ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

КВАЛІФІКАЦІЙНА РОБОТА

Тема: «Вебсайт інтернет магазину «Bird Store» для продажу одягу з використанням багатокритеріальних фільтрів та рекомендаційної системи»

Ступінь вищої освіти – бакалавр
Спеціальність – 122 «Комп’ютерні науки»
Освітня програма «Комп’ютерні науки»

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав: здобувач 4 курсу
групи КН-21
Владислав ЩАДЕЙ
Керівник: к. військ. н. доцент кафедри
комп'ютерних наук
Володимир ТРОЦЬКО

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

м. Київ – 2025 рік

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Щадей Владислав Русланович

Тема роботи	Вебсайт інтернет магазину «Bird Store» для продажу одягу з використанням багатокритеріальних фільтрів та рекомендаційної системи
Номер та дата наказу про затвердження теми	№121-7 від 24 грудня 2024 року
Коротка постановка завдання	Розробка інтернет-магазину одягу, який буде оснащений інтуїтивним пошуком товарів та персоналізованими рекомендаціями. Буде забезпечена зручна адміністративна панель для управління магазином, адаптивний дизайн для комфортного користування з будь-якого пристрою
Посилання на джерела інформації (не більше п'яти найменувань, які рекомендує науковий керівник)	1. Сидоренко М.І., Гриценко О.М. Порівняльний аналіз алгоритмів колаборативної фільтрації для рекомендацій товарів в електронних магазинах. // Вісник Харківського національного університету імені В.Н. Каразіна. Серія: Математика, прикладна математика та інформатика. – 2022. – № 119. – С. 35-42. 2. Фелько М.Р. Дослідження методів побудови рекомендаційних систем для вирішення задач в електронній комерції, методи колаборативної фільтрації. // International Electronic Scientific Journal "Science Online". – 2021.
Вимоги до кваліфікаційної роботи	Кваліфікаційна робота має містити теоретичне, системотехнічне або експериментальне дослідження за темою роботи, яку слід розглядати як складне спеціалізоване завдання або практичну проблему в галузі комп'ютерних наук, яка характеризується комплексністю та невизначеністю умов і потребує застосування теорій і методів інформаційних технологій.

Дата видачі завдання 27 грудня 2024 р.

Керівник Володимир ТРОЦЬКО

Здобувач освітнього ступеня бакалавра Владислав ЩАДЕЙ

КАЛЕНДАРНИЙ ПЛАН

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

Керівник Володимир ТРОЦЬКО

Здобувач освітнього ступеня бакалавра Владислав ЩАДЕЙ

Щадей В.Р. Вебсайт інтернет магазину « Bird Store» для продажу одягу з використанням багатокритеріальних фільтрів та рекомендаційної системи

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

Робота присвячена розробці інтернет-магазину одягу, який буде оснащений інтуїтивним пошуком товарів та персоналізованими рекомендаціями. Буде забезпечена зручна адміністративна панель управління магазином, адаптивний дизайн для комфортного користування з будь-якого пристрою.

Ключові слова: Web-розробка, інтернет магазин, бази даних, аналіз.

Рис. 13. Бібліограф.: 19 найм.

Shchadey V.R. Website of the online store "Bird Store" for selling clothes using multi-criteria filters and a recommendation system

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

The work is dedicated to the development of an online clothing store, which will be equipped with an intuitive search for goods and personalized recommendations. A convenient administrative control panel for the store will be provided, adaptive design for comfortable use from any device.

Keywords: Web-development, online store, databases, analysis.

ЗМІСТ

ВСТУП

Актуальність теми. Розробка сучасних вебсайтів для роздрібної торгівлі вимагає інтеграції новітніх технологій, що дозволяють забезпечити високий рівень персоналізації та ефективності у взаємодії з користувачами. Одним із важливих напрямків розвитку таких сайтів є впровадження багатокритеріальних фільтрів та рекомендаційних систем, які значно покращують якість пошуку товарів та підвищують ймовірність здійснення покупки. Використання багатокритеріальних фільтрів дозволяє користувачам знаходити потрібні товари на основі кількох параметрів одночасно, таких як ціна, бренд, технічні характеристики чи відгуки покупців, а рекомендаційні системи допомагають запропонувати товари, що відповідають індивідуальним вподобанням клієнта. Таким чином, розробка сайту з інтеграцією таких інструментів сприяє не лише підвищенню зручності користувачів, але й оптимізації бізнес-процесів для продавців.

Актуальність даного дослідження полягає у зростаючій конкуренції на ринку електронної комерції, де здатність сайту забезпечувати швидкий та релевантний пошук товарів стає суттєвим фактором успіху. Підвищення рівня персоналізації на сайтах роздрібної торгівлі дозволяє збільшити конверсію відвідувачів у покупців та оптимізувати маркетингові стратегії. Особливо важливим це є в умовах постійного зростання кількості доступних товарів та розширення асортименту, де без ефективних фільтраційних та рекомендаційних механізмів користувачі можуть втрачати інтерес до сайту через складнощі у навігації та виборі продукції.

Мета дослідження. Метою роботи є розробка вебсайту роздрібної торгівлі з використанням багатокритеріальних фільтрів та рекомендаційної системи, що дозволить підвищити ефективність пошуку товарів та покращити користувацький досвід. Для досягнення цієї мети передбачається вирішення таких завдань, як аналіз існуючих програмних рішень у сфері роздрібної торгівлі, розробка алгоритмів для рекомендаційної системи, а також інтеграція розроблених рішень у структуру вебсайту.

Об'єктом дослідження є програмні додатки для роздрібної торгівлі, що використовують багатокритеріальні фільтри та рекомендаційні системи.

Предметом дослідження є вебсайт роздрібної торгівлі, який буде розроблено у межах даної роботи.

Методи дослідження включають аналіз існуючих вебсайтів, які використовують подібні технології, порівняння їх функціональних можливостей та підходів до реалізації багатокритеріальних фільтрів і рекомендаційних систем. Метод аналізу дозволить виявити найкращі практики та потенційні недоліки існуючих рішень, на основі чого буде розроблено оптимальну концепцію для нового сайту.

Новизна роботи полягає у створенні вебсайту, що поєднує у собі багатокритеріальні фільтри та рекомендаційні системи, оптимізовані під конкретні потреби роздрібної торгівлі. Використання таких технологій забезпечить високий рівень адаптивності сайту до вподобань користувачів та сприятиме підвищенню ефективності його роботи в умовах значного обсягу даних про товари та їх характеристики.

Структура та обсяг пояснювальної записки. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків та списку посилань (найменувань). Пояснювальна записка містить 16 рисунків, 1 таблицю, 1 додаток. Загальний обсяг пояснювальної записки складає 68 сторінок, основний зміст викладено на 53 сторінках.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Принцип роботи багатокритеріальних фільтрів та рекомендаційної системи

Багатокритеріальні рекомендаційні системи використовують відгуки або вподобання користувачів за кількома критеріями одночасно, щоб надавати більш персоналізовані результати. На відміну від традиційних підходів, де враховується лише один критерій (наприклад, загальна оцінка товару чи рейтингу фільму), багатокритеріальні фільтри аналізують декілька аспектів як користувацьких оцінок, так і характеристик об’єктів рекомендації[1]. Наприклад, система рекомендацій для ресторанів може враховувати якість їжі, рівень обслуговування, атмосферу закладу та співвідношення ціни/якості – і на основі цих кількох показників підбирати ресторан, що найкраще відповідає вподобанням відвідувача. Врахування множини факторів дозволяє точніше моделювати смаки користувача і підвищити точність рекомендацій. Водночас така деталізація створює нові виклики: потрібно зібрати більше даних про вподобання, врахувати залежності між критеріями та забезпечити, щоб алгоритми лишалися ефективними й масштабованими за більшої складності моделі[2]. Нижче проаналізовано принципи роботи таких багатокритеріальних фільтрів, наведено приклади алгоритмів і методів, а також порівняно їхню складність реалізації за різними критеріями.

Багатокритеріальний підхід у рекомендаційних системах передбачає врахування кількох вимірів оцінки при фільтрації та ранжуванні об’єктів. Кожен об’єкт (товар, фільм, ресторан тощо) може мати кілька оцінок або атрибутів – наприклад, фільм можуть оцінювати за акторською грою, сюжетом, візуальними ефектами і музичним супроводом. Багатокритеріальний фільтр намагається врахувати всі ці аспекти одночасно, щоб зрозуміти, наскільки об’єкт відповідає різним вимогам користувача[3]. Такий підхід покращує якість рекомендацій, оскільки враховує більш

нюансовані вподобання і різноманітні потреби користувачів. Дослідження підтверджують, що використання декількох критеріїв може підвищити точність та надійність рекомендацій порівняно з одно-критеріальними методами.

Принцип роботи багатокритеріальних фільтрів часто зводиться до двох ключових кроків: моделювання переваг користувача за кожним критерієм та агрегація цих оцінок для формування кінцевої рекомендації. Моделювання переваг може виконуватися різними способами – від простого збереження явних оцінок користувача по кожному параметру до побудови складних моделей, що передбачають приховані (латентні) вподобання. Аграгування може бути явним (наприклад, обчислення зваженої суми балів) або імпліцитним, коли модель сама вчиться поєднувати критерії для передбачення загальної оцінки чи рейтингу об’єкта[4].

Розглядаючи кілька критеріїв, система стикається з рядом проблем. По-перше, залежності між критеріями: часто оцінки за різними параметрами не є незалежними (наприклад, у готелі оцінки чистоти номера та якості обслуговування можуть корелювати). Такі міжкритеріальні залежності призводять до проблем мультиколінеарності, що ускладнює побудову моделей і може погіршувати точність, якщо не врахувати ці залежності[5]. По-друге, неповнота даних: не всі користувачі заповнюють оцінки за кожним критерієм. На практиці багато хто ставить оцінку лише за основним показником або кількома з них, ігноруючи решту. В результаті матриця "користувач-критерій-об’єкт" містить багато пропусків, що породжує проблему розрідженості та ускладнює побудову достовірних моделей[6]. По-третє, зростає обчислювальна складність: моделювання вподобань у багатовимірному просторі вимагає більше ресурсів і витонченіших алгоритмів, ніж у випадку одновимірних рейтингових даних[7]. Незважаючи на ці труднощі, правильно спроєктований багатокритеріальний фільтр може значно покращити відповідність рекомендацій інтересам користувача, тому було запропоновано ряд методів для реалізації такого підходу – від відносно простих до дуже

складних. Розглянемо деякі з цих методів та алгоритмів і оцінимо їхню складність[8].

Існує багато способів врахувати декілька критеріїв при побудові рекомендацій. Нижче наведено кілька підходів – від простіших у реалізації до більш складних – з прикладами алгоритмів:

Найбільш прямолінійний спосіб врахувати кілька факторів – обчислити для кожного об’єкта зведену оцінку, що комбінує всі критерії. Наприклад, можна задати ваги для кожного критерію та обчислити зважену суму або середнє. Ваги можуть визначатися експертно (вручну) або автоматично. Простіший автоматизований підхід – застосувати множинну лінійну регресію, щоб навчити модель прогнозувати загальний рейтинг на основі кількох критеріїв. Так, Jhalani та ін. запропонували визначати вагу кожного критерію шляхом навчання регресійної моделі на багатокритеріальних даних і таким чином розраховувати підсумкову оцінку для рекомендації[9]. Реалізація такого методу досить проста: існують готові бібліотеки для лінійної регресії, а обчислення зважених сум потребує мінімум ресурсів. Однак цей підхід фактично зводить багатовимірні вподобання до одновимірного числового значення, що може втрачати частину інформації про смаки.

Методи колаборативної фільтрації (CF) можна адаптувати під кілька критеріїв. У випадку пам’яттю-орієнтованих підходів (memory-based CF) це означає, що подібність між користувачами або між об’єктами визначається з урахуванням кількох розмірів оцінок[10]. Приміром, item-based (об’єкт-орієнтована) колаборативна фільтрація може обчислювати схожість між двома товарами, порівнюючи вектори їх оцінок по всіх критеріях. Така схожість може розраховуватися через метрику відстані (евклідова, косинусна тощо) або агрегацію часткових схожостей по кожному критерію. Bilge та Kaleli показали, що розширення стандартного алгоритму item-based CF для використання багатокритеріальних оцінок підвищує надійність рекомендацій порівняно з одно-критеріальним варіантом[11]. Аналогічно, для user-based (користувацької) фільтрації можна враховувати, наскільки двоє користувачів

схожі за вподобаннями одночасно в кількох вимірах. Впровадження багатокритеріальної CF зазвичай потребує змін лише в обчисленні схожості та оцінки рейтингу: наприклад, для прогнозу рейтингу можна окремо врахувати внесок сусідів по кожному критерію і об’єднати їх[12]. Такий підхід дещо складніший за однокритеріальний CF (потрібно обробляти більше даних і комбінувати результати), але все ще відносно простий у реалізації. Обчислювально він масштабується приблизно лінійно з кількістю критеріїв (наприклад, якщо критеріїв 5, то розрахунок матриці схожості буде в п’ять разів повільнішим), що є прийнятним для невеликого числа аспектів[13].

Більш просунутий клас методів – модельна (model-based) фільтрація, де застосовуються алгоритми машинного навчання для одночасного врахування кількох критеріїв[14]. Один з популярних підходів – матрична факторизація: традиційно вона розкладає матрицю рейтингів (користувач × об’єкт) на латентні фактори. У випадку багатокритеріальних рейтингів замість однієї матриці може використовуватися тензор (трьохвимірний масив) або кілька матриць – по одній на кожен критерій. Модель намагається навчитися представляти як користувача, так і об’єкт у просторі прихованих факторів так, щоб відновлювати оцінки по всіх критеріях[15]. Наприклад, можна навчити модель, яка передбачає відразу всі складові оцінки (сюжет, акторська гра, музика тощо для фільмів) для пари "користувач-фільм". Отримані прогнозовані значення критеріїв потім можуть бути об’єднані (наприклад, зважено) для формування загальної оцінки або рейтингу рекомендованих об’єктів[16]. Перевага латентно-факторних підходів у тому, що вони можуть виявляти приховані зв’язки – наприклад, якщо користувач високо оцінює атмосферу ресторанів, модель може виявити цей фактор навіть без явного вагового коефіцієнта, а потім віддавати перевагу закладам з подібними характеристиками. Однак навчання факторизаційної моделі потребує розв’язання оптимізаційної задачі – зазвичай методом градієнтного спуску або ALS (alternating least squares) – що є значно складнішим процесом, ніж проста фільтрація сусідів[17]. Алгоритмічна реалізація матричної факторизації для

багатьох критеріїв є більш складною, аніж для одного, оскільки збільшується кількість параметрів моделі і потенційно розмірність оптимізаційної задачі. Крім того, факторизації потрібні достатні дані: якщо матриця дуже розріджена (багато відсутніх оцінок по критеріях), модель може переобучитися або давати неточні прогнози. На практиці застосування матричної факторизації виправдане для великих масивів даних з багатокритеріальними оцінками, де є обчислювальні ресурси для її навчання[18].

Сучасні рекомендаційні системи дедалі частіше звертаються до глибинного навчання для врахування складних залежностей. Багатокритеріальні фільтри не є винятком – дослідники пропонують нейронні мережі, які приймають на вхід або прогнозують одразу декілька критеріїв. Один приклад – використання варіаційного автоенкодера (VAE) для моделювання уподобань[19]. Fraihat та ін. запропонували нейромережеву модель на основі VAE, що послідовно навчається на багатьох критеріях і виявляє складні залежності між вподобаннями користувача та властивостями об’єктів[20]. В їх архітектурі врахування множинних аспектів товарів (multi-criteria ratings) дозволило моделі захопити приховані взаємозв’язки між різними критеріями та загальними вподобаннями користувача, що покращило точність рекомендацій. Загалом нейронні мережі можуть навчитися самостійно агрегувати критерії через внутрішні шари, не потребуючи явного завдання ваг. Вони добре справляються з нелінійними залежностями та високорозмірними даними, витягуючи узагальнені ознаки з множини. Проте ціною є дуже висока складність реалізації: потрібно налаштувати архітектуру мережі, мати достатньо даних для навчання (щоб запобігти перенавчанню моделі на одному або двох критеріях), і залучати значні обчислювальні ресурси (GPU/TPU) для тренування[21]. Глибинні моделі є “важковаговими” в плані розробки й обчислень, але забезпечують найвищу гнучкість і потенційно найкращу якість рекомендацій, особливо якщо критерії мають складні взаємозв’язки.

