МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ**

ННІ/факультет Навчально-науковий інститут економіки та бізнес-освіти

Кафедра Економіки та цифрового бізнесу

Спеціальність 122 «Комп'ютерні науки»

Форма навчання Денна

# КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

**Акбар Фаяд Фарідовича**

*(прізвище, ім'я, по батькові здобувача)*

на тему «Розробка та вдосконалення веб-додатка інтернет-магазину з адміністративною панеллю та розширеним функціоналом взаємодії з користувачем»

*(повна назва теми)*

за матеріалами

*(повна назва бази дослідження)*

науковий керівник к.п.н., доцент _____ Павлиш Т.Г.
*(наук. ступінь, вчене звання)* *(підпис)* *(прізвище, ініціали)*

**Робота допущена до захисту в ЕК**
Протокол засідання кафедри
Від 12 червня 2026 р. № 15
Завідувач кафедри _____
(підпис)
к.е.н., доцент Радько В.М.
*наук. ступень, вчене звання* прізвище, ініціали

Кривий Ріг – 2026

ЗАТВЕРДЖЕНО
Наказ Міністерства освіти і науки, молоді та спорту України
29 березня 2012 року № 384

**Форма № Н-9.01**

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ**
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ ТА БІЗНЕС-ОСВІТИ
( повне найменування вищого навчального закладу )

Кафедра економіки та цифрового бізнесу
Освітній ступінь бакалавр
Спеціальність 122 Комп'ютерні науки

**ЗАТВЕРДЖУЮ**

**Завідувач кафедри _____В.М. Радько**

"30" березня 2026 року

**З А В Д А Н Н Я**
**НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ ЗДОБУВАЧУ**

Акбару Фаяду Фарідовичу

1. Тема роботи «Розробка та вдосконалення веб-додатка інтернет-магазину з адміністративною панеллю та розширеним функціоналом взаємодії з користувачем»

науковий керівник роботи к.п.н., доцент Павлиш Т.Г. ,
затверджені наказом вищого навчального закладу від «23» березня 2026 р. № 193-ст (д/ф)

2. Строк подання здобувачем роботи 29.05.2026р.

3. Зміст кваліфікаційної бакалаврської роботи, об'єкт, предмет та мета дослідження:

Розділ 1 Теоретичні аспекти та аналіз предметної області розробки веб-додатків для електронної комерції
Розділ 2 Проєктування архітектури та моделі веб-додатка онлайн-магазину
Розділ 3 Програмна реалізація веб-додатка та адміністративної панелі на основі MERN-стеку
Об'єкт дослідження – процес функціонування веб-додатків електронної комерції з клієнт-серверною архітектурою
Предмет дослідження – методи та засоби проєктування і програмної реалізації SPA-веб-додатка інтернет-магазину на основі стеку MERN із використанням REST API, JWT-автентифікації та документо-орієнтованої бази даних MongoDB
Мета кваліфікаційної бакалаврської роботи – проєктування та програмна реалізація веб-додатка інтернет-магазину Nexora із адміністративною панеллю та розширеним функціоналом взаємодії з користувачем на основі стеку MERN

4. Дата видачі завдання 03.04.2026р.

# КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної бакалаврської роботи | Строк виконання етапів роботи | Відмітка керівника про виконання етапів (дата, підпис) |
|---|---|---|---|
| 1 | Підготовка розділу 1 | до 17.04.2026р. | |
| 2 | Підготовка розділу 2 | до 08.05.2026р. | |
| 3 | Підготовка розділу 3 | до 25.05.2026р. | |
| 4 | Реєстрація завершеної кваліфікаційної роботи | до 29.05.2026р. | |
| 5 | Отримання відгуку від наукового керівника | 04.06.2026р. | |
| 6 | Отримання зовнішньої рецензії | 05.06.2026р. | |
| 7 | Попередній захист кваліфікаційної роботи на кафедрі | 08.06.2026р. | |
| 8 | Перевірка кваліфікаційної роботи на плагіат | 12.06.2026р. д/ф<br>18.06.2026р. з/ф | |
| 9 | Допуск кафедрою кваліфікаційної роботи до захисту | 12.06.2026р. д/ф<br>18.06.2026р. з/ф | |
| 10 | Підготовка студента до захисту в ЕК | до 19.06.2026р. д/ф<br>до 23.06.2026р з/ф | |

**Завдання підготував науковий керівник** _____ Павлиш Т.Г.
(підпис) (прізвище та ініціали)

**Завдання одержав здобувач** _____ Акбар Ф.Ф.
(підпис) (прізвище та ініціали)

# РЕФЕРАТ

Робота містить 84 сторінки, 19 рисунків, 7 таблиць, 3 додатка, 30 джерел.

Об’єкт дослідження: процес функціонування веб-додатків електронної комерції з клієнт-серверною архітектурою.

Предмет дослідження: методи та засоби проєктування і програмної реалізації SPA-веб-додатка інтернет-магазину на основі стеку MERN із використанням REST API, JWT-автентифікації та документо-орієнтованої бази даних MongoDB.

Мета роботи: проєктування та програмна реалізація веб-додатка інтернет-магазину Nexora із адміністративною панеллю та розширеним функціоналом взаємодії з користувачем на основі стеку MERN.

Методи дослідження: методи системного аналізу, проєктування інформаційних систем, принципи клієнт-серверної взаємодії, моделювання структури даних (UML, BPMN).

В результаті дослідження спроєктовано та розроблено SPA-веб-додаток на основі стеку MERN (React, Node.js, Express.js, MongoDB). Реалізовано механізми JWT-автентифікації та рольову модель доступу (user, moderator, admin). Веб-додаток підтримує каталог товарів, багатокрокове оформлення замовлень з інтеграцією Нової Пошти, систему відгуків із модерацією, підтримку кількох списків бажань та порівняння товарів. Створено захищену адміністративну панель для керування контентом.

Область застосування: підприємства малого та середнього бізнесу у сфері електронної комерції для оптимізації онлайн-торгівлі.

ВЕБ-ДОДАТОК, ЕЛЕКТРОННА КОМЕРЦІЯ, ІНТЕРНЕТ-МАГАЗИН, МОДЕРАЦІЯ, JWT-АВТЕНТИФІКАЦІЯ, MERN, MONGODB, NODE.JS, REST API, SPA-АРХІТЕКТУРА, UML-МОДЕЛЮВАННЯ

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – база даних

ПЗ – програмне забезпечення

API – Application Programming Interface (інтерфейс прикладного програмування)

BPMN – Business Process Model and Notation (модель і нотація бізнес-процесів)

BSON – Binary JSON (бінарний формат подання JSON-документів)

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення)

JSON – JavaScript Object Notation (текстовий формат обміну даними)

JWT – JSON Web Token (веб-токен формату JSON)

MERN – MongoDB, Express.js, React, Node.js (стек веб-технологій)

NoSQL – Not Only SQL (нереляційний підхід до організації баз даних)

REST – Representational State Transfer (архітектурний стиль взаємодії веб-сервісів)

SPA – Single Page Application (односторінковий веб-додаток)

UI – User Interface (користувацький інтерфейс)

UML – Unified Modeling Language (уніфікована мова моделювання)

# ВСТУП

Сфера електронної комерції продовжує активно розвиватися під впливом цифровізації бізнес-процесів, поширення онлайн-послуг та зростання кількості користувачів мережі Інтернет. Інтернет-магазини стали важливою складовою сучасного ринку товарів і послуг, оскільки забезпечують можливість дистанційного придбання продукції, швидкий доступ до інформації про товари, автоматизацію процесів оформлення замовлень та взаємодію між продавцем і покупцем у режимі реального часу. Особливо актуальним є створення веб-додатків електронної комерції, які поєднують зручний користувацький інтерфейс, гнучку систему адміністрування та підтримку динамічної взаємодії з користувачами.

Актуальність теми кваліфікаційної роботи обумовлена необхідністю створення веб-додатків інтернет-магазинів, здатних підтримувати роботу великої кількості функціональних модулів у межах єдиної клієнт-серверної системи. В умовах постійного зростання обсягів онлайн-торгівлі особливого значення набувають питання організації зручного каталогу продукції, управління замовленнями, персоналізації взаємодії з користувачем, модерації контенту, а також реалізації адміністративних інструментів для підтримки актуального стану системи. Окремого значення набуває використання SPA-архітектури та сучасних JavaScript-технологій, які забезпечують динамічне оновлення інтерфейсу без повного перезавантаження сторінок і покращують зручність роботи користувача із веб-додатком.

Використання SPA-підходу у сфері електронної комерції пов’язане з необхідністю швидкого оновлення інтерфейсу, зменшення кількості повторних завантажень сторінок та підвищення зручності взаємодії користувача із системою. Для інтернет-магазинів важливим є забезпечення безперервної роботи каталогу товарів, кошика, системи оформлення замовлень та особистого кабінету користувача без втрати поточного стану інтерфейсу. Подібний підхід позитивно впливає на швидкість навігації між сторінками та спрощує виконання

основних користувацьких сценаріїв.

Окрему роль у сучасних веб-додатках електронної комерції відіграють адміністративні модулі та засоби керування контентом. У подібних системах важливим є не лише відображення інформації про товари, а й підтримка механізмів керування категоріями, замовленнями, клієнтами, відгуками та питаннями користувачів. Крім цього, актуальними залишаються засоби персоналізації взаємодії, зокрема списки бажань, історія замовлень, модулі порівняння товарів та інтеграція служб доставки. Це формує потребу у створенні комплексних веб-застосунків, які поєднують клієнтську частину, серверну логіку та систему адміністрування у межах єдиної архітектури.

Під час розробки систем електронної комерції важливим є не лише створення каталогу товарів, а й підтримка повноцінної взаємодії між клієнтською та серверною частинами застосунку. У подібних системах необхідно враховувати питання авторизації користувачів, захисту доступу до адміністративних модулів, організації REST API, роботи з базами даних, реалізації механізмів оформлення замовлень, а також інтеграції зовнішніх сервісів доставки. У таких веб-застосунках активно використовується стек MERN, який поєднує MongoDB, Express.js, React та Node.js і дозволяє створювати SPA-застосунки з використанням JavaScript як на клієнтській, так і на серверній стороні.

Проблеми проєктування та реалізації веб-додатків електронної комерції досліджувалися у працях як зарубіжних, так і вітчизняних науковців. У дослідженнях розглядалися питання побудови клієнт-серверної архітектури, використання REST API, організації взаємодії користувача із системою, проєктування структури баз даних та використання SPA-підходу у веб-розробці. Окремі аспекти створення інтернет-магазинів висвітлювалися у працях, присвячених сучасним JavaScript-фреймворкам, технологіям NoSQL, принципам побудови адаптивного інтерфейсу та системам електронної комерції. Водночас значна частина існуючих досліджень орієнтована переважно на теоретичний опис технологій або окремих компонентів веб-систем, тоді як питання реалізації

повноцінного веб-додатка інтернет-магазину із адміністративною панеллю, модерацією контенту, системою замовлень та інтеграцією служб доставки потребують подальшого практичного опрацювання.

Джерельну базу роботи становлять наукові публікації, навчальні посібники, технічна документація та матеріали, присвячені розробці SPA-застосунків, використанню MERN-стеку, організації REST API, роботі з MongoDB, React, Node.js та Express.js. Окрім цього, під час виконання роботи використовувалися офіційні документації бібліотек і фреймворків, а також практичні матеріали, присвячені побудові систем електронної комерції та організації клієнт-серверної взаємодії.

Об’єктом дослідження є процес функціонування веб-додатків електронної комерції з клієнт-серверною архітектурою.

Предметом дослідження є методи та засоби проєктування і програмної реалізації SPA-веб-додатка інтернет-магазину на основі стеку MERN із використанням REST API, JWT-автентифікації та документо-орієнтованої бази даних MongoDB.

Метою кваліфікаційної роботи є проєктування та програмна реалізація веб-додатка інтернет-магазину Nexora із адміністративною панеллю та розширеним функціоналом взаємодії з користувачем на основі стеку MERN.

Для досягнення поставленої мети необхідно виконати такі завдання:

– проаналізувати предметну область електронної комерції, сучасні підходи до розробки веб-додатків інтернет-магазинів та існуючі програмні рішення;

– дослідити особливості використання SPA-архітектури, клієнт-серверної моделі взаємодії та технологій MERN-стеку у веб-застосунках електронної комерції;

– спроєктувати архітектуру веб-додатка, структуру бази даних та логіку взаємодії між клієнтською і серверною частинами системи;

– реалізувати клієнтську та серверну частини веб-додатка з підтримкою каталогу товарів, кошика, оформлення замовлень, списків бажань, порівняння товарів та інтеграції служби доставки;

– розробити адміністративну панель із рольовою моделлю доступу, модулями керування магазином та системою модерації користувацького контенту;

– провести тестування та верифікацію роботи основних функціональних модулів веб-додатка;

Під час виконання роботи використовувалися методи системного аналізу, методи проєктування інформаційних систем, принципи клієнт-серверної взаємодії, методи моделювання структури даних, а також технології веб-розробки на основі JavaScript. Для реалізації клієнтської частини застосовувалися React, Context API, React Router DOM та SCSS. Серверна логіка побудована із використанням Node.js та Express.js, а для організації роботи з базою даних застосовано MongoDB і Mongoose. Під час проєктування структури системи та взаємодії між її компонентами використовувалися елементи UML та BPMN-моделювання, що дозволило описати основні процеси взаємодії між користувачем, клієнтською частиною та серверною логікою. Для локального розгортання та ізоляції середовища розробки застосовувалися засоби контейнеризації Docker. Тестування REST API та перевірка коректності роботи HTTP-запитів виконувалися за допомогою Postman.

Наукова новизна отриманих результатів полягає у комплексному підході до проєктування та програмної реалізації SPA-веб-додатка інтернет-магазину на основі стеку MERN із поєднанням адміністративної панелі, рольової моделі доступу, системи модерації контенту та інтеграції зовнішніх сервісів доставки у межах єдиної клієнт-серверної архітектури.

Практичне значення роботи полягає у створенні веб-додатка інтернет-магазину Nexora, який підтримує роботу каталогу товарів, оформлення замовлень, адміністративної панелі, системи модерації контенту та механізмів взаємодії користувача із веб-додатком. Результати роботи можуть бути використані як основа для модернізації систем електронної комерції малого та середнього бізнесу, а також для подальшого розширення функціональних можливостей веб-додатка. Запропоновані підходи до організації клієнт-

серверної взаємодії, структури адміністративних модулів та інтеграції сервісів доставки можуть бути адаптовані для інших веб-систем електронної комерції.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі виконано аналіз предметної області електронної комерції, розглянуто сучасні підходи до побудови веб-додатків інтернет-магазинів та проаналізовано існуючі програмні рішення. У другому розділі здійснено проєктування архітектури системи, структури бази даних та алгоритмів взаємодії між компонентами веб-додатка. Третій розділ присвячено програмній реалізації клієнтської та серверної частин системи, адміністративної панелі, механізмів взаємодії користувача із веб-додатком, а також тестуванню та верифікації роботи основних функціональних модулів.

# РОЗДІЛ 1
# ТЕОРЕТИЧНІ АСПЕКТИ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗРОБКИ ВЕБ-ДОДАТКІВ ДЛЯ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

## 1.1 Аналіз сучасного стану ринку електронної комерції та тенденцій розвитку веб-додатків

Глобальна цифровізація економіки та активний розвиток інформаційних технологій спричинили суттєві зміни у сфері роздрібної й гуртової торгівлі. Сьогодні електронна комерція вже не розглядається виключно як додатковий канал збуту, а є важливим елементом функціонування бізнесу [1]. Перехід торговельних процесів у цифровий простір пов’язаний зі зміною споживчої поведінки, поширенням мобільного інтернету та прагненням підприємств оптимізувати витрати й розширити географію реалізації товарів і послуг [2].

Згідно з сучасними статистичними дослідженнями у сфері електронної комерції, онлайн-платформи забезпечують оперативну взаємодію з клієнтами, автоматизацію бізнес-процесів та швидке оновлення інформації про товари й послуги [3].

Питання розвитку електронної комерції активно досліджуються українськими науковцями. Зокрема, І.Є. Максютенко зазначає, що розвиток цифрових технологій суттєво трансформував підходи до організації торгівлі та взаємодії зі споживачами [30]. Науковиця підкреслює, що електронна комерція безпосередньо пов’язана з процесами діджиталізації суспільства, поширенням мобільних сервісів та зміною поведінки користувачів у цифровому середовищі. Це свідчить про те, що онлайн-платформи повинні забезпечувати не лише базову функціональність продажів, а й стабільність роботи, швидкість взаємодії та зручність користування.

Вимоги до систем онлайн-продажів постійно зростають. На сьогодні інтернет-магазини виконують не лише функцію відображення каталогу товарів, а й забезпечує обробку значної кількості даних, підтримку користувацьких сесій,

інтеграцію платіжних сервісів та інструментів адміністрування [4]. Особливого значення набуває автоматизація бізнес-процесів, що охоплює управління складськими залишками, аналітику продажів, персоналізацію контенту та формування звітності. Унаслідок цього підвищуються вимоги до продуктивності, масштабованості та надійності програмних рішень.

Згідно з даними аналітичних платформ, світовий ринок електронної комерції демонструє стабільну тенденцію до зростання (рис. 1.1) [5].

