МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____»_____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБЗАСТОСУНОК З ПРОДАЖУ ТОВАРІВ ТА ІНТЕГРОВАНИМ ШІ-АСИСТЕНТОМ У ЧАТІ ДЛЯ ПІДТРИМКИ КОРИСТУВАЧІВ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____Станіслав ВАСИЛЬЄВ

«___»_____ 2026 р.

Керівник проф., д-р пед. наук _____Олександр МЕЩАНІНОВ

«___»_____ 2026 р.

Миколаїв – 2026

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Васильєва Станіслава Олександровича
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Вебзастосунок з продажу товарів та інтегрованим ШІ-асистентом у чаті для підтримки користувачів».

Керівник роботи: Мещанінов Олександр Павлович, професор, д-р пед. наук.

Затверджена наказом ЧНУ ім. Петра Могили від «25» грудня 2025 р. № 353.

2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: вебзастосунок електронної комерції для продажу товарів із клієнтською та адміністративною частинами; база даних товарів, категорій, характеристик і замовлень; REST API для взаємодії між компонентами системи; інтегрований ШІ-асистент для консультації користувачів та рекомендації товарів; результати тестування функціональності та продуктивності системи.

4. Перелік питань, що підлягають розробці: аналіз сучасних вебзастосунків електронної комерції та існуючих ШІ-рішень для підтримки користувачів; аналіз і вибір технологій для реалізації клієнтської та серверної частин системи; проєктування архітектури вебзастосунку та структури бази даних; розробка серверної частини у вигляді REST API для роботи з товарами, замовленнями та користувачами; розробка клієнтської частини вебзастосунку для взаємодії з каталогом товарів та оформлення замовлень; реалізація адміністративної панелі для керування товарами, категоріями, характеристиками, контентом і замовленнями; інтеграція ШІ-асистента для підтримки користувачів та рекомендації товарів; тестування розробленого програмного забезпечення та аналіз результатів його роботи.

5. Перелік графічних матеріалів: презентація, діаграма архітектури системи, схема бази даних, UML-діаграми, інтерфейси клієнтської та адміністративної частин вебзастосунку, схема інтеграції ШІ-асистента, результати тестування.

Керівник роботи _____ Олександр МЕЩАНІНОВ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Станіслав ВАСИЛЬЄВ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «24» грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Вебзастосунок з продажу товарів та інтегрованим ШІ-асистентом у чаті для підтримки користувачів

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на КР	24.12.2025	24.12.2025	Виконано
2	Аналіз предметної області та постановка задачі	26.01.2026	06.02.2026	Виконано
3	Аналіз існуючих інтернет-магазинів, ШІ-асистентів та сучасних вебтехнологій	09.02.2026	11.02.2026	Виконано
4	Проєктування структури вебзастосунку, бази даних та архітектури ШІ-асистента	12.02.2026	06.03.2026	Виконано
5	Розробка клієнтської та серверної частини вебзастосунку	09.03.2026	20.03.2026	Виконано
6	Реалізація інтеграції ШІ-асистента, API та системи рекомендацій	23.03.2026	03.04.2026	Виконано
7	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	06.04.2026	15.05.2026	Виконано
8	Попередній захист кваліфікаційної роботи на засіданні комісії кафедри	18.05.2026	22.05.2026	Виконано
9	Коригування роботи за результатами попереднього захисту	25.05.2026	05.06.2026	Виконано
10	Остаточне оформлення пояснювальної записки та підготовка матеріалів до захисту	08.06.2026	12.06.2026	Виконано
11	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.06.2026	19.06.2026	Виконано

Керівник роботи _____ Олександр МЕЩАНІНОВ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Станіслав ВАСИЛЬЄВ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2026 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 401 ЧНУ ім. Петра Могили
Васильєва Станіслава Олександровича

на тему: «**ВЕБЗАСТОСУНОК З ПРОДАЖУ ТОВАРІВ ТА ІНТЕГРОВАНИМ ШІ-АСИСТЕНТОМ»**

Актуальність даної роботи полягає у зростанні популярності електронної комерції та необхідності створення сучасних вебзастосунків із використанням технологій штучного інтелекту. Інтеграція ШІ-асистента дозволяє покращити взаємодію користувача із системою, автоматизувати підтримку клієнтів, забезпечити персоналізовані рекомендації товарів та підвищити ефективність роботи інтернет-магазину.

Об’єктом роботи є процес функціонування вебзастосунків електронної комерції.

Предметом роботи є методи, технології та програмні засоби розробки вебзастосунків електронної комерції з інтегрованими системами штучного інтелекту для підтримки користувачів.

Метою роботи є розробка вебзастосунку електронної комерції з адміністративною панеллю керування та інтегрованим ШІ-асистентом для автоматизації консультацій користувачів, покращення пошуку товарів і підвищення ефективності взаємодії з клієнтами.

В результаті виконання роботи було проведено аналіз сучасних вебтехнологій та існуючих рішень у сфері електронної комерції, спроєктовано архітектуру системи, базу даних та структуру взаємодії компонентів. Розроблено клієнтську частину інтернет-магазину, адміністративну панель керування товарами, категоріями, характеристиками та замовленнями, реалізовано REST API, інтеграцію служби доставки та ШІ-асистента для підтримки користувачів.

Дана робота складається з 3 розділів. У першому розділі проведено аналіз сучасних вебзастосунків електронної комерції, розглянуто функціональні можливості інтернет-магазинів, існуючі аналоги та особливості використання ШІ-

технологій у сфері електронної комерції. Другий розділ присвячений аналізу та вибору технологічного стеку, проєктуванню архітектури системи, структури бази даних, клієнтської та серверної частини вебзастосунку, а також проєктуванню інтеграції ШІ-асистента. У третьому описано процес реалізації вебзастосунку, розробку адміністративної панелі, серверної частини, інтеграцію ШІ-асистента, тестування та результати роботи системи. Загальний обсяг роботи – 109 сторінок. Кваліфікаційна робота містить 59 рисунки, 4 таблиці, 4 додатки і 35 джерел посилання.

Ключові слова: вебзастосунок, інтернет-магазин, електронна комерція, штучний інтелект, ШІ-асистент, React, Node.js, PostgreSQL, Prisma, чат-бот.

ABSTRACT

to the qualification work by the student of the group 401 of Petro Mohyla Black Sea National University

Stanislav Vasyliev

«WEB APPLICATION FOR PRODUCT SALES WITH AN INTEGRATED AI ASSISTANT»

The relevance of this work is determined by the growing popularity of e-commerce and the need to develop modern web applications using artificial intelligence technologies. The integration of an AI assistant improves user interaction with the system, automates customer support, provides personalized product recommendations, and increases the overall efficiency of an online store.

The object of the study is the process of functioning and development of e-commerce web applications.

The subject of the study is methods, technologies, and software tools for developing e-commerce web applications with integrated artificial intelligence systems for user support.

The purpose of the work is to develop an e-commerce web application with an administrative management panel and an integrated AI assistant to automate user consultations, improve product search, and enhance customer interaction efficiency.

As a result of the research, modern web technologies and existing e-commerce solutions were analyzed. The architecture of the system, database structure, and component interaction model were designed. The client-side application, administrative panel for managing products, categories, characteristics, and orders, REST API, delivery service integration, and AI assistant for customer support were developed and implemented.

The qualification work consists of three chapters. The first chapter analyzes modern e-commerce web applications, examines the functionality of online stores, existing analogues, and the application of artificial intelligence technologies in e-commerce. The second chapter is devoted to the selection of the technological stack,

system architecture design, database design, development of the client and server parts of the application, and the design of AI assistant integration. The third chapter describes the implementation process of the web application, development of the administrative panel and server-side functionality, AI assistant integration, testing, and analysis of the obtained results. The qualification work consists of 109 pages and includes 59 figures, 4 tables, 4 appendices, and 35 references.

Keywords: web application, online store, e-commerce, artificial intelligence, AI assistant, React, Node.js, PostgreSQL, Prisma, chatbot.

ЗМІСТ

Декларація про використанні ШІ. Під час підготовки наукової роботи (академічного тексту) було використано інструмент GPT-5.5. Відповідно до таксономії GAIDeT (2025), наведені нижче завдання були делеговані інструментам генеративного ШІ за повного людського нагляду:

- Оцінювання здійсненності та ризиків
- Пошук і систематизація літератури
- Переклад
- Оцінювання якості
- Рекомендації

Використаний інструмент генеративного ШІ: ChatGPT-5.

Повну відповідальність за фінальний рукопис несуть автори.

Інструменти генеративного ШІ не зазначаються як автори та не несуть відповідальності за кінцеві результати.

Декларацію подав: Васильєв Станіслав Олександрович

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ШІ – штучний інтелект

AI – Artificial Intelligence, штучний інтелект

API – Application Programming Interface, програмний інтерфейс застосунку

CMS – Content Management System, система керування контентом

CSS – Cascading Style Sheets, каскадні таблиці стилів

ER – Entity-Relationship, модель сутність-зв’язок

HTML – HyperText Markup Language, мова гіпертекстової розмітки

HTTP – HyperText Transfer Protocol, протокол передавання гіпертексту

HTTPS – HyperText Transfer Protocol Secure, захищений протокол передавання гіпертексту

JSON – JavaScript Object Notation, формат обміну даними

JWT – JSON Web Token, токен для безпечної передачі даних

LLM – Large Language Model, велика мовна модель

NoSQL – нереляційна база даних

OAuth – Open Authorization, відкритий протокол авторизації

ORM – Object-Relational Mapping, об’єктно-реляційне відображення

RAG – Retrieval-Augmented Generation, генерація з доповненим пошуком

REST – Representational State Transfer, архітектурний стиль взаємодії API

SEO – Search Engine Optimization, пошукова оптимізація

SPA – Single Page Application, односторінковий застосунок

SQL – Structured Query Language, мова структурованих запитів

UI – User Interface, користувацький інтерфейс

UX – User Experience, користувацький досвід

ВСТУП

Сучасне суспільство слідує технологічним трендам, які постійно змінюються та проникають у більшість сфер нашого життя. Однією з найактуальніших є інтугрування штучного інтелекту в наше повсякденне існування. Зокрема у такі сфери як медицина, фінанси, освіта, транспорт, маркетинг, кібербезпека, державне управління та електронна комерція. Це свідчить про те, що роль штучного інтелекту підвищує ефективність процесів та покращує взаємодію з користувачем та різноманітними системами. Особливо актуальною на даний момент є інтеграція штучного інтелекту у чати, як у підприємницьких цілях, так і у особистих.

Актуальність теми полягає в тому, що сучасні комерційні вебзастосунки повинні бути гнучними та маштобованими, сучасний клієнт вимагає не лише базового доступу до каталогу товарів, він шукає зручний пошук, простий та інтуїтивний інтерфейс, а також, персональні поради, саме тому успіх платформи електронної комерції залежить від того, як ця система може гнучко адаптуватися під запити. Головна проблема стандартних інтернет-магазинів, в тому, що вони мають жортску закодовану логіку, фільтрацію та пошук товарів, що ускладнить маштубуваннння та адптацію до нових можливостей без змін коду у алгоритмі вебзастосунку. Саме тому впровадження ШІ-консультанта у комерційні вебдодатки є актуальним рішенням таких сучасних проблем як: низький рівень персоналізації сервісу, відсутність адаптації під запити кожного користувача та повільні відповіді операторів підтримки, що призводить до затримок в обробці таких запитів, а в подальшому до втрати потенційних клієнтів і зниження рівня продажів.

Штучний інтелект постійно стежить за діями користувачів, підказує, який товар вибрати, і дає швидку, точну, цілодобову підтримку - безперервна обробка запитів підвищує ефективність та продуктивність вебзастосунку. В результаті потрібна менша команда операторів, і відповідно, витрати на обслуговування зменшуються, а час відповіді на запити скорочується.

Метою даної роботи є розробка вебзастосунку для продажу товарів з динамічною систмою керуванням інтернет-магазином, без втручання у код застосунку та інтегрованим ШІ-асистентом, який забепечить персоналізовану взаємодію з користовачем. Впровадження ШІ-помічника зробить пошук зручнішим, рекомендації точнішими і спілкування швидшим.

Завдання на дану роботу:

– проаналізувати сучасні підходи до розробки вебзастосунків та використання штучного інтелекту в електронній комерції;

– спроєктувати архітектуру системи вебзастосунку;

– реалізувати клієнтську та серверну частини у вигляді REST API;

– забезпечити обробку запитів, реалізацію бізнес-логіки та безпечну взаємодію з базою даних;

– розробити гнучку систему управління товарами та їх характеристиками;

– інтегрувати ШІ-асистента для обробки запитів користувачів;

– впровадити гібридну систему пошуку (повнотекстовий та векторний семантичний пошук);

– підвищити релевантність результатів пошуку та покращити користувацький досвід.

Створення такого застосунку з багатьма функціями, орієнтованого на клієнта, потребує пластичності системи. Хоча вебтехнології розвиваються, багато інтернет-магазинів залишаються складними і не гнучкими. Тому розробка повного рішення, яке поєднує сучасний інтерфейс, ШІ-підтримку, надійний сервер і самостійну адмін-панель, важливе і потрібне завдання.

Для досягнення мети буде проаналізовано сучасні підходи до розробки вебзастосунків та використання штучного інтелекту в інтернет торгівлі. Далі спроєктовано архітектуру системи та реалізовано клієнтську та серверну частину вебзастосунку у вигляді REST API для надійної обробки запитів, управління бізнес-логікою та безпечної взаємодії з базою даних. Наступним кроком є розробка гнучкої системи управління товарами і їх характеристиками та інтеграція ШІ

асистента для обробки запитів користувачів для покращення взаємодії з системою, а також, впровадження системи гібридного пошуку (поєднання повнотекстового та семантичного векторного пошуку) для підвищення релевантності результатів та покращення клієнтського досвіду.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Сутність та особливості інтернет-магазинів

Електронна комерція революціонізувала способи купівлі та продажу продуктів та послуг, долаючи географічні кордони. Електронна комерція, електронна купівля та продаж товарів і послуг через Інтернет, стала ключовою силою в сучасній комерції, вплинувши на рішення споживачів, змінивши бізнес-моделі та глобальні ринки, пропонуючи виняткову зручність та доступність. Ринок електронної комерції – це форма віртуального простору, де здійснюється комунікація для заохочення взаємовигідної співпраці між численними економічними суб'єктами, що використовують електронні технології. Поряд з очевидними перевагами ринку, такими як економія часу, важливо наголосити на перевагах роботи в секторі електронної комерції для бізнесу. Ці переваги включають світ без кордонів завдяки глобальному Інтернету та зниженню маркетингових і транзакційних витрат, що значно підвищує конкурентоспроможність і підтримує розширення малих і середніх бізнесів [1-2]. Взаємодія з клієнтами, знання, управління інформацією, винахідницький та творчий потенціал тощо є прикладами об'єктів торгівлі, кількість яких з часом збільшується.

Функціональні архітектури інтернет-магазинів, що зустрічаються в літературі, здебільшого формулювалися з точки зору наявності певних функцій, отриманих на основі дослідження вебзастосунків. Як наслідок, ці моделі ігнорують модулі серверної частини (наприклад, адміністрування, управління контентом тощо) інтернет-магазину. Підсумовуючи вищесказане можна зробити висновок, що функціональність сучасного інтернет-магазину можна умовно поділити на декілька основних модулів: каталоги, рекомендації, персоналізація, маркетинг, транзакції, реалізація замовлення.

1.2 Функціональні можливості сучасних веб-магазинів

Завдання функціональних можливостей сучасних веб-магазинів полягає у можливості охопити пріоритетні потреби користувачів. Зробити так, щоб вебзастосунок для комерції, надавав всі необхідні функціональні можливості для користувача, а також для адміністрації.

Для користувачів, важиливі такі функції онлайн-магазину, як:

– *каталог.* Клієнти можуть переміщатися по категоріях товарів в межах одного або кількох рівнів структури категорій. Часто клієнтам пропонуються альтернативи для фільтрації позицій за назвою, брендом або ціною. Сторінка опису продукту пропонує клієнтам додаткову інформацію про нього, таку як огляд. Функція пошуку дозволяє клієнтам вводити опис товару або назву, щоб знайти його без необхідності переглядати структуру. Після того, як клієнти введуть назву виробу чи опис, що їх цікавить, на сторінці з'явиться список відповідних продуктів [2];

– *рекомендації.* Рекомендації допомагають швидко знаходити популярні позиції чи товари по окремій характеристиці. Рекомендації бувать різних видів. До основних, можна віднисти такі, як популярні одиниці, схожі, супутні, нові надходження, акційні чи товари на знижці, рекомендаціїї на основі інших користувачів, контекстні рекомендації та AI-рекомендації [3-4];

– *персоналізація.* Ключовим маркетинговим ключем є персоналізація, без якого на даний момент не обійтися жодному інтернет-магазину. Онлайн платформи мають скрипт, який збирає всю необхідну інформацію, для того, щоб підібрати для кожного користувача свої рекомендації. Це збільшує прибуток, для інтернет магазинів, тому що покупець більше бачить рекомендаційні товари по його інтересам та потребам [3-4];

– *маркетинг.* Для підвищення ринкової переваги та конкурентоспроможності в Інтернеті онлайн-магазини часто розробляються та структуруються таким чином, щоб ефективно проводити кампанію з пошукової

оптимізації SEO [5] задля того щоб утримати існуючих клієнтів. Також, одним із рішень меркетингу є залучення програми лояльності у систему покупок: бонуси, кешбек, купони або інші програми стимулювання замовлень на платформі;

– *кошик.* Вважається, що кожен відвідувач повинен мати «кошик для покупок». Кожен товар, який цікавить клієнта, можна зберегти в кошику. Під час процесу покупки клієнт може підтримувати кошик, видаляючи товар, збільшуючи кількість придбаного товару або просто спорожняючи весь кошик, щоб розпочати покупку заново;

– *транзакції.* Важливим функціональним елементом вебзастосунку електронної комерції є можливість здіснювати онлайн-транзакції. Миттєва та надійна авторизація та перерахування з одного рахунку на інший за допомогою електронних засобів є важливою передумовою для комфортного та безпечного оформлення замовлень споживачами. Наявність інтегрованих платіжних систем дозволяє спростити процес оплати у онлайн-магазині [6];

– *виконання.* Товари, що продаються онлайн, зрештою мають бути передані покупцеві. Існують різноманітні способи доставки: наземний, повітряний, морський, серед яких перший спосіб є найпоширенішим. Сучасні онлайн-застосунки інтегруються із поштовими службами та логістичними сервісами, що дозволяє автоматизувати процес здійснення доставки товару до споживача [2];

– *адміністративна панель.* Така функція є невід'ємною частиною сучасного веб-магазину, яка забезпечує зручне управління структурою категорій та підкатегорій, товарами, характеристиками та фільтрами товарів, замовленнями і користувачами, знижками, акціями та промокодами. Також адміністратор впливає на клієнську частину, редагує банери, налаштовує рекомендації та інший контент. Крім, цього у адміністративній панелі, повинна бути можливість перегляду аналітики та статистики, наприклад, статистика продажів, аналізу різних показників бізнесу, а також місце для модерації користувацьких відгуків та контентом [7].

1.3 Аналіз існуючих аналогів веб-магазинів

Проанізувавши сучасний ринок електронної комерції, можна зробити висновок, що він пропонує дуже широкий вибір вебзастосунків, які використовують новітні підходи з інноваційними технологіями.

У якості прикладу існуючих онлайн-магазинів, можна виділити наступні: Comfy, Lofree, Sinsay.

Узяті приклади вебзастосунків, мають різні іноваційні технології, які є вже стандартом електронної комерції. Наприклад, інтернет-магазин «Comfy» є українським магазином техніки та електроніки. Продає від аксесуарів до побутової техніки. Має онлайн-чат підтримку для консультації користувачів. Для взаємодії з користувачами, має чат-бота у соцмережі Telegram. Також варто відзначити, що платформа аналізує поведінку споживача та може адаптуватися до інновацій у світі технологій.

Нуступним у якості прикладу було взято «Lofree», це нішевий інтернет-магазин сучасних комп’ютерних девайсів, з сучасним мінімалістичним дизайном, великою кількістю контента для споживача на головній сторінці, детальними сторінками товарів. А також, використанням чат-системи підтримки.

Останнім, прикладом, буде «Sinsay», онлайн-магазин великої кількості одягу, аксесуарів та речей для дому. Має дуже гарну та логічну структуру вебзастосунку. У системі передбачено, мобільний застосунок, католог із багаторівневою структурою, персоналізовані пропозиції та підтримку споживачів. Однією з головних переваг, це дуже якісно продумана адміністративна панель для працівників та адміністраторів. Адміністративна панель дозволяє керувати товарами, категоріями, замовленнями, акціями та контентом. Також данна платформа, використовує сучасні методи електронної комерції, рекомендацій товарів та систему знижок.

Отже, кожен з прикладів, мав свої інноваційні методи електронної комерції, активно використовують рекомендаційні підходи, чат-ботів та інші інструменти.

1.4 Використання штучного інтелекту в електронній комерції

Швидкий розвиток інновацій суттєво змінив ситуацію в комерції, особливо завдяки інтеграції ШІ в електронну комерцію. ШІ включає в себе широкий спектр інструментів та методів, що імітують людські форми розуміння, такі як навчання, мислення та самокорекція. Його використання в електронній комерції дозволяє підприємствам підвищувати операційну продуктивність, покращувати взаємодію з клієнтами та розвивати прийняття рішень на основі даних. Глобальна сфера електронної комерції дедалі більше залежить від технологій ШІ — від чат-ботів та персоналізованих пропозицій до розширеної аналітики та автоматизації, що призводить до покращення виконання та продуктивності підприємств [4].

Одна з найбільш популярних інтелектуальних технологій, це ШІ-асистент. В основі ШІ-бота, лежить сучасна модель на основі трансформатора [8], спеціалізована для діалогу. Зазвичай використовують архітектуру моделі великої мови (LLM), наприклад, модель у стилі GPT [9-10], яка попередньо навчена на різноманітних розмовних даних та налаштована на набори даних клієнтських послуг електронної комерції.

Додатково, база знань моделі може бути збагачена специфічною для предметної області інформацією шляхом її налаштування на поширених запитаннях щодо продуктів, документах політики та історичних журналах чату [11], що дозволяє чат-боту включати глибший досвід електронної комерції. А також, пам'ять боту, якщо користувач увійшов у систему, бот може отримати відповідну інформацію, таку як попередні замовлення, уподобання та попередні запити. Це дозволяє надавати глибоку допомогу користувачам [12].

Для покращення взаємодії з клієнтами ШІ-асистент повинен бути здатним виконувати дії від імені користувача, а не просто надавати інформацію. Чат-бот, має інтегруватися з серверними системами платформи електронної комерції через API, щоб отримувати необхідні дані, обробляти запити та надавати точні відповіді щодо товарів, наявності товарів, характеристик, порівнянь товарів та рекомендацій.

Така інтеграція, не просто дає відповіді користувачу, а використовує актуальну інформацію з бази даних веб-магазину [11-12].

Також, чат-бот чітко повинен ідентифікувати себе як автоматизовану систему, повідомляти про свої обмеження та проактивно пропонувати передачу інформації людині-агенту для складних або конфіденційних запитів. Крім того, під час надання інформації чат-бот може посилатися на політики компанії або джерела даних. Ці практики підвищують прозорість та сприяють більш надійному користувацькому досвіду.

1.5 Переваги та недоліки існуючих рішень

У підрозділі 1.3 були проаналізовані електроні комерційні вебдодатки, які мають велику кількість функціональних можливостей, що покращують взаємодію з користувачем, а також між працівниками та системою. Одними з основних та сучасних рішень онлайн-магазину є багаторівневі каталоги, системи фільрації, рекомендації продуктів, персоналізація контенту, онлайн купівля, оформлення доставки товарів та підтримка користувачів через чат-систему. Такі можливості, дозволяють споживачу ефективніше взаємодіяти з вебдодатком та більш зручно та швидко користуватися послугами, що надає вебзастосунок.

Незважаючи на значну кількість переваг, у кожного з наведених прикладів у підрозділі 1.3 наявні певні недоліки. Один з головних недоліків, це потреба втручання спеціаліста у внутрішній код вебдодатка для модифікації або оновлення його. Якщо брати приклад з онлайн-магазину «Lofree», то на зовнішній погляд, він має сучасний дизайн, проте з боку адміністрування, має певну проблему з додаванням нових товарів чи зміни контенту на клієнській частині, бо щоб, це зробити, потрібно додавати та оновлювати через програмний код або через інструменти, які потребують участі розробників чи спеціалістів. Такий підхід розробки називають «моноліт», він зменшує гнучкість системи та ускладнює інтегрування чи оновлення вебзастосунку [13-15]. Для сучасних вебдодадків, це має суттєвий недолік, оскільки стуктура каталогів та контент можуть змінюватися

досить часто. Пізніше у розділі 2 детальніше будуть проаналізовані види архітектур для вебзастосунків.

Ще одним недоліком сучасних вебмагазинів є обмеженний функціонал чат-ботів, зазвичай має простий алгоритм, наперед визначені сценарії та шаблонні відповіді для споживача. Такий підхід ШІ-асистентів не задовільняє потреби користувача, бо частина з них не здатна аналізувати характеристики товарів, порівнювати їх або формулювати персональні рекомендації на основі запиту клієнта [4]. Через це користувачу все ще необхідно робити основну роботу самотушки, аналізувати товар, порівнювати та знаходити кращний варіант.

Крім того, більшість чат-ботів не враховують індивідуальні потреби споживача, такі як, бюджет, бажані користувачем характеристики, категорії товарів, тим самим пошук товара стає менш зручним та затратним за часом, через що споживач не бачить практичної користі у використанні подібних чат-систем. У таблиці 1.1 наведено основні переваги та недоліки проаналізованих онлайн-магазинів.

Таблиця 1.1 – Переваги та недоліки проаналізованих вебмагазинів

Вебмагазин	**Переваги**	**Недоліки**
Comfy	Великий каталог товарів, система фільтрації та пошуку, онлайн-чат підтримки, Telegram-бот, персоналізація контенту, програма лояльності, мобільний застосунок, детальні характеристики товарів, відгуки,	Частина чат-систем працює за шаблонним сценарієм без повноцінного ШІ, складність масштабування окремих функцій

Кінець таблиці 1.1

Вебмагазин	Переваги	Недоліки
Sinsay	Багаторівневий каталог, персоналізовані рекомендації, продумана адміністративна панель, мобільний застосунок	Велика кількість функцій може перевантажувати інтерфейс, чат-боти мають обмежений функціонал
Lofree	Сучасний мінімалістичний дизайн, деталізовані сторінки товарів, якісний UX/UI, нішевий підхід	Обмежена гнучкість адміністрування, потреба втручання розробників для оновлення контенту або структури

Отже, було наведено у таблиці 1.1 основні переваги та недоліки. Аналіз показав, що онлайн-магазини використовують сучасні підходи, такі як, рекомендаційні системи, персоналізацію та чат-підтримку, але можно зазначити, що є недоліки з гнучкостью адміністрування та обмеженим функціоналом чат-ботів.

1.6 Формулювання вимог до розроблюваної системи

Розробка вебзастосунку для електронної комерції з інтегруванням інтелектуального асистенту охоплює низку важливих етапів, серед яких:

– *розробка плану проєктування:* на початковому етапі, потрібно продумати дизайн, основні функціональні можливості, обрати архітектуру проєкту та сучасні технології, спроєктувати ШІ-асистента, а також обрати середовища розробки та інструменти для створення вебзастосунку;

– *проєктування користувацького та адміністративного інтерфейсу:* інтерфейс повинен бути зрозумілим, зручним і логічним, без перенавантаження;

– *проєктування архітектури проекта:* визначити загальну структуру системи, розподіл функціональних обов’язків;

– *проєктування бази даних:* база даних має бути маштабованою й надійною;

– *проєктування бізнес функцій:* інтеграція платіжних сервісів, а також сервісів для доставки товарів;

– *проєктування для забезпечення безпеки:* вебзастосунок повинен бути захищен від загроз, таких як некоректний користувацький ввід даних, несанкціонований доступ до інформації, витік конфіденційних даних та інші види атак. Для цього необхідно реалізувати механізми автентифікації, авторизації, валідації даних та захисту серверної частини системи;

– *контроль якості коду та тестування:* щоб уникнути помилок у системі, потрібно дотримуватися єдиного стиля написання коду, для того, щоб маштубувати його. Розробити стратегію тестування для уникнення помилок;

– *опублікування та підтримка:* розгорнути вебзастосунок на сервері та забепечувати регулярне оновлення системи, це є обов’язковим, як на початку, так і під час подальшої експлуатації вебзастосунку.

Ці етапи є стандартними для створення ефективного та надійного вебзастосунку для електронної комерції.

Висновки до розділу 1

У процесі аналізу предметної області та постанови задачі для створення вебзастосунку електронної комерції, було виявлено що дана сфера активно розвивається, еволюціонує та розширює функціональність, а особливо, забезпечує якісну взаємодію між користувачем та системою.

Аналіз наявних рішень показав, що на ринку присутня значна кількість вебзастосунків, орієнтованих на продаж та цілодобову підтримку споживачів. Проте їх функціональність відрізняється один від одного, але значна частина усе ще має спільну проблему з гнучкістю адміністрування, обмеженим функціоналом чат-ботів та складністю у швидкому оновленні контенту без втручання розробників.

Ключові етапи розробки такого вебзастосунку охоплюють проєктування архітектури системи, інтерфейсів, бази даних, бізнес-логіки, забезпечення належного рівня безпеки, а також впровадження системи та її підтримки.

Усі зазначені аспекти повинні бути враховані під час формування технічного завдання, щоб гарантувати успішну реалізацію вебзастосунку та його ефективне функціонування.

2 ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ ТА ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Огляд сучасних веб-технологій для розробки

Сучасна веб-розробка – це складна система, яка інтегрує дві ключові сфери: front-end та back-end розробку [16]. Кожна з цих областей має свої теоретичні основи, технологічні підходи та методи рішення, які разом визначають якість, продуктивність та масштабованість вебзастосунку.

Front-end розробка охоплює всі аспекти створення користувацького інтерфейсу, від семантичної розмітки веб-сторінок до складних динамічних взаємодій з користувачем. Основні технології, що лежать в основі front-end розробки, включають: HTML, CSS, JavaScript [16].

Back-end розробка відповідає за логіку на стороні сервера, обробку даних, управління базами даних та виконання бізнес-процесів [2]. Розробка на стороні серевера включає такі ключові компоненти як мови програмування та фреймворки, системи керування базами данних, використання API з іншими інтеграційними рішеннями.

Окрім стандартого підходу з використання front-end та back-end розробки, сучасні вебзастосунки будуються з використанням нових архітектурних та хмарних технологій. На даний момент є такі види архітектур: Monolithic, Service Oriented та Microservice [13-15]. На риуснку 2.1 наведено порівняння основних підходів до побудови архітектури.

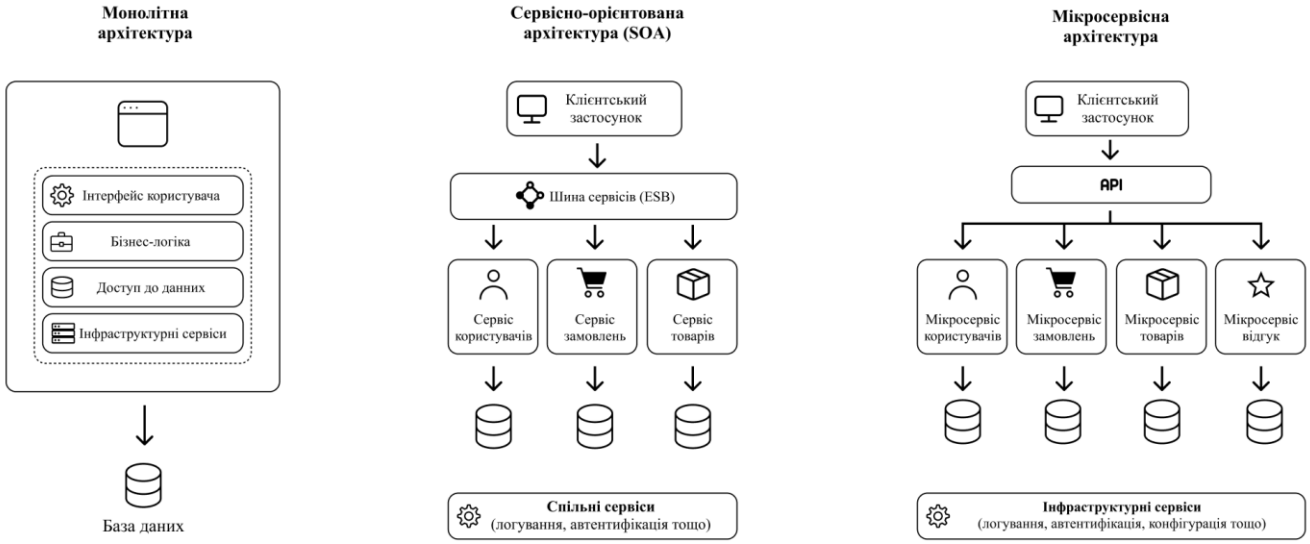

Рисунок 2.1 – Основні типи архітектур вебзастосунків

Проаналізувавши рисунок 2.1, можно побачити основні відмінності між цими трьома видами. Основні відмінності між монолітною, сервісно-орієнтованою та мікросервісною архітектурами полягають в тому, що монолітна архітектура, це єдиний програмний застосунок, у якому всі компоненти програми, як інтерфейс, бізнес-логіка, база даних, об’єднані в єдиний код та використовуються як одне ціле. Такий підхід є простіший у реалізації, проте стає складним у маштубуванні, інтеграції інших модулей та підримці.

Сервісно-орієнтована архітектура вже не має об’єднання усіх компонентів, вона передбачає розділення системи на окремі сервіси, які взаємодіють між собою. У порівнянні з монолітною архітектурою, такий підхід забезпучує кращу пітримку застосунку та інтеграцію компонентів, але часто залежить від сильної інфраструктури або бази даних.

Мікросервісна архітектура є більш сучасним підходом до побудови, вона має високу гнучкість і надійність. У даній архітектурі система розділяється на незалежні сервіси, кожен із яких відповідає за окремий функціонал, це дає легке маштубування компонентів, спрощує оновлення системи та підвищує підтримку вебзастосунку. Однак основними труднощами є складність реалізації та у супроводі такої архітектури, вона вимагає досвіду з мікросервісами [13-15].

Проаналізовані характеристики наведених архітектур для побудови вебзастосунків наведено в таблиці 2.1.

Таблиця 1.2 – Порівняльна характеристика наведених архітектур для побудови вебзастосунку

Характеристика	**Monolithic Architecture**	**Service Oriented Architecture**	**Microservice Architecture**
Структура системи	Єдиний програмний застосунок	Система поділена на сервіси	Система складається з незалежних мікросервісів
База даних	Одна спільна база даних	Часто використовується спільна база даних	Кожен сервіс може мати власну базу даних
Масштабованість	Обмежена	Середня	Висока
Складність розробки	Низька на початкових етапах	Середня	Висока
Підтримка та оновлення	Складні при великому проєкті	Простіші за моноліт	Гнучкі та незалежні
Відмовостійкість	Низька	Середня	Висока
Швидкість розробки	Висока для невеликих систем	Середня	Нижча через складність архітектури
Гнучкість системи	Низька	Середня	Висока

Кінець таблиці 1.2

Характеристика	Monolithic Architecture	Service Oriented Architecture	Microservice Architecture
Використання у сучасних вебсистемах	Невеликі проєкти	Середні корпоративні системи	Великі масштабовані платформи

2.2 Аналіз мов програмування та фреймворків

Сучасні вебзастосунки використовують фреймворки. Напопулярнішими є рішення, що базуються на мовах програмування JavaScript або TypeScript для front-end, а також мови C#, Python, PHP, TypeScript та Java для back-end.

