Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ВЕБПЛАТФОРМИ ДЛЯ ЕЛЕКТРОННОЇ КОМЕРЦІЇ З ІНТЕГРОВАНИМ ІНТЕЛЕКТУАЛЬНИМ ЧАТ-БОТОМ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-21-3
Саблєв А. І.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник ст. викл. Путятіна О. Є.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Саблєву Артему Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка вебплатформи для електронної комерції з інтегрованим інтелектуальним чат-ботом

затверджена наказом університету від 19 травня 2025 року № 381Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 25 травня 2025 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, фреймворк веброзробки з відкритим кодом Express.js.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд основних методів реалізації чат-ботів.
2. Огляд ключових елементів інтерфейсу і функціоналу існуючих інтернет-магазинів.
3. Вибір архітектури застосунку.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) аналіз функціоналу, моделювання структури додатку.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	07.04.2025	
2	Аналіз завдання, підбір літератури	08.04.25-09.04.25	
3	Аналіз літератури з досліджуваної проблеми	10.04.25-12.04.25	
4	Аналіз існуючих рішень	13.04.25-14.04.25	
5	Встановлення вимог	14.04.25-15.04.25	
6	Моделювання архітектури	16.04.25-20.04.25	
7	Програмна реалізація	21.04.25-11.05.25	
8	Оформлення пояснювальної записки	12.05.25-20.05.25	
9	Перевірка на нормоконтроль	21.05.25-01.06.25	
10	Перевірка на плагіат	21.05.25-01.06.25	
11	Рецензування	21.05.25-01.06.25	
12	Підготовка презентації та доповіді	21.05.25-18.06.25	
13	Занесення роботи в електронний архів	02.06.25-18.06.25	
14	Попередній захист кваліфікаційної роботи	02.06.25-18.06.25	

Дата видачі завдання 7 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ____ ст. викл. Путятіна О.Є._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 63 с., 6 табл., 35 рис., 31 джерело.

ВЕБСАЙТ, МОВА ГІПЕРТЕКСТОВОЇ РОЗМІТКИ HTML5, КАСКАДНІ ТАБЛИЦІ СТИЛІВ CSS, МОВА ПРОГРАМУВАННЯ JAVASCRIPT, JAVASCRIPT ОТОЧЕННЯ NODE.JS, ДОКУМЕНТНО ОРІЄНТОВНА СИСТЕМА КЕРУВАННЯ БАЗАМИ ДАННИХ MONGOBD, ЧАТ-БОТ, ВЕЛИКА МОВНА МОДЕЛЬ, ОБРОБКА ПРИРОДНЬОЇ МОВИ.

Об’єктом роботи є розробка вебплатформи для електронної комерції з інтеграцією чат-боту на основі штучного інтелекту.

Метою роботи є розробка зручної платформи зі спрощеним процесом додавання товарів, системою корзини, чат-ботом для оптимізації надання підтримки користувачам.

Сайт має загальну версію для всіх користувачів та адмін-панель для керування товарами, замовленнями та користувачами. Наявна база даних для збереження даних. Є система реєстрації та рівня доступу користувачів. Виконана система кошику з гібридною схемою збереження. Наявний чат-бот з підключенням сторонньої LLM-моделі.

У результаті роботи створено платформу для реалізації товарів будь-якого типу без потреби знання програмного коду вебзастосунку для базових операцій.

WEBSITE, HYPERTEXT MARKUP LANGUAGE HTML5, CASCADING STYLE SHEETS CSS, PROGRAMMING LANGUAGE JAVASCRIPT, JAVASCRIPT ENVIRONMENT NODE.JS, DOCUMENT-ORIENTED DATABASE MANAGEMENT SYSTEM MONGOBD, CHATBOT, LARGE LANGUAGE MODEL, NATURAL LANGUAGE PROCESSING.

The object of work is the development of a web-based e-commerce platform with the integration of an artificial intelligence chatbot.

The aim is to develop a user-friendly platform with a simplified process of adding products, a shopping cart system, and a chatbot to optimize user support.

The site has a version for all users and an admin panel for managing products, orders, and users. There is a database for data storage. There is a system of registration and user access levels. A shopping cart system with a hybrid storage scheme has been implemented. There is a chatbot with the connection of a third-party LLM model.

As a result, a platform for the e-commerce has been created, with no need of programming knowledge to do basic operations.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HTML – HyperText Markup Language (мова розмітки гіпертексту)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

REST – Representational State Transfer (передача репрезентативного стану)

MVC – Model-View-Controller (Модель-Вид-Контролер)

API – інтерфейс прикладного програмування

ВСТУП

Електронна комерція давно стала звичною для користувачів Інтернету з усіх регіонів. За роки розвитку інтернет-магазинів було визначено головні параметри, за якими визначається успішність магазину.

Першим параметром є зручність магазину як для користувачів, так і для його адміністраторів. Якщо користувачу важко користуватися інтерфейсом, а сайт не надає простого доступу до звичних функцій пошуку і фільтрації, то повторного відвідування не відбудеться. Якщо сайт має ускладнену систему керування, то зростає шанс виникнення помилки при створенні товарів.

Другим параметром є асортимент, оскільки багато користувачів надають перевагу купити все необхідне разом в одному місці.

Третім, але не останнім по важливості є якість підтримки користувачів. Якщо покупець не може швидко отримати відповідь на своє запитання або зіткнеться з труднощами при оформленні замовлення без належної допомоги, зростає ризик відмови від покупки. Особливо це стосується технічно складних товарів, де користувачу буде потрібна кваліфікована допомога.

Актуальність роботи полягає у впровадженні системи простого керування товарами, а також впровадження технології останніх років – інтеграції чат-боту на основі LLM з метою оптимізації надання підтримки користувачам. Користувачі могли вирішувати більшість питань за допомогою чат-ботів минулого покоління, але вони не вирізнялися гнучкістю відповідей через потребу у, фактично, ручному відповідей. Сучасний чат-бот на основі LLM дозволяє надавати не тільки заготовлені відповіді, а ще й надавати більш складну консультацію користувачам навіть без технічних знань, чим знімає навантаження на адміністратора сайту та економить час, що витрачений на рутинні питання.

1 ОГЛЯД ОСНОВНИХ ПІДХОДІВ

1.1 Історія основних методів реалізації чат-ботів

1.1.1 Підхід на основі правил (rule-based)

До появи LLM перші покоління чат-ботів базувалися на підході rule-based, де вся логіка взаємодії була побудована на основі правил, які задавалися вручну. Першою подібною програмою була ELIZA. Працюють rule-based чат-боти за такою схемою: спочатку відбувається нормалізація тексту (переведення відмінків у інфінітив, числових значень у прописні, розшифрування скорочень та транскрипцій) та токенізація. Далі запит проходить перевірку із шаблонами розмов з людиною. У випадку якщо шаблон збігається, надається задана відповідь, згідно заданої розробником гілки розвитку подій, що містить цей шаблон. Якщо ж ніякого збігу не знайдено, бот надає відповідь, що не зрозумів користувача.

Такий підхід дозволяє надавати повністю контрольовані відповіді, значно оптимізуючи час як служби підтримки, так і самого користувача, надаючи йому можливість адресувати часті запитання чат-боту із заготовленими відповідями. Водночас, такий спосіб реалізації дуже обмежений у заданні нових правил, оскільки їх прописування вручну займає довгий час, і майже унеможливлює вирішення питань більш глибокого контексту, не говорячи про те, що такий чат-бот не здатний отримувати контекст повідомлення, і якщо користувач перефразує своє запитання, чат-бот може його не зрозуміти [1-2].

1.1.2 Retrieval-based система

Топорність if-then схеми для rule-based ботів була очевидною, тому розвитком стала retrieval-based модель, яка основана на пошуку найліпшої

відповіді у наборі «запит – відповідь». Спочатку запит користувача перетворювався на вектор, а далі обчислювався на схожість із заготовленими запитами, повертаючи відповідь для запиту з найбільшою збіжністю, що дозволило обмежити строгість застосування ключових слів у запиті, за якими отримується відповідь. Проте залишилися мінуси у вигляді потреби у ключових словах для порівняння та, відповідно, потребі у утриманні та регулярному доповненні бази знань на основі запитів вже реальних користувачів, щоб система частіше розпізнавала запити користувачів. Також те, що це лише відносно проста статистична схема, що заснована на схожості змісту і ключових слів, могло призвести до ситуацій, коли запит був або «незнайомим» через значну різницю з усіма запитами у базі, або був опрацьований невірною відповіддю до схожого за вмістом, але іншого по контекстуальному значенню запиту.

На рисунку 1.1 показано приклад обробки запиту чат-ботом на основі правил.

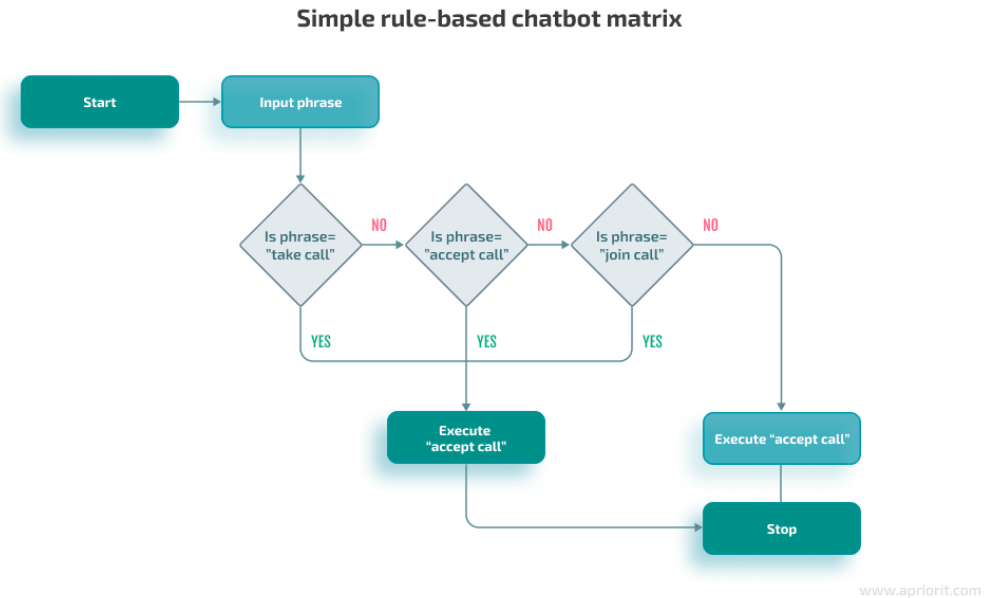

Рисунок 1.1 – Діаграма з прикладом прийняття рішення rule-based ботом [3]

1.1.3 Послідовні мережі на основі LSTM-шарів

Новою ерою чат-ботів стала поява нейронних моделей, що базуються на архітектурі з послідовностей. Такі моделі дозволяють зчитувати довільний рядок і генерувати відповідь, вони називаються Seq2Seq (sequence-to-sequence) моделями і складаються з двох частин – енкодеру та декодеру.

Енкодер перетворює кожен токен з вхідного тексту у вектори, які додаються у загальний контекстний вектор (або «внутрішній стан»), оновлюючи його. Фінальна версія контекстного вектору є основним «сенсом» усього прочитаного тексту. Процес сприйняття вхідної послідовності у енкодері можна покращити додавши, наприклад, два обходи токенів, з кожного «краю», з кожною ітерацією поєднуючи стани, і таким чином враховувати слова як на початку, так і у кінці речення.

Одержаний контекстний вектор передається до декодера, який використовує його як початкову «пам’ять» для генерації відповіді. Декодер, подібно до енкодера, є LSTM-мережею, яка починає роботу з спеціального початкового токена і на кожному кроці генерує наступний токен у відповіді, використовуючи попередній внутрішній стан та щойно згенерований токен. Процес відбудеться до поки не буде згенеровано завершальний токен або не буде досягнуто максимальної довжини відповіді.

Модель Seq2Seq стала проривною, оскільки тепер для навчання потребувала не величезні пари «запит по ключовим словам – відповідь», що потрібно прописувати вручну, а запити реальних користувачів, текстові повідомлення до служби підтримки, будь-яка текстова інформація, з якої можна було отримати набори «вхід – вихід». Модель навчається на них отримуючи статистичні закономірності і вчиться узагальнювати шаблони відповідей без явних інструкцій, як було з retrieval-based системою.

Водночас, через обмеженість LSTM не дозволяє утримувати довготривалу пам’ять, такі системи можуть втрачати частину контексту при

роботі з довгими реченнями або довгими діалогами, оскільки увесь контекст треба стиснути в один вектор.

1.1.4 Seq2Seq з механізмом уваги

З метою подолання проблеми вузького вікна пам’яті контексту було додано механізм уваги. Замість того щоб покладатися лише на фінальний стан енкодера, декодер отримав змогу на кожному кроці озиратися на всі стани вихідної послідовності й обирати, які саме слова найбільш важливі для генерації поточного токена. Кожен стан отримує вагу відносно його релевантності, таким чином відмічаючи найбільш корисні частини речення для генерації нового токену. Таким чином модель краще запам’ятовує контекст довгих речень або кількох діалогів, а також краще розуміє які слова між собою пов’язані контекстуально.

1.1.5 Transformer

Після додання механізму уваги у Seq2Seq, у 2017 році відбувся новий прорив – відмова від рекурентної обробки і перехід виключно до системи уваги. Тепер замість зчитування тексту послідовно, Transformer аналізує усі токени одночасно, визначаючи ступінь зв’язку кожного токена з усіма іншими. Таким чином відбувається відмова від загального контекстного вектору, який занадто узагальнював довгі речення або діалоги – тепер кожен токен отримує доступ до повного контексту.

Архітектура складається з енкодеру та декодеру, але їх принцип роботи відрізняється від Seq2Seq. В енкодері кожен блок уваги складається з кількох паралельних «голів», де кожна вивчає текст на предмет відповідних кожному набору закономірностей. Наприклад, одна мережа шукає граматичні зв’язки, інша шукає логічно пов’язані слова (для моделі це ті, що зазвичай

вживаються поряд), ще одна вивчає порядок слів. Кожному токену додається позиційне кодування, щоб враховувати порядок слів не зважаючи на паралельну обробку. Після цього модель вивчає вже створені вектор кожного токену, але окремо від зв’язку до інших слів.

Декодер при генерації токену спочатку враховує отриману з енкодера важливість кожного токену до іншого у запиті, а потім враховує ще й вже згенеровані у відповіді токени для створення найбільш контекстуально відповідного нового токену.

У результаті, нова архітектура ефективніше навчається, має значно ефективнішу довгострокову пам’ять, що позитивно впливає на утримання інформації у довгих текстах, а також дозволяє моделі краще розуміти зв’язки між словами не лише в одному реченні, а навіть у цілих абзацах або діалогах [4-5].

1.2 Огляд основного функціоналу існуючих інтернет-магазинів

Оглянемо функціонал інтернет-магазинів, що існують на українському ринку довгий час і стали звичними для користувачів. Скріншоти показано на рисунках 1.2 – 1.4.