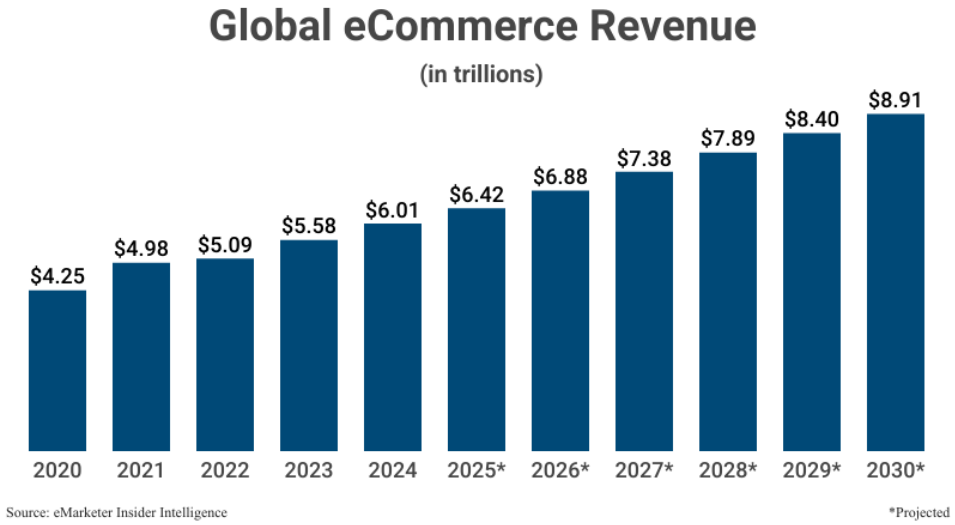


**Рис. 1.1. Динаміка зростання світового обсягу доходів електронної комерції у 2020–2030 рр**

Примітка. Джерело: дані eMarketer Insider Intelligence [5].

Аналіз статистичних даних свідчить про стабільне збільшення обсягів електронної торгівлі у світовому масштабі. Очікуване зростання доходів ринку до понад 8 трильйонів доларів США підтверджує посилення ролі цифрових каналів продажів у діяльності підприємств [3,5]. За таких умов програмні рішення для електронної комерції стають важливим інструментом забезпечення конкурентоспроможності бізнесу. Зростання кількості користувачів та онлайн-транзакцій формує підвищені вимоги до швидкодії систем, стабільності серверної частини та можливості масштабування програмного забезпечення відповідно до навантаження.

Трансформація бізнес-потреб безпосередньо впливає на розвиток

архітектури веб-застосунків. Протягом тривалого часу поширеним підходом залишалися багатосторінкові застосунки (Multi-Page Applications, MPA), у яких кожна взаємодія користувача із системою супроводжується повним перезавантаженням сторінки [6]. Проте такий підхід може спричиняти затримки під час роботи з інтерфейсом і збільшувати навантаження на сервер та мережу. У веб-розробці простежується тенденція до переходу від MPA до односторінкових застосунків (Single Page Applications, SPA) [4,7]. SPA-архітектура передбачає оновлення інтерфейсу без повного перезавантаження сторінки. Це забезпечується передачею даних через API-запити у форматі JSON та їх обробкою за допомогою JavaScript. Такий підхід дозволяє підвищити швидкість взаємодії користувача із системою та забезпечити більш комфортну роботу з інтерфейсом.

Паралельно з архітектурними змінами відбувається розвиток підходів до створення користувацьких інтерфейсів. У сфері електронної комерції інтерфейс є не лише графічною складовою системи, а й одним із факторів, що впливають на зручність користування та рівень взаємодії клієнта з платформою. Актуальні тенденції веб-розробки передбачають використання компонентного підходу, відповідно до якого окремі елементи інтерфейсу створюються як незалежні модулі з можливістю повторного використання [4]. Застосування бібліотеки React [8] у поєднанні з UI-фреймворками, зокрема Material UI [9], дозволяє забезпечити уніфікованість дизайну, скоротити час розробки та спростити підтримку програмного рішення.

Однією з ключових тенденцій веб-розробки є використання підходу Mobile-First, відповідно до якого інтерфейс спочатку проєктується для мобільних пристроїв, а вже після цього адаптується до більших екранів. Актуальність такого підходу пояснюється тим, що значна частина користувачів здійснює пошук товарів і покупки саме за допомогою смартфонів та планшетів [2]. У дослідженні І.Є. Максютенко також наголошується на зростанні ролі мобільної комерції (m-commerce) у структурі електронної торгівлі [30]. У зв’язку з цим адаптивність інтерфейсу, швидкість завантаження сторінок та оптимізація

контенту для мобільних пристроїв стають необхідними умовами ефективного функціонування онлайн-платформи інтернет-магазину.

Зміна підходів до побудови фронтенд-частини веб-застосунків вплинула і на розвиток серверних технологій та систем зберігання даних. Традиційні синхронні серверні моделі не завжди забезпечують ефективну обробку великої кількості одночасних асинхронних запитів, характерних для SPA-застосунків. У зв’язку з цим поширення набули неблокувальні серверні платформи, зокрема Node.js [10]. Використання JavaScript як на клієнтській, так і на серверній стороні у межах MERN-стека дозволяє уніфікувати обмін даними між компонентами системи та спростити процес розробки програмного продукту [7].

Не менш важливим аспектом є розвиток підходів до організації зберігання даних у системах електронної комерції. Специфіка товарних каталогів передбачає роботу з великою кількістю об’єктів, характеристики яких можуть суттєво відрізнятися залежно від категорії товару [6]. За таких умов використання реляційних баз даних може ускладнювати масштабування системи через необхідність підтримки складних зв’язків між таблицями та виконання великої кількості JOIN-операцій. Особливо це проявляється при постійному оновленні асортименту або додаванні нових характеристик товарів. Натомість документо-орієнтовані NoSQL бази даних, зокрема MongoDB [11], дозволяють зберігати інформацію у вигляді гнучких BSON-документів без жорстко визначеної структури. Це спрощує модифікацію даних та підвищує ефективність роботи системи при масштабуванні.

Важливим аспектом розробки систем електронної комерції є забезпечення належного рівня безпеки даних. Зростання кількості онлайн-транзакцій та використання API-орієнтованої взаємодії зумовлюють необхідність впровадження надійних механізмів автентифікації та контролю доступу [12]. Крім того, платформи онлайн-торгівлі повинні забезпечувати захист конфіденційної інформації користувачів, запобігання SQL-ін’єкціям, витоку даних та іншим поширеним кіберзагрозам [13]. Тому питання безпеки є важливою складовою під час проєктування веб-застосунків електронної

комерції.

Окрему увагу сьогодні приділяють інтеграції інструментів штучного інтелекту та аналітики даних. У сфері електронної комерції такі технології використовуються для реалізації рекомендаційних систем, персоналізованих пропозицій, аналізу поведінки користувачів та вдосконалення пошукових механізмів [1,2]. Реалізація подібного функціоналу потребує використання масштабованих програмних рішень та ефективної взаємодії між фронтенд- і бекенд-компонентами системи.

Отже, проведений аналіз стану електронної комерції та тенденцій розвитку веб-технологій дозволяє зробити висновок, що ефективний інтернет-магазин повинен поєднувати сучасні архітектурні підходи, зручний інтерфейс, стабільну роботу серверної частини та ефективне зберігання даних. Використання SPA-архітектури, компонентного підходу до побудови інтерфейсу, принципу Mobile-First та NoSQL баз даних сприяє підвищенню продуктивності системи й покращенню взаємодії користувача з платформою. Саме тому для розробки програмного продукту було обрано MERN-стек, який дозволяє поєднати гнучкість розробки та ефективну взаємодію між клієнтською і серверною частинами системи.

## 1.2 Огляд існуючих аналогів інтернет-магазинів

Проєктування інформаційної системи електронної комерції потребує попереднього аналізу вже існуючих програмних рішень та підходів до організації онлайн-торгівлі [6]. Проведення такого аналізу дозволяє визначити сильні та слабкі сторони популярних платформ, оцінити особливості їхньої архітектури та врахувати практичний досвід реалізації функціоналу інтернет-магазинів.

У підрозділі проведено порівняльний аналіз платформ Amazon, Rozetka, COMFY та WooCommerce. Вибір саме цих аналогів зумовлений тим, що вони представляють різні підходи до реалізації електронної комерції: від великих

міжнародних маркетплейсів до національних платформ та готових CMS-рішень для створення інтернет-магазинів. Аналіз існуючих аналогів дозволяє сформувати вимоги до майбутнього програмного продукту, а також визначити технології та архітектурні рішення, які є доцільними для використання під час розробки.

Розглянуто глобальну платформу електронної комерції Amazon [14] як приклад масштабованої системи онлайн-торгівлі. Платформа функціонує на основі розподіленої серверної інфраструктури та забезпечує стабільну роботу навіть за значної кількості одночасних запитів користувачів. Для зберігання та обробки інформації застосовуються різні типи баз даних, що дозволяє оптимізувати роботу окремих компонентів системи. Водночас інтерфейс платформи залишається досить складним через велику кількість функціональних елементів і сервісів. Крім того, впровадження подібної багаторівневої інфраструктури потребує значних фінансових витрат та постійної технічної підтримки, що є недоцільним для малого та середнього бізнесу.

Проаналізовано також архітектурні рішення українського маркетплейсу Rozetka [15]. Платформа залишається однією з найбільш популярних у сфері електронної комерції в Україні [2]. Під час аналізу встановлено, що система активно використовує компонентний підхід до побудови інтерфейсу та поступово переходить до сучасних SPA-рішень [4]. Важливою особливістю платформи є швидка робота системи фільтрації та пошуку товарів, що забезпечується використанням кешування та оптимізації запитів. Водночас через значний обсяг функціоналу та велику кількість інтеграцій окремі сторінки можуть містити значні JavaScript-бандли, що іноді впливає на швидкість взаємодії користувача з інтерфейсом.

Додатково розглянуто український інтернет-магазин COMFY як приклад спеціалізованої платформи з продажу електроніки та побутової техніки. Інтерфейс платформи орієнтований на швидкий доступ до каталогу товарів, системи фільтрації та персоналізованих пропозицій. Значна увага приділяється адаптації сервісу для мобільних пристроїв, що відповідає сучасним тенденціям

розвитку мобільної комерції. Разом із цим велика кількість рекламних блоків, банерів та додаткових елементів інтерфейсу може ускладнювати сприйняття контенту користувачем і впливати на швидкість завантаження окремих сторінок.

Як ще один аналог розглянуто платформу WooCommerce [16] – поширене рішення для створення інтернет-магазинів на базі WordPress. Архітектурно WooCommerce є монолітним веб-застосунком, побудованим із використанням PHP та реляційної бази даних MySQL. Оновлення WooCommerce 10.5 частково покращило механізми кешування та роботу зі статичними ресурсами, проте основні особливості архітектури залишилися незмінними. Під час аналізу встановлено, що при значному розширенні каталогу товарів або використанні великої кількості плагінів можуть виникати проблеми зі швидкодією системи. Це пов’язано зі складною структурою зберігання метаданих та великою кількістю SQL-запитів під час фільтрації товарів [6]. Крім того, адміністративна панель платформи може працювати повільніше при значному навантаженні або використанні сторонніх модулів.

Для більш наочного представлення результатів проведеного аналізу основні характеристики розглянутих платформ було систематизовано у вигляді порівняльної таблиці (табл. 1.1).

Таблиця 1.1

**Порівняльна характеристика існуючих аналогів інтернет-магазинів**

| Критерії порівняння | Amazon | Rozetka | COMFY | WooCommerce (v10.5) | Власна розробка MERN |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| Тип архітектури веб-додатка | Розподілена мікросервісна архітектура (гібридна MPA з інтенсивним SSR) | Гібридна монолітно-компонентна архітектура з елементами SPA | Компонентна веб-платформа з адаптивним інтерфейсом | Монолітна архітектура (класична MPA з покращеним кешуванням) | Повноцінний односторінковий додаток (SPA) з ізольованим клієнтом та REST API сервером |
| Тип бази даних та структура зберігання | Гетерогенна (корпоративні SQL + NoSQL кластери) | Гетерогенна (PostgreSQL + Redis кеш для оптимізації TTFB) | Реляційні БД із кешуванням | Реляційна (MySQL, реалізація моделі EAV) | Гнучка документо-орієнтована NoSQL база (MongoDB) |

Продовження табл. 1.1

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| Швидкість реакції клієнтського інтерфейсу | Висока, супроводжується частковими перезавантаженнями контенту | Висока, оптимізована за допомогою Redis (TTFB 1-2с) | Достатньо швидка при стандартному навантаженні | Низька при складних запитах JOIN, залежна від налаштувань сервера | Миттєва (візуалізація через Virtual DOM у React та локальний менеджмент стану) |
| Рівень складності розгортання та DevOps підтримки | Високий (потребує професійного супроводу інфраструктури) | Високий (розгалужена серверна інфраструктура) | Середній рівень масштабування | Низький, але має високу складність оптимізації при зростанні навантаження | Середній (потребує налаштування середовища Node.js та статичного розгортання) |
| Ергономічність та швидкість роботи адміністративної панелі | Складна корпоративна система рівня ERP | Комплексна кастомна система управління процесами | Інтегрована система управління товарами | Громіздка, потенційно сповільнена через сторонні розширення | Вузькоспеціалізована, мінімалістична, оптимізована (SPA-адмінка на API) |

Примітка. Джерело: систематизовано автором на основі [14,15,16].

Аналіз результатів, наведених у таблиці, свідчить про те, що сучасні платформи електронної комерції активно переходять до використання компонентних підходів, SPA-архітектури та технологій кешування для підвищення швидкодії системи. Водночас складні корпоративні рішення, орієнтовані на великі маркетплейси, потребують значних ресурсів для підтримки та масштабування. Готові CMS-рішення, зокрема WooCommerce, спрощують запуск інтернет-магазину, однак можуть втрачати ефективність при значному збільшенні кількості товарів або навантаження на систему.

Проведений аналіз також показав, що для більшості сучасних інтернет-магазинів важливими є швидкість взаємодії користувача з інтерфейсом, адаптивність для мобільних пристроїв, зручність адміністрування та можливість подальшого масштабування системи. Саме тому під час розробки власного програмного продукту доцільно використовувати SPA-архітектуру, REST API та документо-орієнтовану базу даних MongoDB, яка дозволяє гнучко працювати з характеристиками товарів без складної структури реляційних зв’язків [11].

Отже, проведений огляд існуючих аналогів дозволив визначити основні

тенденції розвитку платформ електронної комерції, а також сформувати вимоги до розробки власного веб-застосунку інтернет-магазину. Використання MERN-стека дає змогу забезпечити достатню швидкодію системи, зручну взаємодію між клієнтською та серверною частинами, а також створити програмне рішення, яке може бути адаптоване до подальшого розширення функціоналу.

### 1.3 Формулювання поставленої задачі, дослідження бізнес-логіки та обґрунтування актуальності розробки

На основі проведеного аналізу існуючих аналогів інтернет-магазинів виявлено низку особливостей, характерних для багатьох сучасних платформ електронної комерції. Використання класичних монолітних CMS-систем, реляційних баз даних та багатосторінкової архітектури (MPA) може ускладнювати масштабування системи та негативно впливати на швидкість роботи при збільшенні кількості товарів і користувачів. Водночас складні мікросервісні рішення, які використовуються великими міжнародними платформами, потребують значних технічних та фінансових ресурсів, що не завжди є доцільним для малого і середнього бізнесу. У зв’язку з цим сформульовано основне завдання дипломного проєкту – розробка односторінкового веб-застосунку (SPA) з окремою клієнтською частиною, REST API сервером та використанням документо-орієнтованої NoSQL бази даних. Такий підхід дозволяє забезпечити достатню швидкодію системи, зручність роботи з інтерфейсом та можливість подальшого розширення функціоналу [4].

Дослідження бізнес-логіки функціонування системи передбачає формалізацію процесів взаємодії між основними користувачами платформи. У структурі веб-застосунку визначено дві основні ролі: користувач (покупець) та адміністратор магазину. Клієнтська частина системи орієнтована на забезпечення зручної роботи з каталогом товарів, пошуком, фільтрацією, кошиком та списком бажаних товарів [6]. Архітектура SPA дозволяє оновлювати окремі елементи інтерфейсу без повного перезавантаження сторінки. У

результаті взаємодія користувача із системою стає швидшою та більш комфортною.

Бізнес-логіка адміністративної частини спрямована на керування контентом та товарними позиціями. Адміністратор отримує доступ до панелі керування через механізм JWT-автентифікації [12], після чого може виконувати операції створення, редагування, перегляду та видалення товарів. Одним із важливих аспектів системи є підтримка товарів із різними характеристиками. Наприклад, електроніка може містити параметри процесора або обсягу пам’яті, тоді як одяг — розмір або тип матеріалу. У зв’язку з цим система повинна динамічно формувати набір полів залежно від категорії товару.

Для кращого розуміння послідовності дій під час роботи з каталогом було формалізовано алгоритм створення товару. Процес охоплює взаємодію клієнтської частини, яка виконує первинну перевірку введених даних та попередній перегляд зображення, із серверною частиною, що відповідає за обробку та збереження інформації у базі даних. Логіку цієї взаємодії представлено у вигляді BPMN-схеми (рис. 1.2) [4].

Аналіз наведеної схеми показує, що поєднання перевірки даних на стороні клієнта та серверної валідації дозволяє зменшити ризик помилок під час створення товарних карток. Автоматизація окремих етапів також спрощує роботу адміністратора та пришвидшує процес наповнення каталогу.