Більшість сучасних вебзастосунків використовують JavaScript у тій чи іншій формі для того щоб зробити сторінки більш інтерактивними та для обробки функціональності. Традиційні веб-сторінки – це багатосторінкові додатки, де новий HTML-документ завантажується щоразу, коли користувач змінює сторінку або її вміст. Це відносно повільний варіант порівняно з сучаснішою версією моделі розробки односторінкових додатків SPA, де з сервера отримуються та оновлюються лише ті частини, що змінилися. Використання моделі SPA знижує швидкість навантаження додатків та покращує взаємодію з користувачем. Односторінкові додатки набирають популярності, як і фреймворки, в які вони вбудовані.

Фреймворки диктують робочий процес розробки додатків, скорочують час на створення та знижують кількість можливих помилок. Кожен фреймворк має свої переваги та недоліки, тому вибір правильного для програмного проекту може бути стратегічним рішенням як для компанії, так і для молодого розробника, який хоче додати нову навичку до свого портфоліо. Angular, React та Vue є одними з найпопулярніших фреймворків JavaScript.

React - це бібліотека JavaScript [17-18] для створення користувацьких інтерфейсів. React був створений у 2011 році Джорданом Волком у Facebook. Він став відкритим у 2013 році та з того часу отримав доповнення. React розроблений для покращення розробки інтерактивних інтерфейсів, полегшуючи оновлення представлення, коли дані змінюються. Це робиться шляхом поділу представлення на менші компоненти, які можна складати для створення складних інтерфейсів. Компоненти створюються на JavaScript, а не на шаблонах, що забезпечує легкий потік даних.

Vue.js - це легка бібліотека JavaScript [1], створена Еваном Ю у 2014 році він поділився своєю роботою з іншими на GitHub. Коли Еван розпочав працювати з Angular, він вважав, що Vue найбільш схожим на React, оскільки його основна ідея полягає в зв'язуванні даних та компонентах, як у React. Порівняно з React, Vue більше зосереджується на користувацькому досвіді, що полегшує його опанування, якщо користувач знає основи: HTML, JavaScript та CSS.

Angular - це front-end фреймворк [19-21], який підтримується Google та базується на TypeScript. Angular призначений для створення масштабованих вебзастосунків та підтримки розробки для веб, мобільних і desktop платформ. Однією з головних переваг Angular є велика кількість вбудованих інструментів для роботи з маршрутизацією, сервісами, формами та управлінням станом застосунку.

Кожен фреймворк побудований по-різному та працюватиме по-різному. Кожен із них має переваги та недоліки, що дозволить прийняти обґрунтоване рішення під час вибору між ними. Для якісного аналізу наведених фреймворків на рисунку 2.2 представлене порівняння за таким критеріям як популярність, що буде оцінена платформою NPM trends [22], яка пов'язена з розробкою програмного забезпечення.

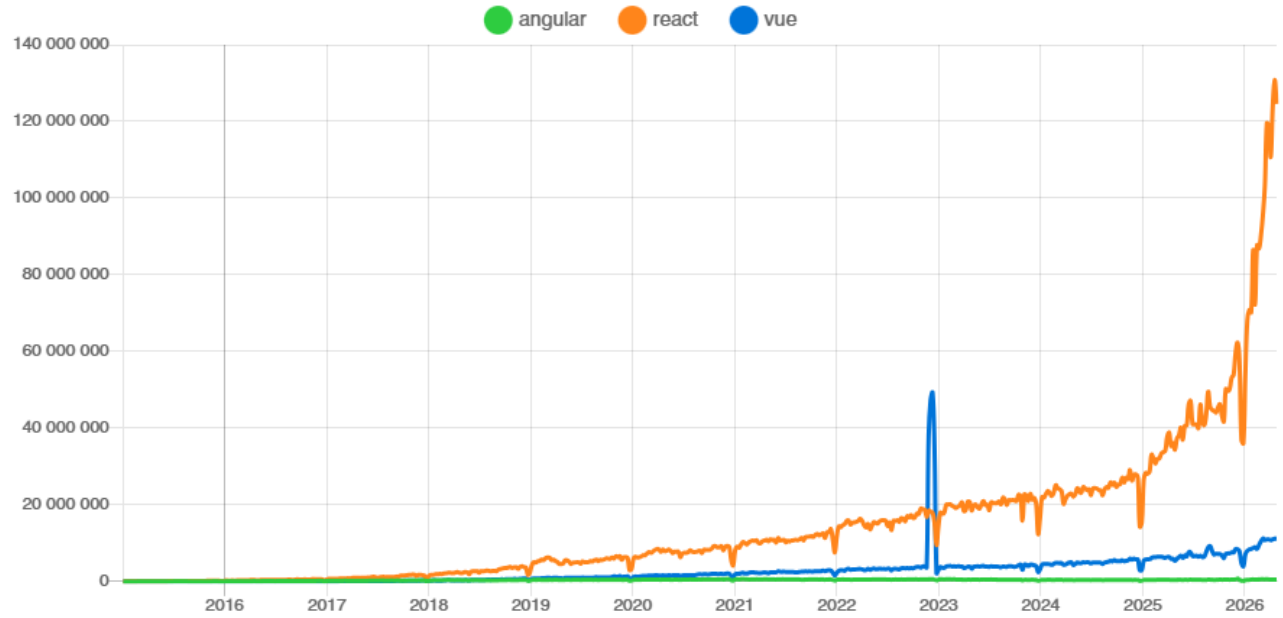

Рисунок 2.2 – Завантаження бібліотек Angular, React та Vue через NPM за весь час [22]

Отже, найпопулярнішим фреймворком для front-end є React, який суттєво перевищує Vue та Angular. Це пояснює велику екосистему, компонентним підходом, високою продуктивністю та широким використанням React у сучасних вебзастосунках.

Node.js, Django та ASP.NET – це три найпопулярніші back-end технології, що використовуються для веб-додатків, кожна з яких відповідає різним філософіям розробки, мовам програмування та варіантам використання [23-26].

Node.js - це середовище виконання JavaScript з відкритим кодом, кросплатформне, побудоване на двигуні V8 Chrome. Уніфікований стек дозволяє розробникам використовувати JavaScript як для фронтенду, так і для бекенду, спрощуючи життєвий цикл розробки. Іншою перевагою є висока продуктивність, завдяки використанню неблокувальної, керованої подіями моделі вводу/виводу, що робить його високоефективним для програм реального часу, таких як чат, потокове передавання та інструменти для спільної роботи. Також підтримується масивною екосистемою npm, що забезпечує доступ до понад мільйона пакетів повторного використання [26-27].

Django - це високорівневий вебфреймворк Python, який сприяє швидкій розробці та чистому, прагматичному дизайну. Однією з головних переваг Django є наявність великої кількості вбудованих інструментів, зокрема адміністративної панелі, системи автентифікації та ORM, що зменшує потребу у використанні сторонніх бібліотек. Також Django орієнтований на безпеку та допомагає розробникам уникати поширених вразливостей, таких як SQL-ін’єкції та міжсайтовий скриптинг. Додатковою перевагою є хороша інтеграція з бібліотеками Python для машинного навчання, аналізу даних та ШІ-рішень [25].

ASP.NET Core - це платформа з відкритим кодом та кросплатформний фреймворк, розроблений Microsoft для створення сучасних хмарних додатків. Даний фреймворк відомий високою продуктивністю та масштабованістю, що дозволяє ефективно використовувати його у системах з великим навантаженням та високим трафіком. ASP.NET Core використовує мову програмування C#, яка має строгу типізацію та допомагає виявляти помилки ще на етапі компіляції програми. Додатковою перевагою є інтеграція з Visual Studio та Azure, завдяки чому ASP.NET Core часто використовується для створення корпоративних та масштабованих вебзастосунків [24].

Таблиця 2.2 – Порівняння Node.js, Django та ASP.NET Core

Характеристика	**Node.js**	**Django**	**ASP.NET Core**
Мова програмування	JavaScript / TypeScript	Python	C# / F#
Тип технології	Середовище виконання	Високорівневий фреймворк	Кросплатформний фреймворк
Основна концепція	Легкість та мінімалізм	“Batteries included”	Корпоративна модульна архітектура

Кінець таблиці 2.2

Характеристика	**Node.js**	**Django**	**ASP.NET Core**
Основна перевага	Realtime та масштабованість	Швидка розробка та ШІ	Продуктивність та типізація
Складність освоєння	Невисока для JavaScript-розробників	Невисока для Python-розробників	Середня

Отже, проаналізувавши таблицю 2.2, можно зробити висновок, що кожна з технологій має свої переваги та недоліки, а також різне використання для різних вебзастосунків. Django добре підходить для проєктів, пов’язаний з ШІ та аналізом даних. ASP.NET Core частіше використовується для великих бізнез-логік у великих корпоративних компаніях, де є важливими продуктивність, строгий контроль типів даних та безпека. Для розробки комерційного вебзастосунку найбільш доцільним, буде використання Node.js у поєднанні з Express. Дане рішення підходить для створення SPA застосунків, REST API та Realtime функціонал. Одна з переваг є використання TypeScript, як для front-end, так і для back-end. Це дуже спрощує підтримку проєкту та робить процес розробки більш зручним, завдяки цьому не потрібно знати декілька різних мов та синтаксисів програмувань, що зменшує навантаження на розробника. Також Node.js має велику кількість готових бібліотек для інтеграції у інтернет комерцію, що дозволяє пришвидшити реалізацію необхідного функціоналу.

2.3 Вибір технологічного стеку та його обґрунтування

Враховуючи викладене у підрозділі 2.2, було обрано використання React для front-end, а також для back-end фреймворк Node.js для даної роботи. У межах повного стеку технологій цього недостатньо, обрані фреймворки лише основа

проєкту. Для успішної та швидкої розробки вебзастосунку необхідне використання додаткових інтструментів та бібліотек.

Для front-end частини, було обрано такі технології, як Vite, Tailwind CSS, Framer Motion та HeroUI. Використання сучасного інструменту Vite пришвидшує запуск та збірку вебзастосунків на React, Vue та JS, який використовується у нативному модулі ES та Rolldown для майже миттєвого запуску сервера розробки. Його переваги полягають у миттєвому запуску сервера, відображенні змін у браузері та здатності працювати для більшості фреймворків. Vite є чудовим вибором для сучасних SPA та багатосторінкових сайтів.

Наступним обраним сучасним додадковим інструментом є CSS-фреймворк TШIlwind, який дозволяє помітно прискорити створення унікального інтерфейсу безпосередньо у HTML коді, використовуючи готові бібліотеки та службові класи. Він не надає готові компоненти як Boostrap, а пропонує гнучкі інструменти для швидкої верстки. Наступні популярні бібліотекі як HeroUI та Framer Motion - відкриті бібліотеки для React-додатків. HeroUI потрібен для готових шаблонних рішень інтерфейс компонентів як Button, Select, ComboBox, Textarea, Input, Modal та інші UI-елементи. Це дозволяє швидко пришвидшити процес розробки інтерфейсів, забезбечити єдиний стиль та зменшити кількість власного коду. Framer Motion також сучасна бібліотека анімацій, яка створює плавні анімації та є простою у використанні, а також сумісна з іншими технологіями та фреймворками.

Також для back-end було обрано додаткові інструменти як Express, це найпопулярніший, швидкий та гнучкий вебфреймворк для платформи Node.js. Він спроєктований як стандартний інструмент для створення вебзастосунків, REST API та серворної частини вебдодатків, забезпечує зручну роботу з маршрутами та HTTP-запитами. Для Node.js та TypeScript дуже підходить сучасна ORM, яка дозволяє процювати з базами данних як PostgreSQL, MySQL, SQLite та інші. Детальніше про Prisma буде розглянуто у підрозділі 2.6.

Для реалізації функціональності авторизації, хмарного сховища для

зберігання фотографій та автоматизації роботи з доставкою було обрано такі технології: Google Oauth, JWT, Cloudinary, Nova Poshta API.

Google Oauth - це стандарт авторізації на основі протоколу OAuth 2.0, що дозволяє стороннім програмам отримувати безпечний доступ до даних користувача Google або входити на сайти за допомогою облікового запису Google. Для передачі інформації авторізації між сторонами у вигляді JSON було обрано технологію JWT. Для зберігання фотографій найкращим безкоштовним рішенням є Cloudinary, він легко інтегрується у код та має гнучкі інструменти для використання. Nova Poshta API має програмний інтерфейс служби доставки Nova Poshta, який дозволить інтегрувати логістичні можливості у даний вебзастосунок електронної комерції за допомогою власної API системи. Ця система може автоматично отримувати дані про відділення, розрахувати вартість доставки, формувати накладні та відстежувати статус замовлення.

Отже, обраний стек технологій дозволяє створити маштабний та гнучкий електроний комерційний застосунок. Використання React, Node.js та додаткових інструментів забезпечить високу швидкість, зручність та можливість інтеграції ШІ функціоналу у вебзастосунок.

2.4 Проєктування структури вебзастосунку

Загальна структура вебзастосунку має складатися з навігаційних елементів, це є правилом основної структури вебзастосунку. Саме навігаційна структура формує основний інтерфейс онлайн-магазину, впливаючи на шквідкість орієнтації користувача на сторінці. Споживач швидко може зорієнтуватися та легко знаходити каталоги, фільтри, пошук, товари та інший функціонал онлайн-магазину. До основних навігаційних елементів належать: header, menu, breadcrumbs, search, filter та footer. Приклад структури та розташування навігаційних компонентів наведено на рисунку 2.5.

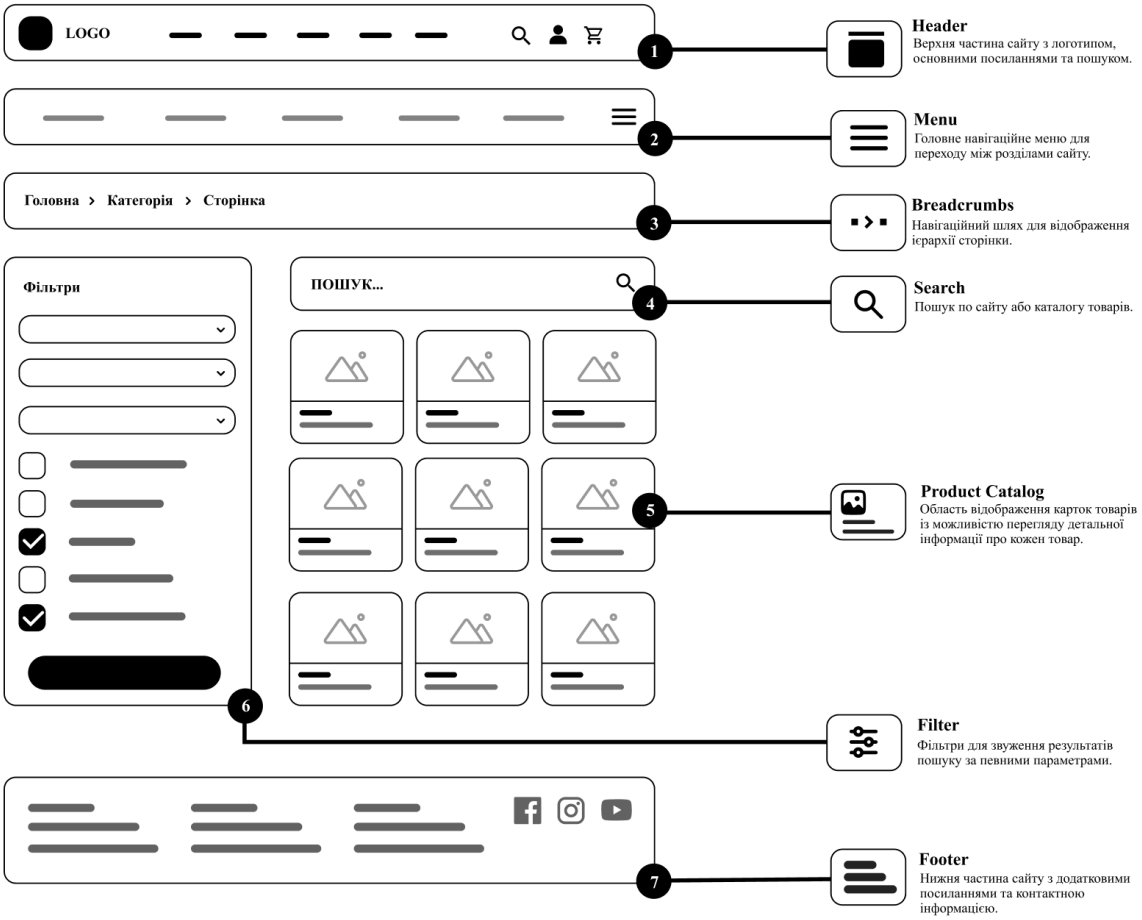

Рисунок 2.5 – Основні навігаційні елементи інтерфейсу вебзастосунку електронної комерції

Проаналізувавши рисунок 2.5, можна побачити, що основні навігаційні елементи мають структуру, яка використовується у більшості сучасних вебзастосунків електронної комерції. Подібний підхід став звичний для користувачів, вони автоматично розуміють структуру, бо стикались з аналогічними вебзастосунками. Чим більш структура сайту інтуїтивно зрозуміла для користувача, тим краще він буде використовувати функціонал вебзастосунку.

Окрім структури, онлайн-магазин повинен мати сучасний та зрозумілий дизайн, який не перевантажує споживача зайвими елементами. Дизайн вебзастосунку впливає не тільки на візуальне сприйняття - якщо невдало побудувати інтерфейс, то можна ускладнити навігацію та знизити зацікавленість користувача до використання вебзастосунку. При розробці вебзастосунку

електронної комерції необхідно враховувати UX/UI дизайн, адаптивність та доступність.

Також важливим є мінімізація кількості дій, необхідних до виконання покупки, чим швидше користувач може знайти товар, додати до кошика та оформити замовлення, тим вища ефективність роботи вебзастосунку для бізнесу, оскільки це впливає на конверсію, кількість успішних завершених покупок та прибуток онлайн-магазину.

2.5 Розробка архітектури системи

У підрозділі 2.1, ми мали змогу проаналізувати види архітектури системи. Зокрема монолітну, сервісно-орієнтовану та мікросервісну архітектуру. За результатами аналізу можно зробити висновок, що для розробки системи найбільш доцільним буде використання модульного клієнт-сервеного підходу з можливістю подальшого переходу до мікросервісної архітектури.

Такий варіант дозволяє розділити основні частини системи, швидко маштубувати його у майбутньому та оновлювати вебдодаток. На рисунку 2.7 можно переглянути архітектурне рішення вебзастосунку на основі модульного клієнт-серверного підходу з можливістю переходу до мікросервісної архітектури.

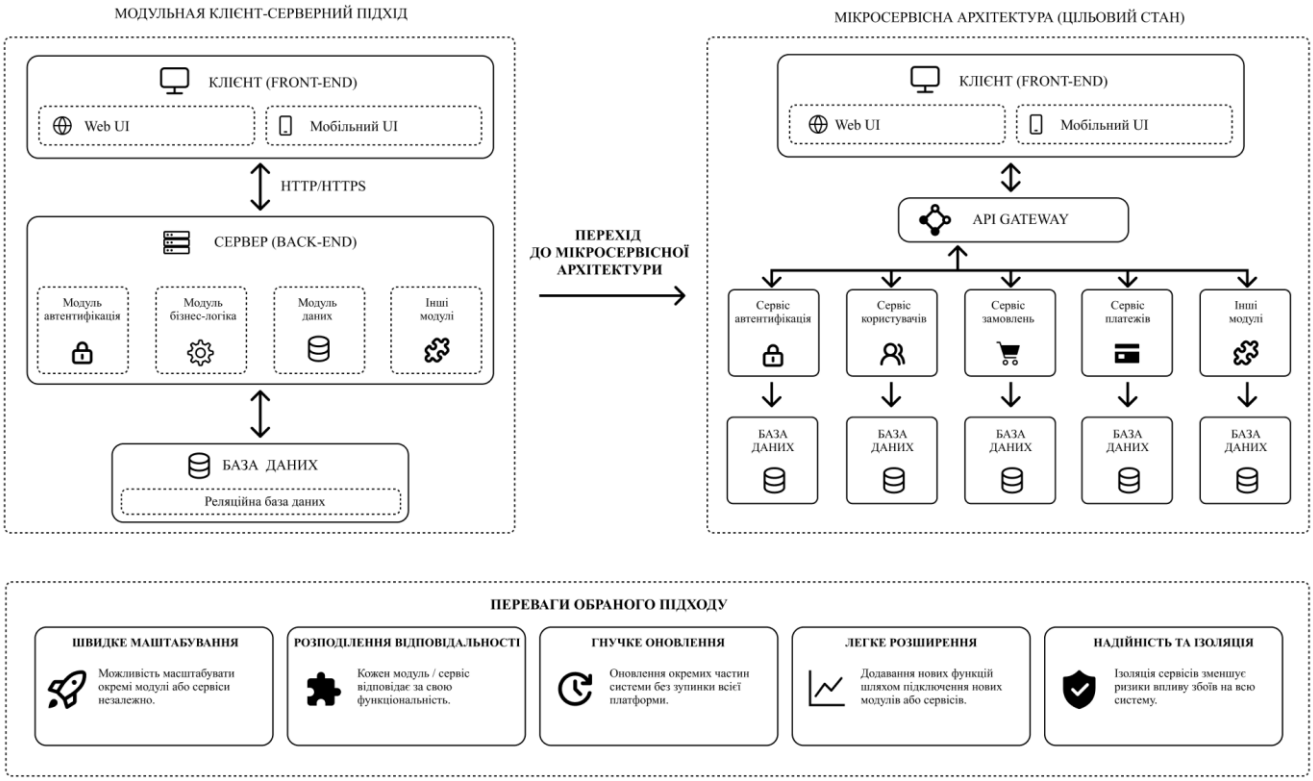

Рисунок 2.7 – Архітектурне рішення вебзастосунку на основі модульного клієнт-серверного підходу з можливістю переходу до мікросервісної архітектури

На рисунку 2.7 ми можемо побачити як клієнт-серверний підхід та мікросервісна архітектура використовуються у подальшому етапі розвитку системи. У лівій частині схеми показано базову архітектуру, де клієнська частина напряму взаємодіє із серверною частиною через HTTP/HTTPS запити. На сервері є модулі, які відповідають за авторизацію, бізнез-логіку, обробку даних та інший функціонал, також модулі використовують спільну базу даних. У правій частині ми можемо розглянути повноцінну мікросервісну архітектуру, у цьому підході система розділяється на незалежні один від одних сервіси та кожен відповідає за окремий функціонал. Взаємодія між клієнтом та сервісами відбувається через API Gateway, який використовується как прошарок. Призначення API Gateway в тому, щоб надати єдину точку входу зі всіх клієнських запитів, яка маршрутизує її до відповідних внутрішніх мікросервісів. Кожен сервіс може використовувати свою базу даних та маштабуватися незалежно від інших частин системи.

2.6 Проєктування бази даних

Для зберігання даних у вебзастосунку використовуються бази даних, вони бувають двух видів: реляційні (SQL) та нереляційні (NoSQL):

реляційні використовують жорстку табличну структуру та мову SQL для складних запитів, забезпечуючи високу цілісність даних;

нереляційні є більш гнучкими, працюють з документами, парами ключ-значення або графами, легко маштабуються і підходять для великих обсягів неструктурованих даних.

Отже, основні відмінності SQL та NoSQL в тому, що різна структура даних, маштабування, мова запитів та їх цілісність. Як і будь-яке прикладне програмне забезпечення, реалізації NoSQL та SQL баз даних постійно змінюються та вдосконалюються. Через це продуктивність, масштабованість та можливості різних СКБД можуть змінюватися з часом. Саме тому вибір бази даних є важливим етапом під час проєктування вебзастосунку. Також, сучасна багаторівнева архітектура програмного забезпечення дозволяє ізолювати серверну частину від конкретної реалізації бази даних, що у майбутньому спрощує можливість переходу на іншу СКБД при необхідності. Для розробки вебзастосунку електронної комерції було обрано PostgreSQL [29], оскільки дана реляційна база даних добре підходить для роботи зі складними зв’язками між товарами, категоріями, користувачами, замовленнями та характеристиками товарів. Додатковою перевагою PostgreSQL є підтримка розширення pgvector [29, 32], що дозволяє реалізувати embeddings та vector search для ШІ-асистента [32-33].

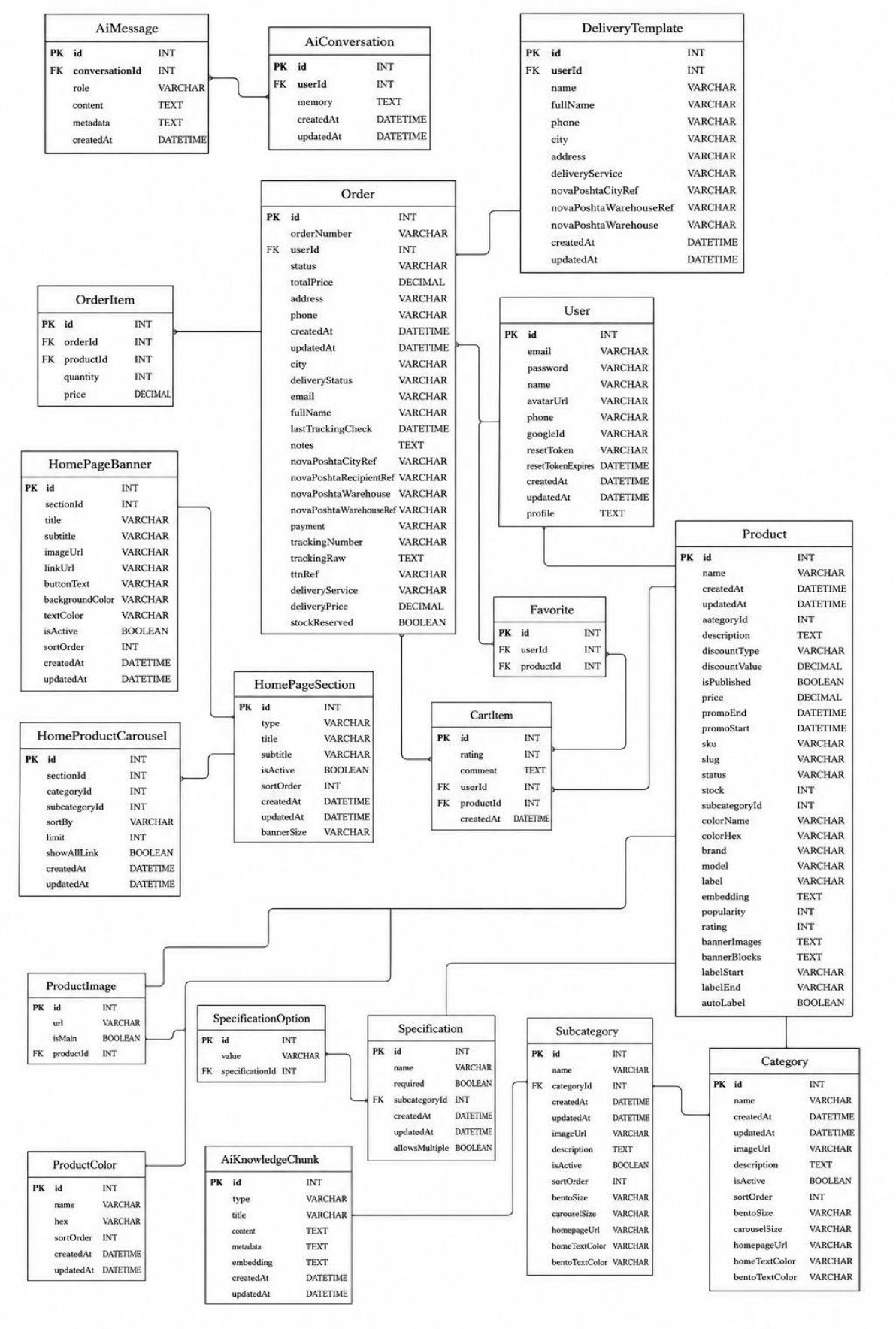

Рисунок 2.4 – ER-діаграма бази даних вебзастосунку електронної комерції

На рисунку 2.6 представлена ER-діаграма бази даних вебзастосунку електронної комерції. Дана структура відображає основні сутності системи, їх атрибути та зв’язки між таблицями. Основними сутностями є User, Product, Category, Subcategory, Order, OrderItem, Review, CartItem, Favorite та ProductSpecification.

Головною сутністю є Product, яка містить основну інформацію про товари, зокрема назву, опис, ціну, характеристики, рейтинг, популярність, embedding для ШІ-пошуку та інші. Сутності Category та Subcategory використовуються для реалізації структури каталогу товарів. Для роботи з характеристиками товарів використовується окрема система Specification, SpecificationOption та ProductSpecification, що дозволяє створювати гнучкі характеристики та фільтри товарів. Сутність User відповідає за зберігання інформації про користувачів, авторизацію та взаємодію з системою. Для реалізації функціоналу замовлень використовуються таблиці Order та OrderItem. Також у системі мають бути реалізовані таблиці CartItem та Favorite для роботи з кошиком та обраними одиницями. Таблиця Review використовується для збереження відгуків та рейтингів товарів. Окремо у структурі бази даних повині бути присутні таблиці Payment та Delivery для роботи з оплатою та доставкою замовлень. Для ШІ-асистента має бути передбачені таблиці ChatSession та ChatMessage, які дозволяють зберігати історію взаємодії користувача з чат-системою.

База даних має використатися з принципу реляційного підходу та містити зв’язки типу one-to-one, one-to-many та many-to-many. Така структура дозволяє забезпечити цілісність даних, гнучкість системи [6] та підтримку складної логіки вебзастосунку електронної комерції.

2.7 Проєктування взаємодії з ШІ-асистентом

Одна з оновних особливостей у створенні вебзастосунку є інтеграція ШІ-асистента у чаті для допомги користувачам у виборі товарів. ШІ-асистент дозволяє атоматизувати взаємодію з користувачем, пришвидшити пошук, надати необхідну

інформацію на онові запиту споживача .

Архітектура взаємодіїї ШІ у чаті складається з front-end, back-end, AI service, OpenAI API, RAG, embedding та vector search [11, 32-34]. Переглянути схему взаємодії можна на рисунку 2.3.

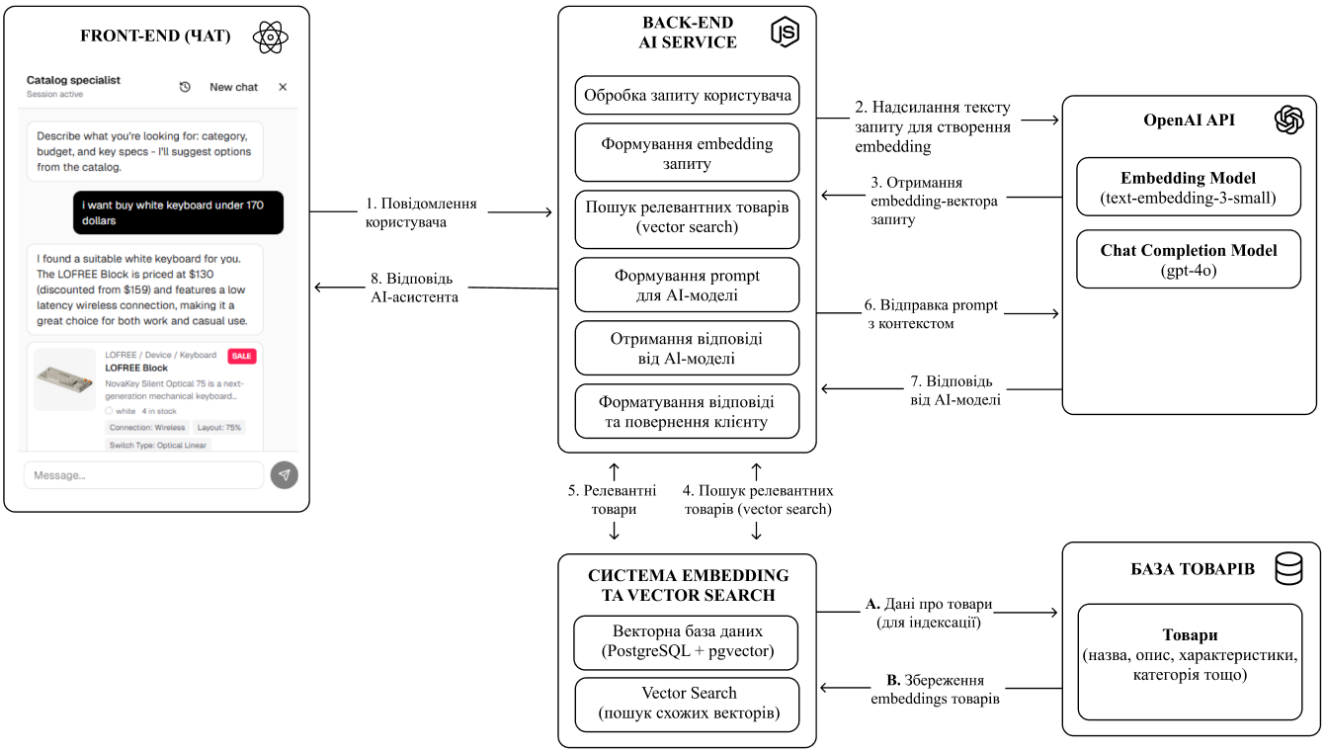

Рисунок 2.3 – Архітектура взаємодії компонентів ШІ-асистента у вебзастосунку

На схемі можна побачити, що front-end частина являє собою модальне вікно чату, яке дає змогу користувачу надсилати повідомлення та отримувати відповіді від ШІ-асистента у режимі реального часу. Чат інтегрований безпосередньо у користувацький інтерфейс вебзастосунку електронної комерції та взаємодіє із серверною service. Серверна частина обробляє повідомлення, аналізує текст запиту та визначає основні параметри пошуку товарів. Далі back-end надсилає запит до OpenAI API для створення embedding [33] запиту користувача. OpenAI повертає embedding-вектор, який використовується частиною через HTTP API. Після надсилання повідомлення користувача запит передається до back-end AI для семантичного пошуку у системі vector search.

Наступним та ключовим етапом є Embedding. Для кожного товару під час його створення формується векторне представлення на основі даних товару [33], наприклад назви, опису, характеристик, категорії, підкатегорії, кольору, розміру та інших параметрів. Дані embeddings зберігаються у векторній базі даних PostgreSQL з використанням розширення pgvector [29, 32]. Даний підхід використовується для семантичного пошуку та порівняння схожості товарів [3, 33] у системі vector search. Після створення embedding запиту back-end AI service виконує vector search [32-33] серед embeddings товарів та отримує найбільш релевантні результати. Знайдені товари використовуються як контекст для подальшої генерації відповіді ШІ-асистента.

Для підвищення точності та актуальності відповідей ШІ-асистента використовується підхід RAG [11]. Основна ідея RAG полягає у поєднанні великої мовної моделі з пошуком актуальної інформації у власній базі даних вебзастосунку. На відміну від звичайних ШІ-чатів, які формують відповідь лише на основі власного навчання, RAG дозволяє додатково передавати до ШІ-моделі знайдені товари та інформацію про них у вигляді контексту. Після виконання vector search система отримує найбільш релевантні товари та передає їх у prompt для ШІ-моделі. Завдяки цьому ШІ-асистент формує відповіді на основі реальних даних із бази вебзастосунку, що зменшує ймовірність помилок або вигаданої інформації. Використання RAG особливо важливе для систем електронної комерції, оскільки асортимент товарів, характеристики та ціни можуть постійно змінюватися. Далі відбувається формування prompt для ШІ-моделі. Prompt містить правила поведінки ШІ-асистента, обмеження та структуру відповіді. Наприклад, ШІ-асистенту заборонено вигадувати товари, яких немає у базі даних, а також формувати відповіді, що не відповідають заданим правилам системи. Після формування prompt разом із контекстом товарів запит надсилається до OpenAI API [11, 32-34]. Мовна модель OpenAI аналізує prompt та знайдені товари, після чого генерує відповідь для користувача. Отримана відповідь повертається до back-end AI service,

де додатково може форматуватися перед відправленням користувачу. На завершальному етапі відповідь ШІ-асистента відображається у чаті вебзастосунку.