Гібридні та багатокрокові алгоритми. Деякі сучасні дослідження пропонують складні композиційні рішення, які поєднують кілька методів задля покращення результатів багатокритеріальних рекомендацій. Наприклад, Nilashi та співавт. використали комбінацію методів: кластеризацію Expectation-Maximization (EM) для сегментації, алгоритм рішучих дерев CART для прогнозування рейтингів в межах кластерів, а також застосували аналіз головних компонент (PCA) для зменшення вимірності критеріїв і подолання проблеми мультиколінеарності між ними[22]. Такий комплексний підхід показав високу точність рекомендацій у їх експериментах, оскільки поєднує класифікацію схожих користувачів/об'єктів, нелінійне моделювання залежностей (через дерева рішень) та спрощення простору даних (через PCA). Інший приклад – робота Farokhi та ін., де запропоновано об'єднати багатокритеріальний user-based і item-based CF: спочатку визначаються схожі користувачі та схожі об'єкти з урахуванням усіх критеріїв, а потім ці підходи інтегруються для генерації рекомендацій. При цьому використовуються алгоритми кластеризації (нечіткий c-середніх та k-середніх) для групування користувачів і об'єктів, що покращує точність, оскільки враховуються шаблони вподобань на рівні сегментів. Ще складніший варіант запропонували Nilashi та ін. для туристичних рекомендацій: вони поєднали EM-кластеризацію, нейро-нечітку систему ANFIS і PCA, щоб використати приховані знання з багатокритеріальних оцінок та одночасно зменшити розмірність даних. Отриманий гібрид значно поліпшив точність прогнозів, хоча його реалізація є багатоступеневою і ресурсомісткою[23]. У таких гібридних системах кожен додатковий компонент (кластеризація, окрема модель, тощо) збільшує складність реалізації та потребує налаштування. Перевага – максимальне використання інформації та компенсація недоліків одного методу іншим, проте ціною є зростання вимог до даних і ресурсів, а також ускладнення підтримки такої системи.

Різні багатокритеріальні фільтри суттєво відрізняються за трудомісткістю впровадження та вимогами. Одні методи можна реалізувати

кількома рядками коду, тоді як інші потребують складного навчання моделей і оптимізації.

Простішим алгоритмам потрібно мінімум ресурсів – наприклад, обчислення зваженого середнього чи пошук найближчих сусідів за декількома критеріями виконується швидко на сучасному CPU. Вага таких методів у тому, що їх можна запускати навіть у реальному часі для великої кількості користувачів. Натомість складні моделі (факторизація, глибинні нейронні мережі) вимагають значно більше обчислень[24]. Їхнє навчання може займати години чи дні на графічних процесорах, оскільки потрібно багаторазово перебрати великі масиви даних для корекції параметрів моделі. Гібридні алгоритми з кількома етапами (кластеризація + прогнозування + об'єднання) також є ресурсомісткими, адже кожен крок додає накладні витрати. Існують підходи для зменшення цих витрат – зокрема, попереднє зниження розмірності даних. Так, застосування PCA або інших методів скорочення особливостей може знизити обчислювальну вартість рекомендації без втрати точності. Дослідники відзначають, що правильно спроєктований гібридний метод із врахуванням багатьох критеріїв здатен одночасно підвищити точність і зменшити обсяг обчислень, необхідних для. Проте загалом, чим складніший алгоритм, тим більше пам'яті та часу він вимагатиме при навчанні і виконанні, тому питання ресурсів є ключовим при виборі підходу.

Простота реалізації багато в чому залежить від того, чи є у нас потрібні дані. Щоб застосувати багатокритеріальний фільтр, система повинна збирати оцінки або інформацію за кількома критеріями від кожного користувача. На практиці отримати такі детальні дані важко – користувачі не завжди готові заповнювати довгі форми відгуків. Якщо дані частково відсутні, прості методи можуть обійти цю проблему: наприклад, використовувати лише наявні критерії чи заміняти відсутні значення середніми. Складні моделі, навпаки, чутливі до розрідженості – вони намагаються навчитися на всіх критеріях одночасно, і велика кількість пропусків може сильно погіршити результати. Деякі алгоритми (особливо глибинного навчання) можуть включати

спеціальні механізми для роботи з неповними даними або використовувати додаткові джерела (наприклад, тексти відгуків, щоб опосередковано оцінити критерії), але це ще більше ускладнює систему. Тому з точки зору доступності даних простіші багатокритеріальні фільтри більш життєздатні, коли повної інформації немає – вони гнучкіше реагують на нестачу даних. У випадку ж наявності великого обсягу детальних оцінок, складні методи отримують необхідне «паливо» для навчання і можуть реалізувати свій потенціал.

Під алгоритмічною складністю маємо на увазі як теоретичну складність алгоритму, так і практичну складність його реалізації. Прості підходи базуються на елементарних операціях – додавання, середнє, вимірювання схожості – і зрозумілих алгоритмах (пошук сусідів, лінійна регресія). Їх легко реалізувати та протестувати, і розробнику не потрібно глибоко занурюватися в математичні деталі. Натомість розвинені багатокритеріальні алгоритми включають складніші компоненти: багатовимірну оптимізацію, нейронні мережі з багатьма шарами, комбінування кількох моделей тощо. Такі методи можуть мати багато гіперпараметрів (наприклад, кількість факторів у факторизації або структура нейронної мережі), від налаштування яких залежить успіх роботи системи. Моделі глибокого навчання і гібридні системи важко налаштовувати і налагоджувати – потрібна експертиза та експерименти, щоб досягти хорошого результату. Крім того, взаємозалежність критеріїв додає складності: якщо критерії корельовані, алгоритму треба спеціальні механізми, щоб це врахувати (як-от додавання PCA для усунення. Загалом, зростання алгоритмічної складності означає більший обсяг коду, більше потенційних точок відмови і вищі вимоги до кваліфікації розробників. В літературі відзначається, що саме через складність моделювання уподобань за кількома аспектами та взаємозв’язків між ними багатокритеріальні системи є нетривіальними у розробці. Відповідно, з точки зору алгоритмічної складності, методи на основі простих евристик або невеликого розширення існуючих алгоритмів (CF, регресія) значно легші, ніж впровадження повноцінної нейронної чи факторизаційної моделі.

Цей фактор визначає, як добре рішення працюватиме при зростанні числа користувачів, об’єктів та критеріїв. Прості методи часто більш масштабовані, або ж їх легше масштабувати шляхом простого паралелізму чи кешування. Наприклад, обчислення зважених оцінок для мільйонів об’єктів можна розпаралелити по об’єктах без особливих проблем. Колаборативну фільтрацію можна прискорити попереднім прорахунком схожостей офлайн. До того ж, якщо критеріїв небагато (скажімо, 3–5), додаткове навантаження несуттєве. Натомість моделі, які потребують навчання, можуть стикатися з обмеженнями при масштабуванні: тренувати факторизацію на десятки мільйонів користувачів або предметів складно, потрібно розподіляти обчислення на кластери. Глибинні нейронні мережі теоретично добре масштабується на великі дані (їх навіть ефективніше навчати на великих вибірках), проте практично обмеження апаратного забезпечення та пам’яті все одно існують. Крім того, додавання кожного нового критерію може вимагати суттєвого перепроєктування моделі (наприклад, збільшення розмірності вихідного шару нейронної мережі чи зміни структури факторизації). Деякі дослідники спеціально концентруються на питаннях масштабованості багатокритеріальних рекомендацій. Зокрема, показано, що введення етапу зниження розмірності або кластеризації може допомогти системі масштабуватися, не втративши точності, а навпаки – інколи й підвищивши її. У роботі Bokde та ін. поєднання багатокритеріальної item-based фільтрації з методами зменшення розмірності дозволило одночасно зменшити обчислювальні витрати і розв’язати проблеми масштабування та розрідженості даних. В цілому, чим складніший алгоритм, тим складніше його масштабувати на дуже великі датасети – може знадобитися розподілена інфраструктура або спеціальні оптимізації. Простішими підходами легше керувати при масштабуванні, хоча вони можуть програвати в точності на величезних обсягах даних, де складні моделі розкривають свій потенціал.

Багатокритеріальні фільтри в рекомендаційних системах дозволяють врахувати різнобічні вподобання користувачів і тим самим підвищити

релевантність рекомендацій. Проте методи реалізації такого підходу можуть суттєво відрізнятися за складністю. Прості методи (зважене об’єднання оцінок, невеликі модифікації колаборативної фільтрації) є більш доступними для впровадження та потребують менше даних і ресурсів, але можуть не повною мірою використовувати всю багатовимірну інформацію. Складні методи (матричні факторизації, нейронні мережі, багатокомпонентні алгоритми) здатні моделювати тонкі взаємозв’язки між критеріями і покращувати точність рекомендацій, проте вимагають серйозних обчислень, великих наборів даних і ретельного налаштування. На практиці вибір підходу має враховувати доступні ресурси та дані: якщо система має просту структуру даних і обмежені можливості, доцільно почати з простіших багатокритеріальних фільтрів. Зі зростанням обсягів даних та потреб у точності можна поступово переходити до складніших моделей, перевіряючи, чи виправдані витрати ресурсів приростом якості рекомендацій. Успішна реалізація багатокритеріальної рекомендаційної системи часто полягає в балансуванні між складністю моделі та її практичною ефективністю, забезпечуючи користувачам більш персоналізовані й корисні результати.

1.2 Аналіз аналогічних розробок

Amazon демонструє приклад масштабованої системи, в якій багатокритеріальні фільтри інтегровані з рекомендаційними механізмами на основі аналізу поведінкових та контекстних даних[25] (рис.1.1).

На платформі залучаються різноманітні параметри, що стосуються характеристик товару, історії переглядів, попередніх покупок, відгуків інших користувачів та низки вторинних показників, отриманих із внутрішніх джерел інформації. Такий підхід дозволяє формувати гнучкі фільтри, які враховують не лише базові критерії на кшталт ціни та бренду, а й специфічні атрибути чи варіанти використання товару, що створює умови для більш адаптивного пошуку й посилює релевантність результатів. Водночас це забезпечує високу масштабованість системи, здатної обробляти великі обсяги даних у реальному

часі, враховуючи розгалуженість асортименту та безперервний потік нових позицій, які одразу підпадають під дію фільтраційних алгоритмів. Основним алгоритмічним ядром рекомендацій виступає колаборативна фільтрація, що доповнюється змішаними підходами персоналізації, включно з побудовою профілів користувачів на основі аналізу актуальних і минулих запитів.

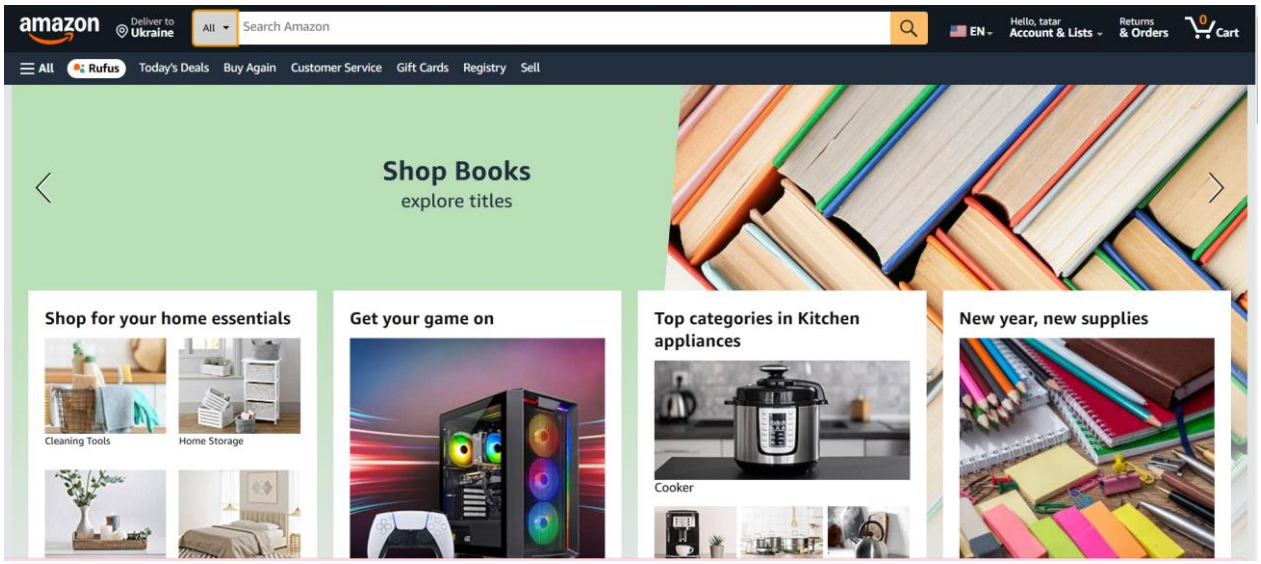

Рисунок 1.1 – Головне вікно сайту Amazon

Джерело:[25]

Такий багаторівневий підхід сприяє швидкому виявленню товарів, що відповідають потребам конкретного відвідувача, забезпечує економію часу й формує довіру до системи через високу влучність пропонованих рекомендацій. Перевага полягає у зручності для аудиторії та підвищенні імовірності здійснення повторних покупок завдяки точним підказкам щодо сумісних або взаємодоповнюючих товарів. Водночас при роботі з базою даних потребується значна обчислювальна потужність і складна інфраструктура, що може ускладнити процес масштабування та потребує професійних рішень для забезпечення належного рівня продуктивності. Складність алгоритмів іноді призводить до виникнення ситуацій, коли рекомендації виявляються надмірно вузькоспеціалізованими або орієнтованими лише на один аспект поведінки

користувача, що зменшує загальний діапазон охоплення асортименту. Крім того, використання персональних даних і детального відстеження активності викликає питання щодо безпеки й конфіденційності, що вимагає впровадження ретельно налагоджених процедур захисту інформації та прозорого інформування відвідувачів про принципи роботи рекомендаційної системи.

Rozetka характеризується комплексною системою багатокритеріальних фільтрів, які дають змогу диференціювати товари за різними параметрами, як-от ціна, бренд, технічні характеристики чи рейтинги покупців[26] (рис.1.2).

