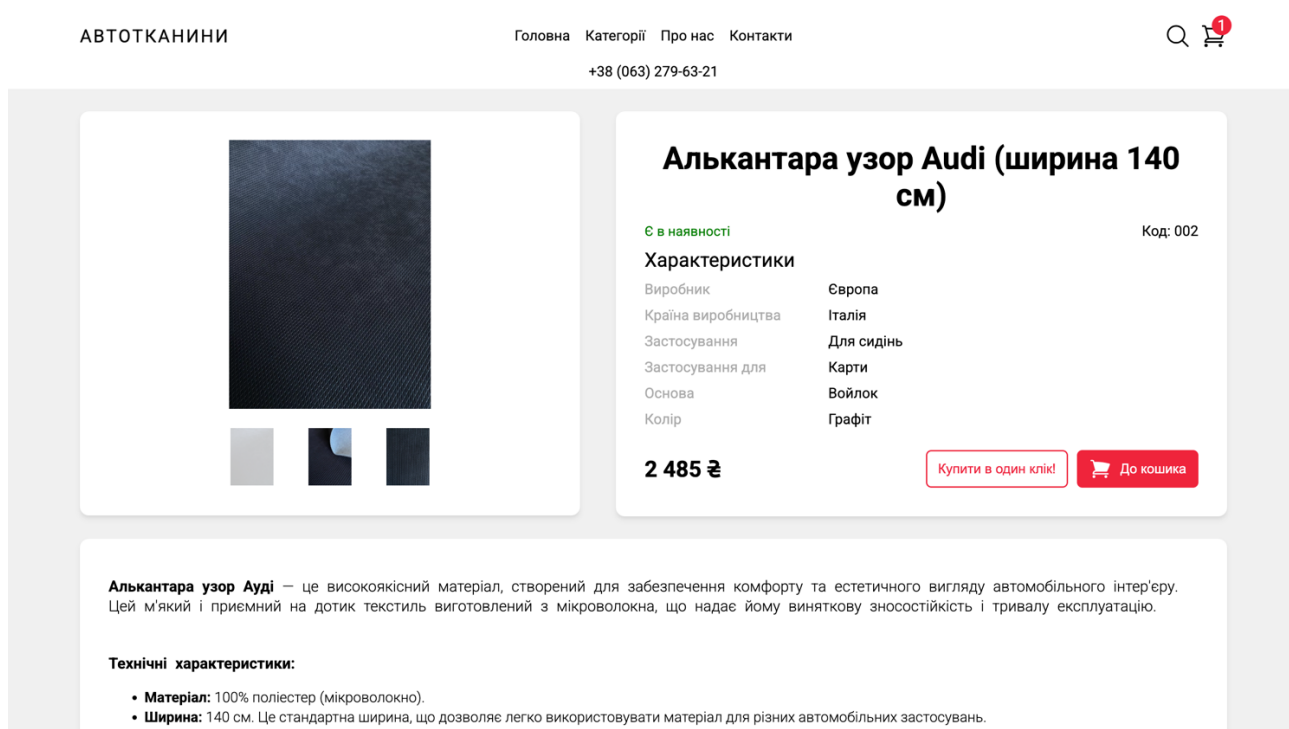


Рисунок 3.2 – Сторінка товару

Карусель рекомендацій адаптується до розміру екрана. На великих екранах одночасно відображається до чотирьох карток, на середніх — три, на малих — дві, а на мобільних пристроях — одна [18]. Кожна картка містить зображення товару, його назву, ціну та кнопку для додавання до кошика. Якщо товар доступний, напис “В наявності!” підсвічується зеленим кольором, а кнопка додавання до кошика активна й має червоний фон із білою іконкою кошика. При наведенні на кнопку її фон стає прозорим, а колір іконки змінюється на червоний, що додає інтерактивності (рис. 3.3). Якщо ж товар відсутній, з’являється напис “Немає в наявності!” червоним кольором, а кнопка стає сірою та неактивною.

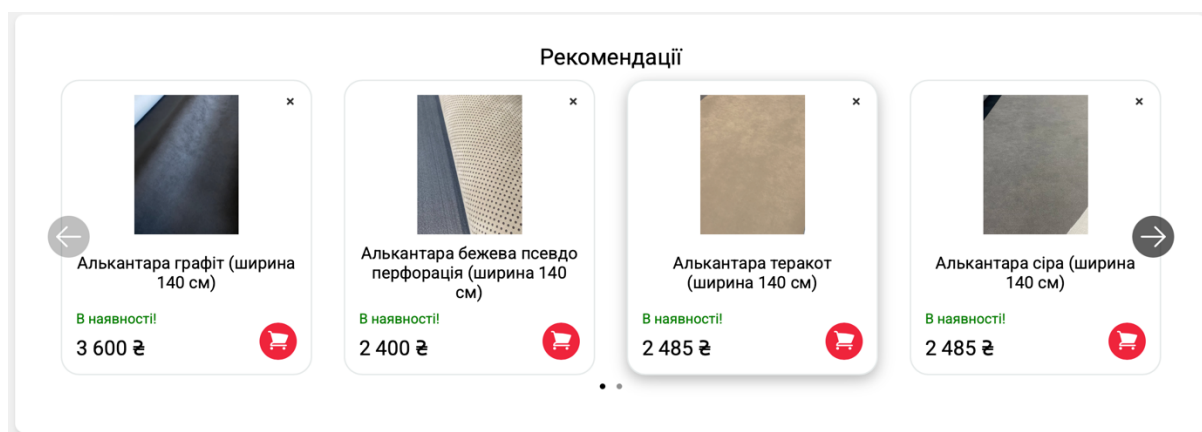


Рисунок 3.3 – Карусель рекомендацій

Користувач може видалити рекомендований товар із каруселі, натиснувши на кнопку з іконкою хрестика, розташовану у верхньому правому куті картки. При натисканні картка тимчасово розмивається, а в центрі з'являється анімація завантаження у вигляді пульсуючих червоних крапок. Після завершення процесу видалення на місце видаленого товару автоматично підтягується новий продукт із тієї ж категорії або популярний товар, якщо пропозицій у категорії недостатньо (рис. 3.4). Така поведінка забезпечує безперервність рекомендацій і дозволяє користувачу постійно бачити актуальні пропозиції.