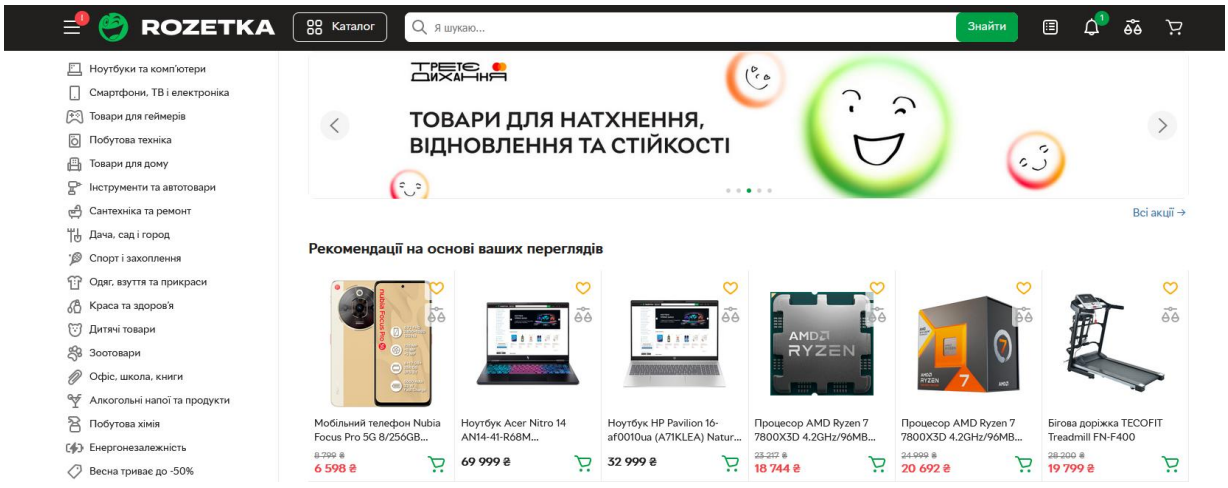

Рисунок 1.2 – Приклад головної сторінки сайту Rozetka.com

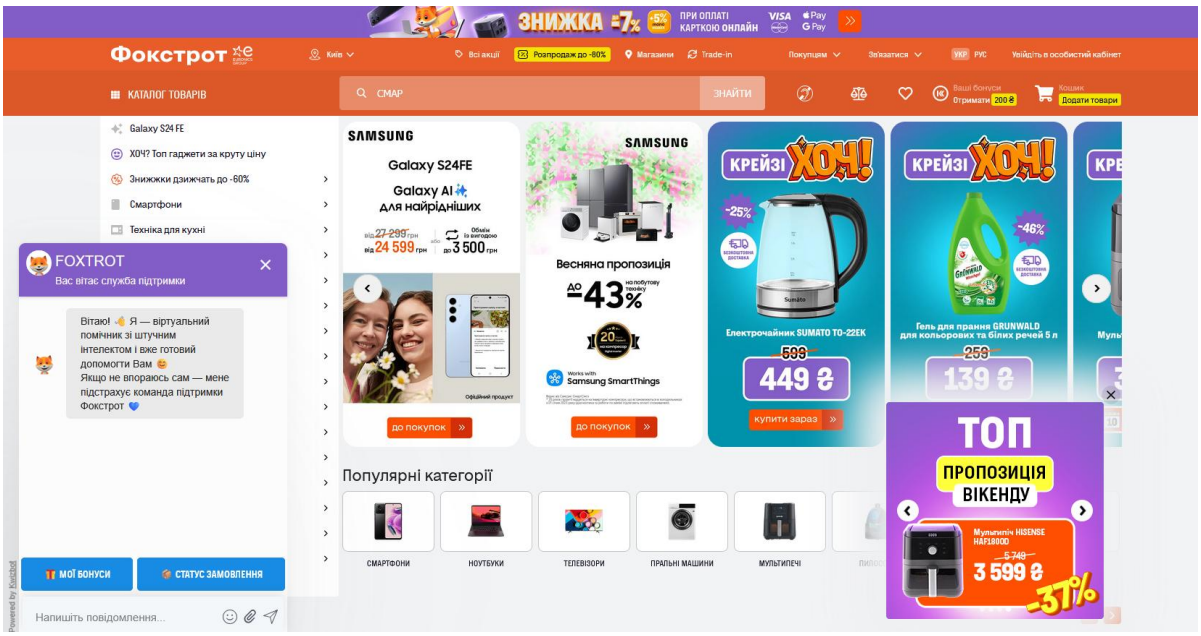

Рисунок 1.3 – Приклад головної сторінки сайту foxtrot.com

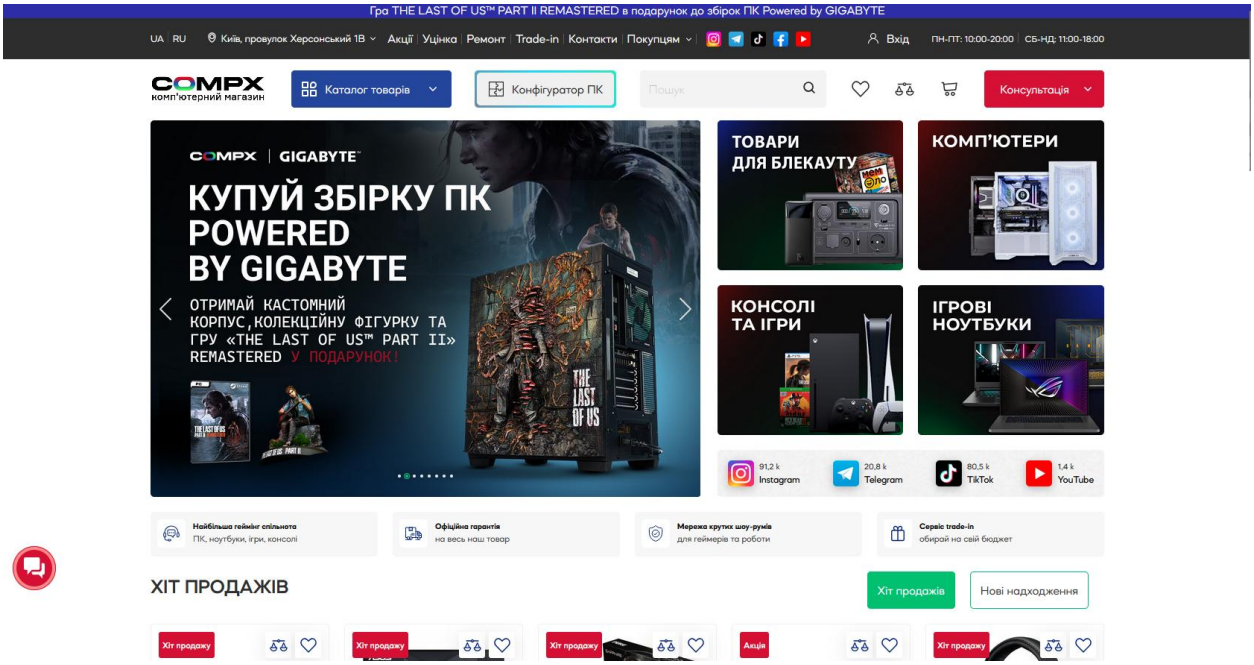

Рисунок 1.4 – Приклад головної сторінки сайту compx.ua

Аналізуючи, можна помітити, що всі сторінки мають такі елементи: пошукова стрічка, категорії, акаунти, функцію кошику, можливість швидкого додавання у кошик, швидкий перегляд товару через карточки. При виборі товарів у конкретних категоріях були наявні фільтри по ціні та моделям [6-8].

Також у магазині «Фокстрот» був реалізований чат-бот, проте він налаштований на лише конкретні питання і при питанням технічного плану одразу перекидає на «живу» службу підтримки, що показано на рисунку 1.5.

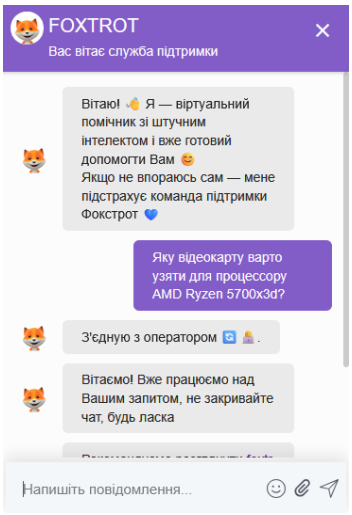

Рисунок 1.5 – Тестування чат-боту від «Фокстрот» [9-10]

1.3 Огляд типових архітектур вебзастосунків

1.3.1 «Моноліт»

Перші вебсервіси будувалися як один великий процес, де маршрути HTTP, бізнес-логіка, робота з базою та рендеринг шаблонів співпрацювали і розгорталися одним контейнером. Така схема отримала назву «монолітна» через як фізичну єдність коду, так і через тісне переплетіння компонентів: зміна моделі в одній частині застосунку часто тягне перекомпіляцію або перезапуск усього сервера. Моноліт зручний на ранніх стадіях розробки проекту, або для невеличких власних проектів. Оскільки структура проекту не роздута на декілька процесів, для розгортання достатньо запустити єдиний контейнер. Проте при масштабуванні з’являються проблеми – чим більше модулів, тим важче підтримувати роботу у команді з кількох розробників, а збій у одній частині коду може блокувати роботу усього застосунку. Також при спробі надати більше потужностей для серверу є проблема

неефективного розподілу ресурсів. Наприклад, під час знижок найбільше навантаження припадає на каталог, проте неможливо виділити нові контейнери на підтримку швидкодії каталогу, не надаючи при цьому більше обчислювальних потужностей і «простоюючим» компонентам [11].

1.3.2 Багатошарові застосунки

У 2000-х було створено шаблон Model-View-Controller. Його суть полягає у чіткому розподілі коду «моноліту» на три частини по ролям. Model відповідає за роботу з даними, описує схеми та інкапсулює запити до СУБД. View – це шар відображення, у якому зберігаються HTML-шаблони або JSON-серіалізація, що повертається клієнту. Controller приймає HTTP-запит, отримує параметри, викликає потрібні моделі й віддає результат у вигляді представлення. Попри те, що MVC часто існує в межах одного контейнеру, сама ідея шарів робить код упорядкованим: змінивши, наприклад, спосіб зберігання файлів, достатньо переписати моделі й не чіпати контролери. Класичний невеличкий проєкт має окремі директорії routes/, controllers/ і models/ – це і є багатошаровий монолітний проект. Перевагою є простота утримання – не потрібна мережа мікросервісів, при цьому легше контролювати і розширювати проект через стандартизовану архітектуру.

1.3.3 Мікросервісна архітектура

Щоб зменшити зчеплення модулів, було створено модель мікросервісів: кожен сервіс виконує єдину бізнес-функцію й має власну базу даних. Наприклад, «Каталог товарів» відповідає лише за читання й оновлення позицій, «Кошик» знає про тимчасові резерви, а «Платіжний шлюз» обробляє оплату. Спілкування між сервісами відбувається або

синхронно через REST/gRPC-запити, або асинхронно через брокер повідомлень. Перевагою є ізольованість кожного елемента – помилка у сервісі рекомендацій не вплине на роботу модуля оформлення замовлень. Кожен модуль масштабується окремо: якщо «Пошук» під навантаженням, піднімаються лише репліки його контейнеру, що дозволяє прискорювати роботу лише тих процесів, що є під навантаженням. Зворотньою стороною такого розподілу по різним контейнерам є операційна складність, оскільки потрібно враховувати необхідність логування, затримку комунікацій між сервісами, що збільшується лінійно від кількості сервісів, що задіяні, а відлагодження помилок потребує трасування з метою вияснення конкретного сервісу, у якому відбувається помилка. Мікросервіси вигідні для великих платформ із незалежними командами, проте надлишкові для проєкту, де підтримкою займається один розробник [12-13].

1.3.4 Serverless-архітектура

Найсвіжіший тренд – відмова від власних постійно запущених серверів. Хмарні провайдери (AWS, Google Cloud, Azure) дозволяють розгортати окремі функції, які прокидаються лише у момент запиту та «засинають» після виконання. Оплата йде за фактичні мілісекунди роботи процесора й об’єм вихідного трафіку, тому під час простою власник не витрачає коштів. У контексті інтернет-магазину це можуть бути, наприклад, функції отримання списку продуктів, створення замовлення, створення цифрового чеку. Масштабування відбувається автоматично – якщо одночасно приходять тисячі запитів, провайдер піднімає стільки контейнерів, скільки потрібно. Головною перевагою є відсутність потреби у утриманні власного кластеру Kubernetes. Недоліки також очевидні – «холодний старт» серверів створює значну додаткову затримку для сервісів із середнім завантаженням, або у яких є необхідність у наданні доволі швидкої відповіді [11, 14].

1.4 Постановка задачі

Таким чином, сучасний інтернет-магазин з використанням чат-боту на основі LLM може дійсно покращити користувацький досвід. Досягнення сьогодення у сфері штучного інтелекту дозволять надавати розгорнуту допомогу користувачам по технічним питанням без залучення живої людини.

Об’єктом роботи є розробка вебплатформи для електронної комерції з інтеграцією чат-боту на основі штучного інтелекту.

Метою роботи є розробка зручної платформи зі спрощеним процесом додавання товарів, системою корзини, чат-ботом для оптимізації надання підтримки користувачам.

Для досягнення мети необхідно вирішити такі завдання:

– провести аналіз систем управління базами даних;
– розробити сторінку адмін-панелі для керування товарами, категоріями та користувачами;
– розробити систему користувачів;
– реалізувати головну сторінку;
– реалізувати відображення товарів;
– розробити пошук товарів;
– розробити фільтрацію товарів;
– розробити систему кошика;
– реалізувати повний цикл від створення товару до покупки;
– реалізувати інтеграцію чат-боту у вікно чату підтримки.

2 ОБҐРУНТУВАННЯ РІШЕНЬ ТА МОДЕЛЮВАННЯ ПРОЄКТУ ВЕБЗАСТОСУНКУ

2.1 Моделювання загального принципу роботи вебзастосунку

Основна ідея вебзастосунку полягає у реалізації повноцінної клієнт-серверної архітектури, що забезпечує миттєву й безпечну взаємодію між відвідувачем і сервером: користувач формує замовлення, а бекенд опрацьовує дані цих замовлень, оновлюючи інформацію на складі, статуси замовлень у реальному часу.

Система підтримує три ролі: зареєстрований клієнт, адміністратор магазину та незареєстрований гість. Для кожної ролі визначено окремий набір прав і обмежень. Щоб гарантувати коректне розмежування доступу, необхідно створити реєстрацію, авторизацію й аутентифікацію на основі даних, що зберігаються у базі даних. Усі паролі мають хешуватися.

Під час моделювання системи необхідно враховувати найтиповіші сценарії взаємодії: перегляд товарів, фільтрація, робота з кошиком, оформлення замовлення, вибір доставки та оплати, а також відстеження статусу. На основі цього має бути спроєктовано структуру бази даних, що включає користувачів, товари, категорії, замовлення.

Адміністратор має доступ до внутрішньої адмін-панелі, де може переглядати, створювати, редагувати та видаляти товари, змінювати статуси замовлень, створювати пресети для більш зручного створення товарів, категорії, а також бачити список зареєстрованих користувачів та надавати їм адмін-права.

Незареєстровані користувачі можуть переглядати головну сторінку сайту, відкривати всі товари, додавати їх у кошик, купувати з доставкою, відстежувати статус замовлення. Незареєстровані користувачі можуть зареєструватися/залогінитися на сайті.

Зареєстровані користувачі можуть переглядати історію покупок, змінювати дані про себе для швидкого заповнення у поля доставки, відкривати всі товари, додавати їх у кошик, купувати з доставкою та відстежувати статус доставки.

Змоделюємо Use Case діаграму для відповідних акторів на сайті з їх можливими діями. Діаграму показано на рисунку 2.1.

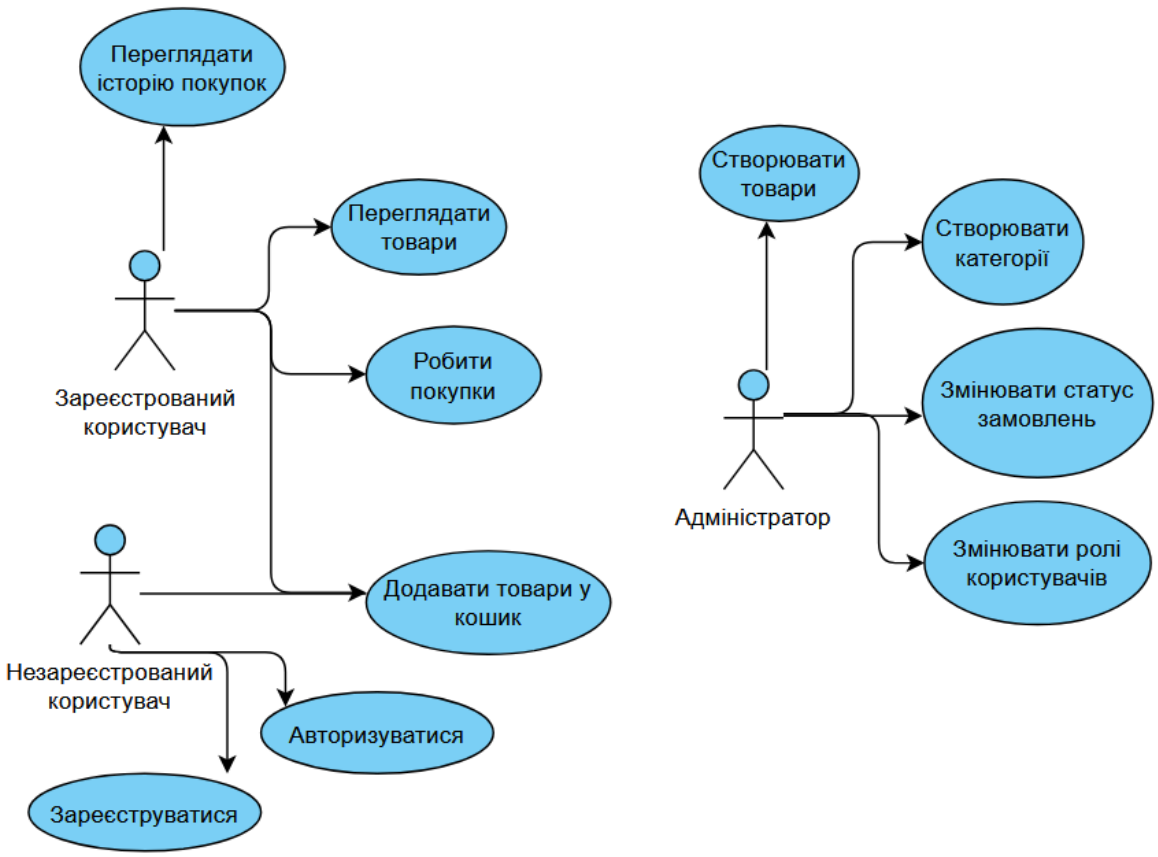

Рисунок 2.1 – Use Case діаграма інтернет-магазину [15]

Має бути реалізовано механізм резервування товарів під замовлення, що дозволяє уникнути перехресних покупки одних і тих же залишків товару різними користувачами.