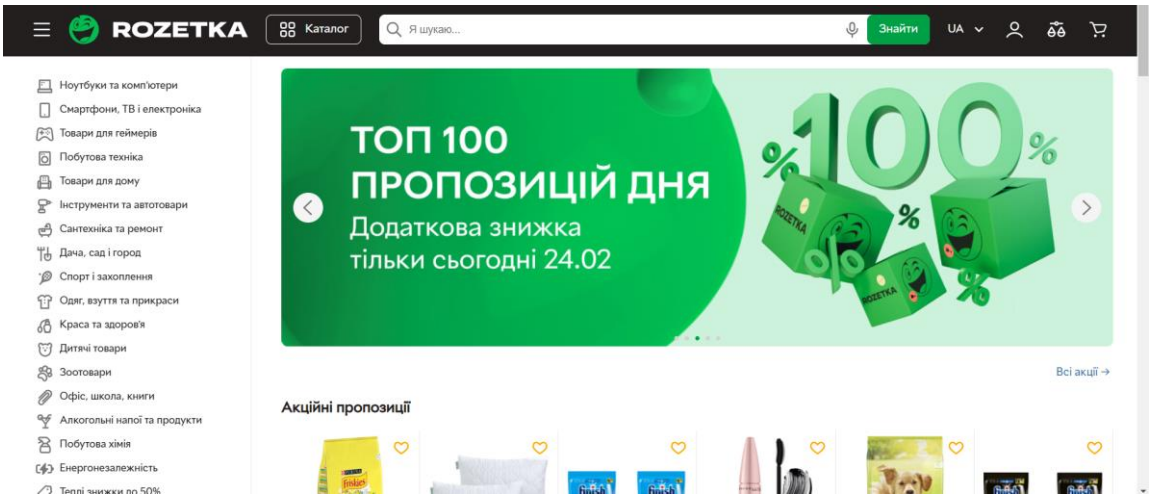

Рисунок 1.2 – Головне вікно сайту Rozetka

Джерело: [26]

Такий підхід спрощує пошук потрібного продукту, зменшуючи кількість непотрібних елементів на екрані. Завдяки цьому користувачі можуть оперативно орієнтуватися у великому асортименті, що сприяє підвищенню точності вибору та економії часу. Система фільтрації реалізована через динамічні алгоритми, які враховують зміни в каталозі та коригують відображення на основі актуальних показників доступності чи популярності. На рівні інтерфейсу впроваджено зручні панелі фільтрації з гнучкими полями для введення числових чи текстових запитів, а також інтерактивні повзунки, які візуально допомагають користувачам встановлювати потрібні діапазони

параметрів. Така реалізація полегшує процес порівняння аналогічних товарів і підвищує імовірність задоволення потреб споживачів, адже пошук стає точнішим і менш обтяжливим.

При цьому існують певні недоліки, пов’язані з ризиком перевантаження користувача великою кількістю можливих критеріїв, що може викликати складнощі у навігації та спричиняти розгубленість у підборі релевантних параметрів. З часом при розширенні асортименту й появі нових товарних категорій постає потреба в регулярному оновленні переліку фільтраційних опцій та узгодженні їх із реальним вмістом каталогу. Неправильно налаштовані або надто обмежувальні фільтри можуть ускладнювати виявлення певних груп товарів, що часом призводить до неповного відображення потенційно цікавих пропозицій і зменшує загальний рівень задоволеності клієнтів. Також у разі високої завантаженості сервера або при обробці великої кількості запитів можуть відбуватися затримки в оновленні результатів, що негативно впливає на динаміку взаємодії.

Рекомендаційна система на Rozetka базується на аналізі попередніх пошукових дій користувачів, історії переглядів та покупок, а також на загальних патернах поведінки аудиторії. Тут використовується зважування різних факторів, зокрема враховується поведінкова інформація, поточні тенденції ринку та індивідуальні налаштування користувача. Під час кожного сеансу фіксуються дані про відвідувані сторінки, переглянуті категорії й конкретні товари, а також тривалість перебування на певних позиціях. Динамічний характер рекомендаційної системи дозволяє пропонувати варіанти, які щойно набули популярності або мають підвищений попит у схожих груп користувачів. Такий підхід поліпшує персоналізацію розділу пропозицій і стимулює збільшення частоти придбань, оскільки більше релевантних товарів виводяться на перші позиції. Водночас існують ризики формування певних “інформаційних бульбашок” через надмірне використання історії пошуку, що зменшує варіативність асортименту, запропонованого конкретному користувачу, й може ускладнити пошук

нестандартних або нових для нього позицій. Однак, завдяки поєднанню багатокритеріальної фільтрації з рекомендаційними модулями, забезпечується відносно висока ефективність у представленні максимально релевантних товарів, а також надається простір для гнучкої зміни фільтраційних параметрів залежно від поточних запитів.

Aliexpress пропонує широкий вибір товарів від великої кількості незалежних продавців, що створює гнучкі умови для формування конкурентоспроможних цінових пропозицій і надає споживачам доступ до різноманітного асортименту продукції[27] (рис.1.3).

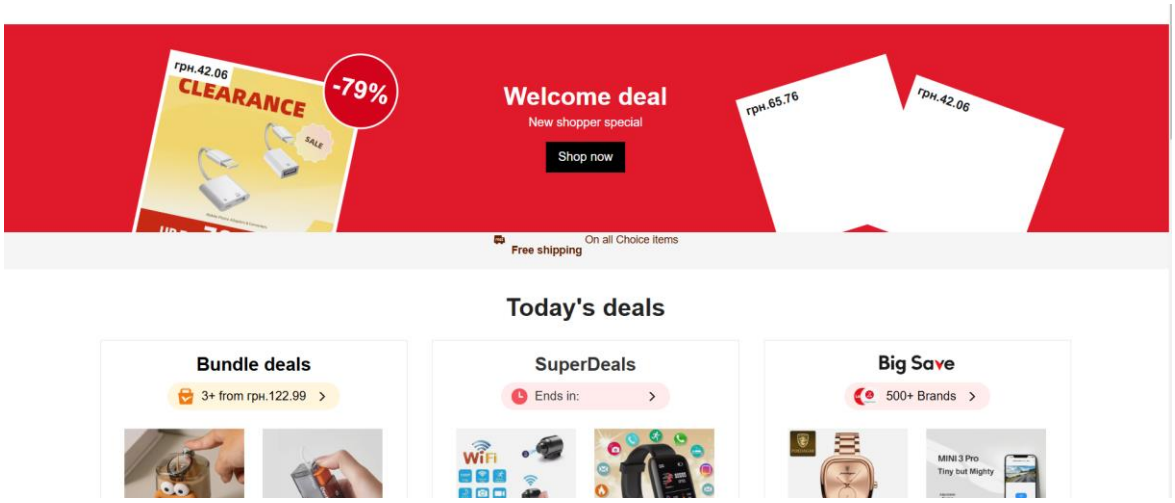

Рисунок 1.3 – Головне вікно сайту Aliexpress

Джерело: [27]

Система багатокритеріальних фільтрів охоплює параметри цінового діапазону, географічну локацію продавця, рейтинг магазину та конкретні характеристики виробів, зокрема наявність певної комплектації чи можливості безкоштовної доставки. Завдяки такому підходу забезпечується деталізований пошук, який дає змогу користувачеві налаштувати низку критеріїв і звузити перелік пропозицій відповідно до персональних пріоритетів. Значна база даних, якою володіє платформа, дозволяє відстежувати тренди продажів, здійснювати перехресний аналіз уподобань покупців і формувати автоматизовані поради щодо суміжних товарів або послуг. Алгоритми

рекомендаційної системи включають елементи колаборативної фільтрації та контент-орієнтованого підходу: перший передбачає встановлення зв’язків між користувачами зі схожими вподобаннями, тоді як другий враховує змістові характеристики товарів, аналізуючи їх описи, категорії та тематичні теги.

Такий масштабний сервіс має і певні недоліки, що зумовлені складністю управління величезним потоком інформації та необхідністю узгодження стандартів між різними продавцями. Перевантаженість інтерфейсу може спричиняти незручності при використанні багатокритеріальних фільтрів, особливо якщо покупець не до кінця обізнаний з відповідною термінологією або не володіє детальною інформацією про специфіку потрібного товару. Іноді зустрічаються неточні або фіктивні відгуки, що ускладнює об’єктивне оцінювання рейтингу продавця і може спотворювати кінцеві рекомендації. Незважаючи на це, Aliexpress постійно вдосконалює механізми контролю за якістю пропонованих позицій і проводить регулярний моніторинг поведінки споживачів, щоб оптимізувати налаштування фільтраційних процесів, посилити захист від шахрайства та створити привабливіші умови для купівлі. Така модель взаємодії підтверджує достатню гнучкість системи і водночас вимагає подальшого розвитку функцій пошуку та рекомендацій, аби зробити роботу з платформою прозорішою й ефективнішою для широкої аудиторії.

В таблиці 1.1 представлено порівняльний аналіз фільтрації на досліджуваних сайтах.

Таблиця 1.1 – Порівняльний аналіз багатокритеріальної фільтрації на сайтах

Критерії фільтрації	**Amazon**	**Rozetka**	**Aliexpress**
Ціна	Реалізовано, підтримка діапазону цін	Реалізовано, зручний повзунок	Реалізовано, іноді неточна відповідність

Бренд	Реалізовано, пошук за брендом	Реалізовано, статичний список	Реалізовано, не завжди коректно зазначено бренд
Категорія товару	Реалізовано, гнучка система підкатегорій	Реалізовано, розширена категоризація	Реалізовано, іноді некоректні категорії
Рейтинг товару	Реалізовано, фільтрація за кількістю зірок	Реалізовано, вибір за середнім рейтингом	Реалізовано, іноді рейтинг не відповідає якості
Кількість відгуків	Реалізовано, доступний фільтр	Реалізовано, вибір за кількістю відгуків	Реалізовано, не завжди якісні відгуки
Характеристики товару	Реалізовано, розширені технічні характеристики	Реалізовано, основні параметри	Реалізовано, але дані можуть бути некоректними
Доставка та наявність	Реалізовано, інформація про склади та доставку	Реалізовано, підтримка самовивозу	Частково реалізовано, не завжди точна інформація
Акційні пропозиції	Реалізовано, фільтрація за акціями	Реалізовано, чітке відображення знижок	Реалізовано, іноді знижки неправдиві
Персоналізовані рекомендації	Частково реалізовано, базується на історії покупок	Не реалізовано	Частково реалізовано, іноді не враховує інтереси
Фільтрація за новизною	Реалізовано, підтримка фільтрації за датою додавання	Реалізовано, вибір за новинками	Реалізовано, можливість сортувати за новизною

Загальний аналіз показує, що сайт Amazon має найбільш розширену та функціональну систему багатокритеріальної фільтрації, забезпечуючи гнучкість у виборі товарів за різними критеріями, тоді як Rozetka відзначається простотою та зручністю використання, але бракує персоналізованих

рекомендацій. Aliexpress також пропонує широкий вибір фільтрів, однак має проблеми з точністю даних щодо доставки та часткову реалізацію рекомендаційної системи. Отже, кожен із сайтів потребує певних покращень, зокрема щодо адаптації до індивідуальних вподобань користувачів та підвищення якості інформації про товари.

1.3 Постановка задачі

Проєкт спрямований на розроблення вебсайту інтернет-магазину, що має забезпечити високу зручність користування завдяки інтеграції розвинених багатокритеріальних фільтрів та персоналізованих рекомендацій. Зручність полягає у можливості швидкого та точного пошуку товарів за допомогою адаптивних фільтрів, які дозволять користувачам відбирати товари не лише за базовими критеріями, такими як ціна, наявність і тип продукту, але й за більш детальними параметрами: рейтингом, кількістю відгуків, характеристиками товару (наприклад, колір, розмір, матеріал), а також за специфічними атрибутами, як-от новизна товару чи його популярність серед покупців. Інтерфейс фільтрів буде реалізовано у вигляді динамічних випадаючих списків, слайдерів для діапазонів числових значень і чекбоксів для вибору декількох параметрів одночасно, що забезпечить інтуїтивність і простоту у використанні.

Персоналізація сайту реалізовуватиметься через рекомендаційну систему, яка враховуватиме індивідуальні вподобання та поведінкові патерни користувачів. Для цього використовуватимуться методи колаборативної фільтрації, які аналізуватимуть дії схожих за поведінкою користувачів, а також контент-орієнтований підхід, що оцінюватиме властивості товарів на основі попередніх виборів конкретного відвідувача. Наприклад, якщо користувач частіше переглядає або купує товари певного бренду чи в конкретній ціновій категорії, система автоматично пріоритизуватиме відповідні пропозиції. Крім того, сайт зможе адаптуватися до поточних запитів, наприклад, показуючи новинки чи акційні товари у вибраних

категоріях, а також надавати рекомендації на основі схожих товарів або аксесуарів до вже обраних продуктів.

Проєкт передбачає створення модулів для автоматичного аналізу поведінки користувачів, що дозволить сайту «вчитися» на основі зібраних даних, оновлюючи алгоритми фільтрації та рекомендацій у відповідь на нові тренди чи зміни у вподобаннях відвідувачів. Таким чином, кінцевий продукт має потенціал не лише забезпечити зручність для поточних користувачів, але й продемонструвати адаптивність до змінюваних умов ринку та особливостей асортименту, що, за сприятливих умов, може позитивно вплинути на його конкурентоспроможність. Однією з переваг проєкту є можливість реалізації якісної фільтрації за умови використання мінімальних або доступних обчислювальних ресурсів, що може сприяти ефективному функціонуванню системи

Висновки до розділу

Унікальність розробки вебсайту інтернет-магазину полягає у впровадженні багатокритеріальних фільтрів та персоналізованої рекомендаційної системи, які інтегруються у загальну структуру платформи таким чином, щоб забезпечити максимально адаптивний користувацький досвід. Комплексний підхід до їх реалізації дозволяє не лише підвищити ефективність пошуку та вибору товарів, а й створити динамічну взаємодію між елементами системи, що враховує сучасні очікування та поведінкові особливості користувачів.

На відміну від провідних платформ, таких як Amazon чи Rozetka, запропонований сайт передбачає не просто стандартні фільтри за ціною, брендом чи характеристиками товарів, а й розширену можливість комбінації критеріїв, врахування контексту вибору та адаптацію до поведінкових патернів відвідувачів у режимі реального часу. Сайт забезпечує підвищену гнучкість у фільтрації, дозволяючи користувачам налаштовувати власні критерії та зберігати їх для майбутнього використання. Персоналізація

базується на поєднанні колаборативної фільтрації з контент-орієнтованим аналізом, що дозволяє не лише враховувати історію покупок, але й передбачати потенційні інтереси на основі динамічної оцінки подібності товарів.

Унікальною особливістю проєкту є впровадження модулів для автоматизованого аналізу тенденцій ринку та поведінкових даних, що дозволяє не лише реагувати на вже існуючі уподобання користувачів, але й проактивно формувати рекомендації на підставі нових трендів. Це забезпечує актуальність пропозицій навіть за умов швидких змін асортименту чи вподобань аудиторії. Сайт буде оснащений механізмами моніторингу ефективності рекомендацій, що дозволить оперативно коригувати алгоритми на основі зворотного зв’язку та аналізу реальних даних.

Таким чином, основна конкурентна перевага проєкту полягає у створенні інтегрованої екосистеми, яка поєднує широкий функціональний потенціал із механізмами персоналізації, що забезпечують динамічне налаштування під індивідуальні вподобання користувачів. Це дозволяє не лише підвищити ефективність взаємодії з платформою, а й забезпечити унікальний користувацький досвід, який виходить за межі традиційних рішень у сфері електронної комерції.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ВЕБСАЙТУ

2.1 Вибір інструментальних засобів розробки

У контексті розробки вебсайту інтернет-магазину з багатокритеріальними фільтрами та рекомендаційною системою було обрано стек інструментальних засобів MERN (MongoDB, Express.js, React, Node.js)[28]. Цей стек забезпечує гнучкість, масштабованість та зручність у розробці як клієнтської, так і серверної частин застосунку.

React (React.js) використовується для створення інтерфейсу користувача. Його компонентна архітектура дозволяє розбивати інтерфейс на незалежні блоки, що спрощує підтримку та повторне використання коду[29]. Віртуальний DOM забезпечує швидке оновлення динамічного контенту – зокрема при зміні фільтрів чи персоналізованих рекомендацій. Також React легко інтегрується з бібліотеками маршрутизації (React Router), управління станом (Redux, Zustand) і UI-компонентами[30].

Node.js обрано як середовище виконання JavaScript на сервері, що дозволяє використовувати одну мову на всіх рівнях проєкту. Разом із фреймворком Express.js, Node.js забезпечує створення REST API, обробку запитів, реалізацію бізнес-логіки, включаючи фільтрацію товарів і обробку користувацьких даних. Його асинхронна архітектура дозволяє ефективно працювати з великою кількістю одночасних з’єднань[31].