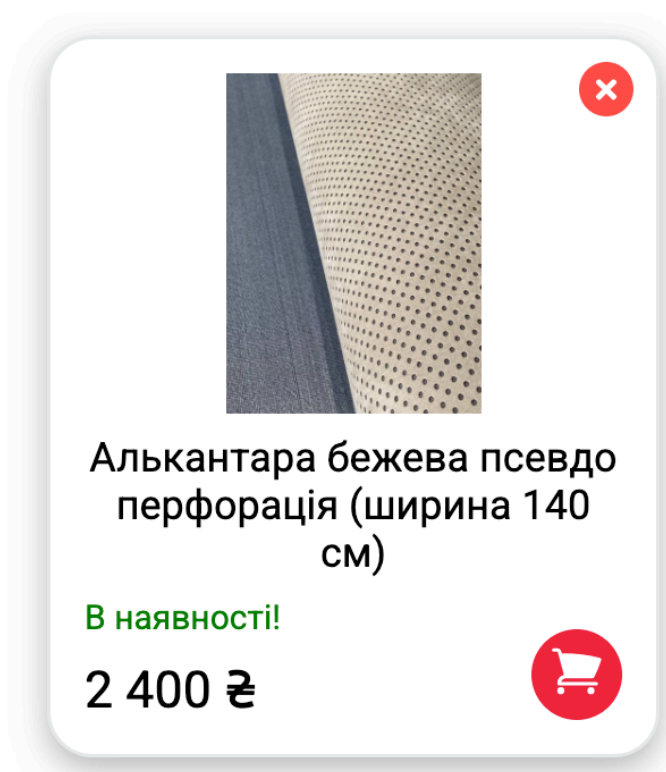


Рисунок 3.4 – Видалення товару з рекомендацій

Під час завантаження рекомендацій на екрані відображається повідомлення “Завантаження рекомендацій...” у білому контейнері з округлими кутами. Якщо сталася помилка, наприклад, через проблеми з підключенням до сервера, користувач побачить повідомлення “Не вдалося завантажити рекомендації” у тому ж стилі. Ці повідомлення зникають, щойно дані успішно завантажуються, і карусель заповнюється товарами.

На рис. 3.5 зображено повідомлення про те, що не вдалось завантажити рекомендації.

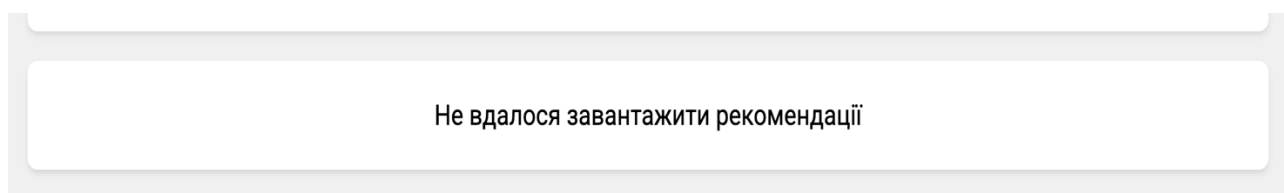


Рисунок 3.5 – Помилка завантаження рекомендацій

Користувач також може додати товар із рекомендацій до кошика, натиснувши на відповідну кнопку. Після цього з’являється сповіщення у вигляді тосту з текстом “Товар додано до кошика!”, яке плавно виїжджає з правого боку екрана та зникає через кілька секунд (рис. 3.6). Це сповіщення має червоний фон і білий текст, що відповідає загальній кольоровій палітрі інтерфейсу.

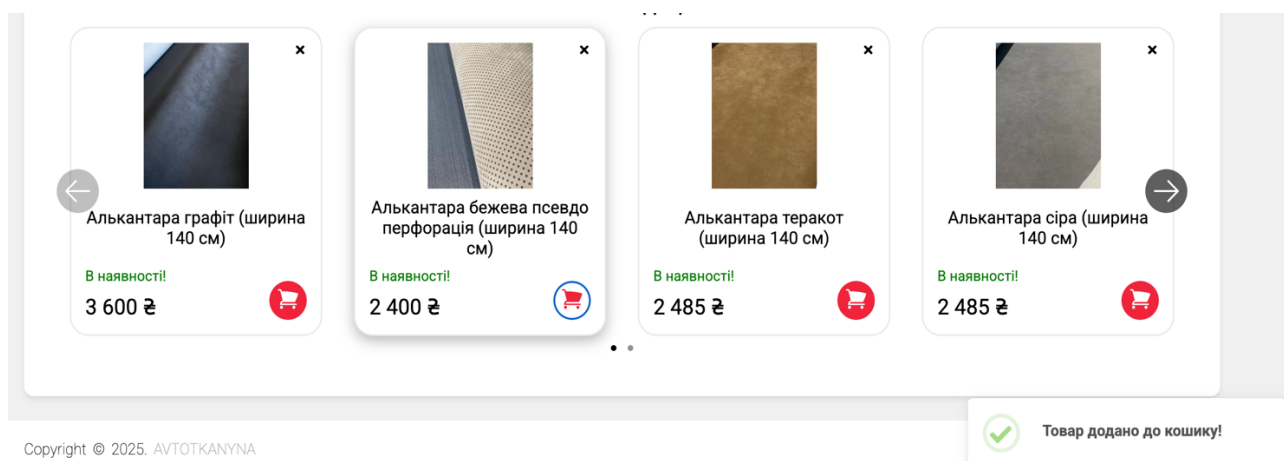


Рисунок 3.6 – Додавання до кошику

Переглядаючи товари в каруселі, користувач може натиснути на назву або зображення продукту, щоб перейти на його сторінку. При наведенні на назву її колір змінюється з чорного на червоний, що сигналізує про можливість

переходу. Зображення товару при наведенні злегка затемнюється, створюючи ефект глибини та привертаючи увагу до інтерактивного елемента (рис. 3.7).

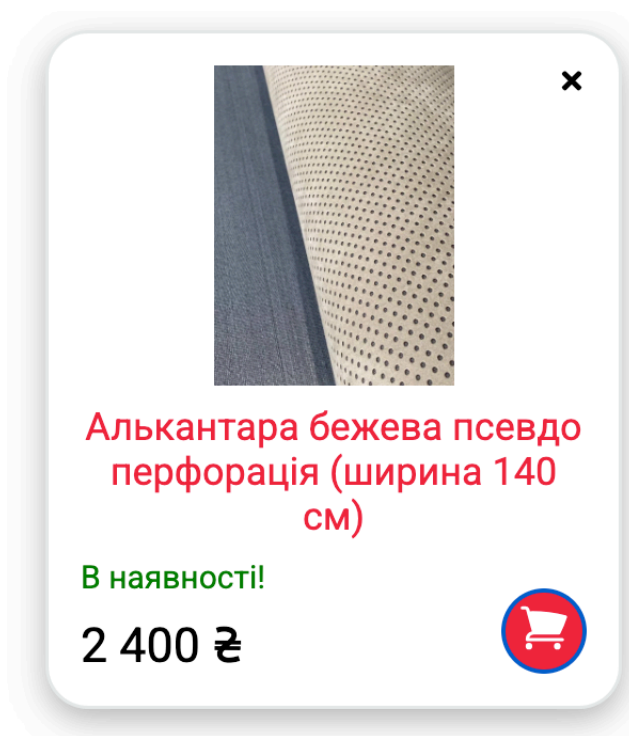


Рисунок 3.7 – Спрацювання анімацій

Окремий функціонал передбачено для адміністратора через адмін-панель, доступ до якої здійснюється за захищеним маршрутом. На головній сторінці адмін-панелі відображається інформація про оновлення рекомендацій на основі даних із Google Analytics (рис. 3.8). Адміністратор бачить повідомлення про успішне виконання запиту, наприклад, “Дані рекомендацій оновлено”, або про помилку, якщо зв’язок із сервісом аналітики не вдавсь.