Для реалізації поставлених завдань було проведено аналіз програмного інструментарію, який може бути використаний під час розробки системи. У сегменті фронтенд-розробки розглянуто JavaScript-бібліотеки React [8] та Vue [17]. Для реалізації клієнтської частини обрано React, оскільки бібліотека підтримує компонентний підхід до побудови інтерфейсу та забезпечує ефективне оновлення сторінок за допомогою механізму Virtual DOM [8]. Крім того, використання JSX дозволяє поєднувати логіку та структуру компонентів у межах одного файлу, що спрощує підтримку коду при розширенні функціоналу системи.

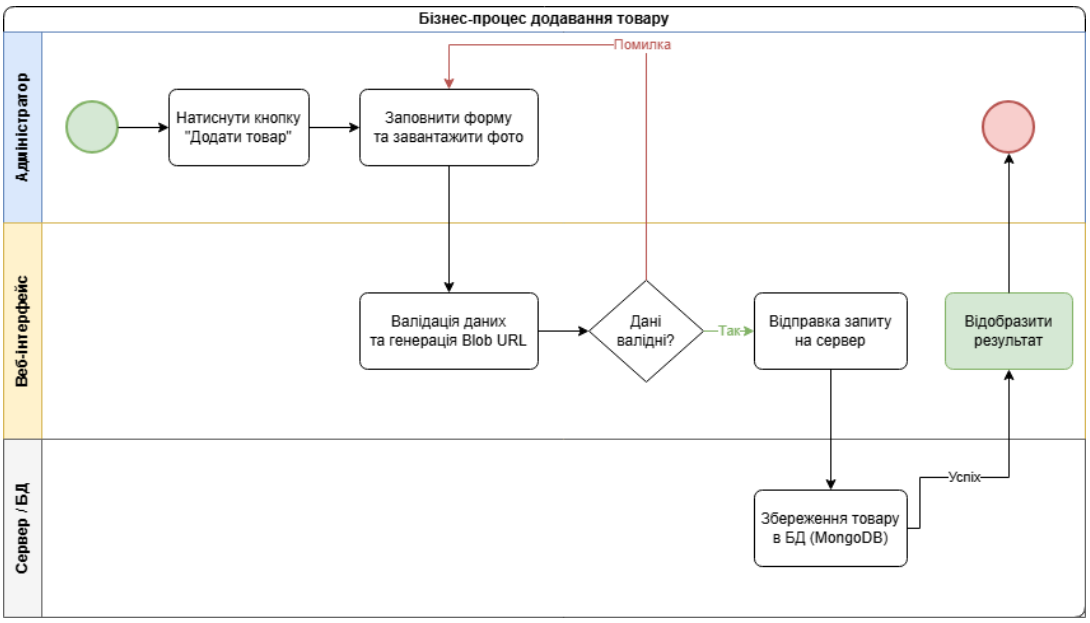


**Рис. 1.2. Модель бізнес-процесу створення товару (BPMN)**

Примітка. Джерело: складено на основі [4,6].

Для створення користувацького інтерфейсу обрано бібліотеку Material UI [9]. На відміну від класичних CSS-фреймворків, Material UI надає набір готових React-компонентів, адаптованих для роботи з сучасними SPA-застосунками. Це дозволяє спростити реалізацію елементів інтерфейсу, забезпечити адаптивність сторінок та підтримувати єдиний стиль оформлення відповідно до принципів Material Design [19].

У серверній частині системи використано середовище Node.js [10] та фреймворк Express.js. Вибір Node.js обґрунтований підтримкою асинхронної моделі обробки запитів, що є важливим для веб-застосунків із великою кількістю одночасних звернень користувачів. Додатковою перевагою є використання JavaScript як на клієнтській, так і на серверній стороні, що спрощує обмін даними між компонентами системи [4].

Окрему увагу приділено вибору системи зберігання даних. У традиційних реляційних базах даних можуть виникати труднощі при роботі з товарами, які мають велику кількість динамічних характеристик [6]. У таких випадках структура бази даних ускладнюється, а виконання складних SQL-запитів може негативно впливати на швидкодію системи. Для розв’язання цієї проблеми у

роботі обрано документо-орієнтовану NoSQL базу даних MongoDB [11]. Використання BSON-документів дозволяє гнучко зберігати інформацію про товари різних категорій без складної структури взаємопов'язаних таблиць. Це спрощує масштабування системи та роботу з динамічними характеристиками товарів.

Для наочного представлення взаємодії компонентів було сформовано загальну схему архітектури MERN-стека, яка демонструє процес обміну даними між клієнтською частиною, сервером та базою даних (рис. 1.3).

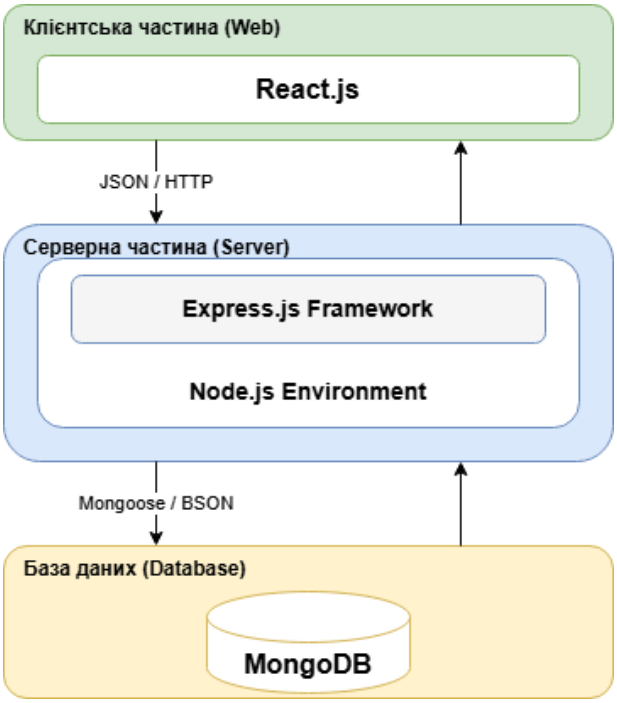


**Рис. 1.3. Архітектура взаємодії компонентів стеку MERN**

Примітка. Джерело: сформовано на основі [8,10].

Проведений аналіз показав, що технологічний стек MERN дозволяє забезпечити ефективну взаємодію між усіма компонентами системи. React відповідає за динамічне оновлення інтерфейсу, Node.js та Express.js – за обробку API-запитів, а MongoDB – за гнучке зберігання інформації у форматі BSON. Такий підхід дозволяє створити веб-застосунок, який відповідає вимогам електронної комерції та може бути адаптований до подальшого розширення функціоналу.

## Висновки до розділу 1

У першому розділі кваліфікаційної роботи проведено аналіз предметної області та сучасних тенденцій розвитку електронної комерції [1,2]. Встановлено, що зростання популярності онлайн-торгівлі та розвиток цифрових технологій формують підвищені вимоги до швидкодії, адаптивності та функціональності веб-застосунків. Визначено, що архітектура SPA дозволяє забезпечити динамічне оновлення контенту без повного перезавантаження сторінок, що позитивно впливає на швидкість взаємодії користувача із системою [4,6].

Під час аналізу існуючих аналогів інтернет-магазинів досліджено особливості платформ Amazon, Rozetka, COMFY та WooCommerce. Встановлено, що складні корпоративні системи потребують значних ресурсів для підтримки та масштабування, тоді як готові CMS-рішення можуть втрачати ефективність при значному збільшенні кількості товарів або навантаження на систему. У результаті визначено доцільність використання SPA-архітектури, REST API та документо-орієнтованої бази даних для створення власного веб-застосунку інтернет-магазину.

У підрозділі 1.3 сформульовано основні вимоги до створюваної системи та досліджено бізнес-логіку її функціонування. Розроблено BPMN-модель процесу створення товару та визначено основні ролі користувачів системи. Для реалізації поставлених завдань обґрунтовано вибір технологічного стеку MERN, до складу якого входять React [8], Node.js [10], Express.js та MongoDB [11]. Встановлено, що використання компонентного підходу, REST API та NoSQL бази даних дозволяє забезпечити достатню швидкодію системи, зручність адміністрування та можливість подальшого розширення функціоналу веб-застосунку.

Отримані результати створюють основу для подальшого проєктування структури бази даних, архітектури системи та реалізації програмної частини веб-застосунку у наступних розділах роботи.

# РОЗДІЛ 2
# ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА МОДЕЛІ ВЕБ-ДОДАТКА ОНЛАЙН-МАГАЗИНУ

## 2.1 Концептуальне проєктування та розробка архітектури системи

На етапі концептуального проєктування веб-додатка основну увагу було приділено формуванню структурованої архітектури, яка забезпечує зручну взаємодію між клієнтською та серверною частинами системи, підтримує подальше розширення функціоналу та дозволяє адаптувати програмний продукт до зростання кількості користувачів і товарів. Актуальна версія проєкту Nexora реалізована як веб-додаток інтернет-магазину на основі стеку MERN, до складу якого входять MongoDB, Express.js, React та Node.js.

Використання MERN-стека дозволяє застосовувати JavaScript як на клієнтській, так і на серверній стороні, що спрощує обмін даними між компонентами системи та полегшує підтримку програмного коду [22]. Для зберігання даних використовується MongoDB у поєднанні з ODM-бібліотекою Mongoose, а взаємодія між клієнтською та серверною частинами реалізована через REST API з передачею даних у форматі JSON.

Основні технології, використані під час розробки веб-додатка Nexora, наведено у таблиці 2.1.

Архітектура системи побудована за клієнт-серверною моделлю. Клієнтська частина реалізована як SPA-додаток (Single Page Application) на основі React, тоді як серверна частина функціонує у вигляді REST API на базі Node.js та Express.js [20]. Подібний підхід дозволяє розділити інтерфейсну та серверну логіку, що позитивно впливає на структурованість проєкту та спрощує подальшу підтримку окремих модулів системи.

SPA-підхід передбачає одноразове завантаження основної HTML-структури під час відкриття застосунку. Подальше оновлення сторінок виконується динамічно через асинхронні HTTP-запити без повного

перезавантаження браузера. Це дозволяє зменшити кількість повторних звернень до сервера та забезпечує більш плавну взаємодію користувача з інтерфейсом [24].

Таблиця 2.1

**Основні технології, використані у веб-додатку Nexora**

| Технологія | Призначення |
|---|---|
| React | Реалізація SPA-інтерфейсу |
| Vite | Збірка та запуск фронтенд-частини |
| Node.js | Серверне середовище виконання |
| Express.js | Побудова REST API |
| MongoDB | Зберігання даних |
| Mongoose | ODM для роботи з MongoDB |
| JWT | Автентифікація користувачів |
| Context API | Керування глобальним станом |

Примітка. Джерело: сформовано автором на основі використаних технологій проєкту.

Клієнтська частина має модульну структуру. Повторно використовувані елементи інтерфейсу винесені в окремі компоненти, сторінки системи згруповані у відповідних модулях, а взаємодія із серверною частиною реалізована через окремий сервісний шар. Такий підхід дозволяє ізолювати логіку HTTP-запитів від UI-компонентів та спрощує повторне використання функціоналу.

Взаємодія між компонентами системи також підтримує принципи масштабованості та ізоляції середовищ виконання, що було враховано під час подальшої контейнеризації застосунку.

Для керування станом клієнтської частини використано Context API та React Hooks. Основний додаток підключає AuthProvider, CompareProvider і CartProvider, які відповідають за збереження стану авторизації, порівняння товарів та кошика користувача. Частина даних додатково зберігається у localStorage. Зокрема, це стосується кошика товарів, списку переглянутих товарів, порівняння та окремих даних авторизації. Подібний підхід дозволяє

зберігати стан між оновленнями сторінки без постійного звернення до серверної частини.

Маршрутизація клієнтської частини реалізована за допомогою react-router-dom. Структура маршрутів охоплює головну сторінку, каталог товарів, картку товару, кошик, сторінку оформлення замовлення, порівняння товарів, особистий кабінет користувача та адміністративну панель. Каталог підтримує пошук, фільтрацію, сортування та пагінацію товарів, а сторінка товару містить характеристики, блок схожих товарів, систему відгуків і питань, а також механізм збереження переглянутих товарів.

Особистий кабінет користувача реалізований у вигляді окремого функціонального модуля з вкладками замовлень, списків бажань, переглянутих товарів, профілю та історії взаємодії з товарами. На відміну від попередньої версії системи, списки бажань підтримують створення декількох окремих добірок користувача. Це дозволяє групувати товари за різними категоріями або сценаріями використання. Частина логіки роботи списків бажань реалізована через локальний стан клієнтської частини, а дані синхронізуються із сервером після авторизації користувача.

Окрему увагу приділено реалізації механізмів порівняння товарів та збереження історії переглядів. Функціонал порівняння побудований на основі CompareContext та localStorage і дозволяє користувачу додавати товари до списку порівняння, видаляти їх або очищати список. Історія переглянутих товарів також реалізована переважно на клієнтському боці: під час відкриття сторінки товар автоматично додається до локального списку, який надалі відображається в особистому кабінеті користувача.

Важливою складовою системи є процес оформлення замовлення, який реалізовано у вигляді багатокрокового сценарію. На першому етапі користувач вводить контактні дані та обирає спосіб доставки. Система підтримує декілька варіантів доставки, зокрема самовивіз, доставку до відділення та кур’єрську доставку. Для роботи з доставкою Нової Пошти використовуються окремі модальні компоненти вибору міста та відділення, які взаємодіють із серверною

частиною через API доставки.

Після введення контактної інформації виконується перевірка обов’язкових полів на стороні клієнта. Наступний етап відповідає за вибір способу оплати та підтвердження замовлення. У поточній реалізації підтримується вибір способу оплати (cash або card), однак повноцінна інтеграція з платіжним шлюзом у коді не виявлена. Після успішного створення замовлення кошик очищується, а користувач перенаправляється до вкладки замовлень особистого кабінету.

Серверна частина системи реалізована на Node.js та Express.js. Серверна частина поєднує використання окремих models, controllers і routes із частковим розміщенням бізнес-логіки безпосередньо у route-модулях. Подібний підхід дозволяє поступово розширювати функціонал системи без повного перепроєктування серверної логіки.

Для взаємодії з базою даних використовується Mongoose. Основні сутності системи представлені окремими моделями користувачів, товарів, категорій, брендів, замовлень, відгуків і питань. Водночас структура даних у системі не є повністю реляційною. Частина зв’язків реалізована через ObjectId, а частина — через рядкові поля. Наприклад, у моделі товару категорія та бренд зберігаються як рядкові значення, а не як прямі посилання на окремі документи.

Одним із важливих компонентів системи є механізм авторизації та розмежування доступу. У поточній реалізації використовуються ролі user, admin та moderator. Це дозволяє реалізувати рольову модель доступу, відповідно до якої користувачі отримують доступ лише до тих функцій, які відповідають їхнім повноваженням.

Роль user використовується для роботи з клієнтською частиною системи: оформлення замовлень, роботи зі списками бажань, перегляду історії замовлень, створення відгуків та питань до товарів. Роль admin має доступ до адміністративної панелі, інформаційної панелі, товарів, категорій, замовлень і системи модерації. Для ролі moderator передбачено доступ до модерації відгуків і питань, а також до сторінки клієнтів.

Адміністративна панель винесена в окремий функціональний модуль із

захищеними маршрутами. Авторизація адміністратора виконується через JWT-механізм із подальшим збереженням токена у localStorage. Захист адміністративних маршрутів реалізований через компонент AdminProtectedRoute, який перевіряє роль користувача та наявність валідного токена.

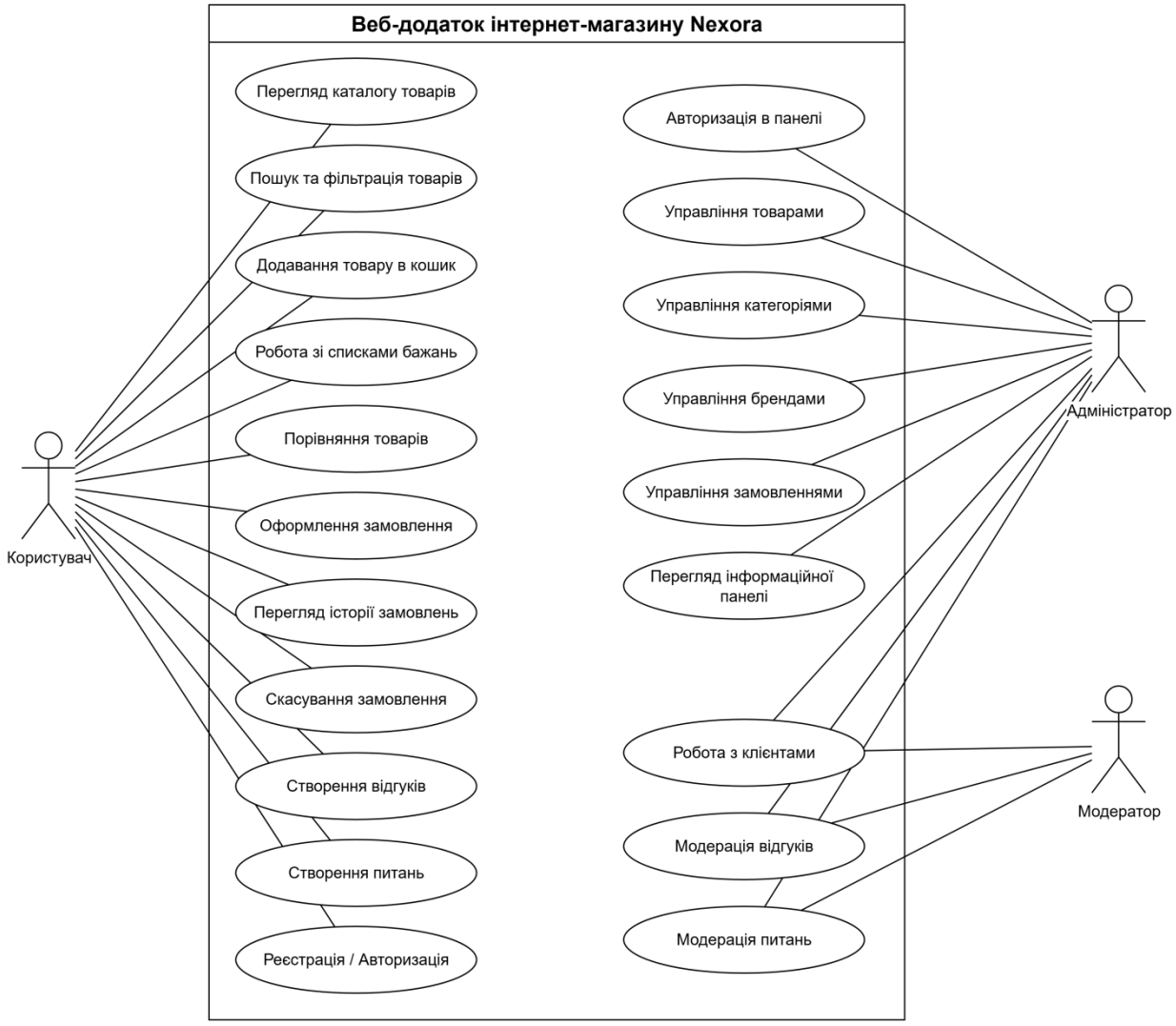


**Рис. 2.1. Діаграма варіантів використання (Use Case Diagram)**

Примітка. Джерело: складено автором на основі [20,25].