Змоделюємо діаграму послідовностей для процесу від створення товарів до їх придбання. Діаграму показано на рисунку 2.2.

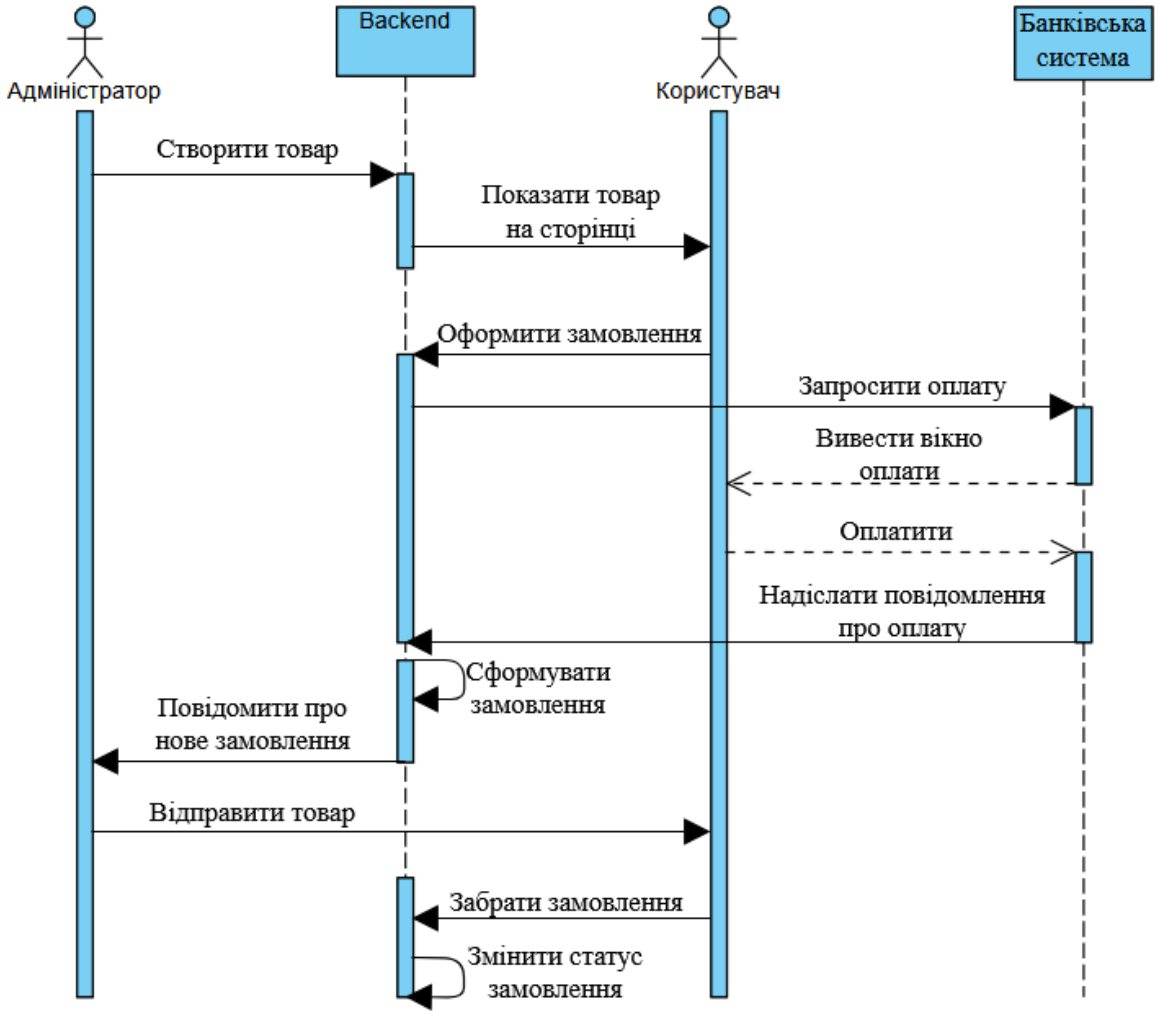

Рисунок 2.2 – Діаграма послідовностей для повного циклу від створення до покупки товару [16-17]

2.2 Архітектура застосунку

Архітектура нашого вебзастосунку базується на багатошаровому моноліті, що є об’єднанням трьох підходів – REST-орієнтованого сервісного шару, шаблону Model-View-Controller на бекенді та компонентно-орієнтованого програмування для фронтенду. Разом вони формують цілісну, передбачувану та підтримувану структуру, у якій кожний рівень виконує чітко окреслені завдання.

Основою взаємодії клієнта і сервера слугує REST. Кожний ресурс системи, наприклад, товар, категорія, користувач, замовлення, має власну адресу у вигляді URL. Будь-яка дія над ресурсом зводиться до стандартних методів HTTP: запит на читання, створення, зміну чи видалення. Наприклад, запит браузера на отримання списку товарів зі знижкою перетворюється на звернення за маршрутом "*/api/products?discount=true*", що повертає JSON-підбірку об'єктів [18].

При проєктуванні серверної частини використаємо шаблон MVC. Основним його плюсом є те, що функціональність кожного елементу відокремлена, що дозволяє утримувати більш організовану файлову структуру та простіше для майбутніх змін. Моделі описують схеми зберігання даних у базі даних і відповідають за валідацію полів, обчислювані властивості та методи, які напряму звертаються до бази. Контролери приймають вхідний HTTP-запит, дістають аргументи та передають їх у сервісний рівень. Сервіси містять бізнес-логіку: під час створення замовлення вони перевіряють наявність товару на складі, формують бронювання, знижують залишок і генерують перший запис історії статусів.

Фронтенд інтернет-магазину буде побудований за принципом компонентно-орієнтованого програмування. Це означає, що весь інтерфейс розділений на невеликі окремі частини – компоненти. Кожен компонент відповідає за певну функцію, наприклад, за відображення товарів, фільтр, форму замовлення, завантаження списку критеріїв або кошик. Кожен компонент містить власний CSS та логіку. Компоненти можна повторно використовувати на різних сторінках, комбінувати один з одним і змінювати незалежно один від одного. Такий підхід також полегшує підтримку та оновлення проєкту – зміну роботи кошика можна виконати лише в одному відповідному компоненті для усіх сторінок, де є доступ до кошику.

Типовий сценарій відбувається таким чином – користувач відкриває головну сторінку, і браузер завантажує кореневий компонент застосунку. Після завантаження бекенд ініціює запит до шляху *"/api/products"*, додаючи

параметри фільтрації, виставлені користувачем. Контролер приймає запит, викликає сервіс, який надає дані з колекції товарів, обчислює акції, ціни зі знижкою та віддає масив об’єктів. Браузер отримує цей масив у вигляді формату JSON, зберігає його у пам’яті й відтворює кожен товар через власний компонент. Якщо користувач натискає «Додати в кошик», браузер надсилає відповідний запит з ідентифікатором товару. Контролер ловить його, надсилає відповідь, оновлює локальний кошик у LocalStorage, якщо користувач навторизований, або у БД, якщо користувач зайшов у свій акаунт. Після переходу до оформлення вміст кошика надсилається одним POST-запитом на /api/orders. Сервер створює документ замовлення, робить записи у таблиці резервів і повертає об’єкт із номером замовлення та початковим статусом «очікує оплати». Сторінка статусу «підписується» на оновлення через періодичні GET-запити й відображає зміни без перезавантаження.

Такий поділ спрощує роботу. Вся логіка роботи з базою даних зосереджена в моделях і сервісах, маршрути описують лише правила HTTP-запитів, а інтерфейс зібраний з невеликих блоків, які можна легко протестувати у відриві від серверу.

2.3 База даних

Майже будь-який вебпроєкт потребує систему для зберігання інформації про користувачів, продукти та замовлення. Загальний термін «база даних» означає організований набір даних, що зберігається на диску або в хмарі. Системи управління базами даних, або СУБД, допомагають керувати цим сховищем. Вони надають інструменти для створення таблиць або колекцій, індексів, резервних копій та визначення прав доступу. Правильне проєктування структури даних є важливим кроком – якщо вибрати неправильний формат, згодом буде набагато складніше впровадити

систему. Дані зазвичай зберігаються в таблицях з рядками і стовпчиками, у яких створюються індекси для прискорення пошуку.

У світі існує кілька типів баз даних. Найпоширеніша – реляційна, в якій інформація представлена таблицями, пов’язаними зв’язками «один-до-одного», «один-до-багатьох» або «багато-до-багатьох». Зв’язки реалізуються за допомогою зовнішніх ключів, а описи та маніпуляції здійснюються мовою SQL. Реляційні СУБД, такі як MySQL або PostgreSQL, популярні завдяки надійним механізмам операцій, строгим схемам зберігання даних та інструментам цілісності. Альтернативою є нереляційні сховища, або NoSQL. Вони не мають обов’язкової табличної структури, тому надають можливість зберігати різні типи документів або записів одночасно. NoSQL підходять для роботи з великими обсягами неструктурованих даних і легко масштабуються. Найвідоміші – MongoDB, Redis та Cassandra.

Для нашого інтернет-магазину було обрано MongoDB. Це документоорієнтована база даних, в якій дані зберігаються не в таблицях, а в документах BSON типу JSON. Це дає можливість гнучко поєднувати інформацію про товар, його категорії та знижки в одному записі без складних SQL-з’єднань. Якщо бізнес-логіка змінюється і потрібно додати нове поле, наприклад, додаткові етикетки або фотографії, структуру можна швидко змінити без тривалої міграції. Це важливо для інтернет-магазину з широким асортиментом товарів: нові параметри можна додавати без необхідності наявності такого ж поля у інших об’єктах такого типу, і схема залишається гнучкою.

Ще однією перевагою MongoDB є її масштабованість. База даних підтримує шардинг: документи можна розміщувати на декількох серверах, рівномірно розподіляючи навантаження. Це особливо актуально, коли кількість замовлень зросте або каталог товарів почне містити сотні тисяч позицій. Якщо під час акції раптово зростає навантаження, кластер шардів дозволяє додати ще один вузол, не зупиняючи роботу сервісу. Гнучкість схеми, відсутність обов’язкових операцій та горизонтальне масштабування

роблять MongoDB найбільш практичним вибором для системи електронної комерції. Документи в MongoDB можуть бути вкладені один в одного. Наприклад, документ «користувач» може містити набір адрес доставки або зміст кошику, без необхідності створювати окремі таблиці. Це полегшує читання: більшість задач до інтернет-магазину по типу «показати користувачу замовлення у його кошику» або «показати картку товару з усіма характеристиками» виконуються одним запитом без зв'язування таблиць командою JOIN як у SQL БД.

Отже, з огляду на динаміку полей у об'єктах електронної торгівлі, особливості товарного каталогу та вимоги до масштабування, документна модель MongoDB краще відповідає нашим потребам, ніж традиційна реляційна, оскільки гнучка схема дає змогу швидко реагувати на зміну умов збереження полів, а розподілена архітектура забезпечує стабільну роботу навіть при різкому зростанні трафіку.

Таблиця 2.1 – Опис майбутніх таблиць у базі даних MongoDB

Номер	**Таблиця**	**Опис**
1	**2**	**3**
1	categories	Таблиця для зберігання категорій
2	orders	Таблиця для зберігання інформації про замовлення
3	presets	Таблиця для зберігання пресетів
4	products	Таблиця для зберігання товарів
5	users	Таблиця для зберігання інформації про користувачів

Окрім полів, що будуть додаватися адміністратором за необхідністю, деякі таблиці матимуть поля, що будуть потрібні для об'єктів такого типу завжди. Такі поля показано у таблицях 2.2 – 2.6.

Таблиця 2.2 – Обов’язкові поля для об’єктів таблиці products

Поле	**Тип значення**	**Значення**
name	String	Зберігання імені товару
price	Number	Зберігання ціни товару
images[]	[String]	Зберігання масиву з адресами картинок на диску
category	ID категорії	Зберігання ID категорії товару
preset	ID пресету	Зберігання ID імені товару
characteristics[]	{name: String, value: String}	Зберігання полів характеристик і їх значень
stock	Number	Кількість на складі/у наявності
discount	Number	Ціна зі знижкою
hidden	Boolean	Відмітка для приховування товару з каталогу

Таблиця 2.3 – Обов’язкові поля для об’єктів таблиці categories

Поле	**Тип значення**	**Значення**
name	String	Зберігання назви категорії
parentCategory	ID іншої категорії	Зберігання ідентифікатору батьківської категорії
createdAt	Date	Дата створення категорії

Таблиця 2.4 – Обов’язкові поля для об’єктів таблиці orders

Поле	**Тип значення**	**Значення**
1	**2**	**3**
fullname	String	Зберігання назви категорії
phone	String	Зберігання номеру користувача
deliveryMethod	String	Тип доставки (оператор)
deliveryInfo	String	Адреса доставки

Продовження таблиці 2.4

1	2	3
user	ID користувача	ID користувача
status	String	Статус замовлення
items	[{ID продукта, кількість, ціна}]	Товари у замовленні
paymentMethod	String, enum {"cod", "card"}	Тип оплати за замовлення
paid	Boolean	Позначка «оплачено»

Таблиця 2.5 – Обов’язкові поля для об’єктів таблиці presets

Поле	**Тип значення**	**Значення**
name	String	Ім’я пресету
category	ID категорії	Категорія, що буде підтягуватися до товару при виборі пресету
characteristics	[{name: String, defaultValue: String}]	Заготовлені поля зі значеннями, що будуть додаватися до полів товару при виборі пресету

Таблиця 2.6 – Обов’язкові поля для об’єктів таблиці users

Поле	**Тип значення**	**Значення**
name	String	Ім’я користувача
phone	String	Номер користувача
email	String	Електронна пошта
passwordHash	String	Пароль у зашифрованому вигляді
role	String, enum: {"user", "admin"}	Роль користувача
cart	[ID товару]	Кошик користувача

2.4 Організація процесу покупки

Необхідно провести деякі уточнення процесу покупки через велику кількість розвитків подій, що є цілком нормальними і обов’язково стануться. Наприклад, придбання якщо користувач не був зареєстрований або якщо він відмовився від покупки, або товар вже викупили інші покупці, коли він ще був у корзині у поточного користувача до переходу до оплати. Опрацювання усіх можливих розвитків є найважливішою частиною, бо у разі помилки магазин понесе репутаційні збитки і ймовірність повторної покупки у разі придбання вже неіснуючого товару зводиться до нуля.

Спочатку змоделюємо процеси при додаванні у кошик. Запис даних кошика відбувається у два етапи: локально на клієнті та у базі даних. Поки відвідувач не увійде в систему, дані кошика зберігаються у локальному сховищі браузера. Як тільки користувач авторизується, фронтенд відправляє вміст локального кошика на сервер, після чого сервер створює або оновлює запис кошика у даних користувача. Будь-які інші зміни типу додавання, видалення або зміна кількості, негайно надсилаються до бекенду, і сервер підтверджує результат операції відповіддю. Щоб гарантувати, що всі відкриті вкладки користувача мають однаковий кошик, сервер надсилає команду на перевірку *“cartUpdated”* після кожної успішної операції. Клієнт, отримавши цю команду, порівнює свій локальний кеш кошика з даними, надісланими сервером, і, якщо вони відрізняються, синхронізує власне сховище та оновлює інтерфейс. Таким чином, на одній і тій же сторінці, відкритій, наприклад, на комп’ютері і мобільному телефоні, миттєво відображається доданий товар, незалежно від того, з якого пристрою він був обраний. Таким же чином система підтягує введені в особистому профілі або за минулої покупки дані доставки при оплаті товару у випадку, якщо користувач був авторизований до переходу до вікна оплати. Принцип роботи показано на рисунку 2.3.