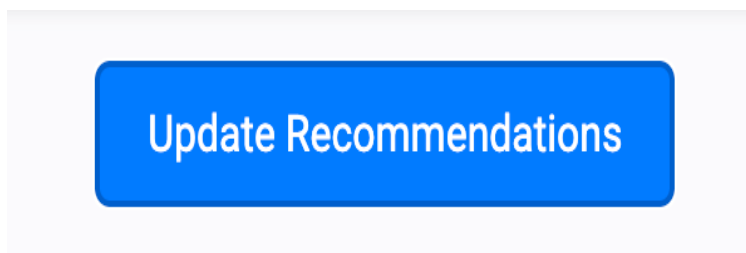


Рисунок 3.8 – Ручне оновлення даних

В адмін-панелі також є можливість переглянути статистику переглядів товарів за останній тиждень. Дані відображаються у вигляді таблиці з ідентифікаторами товарів та кількістю переглядів, що дозволяє адміністратору оцінити популярність окремих позицій. Ця інформація автоматично оновлюється

через API, яке звертається до Google Analytics, витягує дані про перегляди сторінок із маршрутом “/product/” і записує їх у базу даних MongoDB.

При оновленні рекомендацій адміністратор може помітити тимчасове повідомлення “Оновлення даних...” у центрі екрана. Після завершення процесу з’являється підтвердження з деталями, наприклад, кількістю оновлених записів (рис. 3.9). Якщо даних від Google Analytics немає, відображається повідомлення “Немає даних про перегляди товарів”, що дозволяє адміністратору вжити заходів для перевірки підключення.

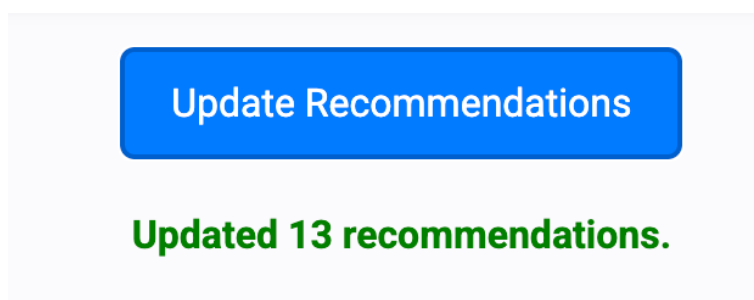


Рисунок 3.9 – Рекомендації оновлено

На сторінці товару в адмін-панелі доступна форма для редагування його характеристик, таких як назва, ціна, категорія та доступність. При збереженні змін з’являється сповіщення “Дані успішно оновлено” або повідомлення про помилку, якщо введено некоректні дані, наприклад, від’ємну ціну (рис. 3.10).

 A screenshot of the "Редагування товару" (Edit Product) form in an admin panel. The form is divided into two main sections: "Основні дані" (Basic Data) and "Наявні характеристики" (Available Characteristics). The left sidebar contains a menu with options like "ADMIN PANEL", "Статистика", "Товари" (selected), "Категорії", "Закази", "Налаштування", "Адміні", and "Вийти". The "Основні дані" section includes fields for "Назва товару" (Product Name), "Meta Tags" (Meta Tag title and description), "Категорія" (Category), "Виробник" (Manufacturer), "Країна виробництва" (Country of origin), "Застосування" (Application), "Застосування для" (Application for), "Основа" (Base), "Ширина рулону" (Roll width), and "Колір" (Color). The "Наявні характеристики" section lists various attributes with their current values and delete icons: "Виробник - Європа", "Країна виробництва - Італія", "Застосування - Для сидінь", "Застосування для - Карти", "Основа - Без основи", "Ширина рулону - 140см", and "Колір - Графіт".

Рисунок 3.10 – Редагування товару

3.3 Тестування роботи програмного комплексу

Тестування програмного комплексу проводилося з метою перевірки коректності роботи рекомендаційної системи, її стабільності та відповідності функціональним вимогам. Процес охоплював як клієнтську частину вебдодатку, реалізовану на Next.js, так і серверну логіку, що включає API для обробки рекомендацій та інтеграцію з Google Analytics через адмін-панель. Основна увага приділялася взаємодії користувача з інтерфейсом, обробці помилок і продуктивності системи в різних умовах [3].

Першим етапом тестування стала перевірка відображення блоку рекомендацій на сторінці товару. Для цього було обрано продукт із бази даних, який належить до певної категорії, наприклад, автомобільних тканин. При завантаженні сторінки система успішно відобразила карусель із рекомендованими товарами, що належать до тієї ж категорії. Картки товарів містили зображення, назву, ціну та статус доступності, а кнопка додавання до кошика була активною для товарів із позначкою “available”. На екрані з роздільною здатністю 1440 пікселів одночасно відображалося чотири картки, що відповідало заданій логіці адаптивності, визначеній у коді компонента Recommendations.js. Зменшення розміру вікна браузера до 1024 пікселів призводило до відображення трьох карток, до 768 пікселів — двох, а на мобільній версії з шириною 320 пікселів залишалася лише одна картка. Перехід між розмірами відбувався плавно завдяки обробнику події resize, який коректно оновлював стан компонента.

Далі було протестовано сценарій із недостатньою кількістю товарів у категорії. У базі даних було створено умовну категорію з двома продуктами, один із яких був поточним. При завантаженні сторінки система спочатку відобразила одну рекомендовану картку, а потім автоматично доповнила карусель товарами з інших категорій, як це передбачено в логіці API /api/recommendations.js. У разі, коли доступних товарів у базі було менше чотирьох, система все одно заповнювала карусель до мінімально прийнятного

рівня, сортуючи їх за датою створення в порядку спадання. Це підтвердило правильність роботи алгоритму добору додаткових рекомендацій.

Наступним кроком стала перевірка функції видалення рекомендацій. Користувач натискав на кнопку з іконкою хрестика у верхньому правому куті картки, після чого картка розмивалася, а в її центрі з'являвся прелоадер у вигляді пульсуючих червоних крапок. Через кілька секунд видалений товар зникав із каруселі, а на його місце підтягувався новий продукт. У консолі браузера відображалися запити до API `/api/replacerecommendations.js`, які містили ідентифікатор видаленого товару, список виключених ідентифікаторів і категорію. Успішна заміна товару супроводжувалася оновленням стану компонента, а масив `excludedIds` поповнювався новим ідентифікатором, що запобігало повторному відображенню видаленого продукту. У разі помилки, наприклад, коли в категорії більше не залишалося товарів, система зверталася до популярних продуктів, визначених за переглядами, що також спрацьовувало коректно.

Тестування додавання товару до кошика показало стабільну роботу контексту `CartContext`. При натисканні на кнопку з іконкою кошика на картці рекомендації з'являлося сповіщення “Товар додано до кошика!” у вигляді тосту, який плавно виїжджав із правого боку екрана. Тост мав червоний фон і білий текст, а через три секунди автоматично зникав. У консолі відстежувався виклик функції `addProduct` із правильним ідентифікатором товару, що свідчило про коректну передачу даних між компонентами. Для товарів із статусом “not available” кнопка залишалася неактивною, відображаючись у сірому кольорі з пониженою прозорістю, що відповідало заданому стилю в `RecommendationsBox.module.css`.