Структура адміністративної панелі охоплює декілька основних модулів. Інформаційна панель містить агреговані показники замовлень, обороту, залишків товарів та модерації контенту. Модуль управління товарами дозволяє виконувати CRUD-операції для товарів, брендів і характеристик. Окремо реалізовано систему керування категоріями, яка підтримує роботу з іконками,

кольорами та типовими характеристиками категорій.

Модуль управління замовленнями підтримує пошук, фільтрацію, сортування та зміну статусів замовлень. Для замовлень використовуються статуси new, confirmed, packing, ready_for_pickup, received та cancelled. Система також підтримує історію змін статусів і механізм скасування замовлення як із боку користувача, так і з боку адміністратора.

Окремий модуль адміністративної панелі призначений для роботи з клієнтами. Сторінка клієнтів дозволяє переглядати список користувачів, аналізувати історію замовлень, середній чек, загальну суму покупок та інші агреговані показники. Також передбачено механізм блокування та розблокування користувачів через зміну статусу облікового запису.

Додаткову увагу приділено системі модерації відгуків і питань користувачів. Нові відгуки та питання створюються зі статусом pending і проходять перевірку перед публікацією. В адміністративній панелі модератор або адміністратор може змінювати статус запису, публікувати або відхиляти контент, а також відповідати на питання користувачів. На сторінці товару відображаються лише записи зі статусом approved.

У процесі концептуального проєктування також було враховано особливості реалізації каталогу товарів. Поточна версія системи використовує переважно клієнтську фільтрацію: після початкового завантаження товарів фільтрація, пошук, сортування та частина пагінації виконуються на стороні клієнта. Такий підхід дозволяє швидше оновлювати інтерфейс без повторних HTTP-запитів під час кожної зміни параметрів фільтрації.

Для запуску та збірки клієнтської частини використовується Vite [29]. Використання цього інструмента дозволяє скоротити час запуску середовища розробки та прискорити оновлення модулів під час роботи з frontend частиною системи.

Отже, у результаті концептуального проєктування було сформовано архітектуру веб-додатка електронної комерції, яка поєднує React SPA, REST API на Express.js та MongoDB із використанням Mongoose. Реалізована структура

підтримує розмежування ролей користувачів, модульну організацію інтерфейсу, систему модерації контенту, багатокрокове оформлення замовлень та адміністративну панель із інформаційною панеллю і клієнтською аналітикою. Це створює основу для подальшого розвитку функціоналу системи та підтримки її масштабування.

## 2.2 Проєктування структури бази даних та опис існуючих обмежень

Одним із ключових етапів розробки веб-додатка електронної комерції є проєктування структури бази даних, оскільки саме від організації зберігання інформації залежить швидкодія системи, зручність роботи з даними та можливість подальшого розширення функціоналу. У проєкті Nexora для зберігання інформації використовується документо-орієнтована NoSQL база даних MongoDB у поєднанні з бібліотекою Mongoose, яка забезпечує створення схем та моделей даних у середовищі Node.js.

Вибір MongoDB обґрунтований особливостями предметної області електронної комерції. Каталог товарів може містити значну кількість категорій із різними характеристиками, що ускладнює використання жорстко фіксованої реляційної структури [11]. На відміну від традиційних SQL-баз даних, MongoDB дозволяє зберігати документи з різною структурою полів, що є важливим для роботи з товарами, які мають відмінні набори характеристик залежно від категорії.

Структура бази даних системи побудована на основі окремих колекцій, що відповідають основним сутностям предметної області. Серед основних колекцій можна виділити користувачі, товари, категорії, бренди, замовлення, відгуки, питання користувачів та списки бажань. Частина зв’язків між сутностями реалізована через ObjectId-посилання, а частина — через рядкові поля або вкладені документи. Подібний підхід дозволяє зберігати гнучкість структури даних та уникати надмірного ускладнення моделей.

Основні колекції бази даних MongoDB та їх призначення наведено у

таблиці 2.2.

Таблиця 2.2

**Основні колекції бази даних MongoDB**

| Колекція | Призначення |
|---|---|
| Users | Зберігання облікових записів користувачів |
| Products | Зберігання інформації про товари |
| Categories | Категорії товарів |
| Brands | Дані про бренди |
| Orders | Інформація про замовлення |
| Reviews | Відгуки користувачів |
| Questions | Питання до товарів |
| WishlistLists | Списки бажань користувачів |

Примітка. Джерело: сформовано автором на основі структури бази даних веб-додатка Nexora.

Однією з центральних сутностей системи є модель користувача (User). Вона містить основні дані облікового запису: ім’я користувача, електронну пошту, пароль, номер телефону, аватар та роль у системі. Для розмежування доступу використовуються ролі user, admin та moderator. Також модель містить службові поля, пов’язані зі статусом облікового запису, історією створення профілю та окремими параметрами активності користувача.

Модель користувача пов’язана із замовленнями, відгуками, питаннями та списками бажань. При цьому частина взаємодій реалізована через ObjectId-посилання, а частина — через вкладені масиви MongoDB. Наприклад, окремі дані про товари у списках бажань можуть дублюватися у вигляді вкладених об’єктів для спрощення відображення інформації на клієнтській стороні.

Основною сутністю каталогу є модель товару (Product). Вона містить назву товару, опис, ціну, знижку, список зображень, характеристики, рейтинг, кількість відгуків та службові параметри. Окремо реалізовано підтримку динамічних атрибутів товарів, що дозволяє використовувати різні набори характеристик для окремих категорій продукції.

Для підтримки механізму акційних цін у структурі моделі Product додатково використовується поле compareAtPrice, яке зберігає попередню вартість товару. Поточна ціна товару визначається через поле price, тоді як

compareAtPrice застосовується для відображення перекресленої старої ціни та формування інформації про знижку у клієнтській частині веб-додатка. Фрагмент структури моделі Product наведено у додатку А.

Особливістю поточної реалізації є змішане використання ObjectId та рядкових полів. Наприклад, категорія та бренд у моделі товару можуть зберігатися у вигляді рядкових значень, а не лише як прямі ObjectId-посилання на окремі документи. Такий підхід спрощує частину операцій фільтрації та відображення даних, однак може призводити до дублювання окремих значень у різних колекціях.

Для структурування каталогу використовується модель категорій (Category). Вона містить назву категорії, slug, іконку, колір оформлення та набір стандартних характеристик (defaultAttributes). Використання стандартних атрибутів категорій дозволяє автоматично формувати перелік характеристик під час створення товарів певної категорії. Наприклад, для смартфонів можуть використовуватися характеристики процесора, обсягу пам’яті або ємності акумулятора, тоді як для ноутбуків — тип відеокарти чи діагональ екрана.

Окремо реалізовано модель брендів (Brand), яка використовується для групування товарів за виробниками та побудови системи фільтрації каталогу. Структура моделі є відносно простою та містить назву бренду, slug і службові поля.

Важливою сутністю системи є модель замовлення (Order). Вона містить інформацію про користувача, перелік товарів, спосіб доставки, контактні дані, адресу отримання, тип оплати та поточний статус замовлення. У структурі замовлення частина даних зберігається у вигляді вкладених документів. Зокрема, це стосується товарів у складі замовлення, де додатково зберігаються назва товару, ціна на момент покупки та кількість одиниць.

У структурі товарів замовлення також може зберігатися значення compareAtPrice, що дозволяє фіксувати попередню вартість товару на момент оформлення покупки та забезпечує коректне відображення акційних цін в історії замовлень незалежно від подальших змін вартості продукції.

Для замовлень реалізовано систему статусів, яка дозволяє відстежувати етапи обробки заявки. Поточна версія системи підтримує статуси new, confirmed, packing, ready_for_pickup, received та cancelled. Крім основного статусу, модель містить історію змін статусів у вигляді окремого масиву. Це дозволяє зберігати інформацію про час зміни статусу та дії адміністратора або модератора.

Також реалізовано механізм скасування замовлення. Скасування може виконуватися як користувачем, так і адміністратором системи. У структурі документа додатково зберігається причина скасування та службові параметри, пов’язані з виконанням цієї операції.

Окремі моделі системи відповідають за роботу з відгуками (Review) та питаннями користувачів (Question). Обидві сутності використовуються для організації взаємодії користувачів із товарами та підтримують механізм модерації контенту. Відгуки містять текст повідомлення, оцінку товару, інформацію про автора та статус модерації. Аналогічно реалізовано й модель питань користувачів.

Нові записи створюються зі статусом pending і не відображаються у відкритій частині платформи до завершення перевірки. Після модерації адміністратор або модератор може змінити статус на approved або rejected. Такий підхід дозволяє зменшити кількість небажаного або некоректного контенту в системі.

Для реалізації функціоналу списків бажань використовується окрема сутність (WishlistList). Поточна версія системи підтримує створення декількох списків бажань одним користувачем. Кожен список містить власну назву та масив товарів, які можуть зберігатися у вигляді ObjectId-посилань або вкладених даних. Подібна структура дозволяє групувати товари за різними сценаріями використання, наприклад окремо для техніки, аксесуарів або майбутніх покупок.

У структурі бази даних також враховано особливості реалізації клієнтської аналітики. Частина агрегованих показників для адміністративної панелі формується на основі даних замовлень та користувачів. Зокрема, використовуються обчислення загальної суми покупок, середнього чека,

кількості замовлень та активності користувачів. При цьому значна частина аналітичних операцій виконується без створення окремого сховища статистичних даних.

Для більш наочного представлення структури бази даних було сформовано логічну модель взаємозв'язків між основними сутностями системи (рис. 2.2).

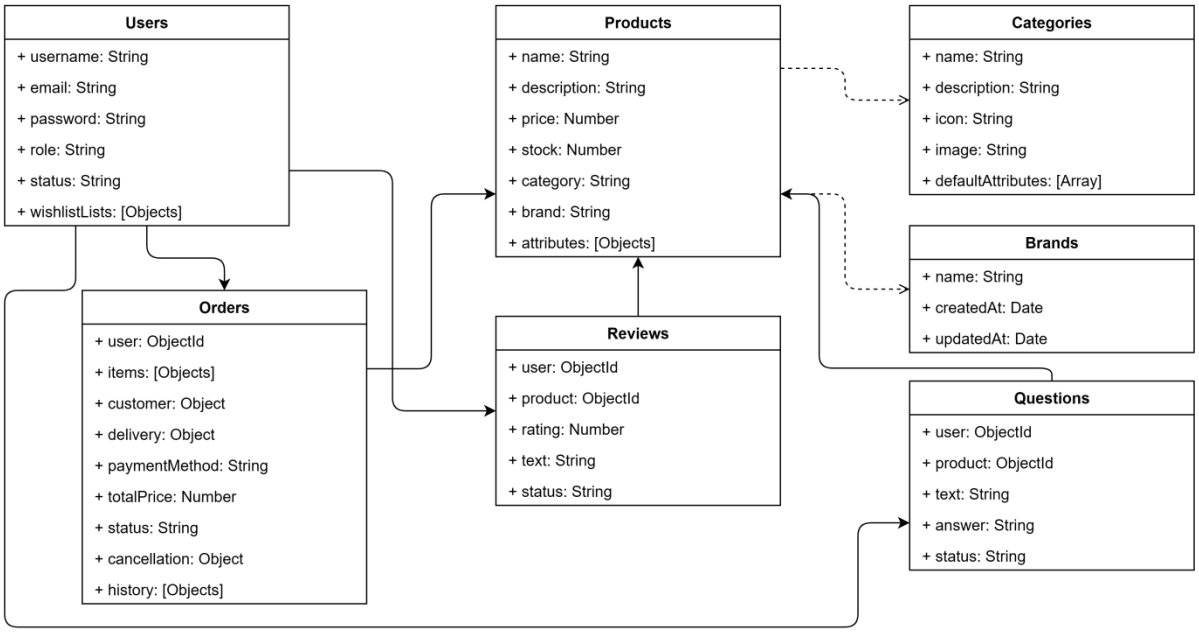


**Рис. 2.2. Логічна модель структури бази даних веб-додатка**

Примітка. Джерело: складено автором на основі [21,23].

Логічна модель відображає взаємозв'язки між користувачами, товарами, категоріями, брендами, замовленнями, відгуками, питаннями та списками бажань. Частина зв'язків реалізована через ObjectId-посилання, тоді як для окремих структур використовуються вкладені документи та масиви MongoDB. Подібний підхід дозволяє поєднати гнучкість документо-орієнтованої моделі з можливістю організації взаємодії між основними сутностями системи.

Проведений аналіз структури бази даних показав, що використання MongoDB дозволяє реалізувати гнучку модель зберігання інформації для системи електронної комерції. Використання вкладених документів, масивів та документо-орієнтованого підходу спрощує роботу з динамічними

характеристиками товарів і дозволяє адаптувати структуру системи до подальшого розширення функціоналу. Водночас змішане використання ObjectId-посилань і рядкових полів може ускладнювати підтримку цілісності окремих даних та потребує додаткового контролю на рівні серверної логіки.

## 2.3 Алгоритмічне забезпечення взаємодії клієнтської та серверної частин

У процесі розробки веб-додатка Nexora було реалізовано набір алгоритмів, які забезпечують взаємодію між клієнтською та серверною частинами системи. Основою обміну даними між компонентами застосунку став REST API, побудований на HTTP-протоколі з використанням JSON-формату для передачі інформації [22,26]. Використання такого підходу дозволяє розділити клієнтську та серверну логіку, спрощує підтримку окремих модулів системи та забезпечує гнучкість під час подальшого розширення функціоналу.

Клієнтська частина реалізована як SPA-додаток на основі React. Взаємодія із сервером виконується через асинхронні HTTP-запити, які надсилаються під час завантаження каталогу товарів, авторизації користувача, оформлення замовлень, роботи зі списками бажань та інших функціональних операцій. Серверна частина на базі Express.js приймає запити, виконує обробку даних, взаємодіє з MongoDB та повертає відповідь у JSON-форматі. Це забезпечує зручне перетворення даних між JavaScript-об’єктами клієнтської та серверної частин.

Важливим компонентом алгоритмічного забезпечення є механізм автентифікації та авторизації користувачів. Для захисту облікових записів використовується JWT-механізм у поєднанні з бібліотекою bcrypt для хешування паролів [12,13]. Під час реєстрації пароль не зберігається у відкритому вигляді. Перед записом до бази даних виконується його хешування із використанням додаткової «солі», що підвищує рівень захисту облікових даних.

Алгоритм автентифікації складається з декількох послідовних етапів.

Спочатку користувач вводить електронну пошту та пароль через форму входу. Після надсилання даних сервер виконує пошук користувача за email у базі MongoDB, а далі за допомогою bcrypt здійснюється порівняння введеного пароля з хешованим значенням, яке зберігається у базі даних.

У разі успішного проходження перевірки сервер генерує JWT-токен із визначеним терміном дії. У службовій частині JWT-токена зберігається службова інформація про користувача, зокрема його ідентифікатор та роль у системі. Згенерований токен повертається клієнтській частині та використовується для доступу до захищених ресурсів API.

Після авторизації клієнт автоматично додає токен до HTTP-запитів через заголовок Authorization: Bearer. Серверна частина перевіряє валідність токена перед наданням доступу до захищених маршрутів. Якщо токен є недійсним або термін його дії завершився, доступ до ресурсу блокується. Такий механізм дає змогу реалізувати безстанну модель авторизації без необхідності зберігання серверних сесій (рис. 2.3).

У системі реалізовано рольову модель доступу, яка передбачає використання ролей user, admin та moderator. Роль user використовується для роботи з клієнтською частиною системи: оформлення замовлень, створення списків бажань, перегляду історії замовлень, додавання відгуків та питань до товарів. Роль admin забезпечує доступ до адміністративної панелі, управління товарами, категоріями, замовленнями та користувачами. Для ролі moderator передбачено доступ до системи модерації контенту та роботи з клієнтськими зверненнями.