MongoDB було обрано як базу даних через її документно-орієнтовану модель, що дозволяє зберігати гнучкі, змінні за структурою дані. Це особливо корисно для збереження інформації про товари з різними характеристиками, а також історії переглядів, фільтрації, оцінок тощо[32]. Завдяки бібліотеці Mongoose, можлива побудова схем, валідація даних та моделювання зв’язків.

Таким чином, обраний стек MERN є оптимальним рішенням для створення функціонального інтернет-магазину з високою продуктивністю, масштабованістю та активною користувацькою взаємодією.

2.2 Структура сайту

Структура вебсайту інтернет-магазину з багатокритеріальними фільтрами та рекомендаційною системою реалізовано за принципом розділення функціональності на три основні рівні – клієнтський інтерфейс, серверну частину та базу даних, що забезпечує модульність та гнучкість при розробці та подальшій підтримці застосунку.

Структура сайту представлена на рисунку 2.1.

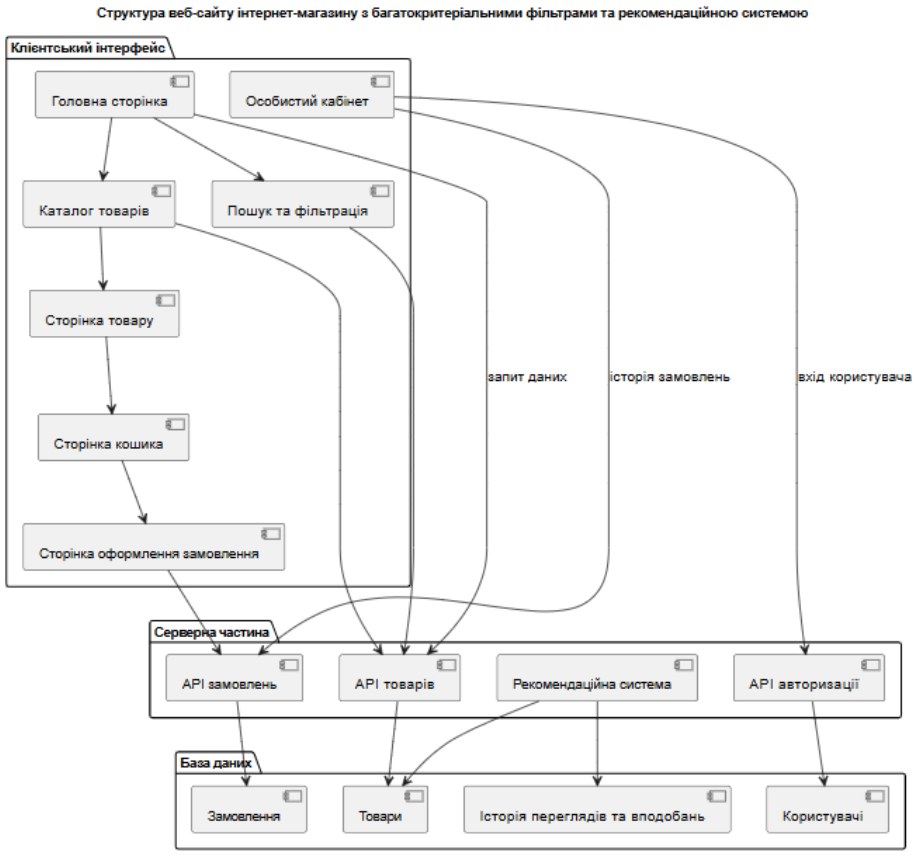

Рисунок 2.1 – Структура вебсайту

Джерело: розроблено автором

Клієнтський інтерфейс включає низку окремих компонентів, кожен з яких виконує певну роль у взаємодії з користувачем. Головна сторінка виступає точкою входу до системи, з якої користувач здійснює перехід до каталогу товарів або до модуля пошуку та фільтрації, де за допомогою

багатокритеріальних інструментів формується набір товарів, що відповідає заданим параметрам. Каталог товарів та сторінка окремого товару забезпечують детальне відображення інформації про продукт, що дозволяє користувачу приймати рішення про подальшу взаємодію, наприклад, додавання товару до кошика. Додатково, сторінка кошика та оформлення замовлення інтегровані між собою таким чином, що дані про обрані товари передаються для подальшої обробки при оформленні замовлення. Особистий кабінет, як окремий модуль, забезпечує доступ до історії замовлень та дозволяє користувачу здійснювати автентифікацію через API авторизації, що гарантує безпечну взаємодію з особистими даними.

Серверна частина, реалізована на базі Node.js, є ядром бізнес-логіки системи, що відповідає за обробку запитів клієнтського інтерфейсу через REST API. Зокрема, API товарів обробляє запити на отримання інформації про продукти, що надходять як із головної сторінки, так і із розділів каталогу та пошуку. Аналогічно, API замовлень відповідає за прийом даних про покупки, їх збереження та формування історії замовлень, доступної в особистому кабінеті користувача. Окремо функціонує API авторизації, що реалізує перевірку облікових даних та управління сесіями користувачів, забезпечуючи тим самим безпеку системи. Компонент рекомендаційної системи інтегровано до серверної логіки, і він взаємодіє із даними історії переглядів та вподобань, що дозволяє формувати персоналізовані пропозиції для кожного користувача на основі попередньої взаємодії із сайтом.

База даних MongoDB виступає сховищем основних сутностей системи. Вона містить окремі колекції, серед яких основними є колекції користувачів, товарів, замовлень, а також даних, що використовуються рекомендаційною системою (історія переглядів, вподобання)[33]. Серверні API безпосередньо звертаються до цих колекцій для виконання операцій читання, запису та оновлення даних. Наприклад, API авторизації здійснює запити до колекції користувачів для перевірки даних при вході до системи, а API товарів – до колекції товарів для відображення актуального асортименту продукції[34].

Рекомендаційна система, у свою чергу, використовує дані з колекцій історії переглядів та вподобань, аналізує їх та генерує набір рекомендованих товарів, який відображається користувачу в режимі реального часу.

Таким чином, взаємодія між компонентами реалізована наступним чином: клієнтський інтерфейс генерує HTTP-запити до серверних API, що, у свою чергу, обробляють запити та взаємодіють із базою даних для отримання або збереження необхідних даних[35]. Отримані дані повертаються у вигляді відповіді, що відображається на відповідних сторінках сайту. Ця інтегрованість дозволяє не лише забезпечити стабільну роботу застосунку, а й створити зручний інтерфейс для користувача, що надає можливість ефективного використання багатокритеріальної фільтрації та рекомендаційних алгоритмів при формуванні персоналізованих пропозицій.

2.3 Алгоритми роботи сайту

Алгоритм багатокритеріальної фільтрації товарів призначений для визначення найбільш релевантного підмножини товарів, що відповідає встановленим критеріям користувача (рис.2.2).

Критерії можуть включати категорію товару, ціновий діапазон, рейтинг, бренд, наявність знижок, а також будь-які інші атрибути, важливі для користувача. На початку алгоритму отримуються вхідні параметри фільтрації, які задають межі або конкретні умови відбору. Після цього виконується запит до бази даних для отримання початкового набору товарів. На кожному кроці перевіряється, чи відповідає поточний товар усім зазначеним умовам фільтрації. Якщо товар задовольняє всі критерії, він додається до кінцевого списку результатів; інакше – пропускається. Після обробки повного переліку товарів алгоритм повертає сформований відфільтрований список, який можна відобразити користувачеві або використати для подальшої аналітики.

Алгоритм колаборативної фільтрації використовується для формування персоналізованих рекомендацій на основі поведінки користувачів, які демонструють схожі вподобання (рис.2.3).

Рисунок 2.2 – Блок-схема алгоритму багатокритеріальної фільтрації

Джерело: розроблено автором

Рисунок 2.3 – Блок-схема алгоритму колаборативної фільтрації

Джерело: розроблено автором

Спочатку система отримує дані профілю поточного користувача, включно з історією переглядів, оцінок чи попередніх покупок. Далі виконується пошук користувачів із подібними патернами поведінки, тобто тих, хто здійснював аналогічні дії або проявляв схожі вподобання. Для кожного знайденого користувача визначаються їхні покупки та оцінки, після чого обчислюється ступінь схожості між поточним і цими користувачами. На основі узагальнених даних формується первинний список рекомендованих товарів, які потенційно можуть зацікавити поточного користувача. Якщо рекомендацій недостатньо або потрібна більша різноманітність, алгоритм доповнює список популярними товарами, щоб надати ширший вибір. Наприкінці користувачеві повертається узгоджений список рекомендацій, що враховує і його особисті вподобання, і колективний досвід інших користувачів.

Алгоритм пошуку з ранжуванням результатів забезпечує зручне та точне відображення товарів, які найбільше відповідають запиту користувача (рис.2.4).

Користувач формує пошуковий запит (ключові слова, фрази, тощо), який алгоритм спочатку обробляє шляхом пошуку відповідних записів у базі даних. У результаті формується первинний список товарів, що відповідають ключовим словам. Для кожного знайденого товару виконується розрахунок релевантності: враховується збіг із ключовими словами, наявність потрібних характеристик, популярність чи інші вагові коефіцієнти. Потім товари сортуються за ступенем релевантності у спадному порядку, щоб користувач отримував найдоцільніші результати пошуку на початку списку. За потреби додатково застосовуються фільтри, які уточнюють вибір (наприклад, діапазон цін, рейтинг чи наявність товару на складі), після чого формується фінальний список. Така послідовність дій дозволяє користувачеві швидко знайти потрібний товар та отримати релевантні пропозиції, що задовольняють його пошукові критерії.

На рисунку 2.5 представлена діаграма варіантів використання системи.

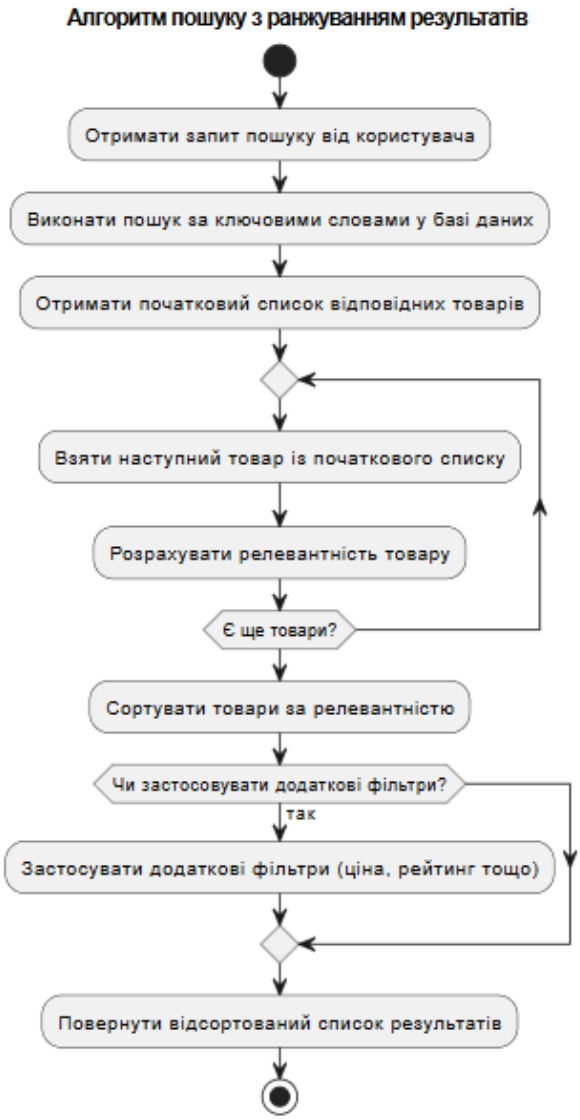

Рисунок 2.4 – Блок-схема алгоритму пошуку з ранжуванням результатів

Джерело: розроблено автором

Діаграма варіантів використання (Use Case Diagram) у контексті проєкту розробки вебсайту інтернет-магазину одягу «Bird Store» із багатокритеріальними фільтрами та рекомендаційною системою репрезентує ключові сценарії взаємодії користувачів із системою, відображаючи функціональні можливості та ролі, що беруть участь у роботі застосунку. Основною метою побудови такої діаграми є виявлення функціональності, яку система повинна реалізувати, виходячи з потреб користувачів і адміністраторів.

Рисунок 2.5 – Діаграма варіантів використання вебсайту

Джерело: розроблено автором

У межах діаграми передбачено два основних актори – «Покупець» та «Адміністратор», які взаємодіють із системою через відповідні варіанти використання. Покупець має доступ до функціоналу, пов’язаного із переглядом та пошуком товарів, персоналізацією, оформленням замовлень і роботою з обліковим записом. Зокрема, до таких сценаріїв належать: перегляд каталогу товарів, застосування багатокритеріальної фільтрації, перегляд детальної інформації про товар, додавання товару до кошика, оформлення замовлення, реєстрація та авторизація, перегляд історії покупок і отримання персоналізованих рекомендацій. Усі ці дії відображають ключову бізнес-

логіку інтернет-магазину, спрямовану на забезпечення зручності, швидкості та персоналізації процесу купівлі.

Зі свого боку, адміністратор взаємодіє з функціональністю, що забезпечує управління вмістом сайту та моніторинг діяльності користувачів. Йому доступні сценарії: керування товарами (додавання, редагування, видалення), перегляд і обробка замовлень, редагування параметрів фільтрації (щоб забезпечити актуальність критеріїв, згідно з новим асортиментом) та аналіз поведінки користувачів (для вдосконалення рекомендаційної системи або маркетингової стратегії).

Діаграма структурована таким чином, щоб забезпечити максимальну читабельність – сценарії згруповані логічно: функціонал покупця зосереджений з одного боку, адміністративні можливості – з іншого, а центральним елементом виступає система, яка реалізує ці варіанти використання. Такий підхід дозволяє не лише ідентифікувати вимоги до системи, а й встановити чіткі межі відповідальності між користувачами та адміністраторами. З погляду інженерного аналізу, діаграма Use Case є підґрунтям для подальшої декомпозиції функціональних вимог на технічні задачі – зокрема, побудови діаграм активностей, класів, компонентів та послідовностей. У випадку цього проєкту така діаграма відіграє важливу роль у формалізації інтерфейсу між фронтендом та бекендом, а також у структуризації API та авторизаційної логіки системи.

Діаграма активностей вебсайту представлена на рисунку 2.6.

Діаграма активностей, побудована для вебсайту інтернет-магазину одягу «Bird Store», відображає повний цикл взаємодії користувача із системою – від початкового перегляду товарів до завершення процесу покупки або її скасування. Вона побудована за принципом поділу відповідальностей між чотирма ролями: Користувач, Система, Адміністратор і Склад, що дозволяє чітко розмежувати дії відповідно до функціонального навантаження кожного учасника бізнес-процесу.