Висновки до розділу 2

У другому розділі було розглянуто та обґрунтовано вибір технологій для розроблення вебзастосунку електронної комерції з інтегрованим ШІ-асистентом. Було проаналізовано сучасні підходи до веброзробки, зокрема front-end та back-end розробку, архітектурні підходи, моделі побудови вебзастосунків, а також роль API, баз даних та хмарних сервісів у сучасних системах електронної комерції.

У процесі аналізу архітектурних підходів було визначено, що для розроблюваної системи найбільш доцільним є використання модульного клієнт-серверного підходу з можливістю подальшого переходу до мікросервісної архітектури. Такий варіант дозволяє розділити клієнтську частину, серверну логіку, базу даних та ШІ-модуль, що спрощує підтримку системи, її розвиток і масштабування.

Було проаналізовано популярні front-end та back-end технології. Для клієнтської частини обрано React, TypeScript, Vite, Tailwind CSS, HeroUI та Framer Motion. Для серверної частини доцільним є використання Node.js, Express та Prisma. Такий стек забезпечує швидку розробку, компонентний підхід, зручну роботу з REST API, типізацію та можливість інтеграції додаткових сервісів.

Окрему увагу було приділено проєктуванню структури вебзастосунку. Було визначено основні навігаційні елементи інтерфейсу, зокрема header, menu, breadcrumbs, search, filter та footer. Їхнє правильне розташування дозволяє покращити користувацький досвід, спростити пошук товарів і підвищити ефективність взаємодії користувача з онлайн-магазином.

Для зберігання даних було обрано PostgreSQL, оскільки ця база даних добре підходить для роботи зі складними зв’язками між користувачами, товарами, категоріями, замовленнями та характеристиками. Додатковою перевагою є підтримка pgvector, що дозволяє зберігати embeddings товарів та реалізувати vector

search для роботи ШІ-асистента. Також було спроєктовано взаємодію з ШІ-асистентом, який використовується для допомоги користувачам у виборі товарів. Було визначено, що поєднання OpenAI API, embeddings, vector search та RAG дозволяє формувати відповіді на основі реальних даних із бази вебзастосунку. Це зменшує ризик появи помилкових відповідей та підвищує якість рекомендацій.

Отже, у результаті другого розділу було сформовано технічну основу майбутнього вебзастосунку. Обрані технології, архітектурні рішення, структура бази даних та принцип взаємодії з ШІ-асистентом забезпечують гнучкість, масштабованість, зручність підтримки та можливість подальшого розвитку системи електронної комерції.

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ З ШІ-АСИСТЕНТОМ

3.1 Опис використаних програмних засобів та інструментів для розробки програмного забезпечення

Для розробки сучасного вебзастосунку одного інструменту не вистачить, для створення електронної комерції було використано сучасне середовище розробки та набір потужних інструментів, які забезпечують проєктування інтерфейсів, написання коду, тестування, керування базою даних, основними інструментами стали Figma, WebStorm, Doker Desktop, Google Chrome DevTools, DBeaver та GitHub. Дані інструменти дозволять організувати повний цикл розробки вебзастосунку, від створення дизайну до тестування та розгортання системи.

Для проєктування інтерфейсу та створення макетів сторінок та інших елементів дизайну було використано сервіс Figma. Даний інструмент дозволяє створювати прототипи сторінок, проєктувати UX/UI дизайн та моделювати взаємодію між користувачем та системою. У процесі розробки використовувася для створення сторінок, компонентів інтерфейсів, банерів, карток товарів, навігаційних елементів та адаптивного дизайну вебзастосунку. Приклад середовища для проєктування UX/UI дизайну наведено на рисунку 3.1.

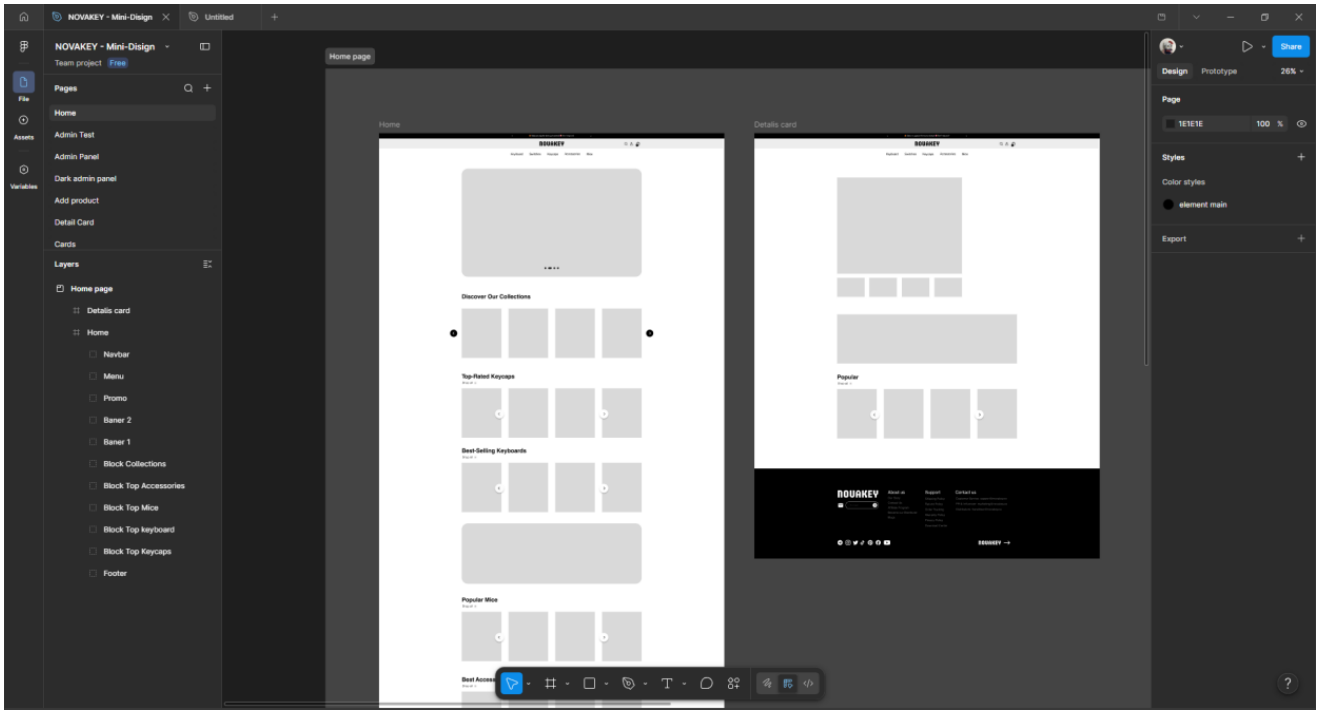

Рисунок 3.1 – Проєктування інтерфейсу вебзастосунку у Figma

Основним середовищем написання програмного коду було обрано WebStorm. Це інтегроване середовище розробки від компанії JetBrains. WebStorm забезпечує підтримку JavaScript, TypeScript, React, Node.JS та інших сучасних вебтехнологій. Середовище має інструменти автоматичного доповнення коду, інтеграцію з GitHub системою, підтримку Docker та роботу з REST API. Це дозволило зробити процес розробки значно ефективніше та зручніше. Приклад робочого середовища WebStorm наведено на рисунку 3.2.

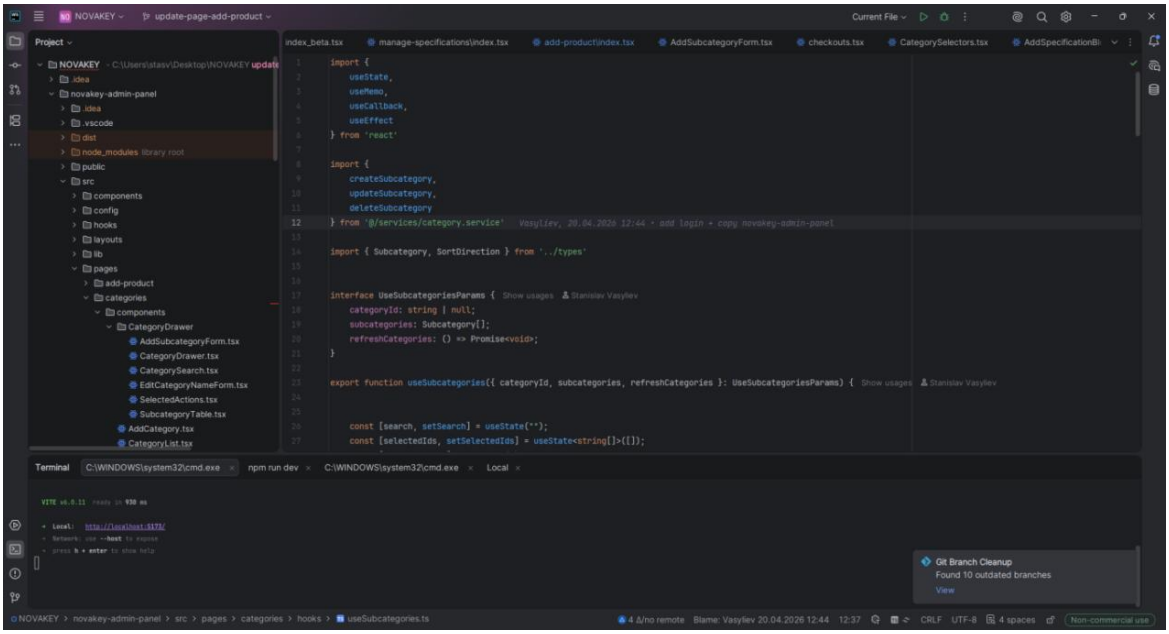

Рисунок 3.2 – Середовище розробки WebStorm

Для контейнеризації застосунку та запуску серверних сервісів було використано Docker Desktop. Docker дозволив ізолювати середовище виконання вебзастосунку, бази даних PostgreSQL та інших сервісів у контейнерах. Використання Docker спростило розгортання проєкту, забезпечило стабільність середовища та сумісність між різними операційними системами. Приклад роботи контейнерів наведено на рисунку 3.3.

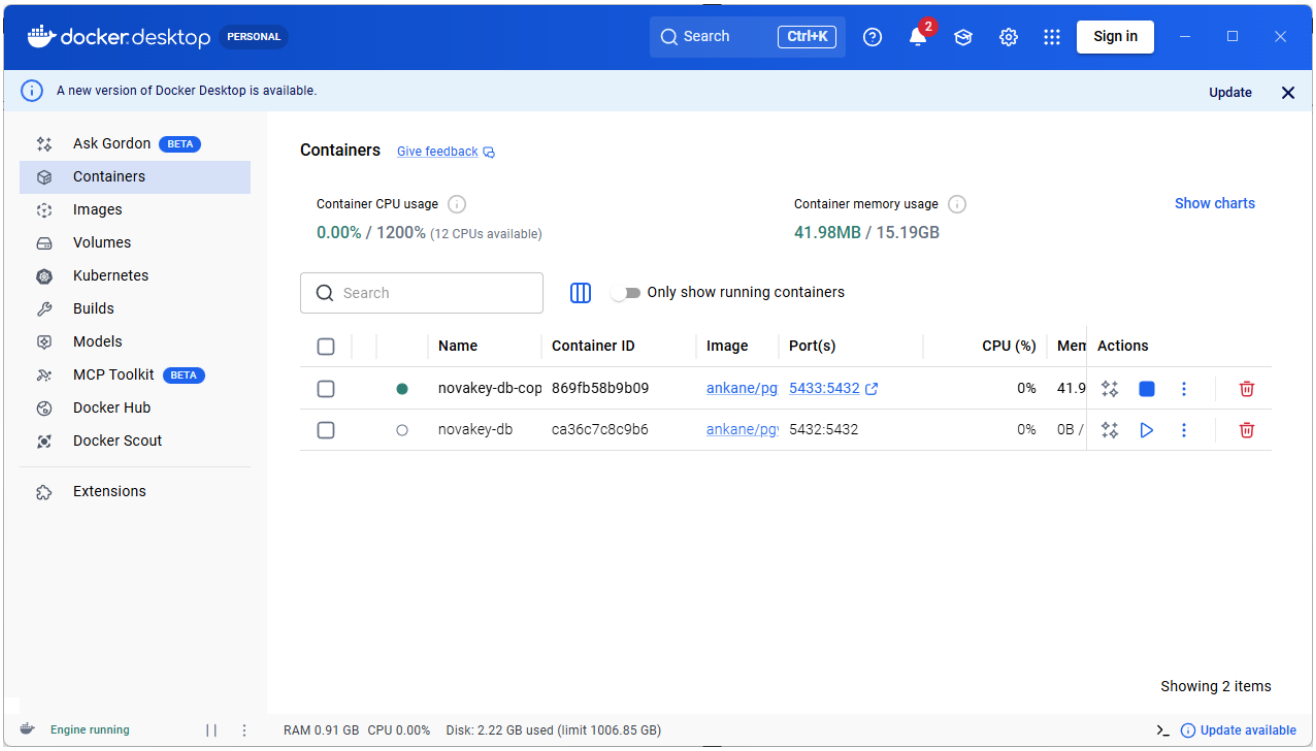

Рисунок 3.3 – Використання Docker Desktop для контейнеризації системи

Для тестування та налагодження клієнтської частини вебзастосунку було використано інструменти розробника браузера Google Chrome DevTools. Chrome DevTools дозволяє аналізувати HTML та CSS структуру сторінки, перевіряти JavaScript код, відстежувати HTTP-запити, тестувати адаптивність інтерфейсу та оптимізувати продуктивність вебзастосунку. Також за допомогою даного інструменту виконувалася перевірка Local Storage, Session Storage, Cookies та JWT-токенів авторизації. Це дозволило контролювати процес автентифікації користувачів, зберігання токенів доступу, передачу cookie-файлів між клієнтською та серверною частинами, а також перевіряти коректність роботи системи авторизації та захищених HTTP-запитів. Додатково Chrome DevTools застосовувався для аналізу використання пам’яті браузера, продуктивності вебзастосунку та пошуку можливих витоків пам’яті під час роботи React-компонентів. Приклад використання інструментів браузера наведено на рисунку 3.4.

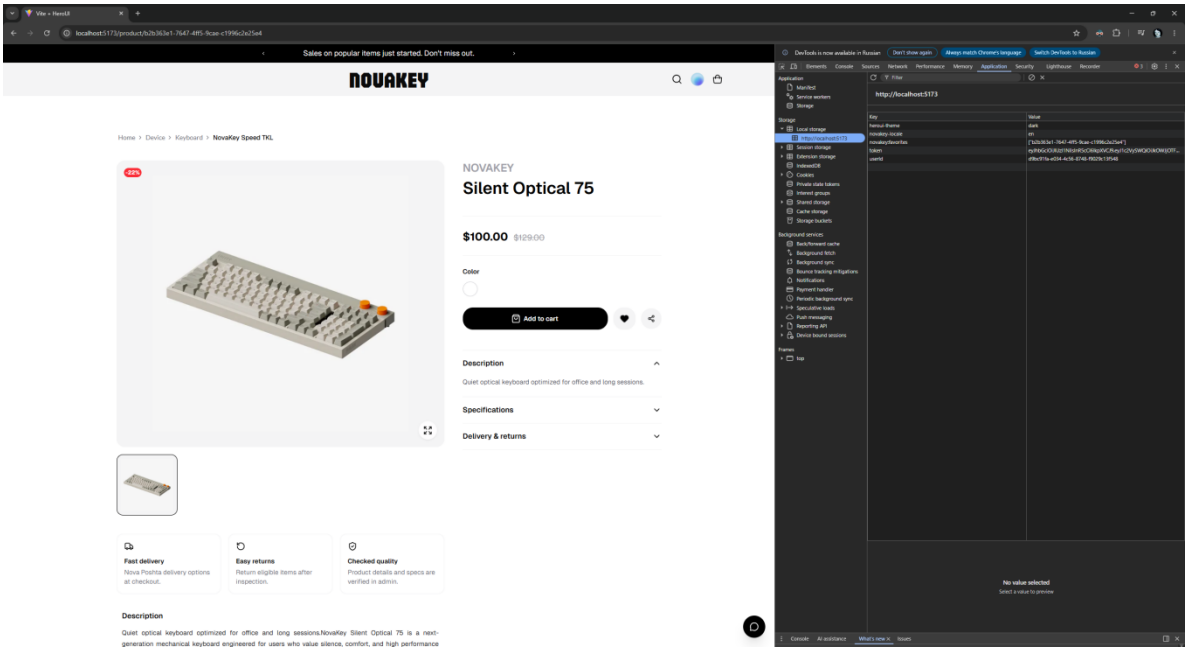

Рисунок 3.4 – Налагодження вебзастосунку за допомогою Chrome DevTools

Для роботи з базою даних PostgreSQL використано інструмент DBeaver. DBeaver забезпечує зручний графічний інтерфейс для створення таблиць, виконання SQL-запитів, перегляду структури бази даних та адміністрування даних. Використання даного інструменту дозволило спростити проєктування та налагодження структури бази даних вебзастосунку. Приклад роботи з PostgreSQL наведено на рисунку 3.5.

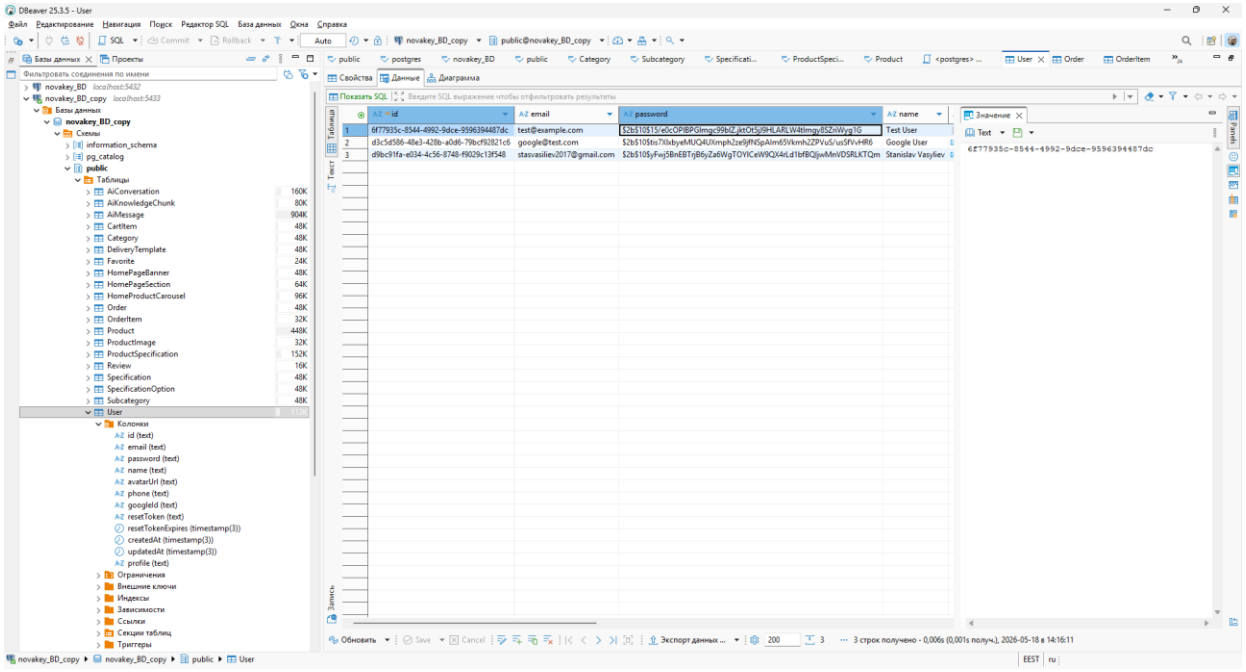

Рисунок 3.5 – Робота з базою даних PostgreSQL у Dbeaver

Для контролю версій та збереження вихідного коду використовувався сервіс GitHub. GitHub дозволяє зберігати репозиторії проєкту, відстежувати зміни у коді, працювати з гілками та забезпечує резервне копіювання програмного коду. Також GitHub використовувався для синхронізації клієнтської та серверної частини вебзастосунку між різними середовищами розробки. Приклад репозиторію наведено на рисунку 3.6.

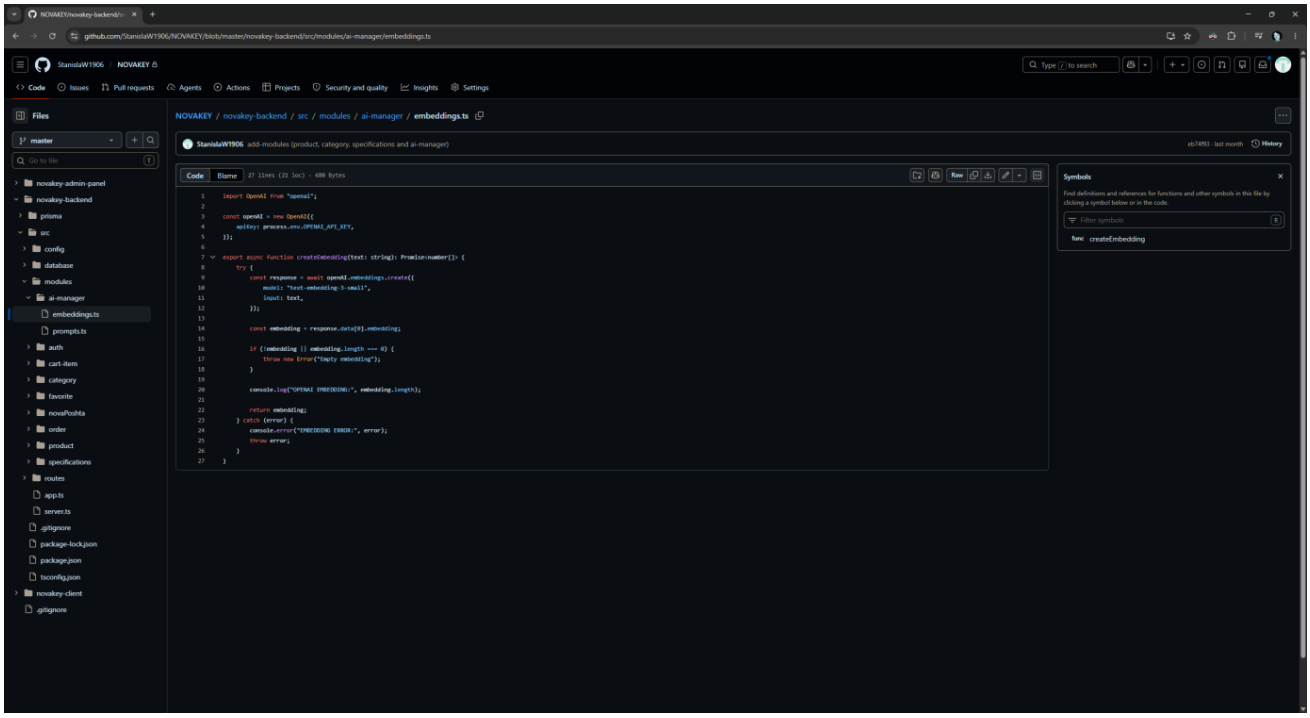

Рисунок 3.6 – Репозиторій проєкту на GitHub

Отже, використані інструменти розробки забезпечили повний набір засобів для створення сучасного вебзастосунку електронної комерції. Поєднання інструментів для дизайну, програмування, тестування, контейнеризації та керування базою даних дозволило організувати ефективний процес створення та підтримки системи.

3.2 Реалізація серверної частини

Серверна частина вебзастосунку реалізована на базі Node.js, Express та TypeScript [27-28]. Вона виконує роль центрального програмного шару між клієнтським сайтом, адміністративною панеллю, базою даних та зовнішніми сервісами. Саме серверна частина відповідає за обробку HTTP-запитів, виконання бізнес-логіки, перевірку прав доступу, роботу з товарами, замовленнями, користувачами, оплатою, доставкою та ШІ-асистентом.

Архітектура backend-частини побудована за модульним принципом. Кожна функціональна область системи винесена в окремий модуль: авторизація, категорії, товари, кошик, обране, замовлення, Nova Poshta, AI-manager, контент головної

сторінки та інше. Такий підхід дозволяє не змішувати різні частини логіки в одному файлі, спрощує підтримку коду та робить систему зручнішою для подальшого розширення.

Загальна схема роботи серверної частини наведена на рисунку 3.7.

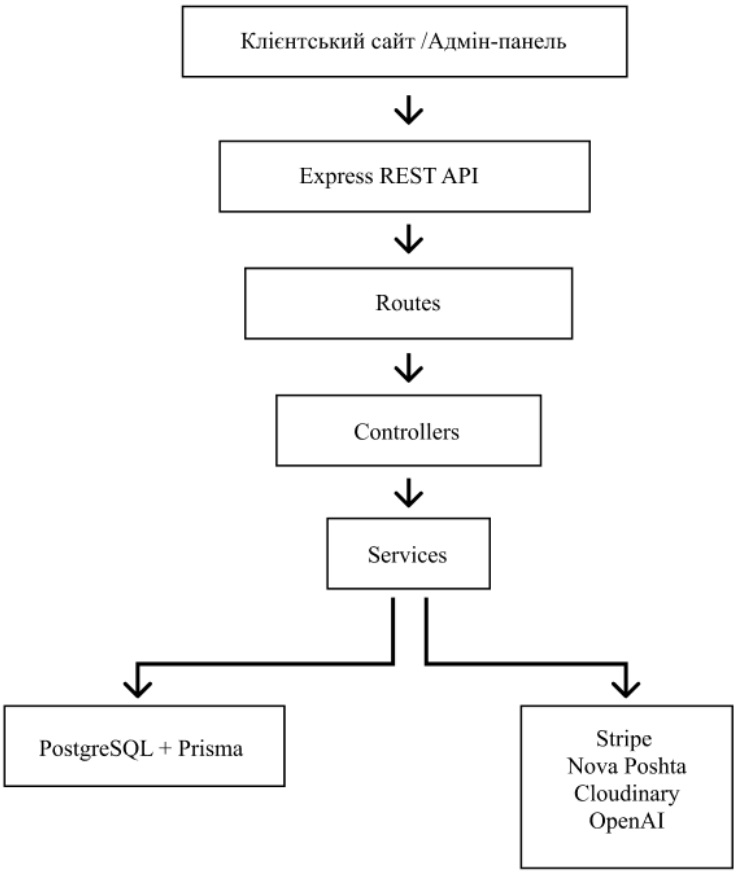

Рисунок 3.7 – Структура взаємодії компонентів серверної частини вебзастосунку

Запит від клієнтського сайту або адміністративної панелі надходить до Express-додатку. Далі він передається у відповідний маршрутизатор, який визначає, який саме контролер повинен обробити запит. Контролер отримує дані з HTTP-запиту, викликає потрібний сервіс та формує відповідь для клієнта. Основна бізнес-логіка винесена у сервісний шар, який взаємодіє з базою даних через Prisma ORM [30] або звертається до зовнішніх API.

У проєкті було реалізовано REST API для взаємодії між клієнтською частиною, адміністративною панеллю та сервером. Основні маршрути API згруповані за функціональними модулями. Наприклад, маршрути /categories відповідають за роботу з категоріями, /products - за товари, /auth - за авторизацію, /cart-item - за кошик, /order - за замовлення, /api/nova-poshta - за інтеграцію з Новою Поштою, а /api/AI-manager - за роботу ШІ-асистента. Приклад реалізації реєстрації основних маршрутів REST API наведено в додатку Г.

Для демонстрації реалізації модульної архітектури серверної частини було розглянуто модуль роботи з категоріями товарів. Даний модуль реалізований за багаторівневим принципом та складається з маршрутизатора, контролера та сервісного шару. Такий підхід дозволяє відокремити HTTP-логіку від бізнес-логіки та взаємодії з базою даних. Модуль містить окремі файли category.routes.ts, category.controller.ts та category.service.ts.

Файл маршрутів відповідає за реєстрацію HTTP endpoint та визначення відповідних обробників запитів. Лістинг реалізації маршрутизатора модуля категорій наведено в додатку В.

Контролер приймає HTTP-запити від клієнтської частини, отримує параметри запиту через req.params або req.body, після чого передає їх у сервісний шар. Після виконання необхідної логіки контролер повертає JSON-відповідь клієнту. Лістинг реалізації контролера наведено в додатку Б.

Основна логіка роботи з категоріями реалізована у сервісному шарі. Сервіс містить методи створення, редагування, видалення та отримання категорій і підкатегорій. Лістинг реалізації сервісу управління категоріями наведено в додатку А. Для взаємодії з PostgreSQL використовується Prisma ORM, що забезпечує типізовану роботу з моделями бази даних та спрощує виконання CRUD-операцій [29-30].

Завдяки використанню модульної архітектури кожна функціональна область серверної частини реалізується незалежно від інших модулів. Аналогічним чином

у системі побудовані модулі товарів, замовлень, кошика, обраного, авторизації, доставки та AI-manager.

У серверній частині було реалізовано такі функціональні можливості:

– REST API для клієнтського сайту та адміністративної панелі;
– реєстрацію й авторизацію користувачів;
– хешування паролів за допомогою bcrypt;
– авторизацію через JWT-токени;
– керування товарами, категоріями, підкатегоріями та характеристиками;
– завантаження зображень товарів через multer і Cloudinary;
– роботу з кошиком та обраними товарами;
– оформлення та обробку замовлень;
– інтеграцію зі Stripe для онлайн-оплати;
– інтеграцію з Nova Poshta API для роботи з містами, відділеннями, ТТН і трекінгом;
– зміну залишків товарів після створення або скасування замовлення;
– AI-manager для пошуку товарів, порівняння характеристик і формування відповідей користувачу.

Важливою частиною серверної реалізації є авторизація користувачів. Після успішного входу в систему сервер формує JWT-токен, який надалі використовується для підтвердження особи користувача під час звернення до захищених маршрутів. Це дозволяє обмежити доступ до адміністративних функцій і персональних даних користувача.

Для роботи із зображеннями товарів використовується multer, який приймає файли з клієнтської частини та Cloudinary, який зберігає ці файли у хмарному сховищі. Такий підхід дозволяє не зберігати медіафайли безпосередньо на сервері та спрощує масштабування системи.

Окремим напрямом реалізації є інтеграція із зовнішніми сервісами. Stripe використано для онлайн-оплати замовлень, Nova Poshta API — для отримання міст, відділень, створення ТТН і перевірки статусу доставки, OpenAI API — для роботи

ШІ-асистента [34]. Завдяки винесенню цієї логіки у сервісний шар зовнішні інтеграції не перевантажують контролери та можуть бути змінені або розширені окремо.

Таким чином, серверна частина вебзастосунку забезпечує основну бізнес-логіку системи та є зв’язувальним елементом між клієнтським інтерфейсом, адміністративною панеллю, базою даних і зовнішніми сервісами. Модульна структура, використання TypeScript, Express і Prisma ORM дозволили створити зрозумілу, масштабовану та підтримувану backend-архітектуру.

3.3 Реалізація адміністративної частини вебзастосунку

Адміністративна частина вебзастосунку призначена для керування основними бізнес-процесами інтернет-магазину: товарами, категоріями, характеристиками, замовленнями, аналітикою бізнесу та контентом сторінок. Вона реалізована як окремий веб-застосунок, що взаємодіє з серверною частиною через REST API.

Адмін-панель дозволяє адміністратору виконувати такі дії:

- додавати, редагувати, публікувати та приховувати товари;
- керувати категоріями та підкатегоріями каталогу;
- налаштовувати характеристики товарів для кожної підкатегорії;
- переглядати та обробляти замовлення користувачів;
- змінювати статуси оплати й доставки;
- генерувати та оновлювати ТТН для доставки через Nova Poshta;
- редагувати банери, каруселі товарів і блоки головної сторінки;
- аналізувати статистуку бізнесу.

Адміністративна частина побудована на основі React, TypeScript та Vite [17-18]. Для стилізації використано Tailwind CSS, компоненти HeroUI, Radix UI. Серверна частина реалізована на Express.js із використанням Prisma ORM та PostgreSQL [28-30 адаптивним].

Структура адміністративної частини вебзастосунку побудована за модульним принципом [35], що забезпечує розділення функціональності на окремі логічні компоненти. Такий підхід спрощує підтримку системи, масштабування функціональності та подальший розвиток вебзастосунку. Кожен модуль відповідає за виконання окремих бізнес-процесів інтернет-магазину та взаємодіє із серверною частиною через REST API. До складу адмінпанелі застосунку входять такі модулі як:

– *модуль керування товарами* - забезпечує створення, редагування, публікацію та приховування товарів, завантаження зображень, налаштування цін, знижок, характеристик і категорій та додавання ;

– *модуль категорій і підкатегорій* - дозволяє формувати структуру каталогу інтернет-магазину, створювати вкладені категорії та організовувати товари за тематичними розділами;

– *модуль характеристик* - використовується для налаштування параметрів товарів залежно від підкатегорії. Наприклад, для клавіатур можуть задаватися тип перемикачів, формат корпусу та тип підключення;

– *модуль замовлень* - призначений для перегляду оформлених замовлень, зміни статусів оплати та доставки, перегляду інформації про покупця та складу замовлення;

– *модуль доставки Nova Poshta* - забезпечує інтеграцію із сервісом доставки Nova Poshta, дозволяє генерувати товарно-транспортні накладні (ТТН), оновлювати статуси відправлень і працювати з відділеннями та поштоматами;

– *модуль керування контентом головної сторінки* - використовується для редагування банерів, товарних каруселей, рекламних блоків та інших елементів інтерфейсу головної сторінки;

– *модуль аналітики* - надає інформацію про кількість замовлень, популярні товари, прибуток та інші статистичні показники діяльності інтернет-магазину.

Для зручності користування всі модулі об’єднані єдиною системою навігації адміністративної панелі. Це дозволяє адміністратору швидко переходити між розділами системи та ефективно керувати бізнес-процесами вебзастосунку.

Таблиця 3.1– Основні модулі адміністративної частини вебзастосунку

Модуль	**Призначення**
Аналітика	Аналіз статистики продажів і активності користувачів
Керування товарами	Створення, редагування та публікація товарів
Замовлення	Обробка та контроль замовлень
Nova Poshta	Генерація ТТН та керування доставкою
Контент головної сторінки	Редагування банерів і каруселей
Категорії та підкатегорії	Формування структури каталогу
Характеристики	Налаштування параметрів товарів

Модуль керування товарами є одним із ключових компонентів адміністративної частини вебзастосунку, оскільки забезпечує повний цикл роботи з товарними позиціями інтернет-магазину. За допомогою цього модуля адміністратор може створювати нові товари, редагувати існуючі записи, керувати відображенням товарів у каталозі та налаштовувати їх характеристики.

Сторінка керування товарами реалізована у вигляді таблиці зі списком усіх товарних позицій. Для зручності роботи передбачено пошук, фільтрацію та сортування товарів. Кожен запис містить основну інформацію про товар: назву, категорію, ціну, наявність знижки, статус публікації, дату створення та функція попереднього перегляду товару. На рисунках 3.12 – 3.15 представлено блоки заповнення інформації про товар. На рисунку 3.2 зображено елементи 1 та 2. Елемент 1 призначений для введення основної інформації про товар: категорії, підкатегорії, бренду, моделі, назви та опису продукту. Елемент 2 реалізує функцію попереднього перегляду товару на клієнтській частині вебзастосунку, що дозволяє

адміністратору перевірити коректність відображення інформації перед публікацією товару. Детальніше роботу елемента 2 буде розглянуто на рисунку 3.16.