Важливим елементом алгоритмічного забезпечення є процес оформлення замовлення. Оформлення реалізовано у вигляді багатокрокового сценарію, під час якого користувач послідовно вводить контактні дані, обирає спосіб доставки та підтверджує замовлення. На першому етапі виконується перевірка обов’язкових полів на стороні клієнта. Після цього інформація передається до серверної частини для створення нового документа замовлення у базі MongoDB.

Система підтримує декілька способів доставки, зокрема самовивіз,

доставку до відділення та кур’єрську доставку. Для взаємодії з сервісом Нової Пошти використовуються окремі API-запити, які дозволяють отримувати список міст і доступних відділень. Вибрані користувачем параметри доставки передаються під час формування замовлення.

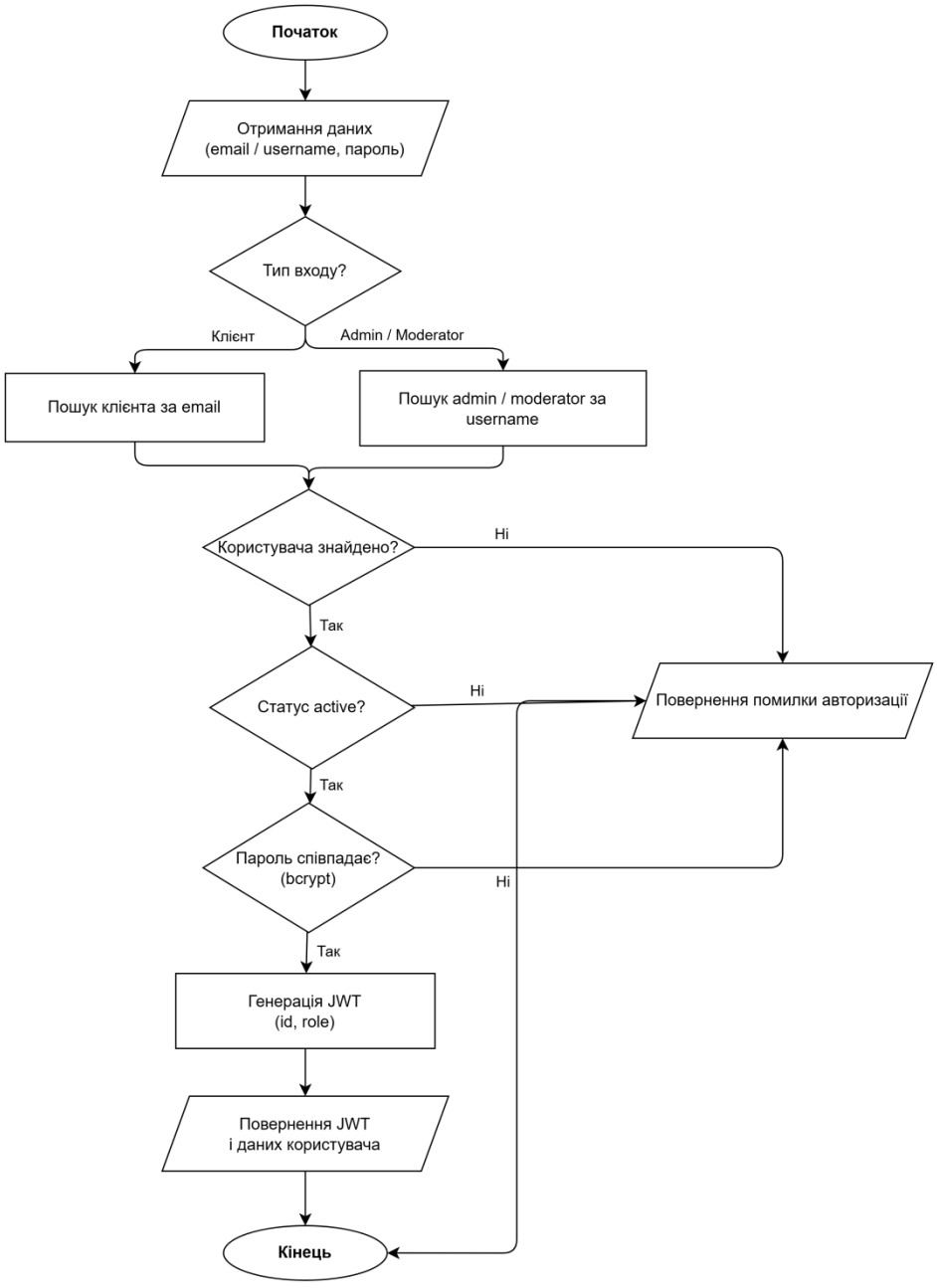


**Рис. 2.3. Схема алгоритму автентифікації користувачів**

Примітка. Джерело: розроблено автором.

Після успішного створення замовлення кошик очищується, а користувач

отримує доступ до перегляду інформації про оформлене замовлення у власному кабінеті. У поточній реалізації підтримується вибір способу оплати cash або card, однак повноцінна інтеграція з платіжним шлюзом у системі відсутня (рис. 2.4).

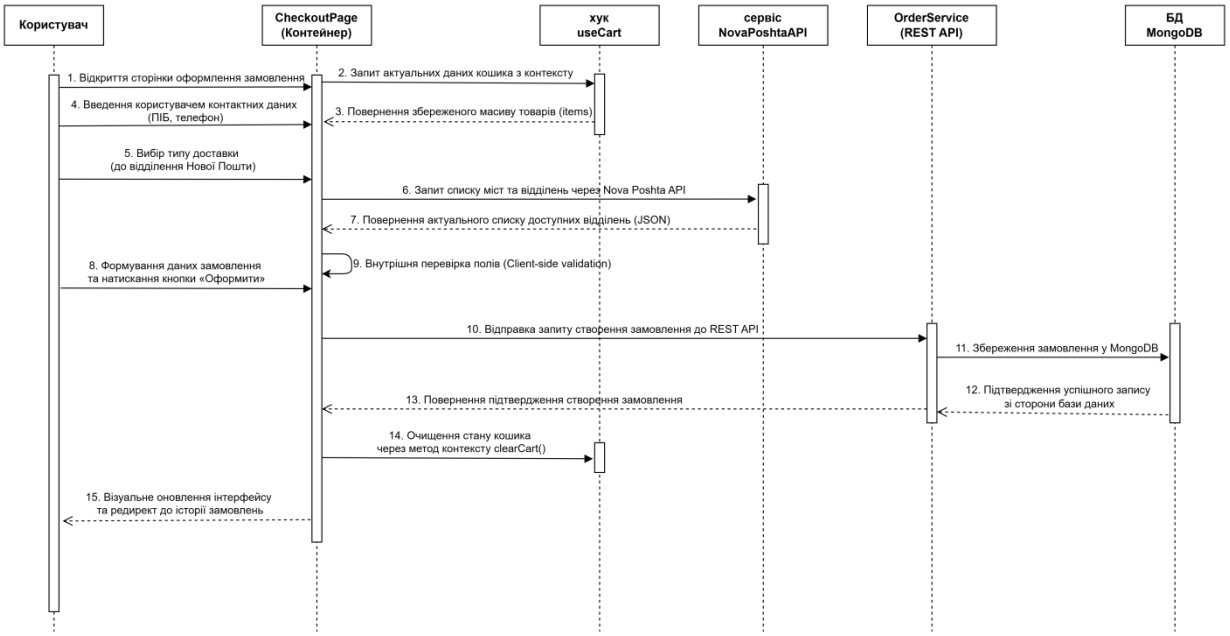


**Рис. 2.4. Діаграма процесу оформлення замовлення**

Примітка. Джерело: розроблено автором.

У системі реалізовано алгоритм керування життєвим циклом замовлення. Після створення замовлення система присвоює йому статус new. У процесі подальшої обробки статус може змінюватися на confirmed, packing, ready_for_pickup, received або cancelled. Історія змін статусів зберігається у вигляді окремого масиву, що дозволяє відстежувати послідовність обробки замовлення.

Також реалізовано механізм скасування замовлення. Користувач може скасувати замовлення до завершення його обробки, тоді як адміністратор має можливість змінювати статус замовлення відповідно до поточного етапу виконання. Додатково зберігається причина скасування та службова інформація про виконану дію.

Значна увага приділена алгоритмам роботи каталогу товарів. Поточна

реалізація використовує переважно клієнтську фільтрацію. Після початкового завантаження список товарів зберігається у стані React-додатка, а подальша фільтрація, сортування та частина пагінації виконуються без повторних звернень до сервера [24]. Це дозволяє зменшити кількість HTTP-запитів та забезпечити більш швидке оновлення інтерфейсу.

Алгоритм пошуку виконує перевірку входження текстового запиту до назви товару без урахування регістру символів. Під час фільтрації також враховуються категорія, бренд, ціновий діапазон та окремі характеристики товарів. Результати можуть додатково сортуватися за ціною або популярністю. Використання клієнтської фільтрації є доцільним для поточного обсягу каталогу, однак при значному збільшенні кількості товарів може виникнути необхідність перенесення частини логіки на серверну сторону (рис. 2.5).

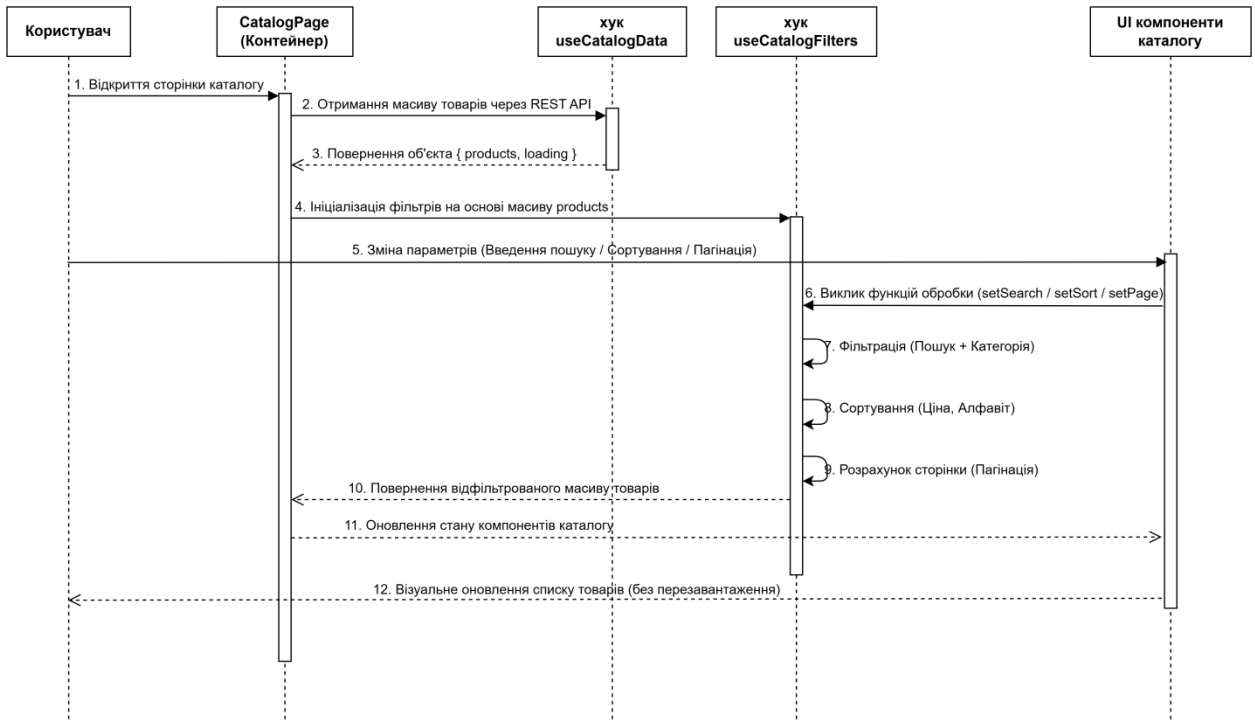


**Рис. 2.5. Діаграма послідовності процесу фільтрації товарів**

Примітка. Джерело: розроблено автором.

Важливим компонентом системи є реалізація кошика товарів та списків бажань. Для керування станом кошика використовується Context API у

поєднанні з useReducer, що дозволяє централізовано організувати роботу зі станом клієнтської частини. Під час додавання товару алгоритм перевіряє його наявність у кошику та, за необхідності, оновлює кількість одиниць товару.

Для збереження даних між оновленнями сторінки використовується localStorage. Після кожної зміни стану оновлена інформація серіалізується у JSON-форматі та записується до локального сховища браузера. Під час повторного відкриття додатка дані автоматично відновлюються у Context API. Це дає змогу зберігати вміст кошика та окремі користувацькі налаштування навіть після закриття браузера [24].

Списки бажань підтримують створення декількох окремих добірок товарів одним користувачем. Частина логіки роботи реалізована на клієнтському боці, однак після авторизації користувача дані синхронізуються із серверною частиною та зберігаються у MongoDB.

У системі також реалізовано механізм порівняння товарів і збереження історії переглянутих товарів. Список порівняння формується через CompareContext та localStorage, що дозволяє додавати або видаляти товари без повторного звернення до сервера. Історія переглянутих товарів формується автоматично під час відкриття сторінки товару та використовується для подальшого відображення у профілі користувача.

Окремим напрямом алгоритмічного забезпечення стала модерація відгуків і питань користувачів. Нові записи створюються зі статусом pending та не відображаються у відкритій частині системи до завершення перевірки. Після модерації адміністратор або модератор може змінити статус запису на approved або rejected. Це забезпечує контроль якості контенту та дозволяє зменшити кількість небажаних повідомлень (рис. 2.6).

У межах адміністративної панелі реалізовано алгоритми формування агрегованих показників для інформаційної панелі та клієнтської аналітики. Для цього використовуються дані замовлень, користувачів та товарів. Система виконує підрахунок кількості замовлень, загального обороту, середнього чека, кількості активних користувачів та інших статистичних показників. Частина

обчислень виконується безпосередньо під час формування відповіді сервера, без використання окремого аналітичного сховища.

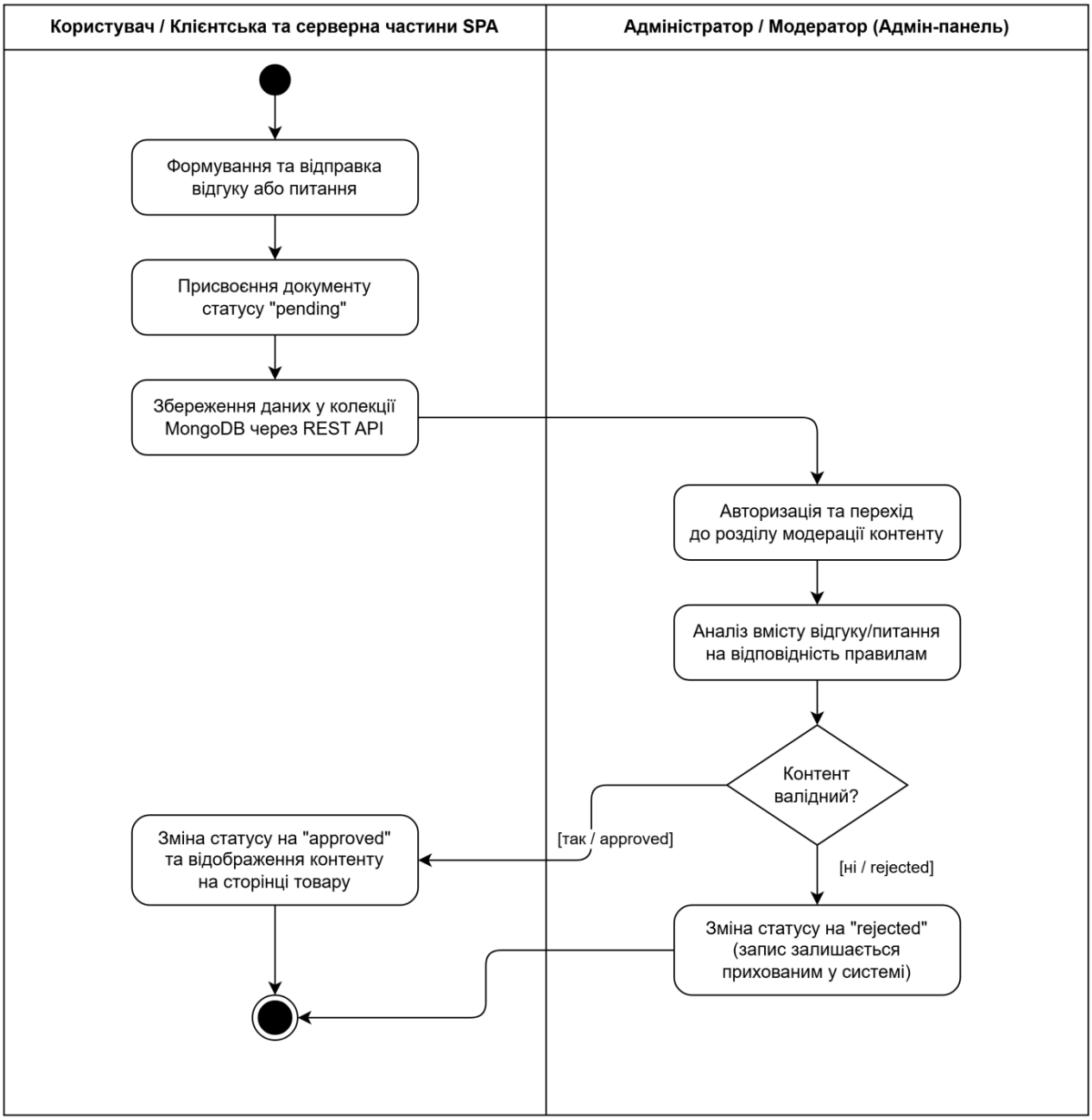


**Рис. 2.6. Схема процесу модерації відгуків і питань**

Примітка. Джерело: розроблено автором.