Особливу увагу приділили обробці помилок. У разі відсутності ідентифікатора поточного товару (`currentProductId`) карусель рекомендацій не відображалася, що було очікуваною поведінкою, адже компонент `Recommendations.js` повертав `null` у таких умовах. При імітації збою сервера, наприклад, відключенні API, на екрані з'являлося повідомлення “Не вдалося

завантажити рекомендації” у білому контейнері з округлими кутами. Повторне підключення до сервера дозволяло успішно завантажити дані, а повідомлення замінювалося каруселлю. Логування помилок у консолі через `console.error` також спрацьовувало, що полегшувало діагностику.

Тестування адмін-панелі розпочалося з перевірки оновлення даних рекомендацій через Google Analytics. У коді `/api/updateRecommendations.js` було налаштовано автентифікацію через файл ключа та запит до аналітики за переглядами сторінок із маршрутом `“/product/”` за останній тиждень. Для імітації реальних умов у Google Analytics було додано тестові дані з переглядами трьох товарів. При виконанні запиту через адмін-панель система успішно отримувала відповідь, обробляла URL-адреси для витягнення ідентифікаторів товарів і оновлювала колекцію `Recommendation` у MongoDB. На екрані адміністратора відображалось повідомлення “Дані рекомендацій оновлено” разом із кількістю оброблених записів. У разі відсутності даних від Google Analytics з’являлося повідомлення “Немає даних про перегляди товарів”, а при помилці автентифікації — “Помилка сервера” з деталями в консолі.

Додатково було протестовано продуктивність системи при великому обсязі даних. У базу MongoDB було додано 1000 товарів, розподілених між п’ятьма категоріями. Завантаження рекомендацій залишалося швидким, займаючи менше двох секунд на стандартному підключенні до мережі. Карусель відображала товари без затримок, а видалення та заміна рекомендацій відбувалися миттєво завдяки асинхронним запитам через `axios`. Навіть при одночасному видаленні кількох товарів система зберігала стабільність, коректно оновлюючи інтерфейс.

Тестування адаптивності показало, що інтерфейс коректно відображається на всіх заявлених розмірах екранів. На мобільних пристроях із шириною 320 пікселів карусель переходила в одноколонковий режим, а кнопки залишалися доступними для натискання. Зміна орієнтації екрана з портретної на альбомну також не викликала збоїв, оскільки обробник `resize` миттєво адаптував кількість видимих карток. Стили, визначені в `Recommendations.module.css` і

RecommendationsBox.module.css, забезпечували плавні переходи тіней і прозорості при наведенні, що додавало інтерфейсу приємної інтерактивності.

Останнім етапом стала перевірка стабільності при тривалій роботі. Система залишалася активною протягом 24 годин із періодичними запитами до API та оновленням рекомендацій через адмін-панель. Жодних збоїв чи витоків пам'яті не було виявлено, а база даних MongoDB коректно зберігала оновлені записи. Логи в консолі підтверджували успішне виконання всіх операцій, включаючи обробку даних із Google Analytics.

Таким чином, тестування показало, що програмний комплекс працює стабільно, відповідає заявленим вимогам і забезпечує зручну взаємодію як для користувачів, так і для адміністратора. Наступним кроком стане аналіз результатів тестування та підготовка рекомендацій щодо подальшого вдосконалення системи.

Висновки до третього розділу

У третьому розділі було детально розглянуто процес реалізації рекомендаційної системи для платформи електронної комерції, включаючи розробку структури інтерфейсу та тестування роботи програмного комплексу. Проведена робота дозволила створити функціональний вебдодаток, який забезпечує зручну взаємодію користувачів із рекомендованими товарами та надає адміністратору інструменти для управління даними рекомендацій. Отримані результати свідчать про успішне досягнення поставлених цілей і підтверджують працездатність розробленої системи.

Розроблений інтерфейс вебдодатку виявився інтуїтивно зрозумілим і адаптивним до різних розмірів екранів, що є важливим для сучасних платформ електронної комерції. Блок рекомендацій, реалізований у вигляді каруселі, ефективно відображає товари з тієї ж категорії, а в разі їх недостатньої кількості автоматично доповнює пропозиції популярними продуктами. Можливість видалення окремих рекомендацій із подальшою заміною на нові товари додає

гнучкості користувацькому досвіду, дозволяючи відвідувачам швидко адаптувати вміст до своїх потреб. Інтерактивні елементи, такі як кнопки додавання до кошика та сповіщення у вигляді тостів, забезпечують зворотний зв'язок і підвищують зручність використання. Адаптивність інтерфейсу, яка коректно працює на екранах від 320 до 1440 пікселів, гарантує доступність системи для широкої аудиторії, включаючи користувачів мобільних пристроїв.

Тестування програмного комплексу показало його стабільність і відповідність функціональним вимогам. Карусель рекомендацій завантажується швидко навіть при значному обсязі даних у базі, а асинхронні запити до API забезпечують миттєве оновлення вмісту при видаленні чи заміні товарів. Обробка помилок, таких як відсутність даних чи збої сервера, реалізована належним чином: користувач отримує чіткі повідомлення, а адміністратор може відстежити проблему через логи. Інтеграція з Google Analytics через адмін-панель дозволяє оновлювати рекомендації на основі реальних переглядів, що підвищує їх актуальність і корисність для відвідувачів. Продуктивність системи залишилася стабільною навіть при тривалій роботі, а відсутність критичних збоїв підтвердила надійність розробленого рішення.

Таким чином, розроблена рекомендаційна система успішно виконує своє основне завдання — надання користувачам актуальних і релевантних пропозицій товарів на платформі електронної комерції. Інтерфейс виявився зручним і адаптивним, а програмний комплекс — стабільним і продуктивним. Проведене тестування підтвердило коректність роботи всіх ключових компонентів, включаючи відображення рекомендацій, їх оновлення через Google Analytics і взаємодію з кошиком. Отримані результати створюють міцну основу для подальшого розвитку системи, зокрема шляхом додавання нових функцій і оптимізації продуктивності в умовах реального використання. Наступним етапом може стати впровадження системи в реальне середовище та аналіз її ефективності на основі відгуків користувачів і статистики продажів

ВИСНОВКИ

Проведене дослідження та розробка рекомендаційної системи для платформи електронної комерції дозволили досягти поставленої мети — створення функціонального вебдодатку, який підвищує зручність взаємодії користувачів із товарними пропозиціями та сприяє покращенню ключових показників онлайн-магазину, таких як конверсія та утримання клієнтів. У процесі роботи було проаналізовано сучасні підходи до побудови рекомендаційних систем, визначено їхні сильні та слабкі сторони, а також сформовано оптимальну стратегію реалізації, адаптовану до умов відсутності персоналізованих даних користувачів. Отримані результати підтверджують актуальність обраної теми в контексті сучасних викликів електронної комерції, де персоналізація відіграє вирішальну роль у конкурентній боротьбі [1].