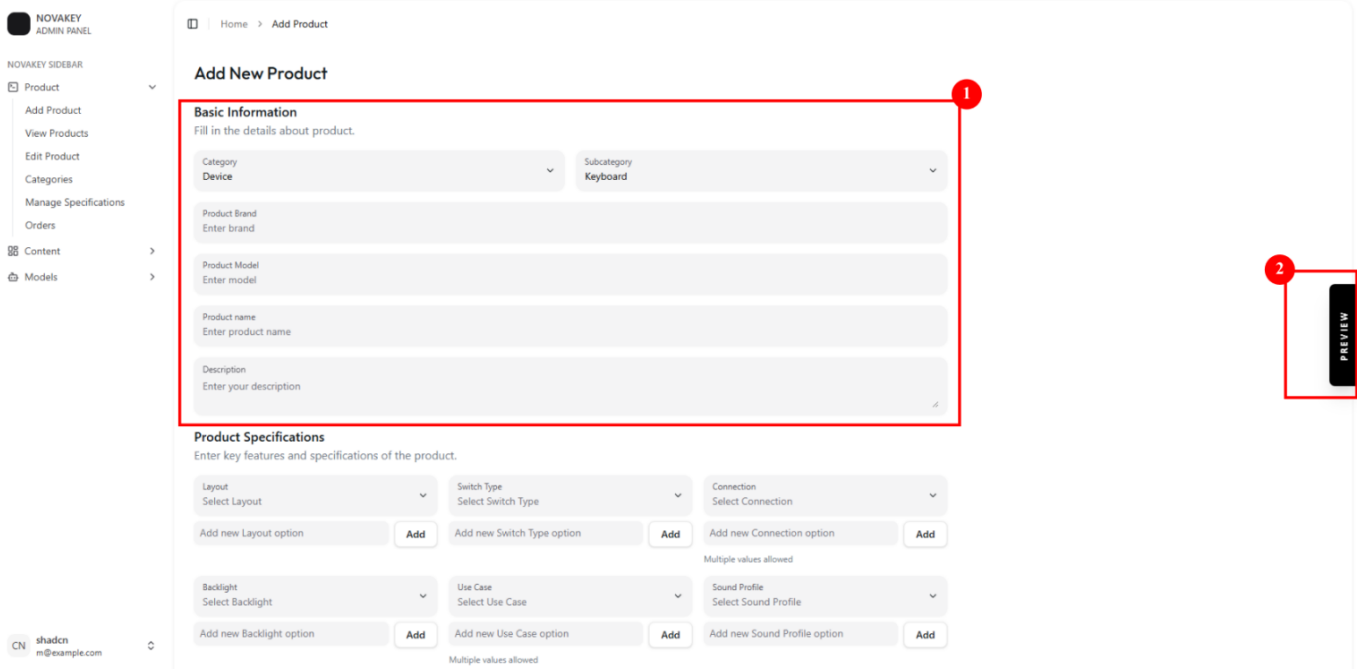

Рисунок 3.12 – Сторінка додавання товару. Основна інформація

Далі на рисунку 3.13 можна побачити елементи 3 та 4, які відповідають за характеристики товару. Елемент під номером 3 призначений для керування характеристиками товару, які налаштовуються в окремому модулі на іншій сторінці адміністративної панелі. Елемент 4 використовується для вибору кольору товару. Користувач може обрати колір із варіантів, що зберігаються в базі даних, або задати власний кастомний варіант кольору.

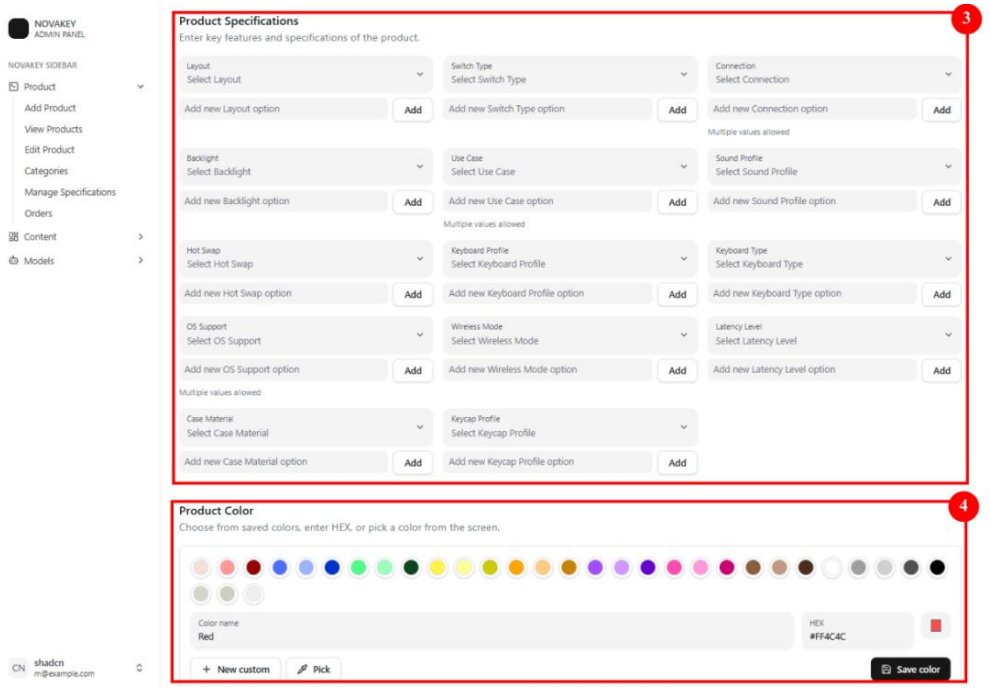

Рисунок 3.13 – Сторінка додавання товару. Характеристики та кольори

Наступні елементи 5 і 6 на рисунку 3.14, а також елементи 7 і 8 на рисунку 3.15 використовуються для налаштування додаткової інформації про товар. Елемент 5 призначений для встановлення статусу товару, зокрема бейджів NEW, SALE, SOLD OUT, HIT та інших позначок. Елемент 6 використовується для завантаження фотографій товару. Елемент 7 призначений для створення детального опису товару, який відображається після основної інформації. За допомогою цього елемента можна додавати банери, інформаційні блоки та інший контент, а також налаштовувати його відображення у стилі Bento. Комірки можуть гнучко налаштовуватися за розміром і кольором, що дозволяє створювати індивідуальний дизайн сторінки товару. Елемент 8 використовується для налаштування знижок, промоакцій та кількості товару. Реалізовано можливість вибору типу знижки - фіксована сума або відсоткове значення, а також налаштування періоду дії акції з точністю до днів і годин.

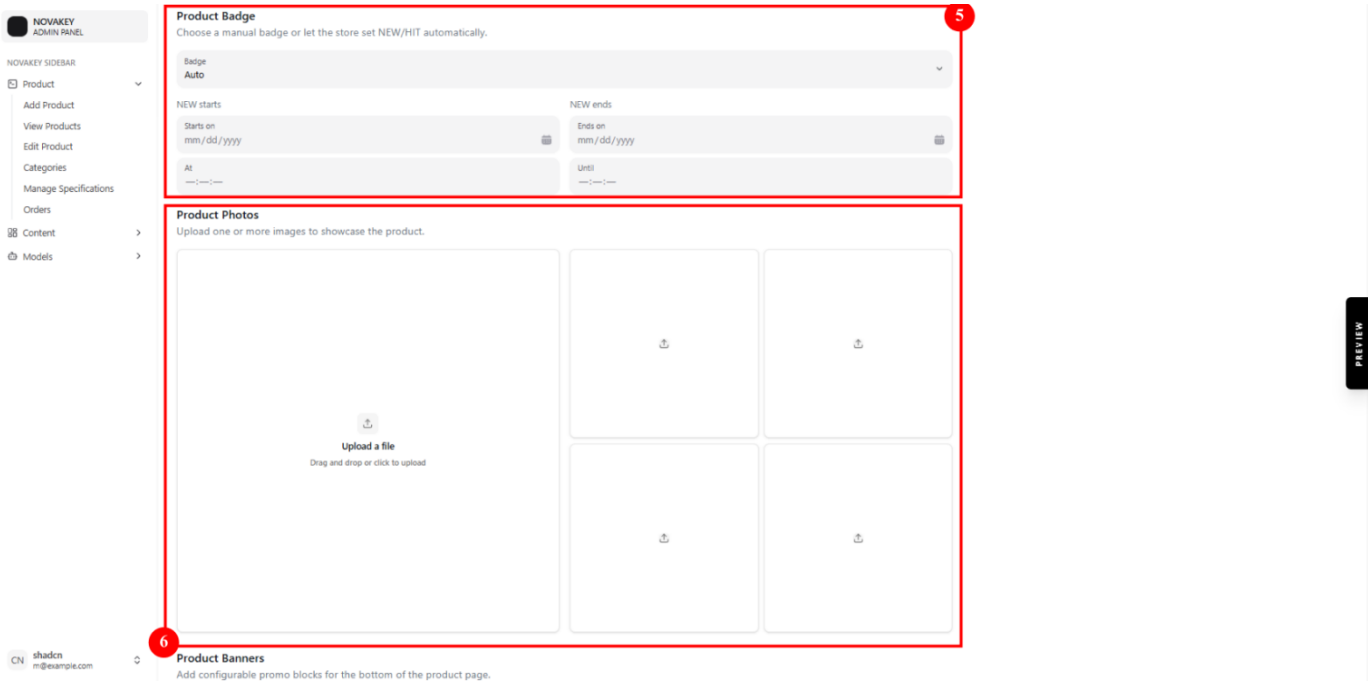

Рисунок 3.14 – Сторінка додавання товару. Статуси та зображення

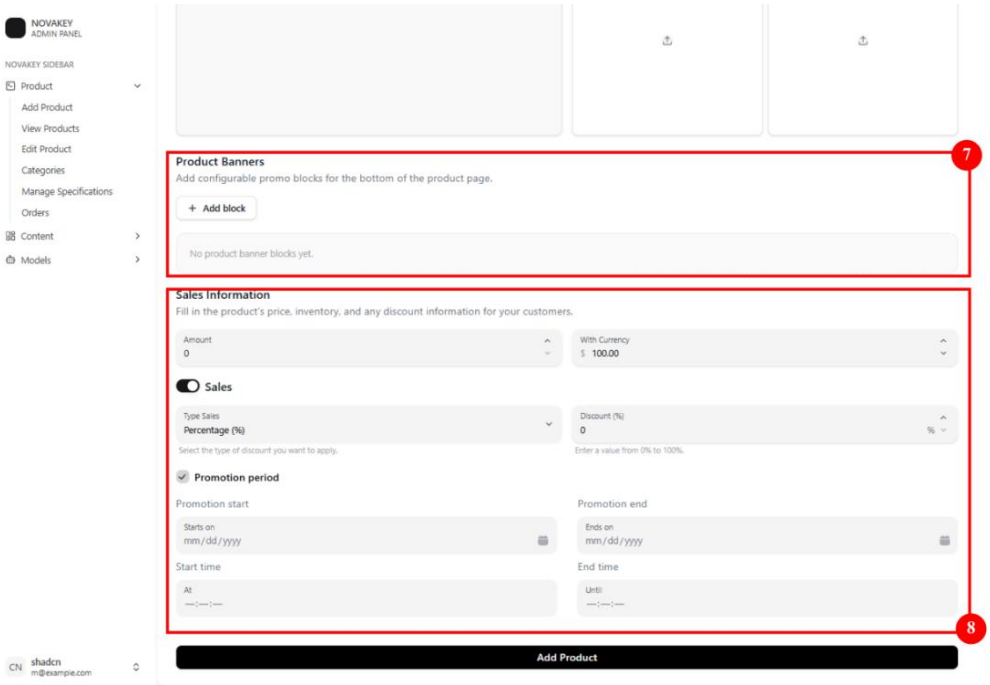

Рисунок 3.15 – Сторінка додавання товару Сторінка додавання товару. Опис і налаштування знижок

Останній елемент 9 на рисунку 3.16 являє собою модальне вікно для попереднього перегляду інформації перед публікацією товару. Дане вікно дозволяє адміністратору перевірити коректність відображення основної інформації,

характеристик, зображень, банерів, цін, знижок та інших елементів сторінки товару на клієнтській частині вебзастосунку перед його остаточною публікацією. У разі виявлення помилок або некоректного відображення інформації адміністратор може швидко повернутися до редагування товару та внести необхідні зміни.

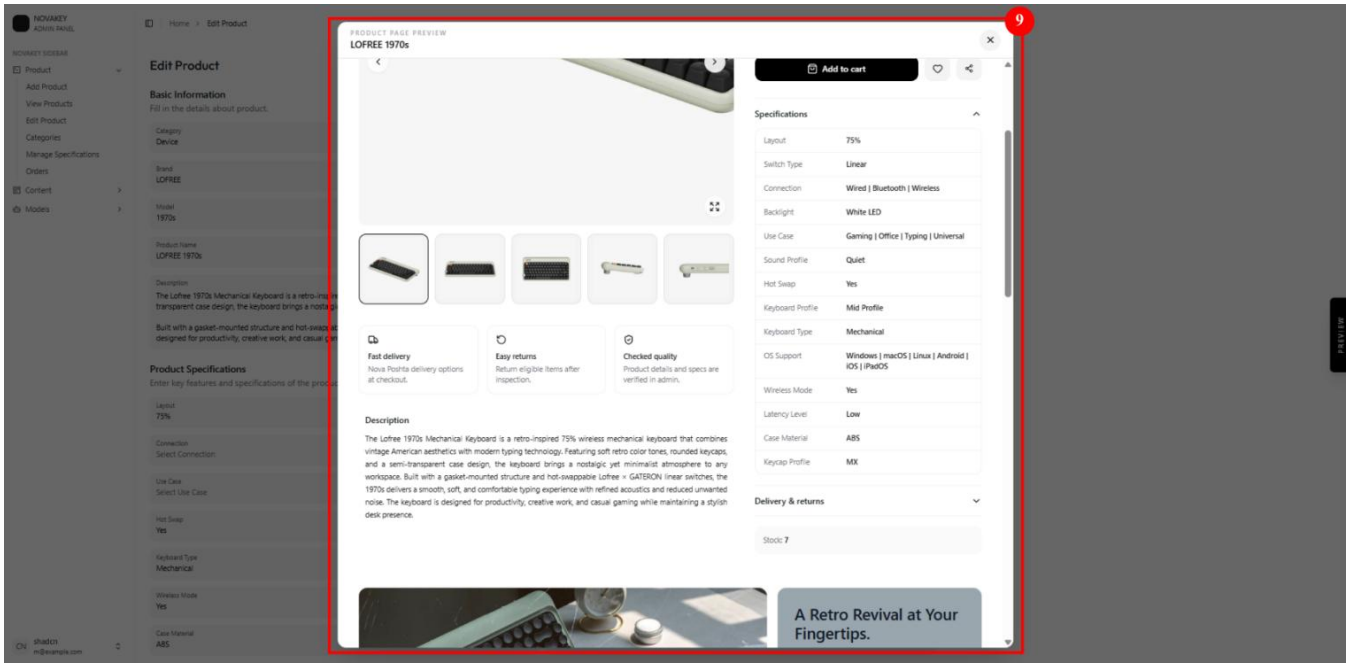

Рисунок 3.16 – Модальне вікно для попереднього перегляду інформації

Сторінка керування категоріями та підкатегоріями реалізована у вигляді таблиці зі списком усіх категорій магазину. Для зручності адміністрування передбачено пошук, фільтрацію та сортування категорій. Кожен запис містить основну інформацію, зокрема назву категорії, кількість підкатегорій та функції редагування й видалення. Також реалізовано можливість створення вкладених підкатегорій та їх редагування.

На рисунку 3.17 можна побачити елементи 1–3, які є основними функціональними елементами сторінки керування категоріями. Елемент 1 являє собою кнопку додавання нової категорії. Після натискання відкривається додатковий блок для введення назви категорії. Даний блок представлено на рисунку 3.18, де зображено елемент 8.

Наступні елементи 2 та 3 відповідають за додаткові функції керування категоріями. Елемент 2 використовується для пошуку категорій за заданими критеріями. Елемент 3 являє собою розкривний блок, який містить функціонал перегляду підкатегорій, видалення категорії разом із усіма вкладеними підкатегоріями, а також відкриття бокового модального вікна для більш гнучкого налаштування як самої категорії, так і її підкатегорій. Дане бокове модальне вікно можна побачити на рисунку 3.19.

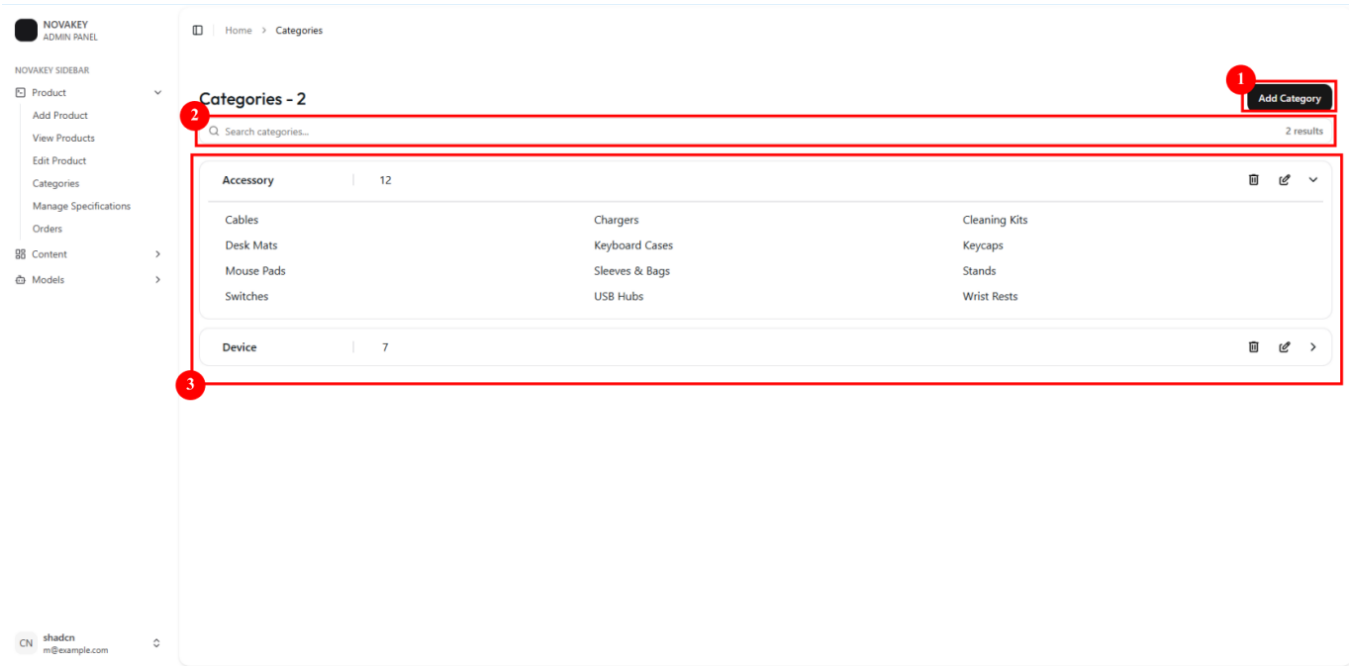

Рисунок 3.17 – Сторінка управління категоріями та підкатегоріями.

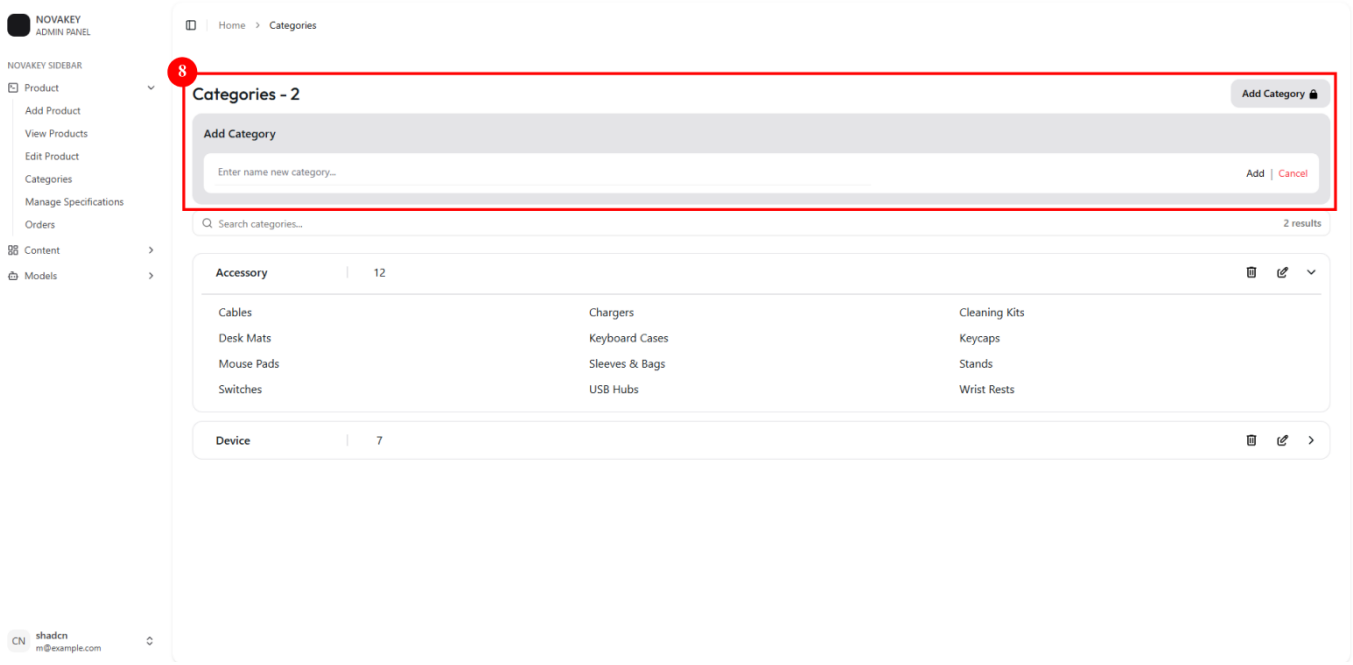

Рисунок 3.18 – Сторінка управління категоріями та підкатегоріями. Налаштування категорій

На рисунку 3.19 зображено елементи 4–7, які забезпечують більш детальне налаштування категорій та підкатегорій. Елемент 4 використовується для пошуку підкатегорій за заданими критеріями. Елемент 5 являє собою блок зі списком підкатегорій, реалізований у вигляді функціональної таблиці. Дана таблиця підтримує вибір декількох підкатегорій, фільтрацію, пагінацію, а також функції редагування назв і видалення підкатегорій. Елемент 6 призначений для додавання нових підкатегорій. Елемент 7 використовується для редагування назви категорії, що дозволяє адміністратору швидко змінювати структуру каталогу без необхідності переходу на інші сторінки адміністративної панелі.

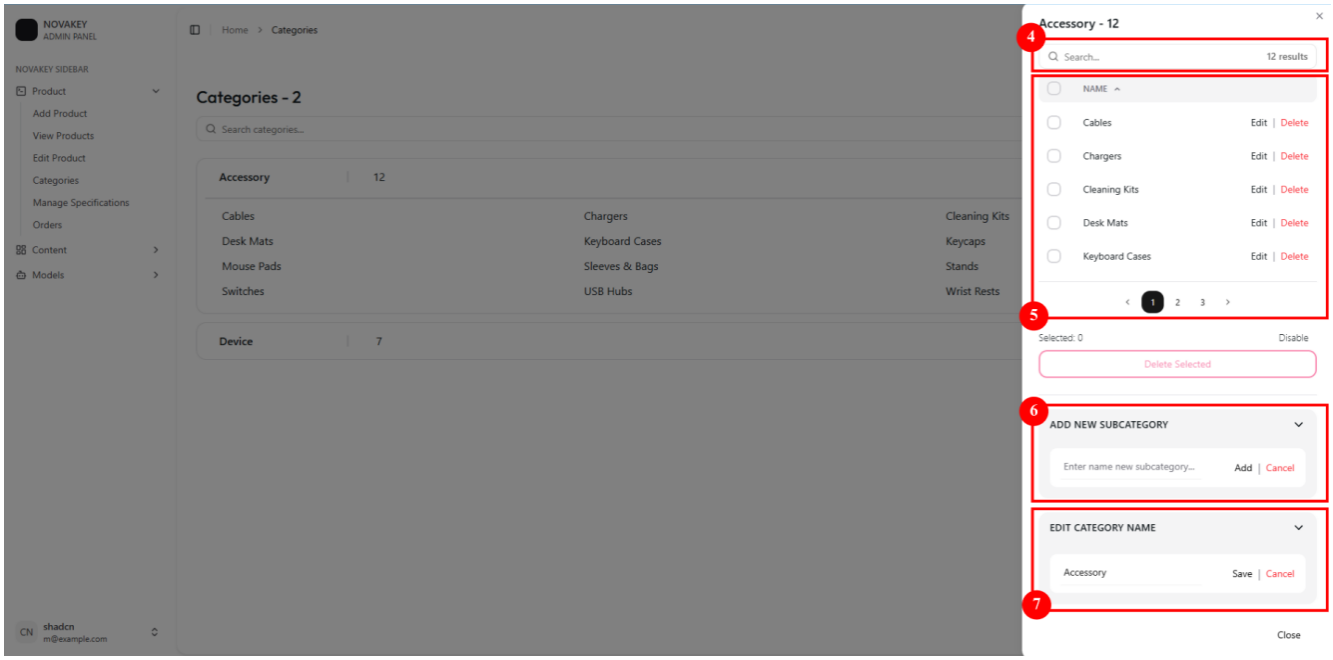

Рисунок 3.19 – Сторінка управління категоріями та підкатегоріями. Редагування підкатегорій

Сторінка перегляду товарів призначена для централізованого керування всіма товарними позиціями інтернет-магазину. Вона дозволяє адміністратору швидко переглянути інформацію про товари, виконувати пошук, сортування та фільтрацію, а також керувати статусом публікації товарів. Основний інтерфейс сторінки реалізований у вигляді функціональної таблиці, що забезпечує зручну роботу з великою кількістю товарів.

На рисунку 3.20 можна побачити основні елементи 1–3 сторінки перегляду товарів. Елемент 1 призначений для швидкого пошуку товарів за різними параметрами, зокрема за назвою, ID товару, категорією, підкатегорією, ціною, статусом товару та іншими характеристиками. Елемент 2 містить функціонал для сортування товарів за категоріями та підкатегоріями, оновлення списку товарів, а також скидання всіх застосованих фільтрів. Елемент 3 являє собою кастомізовану таблицю товарів, яка підтримує додаткову фільтрацію за окремими параметрами, вибір декількох товарів одночасно та виконання групових дій над ними. Також у кожному рядку таблиці реалізовано функції керування товаром, зокрема

відображення або приховування товару на клієнтській частині вебзастосунку, видалення товару, а також перехід до детального перегляду та редагування товару.

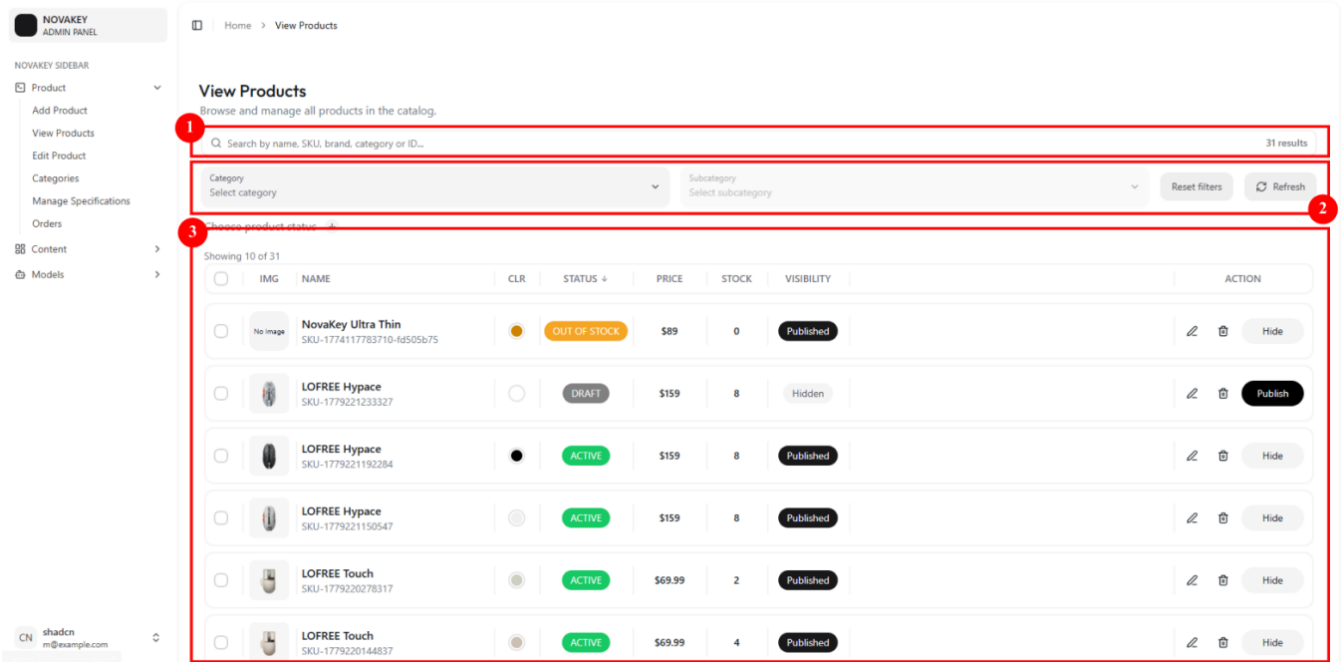

Рисунок 3.20 – Сторінка управління товарами

Наступна сторінка призначена для редагування вже створеного товару та за своєю структурою і функціоналом є подібною до сторінки додавання товару. Інтерфейс сторінки дозволяє адміністратору змінювати основну інформацію про товар, зокрема назву, категорію, підкатегорію, бренд, модель, опис, характеристики, кольори, статуси товару та інші параметри. Приклад сторінки редагування товару наведено на рисунку 3.21.

Також сторінка підтримує редагування фотографій товару, банерів, інформаційних блоків та додаткового контенту, який відображається на клієнтській частині вебзастосунку. Адміністратор може змінювати налаштування знижок, промоакцій, кількість товару на складі та період дії акційних пропозицій.

Наявна функція попереднього перегляду товару перед збереженням змін. Це дозволяє перевірити коректність відображення сторінки товару на клієнтській частині вебзастосунку та, у разі необхідності, швидко повернутися до редагування для виправлення помилок або зміни контенту.

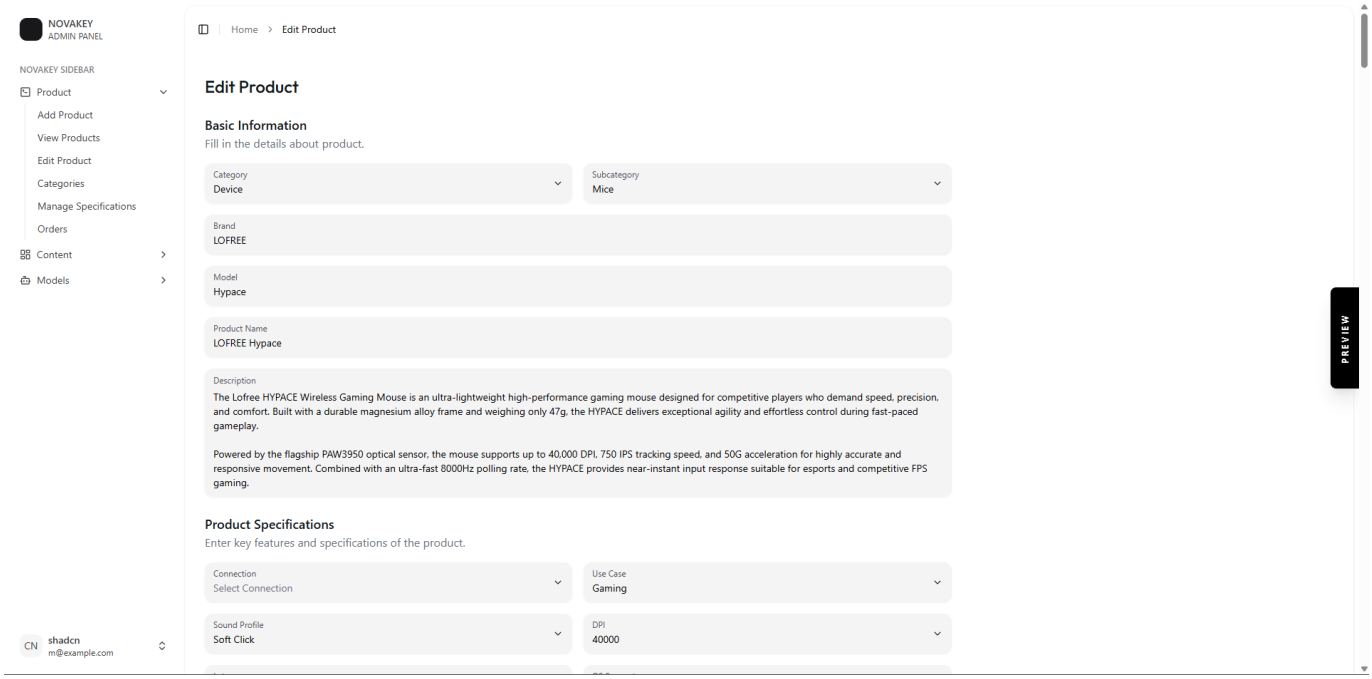

Рисунок 3.21 – Сторінка редагування товару

Сторінка керування характеристиками призначена для централізованого адміністрування параметрів товарів, які використовуються під час формування каталогу та сторінок товарів на клієнтській частині вебзастосунку. Реалізація даного модуля забезпечує уніфікацію характеристик для різних підкатегорій товарів та значно підвищує ефективність керування структурованими даними інтернет-магазину.

Кожна характеристика може бути прив’язана до певної категорії або групи товарів, що дозволяє формувати динамічні набори параметрів залежно від типу продукції. Приклад роботи модуля наведено на рисунках 3.22 – 3.23.

На рисунку 3.22 зображено елементи під номерами 1–4. Елемент 1 використовується для вибору категорії та відповідної підкатегорії для подальшого додавання характеристик товару. Елемент 2 являє собою кнопку додавання нової характеристики. Після натискання відкривається додатковий блок для введення назви характеристики та налаштування її параметрів за допомогою прапорців. Серед доступних параметрів передбачено можливість встановлення характеристики як обов’язкової для заповнення, а також дозвіл на вибір декількох

значень для однієї характеристики. Даний блок можна переглянути на рисунку 3.23, де представлено елемент 5. Наступний елемент 3 призначений для пошуку характеристик або значень характеристик за заданими параметрами. Елемент 4 являє собою функціональну таблицю з підтримкою фільтрації та сортування даних. За допомогою вбудованих фільтрів можна виконувати сортування характеристик за алфавітом, доступними опціями, кількістю значень у характеристиці та іншими параметрами.

Також у кожному рядку таблиці реалізовано активні функції керування характеристиками, зокрема редагування та видалення. Після натискання кнопки редагування відкривається бокове модальне вікно для детального налаштування характеристики та її значень. Дане модальне вікно представлено на рисунку 3.24, де зображено елементи 6–8.