Отже, було розглянуто основні алгоритми взаємодії між клієнтською та серверною частинами веб-додатка Nexora. REST API забезпечує передачу даних між компонентами системи, JWT-механізм відповідає за автентифікацію користувачів, клієнтська фільтрація оптимізує роботу каталогу, а Context API та localStorage використовуються для керування станом кошика, списків бажань і

порівняння товарів. Реалізоване алгоритмічне забезпечення створює основу для стабільної роботи SPA-додатка інтернет-магазину та подальшого розвитку його функціональних можливостей.

## Висновки до розділу 2

У другому розділі кваліфікаційної роботи було розглянуто особливості проєктування архітектури веб-додатка інтернет-магазину Nexora та досліджено принципи організації взаємодії між його основними компонентами. У результаті проведеного аналізу сформовано клієнт-серверну архітектуру на основі стеку MERN, що поєднує React SPA-додаток, серверну частину на Node.js та Express.js, а також документо-орієнтовану базу даних MongoDB. Використання SPA-підходу дозволило реалізувати динамічне оновлення інтерфейсу без повного перезавантаження сторінок, що позитивно впливає на швидкість взаємодії користувача із системою.

У підрозділі 2.2 було спроєктовано структуру бази даних відповідно до актуальної реалізації проєкту. Визначено основні сутності системи, зокрема користувачів, товари, категорії, бренди, замовлення, відгуки, питання та списки бажань. Під час проєктування враховано особливості документо-орієнтованої моделі MongoDB, використання вкладених документів, масивів та змішаного застосування ObjectId-посилань і рядкових полів. Також сформовано логічну модель взаємозв’язків між основними сутностями системи та підготовлено схему структури бази даних.

У підрозділі 2.3 досліджено алгоритмічне забезпечення взаємодії клієнтської та серверної частин веб-додатка. Розглянуто механізми роботи REST API, JWT-автентифікації та рольової моделі доступу для ролей user, admin і moderator. Описано алгоритми оформлення замовлення, клієнтської фільтрації каталогу, роботи списків бажань, порівняння товарів, збереження історії переглядів та модерації користувацького контенту. Для керування станом клієнтської частини використано Context API та localStorage. Також було

визначено доцільність використання UML-діаграм, схем автентифікації, процесу оформлення замовлення та модерації відгуків і питань для візуального представлення основних процесів системи.

Отримані результати створюють основу для подальшої реалізації програмної частини веб-додатка, тестування функціональних модулів та дослідження ефективності роботи системи в умовах реального використання.

# РОЗДІЛ 3
# ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКА ТА АДМІНІСТРАТИВНОЇ ПАНЕЛІ НА ОСНОВІ MERN-СТЕКУ

## 3.1 Обґрунтування вибору інструментарію розробки та вимоги до програмно-технічного забезпечення

Під час розробки веб-додатка інтернет-магазину Nexora основну увагу було приділено вибору програмного інструментарію, який забезпечує підтримку SPA-архітектури, взаємодію між клієнтською та серверною частинами системи, а також можливість подальшого розширення функціоналу. Для реалізації проєкту було обрано стек MERN, до складу якого входять MongoDB, Express.js, React та Node.js. Використання цього стеку є доцільним для створення сучасних веб-додатків електронної комерції, оскільки він дозволяє реалізувати повноцінну клієнт-серверну архітектуру із використанням JavaScript як на клієнтській, так і на серверній стороні.

Застосування єдиної мови програмування для різних частин системи спрощує підтримку програмного коду та обмін даними між компонентами веб-додатка. Крім того, використання SPA-архітектури дозволяє забезпечити динамічне оновлення інтерфейсу без повного перезавантаження сторінок, що позитивно впливає на швидкість роботи інтерфейсу та покращує зручність взаємодії користувача із веб-додатком [24].

Клієнтська частина веб-додатка реалізована із використанням бібліотеки React [8,24]. React підтримує компонентний підхід до побудови інтерфейсу, відповідно до якого окремі елементи сторінок реалізуються у вигляді незалежних компонентів. Такий підхід спрощує повторне використання інтерфейсних елементів, підвищує структурованість проєкту та полегшує подальшу підтримку коду.

Однією з важливих особливостей React є використання механізму Virtual DOM, який забезпечує оптимізоване оновлення інтерфейсу. Під час зміни стану

компонента React виконує порівняння віртуальної копії DOM-структури із поточним станом сторінки та оновлює лише необхідні елементи. Це дозволяє зменшити кількість зайвих операцій перерендерингу та підвищити швидкодію клієнтської частини веб-додатка.

Для реалізації маршрутизації використовується бібліотека React Router DOM. Вона забезпечує переходи між сторінками без повного перезавантаження браузера та дозволяє реалізувати логіку захищених маршрутів для авторизованих користувачів і адміністративної панелі. Використання клієнтської маршрутизації є важливим елементом SPA-архітектури, оскільки дозволяє підтримувати безперервну взаємодію користувача з веб-додатком.

Як інструмент збірки клієнтської частини було обрано Vite [29]. На відміну від традиційних систем збірки, Vite використовує сучасний підхід до роботи з ES-модулями, що дозволяє скоротити час запуску локального середовища розробки та пришвидшити оновлення модулів під час внесення змін до коду. Це є особливо важливим для великих SPA-проєктів, які містять значну кількість компонентів та залежностей.

Для побудови інтерфейсу використовується бібліотека Material UI [9], яка надає набір готових React-компонентів для створення адаптивного інтерфейсу. Використання Material UI дозволяє підтримувати єдиний стиль оформлення сторінок та спрощує реалізацію типових UI-елементів, зокрема кнопок, форм, модальних вікон та елементів навігації.

Додатково для стилізації компонентів і сторінок використовується SCSS. На відміну від звичайного CSS, SCSS підтримує вкладеність, змінні, міксіни та поділ стилів на окремі модулі. Це дозволяє структурувати стилі веб-додатка та спрощує підтримку великої кількості інтерфейсних компонентів.

Для централізованого керування станом клієнтської частини використовується Context API у поєднанні з хуками React. Його застосування дозволяє організувати передачу даних між компонентами без необхідності багаторівневого передавання параметрів через props. У поточній реалізації Context API використовується для роботи зі станом авторизації, кошиком

товарів, списками порівняння та окремими елементами взаємодії користувача із системою.

Серверна частина веб-додатка реалізована на платформі Node.js [10] із використанням фреймворку Express.js [26]. Node.js забезпечує виконання JavaScript-коду на серверній стороні та підтримує неблокувальну асинхронну модель обробки запитів. Це дозволяє ефективно обробляти велику кількість одночасних HTTP-звернень без створення окремого потоку для кожного користувача.

Фреймворк Express.js використовується для побудови REST API та організації серверної логіки. За допомогою Express.js реалізовано обробку HTTP-запитів, маршрутизацію, роботу із middleware-компонентами та взаємодію з базою даних. Використання REST API дозволяє відокремити клієнтську та серверну частини системи, що спрощує масштабування проєкту та підтримку окремих модулів.

Для роботи зі змінними середовища використовується бібліотека dotenv. Її застосування дозволяє зберігати конфіденційні параметри, зокрема JWT-secret, адреси підключення до бази даних та інші службові налаштування, поза межами основного програмного коду.

Авторизація користувачів реалізована за допомогою JSON Web Token. JWT-механізм використовується для створення токенів доступу, які передаються між клієнтською та серверною частинами під час роботи із захищеними ресурсами системи. Для додаткового захисту облікових даних використовується бібліотека bcryptjs, яка забезпечує хешування паролів перед їх збереженням у базі даних.

Для зберігання даних у веб-додатку використовується MongoDB [11] — документо-орієнтована NoSQL база даних. Її використання є доцільним для систем електронної комерції, оскільки структура товарів може містити різні набори характеристик залежно від категорії продукції. Документо-орієнтований підхід дозволяє зберігати дані у форматі BSON-документів без жорстко фіксованої структури таблиць.

Для взаємодії серверної частини з MongoDB використовується бібліотека Mongoose [27]. Вона забезпечує створення схем, моделей та валідацію даних перед записом до бази. Крім того, Mongoose спрощує організацію зв’язків між окремими сутностями системи та дозволяє реалізувати централізовану роботу з моделями даних.

Основні інструменти та технології, використані під час реалізації веб-додатка Nexora, наведено в таблиці 3.1.

Таблиця 3.1

**Основні інструменти та технології веб-додатка Nexora**

| Технологія | Версія | Призначення у проєкті |
|---|---|---|
| React | 19.2.6 | Побудова клієнтського SPA-інтерфейсу |
| Vite | 8.0.11 | Локальний dev server, HMR та production-збірка клієнтської частини |
| React Router DOM | 7.15.0 | Клієнтська маршрутизація без перезавантаження сторінок |
| Material UI | 7.3.11 | Реалізація адаптивних UI-компонентів інтерфейсу |
| SCSS / Sass | 1.99.0 | Стилізація компонентів і сторінок |
| Node.js | 22 LTS | Серверне середовище виконання JavaScript |
| Express.js | 4.22.1 | Побудова REST API серверної частини |
| MongoDB Driver | 5.9.2 | Низькорівнева взаємодія серверної частини з MongoDB |
| Mongoose | 7.8.9 | Моделювання даних і робота зі схемами MongoDB |
| JSON Web Token | 9.0.3 | Авторизація користувачів через токени |
| bcryptjs | 2.4.3 | Хешування та перевірка паролів |
| dotenv | 16.6.1 | Робота зі змінними середовища |
| npm | 11.12.1 | Керування залежностями та запуск скриптів проєкту |
| Git / GitHub | — | Контроль версій і зберігання коду |
| Postman | — | Тестування HTTP-запитів до серверного API |
| Технологія | Версія | Призначення у проєкті |
| Docker | 27.2.0 | Контейнеризація компонентів веб-додатка для ізоляції середовища виконання |
| Docker Compose | 2.29.2 | Координація, оркестрація та одночасний запуск багатоконтейнерної інфраструктури |
| Nginx | 1.25.4 | Зворотний проксі-сервер для маршрутизації API-запитів та роздачі статики фронтенду |

Примітка. Джерело: сформовано автором на основі структури та залежностей проєкту.

Під час розробки веб-додатка також використовувалися допоміжні інструменти, які забезпечують підтримку процесу програмування та тестування. Для контролю версій застосовувався Git, а для зберігання програмного коду та

роботи з репозиторієм — GitHub. Це дозволяло фіксувати зміни у проєкті, працювати з окремими гілками розробки та зберігати історію змін програмного коду.

Для тестування HTTP-запитів до REST API використовувався Postman. За допомогою цього інструмента виконувалася перевірка коректності роботи серверних маршрутів, передачі JSON-даних та механізмів авторизації. Керування залежностями проєкту та запуск локального середовища виконувалися через пакетний менеджер npm.

Для коректної роботи веб-додатка необхідна наявність середовища Node.js 22 LTS або новішої версії, а також встановленої MongoDB або доступу до хмарного сервісу MongoDB Atlas. Клієнтська частина підтримує роботу у сучасних браузерах, зокрема Google Chrome, Mozilla Firefox, Microsoft Edge та інші сучасні веб-браузери.

Розробка та тестування проєкту можуть виконуватися в операційних системах Windows, Linux або macOS. Для стабільної роботи середовища розробки доцільно використовувати персональний комп’ютер із багатоядерним процесором, не менше 8 ГБ оперативної пам’яті та наявністю стабільного доступу до мережі Інтернет. Додатково необхідна наявність вільного місця на диску для встановлення залежностей Node.js, локальної бази даних та збірки клієнтської частини проєкту.

### 3.2 Програмна реалізація клієнтської та серверної частин веб-додатка

Програмна реалізація веб-додатка Nexora виконана відповідно до клієнт-серверної архітектури із фізичним розділенням клієнтської та серверної частин проєкту. Такий підхід дозволяє організувати незалежну обробку інтерфейсної логіки, серверних запитів та взаємодії з базою даних, а також спрощує подальшу підтримку й розширення функціональних можливостей системи.

Клієнтська частина побудована як SPA-додаток на основі React та Vite. Основна структура компонентів клієнтської частини організована за модульним

принципом і містить окремі каталоги для сторінок, компонентів, React hooks та context-модулів, сервісів, стилів та допоміжних утиліт. Загальна структура клієнтської та серверної частин веб-додатка наведена на рисунку 3.1.

Для забезпечення ізоляції компонентів веб-додатка та спрощення процесу локального розгортання у структуру проєкту було інтегровано засоби контейнеризації Docker. У кореневому каталозі розміщено файл docker-compose.yml, який використовується для оркестрації клієнтської, серверної частин і бази даних. Для клієнтської та серверної частин додано окремі Dockerfile, що забезпечують створення ізольованих середовищ виконання застосунку.

У каталозі клієнтської частини також використовується конфігураційний файл nginx.conf, який забезпечує коректну роботу SPA-маршрутизації React Router та проксування API-запитів до серверної частини веб-додатка.

Для локального налагодження клієнтської частини можуть використовуватися змінні середовища, винесені у файл client/.env, який використовується Vite під час запуску застосунку.

Точкою входу клієнтської частини є файл main.jsx, у якому створюється кореневий компонент React-застосунку, підключаються глобальні стилі та виконується ініціалізація React-застосунку. Центральним компонентом SPA-додатка виступає App.jsx, який об’єднує маршрутизацію, глобальні context-модулі та основні елементи інтерфейсу. Для організації переходів між сторінками застосовується React Router DOM, що забезпечує клієнтську маршрутизацію без повного перезавантаження сторінки.

Маршрути веб-додатка поділено на користувацьку та адміністративну частини. Користувацька частина містить сторінки каталогу, товарів, оформлення замовлення, кошика, порівняння товарів та особистого кабінету. Для адміністративної панелі передбачено окрему структуру маршрутів із перевіркою ролей користувача та доступом до модулів керування товарами, категоріями, замовленнями, відгуками та клієнтами.

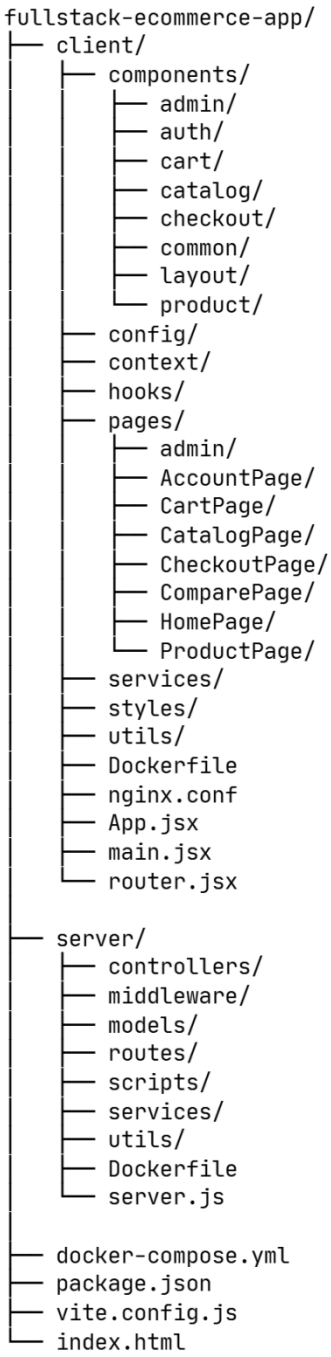


**Рис. 3.1. Структура клієнтської та серверної частин веб-додатка**

Примітка. Джерело: розроблено автором на основі структури проєкту Nexora.

Під час побудови клієнтської логіки застосовано компонентний підхід, відповідно до якого окремі елементи інтерфейсу функціонують як незалежні React-компоненти. Основна частина компонентів згрупована за функціональним

призначенням: каталог товарів, кошик, модулі оформлення замовлення, елементи сторінки товару, адміністративна панель та спільні UI-компоненти. Така організація дозволяє зменшити дублювання коду та спрощує підтримку інтерфейсної частини проєкту.

Для централізованого керування станом клієнтської частини використовується Context API у поєднанні з хуками React. У поточній реалізації застосовуються AuthContext, CompareContext та CartContext, які відповідають за роботу авторизації, порівняння товарів і кошика користувача. Використання Context API дозволяє уникнути надмірного передавання даних через props та забезпечує узгоджену роботу компонентів у різних частинах SPA-додатка.

Важливу роль у структурі компонентів клієнтської частини відіграють custom hooks, за допомогою яких частину логіки роботи зі станом та API-запитами винесено за межі UI-компонентів. Окремі hooks забезпечують завантаження товарів каталогу, фільтрацію, пошук, роботу з відгуками, питаннями, адміністративними модулями та кошиком користувача. Це дозволяє зробити компоненти більш структурованими та спрощує повторне використання логіки у різних частинах системи.

Для взаємодії інтерфейсної частини із сервером у проєкті передбачено окремі service-модулі, розміщені в каталозі services. Через ці модулі організовано HTTP-запити до REST API для роботи з товарами, категоріями, замовленнями, відгуками, питаннями, списками бажань та клієнтською аналітикою. Передача даних між компонентами системи виконується через HTTP-запити у форматі JSON.

Важливим елементом клієнтської логіки є механізм локального збереження даних через localStorage. У локальному сховищі браузера зберігаються JWT-токени авторизації, список товарів для порівняння, стан кошика та окремі користувацькі параметри. Це забезпечує збереження частини стану між оновленнями сторінки та повторними відкриттями браузера.