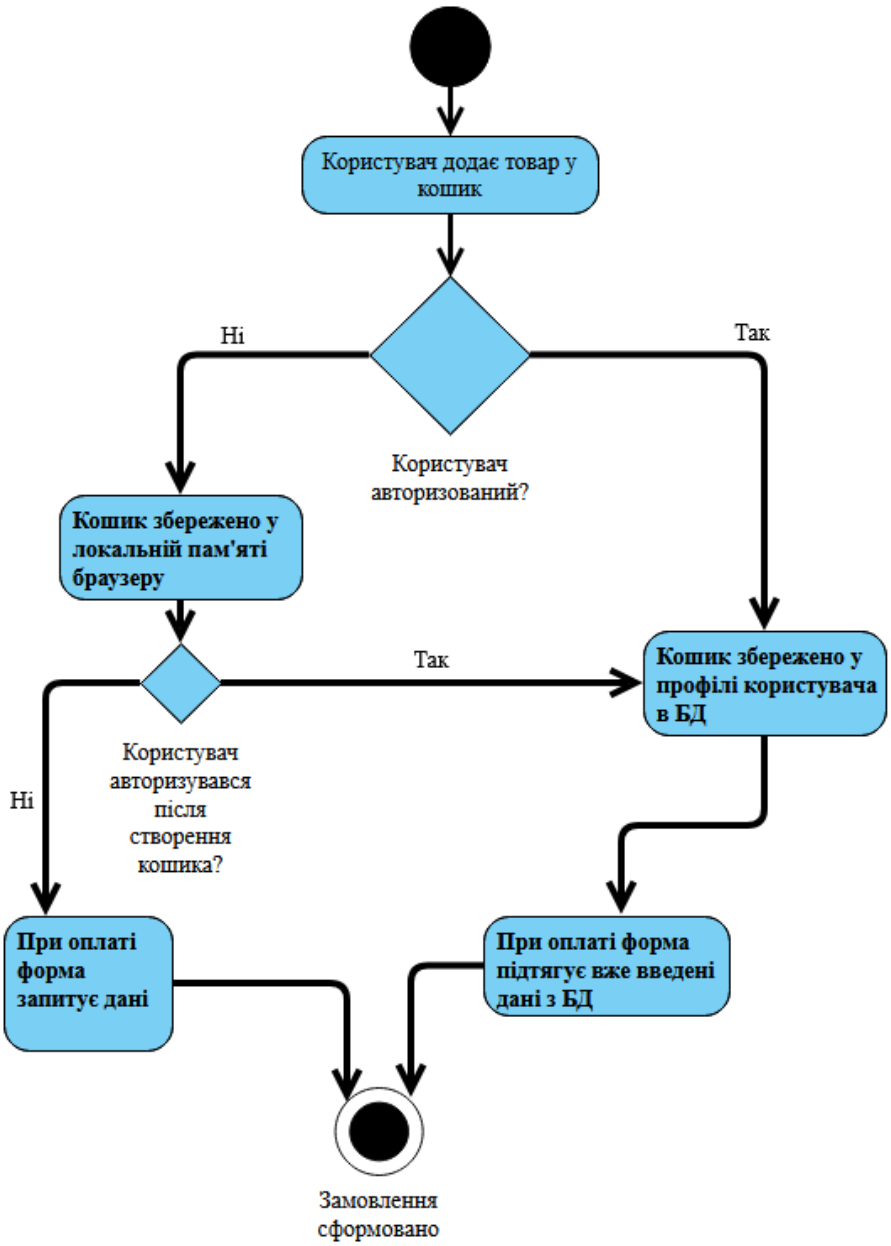

Рисунок 2.3 – Діаграма діяльності системи для авторизованих і неавторизованих користувачів [19]

Система резервування товарів працює схожим чином. Вона потрібна щоб перевіряти, що доданий до кошика товар фактично ще залишається доступним для того ж клієнта на момент оформлення замовлення та оплати, але при цьому не блокує товар на довгий час, якщо користувач передумав. Коли відвідувач натискає на кнопку «Додати до кошика», клієнтська частина надсилає POST-запит за маршрутом /api/reservations сервера. Контролер виконує команду findOneAndUpdate, зменшує поле inStock в документі

продукту і створює новий запис в колекції бронювань. Цей запис містить ідентифікатор товару, кількість, ідентифікатор користувача та пункт expiresAt, що дорівнює десять хвилин. Поки транзакція не підтверджена, жоден інший користувач не зможе додати цей товар у кошик, якщо його немає у достатній кількості, чим вирішується проблема, коли кілька людей одночасно намагаються купити один і той самий товар.

Змоделюємо порядок дій системи при додаванні товару до кошика та до покупки на рисунках 2.4 – 2.5.

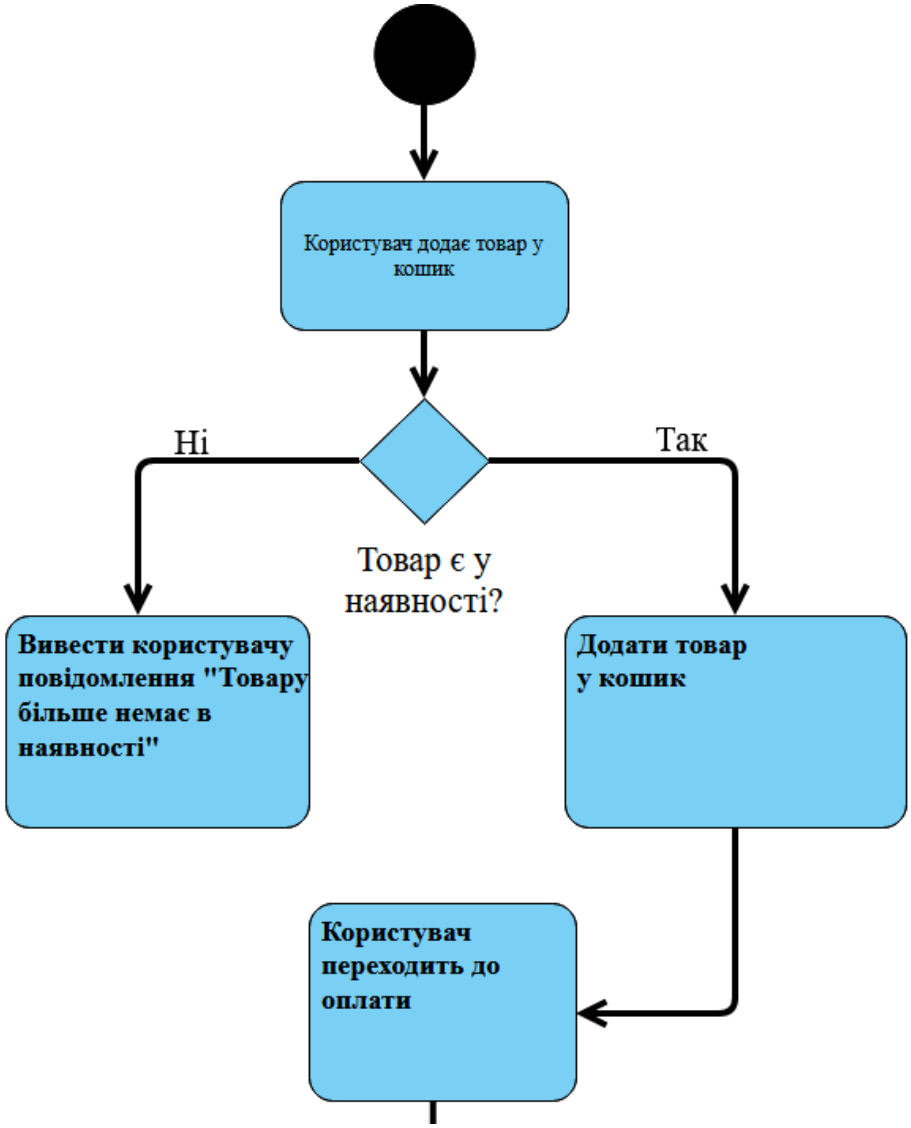

Рисунок 2.4 – Діаграма активності для системи резервування [20]

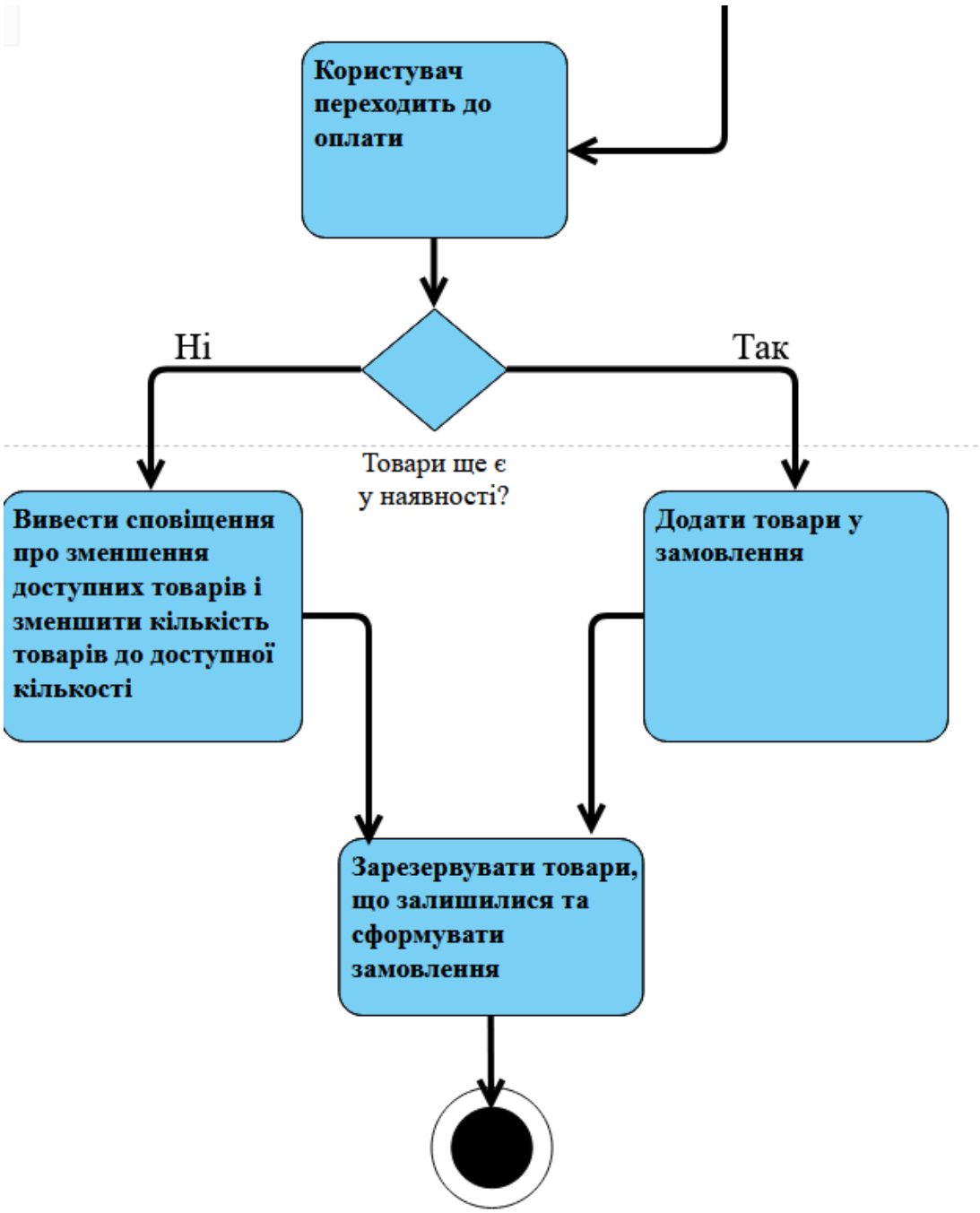

Рисунок 2.5 – Продовження діаграми активності для системи резервування

Коли користувач переходить до поля доставки і оплати, фронтенд надсилає запит до /api/orders зі списком бронювань. Сервіс замовлень перевіряє, що жоден з товарів ще не був викуплений, підтверджує бронювання та додає його до замовлення. Поле зарезервовано в документі товару вираховується з відповідною кількістю. Після успішної оплати

бронювання позначається як «завершене» і більше не бере участі в механізмі автоматичного повернення у загальну кількість.

Якщо користувач закриває вкладку або залишає кошик на довгий час, спеціальний процес у фоновому режимі раз на хвилину аналізує збір замовлень за умовою *“expiresAt < now()”* і звільняє прострочені позиції: збільшує поле *“inStock”* товарів на відповідну кількість, видаляє замовлення і надсилає WebSocket-сигнал на вкладки клієнта, де візок прострочений. Таким чином система гарантує, що вибраний товар не перехопить інший клієнт, навіть якщо той товар швидко розкуповується.

2.5 Моделювання адмін-панелі

Адміністративна частина інтернет-магазину повинна бути розроблена як окремий односторінковий застосунок, що завантажується після успішної перевірки ролі. У момент переходу браузер надсилає запит на сервер, а middleware перевіряє, чи належить поточний ідентифікатор сесії користувачу з правами адміністратора. У разі позитивного результату сервер повертає сторінку, інакше переспрямовує на сторінку авторизації. Основа інтерфейсу – сторінка, на якій зліва колонкою розташовано навігаційне меню, а праворуч відкрита робоча область. Маршрутизація всередині панелі здійснюється браузером і не потребує повторних звернень по HTML. Одразу після завантаження сторінка виконує паралельні запити до серверу, щоб отримувати дані. Доступно створення та редагування товарів, категорій. Також є пресети для більш зручного створення товарів – можна задати поля зі значеннями та категорії, що автоматично будуть підставлятися товару при виборі цього пресету, що значно зменшує потрібність багаторазового заповнення одних і тих же полів. Є список користувачів, де можна надавати користувачам права адміністратора. Меню категорій, керування товарами та користувачами мають пошукову панель.

2.6 Моделювання головної сторінки

Головна сторінка інтернет-магазину мусить мати певні елементи, що стали стандартом для електронної комерції, а саме: стрічку пошуку, для якої при введенні запиту результати виводяться у вигляді списку карток товарів з можливістю переходу до сторінки товару, швидкий доступ до корзини через модальне вікно з можливістю перегляду і редагування товарів у корзині, доступ до власного акаунту, а також перехід до панелі адміністрації при наявності відповідної ролі.

Нижче «хедеру» має бути список товарів у вигляді карток, що мають перше фото товару, ціну, наявність, а також кнопку для швидкого додавання товару у кошик.

Також на головній сторінці має бути розміщено кнопку для спілкування з чат-ботом.

Кожен товар мусить мати свою сторінку, що автоматично генерується і дозволяє переглянути назву, фото, ціну, наявність і усі параметри товару. Також на цій сторінці також мають бути доступними елементи кошика та профілю аналогічно до основної сторінки.

2.7 Інтеграція з API логістичних операторів

Для форми покупки буде необхідно впровадити інтеграцію з API логістичних сервісів, зокрема з «Новою Поштою», для спрощення вибору доставки для користувачів під час оформлення замовлення, зменшення кількості помилок при введенні даних вручну і підвищення швидкості заповнення форми. Інтерфейс повинен автоматично підвантажувати доступні міста та пункти видачі в них при початку введення у відповідні поля.

3 РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

3.1 Огляд загальної архітектури вебзастосунку

Після завершення моделювання діаграм у розділі 2, наступним кроком є безпосередня розробка самого інтернет-магазину. У проєкті обрано стек Node.js та JavaScript. Вебзастосунок надаватиме користувачам зрозумілий інтерфейс для пошуку товарів, формування кошика й оформлення замовлень, а адміністратору – засоби керування каталогом і замовленнями. Сервер взаємодіє з документною базою даних MongoDB через модуль інтеграції Mongoose, отримує, зберігає й оновлює інформацію про товари, категорії, користувачів і статуси. На рівні моделей реалізовано валідацію полів і резервування товарів.

При створенні інтерфейсу враховано три ролі: гість, зареєстрований клієнт і адміністратор магазину. Усі вони працюють із тим же самим сайтом, проте мають інтерфейс з певними відмінностями. Розділення прав забезпечується сесійною автентифікацією через httpOnly-cookie: після авторизації сервер записує ідентифікатор сесії, а захист middleware дозволяє чи забороняє доступ до певних маршрутів. Фронтенд динамічно підвантажує тільки ті модулі, що відповідають поточній ролі, завдяки чому адміністративна панель не потрапляє у публічний доступ. У якості інструментів розробки було обрано Visual Studio Code, оскільки він пропонує швидкий і зручний аналіз коду та багато необхідних бібліотек. Схеми даних перевірялися у MongoDB Compass, а самі запити у додатку Postman [21]. Сайт розгортається локально у Express-контейнері командою npm run dev.