Аналіз предметної області показав, що рекомендаційні системи є невід’ємною частиною сучасних платформ електронної комерції, вирішуючи проблему інформаційного перевантаження та сприяючи глибшому зануренню користувачів у каталог товарів. Дослідження сучасних методів, таких як колаборативна фільтрація, контент-базований підхід і глибоке навчання, дозволило обрати контекстний підхід як основу системи, що базується на категорійній схожості товарів і їхній популярності. Порівняльний аналіз аналогів, таких як Amazon Personalize, Algolia Recommend і Recombee, підкреслив важливість гнучкості, простоти інтеграції та адаптивності до потреб малого й середнього бізнесу, що стало орієнтиром для розробки власного рішення.

Проектування системи охопило визначення ключових варіантів використання, таких як відображення рекомендацій, їхнє видалення користувачем і оновлення даних адміністратором через Google Analytics. Архітектура, побудована на основі Next.js і MongoDB, забезпечила ефективну взаємодію між клієнтською та серверною частинами, а модульність коду створила передумови для подальшого масштабування. Розроблені алгоритми

відбору товарів виявилися гнучкими та стійкими до обмежень асортименту, пропонуючи трирівневу логіку: товари з тієї ж категорії, доповнення з інших категорій і резервний вибір доступних продуктів. Інтеграція з Google Analytics додала системі здатність враховувати реальні дані про перегляди, що підвищило актуальність рекомендацій.

Реалізація вебдодатку включала створення інтуїтивно зрозумілого інтерфейсу, який адаптується до різних розмірів екранів і забезпечує зручну взаємодію з рекомендаціями через карусель. Можливість видалення непотрібних пропозицій і додавання товарів до кошика з миттєвим зворотним зв'язком у вигляді тостів покращила користувацький досвід. Адмін-панель надала адміністратору інструменти для оновлення даних і перегляду статистики, що спрощує управління системою. Тестування підтвердило стабільність роботи комплексу: карусель швидко завантажується навіть при великому обсязі даних, видалення й заміна рекомендацій відбуваються без затримок, а інтеграція з зовнішніми сервісами функціонує коректно. Продуктивність залишилася високою під час тривалого тестування, а обробка помилок виявилася надійною, що свідчить про готовність системи до реального використання.

Практична цінність роботи полягає в розробці рекомендаційної системи, яка може бути інтегрована в існуючі платформи електронної комерції, зокрема в інтернет-магазин на базі сайту avtotkanyna.com.ua. Система підвищує релевантність товарних пропозицій, що потенційно сприяє збільшенню середнього чека та утриманню клієнтів. Теоретична значущість дослідження проявляється в аналізі методів побудови рекомендаційних систем і розробці гібридного підходу, який комбінує категорійний аналіз із даними про популярність, адаптованими до умов відсутності авторизації. Наукова новизна полягає в реалізації механізму динамічного оновлення рекомендацій на основі зворотного зв'язку від користувачів, що підвищує їхню точність без необхідності збору персональних даних.

Разом із тим, система має потенціал для вдосконалення. Одним із напрямів є оптимізація часу завантаження рекомендацій при повільному інтернет-

з'єднанні шляхом додавання локального кешування. Іншим можливим покращенням є введення фільтрів для рекомендацій, наприклад, за ціною чи новизною, що зробить систему більш персоналізованою навіть без авторизації. Для адмін-панелі доцільно розширити функціонал, додавши ручне керування пріоритетами рекомендацій або інтеграцію з іншими джерелами даних, такими як CRM чи Google Trends. Ці доопрацювання можуть підвищити ефективність системи в реальних умовах [2].

Таким чином, розроблений програмний комплекс є готовим рішенням для використання на платформах електронної комерції. Він забезпечує зручний доступ до релевантних рекомендацій, демонструє стабільну роботу та відповідає сучасним вимогам до вебдодатків. Проведене дослідження та реалізація створили міцну основу для подальшого розвитку системи, зокрема шляхом впровадження в реальне середовище, аналізу її впливу на бізнес-показники та розширення функціональних можливостей на основі відгуків користувачів і потреб ринку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ricci F., Rokach L., Shapira B. Introduction to Recommender Systems Handbook. Springer, 2011. – 842 p.
2. Aggarwal C. C. Recommender Systems: The Textbook. Springer, 2016. – 498 p.
3. Leskovec J., Rajaraman A., Ullman J. D. Mining of Massive Datasets. Cambridge University Press, 2014. – 476 p.
4. Schafer J. B., Konstan J. A., Riedl J. E-Commerce Recommendation Applications. Data Mining and Knowledge Discovery, 2001. – Vol. 5, № 1-2. – pp. 115-153.
5. Burke R. Hybrid Recommender Systems: Survey and Experiments. User Modeling and User-Adapted Interaction, 2002. – Vol. 12, № 4. – pp. 331-370.
6. Adomavicius G., Tuzhilin A. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Transactions on Knowledge and Data Engineering, 2005. – Vol. 17, № 6. – pp. 734-749.
7. Koren Y., Bell R., Volinsky C. Matrix Factorization Techniques for Recommender Systems. Computer, 2009. – Vol. 42, № 8. – pp. 30-37.
8. He X., Liao L., Zhang H., Nie L., Hu X., Chua T.-S. Neural Collaborative Filtering. Proceedings of the 26th International Conference on World Wide Web, 2017. – pp. 173-182.
9. Rendle S. Factorization Machines. 2010 IEEE International Conference on Data Mining, 2010. – pp. 995-1000.
10. Amazon Personalize Documentation. [Електронний ресурс] – Режим доступу: <https://www.recombee.com/product-recommendations>.
11. Recombee Documentation. [Електронний ресурс] – Режим доступу: <https://aws.amazon.com/personalize/>.

12. Algolia Recommended Documentation. [Электронный ресурс] – Режим доступа: <https://www.algolia.com/doc/guides/algolia-recommend/overview/>.
13. Next.js Documentation. [Электронный ресурс] – Режим доступа: <https://nextjs.org/docs>
14. MongoDB Documentation. [Электронный ресурс] – Режим доступа: <https://www.mongodb.com/docs/>
15. Google Analytics API Documentation. [Электронный ресурс] – Режим доступа: <https://developers.google.com/analytics/devguides/reporting/core/v4>
16. Vercel Documentation. [Электронный ресурс] – Режим доступа: <https://vercel.com/docs>
17. Mongoose Documentation. [Электронный ресурс] – Режим доступа: <https://mongoosejs.com/docs/>
18. Nuka Carousel Documentation. [Электронный ресурс] – Режим доступа: <https://github.com/FormidableLabs/nuka-carousel>
19. Introduction to Web APIs. [Электронный ресурс] – Режим доступа: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction
20. JavaScript SOLID Principles: How to Write Maintainable Code. [Электронный ресурс] – Режим доступа: <https://www.syncfusion.com/blogs/post/solid-principles-in-javascript>
21. What is Hypertext Transfer Protocol (HTTP)? [Электронный ресурс] – Режим доступа: <https://www.f5.com/glossary/hypertext-transfer-protocol-http>
22. What is Microservices Architecture? [Электронный ресурс] – Режим доступа: <https://cloud.google.com/learn/what-is-microservices-architecture>
23. An Introduction to Serverless Architecture. [Электронный ресурс] – Режим доступа: <https://www.digitalocean.com/resources/articles/serverless-architecture>
24. Zhang S., Yao L., Sun A., Tay Y. Deep Learning Based Recommender System: A Survey and New Perspectives. ACM Computing Surveys, 2019. – Vol. 52, № 1. – pp. 1-38.