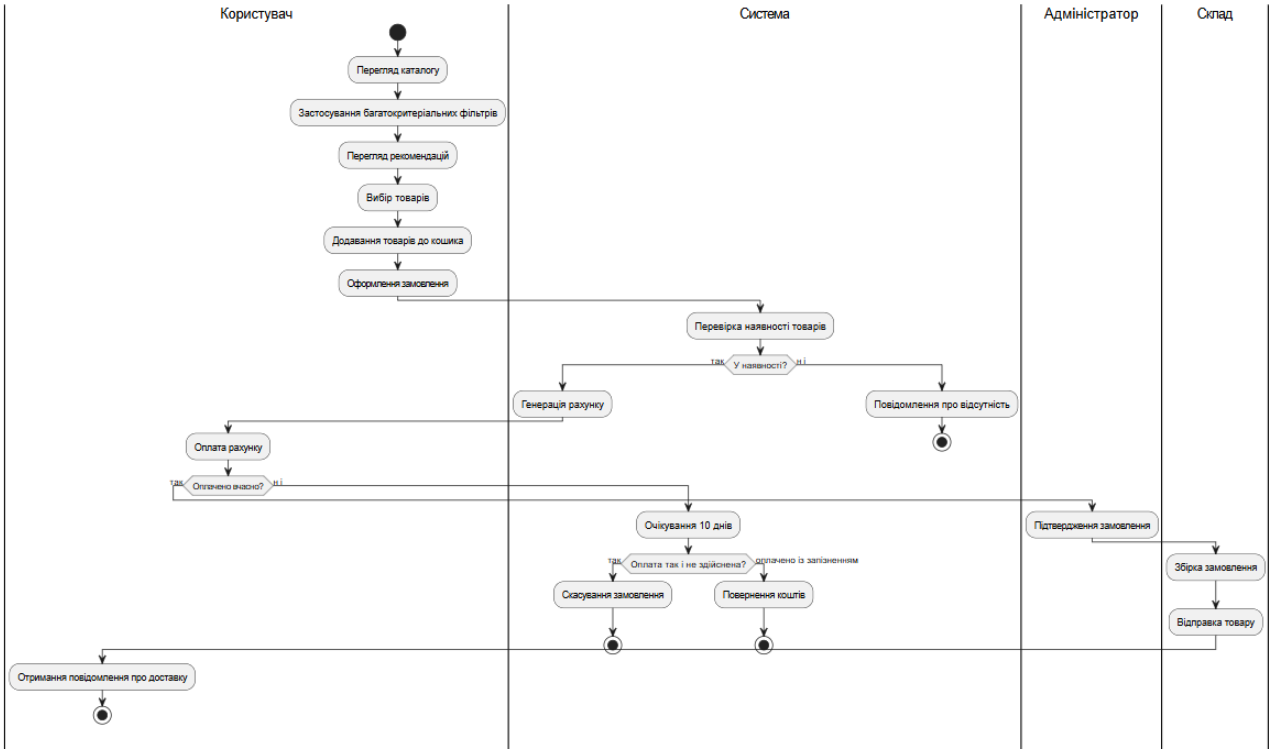

Рисунок 2.6 – Діаграма активностей вебсайту

Джерело: розроблено автором

У початковій фазі користувач здійснює навігацію по сайту: переглядає каталог товарів, застосовує багатокритеріальні фільтри для звуження вибору за параметрами (наприклад, розмір, колір, бренд, ціна, рейтинг), після чого отримує персоналізовані рекомендації, сформовані на основі історії переглядів, попередніх покупок або переваг схожих користувачів. На основі цих дій користувач додає товари до кошика та переходить до етапу оформлення замовлення.

Після підтвердження замовлення система перевіряє наявність товарів на складі. Якщо замовлений товар наявний, система автоматично генерує рахунок на оплату, який надсилається користувачу. Оплата має бути здійснена протягом визначеного терміну (10 днів). У разі своєчасної оплати система фіксує факт транзакції, і до процесу залучається адміністратор, який підтверджує замовлення та передає інформацію на склад для фізичної обробки. На цьому етапі складська служба виконує збірку замовлення, готує

товари до відправлення, і в подальшому здійснюється відвантаження, про що користувача інформують через відповідне повідомлення.

Якщо ж рахунок не було оплачено протягом зазначеного періоду, система здійснює перевірку повторно. У випадку несвоєчасної або відсутньої оплати ініціюється процедура скасування замовлення, а в разі затриманої, але вже здійсненої оплати – система автоматично виконує повернення коштів. Обидва ці сценарії призводять до завершення процесу без доставки товару, проте з урахуванням прав користувача на повернення коштів або з повідомленням про скасування замовлення.

Ця діаграма також моделює логіку автоматизованої взаємодії між компонентами системи, включаючи обробку подій, таймерів, умовних переходів і обробку виключних ситуацій (нестача товару, прострочення оплати). Вона служить не лише як інструмент аналізу бізнес-логіки, але й як основа для реалізації серверної логіки (наприклад, REST-ендпойнтів для зміни статусу замовлення), інтеграції з платіжними сервісами, а також для моделювання взаємодії з інтерфейсом користувача (наприклад, повідомлення, статуси замовлення, рекомендаційні блоки).

Таким чином, діаграма активностей демонструє лінійно-розгалужену структуру бізнес-процесу, враховуючи не лише позитивний сценарій (успішне оформлення і доставка), а й альтернативні сценарії обробки виняткових ситуацій, що робить її незамінною при проектуванні, тестуванні і документуванні логіки системи «Bird Store».

2.4 Створення бази даних

Структура бази даних представлена на рисунку 2.7.

У процесі проєктування системи було використано документно-орієнтовану базу даних MongoDB, що належить до класу нереляційних (NoSQL) СУБД[36]. Такий вибір обумовлений потребою в гнучкій схемі зберігання даних, високою продуктивністю при роботі з великим обсягом напівструктурованої інформації, а також зручністю масштабування. MongoDB

дозволяє зберігати дані у вигляді BSON-документів (розширення формату JSON) [37], що робить її ідеальною для динамічних застосунків, таких як інтернет-магазин з багатокритеріальними фільтрами та рекомендаційною системою.

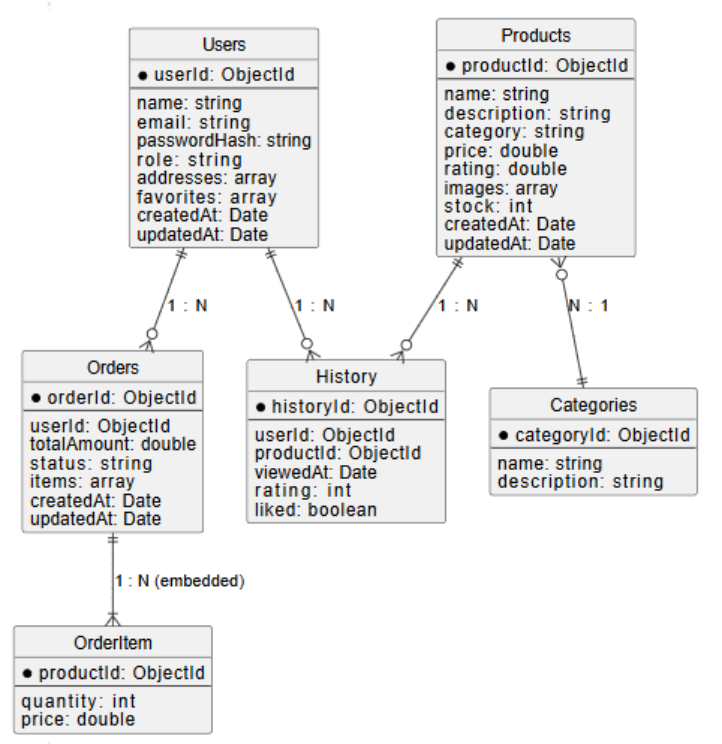

Рисунок 2.7 – Структура бази даних

Джерело: розроблено автором

Основні колекції бази даних:

- Users – колекція, що зберігає дані користувачів. Основні поля: userId (типу ObjectId, автоматично створюється MongoDB), email, name, passwordHash, role, createdAt, updatedAt.

За потреби, до документів можуть додаватися вкладені масиви адрес доставки чи улюблених товарів.

- Products – колекція товарів, що включає: productId (ObjectId), name, description, category, price, rating, images (масив URL-адрес зображень), stock, createdAt, updatedAt.

Гнучка структура дозволяє кожному товару мати унікальний набір характеристик, що полегшує фільтрацію.

- Orders – колекція замовлень, яка містить: orderId (ObjectId), userId (зв’язок з Users), totalAmount, status, createdAt, updatedAt.

Кожне замовлення зберігає базову інформацію про транзакцію.

- OrderItems – допоміжна колекція з деталями позицій замовлення: orderItemId, orderId (зв’язок з Orders), productId (зв’язок з Products), quantity, price.

- History – колекція для фіксації взаємодій користувача з товарами: historyId, userId, productId, viewedAt, liked (булеве значення), rating.

Ці дані використовуються для побудови рекомендацій на основі історії переглядів та оцінок.

Зв’язки між колекціями реалізуються за допомогою посилань (ObjectId) і обробляються на рівні додатку. Наприклад, зв’язок між користувачами і їхніми замовленнями реалізується через поле userId у колекції Orders.

Таким чином, використання MongoDB дозволяє досягти гнучкої, масштабованої архітектури бази даних, яка ефективно підтримує ключовий функціонал інтернет-магазину.

Висновки до розділу

Було розроблено єдину архітектуру системи, що ґрунтується на чіткому розділенні функціональних компонентів: клієнтської частини, серверної логіки та бази даних. У процесі проєктування передбачалося використання React для створення динамічного та адаптивного інтерфейсу користувача, Node.js для реалізації серверної логіки з використанням неблокуючих асинхронних операцій, а також MongoDB як документно-орієнтованої бази даних для зберігання гнучкої та змінної за структурою інформації.

Вибрані інструментальні засоби були обрані з урахуванням їхньої доведеної ефективності в аналогічних проєктах та широкого використання в промисловій розробці. React, Node.js і MongoDB добре зарекомендували себе

як надійні та масштабовані рішення для реалізації складної логіки в системах електронної комерції, що стало вирішальним фактором для їхнього використання в даному проєкті.

У межах архітектурного проєктування також було передбачено реалізацію ключових алгоритмів: багатокритеріальної фільтрації товарів, алгоритму колаборативної фільтрації для персоналізованих рекомендацій, а також пошуку з ранжуванням результатів. Ці алгоритми спроєктовано з урахуванням потреб користувачів у швидкому доступі до релевантного контенту, покращенні користувацького досвіду та ефективному використанні історичних даних взаємодій.

Структура бази даних розроблялася з урахуванням специфіки MongoDB, де дані зберігаються у форматі BSON-документів. Було передбачено створення таких основних колекцій: Users, Products, Orders, OrderItems та History. Кожна колекція містить атрибути, необхідні для виконання функцій автентифікації, персоналізації, обробки замовлень та побудови рекомендацій. Зв’язки типу «один до багатьох» реалізуються через посилання (`ObjectId`) між документами, що дозволяє зберігати логічну цілісність даних навіть у документно-орієнтованій моделі.

Завдяки такій структурі системи забезпечується гнучкість, розширюваність та можливість реалізації складної бізнес-логіки на рівні як клієнта, так і сервера.

РОЗДІЛ 3
ТЕСТУВАННЯ ВЕБСАЙТУ

3.1 Інтерфейс користувача

Інтерфейс онлайн-магазину Bird Store розроблений відповідно до принципів сучасного веб-дизайну: адаптивний, зручний і логічно структурований, він забезпечує комфортну та інтуїтивну взаємодію користувача з платформою (рис.3.1).

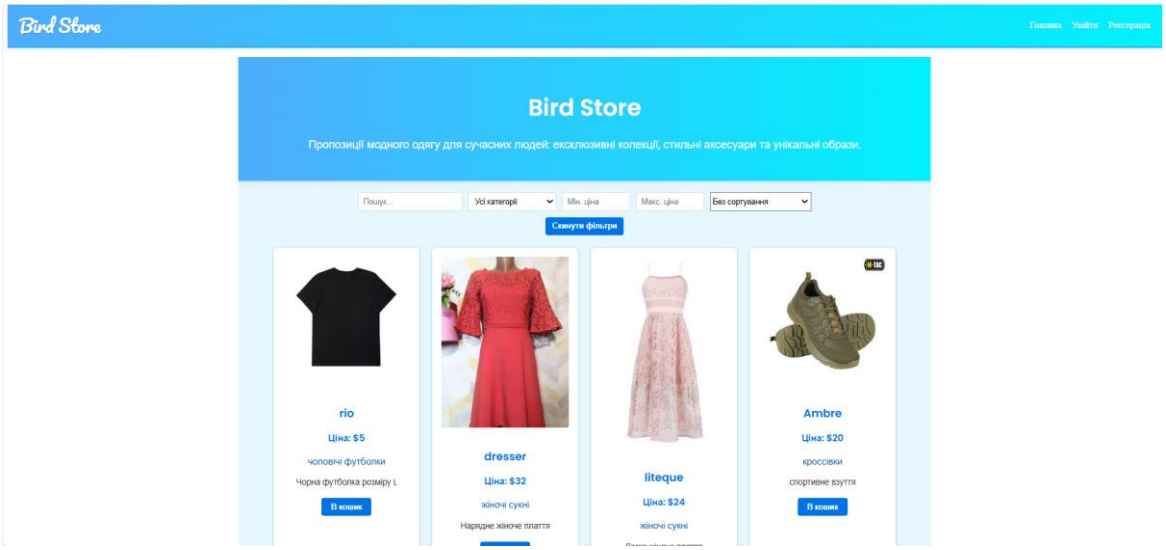

Рисунок 3.1 – Головна сторінка

Джерело: розроблено автором

Головна сторінка сайту оформлена у простому, але виразному стилі. Основний акцент зроблено на банері, де вказано назву магазину та короткий опис асортименту: зручний одяг для щоденного використання. Це дає змогу користувачу з першого погляду зрозуміти тематику сайту.

Для полегшення навігації реалізовано зручний пошук та фільтрацію товарів. Користувач може швидко знайти потрібну річ, обравши категорію (наприклад, футболки чи штани), розмір, стиль або колір. Такий підхід спрощує вибір і допомагає уникнути зайвих переходів по сайту.

У верхній частині сторінки розміщено головне меню. Через нього можна потрапити до основних розділів: вхід, реєстрація, перегляд профілю, перегляд

кошика, а також адміністративна частина (доступна тільки для адміністратора). Це меню працює однаково добре на різних пристроях – як на комп’ютерах, так і на телефонах, що робить сайт зручним для більшості користувачів.

Інтерфейс сайту витримано в єдиному стилі: всі кнопки, повідомлення, текстові поля мають однакове оформлення з використанням синіх відтінків. Це не тільки покращує зовнішній вигляд, а й полегшує сприйняття контенту.

Основні функції, такі як додавання товарів у кошик, оформлення замовлення чи редагування профілю, працюють без оновлення сторінки. Це зроблено для того, щоб користувач не втрачав час і міг взаємодіяти з сайтом швидше. Така інтерактивність покращує зручність у користуванні, особливо на мобільних пристроях.

Уся структура побудована таким чином, щоб користувач легко орієнтувався в магазині й міг виконати потрібні дії без складних інструкцій.

3.2 Тестування функціоналу

У процесі тестування вебсайту було використано метод функціонального тестування, оскільки перевірка охоплювала ключові користувацькі сценарії: реєстрацію, авторизацію, перегляд та фільтрацію товарів, додавання до кошика, оформлення замовлення через форми, а також роботу персоналізованих рекомендацій. Це відповідає загальноприйнятому визначенню функціонального тестування, згідно з яким основною метою є перевірка коректності реалізації функцій відповідно до вимог.

Метою цього етапу було перевірити основні функції сайту інтернет-магазину «Bird Store» та переконатись, що вони працюють правильно. Аналіз також дозволив виявити можливі помилки в роботі та оцінити зручність користування.

Окрему увагу було приділено формі реєстрації, оскільки вона є першим елементом взаємодії користувача з сайтом. Перевірялися різні варіанти вводу: як правильні, так і помилкові. Зокрема, тестувалися ситуації з неправильними

логінами, занадто короткими паролями, відсутністю заповнення обов'язкових полів тощо.

Також було перевірено, чи правильно працює перехід між формами реєстрації та входу. Тестувалася валідація введених даних як на стороні клієнта (тобто у браузері), так і на сервері. Крім того, перевірялися збереження сесії користувача після входу та правильне відображення особистого кабінету після авторизації (рис.3.2).