Для стилізації інтерфейсу застосовується SCSS із модульною організацією

стилів. У проєкті передбачено окремі SCSS-файли для сторінок, адміністративної панелі та повторно використовуваних компонентів. Додатково використовуються partial-файли, які дозволяють розділити великі стилі на окремі логічні модулі. Подібний підхід спрощує підтримку інтерфейсу, забезпечує більш структуровану організацію стилів та дозволяє реалізувати сучасний користувацький інтерфейс каталогу товарів (рис. 3.2).

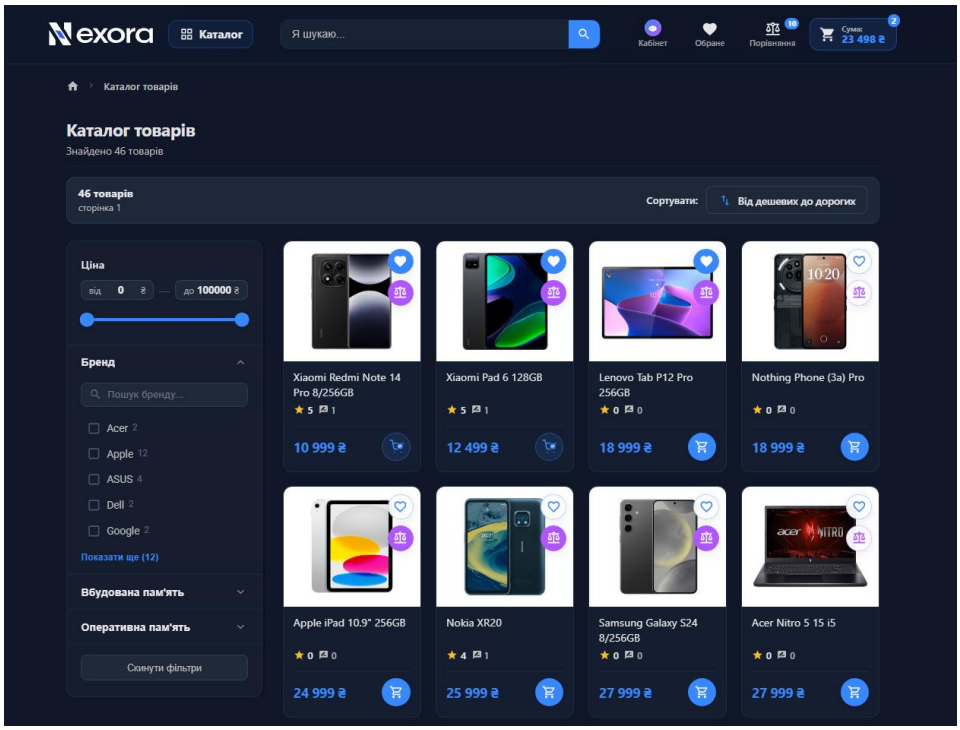


**Рис. 3.2. Приклад користувацького інтерфейсу каталогу товарів**

Примітка. Джерело: розроблено автором на основі інтерфейсу веб-додатка Nexora.

Серверна частина веб-додатка побудована на платформі Node.js із використанням Express.js. Основною точкою входу серверної частини є файл server.js, у якому виконується створення Express-застосунку, підключення middleware, реєстрація REST API-маршрутів та підключення до MongoDB.

Структура серверної логіки містить окремі каталоги для маршрутів, моделей, middleware, controllers, сервісів та допоміжних утиліт. Серверна частина має шар маршрутизації, моделей, middleware та частково виділених controllers. Для базових сутностей, таких як товари, категорії та бренди, логіку

винесено в controller-файли, тоді як частина модулів організована безпосередньо у route-файлах. Подібна структура відповідає поточній реалізації проєкту та дозволяє розділити основні функціональні блоки серверної частини.

Для зберігання даних використано MongoDB у поєднанні з бібліотекою Mongoose. Використання Mongoose дозволяє організувати роботу зі схемами документів MongoDB, виконувати валідацію даних та спростити взаємодію серверної частини з базою даних. У системі передбачено окремі моделі для користувачів, товарів, категорій, брендів, замовлень, відгуків та питань користувачів.

Модель User містить інформацію про користувачів системи, ролі доступу, списки бажань та параметри авторизації. Для моделі Product організовано збереження характеристик товарів, категорій, брендів та окремих атрибутів продукції. Модель Order застосовується для збереження інформації про оформлені замовлення, історію зміни статусів, способи доставки та скасування замовлень.

Взаємодія між компонентами системи виконується через REST API (рис. 3.3).

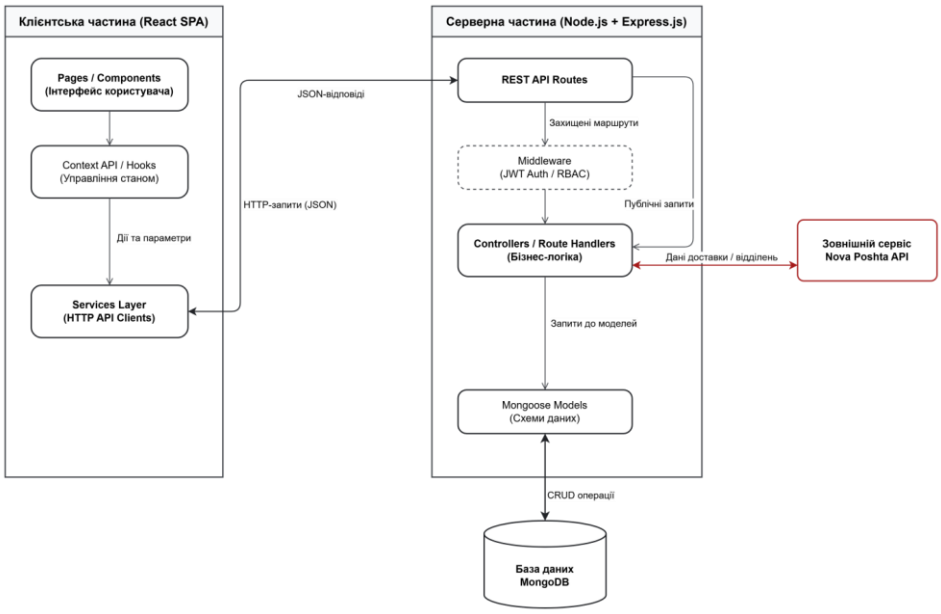


**Рис. 3.3. Схема взаємодії клієнтської та серверної частин веб-додатка**

Примітка. Джерело: розроблено автором.

Після отримання HTTP-запиту серверна логіка виконує перевірку параметрів запиту, обробку даних, взаємодію з моделями Mongoose та формує JSON-відповідь для клієнтської частини. Основні REST API-маршрути організовано для роботи з товарами, категоріями, авторизацією, замовленнями, відгуками, питаннями та списками бажань.

Для захисту облікових записів у системі застосовується JWT-автентифікація. Після успішного входу користувача сервер генерує JWT-токен, який передається клієнтській частині та зберігається у localStorage. Під час виконання захищених HTTP-запитів токен додається до заголовка Authorization . Серверна логіка перевіряє валідність токена через middleware авторизації та визначає права доступу користувача відповідно до його ролі.

У системі підтримується role-based access control для ролей user, admin та moderator. Користувачі з роллю admin мають доступ до повного функціоналу адміністративної панелі, включно з керуванням товарами, категоріями, замовленнями та користувачами. Для ролі moderator передбачено доступ до системи модерації контенту та роботи з клієнтськими зверненнями (рис. 3.4). Фрагмент middleware-функцій автентифікації та перевірки ролей наведено у додатку Б.

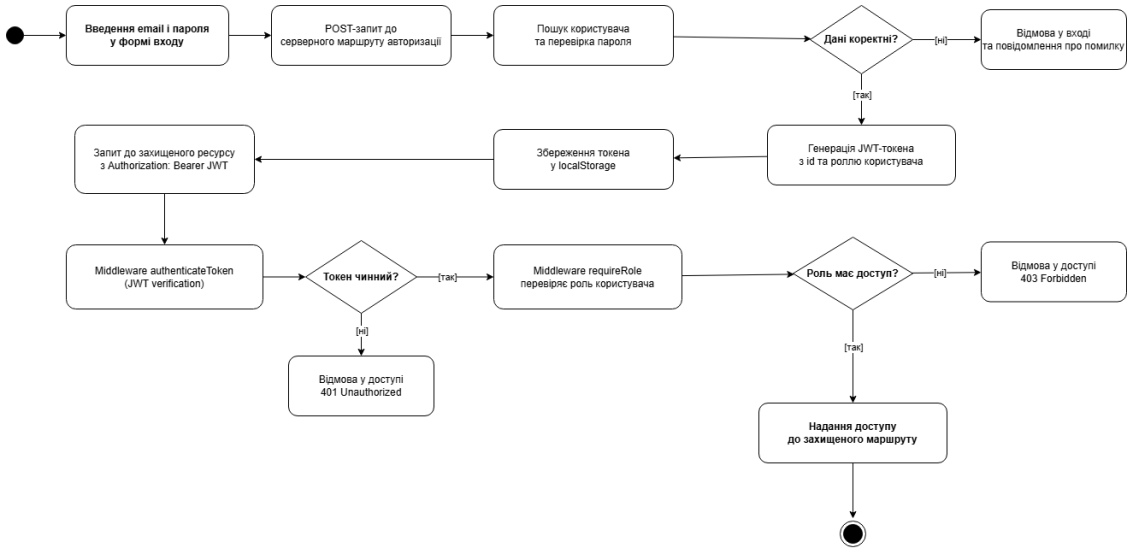


**Рис. 3.4. Схема JWT-авторизації та перевірки доступу користувачів**

Примітка. Джерело: розроблено автором.

Одним із ключових функціональних модулів системи є каталог товарів. На клієнтській стороні забезпечується завантаження товарів, фільтрація, пошук, сортування та пагінація каталогу. Значна частина логіки фільтрації виконується безпосередньо на клієнтській стороні після отримання даних із сервера. Це дозволяє зменшити кількість повторних HTTP-запитів та забезпечує більш швидке оновлення інтерфейсу.

У клієнтській частині також реалізовано підтримку акційних цін товарів. Для цього використовується поле compareAtPrice, яке дозволяє відображати попередню вартість товару у вигляді перекресленої ціни, а також формувати інформацію про знижку у каталозі, картці товару, кошику та checkout-модулі.

Модуль авторизації підтримує реєстрацію користувачів, вхід до системи, вихід із облікового запису та перевірку доступу до захищених маршрутів. Для додаткового захисту паролів застосовується бібліотека bcryptjs, яка забезпечує хешування паролів перед їх збереженням у базі даних.

Модуль оформлення замовлень забезпечує створення замовлення, збереження інформації про товари, контактні дані користувача, параметри доставки та статуси виконання. Для взаємодії із сервісом доставки використовуються окремі компоненти клієнтської та серверної частин, пов’язані з API Нової Пошти. У поточній реалізації підтримується отримання інформації про населені пункти та відділення доставки.

У системі також передбачено підтримку списків бажань користувачів. Для авторизованих користувачів списки бажань синхронізуються із серверною частиною та зберігаються у MongoDB. Механізм порівняння товарів організовано переважно на клієнтській стороні через Context API та localStorage, без створення окремої серверної моделі.

Окремим функціональним напрямом виступає система відгуків і питань користувачів. Користувачі можуть залишати відгуки та питання щодо товарів, тоді як адміністратори та модератори мають можливість виконувати перевірку та зміну статусів користувацького контенту. Для цього передбачено moderation flow, який дозволяє приховувати неперевірені записи до завершення модерації.

У межах адміністративної панелі організовано окремі модулі для роботи з товарами, категоріями, замовленнями, клієнтами, відгуками та статистичними показниками. Доступ до адміністративних маршрутів обмежується middleware перевірки ролей та перевіркою дозволених маршрутів на клієнтській стороні (рис. 3.5).

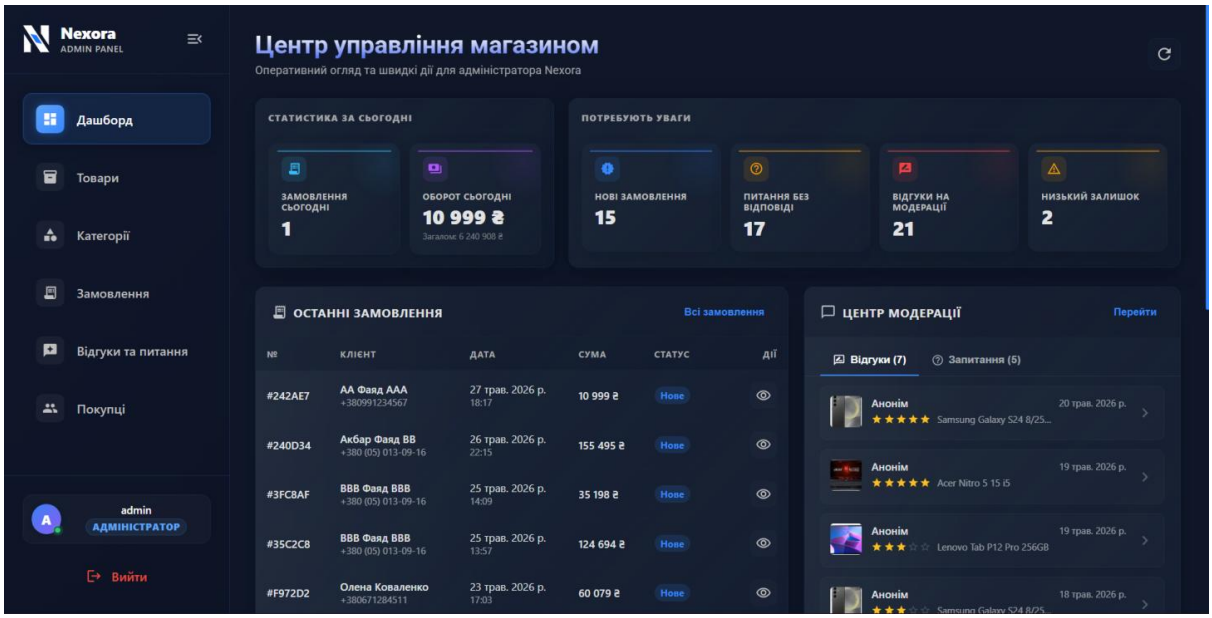


**Рис. 3.5. Приклад адміністративної панелі веб-додатка**

Примітка. Джерело: розроблено автором на основі адміністративної панелі Nexora.

У межах адміністративної панелі також реалізовано механізм керування акційними цінами товарів. Адміністратор може встановлювати значення compareAtPrice для окремих товарів, що дозволяє автоматично відображати попередню вартість продукції та інформацію про знижку у клієнтській частині веб-додатка.

Інтеграція засобів контейнеризації Docker дозволяє спростити локальне розгортання веб-додатка, ізолювати клієнтську та серверну частини системи, а також забезпечити узгоджену конфігурацію середовища виконання для всіх компонентів проєкту. Фрагмент конфігурації docker-compose.yml наведено у додатку В.

Отже, програмна реалізація веб-додатка Nexora базується на поєднанні SPA-архітектури клієнтської частини та REST API серверної логіки. Використання React, Express.js, MongoDB та Mongoose дозволило організувати модульну структуру проєкту, забезпечити взаємодію між компонентами системи та підтримати основні функціональні можливості інтернет-магазину. Архітектурні рішення, застосовані під час реалізації клієнтської та серверної частин, створюють основу для подальшого розвитку функціоналу веб-додатка та його масштабування.

### 3.3 Реалізація адміністративної панелі та розширеного функціоналу взаємодії з користувачем

У процесі розробки веб-додатка Nexora значну увагу було приділено реалізації адміністративної панелі та механізмів взаємодії користувачів із системою. На відміну від базового каталогу товарів, адміністративна частина забезпечує централізоване керування товарами, категоріями, замовленнями, клієнтами та процесами модерації контенту. Поряд із цим у проєкті організовано низку функціональних модулів, спрямованих на покращення взаємодії користувача із веб-додатком та спрощення роботи з каталогом і замовленнями.

Адміністративна панель організована як окрема захищена зона SPA-додатка, доступ до якої здійснюється через маршрути адміністративної частини системи. Для побудови структури адміністративного інтерфейсу використовується окремий компонент AdminLayout, який відповідає за формування навігації, відображення sidebar-панелі, роботу вкладених сторінок та перевірку доступу до окремих модулів.

Для адміністративної частини застосовано рольову модель доступу для ролей admin та moderator. Залежно від ролі користувача система формує доступний набір модулів і маршрутів. Користувачі з роллю admin мають доступ до повного функціоналу адміністративної панелі, включно з керуванням товарами, категоріями, замовленнями та клієнтами. Для ролі moderator відкрито доступ до модулів модерації відгуків і питань, а також роботи з клієнтськими

зверненнями. Подібний підхід дозволяє розмежувати адміністративні функції та зменшити ризик несанкціонованого доступу до окремих компонентів системи.

Важливим елементом адміністративної частини є механізм protected routes, який обмежує доступ до сторінок адміністративної панелі для неавторизованих користувачів. Після проходження авторизації інформація про адміністративну сесію частково зберігається у localStorage, що дозволяє підтримувати стан входу між оновленнями сторінки та повторними відкриттями браузера.

Інтерфейс адміністративної панелі адаптований для роботи на різних типах пристроїв. Sidebar-навігація підтримує responsive-режим і можливість згортання меню, що спрощує використання системи на екранах із меншою шириною. У layout-компоненті також організовано блок профілю адміністратора та механізм виходу із системи.