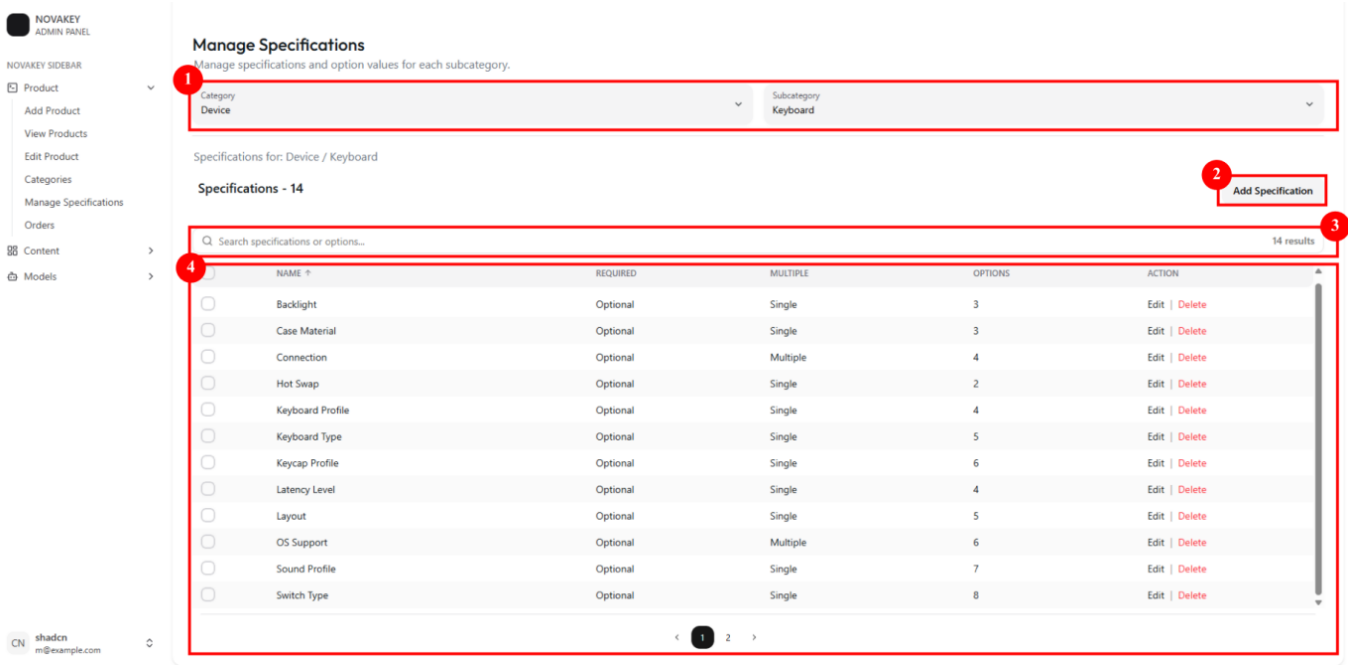

Рисунок 3.22 – Сторінка управління характеристиками. Загальний інтерфейс

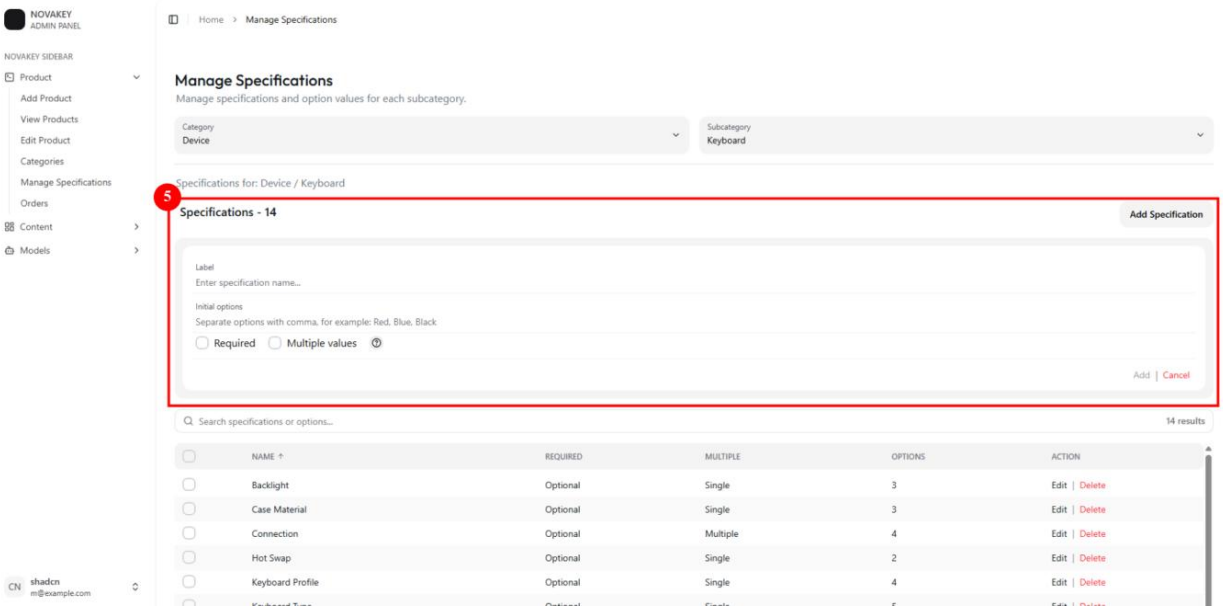

Рисунок 3.23 – Сторінка управління характеристиками. Таблиця характеристик

У даному модальному вікні реалізовано детальні налаштування характеристик. Елемент під номером 6 призначений для редагування назви характеристики та її параметрів, а також містить функцію пошуку значень характеристики. Елемент 7 являє собою кнопку, яка відкриває додатковий блок для додавання нового значення характеристики.

Елемент 8 представлений у вигляді таблиці зі списком значень характеристики. Таблиця підтримує сортування даних та містить функції редагування назв значень і їх видалення, що забезпечує зручне керування параметрами товарів.

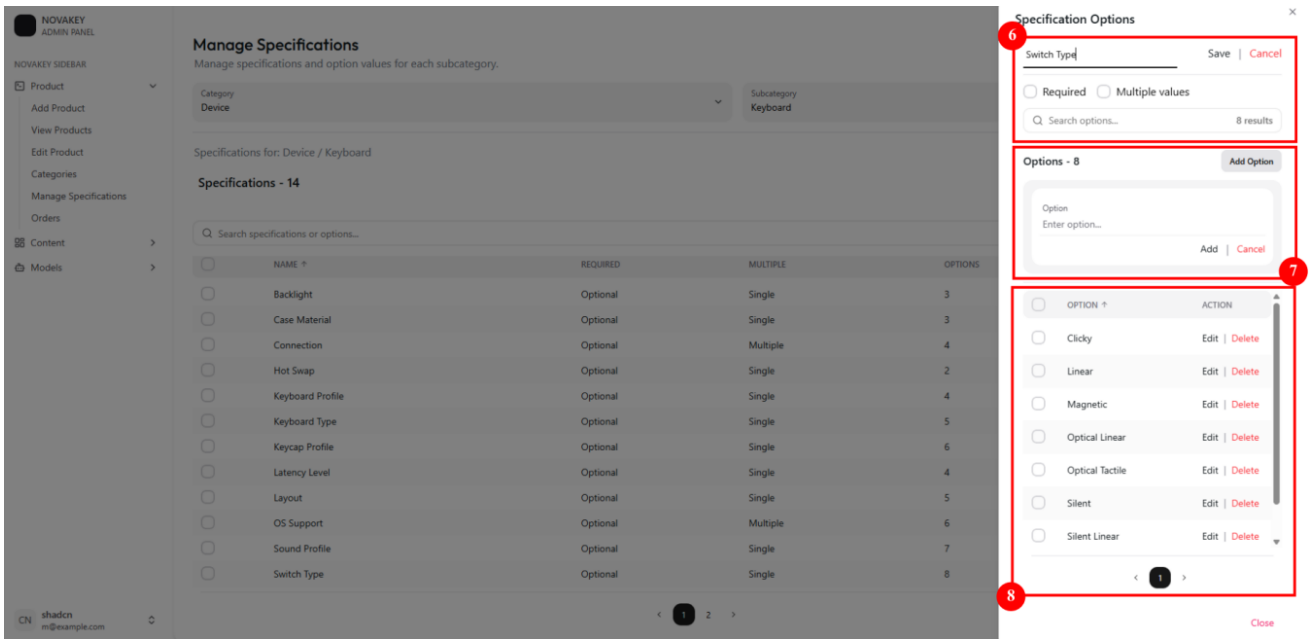

Рисунок 3.24 – Сторінка управління характеристиками. Редагування характеристик

Використання окремого модуля керування характеристиками забезпечує гнучкість системи, спрощує масштабування каталогу товарів та дозволяє повторно використовувати характеристики в різних товарах без дублювання даних.

Сторінка керування контентом головної сторінки та каталогу призначена для адміністрування візуального та інформаційного наповнення клієнтської частини вебзастосунку. Даний модуль дозволяє централізовано керувати банерами, секціями каталогу, інформаційними блоками, каруселями товарів та іншими елементами інтерфейсу без необхідності внесення змін до програмного коду. Реалізація даного функціоналу забезпечує гнучке налаштування структури головної сторінки та сторінки каталогу, що дозволяє адаптувати відображення контенту відповідно до маркетингових потреб інтернет-магазину. Адміністратор має можливість змінювати порядок відображення блоків, налаштовувати їх зовнішній вигляд, кольори, розміри та тип контенту. На рисунку 3.25 можна побачити елементи 1–5, які відповідають за керування контентом головної сторінки та каталогу.

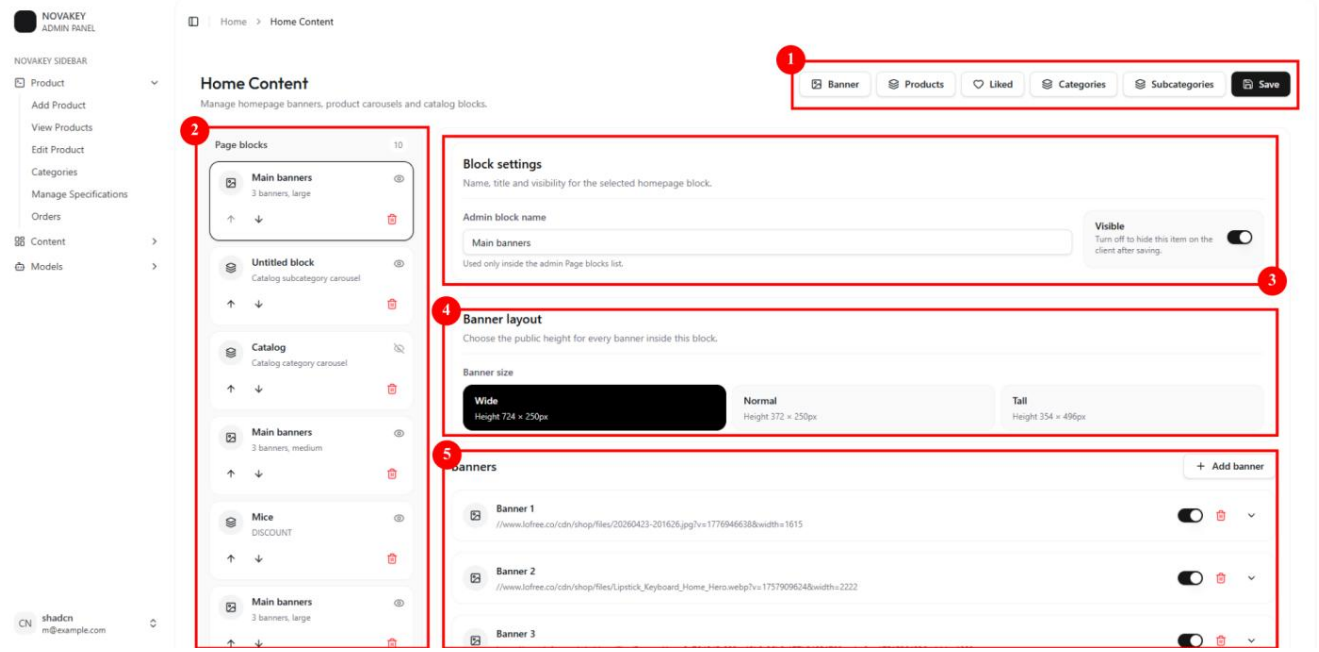

Рисунок 3.25 – Сторінка управління контентом клієнтської частини. Налаштування банерів

Елемент 1 являє собою набір кнопок для додавання різних типів контенту, зокрема банерів, каруселей товарів та інформаційних блоків. Також реалізовано можливість створення динамічних каруселей товарів за різними критеріями, наприклад за популярністю, новизною або наявністю знижок. Елемент 2 використовується для керування порядком відображення контенту на клієнтській частині вебзастосунку. Даний блок підтримує зміну позицій контенту, приховування окремих блоків та їх видалення. Це дозволяє адміністратору гнучко налаштовувати структуру сторінки без необхідності редагування програмного коду.

Елементи 3–5 призначені для детального налаштування окремого типу контенту, зокрема банерів. Елемент 3 містить поля для введення основної інформації, такої як назва, опис, кнопки переходу та інші параметри банера. Елемент 4 використовується для вибору розміру банера та налаштування його відображення на сторінці. Елемент 5 являє собою область керування самим контентом банера, якщо відкрити один із банерів, можна побачити додаткові

налаштування, представлені на рисунку 3.26 елементами 6 та 7. Дані елементи дозволяють завантажувати зображення з локального пристрою або використовувати посилання на зображення. Також реалізовано окремий шар для накладання текстового контенту поверх банера з можливістю налаштування кольору, стилізації тексту.

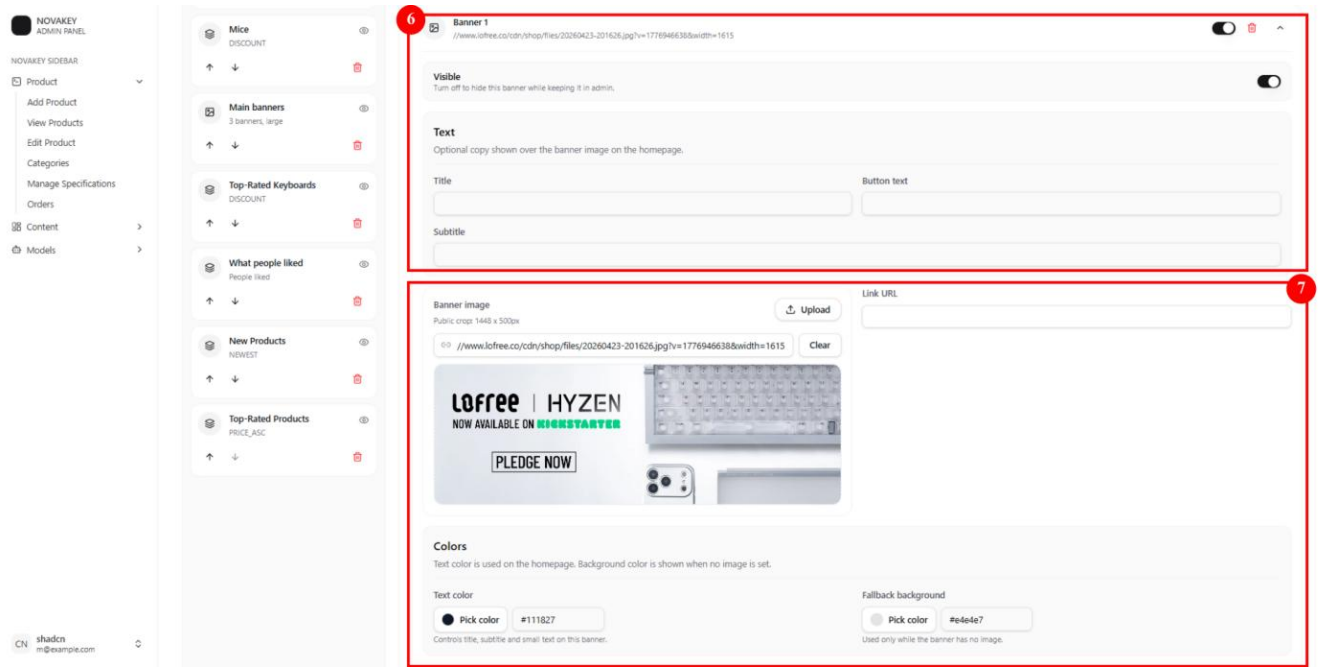

Рисунок 3.26 – Сторінка управління контентом клієнтської частини. Редагування контенту банера

Також, окрім налаштування банерів, реалізовано можливість керування каруселями товарів. На рисунку 3.27 зображено елементи 8 та 9, які відповідають за налаштування даного типу контенту. Елемент 8 призначений для введення основної інформації про карусель товарів, зокрема назви секції, опису та додаткових параметрів відображення. Елемент 9 використовується для детального налаштування логіки формування каруселі. Адміністратор має можливість обрати категорію, підкатегорію або залишити відображення без прив’язки до конкретної категорії. Також реалізовано вибір типу сортування товарів, наприклад за популярністю, наявністю знижок, новизною, ціною від дешевих до дорогих або за

рейтингом товарів. Додатково передбачено налаштування кількості товарів, які будуть відображатися в каруселі на клієнтській частині вебзастосунку.

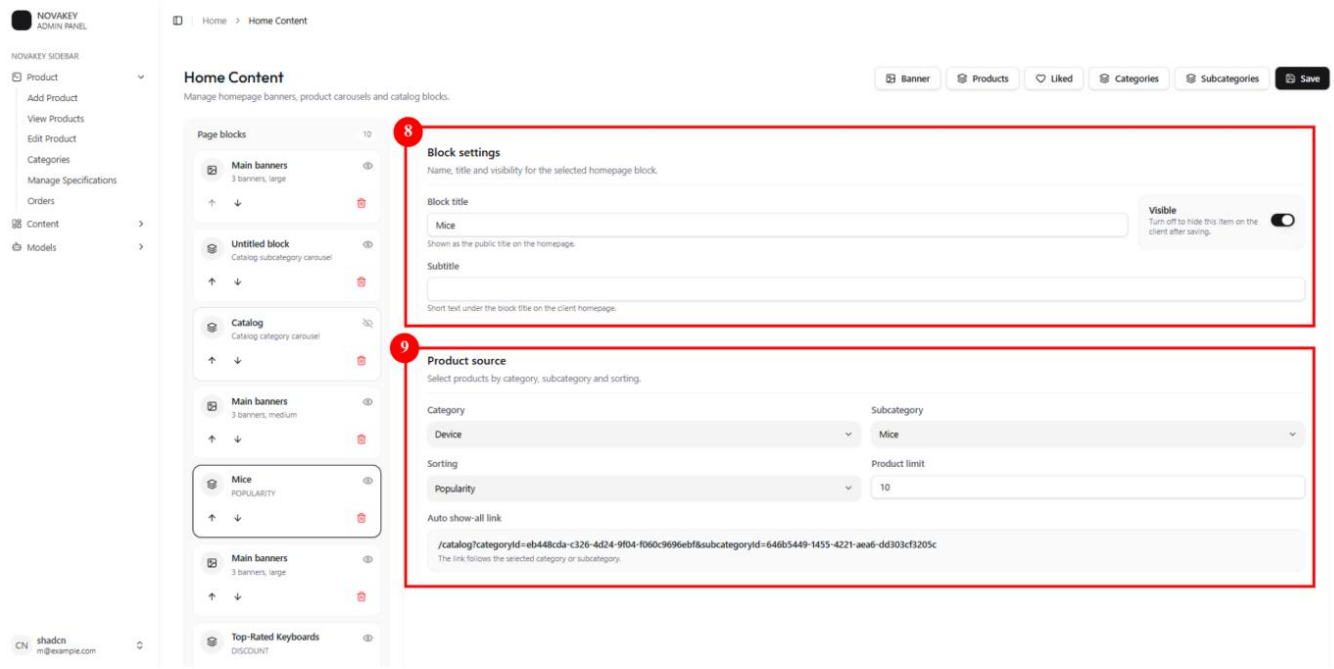

Рисунок 3.27 – Сторінка управління контентом клієнтської частини. Налаштування каруселі товарів

Таким чином, реалізований модуль керування контентом дозволяє гнучко налаштовувати головну сторінку та сторінку каталогу інтернет-магазину без необхідності внесення змін до програмного коду. Адміністратор може самостійно керувати банерами, каруселями товарів, порядком відображення контенту та параметрами його стилізації. Це значно спрощує процес адміністрування вебзастосунку, підвищує швидкість оновлення контенту та забезпечує адаптацію інтерфейсу до маркетингових потреб інтернет-магазину.

Сторінка керування замовленнями призначена для централізованого перегляду та адміністрування замовлень користувачів інтернет-магазину. Даний модуль дозволяє адміністратору контролювати процес обробки замовлень, змінювати їх статуси, переглядати інформацію про товари, доставку та платіжні дані. Інтерфейс сторінки реалізований у вигляді функціональної таблиці з підтримкою пошуку, фільтрації та детального перегляду кожного замовлення. На

рисунку 3.28 можна побачити основні елементи сторінки керування замовленнями. Елемент 1 являє собою панель швидкого пошуку та фільтрації замовлень. Даний блок дозволяє знаходити замовлення за номером, ім’ям користувача, електронною поштою, номером телефону або іншими параметрами. Також реалізовано фільтрацію замовлень за статусом обробки, оплатою та доставкою. У правій частині блоку відображається коротка статистика кількості замовлень за різними статусами. Елемент 2 представлений у вигляді інтерактивної таблиці замовлень, у якій відображається основна інформація про кожне замовлення: номер замовлення, дані клієнта, дата створення, адміністратор, спосіб оплати, статус оплати, загальна сума, тип доставки та статус доставки. Також у таблиці реалізовано функції розкриття детальної інформації про замовлення та сортування.

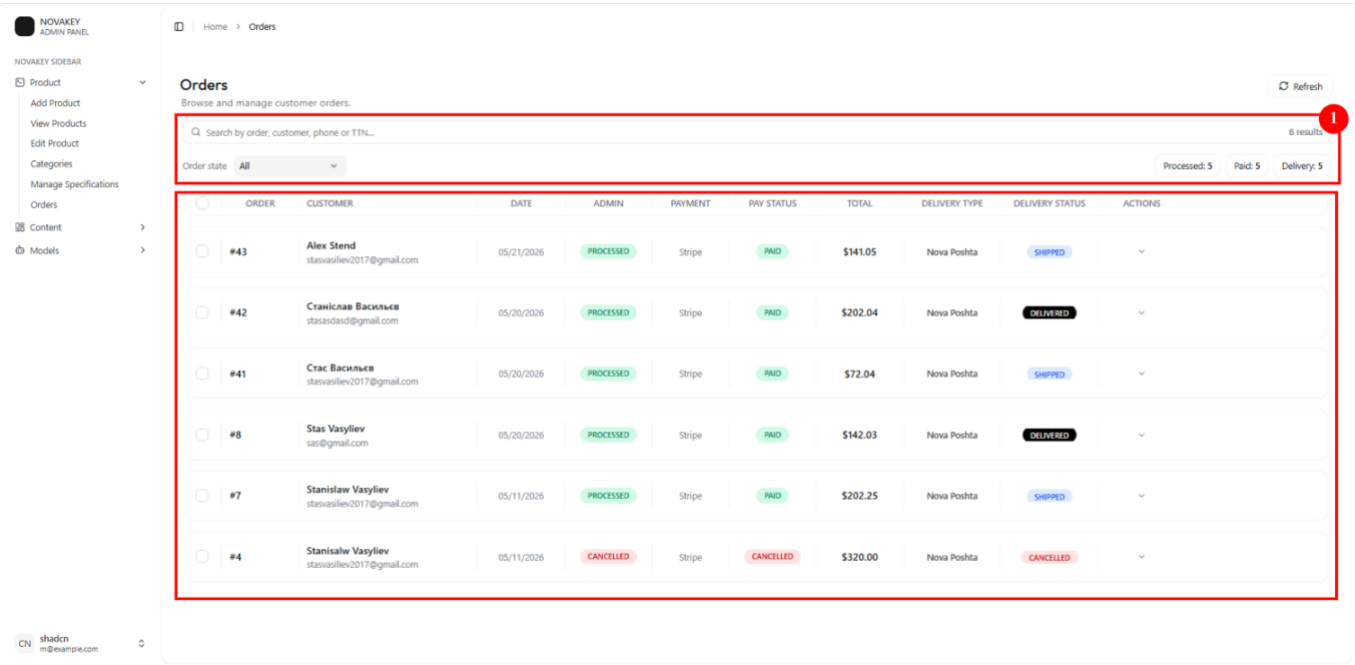

Рисунок 3.28 – Сторінка управління замовленнями. Список замовлень

На рисунку 3.29 можна побачити детальний перегляд замовлення. Елемент 3 містить список товарів, які входять до замовлення, із зображенням товару, назви, кількості та вартості. Елемент 4 використовується для перегляду та редагування інформації про клієнта, зокрема імені, електронної пошти, номера телефону, міста та адреси доставки. Елемент 5 призначений для керування параметрами

замовлення. Адміністратор може змінювати статус обробки замовлення, статус оплати, статус доставки, спосіб доставки та платіжний метод. Також реалізовано можливість додавання автоматично номер ТТН для служби доставки Nova Poshta, коментарів до замовлення та переходу до функцій відстеження доставки. Це дозволяє забезпечити повний цикл адміністрування замовлення без необхідності використання сторонніх сервісів .

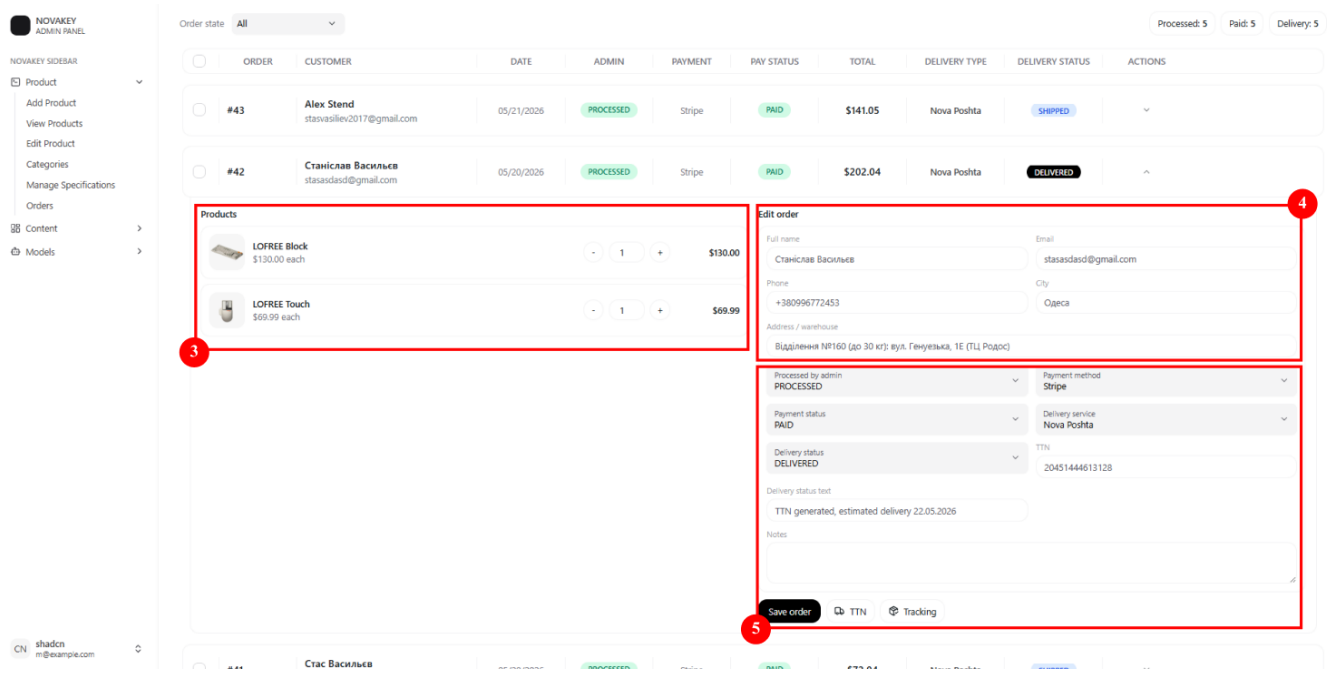

Рисунок 3.29 – Сторінка управління замовленнями. Детальна інформація про замовлення

На рисунку 3.30 представлено підтвердження інтеграції вебзастосунку зі службою доставки Nova Poshta. На зображенні можна побачити створені товарно-транспортні накладні (ТТН), які були автоматично сформовані через API Nova Poshta після оформлення замовлення в адміністративній панелі. Даний функціонал дозволяє адміністратору автоматизувати процес створення доставки без необхідності ручного введення даних у сторонніх сервісах. Після створення ТТН номер накладної зберігається в системі та відображається на сторінці керування замовленнями. Також реалізовано можливість подальшого відстеження статусу доставки безпосередньо через адміністративну панель вебзастосунку.

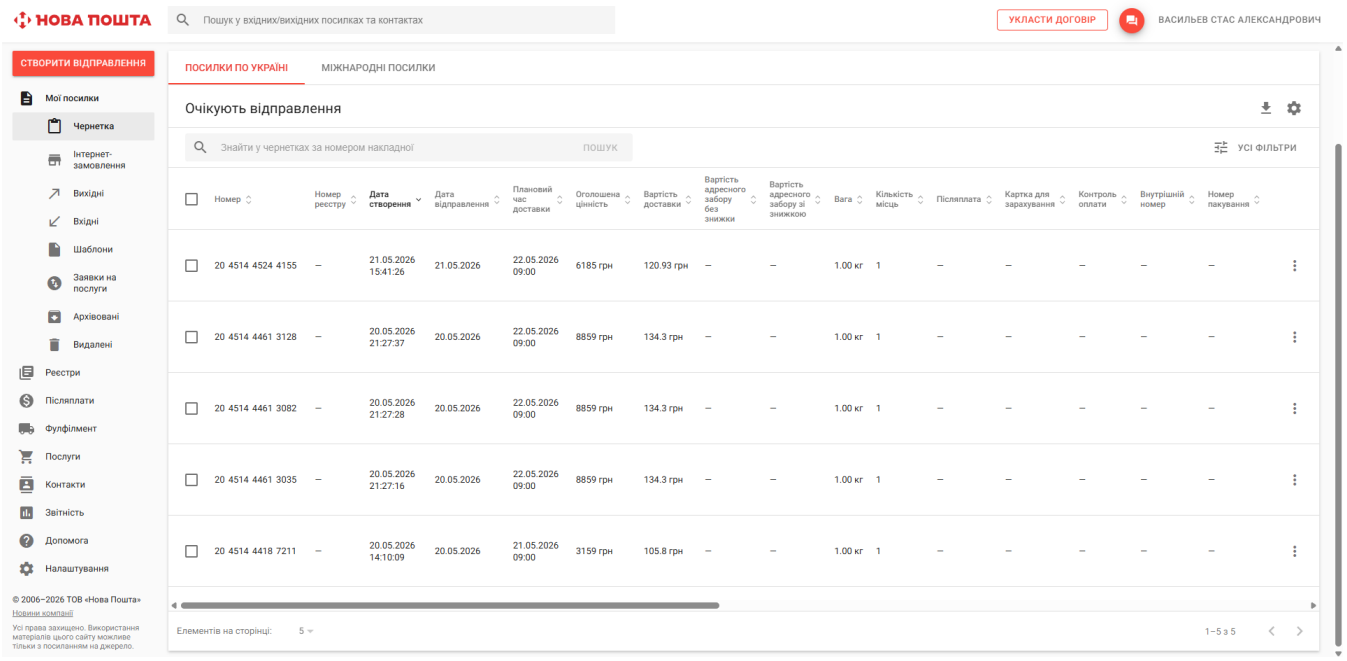

Рисунок 3.30 – Перегляд сформованої товарно-транспортної накладної (ТТН) у сервісі Нова пошта

Таким чином, реалізований модуль керування замовленнями забезпечує повний цикл адміністрування процесу оформлення та доставки товарів в інтернет-магазині. Адміністратор має можливість централізовано переглядати замовлення, змінювати їх статуси, редагувати інформацію про клієнта, контролювати оплату та доставку, а також автоматично створювати товарно-транспортні накладні через інтеграцію з сервісом Nova Poshta. Це значно спрощує процес обробки замовлень, зменшує кількість ручних операцій та підвищує ефективність роботи адміністративної частини вебзастосунку.

Сторінка аналітики призначена для моніторингу основних показників роботи інтернет-магазину та візуального аналізу бізнес-процесів у реальному часі. Даний модуль дозволяє адміністратору отримувати статистичну інформацію про продажі, замовлення, товари, складські залишки та ефективність роботи магазину без необхідності використання сторонніх аналітичних сервісів.

На рисунку 3.31 представлено головну сторінку аналітики адміністративної панелі. У верхній частині сторінки розташовані інформаційні картки з основними

показниками системи. Вони відображають загальний дохід магазину, кількість створених замовлень, кількість товарів у каталозі та інформацію про товари з низьким залишком на складі. Також реалізовано графічне відображення змін показників за певний період часу.

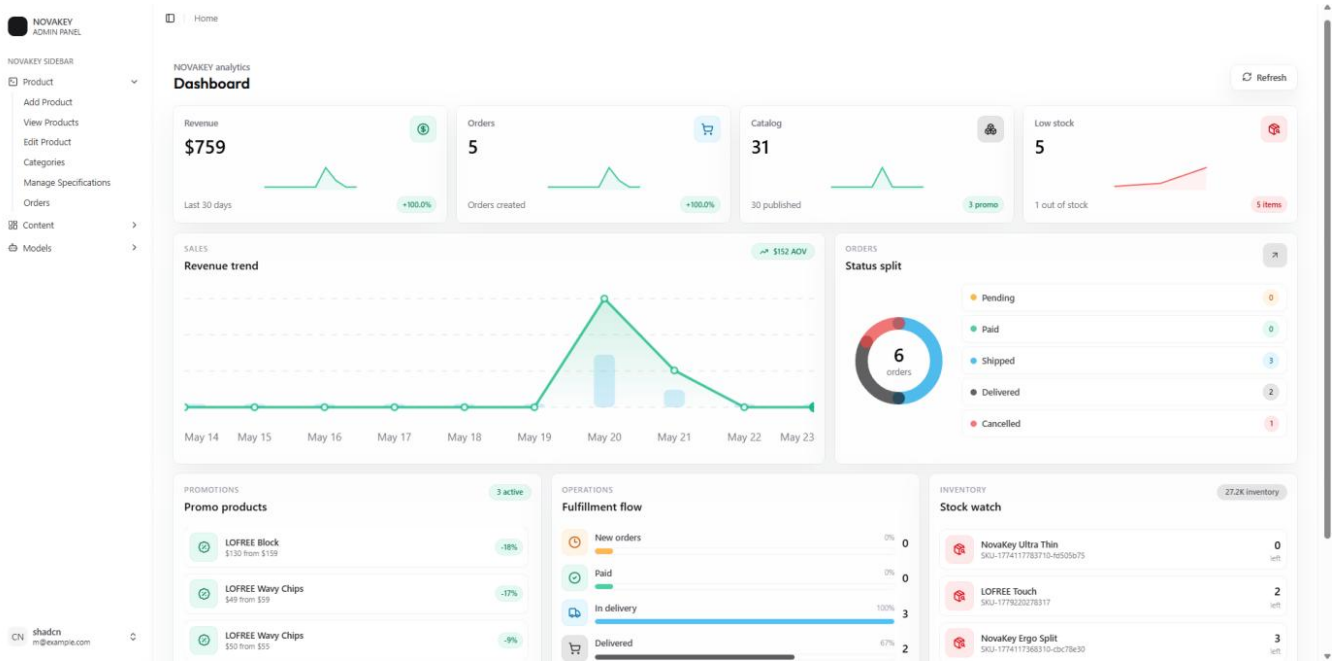

Рисунок 3.31 – Сторінка аналітики

Центральна частина сторінки містить графік динаміки продажів, який дозволяє аналізувати зміну доходу залежно від дати. Поруч розташовано блок статистики статусів замовлень, який відображає співвідношення замовлень за різними станами, наприклад очікування обробки, оплачена, відправлено, доставлено та скасовано. Це дозволяє швидко оцінювати стан обробки замовлень у системі. У нижній частині сторінки розміщено додаткові аналітичні блоки. Один із них відображає товари, які беруть участь у промоакціях та мають активні знижки. Інший блок демонструє етапи обробки замовлень, включаючи нові замовлення, оплату, доставку та завершені покупки. Також реалізовано окремий блок моніторингу складських залишків, який дозволяє контролювати кількість товарів на складі та своєчасно виявляти товари, що закінчуються.

Таким чином, реалізований модуль аналітики забезпечує централізований контроль основних бізнес-процесів інтернет-магазину, спрощує аналіз статистичних даних та дозволяє оперативно приймати рішення щодо керування товарами, замовленнями та маркетинговими активностями.