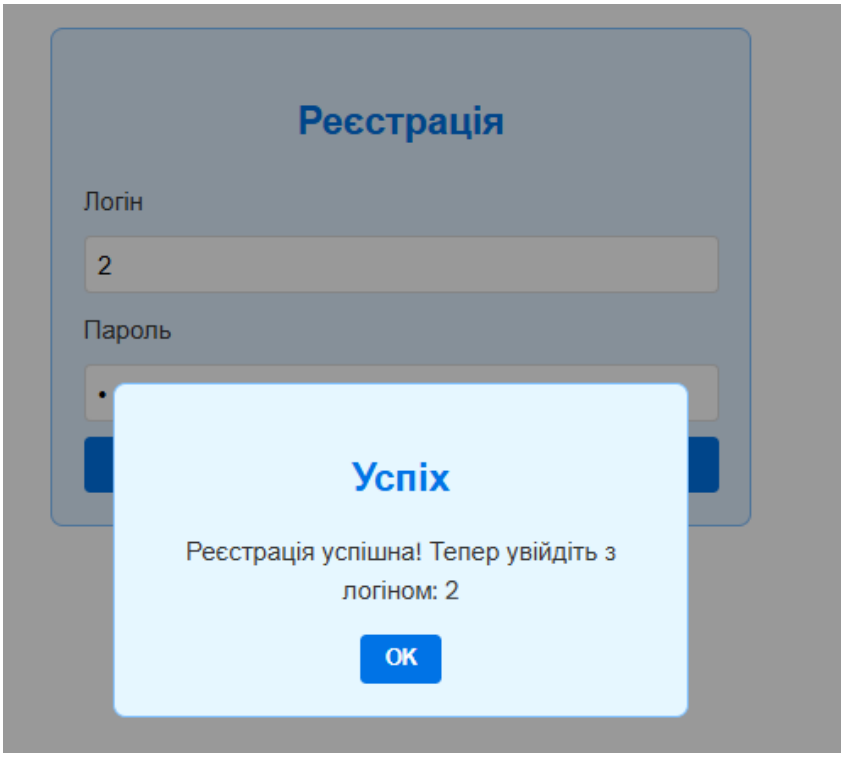

Рисунок 3.2 – Форма реєстрації користувача

Джерело: розроблено автором

У процесі тестування модуля фільтрації основна увага приділялася перевірці того, як система реагує на задані користувачем параметри. Серед основних критеріїв перевірки були: вибір категорії товару та зазначення діапазону цін.

Було протестовано як прості випадки (з фільтрацією лише за одним параметром), так і складніші комбінації з кількома умовами одночасно. Крім того, окремо перевірялась реакція системи на запити, які не мали результатів

– наприклад, коли не існує товарів за заданими критеріями. У всіх перевірених випадках система коректно відображала відповідний список або повідомлення про відсутність результатів (рис.3.3).

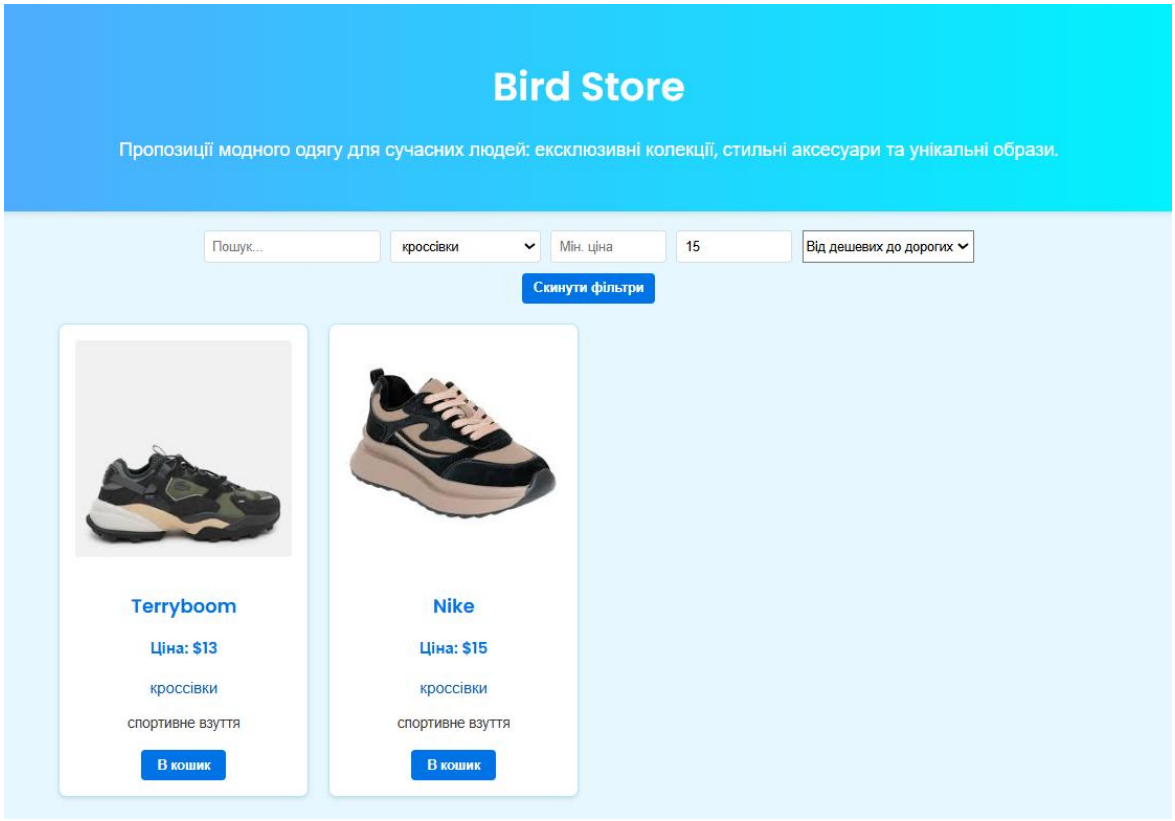

Рисунок 3.3 – Застосування багатокритеріальної фільтрації до каталогу товарів

Джерело: розроблено автором

При використанні декількох фільтрів одночасно система коректно виводила результати, що відповідали всім обраним параметрам. Це дозволило переконатися, що фільтрація працює як при окремих, так і при комбінованих запитах до бази даних.

Ще одним важливим елементом є функціональність кошика. Було перевірено можливість додавання товарів, зміни їх кількості, видалення окремих позицій, а також автоматичне оновлення загальної суми.

Тестування охоплювало як звичайне, так і швидке послідовне додавання кількох товарів. Інтерфейс коректно реагував на зміни без перезавантаження

сторінки. Також перевірено, чи зберігаються дані кошика після оновлення сторінки або виходу з облікового запису – збереження працювало коректно (рис.3.4).

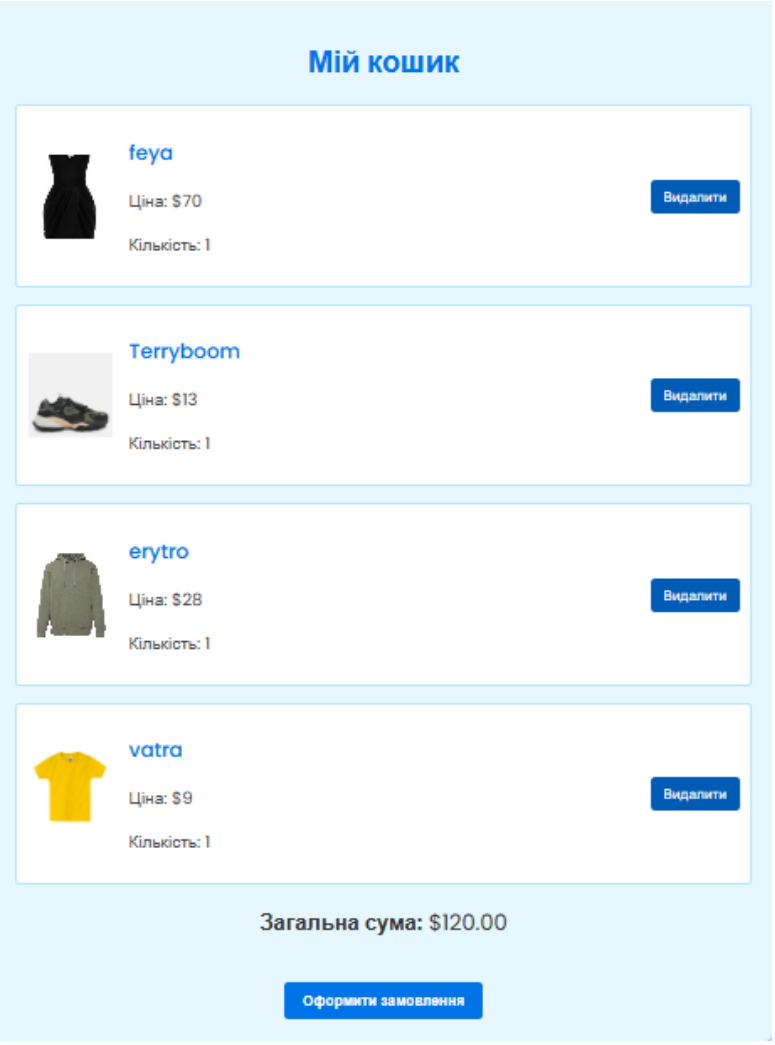

Рисунок 3.4 – Інтерфейс кошика з динамічним оновленням кількості товарів

Джерело: розроблено автором

У процесі тестування була перевірена можливість зміни кількості одиниць товару в кошику. При зміні значень система миттєво перераховувала загальну суму замовлення. Також було протестовано ситуацію з повним видаленням усіх товарів – у цьому випадку з’являлося повідомлення про порожній кошик.

Окрему увагу приділено перевірці форми оформлення замовлення. Тестувалося заповнення полів доставки, вибір методу оплати та підтвердження замовлення. Крім того, перевірялась поведінка системи у разі, якщо користувач залишає поля незаповненими. У таких випадках з’являлися відповідні повідомлення про помилки, що не дозволяли відправити неповну форму (рис.3.5).

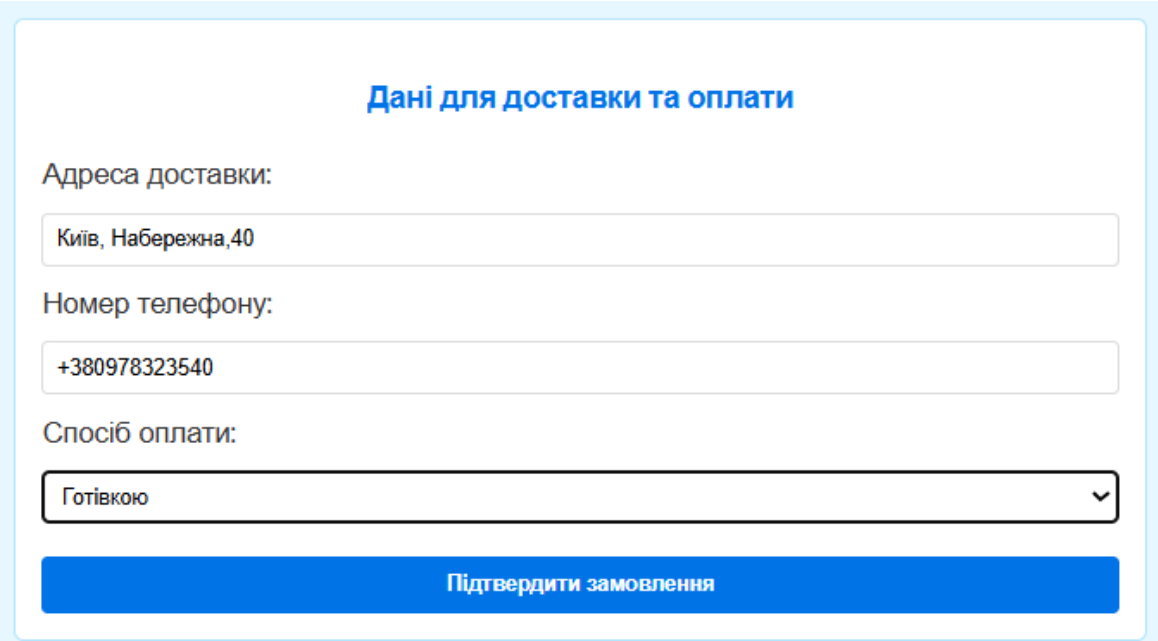

Рисунок 3.5 – Екран оформлення замовлення з формою для введення контактних даних

Джерело: розроблено автором

Було протестовано блок із персоналізованими рекомендаціями. Для цього змінювалась поведінка користувача на сайті – наприклад, здійснювались переходи до певних категорій товарів або переглядались окремі бренди. Після цього перевірявся склад товарів, що відображаються в блоці рекомендацій.

У результаті тестування зафіксовано, що система оновлює список пропозицій відповідно до попередніх дій користувача. Виводилися товари зі схожих категорій або з подібними характеристиками. Це свідчить про те, що механізм підбору рекомендацій враховує активність користувача на сайті (рис.3.6).

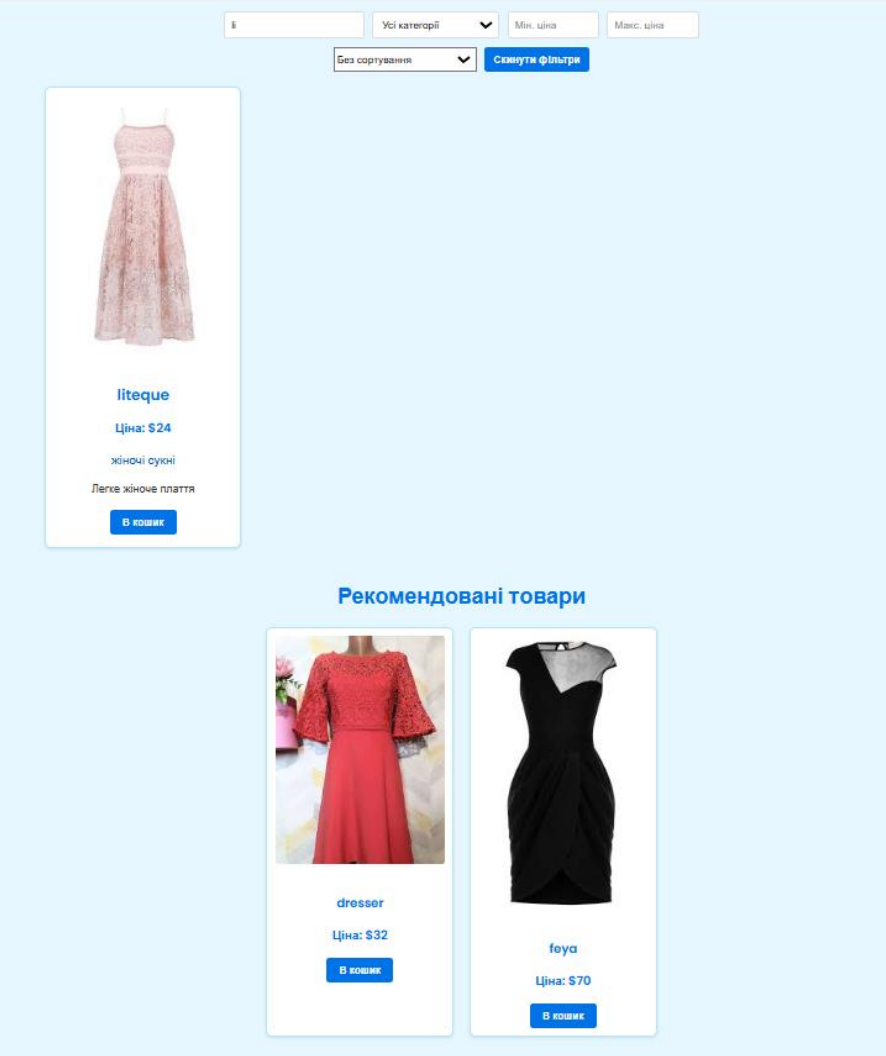

Рисунок 3.6 – Відображення персоналізованих товарів на головній сторінці

Джерело: розроблено автором

Система зберігає інформацію про товари, які користувач переглядав раніше, і на основі цих даних оновлює список рекомендацій. Для цього використовується підхід, що враховує схожість між товарами. Це дозволяє показувати схожі за категорією або характеристиками позиції.