Лістинг 3.1 Скрипт команди запуску сайту:

```
{
  "name": "diploma",
  "version": "1.0.0",
```

```
  "main": "index.js",
  "scripts": {
    "dev": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "cors": "^2.8.5",
    "dotenv": "^16.5.0",
    "express": "^4.21.2",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^8.10.0",
    "multer": "^1.4.5-lts.2",
    "openai": "^4.98.0"
  }
}
```


Увесь сайт розгортається локально у середовищі Node.js за допомогою фреймворку Express. Для зручності запуску в package.json проєкту визначено скрипт dev, який виконує команду node server.js. Це означає, що коли користувач виконує в терміналі npm run dev, фактично запускається файл server.js, який запускає сервер вебсторінок.

У package.json також вказано залежності, які необхідні для роботи сайту. Наприклад, express – це ядро вебсерверу. Mongoose потрібен для підключення до бази даних MongoDB і роботи з нею. Multer використовується для збереження фото товарів на локальному диску, які завантажують адміністратори. Bcrypt дозволяє хешувати паролі, jsonwebtoken

керує авторизацією користувачів, а openai забезпечує інтеграцію з API ChatGPT.

Коли запускається server.js, Express-сервер налаштовується для обробки запитів. Підключається до MongoDB за допомогою Mongoose, а якщо підключення проходить успішно, то у консолі Visual Studio Code виводиться повідомлення «Підключено до MongoDB». Потім сервер вмикає обробку JSON-запитів, вмикає доступ до вебресурсу із зовнішніх джерел, а також робить доступною статичну папку public для зберігання зображень товарів.

Далі до застосунку підключаються різні модулі маршрутів, наприклад, /api/products для роботи з товарами, /api/orders для замовлень, /api/chat для інтеграції з OpenAI. Усі ці маршрути обробляються через відповідні файли в папці routes/, де описана бізнес-логіка кожного з них. Наприклад, routes/products.js дозволяє отримувати список товарів, створювати нові, редагувати або видаляти існуючі.

У кінці server.js викликається app.listen, що запускає сервер на певному порту 5000. Таким чином, сайт стає доступним локально за адресою http://localhost:5000, і користувач може взаємодіяти з ним у браузері.

У кодовій базі проєкт чітко поділено на дві великі папки – admin і main. У серверній частині окремі підкаталоги models, scripts та routes, що полегшує навігацію та подальший рефакторинг. Клієнтська частина згрупована за функціональністю або окремими модулями, кожна сторінка містить власні компоненти і стилі, завдяки чому новий функціонал можна додавати ізольовано, не зачіпаючи решту інтерфейсу. Така побудова відповідає принципам патерну Model-View-Controller, що поділяє додаток на файли для контролю даних, інтерфейсу і логіку, і є багатошаровим монолітом, відповідним для проекту, що керується невеличкою кількістю людей.

Розглянемо файлову структуру проєкту, що показана на рисунку 3.1.

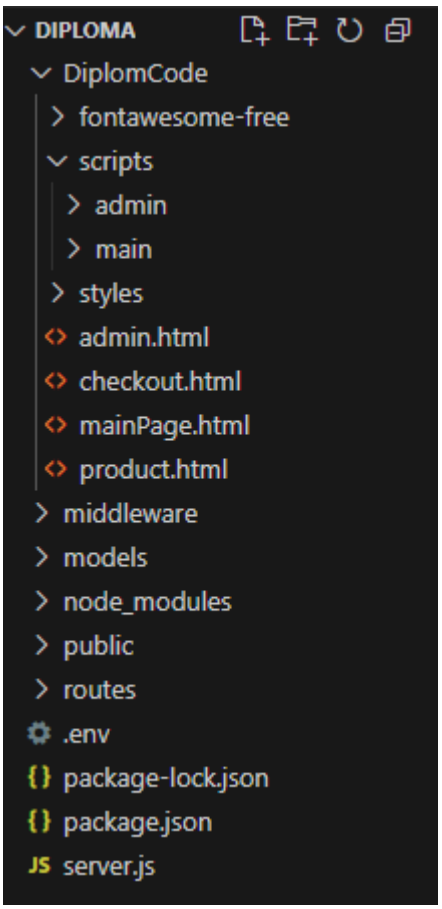

Рисунок 3.1 – Файлова структура проєкту

admin.html – файл з розміткою адмін-панелі. checkout.html – розмітка для сторінки оплати. mainPage.html – розмітка основної сторінки. Product.html – розмітка сторінки для перегляду конкретного товару.

Папка fontawesome-free містить іконки та шрифту для декорації сайту додатковими візуальними елементами. Папка styles/ містить css-файли для усіх сторінок розмітки, а також файли global.css і cartStyle.css для уточнення стилю таких утилітарних функцій як сповіщення, чат-бот і кошик.

Папка scripts поділена на папки admin та main з модулями логіки для адміністративної панелі та основного сайту відповідно. Так, у папці admin наявні JS-модулі для логіки роботи меню панелі, реалізована логіка керування замовленнями, користувачами, пресетами, товарами та їх категоріями. У папці main знаходяться модульні елементи для логіки кнопок основної сторінки та сторінки окремого товару, логіка кошику,

універсальних сповіщень про певні події (товар закінчився при спробі придбати його тощо), логіка оплати та коректної роботи меню чат-боту.

Папка models/ містить у собі схеми для передачі у СУБД MongoDB через модуль Mongoose. Наявні схеми для таких об’єктів як «категорія», «замовлення», «пресет», «продукт» та «користувач».

Папка middleware містить код для обробки авторизації користувачів.

Папка public\uploads є локальним сховищем для збереження зображень товарів.

.env – файл конфігурації, що містить глобальні системні змінні, такі як порт, адреса підключення до СУБД MongoDB та API-ключ для інтеграції зі стороннім LLM-ботом OpenAI для забезпечення просунутої логіки чат-боту.

server.js – ключовий файл серверної частини застосунку. У ньому відбувається підключення до БД, конфігурація Express (фреймворк для Node.js зі швидким налаштуванням вебсервісів), підключення шляхів (routes) та запуск сервера.

3.2 Програмна реалізація головної сторінки

Головна сторінка вебзастосунку – це стартова точка взаємодії користувача з інтерфейсом магазину. Вона відображає перелік доступних товарів, дозволяє шукати їх, додавати у кошик, а також відкривати сторінки з детальним описом. Для цього застосовано динамічне завантаження контенту через API. Також на головній сторінці є доступ до акаунту, а також до кошику з можливістю керування кількістю товарів, а також швидким переходом до оплати. Товари завантажуються на головну сторінку автоматично, без необхідності виставляти щось у розмітці сторінки, за допомогою JavaScript.

На рисунку 3.2 показано вигляд головної сторінки.

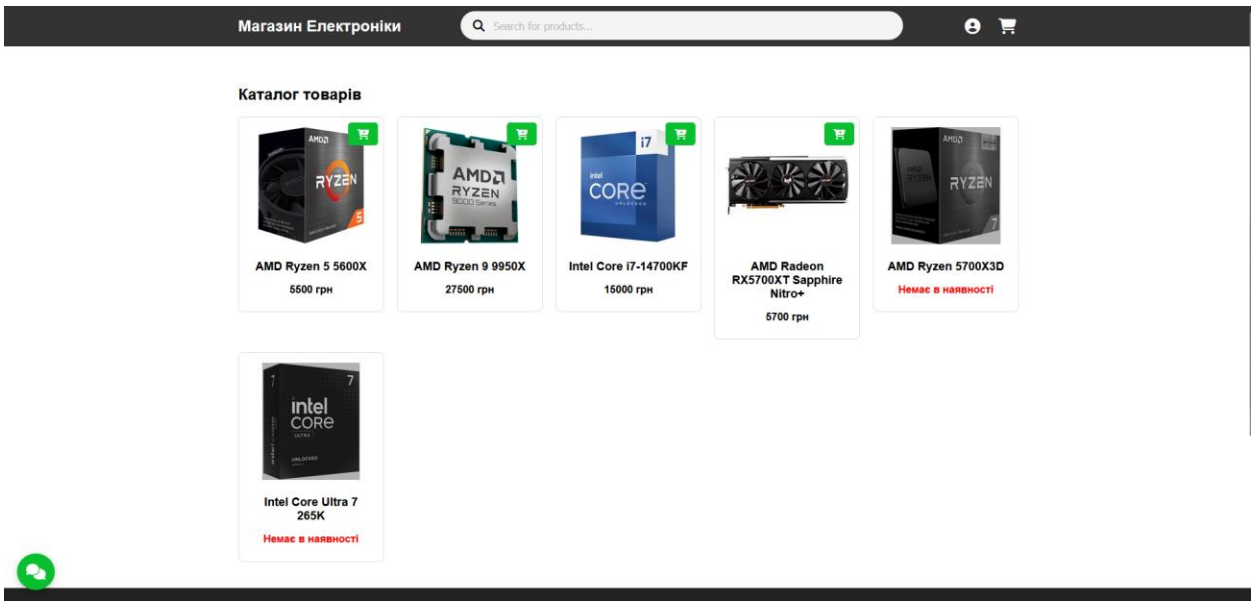

Рисунок 3.2 – Вигляд головної сторінки

Також є автоматичне формування дизайну картки товару в залежності від кількості – при наявному товарі пишеться його ціна, фото залишається кольоровим і товар можна додати у кошик. У випадку, якщо товар закінчився, то колір зображення на картці буде монохромним, кнопки швидкого додання у кошик не буде, а замість цінника буде відповідний напис «Немає в наявності».

Лістинг 3.2 Реалізація алгоритму завантаження товарів:

```
fetch("http://localhost:5000/api/products")
  .then(res => res.json())
  .then(products => {
    const list = document.getElementById("product-list");
    list.innerHTML = products
      .filter(p => !p.hidden) // Не показувати приховані товари
      .map(renderProductCard)
      .join("");
  });
```

Лістинг 3.3 Реалізація генерації карток товарів:

```
const inStock = Number(product.stock) > 0
 return `
  <a href="product.html?id=${product._id}" class="product-card-link">
   <div class="product-card">
    <div class="product-image-wrapper ${!inStock ? 'out-of-stock' : ''}">
     <img src="http://localhost:5000${product.images?.[0] || '/placeholder.jpg'}" alt="${product.name}">
    </div>
    <h3>${product.name}</h3>
    <p class="price" style="color: ${inStock ? 'black' : 'red'};">
     ${inStock ? `${product.price} грн` : 'Немає в наявності'}
    </p>
    ${inStock ? `
        <button class="add-to-cart" onclick="event.preventDefault(); event.stopPropagation(); handleAddToCart('${product._id}', ${product.stock})">
         <i class="fa-solid fa-cart-plus"></i>
        </button>
     ` : ''}
   </div>
  </a>
 `;
```

3.3 Пошук товарів

Пошук товарів є базової функцією, що забезпечує зручність користування користувачу. Пошук товарів реалізовано за .допомогою

інтеграції системи фільтрів у GET-запити. При введенні користувачем запиту в полі пошуку, фронтенд формує HTTP-запит до сервера з параметром, який містить текст запиту (рис. 3.3).

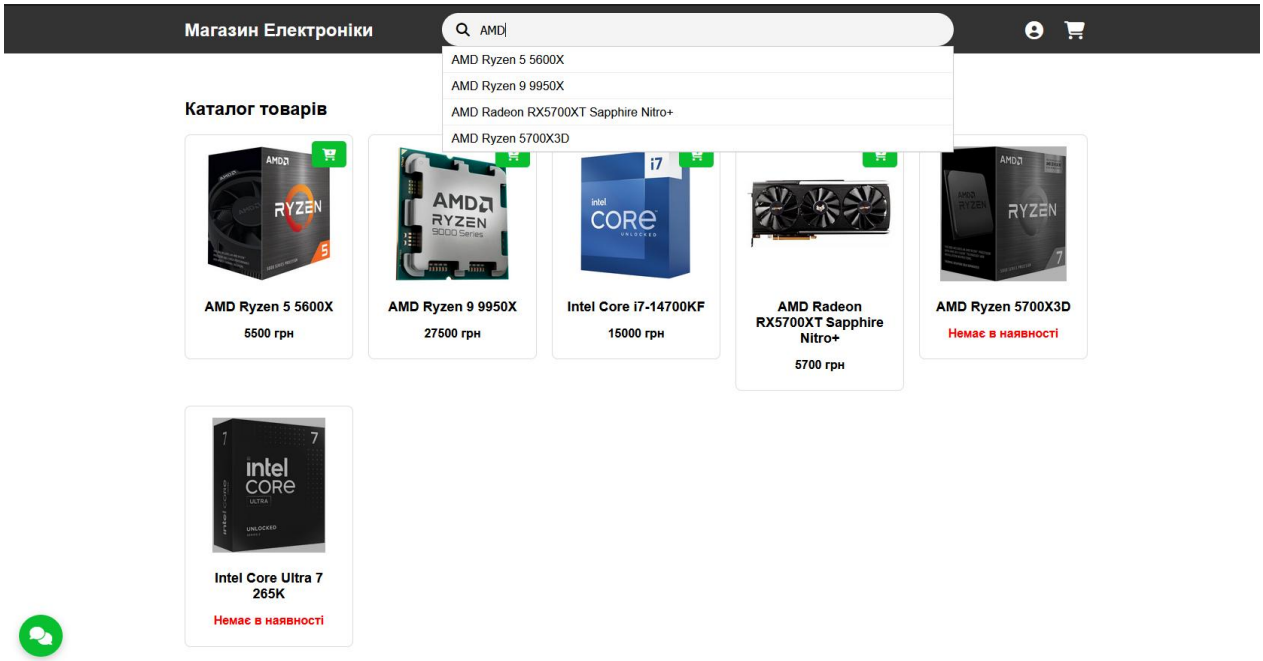

Рисунок 3.3 – Пошук товару за назвою

Лістинг 3.4 Реалізація пошуку товарів:

```
if (req.query.search) {
filter.name = { $regex: req.query.search, $options: "i" };
}
searchInput.addEventListener("input", async () => {
const query = searchInput.value.trim();
if (!query) {
searchResults.innerHTML = "";
searchResults.style.display = "none";
return;
}
try {
const res = await
```

```
fetch(`${API_URL}/products?search=${encodeURIComponent(query)}`);
const products = await res.json();
if (!Array.isArray(products)) throw new Error("Некоректна відповідь сервера");
if (products.length === 0) {
searchResults.innerHTML = "<div style='padding: 8px;'>Нічого не знайдено</div>";
searchResults.style.display = "block";
return;
}
searchResults.innerHTML = products.map(p => `
<a href="product.html?id=${p._id}" target="_blank">${p.name}</a>
`).join("");
searchResults.style.display = "block";
} catch (err) {
console.error("Помилка пошуку:", err);
searchResults.innerHTML = "<div style='padding: 8px;'>Сталася помилка</div>";
searchResults.style.display = "block";
}
});
```

3.4 Реалізація роботи з кошиком

Інтерфейс взаємодії з кошиком побудовано навколо двох основних сценаріїв: швидкого додавання товару з головної сторінки та повноцінної взаємодії з кошиком через модальне вікно. Так користувач може одразу додати товар до кошика з картки продукту або переглянути і відредагувати всі додані товари у будь-який момент, не покидаючи поточної сторінки.

На кожній картці товару, що відображається на головній сторінці, якщо товар є в наявності, з’являється зелена кнопка із іконкою кошика. При натисканні викликається функція, що перевіряє поточну кількість одиниць цього товару в кошику й співвідносить її з полем stock, що відповідає за кількість наявних товарів. Якщо кількість товару у кошику перевищує залишок на складі – виводиться сповіщення, і товар не додається. Після додання товарів лічильник біля іконки кошика змінюється, показуючи кількість унікальних товарів у ньому (рис. 3.4).