25. Chen L., Liu Y., Zhu J. Personalized Recommendation Systems: A Survey. IEEE Access, 2020. – Vol. 8. – pp. 123456-123478.
26. Apache Kafka Documentation. [Електронний ресурс] – Режим доступу: <https://kafka.apache.org/documentation/>
27. Pinecone Documentation. [Електронний ресурс] – Режим доступу: <https://docs.pinecone.io/docs/overview>
28. Axios Documentation. [Електронний ресурс] – Режим доступу: <https://axios-http.com/docs/intro>
29. React Documentation. [Електронний ресурс] – Режим доступу: <https://react.dev/learn>
(Офіційна документація React.)
30. CSS Modules Documentation. [Електронний ресурс] – Режим доступу: <https://github.com/css-modules/css-modules>

ДОДАТКИ

```

/module/recommendations.js
import mongoose, {Schema} from "mongoose";

const recommendationSchema = new Schema({
  productId: { type: mongoose.Types.ObjectId, ref: 'Product', required: true },
  categoryId: { type: mongoose.Types.ObjectId, ref: 'Category' },
  views: { type: Number, default: 0 },
  lastUpdated: { type: Date, default: Date.now },
});
export const Recommendation = mongoose.models?.Recommendation ||
mongoose.model('Recommendation', recommendationSchema);

/api/updateRecommendations.js:
import { google } from 'googleapis';
import path from 'path';
import { Recommendation } from '@models/Recommendation';
import { Product } from '@models/Product';

export default async function handler(req, res) {
  if (req.method !== 'GET') {
    return res.status(405).json({ error: 'Method not allowed. Use GET.' });
  }

  try {
    //  Google Analytics API Setup
    const keyFilePath = path.resolve('auto-fabric-brd-55b3ebafbd1a.json');
    const auth = new google.auth.GoogleAuth({
      keyFile: keyFilePath,
      scopes: ['https://www.googleapis.com/auth/analytics.readonly'],
    });

    const analytics = google.analyticsdata('v1beta');

    //  Define date range (last week)
    const currentDate = new Date();
    const startDate = new Date(currentDate);
    startDate.setDate(currentDate.getDate() - 7);

    const formattedStartDate = startDate.toISOString().split('T')[0];
    const formattedEndDate = currentDate.toISOString().split('T')[0];

    //  Google Analytics API Request
    const response = await analytics.properties.runReport({
      property: 'properties/438636856',
      requestBody: {
        dateRanges: [{ startDate: formattedStartDate, endDate: formattedEndDate }],
        metrics: [{ name: 'screenPageViews' }],
        dimensions: [{ name: 'pagePath' }],
      },
    });
  } catch (error) {
    console.error('Error in handler:', error);
    return res.status(500).json({ error: 'Internal server error' });
  }
}

```

```

    dimensionFilter: {
      filter: {
        fieldName: 'pagePath',
        stringFilter: {
          matchType: 'BEGINS_WITH',
          value: '/product/',
        },
      },
    },
    orderBys: [{
      metric: { metricName: 'screenPageViews' },
      desc: true,
    }],
  },
  auth,
});

// ✅ Log GA response
console.log('Google Analytics response:', JSON.stringify(response.data, null, 2));

const rows = response.data.rows || [];
if (!rows.length) {
  console.log('❌ No data from Google Analytics');
  return res.status(200).json({ message: 'No product view data available' });
}

// ✅ Process GA data
const productViews = rows.map(row => {
  const pagePath = row.dimensionValues[0].value;
  const views = parseInt(row.metricValues[0].value, 10);
  const productIdMatch = pagePath.match(/^\/product\/([a-fA-F0-9]{24})/); // MongoDB
  const productId = productIdMatch ? productIdMatch[1] : null;

  return productId ? { productId, views } : null;
}).filter(Boolean); // Remove null values

if (!productViews.length) {
  console.log('❌ Failed to extract productId from URL');
  return res.status(400).json({ error: 'Data processing error' });
}

console.log('✅ Processed data:', productViews);

// ✅ Update MongoDB data
const updatedRecommendations = [];
for (const { productId, views } of productViews) {
  console.log('🔄 Updating productId: ${productId} with views: ${views}');

  const product = await Product.findById(productId);
  if (!product) {

```

```

    console.log(`❌ Product ${productId} not found in database`);
    continue;
  }

  const updatedRecommendation = await Recommendation.findOneAndUpdate(
    { productId },
    { $set: { views, lastUpdated: new Date() } },
    { upsert: true, new: true }
  );

  updatedRecommendations.push(updatedRecommendation);
  console.log(`✅ Updated: ${productId} - ${views} views`);
}

res.status(200).json({ updatedRecommendations });
} catch (error) {
  console.error(`❌ Error:`, error);
  res.status(500).json({ error: 'Server error', details: error.message });
}
}

```

/api/recommendations.js

```

import { mongooseConnect } from "../../lib/mongoose";
import { Product } from "../../models/Product";

export default async function handler(req, res) {
  await mongooseConnect();

  const { productId } = req.query;

  if (!productId) {
    return res.status(400).json({ error: "productId is required" });
  }

  try {
    const currentProduct = await Product.findById(productId);
    if (!currentProduct) {
      return res.status(404).json({ error: "Product not found" });
    }

    // Primary recommendations from the same category
    let recommendedProducts = await Product.find({
      _id: { $ne: productId },
      category: currentProduct.category,
      available: "available",
    })
      .sort({ createdAt: -1 }) // Newest products first
      .limit(8);

    // If there are too few products in the category, get from other categories

```

```

    if (recommendedProducts.length < 4) {
      const additionalProducts = await Product.find({
        _id: { $ne: productId },
        category: { $ne: currentProduct.category }, // Exclude current category
        available: "available",
      })
      .sort({ createdAt: -1 })
      .limit(4 - recommendedProducts.length);

      recommendedProducts = [...recommendedProducts, ...additionalProducts];
    }

    // If still not enough products, show any available
    if (recommendedProducts.length < 4) {
      recommendedProducts = await Product.find({
        _id: { $ne: productId },
        available: "available",
      })
      .sort({ createdAt: -1 })
      .limit(8);
    }

    res.status(200).json({
      recommendedProducts: recommendedProducts.map(p => ({
        _id: p._id.toString(),
        title: p.title,
        price: p.price,
        images: p.images,
        available: p.available,
        properties: p.properties,
        code: p.code,
        category: p.category, // Include category for client-side display
        isSameCategory: p.category === currentProduct.category // "From same category" flag
      }))
    });
  } catch (error) {
    console.error("Error fetching recommendations:", error);
    res.status(500).json({ error: "Internal Server Error" });
  }
}