3.3 Реалізація клієнтської частини вебзастосунку

Клієнтська частина вебзастосунку призначена для взаємодії користувача з інтернет-магазином, перегляду каталогу товарів, оформлення замовлень та використання додаткового функціоналу системи. Основною метою клієнтської частини є забезпечення зручного, швидкого та інтуїтивно зрозумілого інтерфейсу для користувачів. Інтерфейс клієнтської частини реалізовано у сучасному стилі з адаптивним дизайно [17-18]. Основна структура вебзастосунку складається з головної сторінки, каталогу товарів, сторінки окремого товару, кошика, оформлення замовлення, особистого кабінету користувача та ШІ-асистента для підтримки користувачів. На головній сторінці реалізовано систему динамічного контенту, яка дозволяє відображати банери, інформаційні блоки та каруселі товарів залежно від налаштувань адміністративної панелі. Для сторінки каталогу реалізовано функціонал пошуку, фільтрації та сортування товарів за різними параметрами, такими як категорія, ціна, популярність, новизна та наявність знижок. Основні елементи головної сторінки та сторінки каталогу наведено на рисунках 3.32 – 3.34.

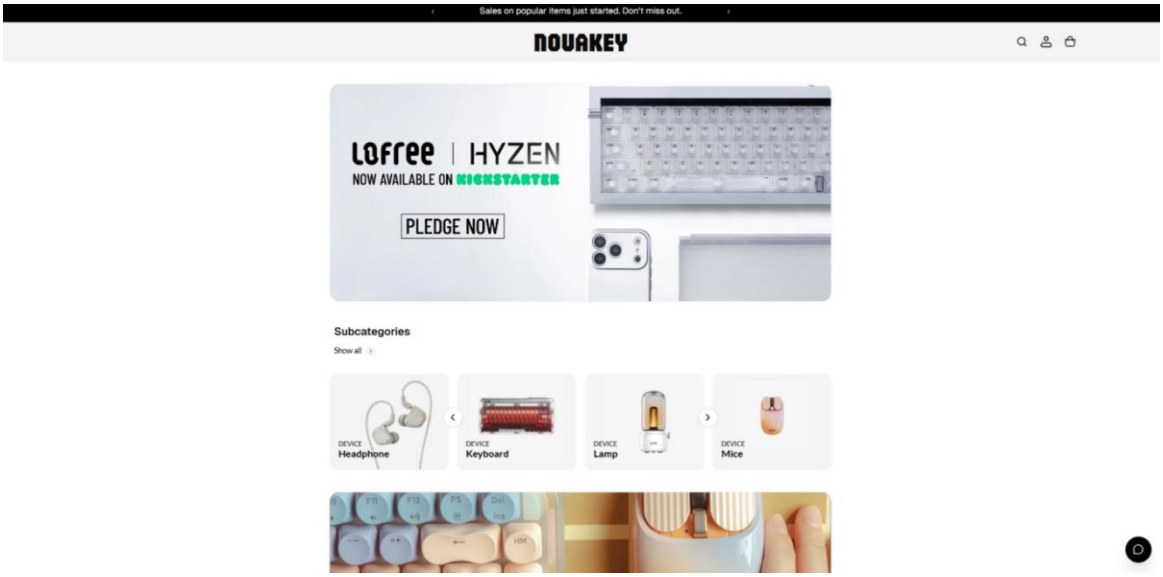

Рисунок 3.32 – Головна сторінка клієнтського застосунку. Верхня частина

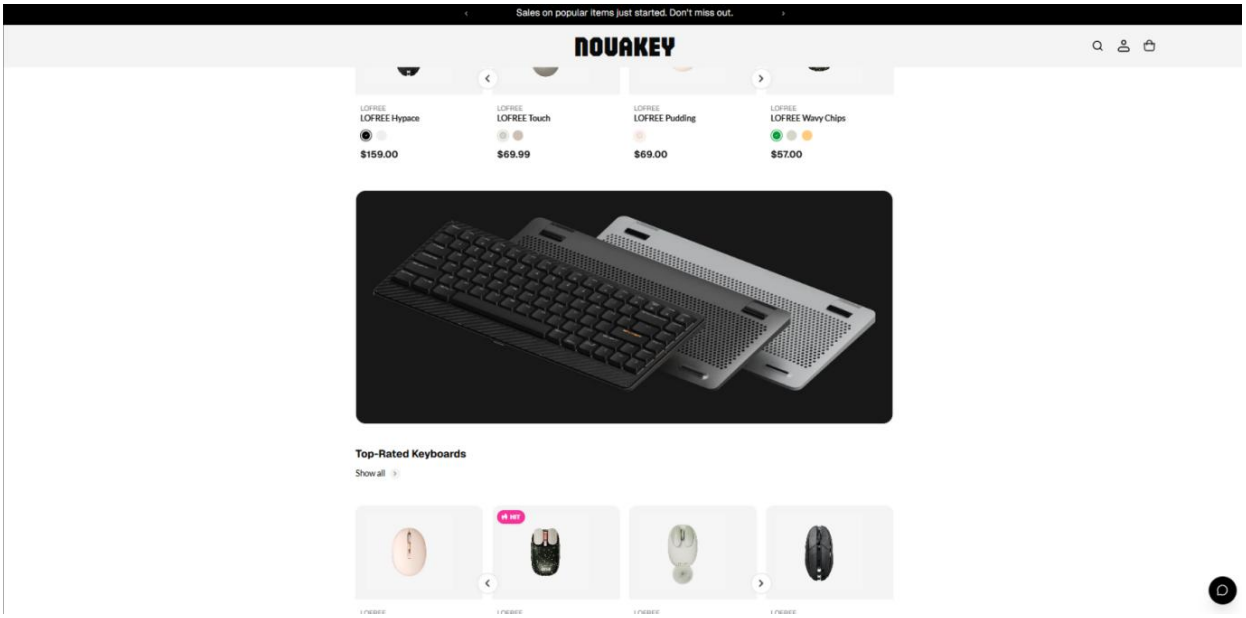

Рисунок 3.33 – Головна сторінка клієнтського застосунку. Контентні блоки

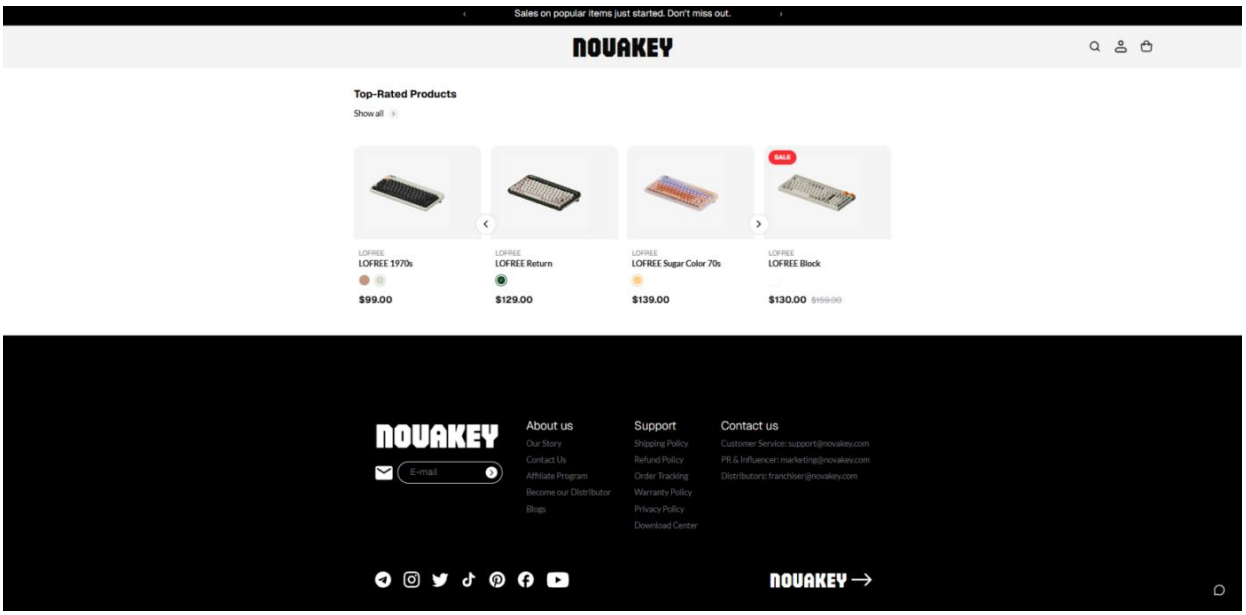

Рисунок 3.34 – Головна сторінка клієнтського застосунку. Нижня частина та footer

На рисунках 3.32 – 3.34 можна побачити основні елементи інтерфейсу клієнтської частини вебзастосунку, зокрема хедер та футер, які формують основний layout системи. Дані елементи забезпечують навігацію користувача між сторінками вебзастосунку та надають доступ до основного функціоналу інтернет-магазину.

Хедер розташований у верхній частині сторінки та містить логотип вебзастосунку, навігаційне меню, пошукову систему, кнопки переходу до каталогу, обраних товарів, кошика та особистого кабінету користувача. Футер розташований у нижній частині вебзастосунку та містить додаткову інформацію про інтернет-магазин, навігаційні посилання, контакти, інформацію про доставку та оплату, а також посилання на соціальні мережі. Даний елемент забезпечує швидкий доступ до допоміжної інформації та покращує зручність користування вебзастосунком.

Реалізація layout структури дозволяє забезпечити єдиний стиль інтерфейсу для всіх сторінок клієнтської частини вебзастосунку, спрощує навігацію та покращує взаємодію користувача з системою.

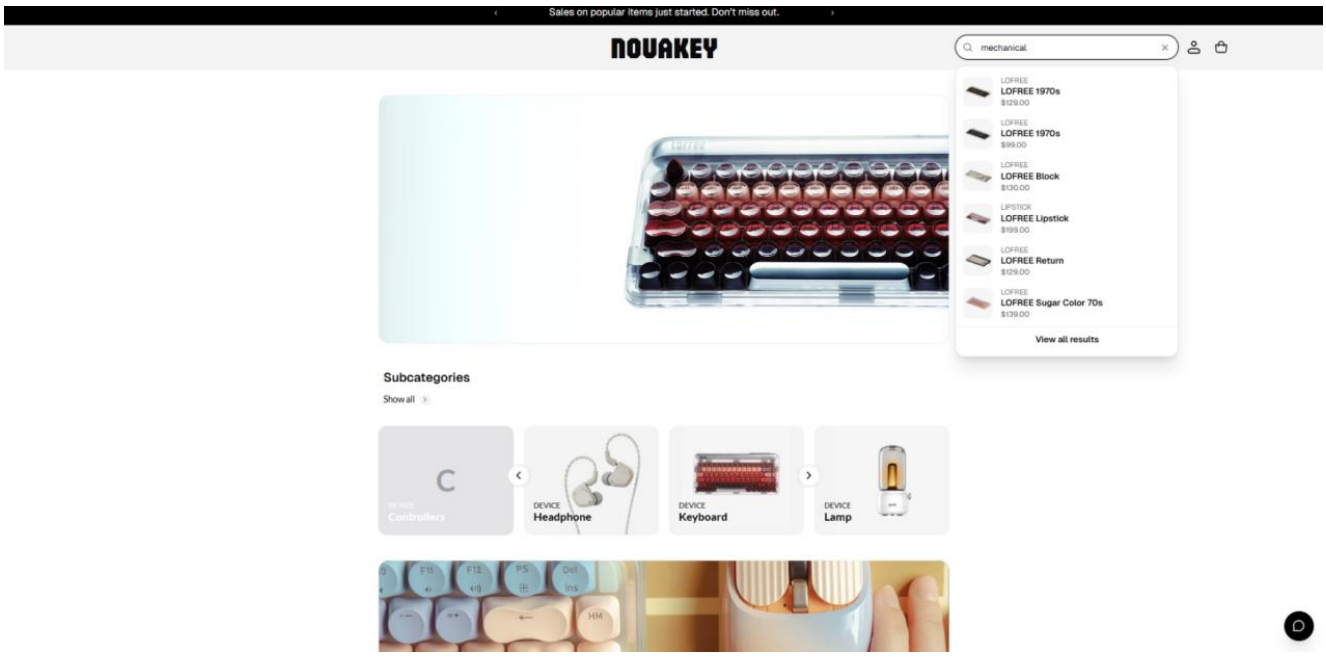

Рисунок 3.35 – Відображення функціоналу пошуку

Також на рисунку 3.35 можна побачити додаткові елементи інтерфейсу, зокрема систему гібридного пошуку, профіль користувача та кошик покупок. Дані елементи забезпечують швидку взаємодію користувача з основним функціоналом вебзастосунку. Гібридний пошук реалізовано для швидкого пошуку товарів за різними параметрами, такими як назва товару, категорія, характеристики або ключові слова. Даний функціонал поєднує класичний пошук із системою інтелектуального підбору результатів [3, 25, 31], що дозволяє підвищити точність та швидкість знаходження товарів у каталозі.

Елемент профілю користувача надає доступ до особистого кабінету, де користувач може переглядати інформацію про власні замовлення, обрані товари, налаштування облікового запису та інші персональні дані. Елемент кошика використовується для перегляду доданих товарів, зміни їх кількості та переходу до оформлення замовлення, можно подивитися на Рисунку 3.36. Також у кошику відображається підсумкова інформація про вартість товарів та активні знижки.

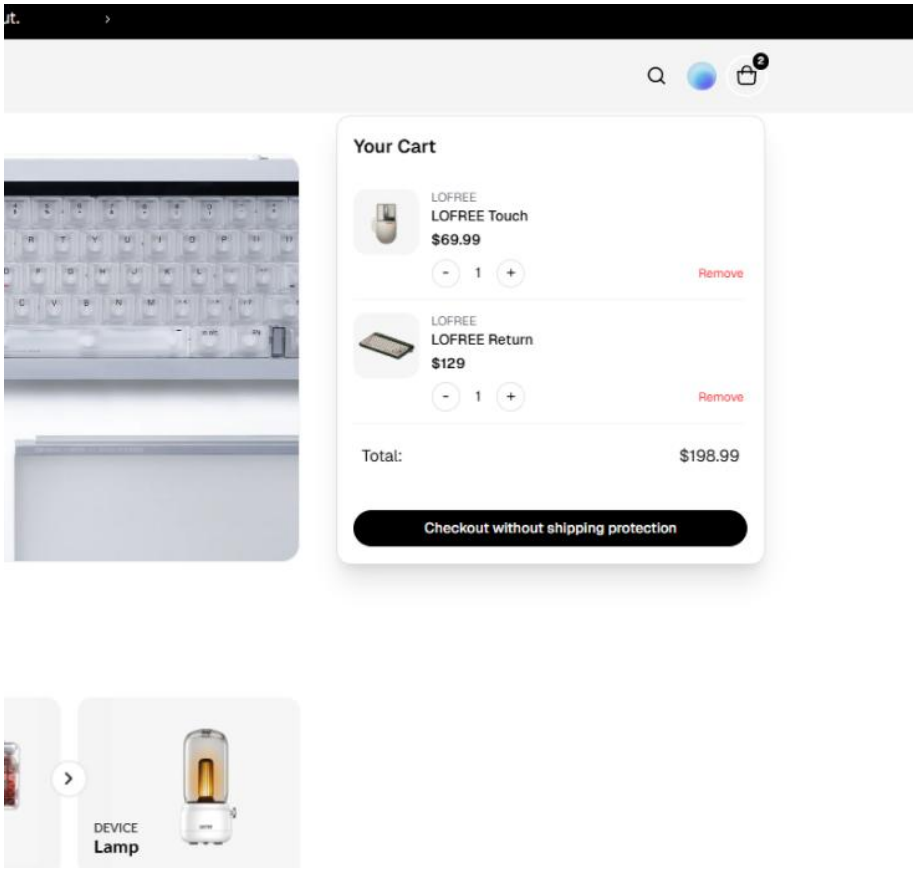

Рисунок 3.36 – Відображення функціоналу кошика

На рисунку 3.37 представлено сторінку каталогу клієнтської частини вебзастосунку, яка використовується для навігації між категоріями та підкатегоріями товарів. Даний модуль дозволяє користувачу швидко обирати потрібний тип продукції та переходити до відповідного списку товарів. У центральній частині сторінки відображаються основні категорії товарів. Після вибору певної категорії користувач переходить до сторінки підкатегорій, де може обрати більш конкретний тип товару. Такий підхід забезпечує багаторівневу структуру каталогу та спрощує навігацію у великій кількості товарних позицій.

Після вибору підкатегорії відкривається сторінка зі списком товарів даної підкатегорії. Для зручності користувача реалізовано систему фільтрації та сортування товарів за різними параметрами, такими як ціна, популярність, новизна, рейтинг та наявність акційних пропозицій. Це дозволяє швидко знаходити необхідні товари серед великої кількості позицій каталогу.

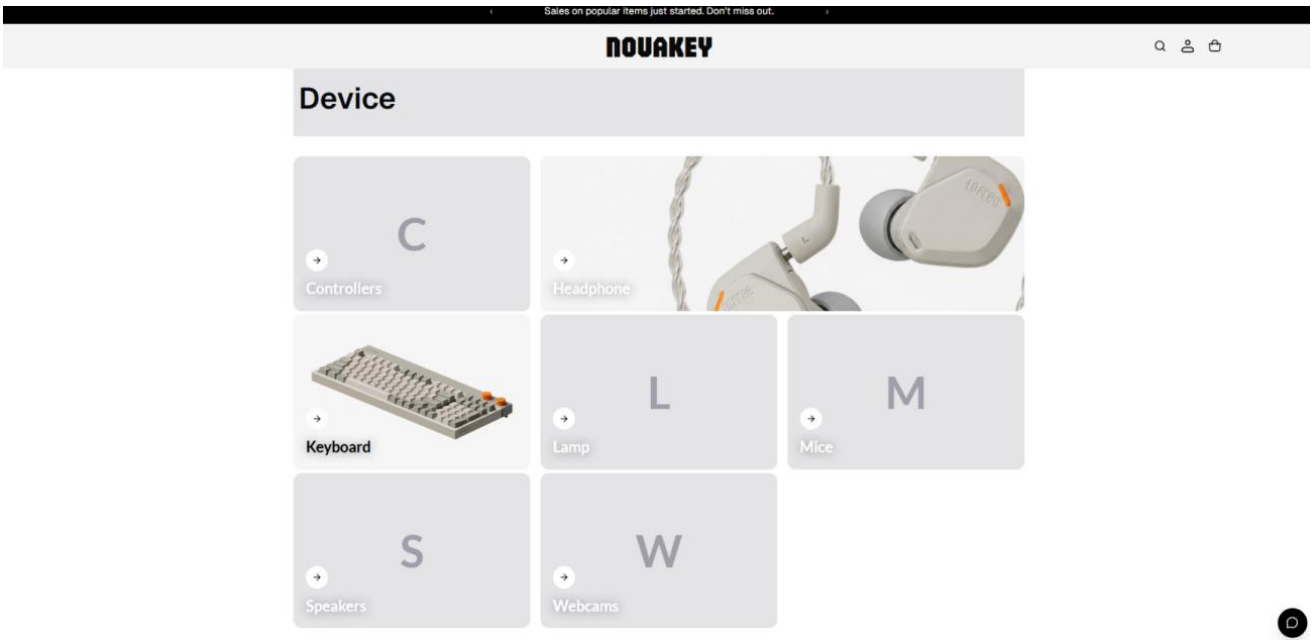

Рисунок 3.37 – Сторінка категорій

Центральна частина сторінки містить список товарів у вигляді адаптивної сітки. Для кожного товару відображається основна інформація, зокрема зображення та назва. Також реалізовано функції швидкого переходу до сторінки товару, додавання товару до кошика та списку обраного. Додатково сторінка каталогу підкатегорії підтримує систему фільтрації та сортування товарів за різними параметрами, такими як ціна, популярність, новизна та наявність акційних пропозицій. Це дозволяє користувачу швидко знаходити необхідні товари серед великої кількості позицій у каталозі.

Список товарів реалізовано у вигляді адаптивної сітки, де для кожного товару відображається основна інформація, зокрема зображення, назва, ціна, статус товару та наявність знижки. Також реалізовано функції швидкого переходу до сторінки товару, додавання товару до кошика та списку обраного можно переглянути на рисунку 3.38.

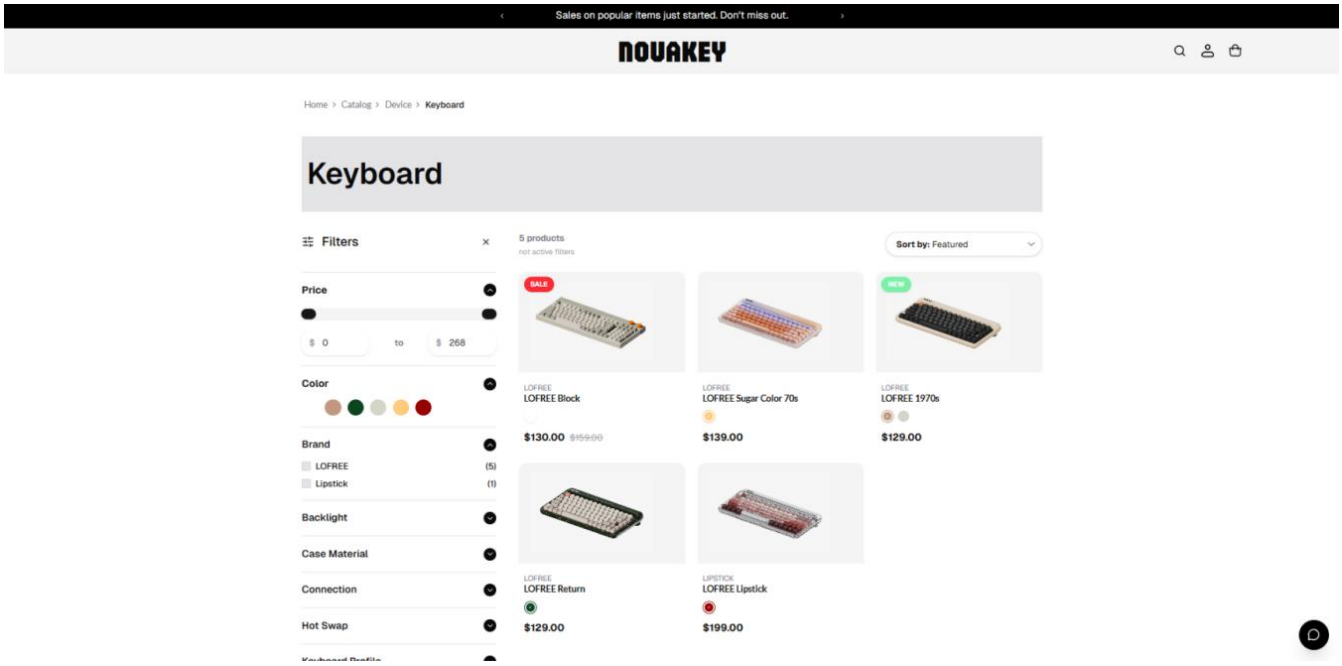

Рисунок 3.38 – Сторінка підгатегорії

На рисунках 3.39 – 3.40 представлено реалізацію системи вибору кольорів товару в клієнтській частині вебзастосунку. Даний функціонал дозволяє користувачу переглядати різні варіанти кольору товару без необхідності переходу на окрему сторінку продукту. У картках товарів реалізовано інтерактивні кольорові перемикачі, за допомогою яких користувач може змінювати варіант товару безпосередньо в каталозі. Після вибору іншого кольору система в режимі реального часу оновлює головне зображення товару, ціну, статуси та іншу інформацію залежно від обраного варіанта. Це дозволяє значно покращити взаємодію користувача з каталогом товарів та спрощує процес перегляду різних модифікацій продукту. Аналогічний функціонал реалізовано і на сторінці окремого товару, де користувач може детальніше переглядати доступні кольори та варіанти продукту. Завдяки динамічному оновленню контенту забезпечується більш зручний та сучасний користувацький інтерфейс без необхідності перезавантаження сторінки.

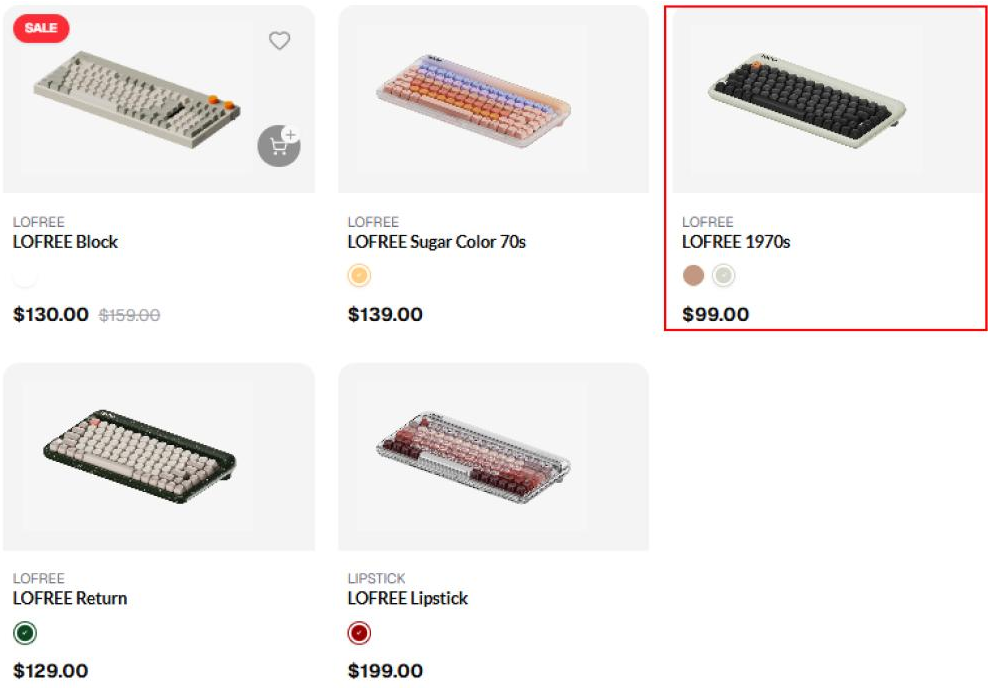

Рисунок 3.39 – Зміна картки товару при виборі кольору (до)

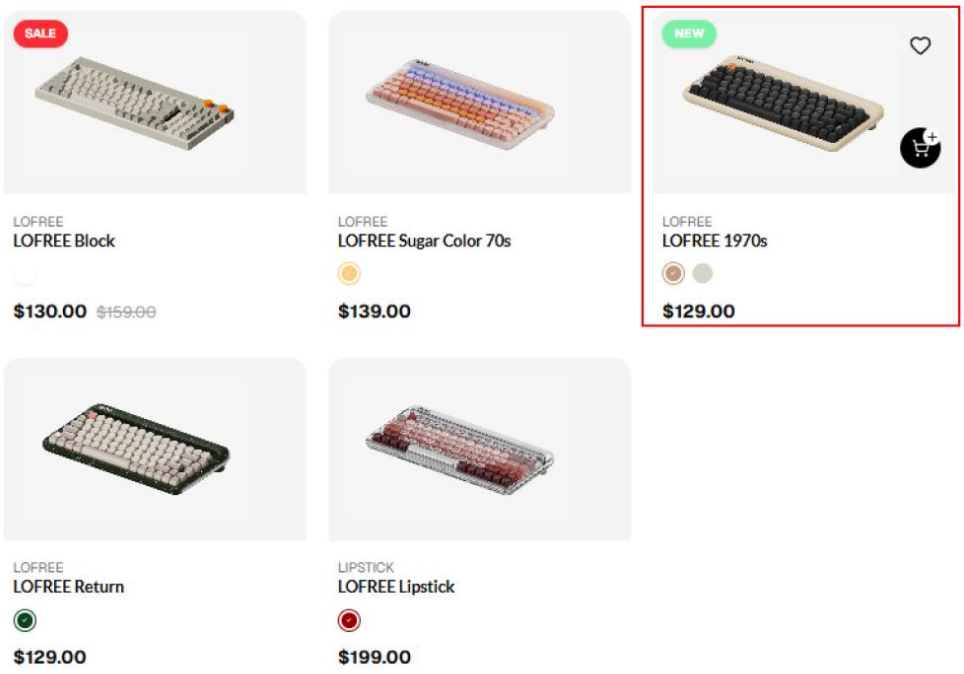

Рисунок 3.40 – Зміна картки товару при виборі кольору (після)

На рисунках 3.41 – 3.43 представлено сторінку товару клієнтської частини вебзастосунку. Дана сторінка призначена для детального перегляду інформації про товар, його характеристик, варіантів, додаткового контенту та оформлення

покупки. У верхній частині сторінки розміщено основну інформацію про товар, зокрема назву, бренд, ціну, статус знижки та систему вибору кольору. Також реалізовано галерею зображень товару з можливістю перегляду декількох фотографій та зміни головного зображення без перезавантаження сторінки.

Користувач може додати товар до кошика, списку обраного або поділитися посиланням на товар. Нижче розташовано блок характеристик товару, який містить технічні параметри продукції, наприклад тип підключення, формат клавіатури, тип перемикачів, підтримку операційних систем та інші характеристики. Також реалізовано окремі інформаційні блоки щодо доставки, повернення товару та перевірки якості продукції.

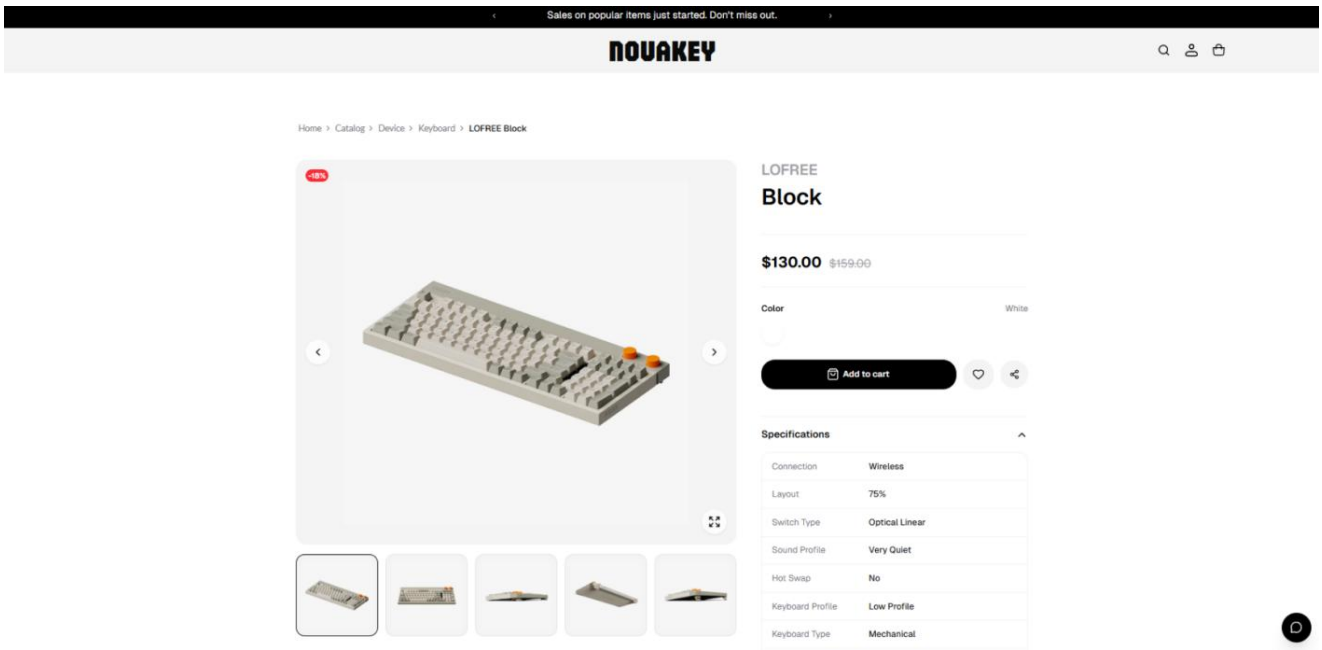

Рисунок 3.41 – Сторінка товару. Основна інформація про товар

Середня частина сторінки містить розширений опис товару та динамічний контент, який формується через адміністративну панель. Даний контент реалізовано у стилі Bento Grid, що дозволяє гнучко компонувати інформаційні блоки різного розміру, текст, зображення та банери. Це забезпечує сучасний вигляд сторінки товару та покращує візуальне сприйняття інформації користувачем.

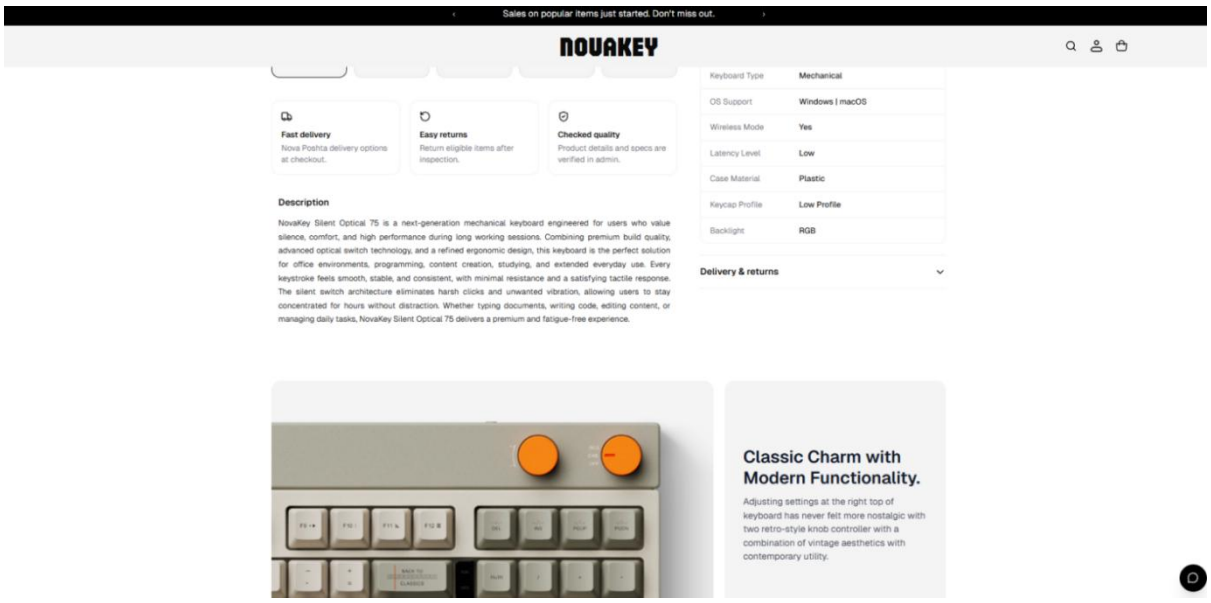

Рисунок 3.42 – Сторінка товару. Розширений опис та Bento Grid

У нижній частині сторінки реалізовано блок рекомендацій схожих товарів, який дозволяє користувачу переглядати інші продукти каталогу без необхідності повернення до сторінки каталогу. Такий підхід покращує навігацію вебзастосунком та підвищує зручність взаємодії користувача з системою.