Основні функції сайту було перевірено вручну, використовуючи найтиповіші сценарії поведінки користувача – перегляд товарів, фільтрація, додавання в кошик, оформлення замовлення тощо. Усі ключові елементи працювали стабільно й відповідали очікуванням. Окремо були зафіксовані незначні недоліки – зокрема, трохи довша обробка запитів при складній

фільтрації та обмежена кількість рекомендацій для нових користувачів. Ці моменти зафіксовано для можливого вдосконалення в майбутньому.

3.3 Аналіз результатів тестування

Під час функціонального тестування сайту онлайн-магазину перевірялась робота всіх основних розділів. Охоплено ключові сценарії: перегляд товарів, фільтрація, реєстрація, авторизація, оформлення замовлення, робота з кошиком, система рекомендацій і тестові платіжні дії.

Система пошуку та фільтрації коректно реагувала на введені параметри. Запити з кількома критеріями оброблялись без помилок і повертали відповідні результати. Кошик дозволяє додавати й видаляти товари, а також зберігає дані після оновлення сторінки.

Процес оформлення замовлення містить перевірку правильності заповнених полів, що допомагає уникнути типових помилок. Авторизація та реєстрація працюють стабільно, включно з можливістю вийти з акаунту, відновити пароль і повторно увійти.

Під час перевірки не було виявлено серйозних збоїв або критичних помилок. Усі основні функції працювали відповідно до заданих вимог і забезпечували нормальний користувацький досвід.

Висновки до розділу

У цьому розділі завершено перевірку основних компонентів вебсайту. Це дозволило оцінити, наскільки ефективно працюють реалізовані функції. Було підтверджено, що взаємодія між клієнтською частиною (інтерфейсом) і сервером відбувається коректно: дані передаються швидко, а вміст оновлюється без затримок.

Модуль фільтрації успішно обробляє запити з кількома параметрами одночасно. Навіть за великої кількості умов система зберігає стабільність і продуктивність суттєво не знижується. Під час тестування змін параметрів фільтрації інтерфейс залишався зручним у користуванні.

Функціонал персоналізованих рекомендацій коректно враховує попередню активність користувача. Виводяться товари, які дійсно відповідають інтересам, що дозволяє зробити висновок про правильну роботу відповідного механізму.

Загалом, проведене тестування показало, що сайт відповідає технічному завданню, справляється з навантаженням і може використовуватись у реальному середовищі.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було здійснено повний цикл проєктування, розробки та тестування вебсайту інтернет-магазину одягу Bird Store, орієнтованого на сучасні вимоги до зручності, персоналізації та масштабованості. Вивчення предметної області дозволило виявити основні тенденції у сфері електронної комерції та підтвердити, що застосування багатокритеріальних фільтрів і рекомендаційних систем суттєво підвищує якість користувацького досвіду, сприяючи точнішому пошуку товарів та підвищенню конверсії.

На основі аналізу наявних рішень та відповідної наукової літератури було ідентифіковано ключові алгоритмічні підходи до реалізації персоналізованих механізмів. Було розглянуто переваги та недоліки простих фільтраційних методів (зокрема зваженого ранжування), методів колаборативної фільтрації (user- та item-based), модельних підходів на основі матричної факторизації, а також нейронних мереж і гібридних систем. Це дозволило сформувати уявлення про оптимальні варіанти впровадження залежно від доступних ресурсів, структури даних та очікуваної точності рекомендацій.

У другому розділі було обґрунтовано вибір технологій для розробки: стек MERN (MongoDB, Express.js, React, Node.js) надавав змогу ефективно організувати як фронтенд, так і бекенд частину системи. Архітектура сайту була побудована із дотриманням принципів модульності та розділення відповідальностей: клієнтська частина забезпечувала зручний інтерфейс із можливістю багатокритеріального пошуку, серверна обробляла запити та взаємодіяла з базою даних, а модулі авторизації, реєстрації та рекомендаційної системи працювали незалежно, що дозволяло масштабувати або замінювати окремі компоненти без втрати функціональності.

База даних MongoDB використовувалась для зберігання різноманітних структурованих і напівструктурованих даних, зокрема: профілів користувачів, каталогу товарів, історії переглядів і замовлень, а також мета-даних для

генерації рекомендацій. Це забезпечувало високу гнучкість у роботі з різнорідною інформацією, зокрема у випадку розширення функціоналу чи зміни структури товарів.

У рамках функціонального тестування було перевірено всі ключові функції вебсайту: багатокритеріальну фільтрацію, обробку пошукових запитів, авторизацію, управління профілем, роботу з кошиком, оформлення замовлень і динамічне формування рекомендацій. Фільтрація дозволяла комбінувати різні критерії (наприклад, бренд, ціну, рейтинг, матеріал тощо) та отримувати точні результати без затримок. Система рекомендацій формувала добірки на основі історії переглядів і поведінкових патернів, адаптуючи контент під конкретного користувача в режимі реального часу.

Під час тестування не було виявлено критичних помилок, що дозволяє зробити попередній висновок про задовільну роботу реалізованого функціоналу. Виявлені дрібні неточності були усунуті в процесі доопрацювання, що свідчить про певну гнучкість структури сайту.

У підсумку, результати кваліфікаційної роботи підтвердили досягнення поставленої мети – розробка вебсайту роздрібної торгівлі з використанням багатокритеріальних фільтрів і рекомендаційної системи, було проаналізовано підходи до фільтрації та рекомендацій, спроєктовано архітектуру сайту з урахуванням багатокритеріальних запитів, реалізовано функціонал інтернет-магазину на основі технологій MERN.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Сидоренко М.І., Гриценко О.М. Порівняльний аналіз алгоритмів колаборативної фільтрації для рекомендацій товарів в електронних магазинах. // Вісник Харківського національного університету імені В.Н. Каразіна. Серія: Математика, прикладна математика та інформатика. – 2022. – № 119. – С. 35-42.

2 .Смирнов Д.О., Ткаченко А.В. Оптимізація продуктивності веб-додатків електронних магазинів за допомогою технології Serverless. // Наукові праці Одеського національного політехнічного університету. – 2022. – № 1. – С. 125-130.

3. Фелько М.Р. Дослідження методів побудови рекомендаційних систем для вирішення задач в електронній комерції, методи колаборативної фільтрації. // International Electronic Scientific Journal "Science Online". – 2021.

4. Кляйнер, М. Designing data-intensive applications: The big data ecosystem. O'Reilly Media. – 2021. – № 17. – С. 590.

5. Wang, J. & Liu, X. (2015). Hybrid Web Content Filtering for Personalized Information Retrieval. Information Processing & Management, 51(3), 547–565.

6. Kumar, R. & Singh, A. (2015). A Survey of Web Content Filtering Techniques: Current Approaches and Future Challenges. Journal of Network and Computer Applications, 55, 67–86.

7. Patel, M. & Desai, R. (2015). Machine Learning Approaches for Real-time Web Content Filtering. IEEE Access, 3, 2382–2395.

8. Bose, I. & Kumar, S. (2015). Context-Aware Filtering of Web Content: Techniques, Frameworks, and Research Directions. International Journal of Web Information Systems, 11(1), 72–89.

9. Ahmed, M. & Rahman, T. (2015). Adaptive Filtering for Dynamic Web Content. Journal of Web Engineering, 14(3), 245–262.

10. Singh, P. & Gupta, N. (2015). Internet Content Filtering Using Hybrid Algorithms: A Comparative Study. In Proceedings of the International Conference on Cyber-Physical Systems and Internet of Things, 45–52.

11. Chen, L. & Zhang, Y. (2015). Effective Filtering of Web Content Using Neural Networks: A Case Study. Journal of Information Science, 41(2), 151–165.

12. Zhang, Y. et al. (2015). Personalized Information Filtering on the Web: Techniques and Applications. ACM Computing Surveys, 48(3), Article 42.

13. Hidasi, B., Karatzoglou, A., Baltrunas, L. та Tikk, D. (2015). Session-based Recommendations with Recurrent Neural Networks. arXiv:1511.06939.

14. Lu, Z. та Shen, H. (2015). An Accuracy-Assured Privacy-Preserving Recommender System for Internet Commerce. arXiv:1505.07897.

15. Zhu, J., He, P., Zheng, Z. та ін. (2015). A Privacy-Preserving QoS Prediction Framework for Web Service Recommendation. arXiv:1502.06084.

16. Madadipouya, K. (2015). A Location-Based Movie Recommender System Using Collaborative Filtering. arXiv:1508.01696.

17. Solanki, S. та Batra, S. (2015). Recommender System using Collaborative Filtering and Demographic Characteristics of Users. International Journal on Recent and Innovation Trends in Computing and Communication, том 3, вип. 7, 2015.

18. Thorat, P. B., Goudar, R. M. та Barve, S. (2015). Survey on Collaborative Filtering, Content-Based Filtering and Hybrid Recommendation System. International Journal of Computer Applications, том 110, вип. 4, січень 2015.

19. Blanda, S. (2015). Online Recommender Systems – How Does a Website Know What I Want?. Published by the American Mathematical Society, 25 травня 2015.

20. Kulkarni, A. та Wagh, S. (2015). A Study of Recommender Systems with Hybrid Collaborative Filtering. In: Proceedings of the 9th International Conference on Recommender Systems, 2015.

21. Li, X. et al. (2015). A Survey of Cross-Domain Recommender Systems. Journal of Data Science, том 13, вип. 3, с. 235–256, 2015.

22. Chen, Y. та Zhang, M. (2015). An Ensemble Approach for Improving Recommendation Accuracy in Recommender Systems. Journal of Intelligent Information Systems, том 44, вип. 2, с. 395–412, 2015.

23. Gupta, R. (2015). User Satisfaction in Recommender Systems: An Empirical Study. International Journal of Human-Computer Studies, том 72, вип. 5, с. 515–528, 2015.

24. Ahmed, S. та Kumar, R. (2015). Privacy-Preserving Recommender Systems: Challenges and Solutions. Journal of Computer Security, том 23, вип. 2, с. 185–202, 2015.

25. Amazon. URL: https://www.amazon.com/ (дата звернення: 20.03.2025)

26. Rozetka. URL: https://rozetka.com.ua/ua/ (дата звернення: 20.03.2025)

27. AliExpress. URL: https://www.aliexpress.com/ (дата звернення: 20.03.2025)

28. React Node.js MongoDB Full-Stack Pinterest App Tutorial. URL: https://www.youtube.com/watch?v=x4LhjMk-mkA (дата звернення 08.04.2025)

29. How I Integrated a MongoDB Database into my React.js Project. URL: https://javascript.plainenglish.io/how-i-integrated-a-mongodb-database-into-my-react-js-project-6cdc331923d3 (дата звернення 08.04.2025)

30. Best Practices for React, Node.js API and MongoDB. URL: https://www.linkedin.com/pulse/best-practices-react-nodejs-api-mongodb-thomas-jay (дата звернення 08.04.2025)

31. MongoDB, Express, React & Node.js Full Project. URL: https://www.youtube.com/watch?v=korRfKTDoxE (дата звернення: 20.03.2025)

32. MERN Part I: Building RESTful APIs with Node.js and Express. URL: https://medium.com/weekly-webtips/building-restful-apis-with-node-js-and-express-a9f648219f5b (дата звернення 08.04.2025)

33. React POST requests with Express/Node and MongoDB. URL: https://stackoverflow.com/questions/50617351/react-post-requests-with-express-node-and-mongodb (дата звернення 08.04.2025)

34. How To Use MERN Stack: A Complete Guide. URL: https://www.mongodb.com/en-us/resources/languages/mern-stack-tutorial (дата звернення 08.04.2025)

35. MERN stack tutorial: The Complete Guide with Examples. URL: https://deadsimplechat.com/blog/mern-stack-the-complete-guide/ (data zvernennia: 08.04.2025)

36. Building a Simple CRUD Application with MERN Stack. URL: https://www.digitalocean.com/community/tutorials/build-a-to-do-application-using-docker-and-mern-stack (data zvernennia: 08.04.2025)

37. Secure Your MERN Stack Application: Best Practices. URL: https://www.mongodb.com/blog/post/secure-your-mern-stack-application-best-practices (дата звернення : 08.04.2025)

ДОДАТКИ

Додаток А «Лістинг програмного коду»

birdshop-backend/package.json

```
{
  "name": "birdshop-backend",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js",
    "seed": "node seed.js"
  },
  "dependencies": {
    "bcrypt": "^5.1.0",
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "jsonwebtoken": "^8.5.1",
    "mongoose": "^6.7.0",
    "multer": "^1.4.5-lts.1",
    "stripe": "^11.0.0"
  },
  "devDependencies": {
    "nodemon": "^2.0.20"
  }
}
```

birdshop-backend/.env

```
PORT=5000
MONGODB_URI=mongodb://127.0.0.1:27017/birdshop
JWT_SECRET=my_super_secret_key
STRIPE_SECRET_KEY=sk_test_example123
```

birdshop-backend/server.js

```
require('dotenv').config();
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');

const userRoutes = require('./routes/userRoutes');
const productRoutes = require('./routes/productRoutes');
const cartRoutes = require('./routes/cartRoutes');
const orderRoutes = require('./routes/orderRoutes');
const reviewRoutes = require('./routes/reviewRoutes');

const app = express();
const PORT = process.env.PORT || 5000;

mongoose.connect(process.env.MONGODB_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
  .then(() => console.log('> Підключено до MongoDB (birdshop)'))
  .catch(err => console.error('Помилка підключення до MongoDB:', err));

app.use(cors());
```

```
app.use(express.json());
app.use('/uploads', express.static('uploads'));

app.use('/api/users', userRoutes);
app.use('/api/products', productRoutes);
app.use('/api/cart', cartRoutes);
app.use('/api/orders', orderRoutes);
app.use('/api/reviews', reviewRoutes);

app.listen(PORT, () => {
  console.log(`> Сервер запущено на порті ${PORT}`);
});
```

birdshop-backend/models/User.js

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  login: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  role: { type: String, default: 'user' }
}, { timestamps: true });

module.exports = mongoose.model('User', userSchema);
```

birdshop-backend/models/Product.js

```
const mongoose = require('mongoose');

const productSchema = new mongoose.Schema({
  name: { type: String, required: true },
  description: { type: String, default: '' },
  price: {
    type: Number,
    required: true,
    min: [0, 'Ціна не може бути від’ємною']
  },
  category: { type: String, default: 'other' },
  imagePath: { type: String, default: '' },
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Product', productSchema);
```

birdshop-backend/models/Cart.js

```
const mongoose = require('mongoose');

const cartItemSchema = new mongoose.Schema({
  product: { type: mongoose.Schema.Types.ObjectId, ref: 'Product' },
  quantity: { type: Number, default: 1 }
});

const cartSchema = new mongoose.Schema({
  user: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  items: [cartItemSchema]
}, { timestamps: true });

module.exports = mongoose.model('Cart', cartSchema);
```

birdshop-backend/models/Order.js

```
const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({
  user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true
},
  items: [{
    product: { type: mongoose.Schema.Types.ObjectId, ref: 'Product' },
    quantity: Number
  }],
  totalAmount: { type: Number, default: 0 },
  status: { type: String, default: 'Processing' },
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Order', orderSchema);
```

birdshop-backend/models/Review.js

```
const mongoose = require('mongoose');

const reviewSchema = new mongoose.Schema({
  user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true
},
  product: { type: mongoose.Schema.Types.ObjectId, ref: 'Product', required:
true },
  rating: {
    type: Number,
    required: true,
    min: [1, 'Мінімальний рейтинг 1'],
    max: [5, 'Максимальний рейтинг 5']
  },
  text: { type: String, default: '' },
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Review', reviewSchema);
```

birdshop-backend/controllers/userController.js

```
const User = require('../models/User');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');

exports.registerUser = async (req, res) => {
  try {
    const { login, password } = req.body;
    const existing = await User.findOne({ login });
    if (existing) return res.status(400).json({ message: 'Користувач із таким
логіном вже існує' });
    const hashedPass = await bcrypt.hash(password, 10);
    const newUser = new User({ login, password: hashedPass });
    await newUser.save();
    res.status(201).json({ message: 'Користувача створено' });
  } catch (error) {
    console.error('Помилка реєстрації:', error);
    res.status(500).json({ message: 'Помилка реєстрації', error });
  }
};

exports.loginUser = async (req, res) => {
  try {
    const { login, password } = req.body;
```