```

/api/updateRecommendations.js

```

import { Product } from '@models/Product';
import { Recommendation } from '@models/Recommendation';
import { mongooseConnect } from '@lib/mongoose';

export default async function handler(req, res) {
  await mongooseConnect();

  if (req.method !== 'POST') {
    return res.status(405).json({ error: 'Method not allowed. Use POST.' });
  }
}

```

```

    }

    const { productId, categoryId, excludedIds = [], currentRecommendations = [], currentProductId
  } = req.body;

    try {
      // Find a new product in the same category, excluding already shown items, current
      recommendations, and current product
      let newProduct = await Product.findOne({
        _id: { $nin: [...excludedIds, ...currentRecommendations, currentProductId] }, // Exclude
        removed items, current recommendations and current product
        category: categoryId,
        available: "available", // Only in-stock items
      });

      // If no products left in category, find popular products from other categories
      if (!newProduct) {
        const popularRecommendation = await Recommendation.findOne({
          productId: { $nin: [...excludedIds, ...currentRecommendations, currentProductId] },
        })
          .sort({ views: -1 }) // Sort by popularity
          .populate('productId');

        if (popularRecommendation) {
          newProduct = popularRecommendation.productId;
        }
      }

      if (!newProduct) {
        return res.status(404).json({ error: "No more products available" });
      }

      res.status(200).json({ newProduct });
    } catch (error) {
      console.error("Error replacing recommendation:", error);
      res.status(500).json({ error: "Internal Server Error" });
    }
  }
}

```

/components/recommendations.js

```

import { useEffect, useState } from "react";
import Carousel from "nuka-carousel";
import styles from "../Recommendations.module.css";
import WhiteBox from "../WhiteBox/WhiteBox";
import RecommendationsBox from
"@/components/RecommendationsSection/RecommendationsBox/RecommendationsBox";
import {
  renderCenterLeftControls,
  renderCenterRightControls,
} from "@/components/RecommendationsSection/controls/controls";

```

```

import { removeRecommendation } from "@lib/removeRecommendation";
import axios from "axios";

export default function Recommendations({ currentProductId, categoryId }) {
  const [isLargeScreen1440, setIsLargeScreen1440] = useState(false);
  const [isLargeScreen1024, setIsLargeScreen1024] = useState(false);
  const [isLargeScreen768, setIsLargeScreen768] = useState(false);
  const [products, setProducts] = useState([]);
  const [isLoading, setIsLoading] = useState(true);
  const [error, setError] = useState(null);
  const [excludedIds, setExcludedIds] = useState([]);

  useEffect(() => {
    const fetchRecommendations = async () => {
      if (!currentProductId) return;

      setIsLoading(true);
      try {
        const response = await axios.get("/api/recommendations", {
          params: { productId: currentProductId },
        });
        setProducts(response.data.recommendedProducts);
        setError(null);
      } catch (error) {
        console.error("Error fetching recommendations:", error);
        setError("Не вдалося завантажити рекомендації");
        setProducts([]);
      } finally {
        setIsLoading(false);
      }
    };

    fetchRecommendations();
  }, [currentProductId]);

  function handleResize() {
    const windowWidth = window.innerWidth;
    setIsLargeScreen1440(windowWidth >= 1440);
    setIsLargeScreen1024(windowWidth >= 1024);
    setIsLargeScreen768(windowWidth >= 768);
  }

  useEffect(() => {
    window.addEventListener("resize", handleResize);
    handleResize();
    return () => {
      window.removeEventListener("resize", handleResize);
    };
  }, []);

  const handleRemove = async (productId) => {
    try {

```

```

const { newProduct } = await removeRecommendation({
  productId,
  categoryId,
  excludedIds,
  currentRecommendations: products.map((p) => p._id),
  currentProductId,
});

setProducts((prev) => prev.filter((item) => item._id !== productId));
if (newProduct) {
  setProducts((prev) => [...prev, newProduct]);
}
setExcludedIds((prev) => [...prev, productId]);
} catch (error) {
  console.error("Error replacing recommendation:", error);
}
};

if (isLoading) {
  return (
    <WhiteBox>
      <h2 className={styles.productsTitle}>Завантаження рекомендацій...</h2>
    </WhiteBox>
  );
}

if (error) {
  return (
    <WhiteBox>
      <h2 className={styles.productsTitle}>{error}</h2>
    </WhiteBox>
  );
}

if (products.length === 0) {
  return null;
}

return (
  <WhiteBox>
    <h2 className={styles.productsTitle}>Рекомендації</h2>
    <Carousel
      className={styles.productsWrapper}
      slidesToShow={
        isLargeScreen1440 ? 4 : isLargeScreen1024 ? 3 : isLargeScreen768 ? 2 : 1
      }
      renderCenterLeftControls={renderCenterLeftControls}
      renderCenterRightControls={renderCenterRightControls}
    >
      {products.map((product) => (
        <div key={product._id} className={styles.slideContentWrapper}>
          <div className={styles.productBoxWrapper}>

```

```

        <RecommendationsBox {...product} onRemove={handleRemove} />
      </div>
    </div>
  )))
</Carousel>
</WhiteBox>
);
}

```

/replaceRecommendations.js

```

import { Product } from '@models/Product';
import { Recommendation } from '@models/Recommendation';
import { mongooseConnect } from '@lib/mongoose';

export default async function handler(req, res) {
  await mongooseConnect();

  if (req.method !== 'POST') {
    return res.status(405).json({ error: 'Method not allowed. Use POST.' });
  }

  const { productId, categoryId, excludedIds = [], currentRecommendations = [], currentProductId
} = req.body;

  try {
    // Search for a new product in the same category, excluding already shown items, current
    recommendations and current product
    let newProduct = await Product.findOne({
      _id: { $nin: [...excludedIds, ...currentRecommendations, currentProductId] }, // Exclude
      removed items, current recommendations and current product
      category: categoryId,
      available: "available", // Only in-stock products
    });

    // If no products left in this category, find popular products from other categories
    if (!newProduct) {
      const popularRecommendation = await Recommendation.findOne({
        productId: { $nin: [...excludedIds, ...currentRecommendations, currentProductId] },
      })
        .sort({ views: -1 }) // Sort by popularity (descending)
        .populate('productId');

      if (popularRecommendation) {
        newProduct = popularRecommendation.productId;
      }
    }

    if (!newProduct) {
      return res.status(404).json({ error: "No more products available" });
    }
  }
}