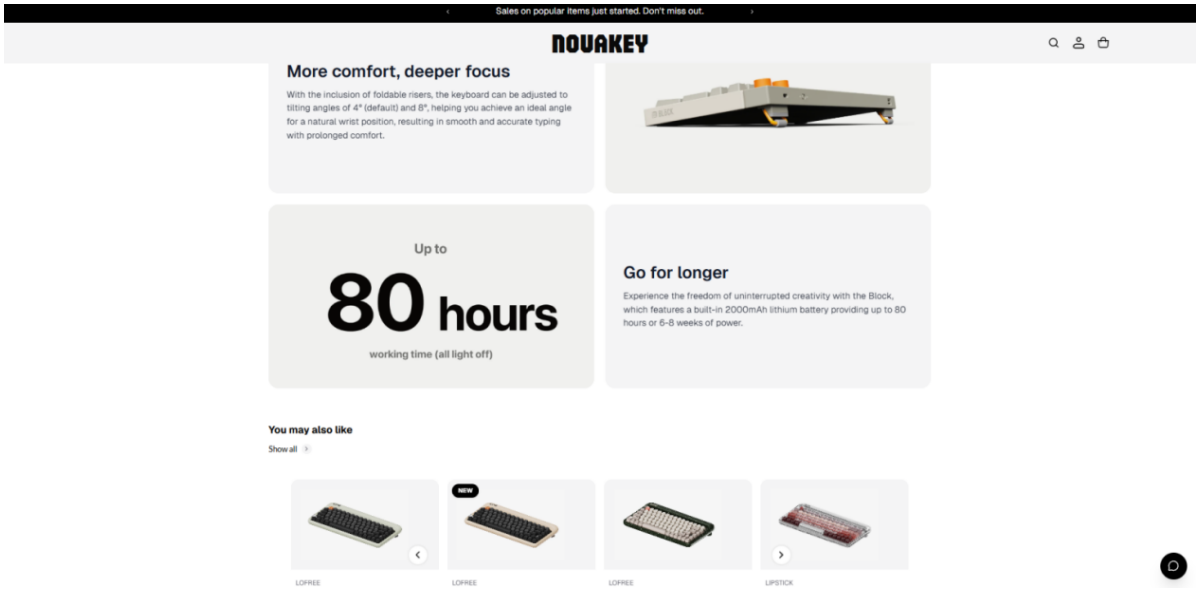

Рисунок 3.43 – Сторінка товару. Рекомендовані товари

На рисунках 3.44 – 3.45 представлено сторінку оформлення замовлення (Checkout) клієнтської частини вебзастосунку. Даний модуль призначений для введення контактної інформації користувача, вибору способу доставки та оплати, а також підтвердження покупки товарів. У верхній частині сторінки реалізовано блок введення контактних даних користувача, який містить поля для електронної пошти, імені, прізвища та номера телефону. Також реалізовано можливість швидкого вибору збережених адрес доставки з профілю користувача, що значно спрощує процес оформлення замовлення для авторизованих користувачів. У правій частині сторінки розташовано блок замовлення, де відображається список доданих товарів, їх кількість, ціна та загальна вартість покупки. Користувач може переглядати підсумкову інформацію про замовлення перед підтвердженням оплати.

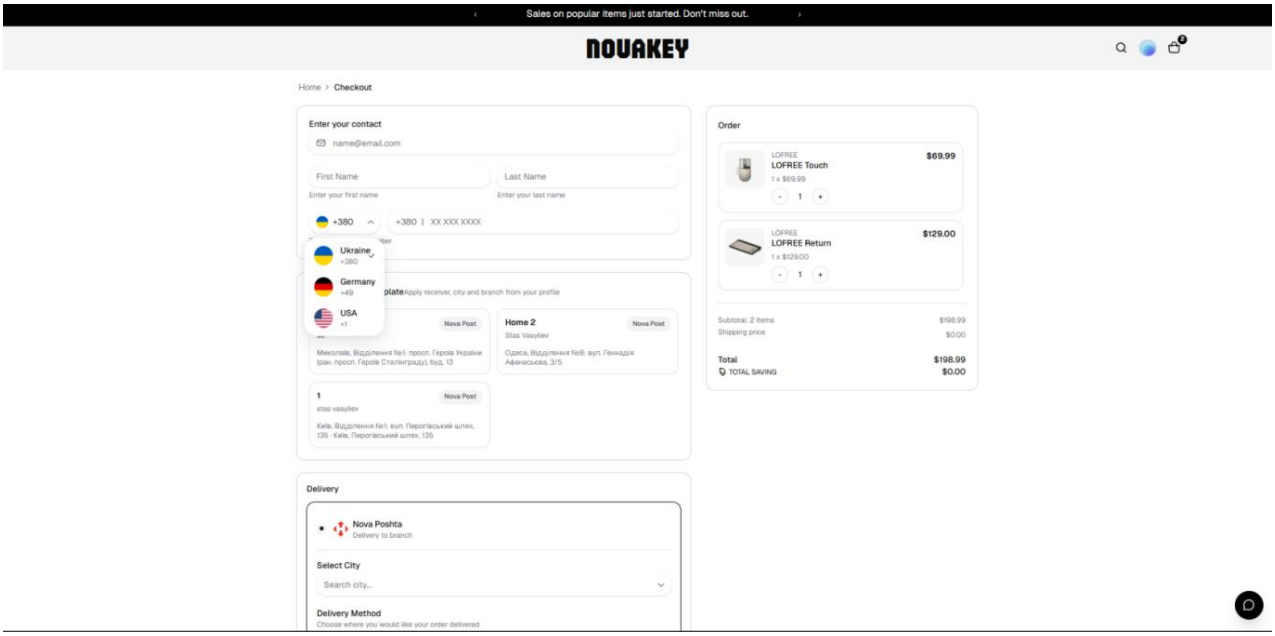

Рисунок 3.44 – Сторінка оформлення замовлення. Контактні дані та склад замовлення

Для доставки товарів реалізовано інтеграцію зі службою Nova Poshta. Користувач має можливість обрати місто доставки, тип доставки, зокрема у відділення або поштомат, а також конкретне відділення Nova Poshta. Також

підтримується можливість доставки кур’єрськими сервісами за адресою користувача. Нижче розташовано блок вибору способу оплати. Система підтримує декілька варіантів оплати, зокрема Privat24, Link та оплату банківською карткою. Після заповнення всіх необхідних даних користувач може підтвердити замовлення за допомогою кнопки оформлення покупки.

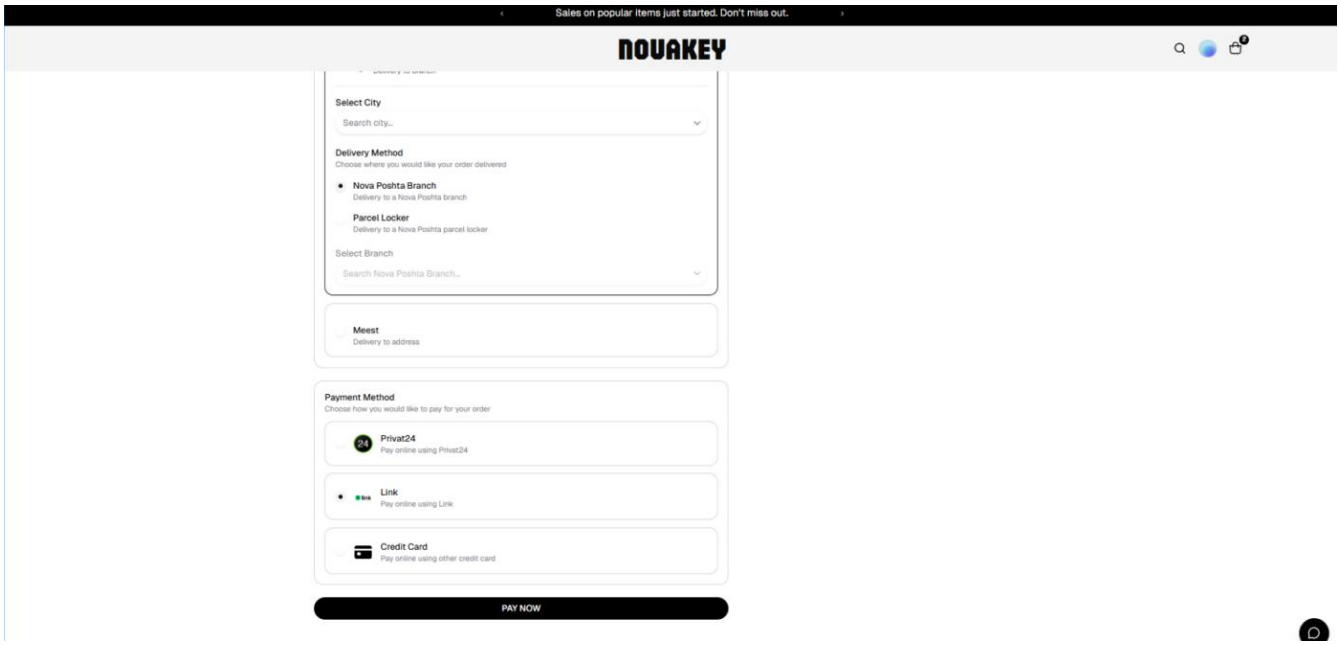

Рисунок 3.45 – Сторінка оформлення замовлення. Доставка та оплата

Таким чином, реалізована сторінка Checkout забезпечує повний цикл оформлення замовлення, поєднуючи введення контактної інформації, інтеграцію зі службами доставки та підтримку декількох способів оплати в межах єдиного інтерфейсу вебзастосунку. Після успішного оформлення замовлення користувача автоматично перенаправляє на сторінку перегляду власних замовлень, приклад якої наведено на рисунку 3.46. Дана сторінка призначена для перегляду історії покупок, контролю статусу доставки та отримання детальної інформації про оформлені замовлення. У верхній частині сторінки відображається коротка статистика користувача, зокрема загальна кількість замовлень, активні замовлення, доставлені товари та загальна сума покупок. Це дозволяє користувачу швидко отримувати інформацію про власну активність у системі.

Основна частина сторінки містить список замовлень із детальною інформацією про кожне з них. Для кожного замовлення відображається номер замовлення, дата створення, статус доставки, спосіб оплати, адреса доставки, номер ТТН та перелік придбаних товарів. Також реалізовано таймлайн статусів замовлення, який дозволяє користувачу відстежувати етапи обробки та доставки покупки. У правій частині сторінки розташовано окремий інформаційний блок із детальним переглядом вибраного замовлення та можливістю швидкого переходу до оформлення або перегляду інформації про доставку.

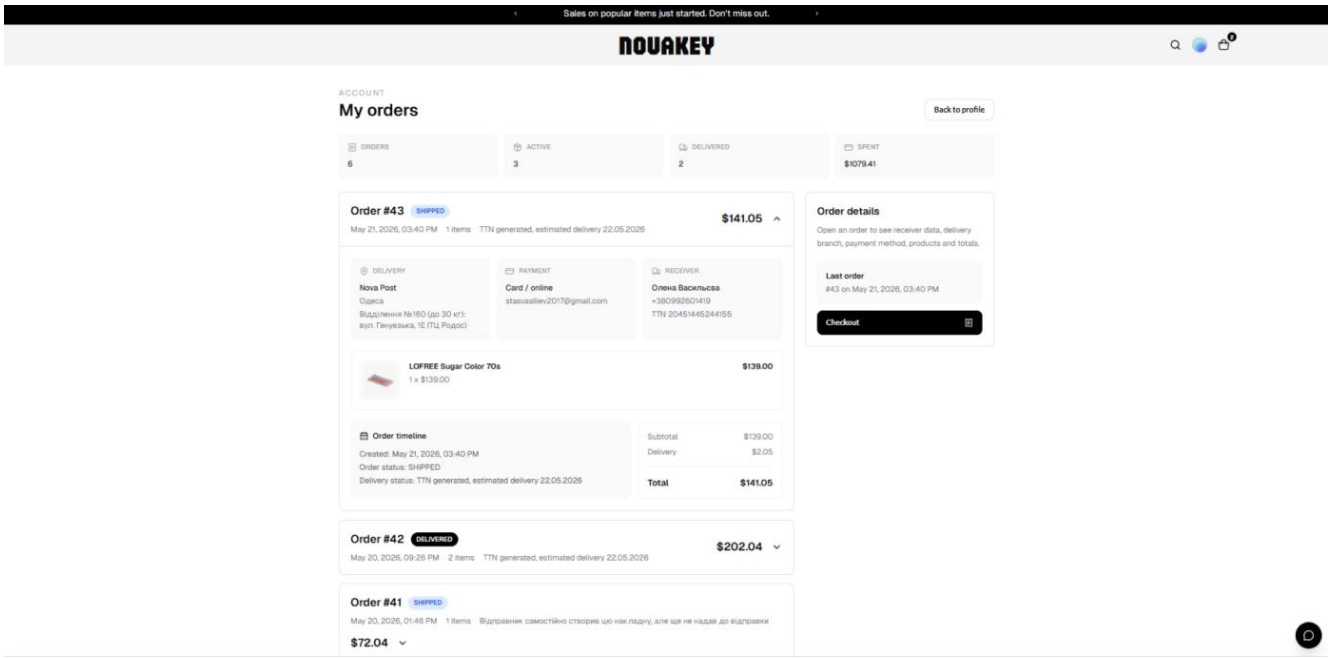

Рисунок 3.46 – Сторінка перегляду замовлень користувача

Таким чином, реалізований модуль перегляду замовлень забезпечує користувачу зручний доступ до історії покупок, інформації про доставку та поточного стану замовлень у межах клієнтської частини вебзастосунку.

На рисунку 3.47 представлено сторінку профілю користувача клієнтської частини вебзастосунку. Даний модуль призначений для централізованого керування персональними даними користувача, перегляду історії замовлень, збережених товарів та адрес доставки.

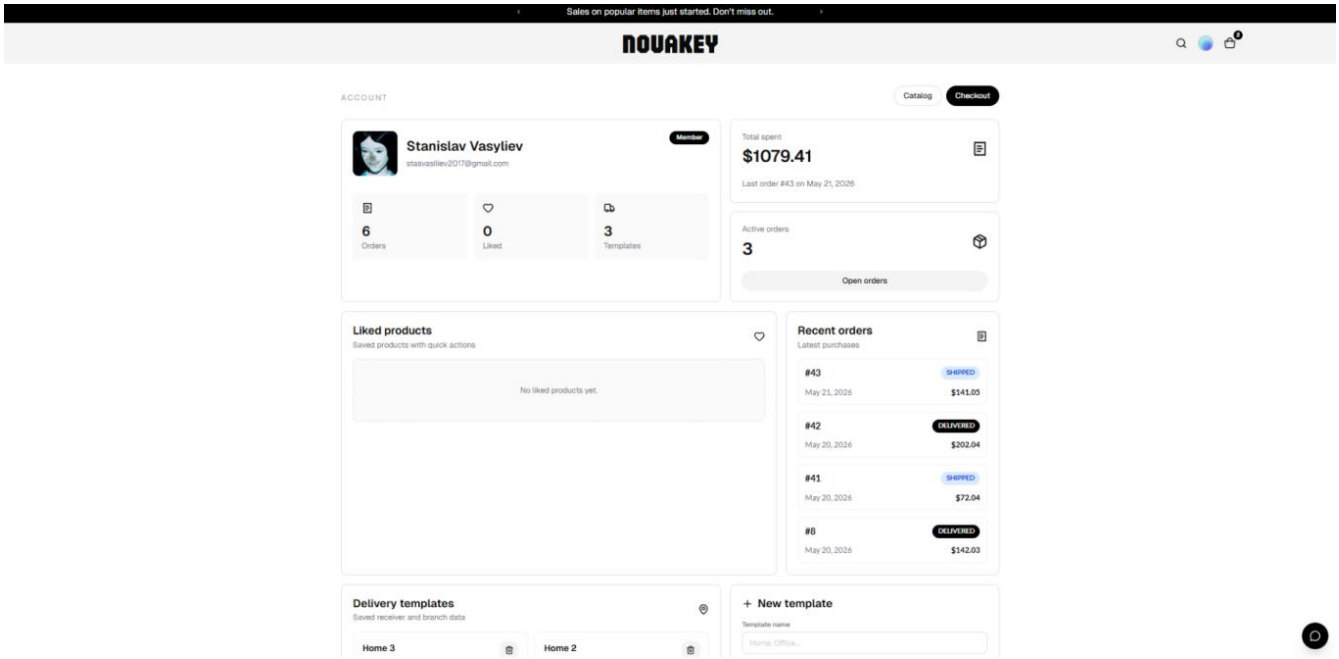

Рисунок 3.47 – Сторінка профилю користовача

У верхній частині сторінки відображається основна інформація про користувача, зокрема ім’я, електронна пошта, аватар профілю та статус облікового запису. Також реалізовано коротку статистику активності користувача, яка містить кількість замовлень, обраних товарів та збережених шаблонів доставки. У правій частині сторінки розташовано інформаційні блоки зі статистикою покупок користувача, зокрема загальна сума витрат, кількість активних замовлень та список останніх оформлених покупок із поточними статусами доставки. Центральна частина сторінки містить блок обраних товарів, у якому користувач може переглядати збережені продукти та швидко переходити до їх оформлення або перегляду сторінки товару. Також реалізовано блок керування адресами доставки, який дозволяє зберігати декілька адрес для швидкого оформлення майбутніх замовлень. Додатково реалізовано функціонал створення шаблонів доставки, що дозволяє користувачу зберігати готові параметри оформлення замовлення та використовувати їх повторно під час наступних покупок.

Таким чином, сторінка профілю забезпечує зручне керування персональними даними користувача, історією покупок та параметрами доставки в межах єдиного інтерфейсу клієнтської частини вебзастосунку.

3.4 Інтеграція ШІ-асистента у вебзастосунок

Інтеграція ШІ-асистента у вебзастосунок є одним із сучасних способів покращення взаємодії користувача із системою та автоматизації підтримки клієнтів. Основною метою впровадження ШІ-асистента є надання користувачу швидких відповідей на типові запити, допомога з навігацією по сайту, консультація щодо товарів, а також підтримка під час оформлення замовлення. На відміну від звичайного пошуку або статичних FAQ-сторінок, ШІ-асистент здатний аналізувати природну мову користувача, враховувати контекст діалогу та формувати адаптивні та персоналізовані відповіді.

У межах розробленого вебзастосунку ШІ-асистент реалізовано як окремий функціональний модуль, що складається з клієнтської частини, серверної логіки, ШІ-моделі, бази знань та системи збереження історії діалогів.

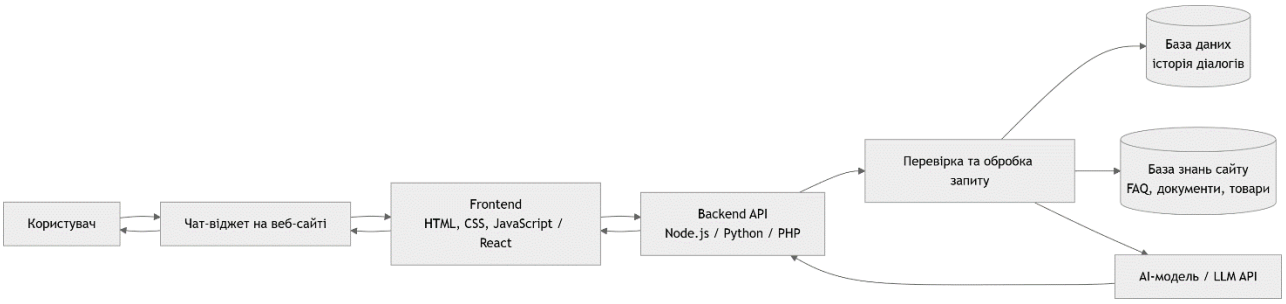

Рисунок 3.48 – Загальна архітектура інтеграції ШІ-асистента

На рисунку 3.48 наведено загальну архітектуру інтеграції ШІ-асистента у вебзастосунок. Користувач взаємодіє із чат-віджетом у вебзастосунку, після чого повідомлення передається до frontend-частини. Frontend надсилає HTTP-запит на backend API, який виконує перевірку та обробку повідомлення. Далі сервер може звертатися до бази знань, історії діалогів та ШІ-моделі для формування відповіді

користувачу.Клієнтська частина ШІ-асистента відповідає за відображення інтерфейсу чату та забезпечення взаємодії користувача із системою. Вона реалізована у вигляді інтерактивного чат-вікна, яке містить історію повідомлень, поле введення тексту, кнопку надсилання повідомлення та індикатор генерації відповіді. Такий підхід забезпечує зручний інтерфейс взаємодії, подібний до сучасних месенджерів або онлайн-чатів підтримки. Для реалізації клієнтської частини використано React та TypeScript. React дозволяє створити динамічний SPA-інтерфейс, у якому повідомлення оновлюються без перезавантаження сторінки. TypeScript забезпечує типізацію даних, що зменшує кількість помилок під час розробки та полегшує підтримку програмного коду. Після введення повідомлення frontend формує HTTP-запит та передає його на серверну частину через REST API.

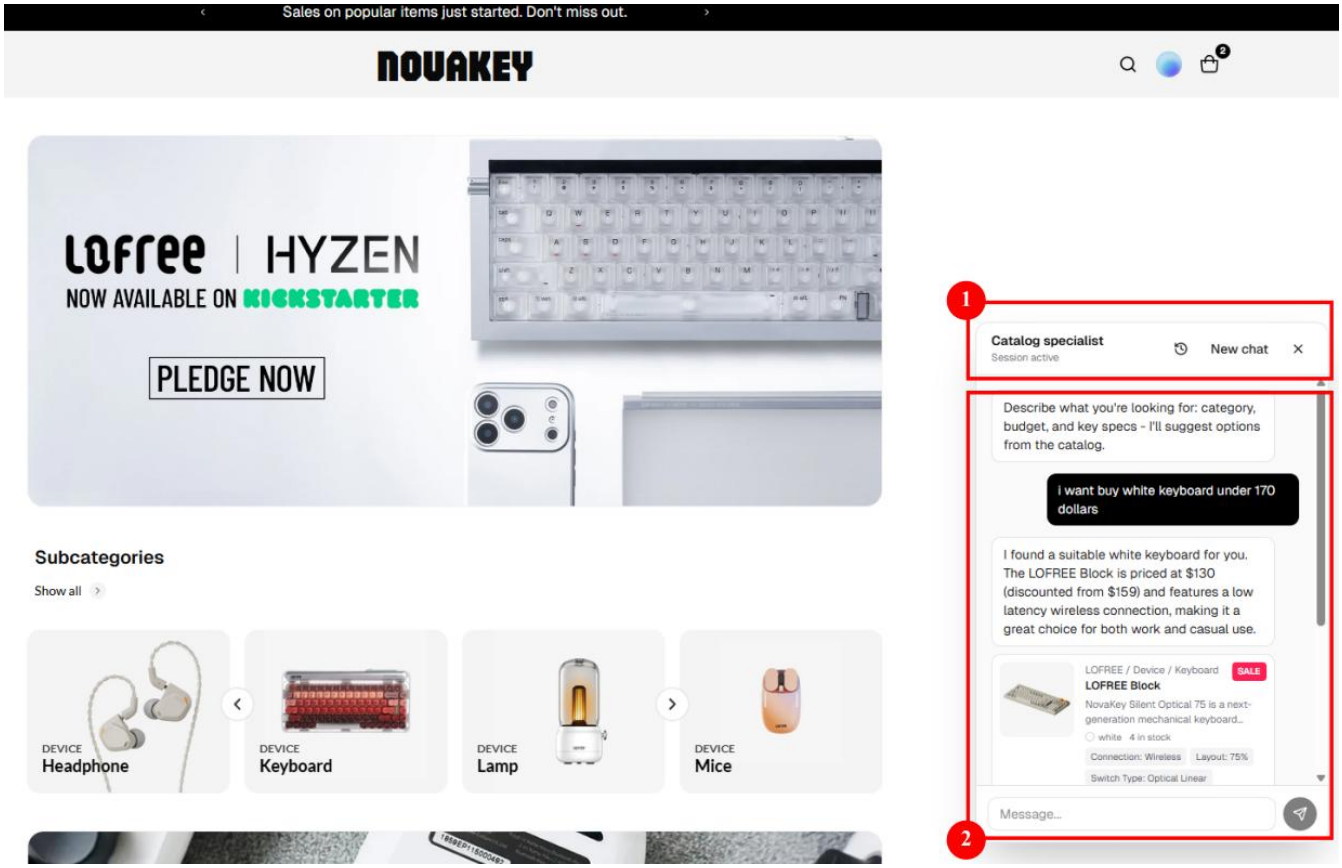

Рисунок 3.49 – Інтерфейс ШІ-асистента у клієнтській частині вебзастосунку

На рисунку 3.49 представлено інтерфейс ШІ-асистента у клієнтській частині вебзастосунку. Елемент 1 відповідає за навігацію та керування чатами: перегляд історії запитів, створення нового чату та закриття модального вікна. Елемент 2 містить основну область чату, у якій відображаються повідомлення користувача та відповіді ШІ-асистента, а також поле введення текстового запиту.

Серверна частина ШІ-асистента реалізована на основі Node.js та Express.js. Вона виконує роль проміжного шару між frontend, базою даних та ШІ-сервісом. Backend відповідає за обробку повідомлень, перевірку даних, формування контексту діалогу, взаємодію з базою знань та передачу запитів до ШІ-моделі.

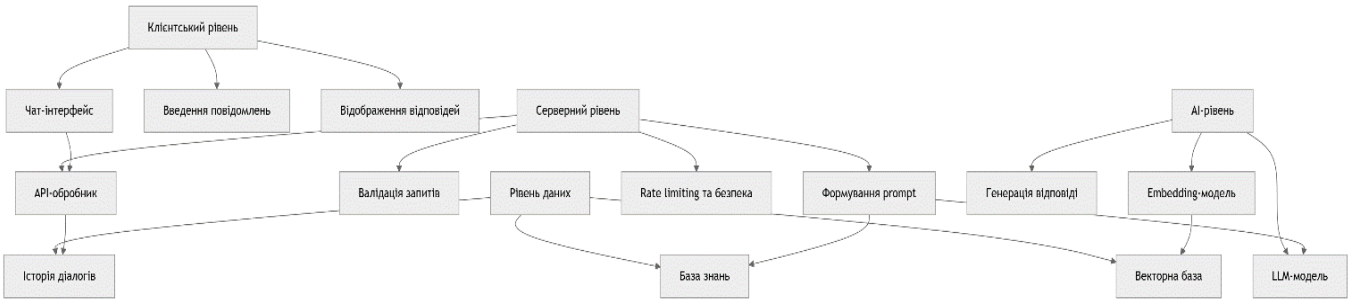

Рисунок 3.50 - Технологічна структура ШІ-асистента

На рисунку 3.50 наведено технологічну структуру ШІ-асистента та взаємозв’язок його компонентів. Клієнтський рівень відповідає за взаємодію користувача із чатом та відображення повідомлень. Серверний рівень забезпечує валідацію запитів, формування prompt-запитів, взаємодію з базою знань і системою безпеки. ШІ-рівень виконує генерацію відповідей та семантичний пошук за допомогою embedding-моделей і LLM-моделі.

Загальна логіка роботи ШІ-асистента складається з декількох етапів. Спочатку користувач вводить повідомлення у чаті вебзастосунку. Frontend передає повідомлення на backend через REST API. Сервер перевіряє коректність повідомлення, виконує очищення даних та за необхідності звертається до бази знань для отримання додаткового контексту. Після цього формується prompt-запит

для ШІ-моделі, яка генерує відповідь. Отриманий результат повертається на frontend та відображається у чаті користувача.

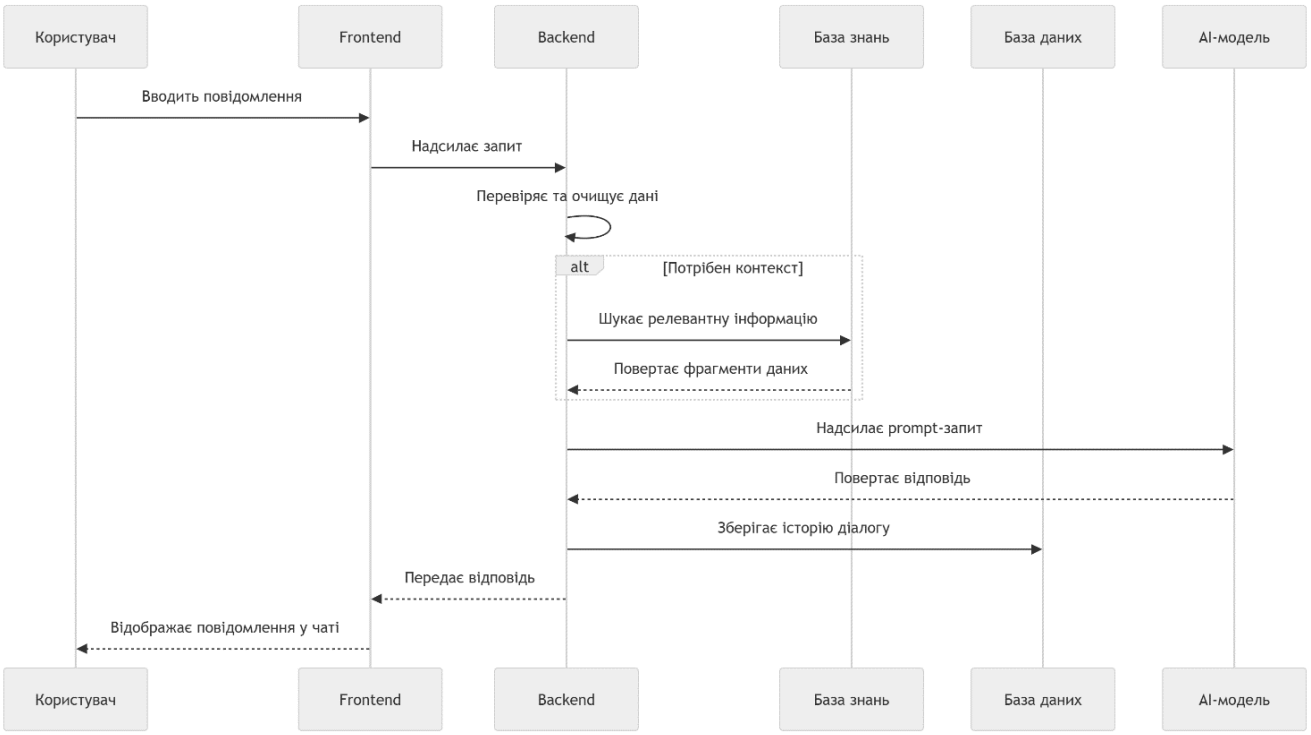

Рисунок 3.51 – Послідовність взаємодії компонентів

На рисунку 3.51 наведено UML-діаграму послідовності взаємодії компонентів системи. Діаграма демонструє процес передавання повідомлення від користувача до frontend, backend, бази знань та ШІ-моделі, а також механізм повернення сформованої відповіді користувачу. Для тестування та перевірки роботи ШІ-асистента використовувалося середовище Postman. Це дозволило перевірити коректність роботи REST API, формат передавання JSON-даних та правильність формування відповідей сервером.

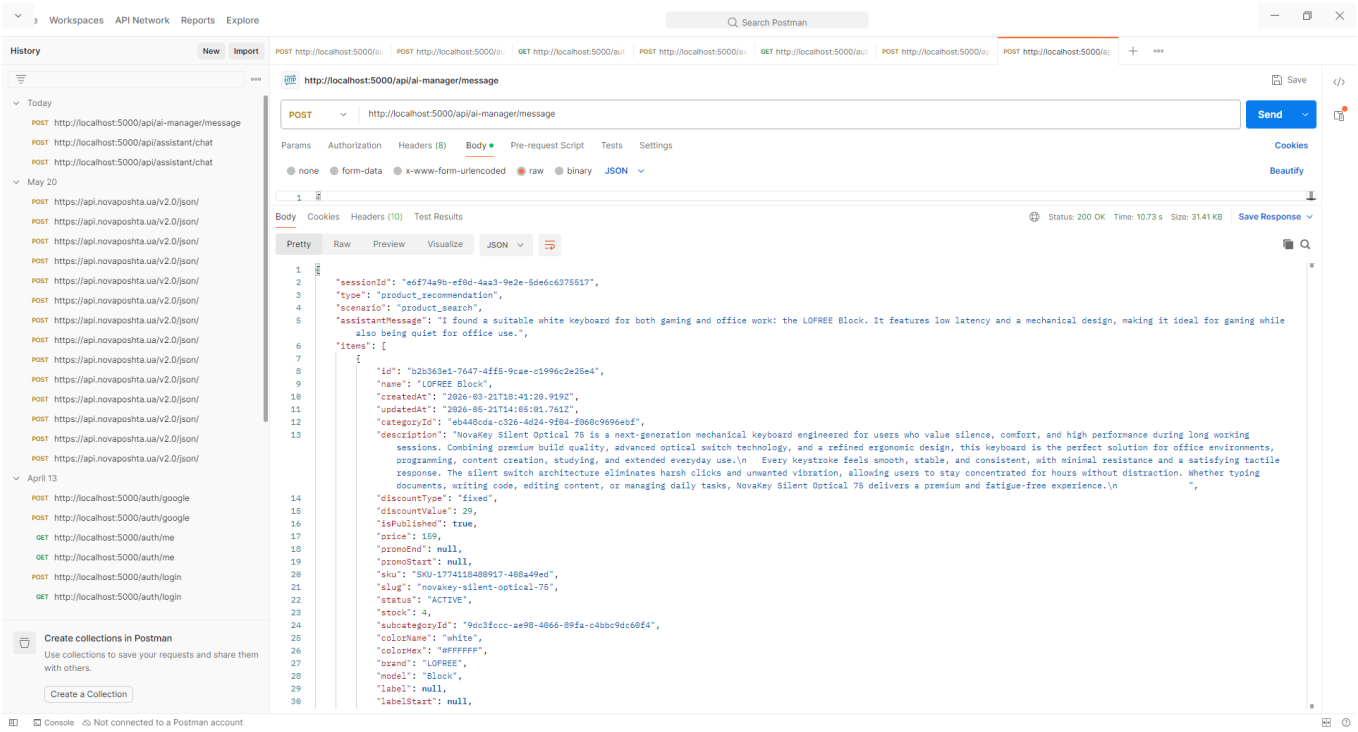

Рисунок 3.52 – Приклад відповіді ШІ-асистента через API у Postman

На рисунку 3.52 представлено приклад тестування ШІ-асистента через Postman. У відповіді сервера відображається текстова рекомендація ШІ-моделі, а також структуровані дані про знайдені товари: назва, опис, ціна, категорія та інші характеристики. Такий підхід дозволяє frontend-частині динамічно відображати рекомендовані товари [3-4] без додаткових запитів до сервера. Для обробки повідомлень користувача у системі реалізовано окремий алгоритм, який визначає послідовність дій під час взаємодії з ШІ-моделлю.

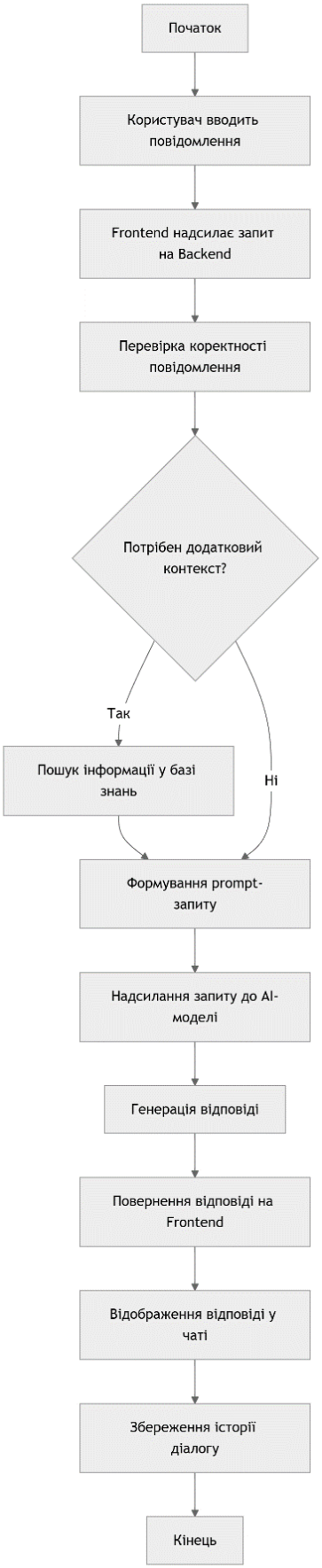

Рисунок 3.53 – Алгоритм обробки запиту користувача

На рисунку 3.53 наведено блок-схему алгоритму обробки запиту користувача. Після отримання повідомлення система перевіряє його коректність та визначає, чи потрібен додатковий контекст із бази знань. Якщо необхідно, виконується пошук релевантної інформації, після чого формується prompt-запит до ШІ-моделі. Згенерована відповідь повертається користувачу та зберігається в історії діалогів. Для підвищення точності відповідей у системі використовується підхід RAG. Перед генерацією відповіді ШІ-асистент виконує пошук релевантної інформації у внутрішній базі знань вебзастосунку. Це дозволяє моделі працювати не лише на основі загальних знань, а й використовувати актуальну інформацію про товари, характеристики, доставку або акційні пропозиції. Для реалізації семантичного пошуку використовуються embeddings - векторні представлення текстової інформації. Текст із бази знань або каталогу товарів перетворюється на числові вектори, після чого система порівнює їх із вектором запиту користувача та знаходить найбільш релевантні фрагменти інформації.