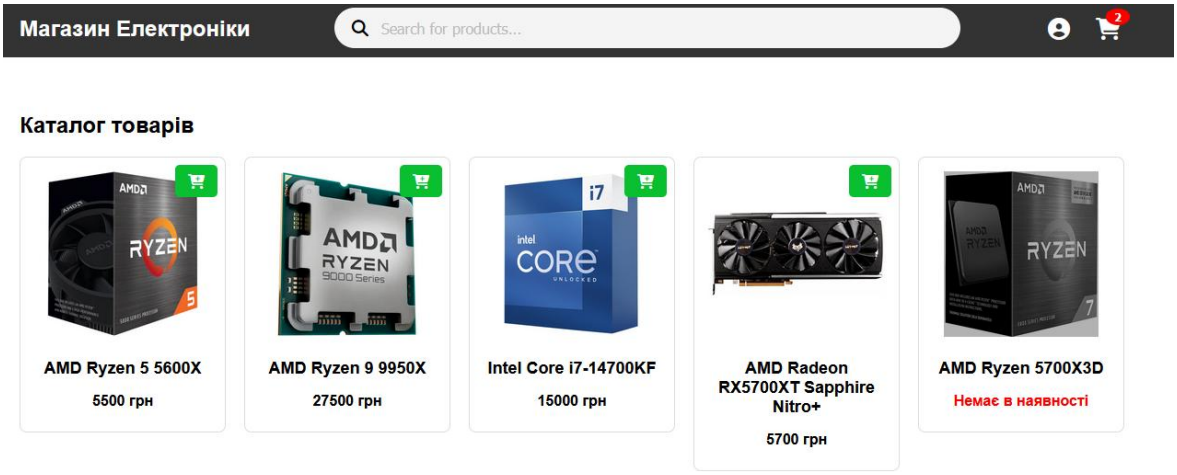

Рисунок 3.4 – Декілька товарів було додано у кошик

Після додання товару можна відкрити кошик та відредагувати кількість товару у ньому. У неавторизованих користувачів дані кошику зберігаються локально у браузері. Це дозволяє зберігати обрані товари навіть після перезавантаження сторінки або закриття вкладки. Коли користувач виконує вхід до свого облікового запису, система виконує синхронізацію локального кошика з кошиком, збереженим на сервері. Після успішної авторизації клієнтська частина викликає запит на /api/cart/merge, передаючи поточний вміст локального кошика. Сервер отримує ці дані, об’єднує їх із тими, що вже зберігалися у базі даних для цього користувача, і зберігає оновлений варіант. Після цього локальний кошик очищується, і далі кошик зберігається у базі даних у профілі користувача.

Варто відзначити, що якщо користувач спробує додати більше товарів, ніж є у наявності, як через швидке додавання, так і через редагування кількості у кошику, то він отримає повідомлення про це. Таким чином система не дасть користувачу купити більше товару, аніж є у наявності (рис. 3.5-3.6).

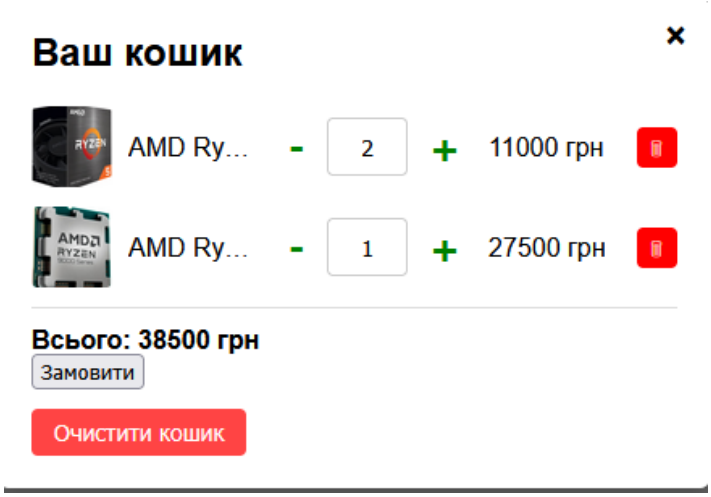

Рисунок 3.5 – Перегляд кошика через модальне вікно

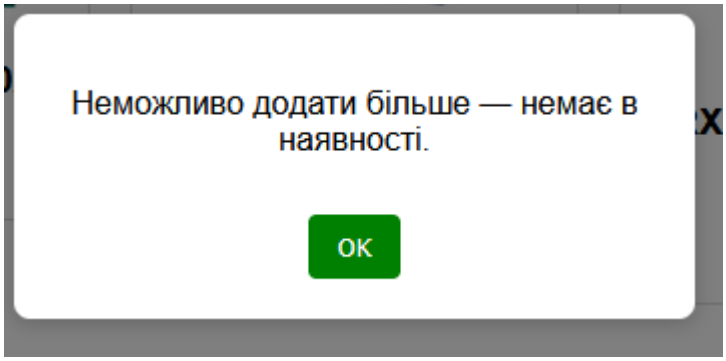

Рисунок 3.6 – При спробі додати більше наявного користувач отримує сповіщення

3.5 Реалізація окремої сторінки товару

Окрема сторінка товару у вебзастосунку реалізована як самостійний HTML-файл з динамічним завантаженням інформації про товар за його

ідентифікатором. Вона відкривається при переході за посиланням виду product.html?id. Після завантаження сторінки скрипт зчитує параметр id з URL-адреси та надсилає запит до серверного маршруту /api/products/:id. Отримані дані (назва, зображення, ціна, усі інші поля) вставляються в сторінку для відображення. Фото обрізається до розмірів вікна передперегляду. Якщо товар не знайдено або сталася помилка, відображається відповідне повідомлення. Також товар можна додати у кошик у випадку якщо він доступний (рис. 3.7).

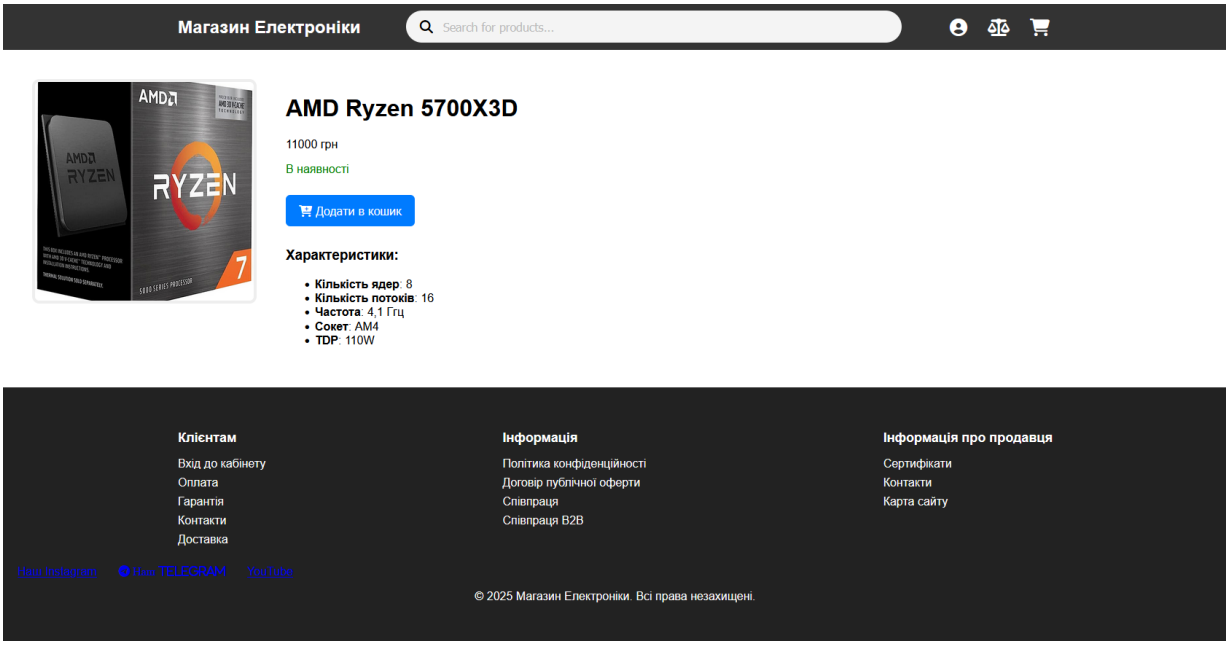

Рисунок 3.7 – Сторінка товару

Для сторінки перегляду товару було створено функціонал, пов’язаний із комфортним переглядом зображень, що відповідає сучасним стандартам інтернет-магазинів. Якщо у товару більше одного зображення, то на сторінці товару відображається повноцінна міні-галерея для передперегляду з можливістю перемикання між фото, як натискаючи по «стрілкам» по бокам від фото, так і при виборі міні-фото знизу. Поточне фото виділяється у міні-галереї синім маркером. Також наявний номер вибраного фото (рис. 3.8).

Рисунок 3.8 – Міні-галерея зі швидким переглядом [22]

У випадку, якщо товар відсутній, то користувач не матиме змоги додати його у кошик, а також буде написано «Немає в наявності» (рис. 3.9).

Рисунок 3.9 – Сторінка товару, що закінчився

Крім того, реалізовано можливість відкрити фото у збільшеному вигляді у модальному вікні. Це вікно також підтримує перемикання фото за допомогою стрілок, як в інтерфейсі, так і клавішами клавіатури. Таким чином, реалізована зручна взаємодія з великою кількістю фотографій товару, що важливо для підтримки конкурентоспроможності та підвищення зручності користування сайтом (рис. 3.10).

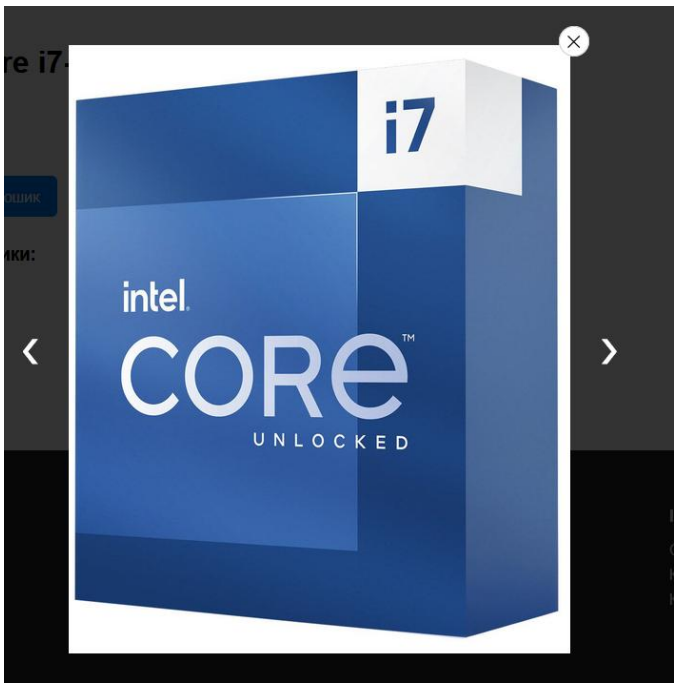

Рисунок 3.10 – Повновіконна галерея

3.6 Реалізація системи авторизації

У вебзастосунку реалізована базова система автентифікації користувачів із підтримкою реєстрації, входу в обліковий запис, перевірки ролі (адміністратор або звичайний користувач) та доступу до обмежених функцій. Під час заповнення форми реєстрації користувач надсилає email та пароль. На бекенді пароль одразу хешується за допомогою бібліотеки bcrypt, яка використовує алгоритм з додаванням «солі» – випадкового значення, що додається до паролю перед хешуванням. Це унеможливлює зворотне розшифрування пароля або підбір за словником.

Лістинг 3.5 Алгоритм регістрації:

```
const bcrypt = require("bcrypt");
const User = require("../models/User");
router.post("/register", async (req, res) => {
  const { email, password } = req.body;
```

```
const candidate = await User.findOne({ email });
if (candidate) {
      return res.status(400).json({ message: "Такий користувач вже
   існує" });
}
const hashedPassword = await bcrypt.hash(password, 10)
const user = new User({
  email,
  password: hashedPassword,
  role: "user"
});
await user.save();
res.status(201).json({ message: "Реєстрація успішна" });
});
```

Також система має перевірку на вже існуючу у системі пошту при реєстрації (рис. 3.11).

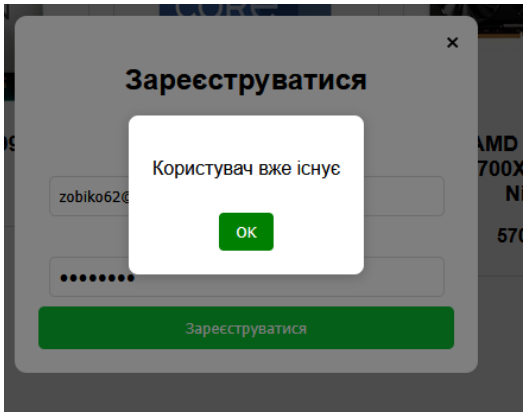

Рисунок 3.11 – Перевірка при реєстрації

При вході користувач надає email і пароль. Пароль порівнюється з хешем у базі за допомогою функції bcrypt.compare. Якщо все вірно, то користувачу видається JWT-токен (JSON Web Token), який зберігається в

браузері. Цей токен містить інформацію про ID та роль користувача. Якщо дані були невірними, то користувач отримає відповідне повідомлення (рис. 3.12).

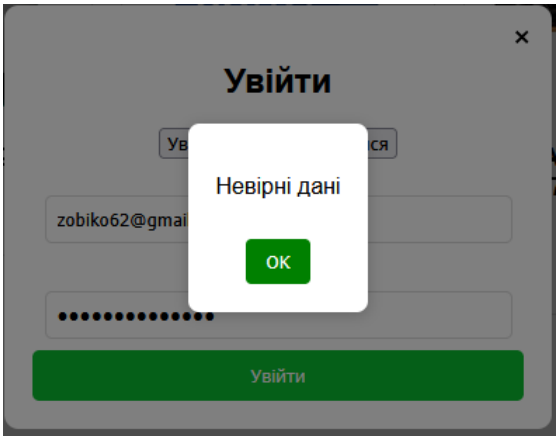

Рисунок 3.12 – Перевірка при спробі авторизації

Лістинг 3.6 Алгоритм логіну

```
const jwt = require("jsonwebtoken");
router.post("/login", async (req, res) => {
  const { email, password } = req.body;
  const user = await User.findOne({ email });
  if (!user) {
    return res.status(400).json({ message: "Невірний email або пароль" });
  }
  const isMatch = await bcrypt.compare(password, user.password);
  if (!isMatch) {
    return res.status(400).json({ message: "Невірний email або пароль" });
  }
  const token = jwt.sign(
    { userId: user._id, role: user.role },
    process.env.JWT_SECRET,
    { expiresIn: "7d" }
```

```
  );
  res.json({ token });
});
```

Якщо ж користувач успішно здійснив логін, то отримає відповідне сповіщення (рис. 3.13).

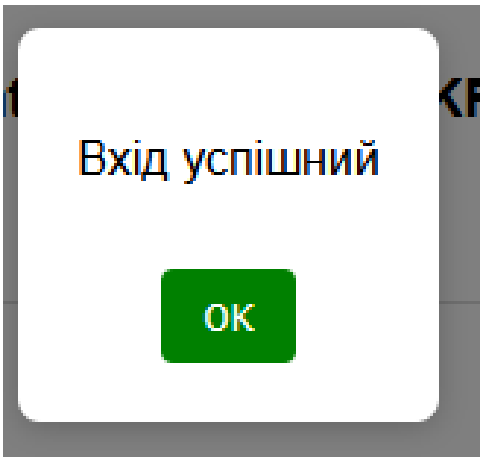

Рисунок 3.13 – Успішний логін

Також після успішного логіну іконка профілю стане зеленою, а при натисненні на неї висвітиться вікно з привітанням, а також можливістю виходу з аккаунту. Також, якщо користувач є адміністратором, то йому також буде надана можливість перейти у панель адміністрації (рис. 3.14).

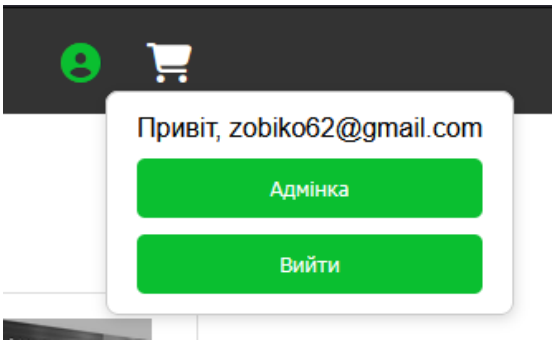

Рисунок 3.14 – Модальне меню адміністратора після логіну

3.7 Реалізація системи оплати

Сторінка оплати дозволяє користувачу переглянути товари в кошику, вказати контактні дані, обрати відділення доставки та спосіб оплати. На початку сторінка завантажує поточний вміст кошика з локального сховища або з сервера, залежно від того, чи авторизований користувач. У момент переходу на сторінку оформлення замовлення, система перевіряє, чи не перевищує замовлена кількість товарів їхню фактичну наявність. Якщо товарів недостатньо – виводиться повідомлення і кількість товарів зменшується до можливого. Дані про товари надсилаються на сервер, де кожен товар тимчасово резервується – тобто його кількість зменшується, і він не доступний для замовлень іншими користувачами [23].