```

```

    res.status(200).json({ newProduct });
  } catch (error) {
    console.error("Error replacing recommendation:", error);
    res.status(500).json({ error: "Internal Server Error" });
  }
}

//RecomendationBox.js
import Link from "next/link";
import React, { useContext, useState } from "react";
import styles from "../RecommendationsBox.module.css";
import SolidProductCartItem from "@components/icons/SolidProductCartItem";
import { Toast } from "@lib/toast";
import { CartContext } from "@components/CartContext";
import { priceFormat } from "@lib/priceFormat";
import Image from "next/image";
import { FaTimes } from "react-icons/fa";
import { PulseLoader } from "react-spinners"; // Importing loader

export default function RecommendationsBox({ _id, title, price, images, available, onRemove }) {
  const { addProduct } = useContext(CartContext);
  const uri = '/product/' + _id;
  const [isRemoving, setIsRemoving] = useState(false); // State for loader

  function addToCart(productID) {
    addProduct(productID);
    Toast.fire({
      title: "Product added to cart!"
    });
  }

  const handleRemove = async () => {
    setIsRemoving(true); // Show loader and blur
    await onRemove(_id); // Call remove function from parent component
    setIsRemoving(false); // Hide loader and blur
  };

  return (
    <div className={` ${styles.wrapper} ${isRemoving ? styles.blurred : ""}` >
      </* Close button */>
      <button
        className={styles.removeButton}
        onClick={handleRemove}
        aria-label="Remove recommendation"
        disabled={isRemoving} // Disable button during removal
      >
        <FaTimes />
      </button>

      </* Loader */>
      {isRemoving && (
        <div className={styles.loaderOverlay}>

```

```

        <PulseLoader color="#ef233c" size={10} /> {/* Centered loader */}
      </div>
    )}

    <div className={styles.box}>
      <Link href={uri} key={images[0]} className={styles.imageWrapper}>
        <Image
          fill
          loading="eager"
          src={images && images.length > 0 ? images[0] : "/no-data.png"}
          className={styles.image}
          sizes="(max-width: 768px) 100vw, (max-width: 1200px) 50vw, 33vw"
          alt={`Product image - ${title}`}
        />
      </Link>
    </div>
    <Link href={uri} className={styles.title}>{title}</Link>
    <div className={styles.bottomWrapper}>
      {available === "available" ? (
        <div>
          <div className={styles.availableTitle}>In stock!</div>
          <div className={styles.price}>
            {priceFormat(price)} <span>€</span>
          </div>
        </div>
        <button onClick={() => { addToCart(_id) }} className={styles.btnCart} aria-
label="Add to cart">
          <SolidProductCartIcon />
        </button>
      </div>
      ) : (
        <div>
          <div className={styles.notAvailableTitle}>Out of stock!</div>
          <div className={styles.price}>
            {priceFormat(price)} <span>€</span>
          </div>
        </div>
        <button aria-label="Out of stock" disabled={true}
className={styles.disabledBtnCart}>
          <SolidProductCartIcon />
        </button>
      </div>
    )}
  </div>
</div>
);
}

```

```

/ProductPage

```

```

import React, { useContext, useEffect, useState } from "react";

```

```

import Head from "next/head";
import { CartContext } from "../../components/CartContext";
import { Product } from "../../models/Product";
import { priceFormat } from "../../lib/priceFormat";
import styles from "./SingleProduct.module.css";
import Center from "../../components/Center/Center";
import ProductImages from "../../components/ProductImages/ProductImages";
import WhiteBox from "../../components/WhiteBox/WhiteBox";
import Recommendations from "../../components/RecommendationsSection/Recommendations";
import { toLower } from "../../lib/toLower";
import { Toast } from "../../lib/toast";
import OneClickBuy from "@components/OneClickBuy/OneClickBuy";
import DescriptionSection from "@components/DescriptionSection/DescriptionSection";
import SolidProductCartIcon from "@components/icons/SolidProductCartIcon";
import { PulseLoader } from "react-spinners";
import { useRouter } from "next/router";

export default function ProductPage({ product }) {
  const { addProduct } = useContext(CartContext);
  const [isLoading, setIsLoading] = useState(true);
  const router = useRouter();

  useEffect(() => {
    const handleRouteChange = () => {
      setIsLoading(true);
    };

    router.events.on("routeChangeStart", handleRouteChange);
    return () => {
      router.events.off("routeChangeStart", handleRouteChange);
    };
  }, [router]);

  useEffect(() => {
    if (product) {
      setIsLoading(false);
    }
  }, [product]);

  function addToCart(productID) {
    addProduct(productID);
    Toast.fire({
      title: "Товар додано до кошику!",
    });
  }

  if (!product) {
    return (
      <Center>
        <WhiteBox>
          <h1>Товар не знайдено</h1>
        </WhiteBox>
      </Center>
    );
  }

```

```

    </Center>
  );
}

return (
  <
    <Head>
      <title>{product.metaTitle || `${product.title} | Avtotkanyna`} </title>
      <meta
        name="description"
        content={
          product.metaDescription ||
          `Купити ${toLower(product.title)}. Доступні ціни, гарантія якості. Доставка по
Україні`
        }
      />
    </Head>

    {isLoading && (
      <div className={styles.loaderOverlay}>
        <PulseLoader color="#ef233c" size={15} />
      </div>
    )}

    <div className={` ${styles.contentWrapper} ${isLoading ? styles.blurred : ""} `}>
      <Center>
        <div className={styles.colWrapper}>
          <div>
            <ProductImages key={product._id} images={product.images} />
          </div>
          <WhiteBox>
            <div className={styles.productInfoContainer}>
              <div>
                <h1 className={styles.title}>{product.title}</h1>
                <div className={styles.topWrapper}>
                  {product.available === "available" ? (
                    <div className={styles.availableTitle}>Є в наявності</div>
                  ) : (
                    <div className={styles.notAvailableTitle}>Немає в наявності!</div>
                  )}
                <div className={styles.code}>Код: {product.code}</div>
              </div>
              {product.properties && (
                <div className={styles.propertiesWrapper}>
                  <h2>Характеристики</h2>
                  {Object.keys(product.properties).map((key) => (
                    product.properties[key] && (
                      <div className={styles.propertiesContainer} key={key}>
                        <div className={styles.propertyName}>{key}</div>
                        <div>{product.properties[key]}</div>
                      </div>
                    )
                  )}
                </div>
              )}
            </div>
          </div>
        </div>
      </Center>
    </div>
  )
)

```

```

    )))
  </div>
  })
</div>
<div className={styles.bottomWrapper}>
  <div className={styles.price}>{priceFormat(product.price)} &lt;/div>
  {product.available === "available" ? (
    <div className={styles.buyButtonsWrapper}>
      <OneClickBuy productName={product.title} />
      <button
        onClick={() => addToCart(product._id)}
        className={styles.btnToCart}
      >
        <SolidProductCartIcon /> До кошика
      </button>
    </div>
  ) : (
    <button className={styles.notAvailable} disabled={true}>
      <SolidProductCartIcon /> До кошика
    </button>
  )}
</div>
</div>
</WhiteBox>
</div>
<DescriptionSection text={product.description} />
<Recommendations
  key={product._id}
  currentProductId={product._id}
  categoryId={product.category}
/>
</Center>
</div>
</>
);
}

export async function getServerSideProps(context) {
  const { id } = context.query;

  try {
    const product = await Product.findById(id).lean();
    if (!product) {
      return { notFound: true };
    }

    return {
      props: {
        product: JSON.parse(JSON.stringify(product)),
      },
    };
  } catch (error) {

```

```
    console.error("Error in getServerSideProps:", error);  
    return {  
      props: {  
        product: null,  
      },  
    };  
  }  
}
```