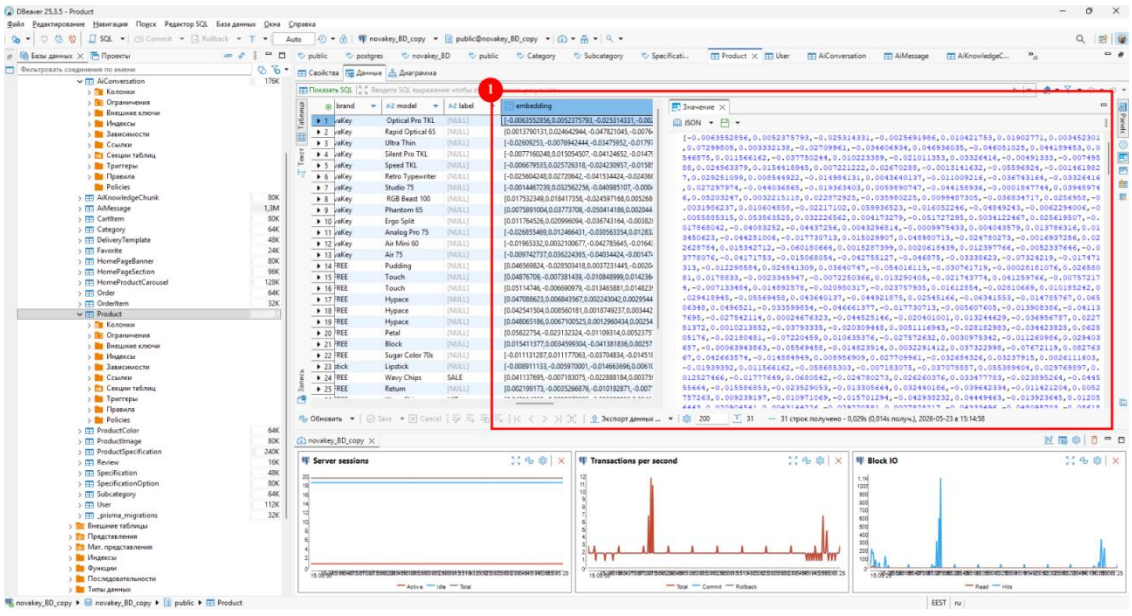

Рисунок 3.54 – Приклад, як відображується embeddings

На рисунку 3.54 наведено приклад збереження embeddings у базі даних PostgreSQL. Для кожного товару або текстового фрагмента формується векторне

представлення, яке використовується під час семантичного пошуку релевантної інформації.

Окрему роль у роботі ШІ-асистента відіграє prompt engineering - формування системних інструкцій для ШІ-моделі. Prompt містить правила поведінки ШІ-асистента, стиль відповідей, вимоги до безпеки та логіку обробки запитів.

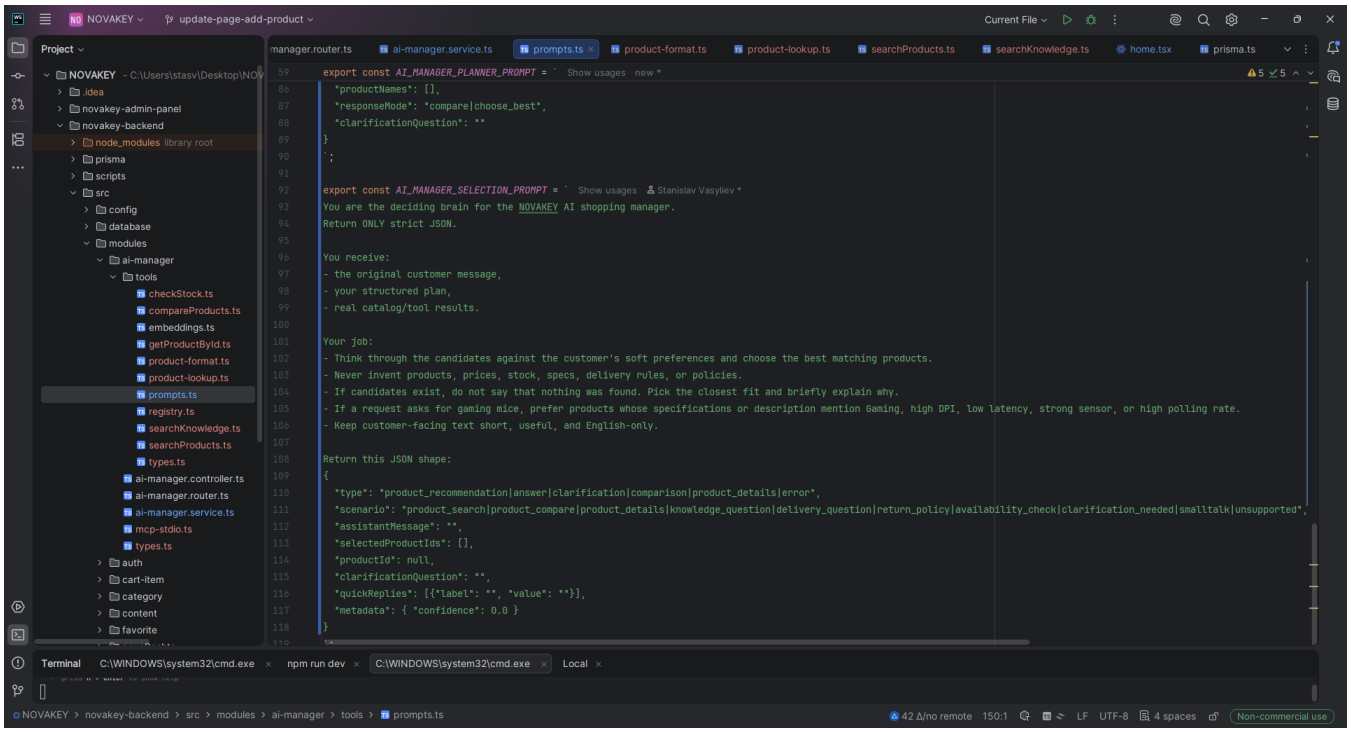

Рисунок 3.55 – Фрагмент prompt-інструкцій для ШІ-асистента

На рисунку 3.55 наведено приклад системних prompt-інструкцій для ШІ-асистента. Prompt визначає структуру JSON-відповіді, сценарії роботи асистента, правила формування рекомендацій та обмеження щодо генерації недостовірної інформації. Завдяки цьому ШІ-асистент може надавати більш структуровані та контрольовані відповіді. Для збереження історії повідомлень та контексту діалогів використовується база даних PostgreSQL. Це дозволяє зберігати історію чатів користувачів, аналізувати типові запити та покращувати якість роботи ШІ-асистента. Також у системі реалізовано механізми безпеки: обмеження доступу до API, захист приватних ключів та rate limiting для запобігання надмірній кількості запитів. Таким чином, інтеграція ШІ-асистента у вебзастосунок дозволила

реалізувати сучасний інструмент підтримки користувачів, який автоматизує обробку запитів, покращує навігацію по каталогу та забезпечує персоналізовану взаємодію користувача з інтернет-магазином. Поєднання frontend-інтерфейсу, backend-логіки, ШІ-моделі та механізмів семантичного пошуку дозволяє створити гнучку та масштабовану систему підтримки користувачів.

3.5 Тестування та аналіз продуктивності вебзастосунку

Після реалізації клієнтської та адміністративної частин вебзастосунку було проведено тестування продуктивності, доступності, SEO-оптимізації та відповідності сучасним вебстандартам. Для цього використовувався інструмент Google Lighthouse, інтегрований у середовище розробника браузера Google Chrome. Lighthouse дозволяє автоматично аналізувати швидкодію вебзастосунку, оптимізацію ресурсів, правильність структури сторінок, а також виявляти потенційні проблеми продуктивності. Тестування проводилося як для клієнтської частини вебзастосунку, так і для адміністративної панелі. Аналіз виконувався у локальному середовищі розробки (localhost), що є стандартною практикою під час створення та налагодження вебзастосунків. Варто зазначити, що тестування на localhost може частково впливати на результати Lighthouse, оскільки сайт запускається у режимі розробки, без повної production-оптимізації, HTTPS та серверного кешування. Незважаючи на це, отримані результати дозволяють оцінити рівень оптимізації та ефективність архітектури застосунку.

У процесі тестування адміністративної панелі було отримано високі результати продуктивності, показник Performance склав 97 балів, Best Practices - 100 балів, а SEO - 91 бал. Такі результати свідчать про ефективну оптимізацію інтерфейсу, невеликий обсяг завантажуваних ресурсів та правильну організацію структури сторінок. Адміністративна панель містить переважно текстову інформацію, таблиці, графіки та аналітичні блоки, тому її продуктивність є вищою порівняно з клієнтською частиною вебзастосунку. Клієнтська частина вебзастосунку продемонструвала дещо нижчі результати продуктивності, зокрема

показник Performance склав 77 балів. Це пов’язано з використанням великої кількості графічного контенту високої якості, банерів, каруселей, анімацій та інтерактивних UI-компонентів. Незважаючи на це, показники Accessibility та SEO залишилися на високому рівні, що свідчить про правильну структуру HTML-документів, використання семантичної розмітки та базову SEO-оптимізацію сторінок. Під час аналізу Lighthouse було виявлено декілька факторів, що впливають на швидкодію клієнтської частини вебзастосунку, такі як:

– використання великих зображень без повного стиснення;
– завантаження значного обсягу JavaScript-коду;
– наявність анімацій та динамічних компонентів;
– додаткове навантаження через режим розробки Vite;
– відсутність production-кешування у локальному середовищі.

Для покращення продуктивності було застосовано:

– оптимізацію та стиснення зображень;
– lazy loading для графічного контенту;
– code splitting та динамічне завантаження компонентів;
– мінімізацію JavaScript-коду;
– оптимізацію React-компонентів і повторних рендерів.

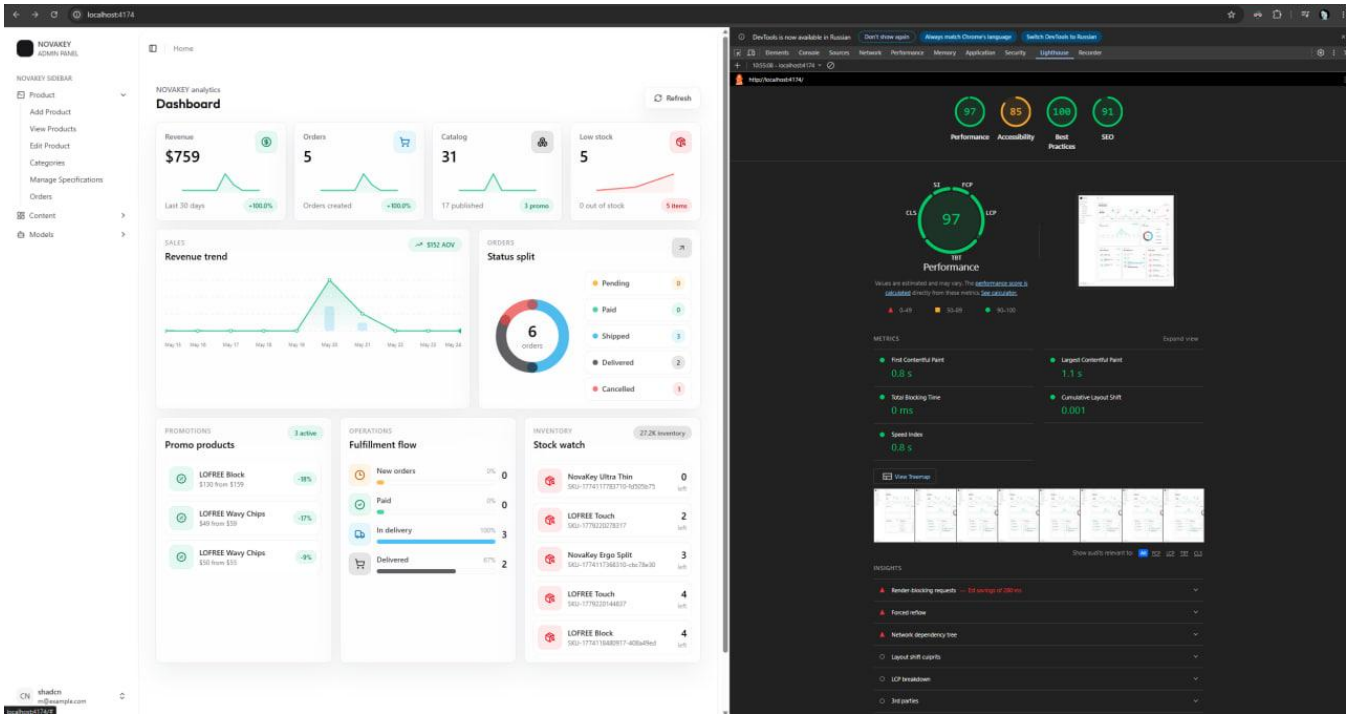

Рисунок 3.56 – Результати тестування адміністративної панелі за допомогою Google Lighthouse

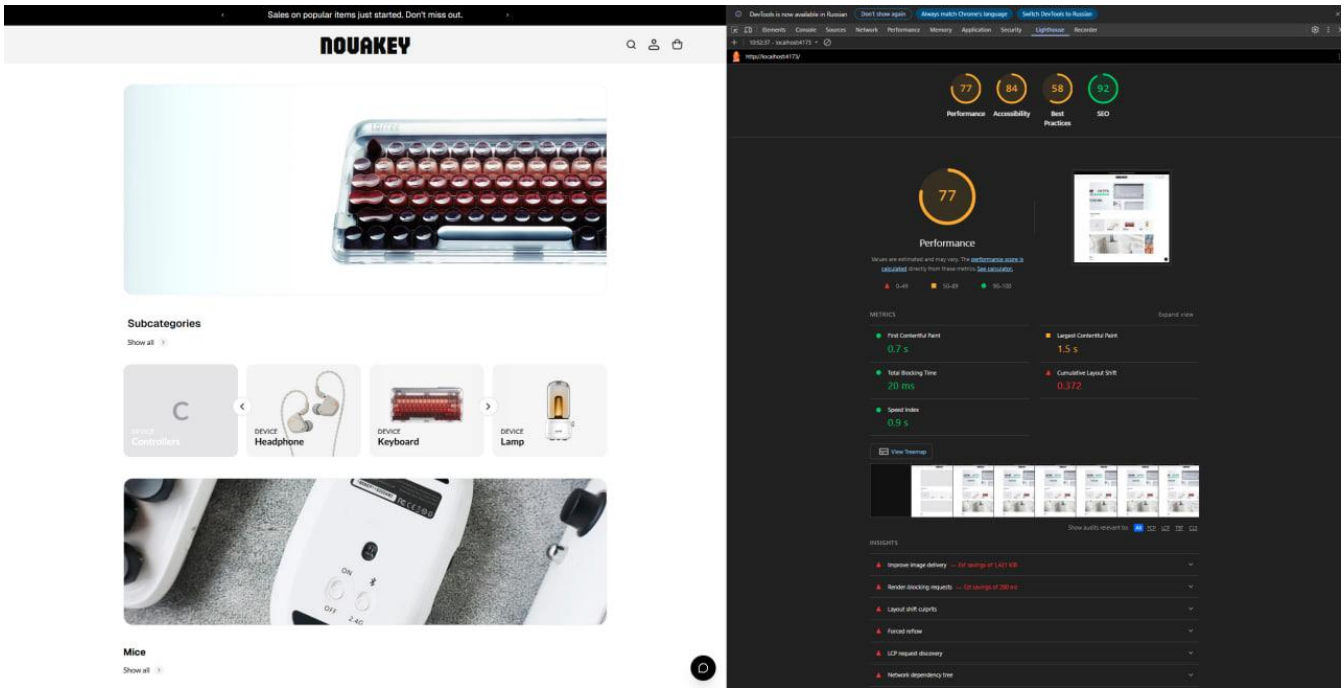

Рисунок 3.57 – Результати тестування клієнтської частини вебзастосунку за допомогою Google Lighthouse

Висновки до розділу 3

У третьому розділі було успішно реалізовано вебзастосунок електронної комерції з інтегрованим ШІ-асистентом. У процесі роботи організовано стабільну інфраструктуру розробки з використанням Figma для дизайну, WebStorm для кодування, GitHub для контролю версій та Docker Desktop для контейнеризації сервісів і бази даних PostgreSQL. На основі Node.js, Express та TypeScript створено модульне REST API серверної частини, де бізнес-логіку повністю ізольовано за окремими модулями. Взаємодію з базою даних налаштовано через Prisma ORM, а безпеку користувачів забезпечено шируванням паролів через bcrypt та автентифікацією за допомогою JWT-токенів. Для адміністрування магазину розроблено функціональну панель на React, яка дозволяє автономно керувати товарами, гнучко налаштовувати характеристики, змінювати контент головної сторінки у стилі Bento Grid та відстежувати бізнес-аналітику. Клієнтську частину реалізовано як швидкий SPA-застосунок з інтерактивним вибором кольорів товарів та автоматизованим Checkout-процесом, що містить інтеграцію зі Stripe для оплати та Nova Poshta API для генерації ТТН і вибору відділень.

Ключовою особливістю системи стала успішна інтеграція ШІ-асистента у чат на базі OpenAI API. Завдяки використанню підходу RAG та векторного пошуку pgvector помічник формує точні відповіді й порівняння на основі реальних товарів із бази даних, що виключає появу помилкової інформації.

Фінальне тестування продуктивності за допомогою Google Lighthouse зафіксувало відмінну швидкодію адмін-панелі (97 балів) та стабільний показник клієнтського сайту (77 балів), роботу якого було додатково оптимізовано через lazy loading та code splitting. Розроблений веебзастосунок повністю відповідає технічному завданню та готовий до експлуатації.

ВИСНОВКИ

У кваліфікаційній бакалаврській роботі здійснено розв’язання актуального науково-практичного завдання, що полягає у проєктуванні, реалізації та тестуванні сучасного масштабованого вебзастосунку електронної комерції з інтегрованим інтелектуальним асистентом підтримки користувачів на основі архітектури RAG. Отримані результати та узагальнені висновки за трьома основними напрямами дослідження дозволяють сформувати цілісне уявлення про виконану роботу.

За результатами аналізу предметної області та існуючих рішень встановлено, що ринок електронної комерції демонструє стрімке зростання, висуваючи нові вимоги до персоналізації сервісу та автоматизації клієнтської підтримки. Дослідження сучасних комерційних аналогів виявило низку системних недоліків, серед яких найважливішими є монолітна структура, низька гнучкість адміністрування через потребу залучення розробників для оновлення контенту та обмеженість традиційних чат-ботів, які працюють за жорсткими шаблонними сценаріями. У зв'язку з цим обґрунтовано доцільність впровадження Ші-консультанта нового покоління, спроможного аналізувати природну мову, оцінювати технічні характеристики та генерувати гнучкі індивідуальні рекомендації. На основі проведеного аналізу було успішно сформульовано повний перелік функціональних, архітектурних і технічних вимог до розроблюваної системи. В етапі обґрунтування вибору технологій та проєктування системи розроблено модульне клієнт-серверне архітектурне рішення з чітким розподілом відповідальності, орієнтоване на подальшу трансформацію у мікросервісну структуру через прошарок API Gateway. Для реалізації проєкту сформовано сучасний високопродуктивний стек технологій, що включає бібліотеку React, інструментарій Vite та фреймворк Tailwind CSS для розробки клієнтського інтерфейсу, а також середовище Node.js, фреймворк Express та Prisma ORM для побудови серверної логіки. В якості СКБД обрано реляційну базу даних PostgreSQL з розширенням pgvector для семантичного векторного пошуку. Спроєктовано

гнучку структуру бази даних, яка забезпечує цілісність інформації при збереженні складної ієрархії товарів, замовлень та характеристик. Окремо розроблено архітектуру взаємодії з Ші-асистентом на базі OpenAI API з використанням методології RAG, що дозволило забезпечити формування відповідей асистента виключно на основі реального актуального контексту інтернет-магазину.

Під час безпосередньої розробки та реалізації вебзастосунку створено повноцінне програмне рішення, що складається з клієнтської частини, адміністративної панелі та серверного REST API. У межах backend-сервісу успішно реалізовано бізнес-логику обробки замовлень, JWT-автентифікацію, безпечне хмарне збереження медіафайлів через Cloudinary, а також інтегровано платіжний шлюз Stripe та логістичне Nova Poshta API для автоматичного керування ТТН та трекінгу відправлень. Адміністративна панель надає інтуїтивні інструменти для управління асортиментом, налаштування характеристик, зміни контенту головної сторінки у стилі Bento Grid та відстеження детальної бізнес-аналітики продажів у реальному часі. Клієнтська частина реалізована як швидкий односторінковий застосунок з високим рівнем адаптивності, інтерактивною системою вибору кольорів та автоматизованим Checkout-процесом. Фінальне тестування продуктивності за допомогою інструменту Google Lighthouse підтвердило високу швидкодію створеного вебзастосунку, чудові показники SEO-оптимізації та доступності, що повністю доводить його відповідність поставленим вимогам і готовність до реальної експлуатації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Popa D., Buciu I. Laravel and Vue.js as tools to control IoT devices over the internet. Current state-of-the-art. // International Journal of Computers Communications & Control. 2025. Вип. 20, № 3.
2. Lamm T. Developing a full-stack web application: from architecture to dockerized deployment: a complete internship monitoring application. 2026.
3. Hansson I. K. M., Wiklund E., Bucaioni A., Provenzano L. AI-Powered Semantic Search for Historical Documentation: A Collaborative Research with Hitachi Energy. // International Conference on Information Technology – New Generations. Cham: Springer Nature Switzerland, 2025. С. 503–516.
4. Huyen C. AI Engineering: Building Applications with Foundation Models. Sebastopol: O'Reilly Media, 2025. 350 с.
5. Osmani A. Learning JavaScript Design Patterns. 2nd ed. Sebastopol: O'Reilly Media, 2023. 280 с.
6. Kleppmann M. Designing Data-Intensive Applications. Sebastopol: O'Reilly Media, 2017. 616 с.
7. Montgomery P. A. Modernizing Enterprise Web Platforms: An Evolutionary Analysis of ASP.NET to ASP.NET Core Transition Strategies. // The American Journal of Engineering and Technology. 2025. Вип. 7, № 11. С. 227–234.
8. Vaswani A., Shazeer N., Parmar N. та ін. Attention Is All You Need. // Advances in Neural Information Processing Systems. 2017. Вип. 30. С. 5998–6008.
9. Brown T. B., Mann B., Ryder N. та ін. Language Models are Few-Shot Learners. // Advances in Neural Information Processing Systems. 2020. Вип. 33. С. 1877–1901.
10. OpenAI. GPT-4 Technical Report [Електронний ресурс]. 2023. URL: https://arxiv.org/abs/2303.08774 (дата звернення: 27.04.2026).

11. Lewis P., Perez E., Piktus A. та ін. Retrieval-augmented generation for knowledge-intensive NLP tasks. // Advances in Neural Information Processing Systems. 2020. Вип. 33. С. 9459–9474.

12. Alto V. Practical Generative AI with ChatGPT: Unleash your prompt engineering potential with OpenAI technologies. Birmingham: Packt Publishing Ltd., 2025. 380 с.

13. Abharian Y. Justification of microservice approach to web development. // Innovative Solution in Modern Science. 2022. Вип. 8 (52). С. 5–19.

14. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol: O'Reilly Media, 2021. 612 с.

15. Richardson C. Microservices Patterns. Shelter Island: Manning Publications, 2018. 520 с.

16. Abdullayev M. Web development. Front-end programming. // German International Journal of Modern Science. 2024. Вип. 93. С. 45–52.

17. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. Sebastopol: O'Reilly Media, 2024. 300 с.

18. React Documentation [Електронний ресурс]. URL: https://react.dev/ (дата звернення: 27.04.2026).

19. Singh P., Srivastava M., Kansal M. та ін. A Comparative Analysis of Modern Frontend Frameworks for Building Large-Scale Web Applications. // 2023 International Conference on Disruptive Technologies (ICDT). Greater Noida, India, 2023. С. 531–535.

20. Mehmood A. Web development frameworks: comparing React, Angular, and Vue. // Computer Science Bulletin. 2023. Вип. 6, № 01. С. 29–43.

21. Vepsäläinen J., Hellas A., Vuorimaa P. The rise of disappearing frameworks in web development. // International Conference on Web Engineering. Cham: Springer Nature Switzerland, 2023. С. 319–326.

22. NPM Trends: Angular vs React vs Vue [Електронний ресурс]. URL: https://npmtrends.com/angular-vs-react-vs-vue (дата звернення: 27.04.2026).

23. Lunnikivi V. Web framework modernizations. 2025.

24. Freeman A. Pro ASP.NET Core 9. New York: Apress, 2025. 1000 с.

25. Shafaq M. Z., Khaliqyar R., Sadat S. A. Comparative Security Analysis of Django and Laravel Web Development Frameworks. // International Journal of Research and Scientific Innovation (IJRSI). 2026. Вип. 13, № 2.

26. Wanjale K., Shah S. A., Kharole C. та ін. Detecting DDoS Attacks in Real-Time with Minimal Infrastructure: A Node.js Approach. // 2025 International Conference on Future Technologies (ICFT). IEEE, 2025. С. 1–8.

27. Node.js Documentation [Електронний ресурс]. URL: https://nodejs.org/docs (дата звернення: 27.04.2026).

28. Express.js Documentation [Електронний ресурс]. URL: https://expressjs.com/ (дата звернення: 27.04.2026).

29. PostgreSQL Documentation [Електронний ресурс]. URL: https://www.postgresql.org/docs/ (дата звернення: 27.04.2026).

30. Prisma ORM Documentation [Електронний ресурс]. URL: https://www.prisma.io/docs (дата звернення: 27.04.2026).

31. PostgreSQL Full Text Search Documentation [Електронний ресурс]. URL: https://www.postgresql.org/docs/current/textsearch.html (дата звернення: 27.04.2026).

32. pgvector Documentation [Електронний ресурс]. URL: https://github.com/pgvector/pgvector (дата звернення: 27.04.2026).

33. Johnson J., Douze M., Jégou H. Billion-scale similarity search with GPUs. // IEEE Transactions on Big Data. 2019. Вип. 7, № 3. С. 535–547.

34. OpenAI API Documentation [Електронний ресурс]. URL: https://platform.openai.com/docs/ (дата звернення: 27.04.2026).

35. Zhang M., Arcuri A., Li Y. та ін. Fuzzing microservices: A series of user studies in industry on industrial systems with EvoMaster. // Science of Computer Programming. 2025. Вип. 246. С. 103322.

ДОДАТОК А

Лістинг сервісу управління категоріями (category.service.ts)

```
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

type CategoryInput = {
  name?: unknown;
  imageUrl?: unknown;
  homeImageUrl?: unknown;
  description?: unknown;
  isActive?: unknown;
  sortOrder?: unknown;
  bentoSize?: unknown;
  carouselSize?: unknown;
  homeTextColor?: unknown;
  bentoTextColor?: unknown;
};

type SubcategoryInput = CategoryInput & {
  categoryId?: unknown;
};

function compactString(value: unknown): string | null {
  if (typeof value !== "string") return null;
  const trimmed = value.trim();
  return trimmed ? trimmed : null;
}

function toBoolean(value: unknown): boolean | undefined {
  return typeof value === "boolean" ? value : undefined;
}

function toInt(value: unknown): number | undefined {
  if (value === undefined || value === null || value === "") return undefined;

  const numberValue = Number(value);

  if (!Number.isFinite(numberValue)) return undefined;

  return Math.max(0, Math.round(numberValue));
}

function toBentoSize(value: unknown): string | undefined {
  return value === "NORMAL" || value === "WIDE" || value === "TALL"
    ? value
    : undefined;
}

function toCarouselSize(value: unknown): string | undefined {
  return value === "SMALL" || value === "MEDIUM" || value === "LARGE"
    ? value
    : undefined;
}

function toColor(value: unknown): string | null | undefined {
  if (value === undefined) return undefined;
  const color = compactString(value);
```

```
  return color && /^#[0-9a-f]{6}$/i.test(color) ? color : null;
}

function normalizeCategoryInput(input: string | CategoryInput, requireName = false)
{
  const source = typeof input === "string" ? { name: input } : input ?? {};
  const name = compactString(source.name);
  const data: Record<string, unknown> = {};

  if (name) {
    data.name = name;
  } else if (requireName) {
    throw new Error("Category name is required");
  }

  if ("imageUrl" in source) data.imageUrl = compactString(source.imageUrl);
  if ("homeImageUrl" in source) data.homeImageUrl =
compactString(source.homeImageUrl);
  if ("description" in source) data.description = compactString(source.description);

  const isActive = toBoolean(source.isActive);
  if (isActive !== undefined) data.isActive = isActive;

  const sortOrder = toInt(source.sortOrder);
  if (sortOrder !== undefined) data.sortOrder = sortOrder;

  const bentoSize = toBentoSize(source.bentoSize);
  if (bentoSize !== undefined) data.bentoSize = bentoSize;

  const carouselSize = toCarouselSize(source.carouselSize);
  if (carouselSize !== undefined) data.carouselSize = carouselSize;

  const homeTextColor = toColor(source.homeTextColor);
  if (homeTextColor !== undefined) data.homeTextColor = homeTextColor;

  const bentoTextColor = toColor(source.bentoTextColor);
  if (bentoTextColor !== undefined) data.bentoTextColor = bentoTextColor;

  return data;
}

function normalizeSubcategoryInput(
  input: string | SubcategoryInput,
  categoryId?: string,
  requireName = false
) {
  const source = typeof input === "string" ? { name: input, categoryId } : input ??
{};
  const data = normalizeCategoryInput(source, requireName);
  const resolvedCategoryId = compactString(categoryId) ??
compactString(source.categoryId);

  if (resolvedCategoryId) {
    data.categoryId = resolvedCategoryId;
  }

  return data;
}

async function getCategoryById(id: string) {
  const categories = await getAllCategories();
```

```
  return categories.find((item: any) => item.id === id);
}

async function rawUpdate(
    table: "Category" | "Subcategory",
    id: string,
    data: Record<string, unknown>
) {
  const entries = Object.entries(data);

  if (!entries.length) return;

  const assignments = entries
      .map(([key], index) => {
        const param = `$${index + 1}`;

        if (key === "bentoSize") {
          return `"${key}" = ${param}::"CatalogBentoSize"`;
        }

        if (key === "carouselSize") {
          return `"${key}" = ${param}::"CatalogCarouselSize"`;
        }

        return `"${key}" = ${param}`;
      })
      .join(", ");

  const values = entries.map(([, value]) => value);

  await (prisma as any).$executeRawUnsafe(
      `
        UPDATE "${table}"
        SET ${assignments},
            "updatedAt" = CURRENT_TIMESTAMP
        WHERE "id" = $${values.length + 1}
      `,
      ...values,
      id
  );
}
export const getAllCategories = async () => {
  const categories = await (prisma as any).$queryRaw`
    SELECT
      "id",
      "name",
      "imageUrl",
      "homeImageUrl",
      "description",
      "isActive",
      "sortOrder",
      "bentoSize",
      "carouselSize",
      "homeTextColor",
      "bentoTextColor",
      "createdAt",
      "updatedAt"
    FROM "Category"
    ORDER BY "sortOrder" ASC, "name" ASC
  `;
```

```
  const subcategories = await (prisma as any).$queryRaw`
    SELECT
      "id",
      "name",
      "categoryId",
      "imageUrl",
      "homeImageUrl",
      "description",
      "isActive",
      "sortOrder",
      "bentoSize",
      "carouselSize",
      "homeTextColor",
      "bentoTextColor",
      "createdAt",
      "updatedAt"
    FROM "Subcategory"
    ORDER BY "sortOrder" ASC, "name" ASC
  `;

  return (categories as any[]).map((category) => ({
    ...category,
    subcategories: (subcategories as any[]).filter(
      (subcategory) => subcategory.categoryId === category.id
    ),
  }));
};

export const createCategory = async (input: string | CategoryInput) => {
  return (prisma as any).category.create({
    data: normalizeCategoryInput(input, true),
    include: {
      subcategories: true,
    },
  });
};

export const updateCategory = async (id: string, input: string | CategoryInput) =>
{
  await rawUpdate("Category", id, normalizeCategoryInput(input));
  return getCategoryById(id);
};

export const deleteCategory = async (id: string) => {
  return prisma.category.delete({
    where: { id },
  });
};

export const createSubcategory = async (
  input: string | SubcategoryInput,
  categoryId?: string
) => {
  return (prisma as any).subcategory.create({
    data: normalizeSubcategoryInput(input, categoryId, true),
  });
};

export const updateSubcategory = async (id: string, input: string | SubcategoryInput)
=> {
  await rawUpdate("Subcategory", id, normalizeSubcategoryInput(input));
```

```
  const categories = await getAllCategories();
  return categories
    .flatMap((category: any) => category.subcategories ?? [])
    .find((subcategory: any) => subcategory.id === id);
};

export const deleteSubcategory = async (id: string) => {
  return prisma.subcategory.delete({
    where: { id },
  });
};
```

ДОДАТОК Б

Лістинг контролера управління категоріями (category.controller.ts)

```
import { Request, Response } from "express";
import * as service from "./category.service";

export const getCategories = async (_: Request, res: Response) => {
  const data = await service.getAllCategories();
  res.json(data);
};

export const createCategory = async (req: Request, res: Response) => {
  const data = await service.createCategory(req.body);
  res.json(data);
};

export const updateCategory = async (req: Request, res: Response) => {
const id = req.params.id as string;
  const data = await service.updateCategory(id, req.body);
  res.json(data);
};

export const deleteCategory = async (req: Request, res: Response) => {
const id = req.params.id as string;
  await service.deleteCategory(id);
  res.json({ message: "Deleted" });
};

export const createSubcategory = async (req: Request, res: Response) => {
  const { categoryId } = req.body;
  const data = await service.createSubcategory(req.body, categoryId);
  res.json(data);
};

export const updateSubcategory = async (req: Request, res: Response) => {
const id = req.params.id as string;
  const data = await service.updateSubcategory(id, req.body);
  res.json(data);
};

export const deleteSubcategory = async (req: Request, res: Response) => {
const id = req.params.id as string;
  await service.deleteSubcategory(id);
  res.json({ message: "Deleted" });
};
```

ДОДАТОК В

Лістинг маршрутів управління категоріями (category.routes.ts)

```
import { Router } from "express";
import * as controller from "./category.controller";

const router = Router();

router.get("/", controller.getCategories);
router.post("/", controller.createCategory);
router.patch("/:id", controller.updateCategory);
router.delete("/:id", controller.deleteCategory);

router.post("/subcategory", controller.createSubcategory);
router.patch("/subcategory/:id", controller.updateSubcategory);
router.delete("/subcategory/:id", controller.deleteSubcategory);

export default router;
```

ДОДАТОК Г

Лістинг реєстрації основних маршрутів REST API

```
import type { Express } from "express";
import categoryRoutes from "../modules/category/category.routes";
import specificationsRoutes from "../modules/specifications/specifications.routes";
import productRoutes from "../modules/product/product.routes";
import authRouter from "../modules/auth/auth.routes";
import favoriteRouter from "../modules/favorite/favorite.router";
import orderRouter from "../modules/order/order.router";
import cartItemRouter from "../modules/cart-item/cart-item.router";
import novaPoshtaRoutes from  "../modules/novaPoshta/novaPoshta.router"
import aiManagerRouter from "../modules/ai-manager/ai-manager.router";
import homeContentRoutes from "../modules/content/home-content.routes";
import productColorRoutes from "../modules/product-color/product-color.routes";

export function registerRoutes(app: Express): void {
    app.use("/categories", categoryRoutes);
    app.use("/specifications", specificationsRoutes);
    app.use("/products", productRoutes);
    app.use("/product-colors", productColorRoutes);
    app.use("/content", homeContentRoutes);
    app.use("/auth", authRouter);
    app.use("/favorite", favoriteRouter);
    app.use("/order", orderRouter);
    app.use("/cart-item", cartItemRouter);
    app.use("/api/nova-poshta", novaPoshtaRoutes);
    app.use("/api/ai-manager", aiManagerRouter);
}
```