```
    const user = await User.findOne({ login });
    if (!user) return res.status(401).json({ message: 'Невірний логін або
пароль' });
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(401).json({ message: 'Невірний логін або
пароль' });
    const token = jwt.sign({ userId: user._id, role: user.role },
process.env.JWT_SECRET, { expiresIn: '1d' });
    res.status(200).json({ message: 'Успішний вхід', token, user: { role:
user.role } });
  } catch (error) {
    console.error('Помилка логіну:', error);
    res.status(500).json({ message: 'Помилка логіну', error });
  }
};

exports.getProfile = async (req, res) => {
  try {
    const user = await User.findById(req.user.userId);
    if (!user) return res.status(404).json({ message: 'Користувача не
знайдено' });
    res.json({ login: user.login });
  } catch (error) {
    res.status(500).json({ message: 'Помилка отримання профілю', error });
  }
};

exports.updateUser = async (req, res) => {
  try {
    const { login, password } = req.body;
    const user = await User.findById(req.user.userId);
    if (!user) return res.status(404).json({ message: 'Користувача не
знайдено' });
    if (login) user.login = login;
    if (password) {
      const hashedPass = await bcrypt.hash(password, 10);
      user.password = hashedPass;
    }
    await user.save();
    res.json({ message: 'Дані оновлено' });
  } catch (error) {
    res.status(500).json({ message: 'Помилка оновлення даних', error });
  }
};
```

birdshop-backend/controllers/productController.js

```
const Product = require('../models/Product');

exports.getAllProducts = async (req, res) => {
  try {
    const { search, category } = req.query;
    let filter = {};
    if (search) {
      const regex = new RegExp(search, 'i');
      filter.$or = [{ name: { $regex: regex } }, { description: { $regex:
regex } }];
    }
    if (category && category !== 'Усі категорії') {
      filter.category = category;
    }
    const products = await Product.find(filter);
    res.json(products);
```

```
  } catch (error) {
    res.status(500).json({ message: 'Помилка отримання товарів', error });
  }
};

exports.createProduct = async (req, res) => {
  try {
    if (req.user.role !== 'admin') {
      return res.status(403).json({ message: 'Недостатньо прав доступу' });
    }
    const { name, description, price, category } = req.body;
    const numericPrice = Number(price);
    if (numericPrice < 0) {
      return res.status(400).json({ message: 'Ціна не може бути від’ємною'
});
    }
    let imagePath = '';
    if (req.file) {
      imagePath = req.file.path;
    }
    const newProduct = new Product({ name, description, price: numericPrice,
category, imagePath });
    await newProduct.save();
    res.status(201).json({ message: 'Товар створено', product: newProduct });
  } catch (error) {
    console.error('Помилка створення товару:', error);
    res.status(500).json({ message: 'Не вдалося зберегти товар', error });
  }
};

exports.getProductById = async (req, res) => {
  try {
    const product = await Product.findById(req.params.id);
    if (!product) return res.status(404).json({ message: 'Товар не знайдено'
});
    res.json(product);
  } catch (error) {
    res.status(500).json({ message: 'Помилка сервера', error });
  }
};

exports.updateProduct = async (req, res) => {
  try {
    if (req.user.role !== 'admin') {
      return res.status(403).json({ message: 'Недостатньо прав доступу' });
    }
    const { name, description, price, category } = req.body;
    const numericPrice = Number(price);
    if (numericPrice < 0) {
      return res.status(400).json({ message: 'Ціна не може бути від’ємною'
});
    }
    let updateFields = { name, description, price: numericPrice, category };
    if (req.file) {
      updateFields.imagePath = req.file.path;
    }
    const updatedProduct = await Product.findByIdAndUpdate(req.params.id,
updateFields, { new: true });
    if (!updatedProduct) {
      return res.status(404).json({ message: 'Товар не знайдено' });
    }
    res.json({ message: 'Товар оновлено', product: updatedProduct });
  } catch (error) {
    console.error('Помилка оновлення товару:', error);
```

```
    res.status(500).json({ message: 'Не вдалося оновити товар', error });
  }
};

exports.deleteProduct = async (req, res) => {
  try {
    if (req.user.role !== 'admin') {
      return res.status(403).json({ message: 'Недостатньо прав доступу' });
    }
    const deletedProduct = await Product.findByIdAndDelete(req.params.id);
    if (!deletedProduct) {
      return res.status(404).json({ message: 'Товар не знайдено' });
    }
    res.json({ message: 'Товар видалено' });
  } catch (error) {
    console.error('Помилка видалення товару:', error);
    res.status(500).json({ message: 'Не вдалося видалити товар', error });
  }
};
```

birdshop-backend/controllers/cartController.js

```
const Cart = require('../models/Cart');

exports.getCart = async (req, res) => {
  try {
    const cart = await Cart.findOne({ user: req.user.userId
}).populate('items.product');
    if (!cart) return res.json({ items: [] });
    res.json(cart);
  } catch (error) {
    res.status(500).json({ message: 'Помилка отримання кошика', error });
  }
};

exports.addToCart = async (req, res) => {
  try {
    const { productId, quantity } = req.body;
    if (quantity <= 0) return res.status(400).json({ message: 'Кількість має
бути додатньою' });
    let cart = await Cart.findOne({ user: req.user.userId });
    if (!cart) {
      cart = new Cart({ user: req.user.userId, items: [] });
    }
    const index = cart.items.findIndex(item => item.product.toString() ===
productId);
    if (index >= 0) {
      cart.items[index].quantity += quantity;
    } else {
      cart.items.push({ product: productId, quantity });
    }
    await cart.save();
    res.json({ message: 'Товар додано до кошика', cart });
  } catch (error) {
    res.status(500).json({ message: 'Помилка додавання до кошика', error });
  }
};

exports.removeFromCart = async (req, res) => {
  try {
    const productId = req.params.id;
    const cart = await Cart.findOne({ user: req.user.userId });
    if (!cart) return res.status(404).json({ message: 'Кошик не знайдено' });
```

```
    cart.items = cart.items.filter(item => item.product.toString() !==
productId);
    await cart.save();
    res.json({ message: 'Товар видалено з кошика', cart });
  } catch (error) {
    res.status(500).json({ message: 'Помилка видалення товару з кошика',
error });
  }
};
```

birdshop-frontend/src/components/NavBar.jsx

```
import React, { useEffect, useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import './NavBar.css';

function NavBar() {
  const navigate = useNavigate();
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  const [role, setRole] = useState('');

  useEffect(() => {
    const token = localStorage.getItem('token');
    const r = localStorage.getItem('role') || '';
    setIsLoggedIn(!!token);
    setRole(r);
  }, []);

  const handleLogout = () => {
    localStorage.removeItem('token');
    localStorage.removeItem('role');
    navigate('/', { replace: true });
    window.location.reload();
  };

  return (
    <nav className="nav-bar fade-in-animation">
      <div className="nav-logo">Birdshop</div>
      <ul className="nav-links">
        <li><Link to="/">Головна</Link></li>
        {isLoggedIn ? (
          <>
            {role === 'admin' ? (
              <li><Link to="/admin">Адмін-панель</Link></li>
            ) : (
              <>
                <li><Link to="/account">Акаунт</Link></li>
                <li><Link to="/cart">Кошик</Link></li>
              </>
            )}
            <li><button className="logout-btn"
onClick={handleLogout}>Вийти</button></li>
          </>
        ) : (
          <>
            <li><Link to="/login">Увійти</Link></li>
            <li><Link to="/register">Реєстрація</Link></li>
          </>
        )}
      </ul>
    </nav>
  );
}
```

```
export default NavBar;
```

birdshop-frontend/src/components/NavBar.css

```
.nav-bar {
  display: flex;
  justify-content: space-between;
  align-items: center;
  background: linear-gradient(to right, #4facfe, #00f2fe);
  padding: 10px 20px;
  border-bottom: 3px solid #00b4f0;
  box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);
}

.fade-in-animation {
  animation: fadeIn 0.7s ease forwards;
}

@keyframes fadeIn {
  from { opacity: 0; }
  to { opacity: 1; }
}

.nav-logo {
  font-size: 1.9rem;
  font-family: 'Pacifico', cursive;
  color: #ffffff;
  text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.2);
}

.nav-links {
  list-style: none;
  display: flex;
  gap: 18px;
  margin: 0;
  padding: 0;
}

.nav-links li a {
  color: #ffffff;
  text-decoration: none;
  font-weight: 500;
  transition: color 0.3s ease;
}

.nav-links li a:hover {
  color: #cce6ff;
  text-shadow: 0 0 5px rgba(204, 230, 255, 0.8);
}

.logout-btn {
  background-color: transparent;
  border: 1px solid #ffffff;
  color: #ffffff;
  padding: 6px 12px;
  border-radius: 4px;
  cursor: pointer;
  transition: background-color 0.3s;
}

.logout-btn:hover {
  background-color: rgba(255, 255, 255, 0.3);
```

```
}
```

birdshop-frontend/src/components/ModalMessage.jsx

```
import React from 'react';
import './ModalMessage.css';

function ModalMessage({ show, onClose, title, message }) {
  if (!show) return null;
  return (
    <div className="modal-backdrop">
      <div className="modal-window">
        <h2>{title}</h2>
        <p>{message}</p>
        <button onClick={onClose} className="modal-close-btn">OK</button>
      </div>
    </div>
  );
}

export default ModalMessage;
```

birdshop-frontend/src/components/ModalMessage.css

```
.modal-backdrop {
  position: fixed;
  top: 0;
  left: 0;
  width: 100vw;
  height: 100vh;
  background-color: rgba(0, 0, 0, 0.4);
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 999;
}

.modal-window {
  background: #e6f7ff;
  border: 2px solid #80c1ff;
  border-radius: 8px;
  width: 320px;
  padding: 20px;
  text-align: center;
}

.modal-window h2 {
  color: #0073e6;
  margin-bottom: 10px;
}

.modal-window p {
  color: #333;
}

.modal-close-btn {
  background-color: #0073e6;
  border: none;
  padding: 8px 16px;
  border-radius: 4px;
  color: #fff;
  font-weight: 600;
```

```
  cursor: pointer;
  transition: background-color 0.3s ease;
}

.modal-close-btn:hover {
  background-color: #005bb5;
}
```

birdshop-frontend/src/pages/HomePage.jsx

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import ModalMessage from '../components/ModalMessage';
import './HomePage.css';

function HomePage() {
  const [products, setProducts] = useState([]);
  const [modal, setModal] = useState({ show: false, title: '', message: ''
});
  const [search, setSearch] = useState('');
  const [category, setCategory] = useState('Усі категорії');

  const closeModal = () => setModal({ show: false, title: '', message: '' });

  useEffect(() => {
    axios.get('http://localhost:5000/api/products')
      .then(res => setProducts(res.data))
      .catch(() => setModal({ show: true, title: 'Помилка', message: 'Не
вдалося завантажити товари' }));
  }, []);

  const categories = ['Усі категорії', ...Array.from(new
Set(products.map(prod => prod.category)))];

  const filteredProducts = products.filter(prod => {
    const matchesSearch =
prod.name.toLowerCase().includes(search.toLowerCase()) ||
prod.description.toLowerCase().includes(search.toLowerCase());
    const matchesCategory = category === 'Усі категорії' || prod.category ===
category;
    return matchesSearch && matchesCategory;
  });

  const handleAddToCart = (productId) => {
    const token = localStorage.getItem('token');
    if (!token) {
      setModal({ show: true, title: 'Помилка', message: 'Спочатку увійдіть у
систему, щоб додати товар у кошик' });
      return;
    }
    axios.post('http://localhost:5000/api/cart/add', { productId, quantity: 1
}, {
      headers: { Authorization: `Bearer ${token}` }
    })
      .then(() => setModal({ show: true, title: 'Успіх', message: 'Товар
додано до кошика' }))
      .catch(() => setModal({ show: true, title: 'Помилка', message: 'Не
вдалося додати товар у кошик' }));
  };

  return (
    <div className="home-page">
```

```
      <ModalMessage show={modal.show} onClose={closeModal}
title={modal.title} message={modal.message} />

      <div className="banner">
        <h1 className="site-title">Birdshop</h1>
        <p className="site-subtitle">Тут можна знайти все необхідне для
катання на лижах: одяг, аксесуари та спорядження</p>
      </div>

      <div className="search-bar">
        <input
          type="text"
          placeholder="Пошук..."
          value={search}
          onChange={(e) => setSearch(e.target.value)}
          className="search-input"
        />
        <select
          className="category-select"
          value={category}
          onChange={(e) => setCategory(e.target.value)}
        >
          {categories.map(cat => (
            <option key={cat} value={cat}>{cat}</option>
          ))}
        </select>
      </div>

      <div className="products-list">
        {filteredProducts.length === 0 ? (
          <p className="no-products">Немає товарів за вашим запитом</p>
        ) : (
          filteredProducts.map(product => (
            <div key={product._id} className="product-card">
              {product.imagePath ? (
                <img src={`http://localhost:5000/${product.imagePath}`}
alt={product.name} className="product-image" />
              ) : (
                <div className="no-image">Немає фото</div>
              )}
              <h3 className="product-name">{product.name}</h3>
              <p className="product-price">Ціна: ${product.price}</p>
              <p className="product-category">{product.category}</p>
              <p className="product-description">{product.description}</p>
              <button className="add-cart-btn" onClick={() =>
handleAddToCart(product._id)}>В кошик</button>
            </div>
          ))
        )}
      </div>
    </div>
  );
}

export default HomePage;
```

birdshop-frontend/src/pages/HomePage.css

```
.home-page {
  min-height: 100vh;
  background: #e6f7ff;
  font-family: 'Poppins', sans-serif;
  padding-bottom: 40px;
```

```
}

.banner {
  text-align: center;
  padding: 30px 20px;
  background: linear-gradient(to right, #4facfe, #00f2fe);
  color: #fff;
  margin-bottom: 20px;
  box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);
}

.site-title {
  font-size: 2.5rem;
  margin-bottom: 10px;
}

.site-subtitle {
  font-size: 1.2rem;
}

.search-bar {
  display: flex;
  justify-content: center;
  gap: 10px;
  margin: 0 auto 20px auto;
  max-width: 500px;
}

.search-input {
  flex: 1;
  padding: 8px;
  border: 1px solid #ddd;
  border-radius: 4px;
}

.category-select {
  padding: 8px;
  border: 1px solid #ddd;
  border-radius: 4px;
  background: #fff;
}

.products-list {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(240px, 1fr));
  gap: 20px;
  width: 90%;
  max-width: 1200px;
  margin: 0 auto;
}

.no-products {
  text-align: center;
  color: #0073e6;
  font-weight: 600;
  font-size: 1.2rem;
  margin-top: 40px;
}

.product-card {
  background: #ffffff;
  border: 1px solid #b3e5fc;
  border-radius: 8px;
  padding: 16px;
```

```
  text-align: center;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
  position: relative;
}

.product-image {
  width: 100%;
  height: auto;
  object-fit: cover;
  margin-bottom: 10px;
  border-radius: 4px;
}

.no-image {
  width: 100%;
  height: 180px;
  background: #cceeff;
  display: flex;
  align-items: center;
  justify-content: center;
  margin-bottom: 10px;
  border-radius: 4px;
}

.product-name {
  color: #0073e6;
  margin-bottom: 6px;
  font-size: 1.2rem;
}

.product-price {
  color: #0073e6;
  font-weight: 600;
  margin-bottom: 4px;
}

.product-category {
  color: #005bb5;
  font-weight: 500;
  margin-bottom: 8px;
}

.product-description {
  color: #333;
  font-size: 0.9rem;
  margin-bottom: 16px;
}
```