Для доставки реалізовано інтеграцію з API Нової Пошти. Користувач може обрати місто та відділення з форми, що автоматично підтягує і показує дані про можливі точки видачі з вебсервісу Нової Пошти [24]. Окрім Нової Пошти можливо вибрати доставку УкрПоштою, але тоді індекс необхідно ввести власноруч у поле з перевіркою на п’ятизначний індекс. Приклад вікна з оформленням доставки показано на рисунку 3.15.

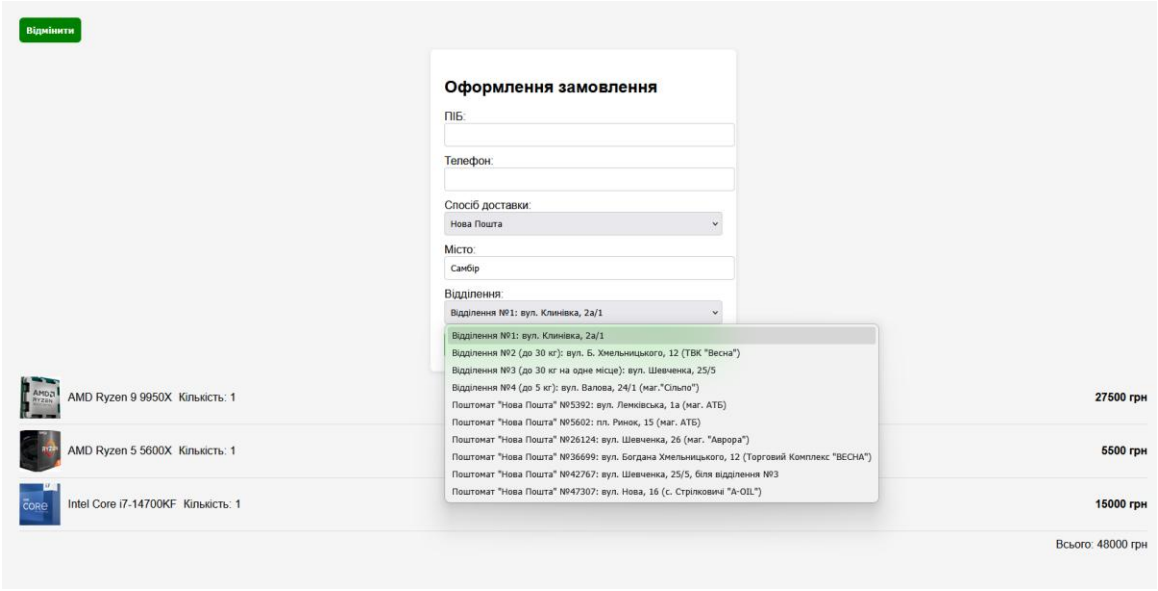

Рисунок 3.15 – Вікно доставки з обраною Новою Поштою

Також є вибір способу оплати: оплата карткою або післяплата. Якщо обрано оплату карткою, замовлення вважається оплаченим. Приклад вікна вибору оплати показано на рисунку 3.16.

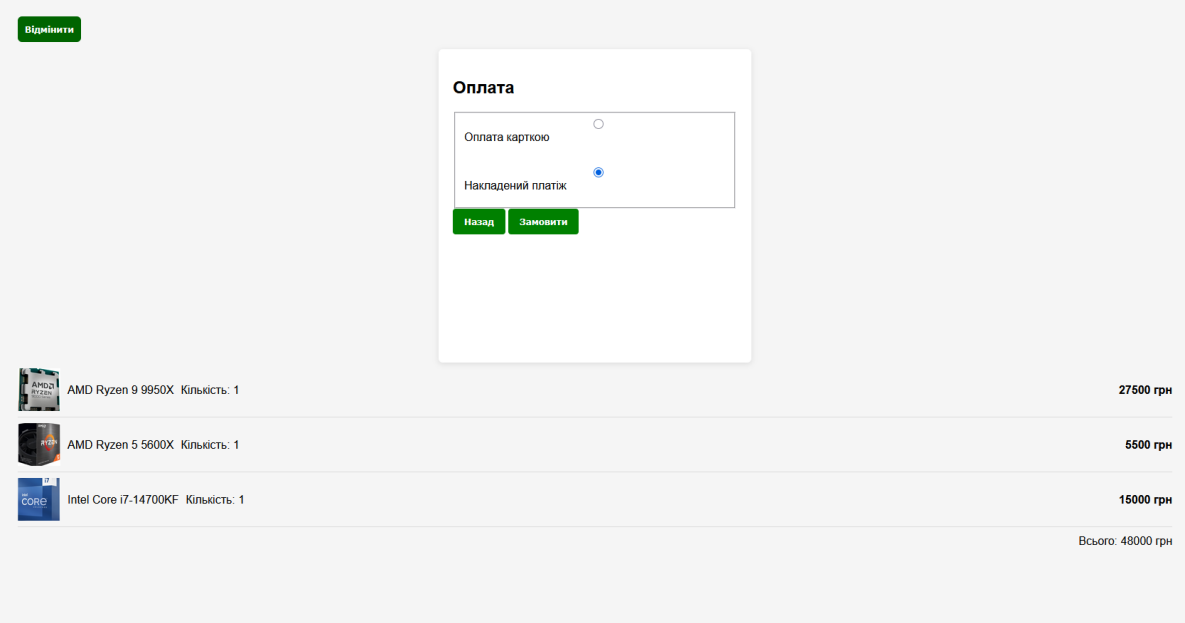

Рисунок 3.16 – Вікно з вибором оплати

Якщо було обрано післяплату – замовлення створюється одразу, але статус оплати буде «не оплачено» і зміниться лише у випадку якщо користувач «забере» товар. Якщо користувач здійснив замовлення, то воно буде сформовано і записане у базі даних, якщо ж користувач натиснув кнопку «Відмінити», то резервація з товарів знімається і відповідна кількість повертається у загальний доступ для придбання.

3.8 Реалізація меню адміністративної панелі

Адміністративна панель вебзастосунку реалізована у вигляді окремої сторінки admin.html, доступ до якої мають лише авторизовані користувачі з роллю адміністратора. Панель надає інтерфейс для керування користувачами, категоріями, товарами, пресетами та замовленнями. Основний принцип її

роботи – динамічне перемикання між вкладками, які відображаються у центральній частині сторінки після натискання на відповідний пункт у меню зліва (рис. 3.17).

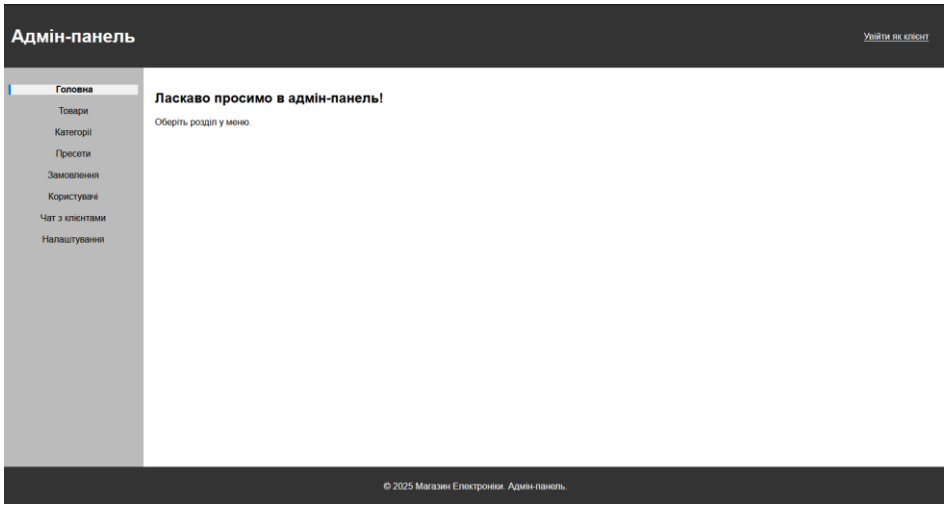

Рисунок 3.17 – Головна сторінка адмін-панелі

3.9 Реалізація створення категорій

Цей розділ надає можливість адміністраторам керувати структурою каталогу магазину, надаючи можливість створення, редагування та видалення категорій товарів з системою батьківських та дочірніх категорій (рис. 3.18-3.19).

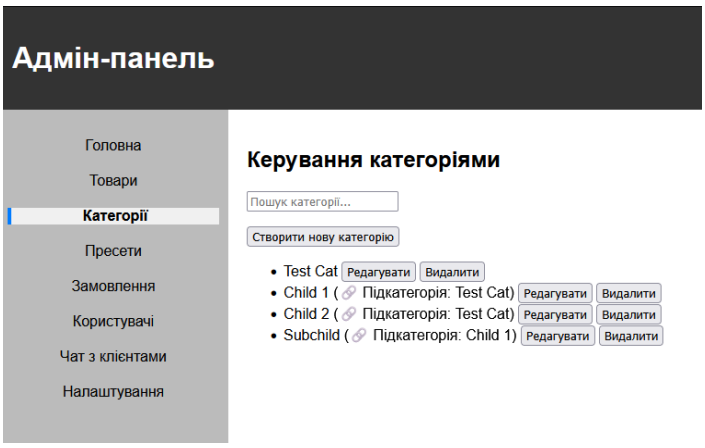

Рисунок 3.18 – Підрозділ меню з категоріями

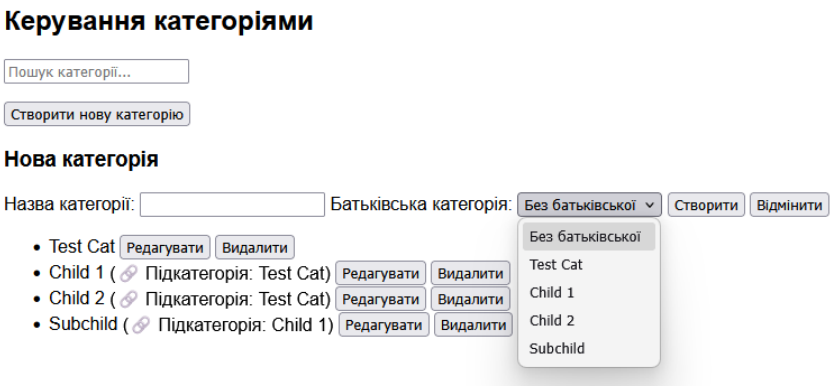

Рисунок 3.19 – Меню для створення категорій

3.10 Реалізація створення пресетів

Цей розділ надає можливість адміністраторам створювати пресети для товарів. Пресети надають змогу створювати готові шаблони, що значно полегшують створення однакових товарів зі схожими полями. Можна заготовлювати готові поля та навіть вже готові значення для цих полів, а також зазначати пресет [25]. При створенні товару, обираючи пресет, створені поля та обрана категорія перенесуться у товар. Приклад вікна для створення нового пресету показано на рисунку 3.20.

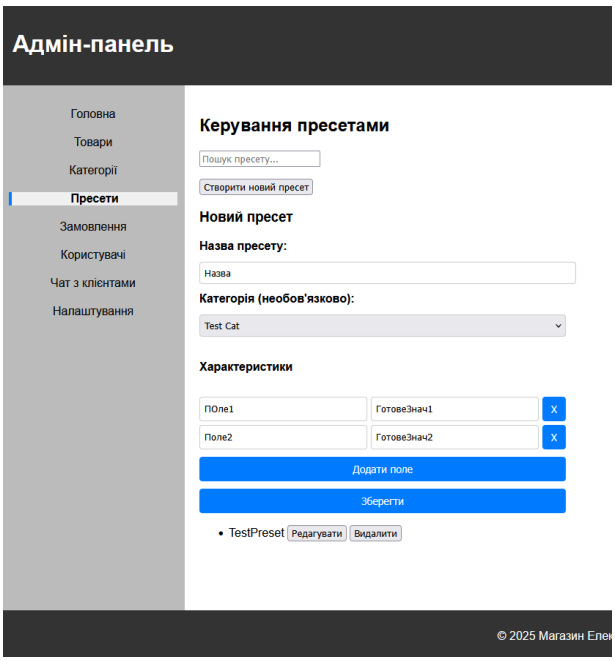

Рисунок 3.20 – Меню для створення категорій

3.11 Реалізація створення товарів

Інтерфейс створення товарів дозволяє адміністраторам додавати нові товари, шукати, редагувати та видаляти існуючі.

При створенні можна обрати категорію, завантажити одне чи декілька зображень, додати назву, опис, ціну, кількість, позначити товар як прихований, додати власні поля зі значеннями, а також обрати один із пресетів. Товар зберігається у базі даних та стає доступним на основній сторінці. На рисунку 3.21 показано редагування існуючого товару.

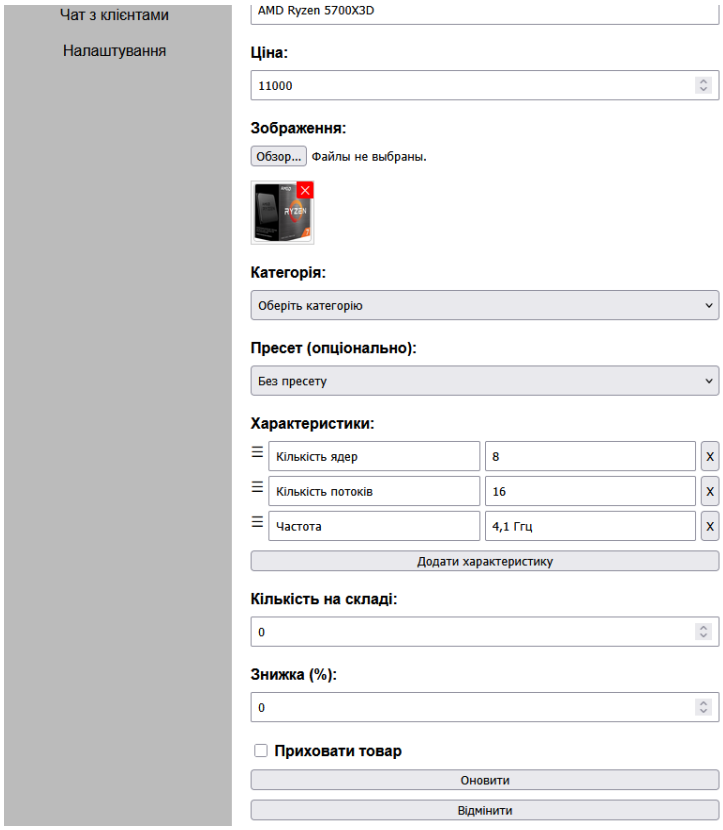

Рисунок 3.21 – Меню редагування товару

Якщо було обрано пресет, його поля автоматично додаються до форми. Фотографії, а також створені власноруч поля можна перетягувати, щоб змінювати їх положення. При видаленні товару або конкретного зображення з бази даних відповідні зображення видаляються з локального диску. На рисунку 3.22 показано меню товарів з пошуком товарів.

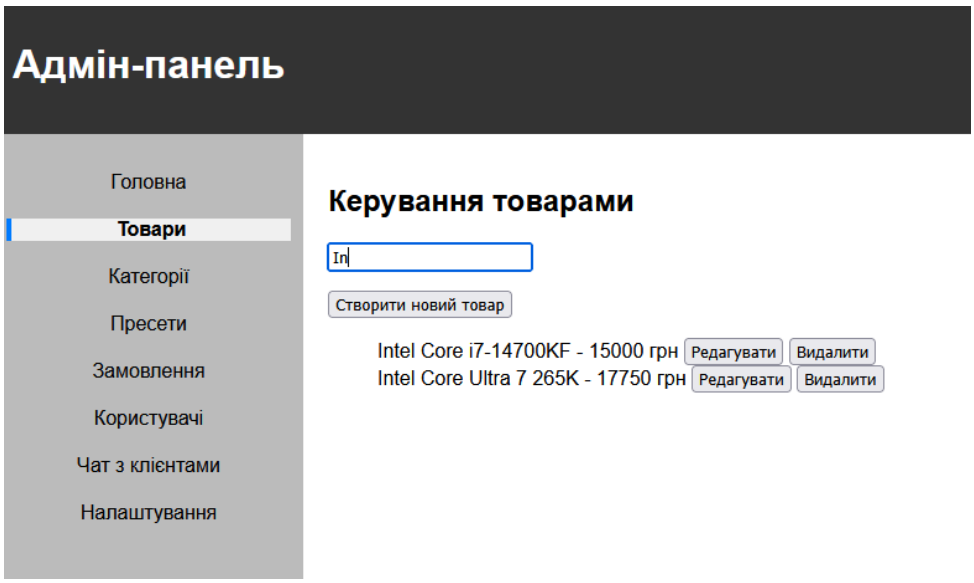

Рисунок 3.22 – Пошук у розділі товарів

3.12 Реалізація таблиці керування користувачами

Розділ користувачів дозволяє адміністраторам переглядати список усіх зареєстрованих користувачів, фільтрувати їх за поштою, змінювати ролі або видаляти з системи. Усі користувачі мають одну з двох ролей: звичайний користувач або адміністратор. Також є пошук конкретних користувачів.

Вигляд таблиці користувачів показано на рисунку 3.23.

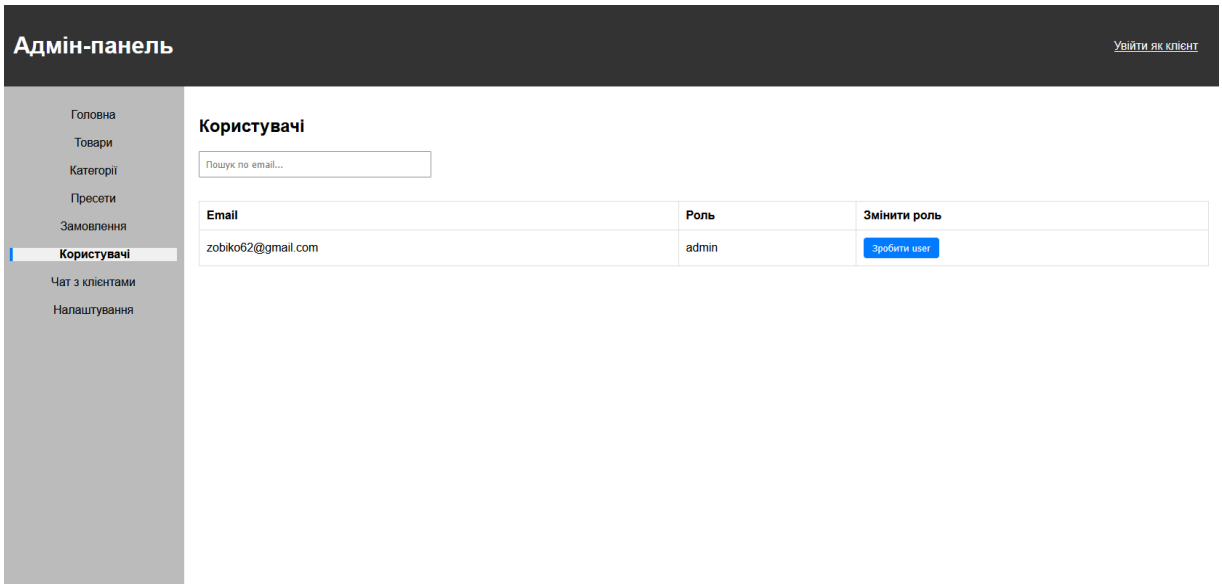

Рисунок 3.23 – Таблиця користувачів

3.13 Реалізація керування замовленнями

Розділ замовлень дозволяє переглядати усі оформлені користувачами замовлення. У таблиці представлено основну інформацію: контактні дані користувача, вибраний спосіб доставки, статус оплати, а також перелік замовлених товарів з їх кількістю.

Для кожного замовлення можна змінити статус вручну, наприклад, позначити як «оплачено» або «відправлено». Також адміністраторам доступна можливість скасування замовлення, після чого усі зарезервовані товари повертаються у доступну кількість на складі. Є можливість пошуку замовлень по даті, імені або телефону. На рисунку 3.24 показано приклад списку замовлень у панелі керування.

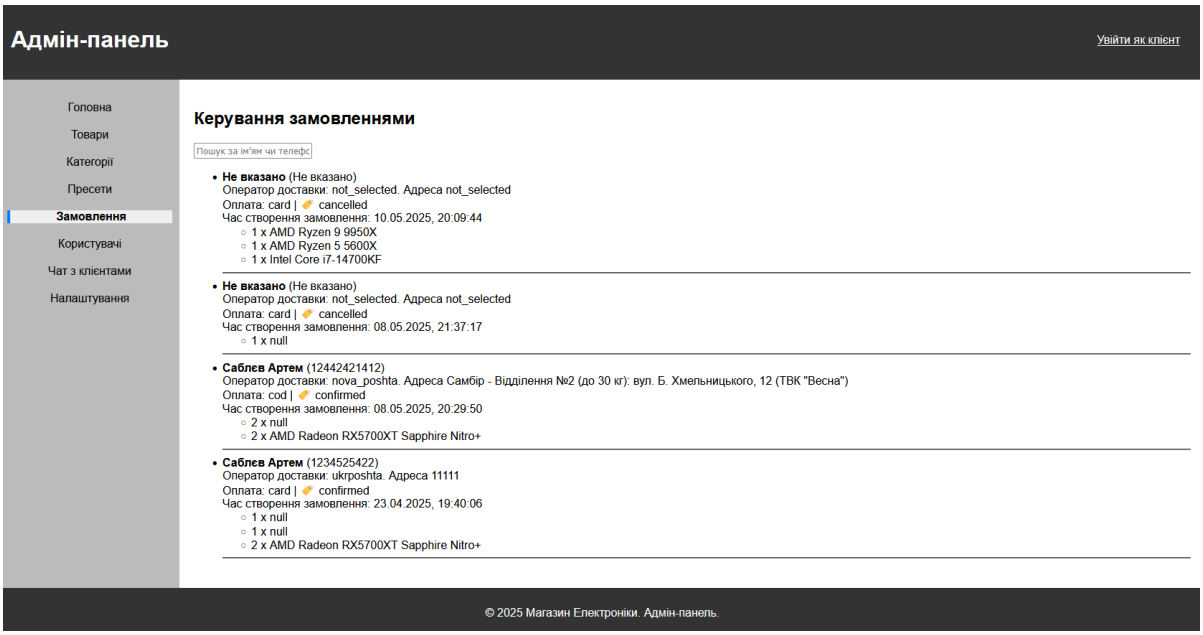

Рисунок 3.24 – Таблиця з замовленнями

3.14 Реалізація чат-боту

Для покращення зручності користувачів у вебзастосунок інтегровано інтелектуального чат-бота, який базується на моделі GPT від OpenAI [26-27].

Його основна функція – надавати консультацію на запитання клієнтів щодо товарів. Бот не просто відповідає на шаблонні запити, а генерує змістовні відповіді залежно від запиту користувача.

У нижньому лівому куті сторінки розміщено інтерактивну іконку чату у вигляді кнопки з іконкою повідомлень. При натисканні відкривається вікно діалогу з історією повідомлень, полем для введення тексту та кнопкою надсилання. На рисунку 3.25 показано приклад спілкування з ботом.

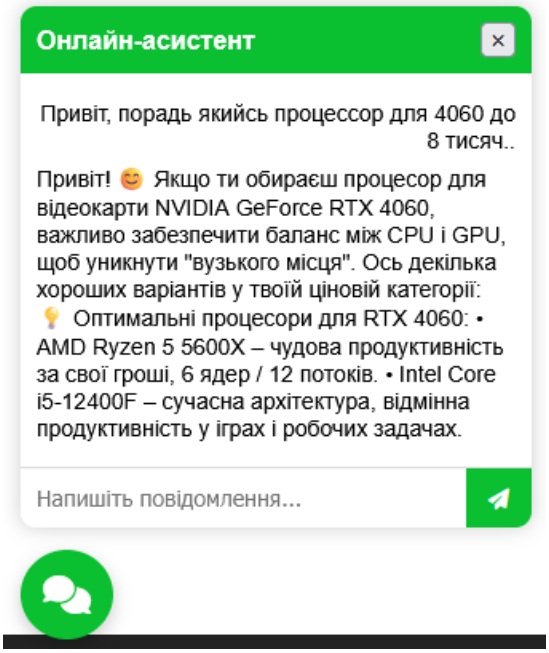

Рисунок 3.25 – Спілкування з чат-ботом

Вся взаємодія з ботом реалізується через POST-запити на серверний маршрут /api/chat. Коли користувач вводить повідомлення, фронтенд надсилає запит на сервер, де виконується звернення до API OpenAI [28]. На бекенді також задано системне повідомлення, яке формує стартову установку бота у якості помічника магазину електроніки. Також бот отримує список поточних товарів сайту, які є у наявності, а також може обирати товари з цього списку базуючись на цінах на сайті [29].

Лістинг 3.7 Серверна обробка повідомлення бота:

```
router.post("/", async (req, res) => {
```

```
 const { message } = req.body;
   if (!message) return res.status(400).json({ message: "Порожнє повідомлення" });
 try {
  const products = await Product.find({ stock: { $gt: 0 } })
   .select("name price description")
   .limit(20)
   .lean();
  const productSummary = products.map(p =>
   `• ${p.name} - ${p.price} грн. ${p.description || ""}`
  ).join("\n");
  const systemPrompt = `
        Ти асистент магазину електроніки. Ось список товарів, які зараз у наявності:
   ${productSummary}
       Відповідай коротко, давай поради і по можливості пропонуй товари з цього списку, якщо це доречно.
  `
  const response = await openai.chat.completions.create({
   model: "gpt-4.0 ",
   messages: [
    { role: "system", content: systemPrompt },
    { role: "user", content: message }
   ],
  });
  const reply = response.choices?.[0]?.message?.content;
  res.json({ reply });
 } catch (err) {
  console.error("Chat error:", err);
```

```
    res.status(500).json({ message: "Помилка OpenAI", error: err.message
});
  }
});
```

ВИСНОВКИ

У рамках кваліфікаційної роботи створено повноцінний вебзастосунок інтернет-магазину. Результатом роботи став повнофункціональний інтернет-магазин, у якому реалізовано основні елементи – каталог, кошик, оформлення замовлення, резервування залишків, авторизація та адміністрування. Побудовано сервер на Node.js з винесенням шляху для кожного типу об’єкту винесено у власний модуль. Реалізовано систему запису до БД, що містить перевірки типів, обмеження діапазонів і каскадні Таблиця дії для пов’язаних файлів, що гарантує надійність і цілісність бази. Процес оформлення замовлення супроводжується перевірками для усунення колізій при покупці товарів.

На клієнтському боці також застосовано компонентну модель коду: головна сторінка, кошик і сторінка товару розміщені у відокремлених файлах. Було реалізовано кошик з гібридною системою зберігання.

Головним впровадженням стала інтеграція чат-боту на основі моделі GPT від OpenAI. Чат-бот може надавати розгорнуті рекомендації технічного характеру користувачам без досвіду з комп’ютерами, а також має доступ до асортименту і цін на вебсайті.

Результати роботи апробовано у вигляді тез доповідей під час Міжнародного молодіжного форуму «Радіоелектроніка і молодь у XXI столітті» [30].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Al-Amin, M., Ali, M. S., Salam, A., Khan, A., Ali, A., Ullah, A., ... & Chowdhury, S. K. (2024). History of generative Artificial Intelligence (AI) chatbots: past, present, and future development. arXiv preprint arXiv:2402.05122.

2. Eisenstein, J. (2019). Introduction to natural language processing. MIT press.

3. Rule-based Chatbot vs AI Chatbot: Which to Choose for Your Business. URL: https://www.apriorit.com/dev-blog/web-ai-vs-rule-based-chatbots (дата звернення 15.05.2025).

4. Alsharhan, A., Al-Emran, M., & Shaalan, K. (2023). Chatbot adoption: A multiperspective systematic review and future research agenda. IEEE Transactions on Engineering Management, 71, 10232-10244.

5. Olena, Y., Ľubomír, N., & Kirichenko, A. (2023). Using the GPT models for responses based on custom content to develop neural consultant for university applicants. In The V International Scientific and Practical Conference" Trends in science regarding the creation of new teaching methods", October 16-18, 2023, Madrid, Spain. 199 p. Text Copyright© 2023 by the European Conference (https://eu-conf. com/). Illustrations© 2023 by the European Conference. Cover design: European Conference (https://eu-conf. com/). (p. 172).

6. Kuzomin, O., Tolmachova, T., & Astappiev, O. (2017). Analysis of Web user activity data. International Journal of Information Models and Analyses, 6(2), 108-118.

7. Daroch, B., Nagrath, G., & Gupta, A. (2021). A study on factors limiting online shopping behaviour of consumers. Rajagiri Management Journal, 15(1), 39-52.

8. Тітов, С. В., & Тітова, О. В. (2015). Оцінка юзабіліті освітніх сайтів: методи і технології.

9. Hsu, C. L., & Lin, J. C. C. (2023). Understanding the user satisfaction and loyalty of customer service chatbots. Journal of Retailing and Consumer

Services, 71, 103211. Hsu, C. L., & Lin, J. C. C. (2023). Understanding the user satisfaction and loyalty of customer service chatbots. Journal of Retailing and Consumer Services, 71, 103211.

10. Євтушенко, Д., & Творошенко, І. С. (2024). Особливості застосування методів оптимізації бізнес-процесів, реалізованих засобами машинного навчання та нейронних мереж.

11. The Rise of the Serverless Monoliths. URL: https://medium.com/@dbottiau/the-rise-of-the-serverless-monoliths-63d3d2d98164 (дата звернення 15.05.2025).

12. Golovianko, M., Gryshko, S., Titova, L., & Filatov, V. (2022). Good practices of Industry 4.0 in Ukraine.

13. Rappl, F. (2021). The Art of Micro Frontends: Build websites using compositional UIs that grow naturally as your application scales. Packt Publishing Ltd.

14. Jain, V., Malviya, B. I. N. D. O. O., & Arya, S. A. T. Y. E. N. D. R. A. (2021). An overview of electronic commerce (e-Commerce). Journal of Contemporary Issues in Business and Government, 27(3), 666.

15. Творошенко, І. С. (2021). Технології прийняття рішень в інформаційних системах.

16. Гороховатський, В.О., & Творошенко, І.С. (2021). Методи інтелектуального аналізу та оброблення даних: навч. посібник.

17. Vechirska, A., Shyrokorad, K., & Vechirska, I. (2022). VISUAL MODELING OF PURCHASE PROCESS MANAGEMENT IN THE ELECTRONICS ONLINE STORE. Матеріали конференцій МНЛ, (3 червня 2022 р., м. Львів), 189-192.

18. Iryna, T., & Anastasiia, A. (2021). DEVELOPMENT OF WEB APPLICATIONS FOR REMOTE LEARNING OF ENGLISH. *EDITORIAL BOARD*, 645.

19. Tvoroshenko, I. (2019). Development of models of spatial analysis of status of interactive processes of complex systems.

20. Gupta, R. D. (2023). Software Development With UML Modelling and Software Testing Techniques. In The Software Principles of Design for Data Modeling (pp. 155-167). IGI Global.

21. Tvoroshenko, I. S., & Maksimenko, H. (2021). To the question of analysis of existing mechanisms of web application testing.

22. Кобилін, О.А., & Творошенко, І.С. (2021). Методи цифрової обробки зображень.

23. Sajeesh, S., Singh, A., & Bhardwaj, P. (2022). Optimal checkout strategies for online retailers. Journal of Retailing, 98(3), 378-394.

24. Руденко, Д. О., & Маренич, В. В. (2024, September). ДОСЛІДЖЕННЯ МЕТОДІВ РОЗРОБКИ ПРОГРАМНИХ ДОДАТКІВ З ІНТЕГРОВАНИМИ API. In *The 2 nd International scientific and practical conference "Scientific research: modern challenges and future prospects"(September 23-25, 2024) MDPC Publishing, Munich, Germany. 2024. 409 p.* (p. 144).

25. Maryniuk, A., Khamula, O., & Sosnovska, O. (2024). The Impact of Key Aspects on Admin Panel Design for Online Media Site.

26. Tvoroshenko, I., & Temchur, K. (2021). Features of software application development for food recognition using deep machine learning methods.

27. Tvoroshenko, I., & Kharchenko, A. (2021). Some aspects of modern development for sign language recognition systems.

28. OpenAI API. URL: https://openai.com/index/openai-api (дата звернення 08.05.2025).

29. Tvoroshenko I., and Gorokhovatskyi V. (2022) The Application of Hybrid Intelligence Systems for Dynamic Data Analysis, *International Journal of Engineering and Information Systems*, 6(2), pp. 40-48.

30. Саблєв А. І. (2025) Розробка вебплатформи для електронної комерції з інтегрованим інтелектуальним чат-ботом. *Радіоелектроніка і молодь у XXI столітті: тези доповідей 29-го Міжнародного молодіжного форуму (Харків, 16–19 квітня 2025 р.)*. Харків: ХНУРЕ, 2025. Т.7. С. 126-128.