Центральним елементом адміністративної частини виступає інформаційна панель із віджетами статистики та моніторингу активності системи. У dashboard-компонентах відображаються дані про нові замовлення, оборот, відгуки на модерації, питання без відповіді та товари з низьким залишком. Частина віджетів орієнтована на адміністратора, тоді як для moderator використовується спрощений варіант інформаційної панелі без доступу до модулів керування замовленнями.

Окремий функціональний блок адміністративної частини пов’язаний із керуванням товарами та категоріями каталогу. У системі підтримуються CRUD-операції для створення, редагування та видалення товарів і категорій. Робота з товарами організована через таблицю із підтримкою пошуку, сортування, фільтрації та пагінації, що дозволяє ефективно працювати навіть із великою кількістю позицій каталогу.

Під час створення або редагування товару використовується форма з вкладками, у якій окремо згруповано базову інформацію, опис товару та характеристики. Це дозволяє зменшити перевантаження інтерфейсу та зробити процес редагування більш структурованим. Для окремих категорій підтримується групування характеристик товарів, що дає змогу організовувати

характеристики товарів у логічні групи відповідно до типу продукції.

Механізм defaultAttributes для категорій забезпечує автоматичне формування базового набору характеристик під час створення товарів певної категорії та спрощує процес заповнення даних адміністратором. Для категорій також підтримується збереження іконок, кольорів та зображень, які використовуються для оформлення каталогу на клієнтській стороні.

Для товарів організовано механізм керування залишками на складі. Адміністратор може оперативно оновлювати кількість товарів без відкриття повної форми редагування. Підтримується також фільтрація товарів із низьким залишком, що спрощує контроль доступності продукції та формування статистики для dashboard-віджетів.

У модулі категорій додатково відображаються статистичні блоки з підрахунком кількості товарів у кожній категорії. Це дозволяє швидко оцінювати структуру каталогу та виявляти категорії з недостатнім наповненням (рис. 3.6).

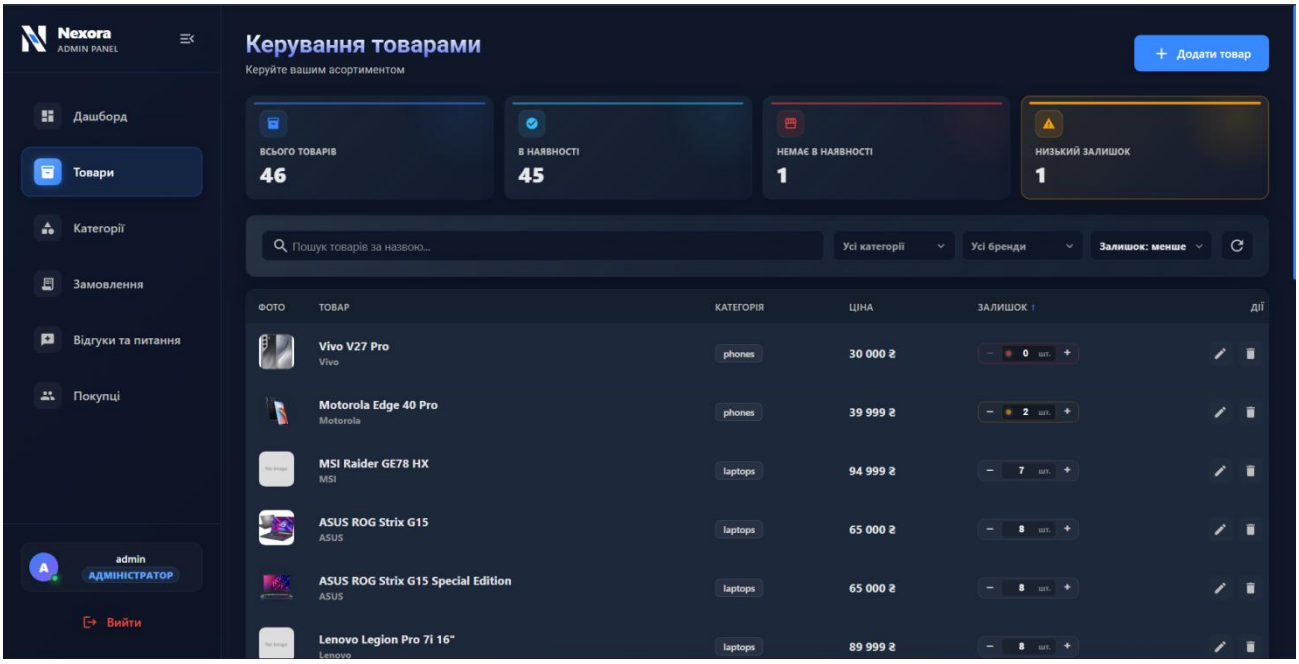


**Рис. 3.6. Інтерфейс керування товарами веб-додатка**

Примітка. Джерело: розроблено автором на основі адміністративної панелі Nexora.

Важливим функціональним напрямом системи є керування замовленнями

та робота з клієнтськими даними. В адміністративній панелі організовано окремий модуль для перегляду списку замовлень із підтримкою пошуку, сортування, фільтрації за статусами та пагінації. Для кожного замовлення доступний перегляд детальної інформації, включно зі списком товарів, контактними даними користувача, способом доставки, способом оплати та коментарями до замовлення.

У системі організовано механізм зміни статусів замовлення відповідно до визначеного життєвого циклу. Основними статусами є new, confirmed, packing, ready_for_pickup, received та cancelled. Для окремих статусів передбачено допустимі переходи, що дозволяє підтримувати послідовність обробки замовлення та уникати некоректних змін стану.

Значну увагу приділено механізму скасування замовлень. Система підтримує можливість скасування як з боку користувача, так і з боку адміністратора. Для кожного випадку зберігаються причина скасування, дата виконання операції та інформація про користувача, який ініціював зміну статусу. У моделі замовлення також ведеться історія зміни статусів, що дозволяє відстежувати послідовність обробки замовлення та формувати часову шкалу виконання операцій. Подібна організація статусів дозволяє підтримувати послідовність обробки замовлення та спрощує контроль поточного стану покупки (рис. 3.7).

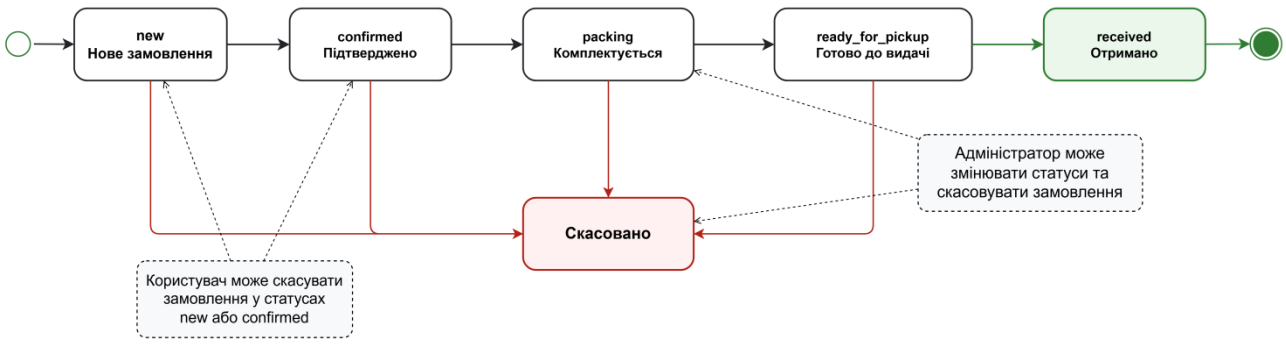


**Рис. 3.7. Схема життєвого циклу замовлення**

Примітка. Джерело: розроблено автором на основі реалізованої логіки обробки замовлень у веб-додатку Nexora.

Модуль роботи з клієнтами дозволяє адміністраторам і модераторам переглядати список користувачів, виконувати пошук, сортування та фільтрацію клієнтів за статусами. Для окремих користувачів передбачено перегляд статистики замовлень, останніх покупок та активності у системі.

Підтримується також механізм блокування та розблокування користувачів. Перед зміною статусу відображається вікно підтвердження для підтвердження дії, що зменшує ризик випадкової зміни стану облікового запису. У статистичних блоках адміністративної панелі додатково відображаються дані про нових клієнтів, активних покупців та постійних користувачів системи.

Окремим функціональним напрямом веб-додатка є система модерації відгуків і питань користувачів. Для цього в адміністративній панелі організовано центр модерації із підтримкою фільтрації, пошуку, сортування та зміни статусів користувацького контенту.

Передбачено три основні статуси модерації: pending, approved та rejected. Після створення нового відгуку або питання запис автоматично отримує статус pending і не відображається у відкритій частині веб-додатка до завершення перевірки. Адміністратори та модератори можуть схвалювати або відхиляти записи, а також повертати контент на повторну модерацію.

Для відгуків використовується додаткова фільтрація за рейтингом товару, тоді як для питань передбачено фільтр за наявністю відповіді. У модулі питань реалізовано механізм відповіді користувачу через окремий висувний компонент (drawer). Після надсилання відповіді статус питання автоматично змінюється на approved, що дозволяє публікувати перевірений контент без додаткових дій адміністратора.

Модераційний центр інтегровано з dashboard-віджетами адміністративної панелі. На інформаційній панелі відображається кількість відгуків на модерації та питань без відповіді, що дозволяє оперативно реагувати на новий користувацький контент та спрощує контроль процесів перевірки публікацій (рис. 3.8).

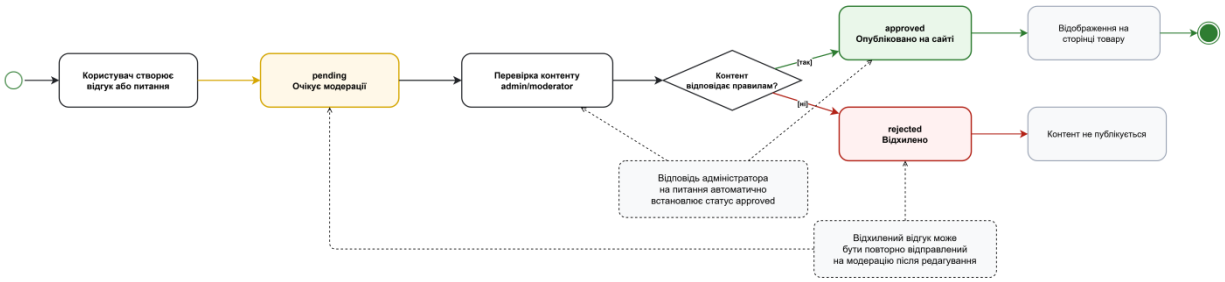


**Рис. 3.8. Схема процесу модерації відгуків і питань**

Примітка. Джерело: розроблено автором на основі реалізованої логіки модерації контенту у веб-додатку Nexora.

Поряд із адміністративною частиною у проєкті реалізовано розширений функціонал взаємодії користувача із системою. Одним із таких модулів є wishlist multi-lists, який дозволяє користувачам створювати декілька окремих списків бажань, перейменовувати їх, переміщувати товари між списками та очищувати окремі добірки. Збереження списків синхронізується із серверною частиною та пов’язується з обліковим записом користувача.

Для порівняння товарів використовується окремий модуль порівняння товарів, що працює переважно на клієнтській стороні. Користувач може додавати товари до списку порівняння, переглядати характеристики продукції та активувати режим відображення лише відмінностей між товарами. Дані модуля частково зберігаються у localStorage, що дозволяє зберігати список порівняння між сесіями браузера.

Навігацію між товарами доповнює механізм recently viewed, який автоматично формує список нещодавно переглянутих позицій каталогу. Завдяки цьому користувач може швидко повертатися до раніше відкритих товарів без повторного пошуку.

В особистому кабінеті організовано модуль історії замовлень користувача. Підтримується перегляд статусів замовлень, часової шкали змін, деталей доставки та можливість скасування окремих замовлень. Окрім цього, користувач може створювати відгуки на товари безпосередньо зі сторінки історії покупок.

Для сторінок товарів передбачено модулі відгуків і питань користувачів. Після створення записи автоматично передаються на модерацію та отримують статус pending. Такий підхід дозволяє зменшити кількість небажаного контенту

та забезпечити додатковий контроль публікацій.

Для покращення навігації у веб-додатку використовуються breadcrumbs, які відображають поточне положення користувача у структурі каталогу. Додатково в інтерфейсі реалізовано висувний блок кошика — cart drawer, який автоматично відкривається після додавання товару та дозволяє швидко перейти до кошика або оформлення замовлення.

Взаємодія користувача із системою супроводжується інтерфейсними повідомленнями та toast-повідомленнями, які відображають результати окремих дій, зокрема під час додавання товарів до порівняння, списків бажань або оформлення замовлення. Це робить роботу з інтерфейсом більш зрозумілою та забезпечує додатковий зворотний зв’язок для користувача.

Окремим напрямом функціоналу веб-додатка є реалізація процесу оформлення замовлення та інтеграція із сервісом доставки Нової Пошти. На сторінці оформлення замовлення користувач може обирати спосіб доставки, заповнювати контактні дані, переглядати склад замовлення та підтверджувати створення покупки (рис. 3.9).

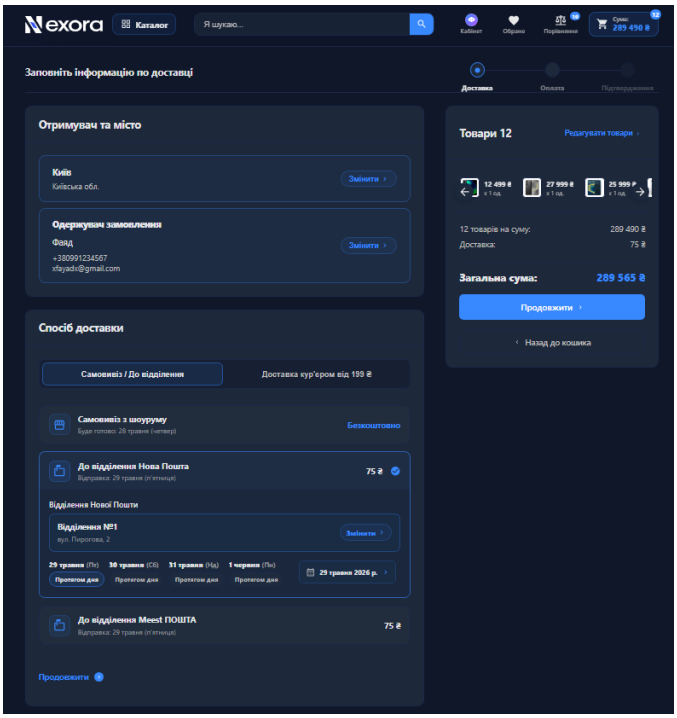


**Рис. 3.9. Приклад сторінки оформлення замовлення веб-додатка**

Примітка. Джерело: розроблено автором на основі checkout-модуля веб-додатка Nexora.

Підтримується декілька варіантів доставки, серед яких самовивіз, доставка до відділення Нової Пошти, доставка до відділення Meest, кур’єрська доставка компанії та кур’єрська доставка Нової Пошти. Для взаємодії із сервісом Нової Пошти використовуються окремі клієнтські та серверні модулі, які забезпечують завантаження списку міст і відділень через API сервісу доставки.

Під час введення користувачем населеного пункту система виконує пошук відповідних міст через backend-маршрут, після чого користувач може обрати потрібне відділення доставки. Дані про місто та відділення використовуються під час формування структури замовлення та передаються на сервер разом із інформацією про товари, контактні дані користувача, спосіб доставки та метод оплати.

Після успішного створення замовлення система очищує кошик користувача, частково оновлює дані профілю та перенаправляє користувача до історії замовлень. Така організація checkout-процесу забезпечує більш цілісну взаємодію між компонентами системи та дозволяє інтегрувати механізми доставки без ускладнення клієнтської логіки.

У результаті реалізації адміністративної панелі та розширених модулів взаємодії користувача у веб-додатку Nexora забезпечено підтримку керування основними бізнес-процесами інтернет-магазину, модерації контенту, роботи із замовленнями та взаємодії користувача з каталогом продукції. Реалізовані функціональні модулі дозволяють підтримувати актуальний стан системи, покращують зручність використання веб-додатка та створюють основу для подальшого розвитку проєкту.

### 3.4 Тестування та верифікація роботи системи

Після завершення основних етапів реалізації веб-додатка Nexora було проведено тестування та верифікацію роботи ключових функціональних модулів системи. Основна увага приділялася перевірці коректності взаємодії між клієнтською та серверною частинами, стабільності виконання основних користувацьких сценаріїв, а також перевірці механізмів авторизації, обмеження

доступу та обробки даних.

Оскільки проєкт реалізований у форматі SPA-додатка, тестування значної частини функціоналу виконувалося безпосередньо у браузері шляхом перевірки поведінки інтерфейсу та взаємодії React-компонентів із серверною логікою. Значна увага приділялася перевірці основних SPA-сценаріїв у браузері, коректності оновлення стану компонентів, роботі форм введення, передачі даних через REST API та динамічному оновленню інтерфейсу без перезавантаження сторінки.

Функціональна перевірка охоплювала як користувацькі сценарії, так і сценарії роботи адміністративної панелі. Для користувацької частини аналізувалася робота реєстрації та авторизації користувачів, каталогу товарів, сторінок товарів, кошика, оформлення замовлення, історії замовлень, списків бажань, порівняння товарів та модулів відгуків і питань.

Під час тестування каталогу товарів аналізувалася коректність пошуку, фільтрації, сортування та пагінації. Особлива увага приділялася динамічному оновленню інтерфейсу після зміни параметрів пошуку та застосування фільтрів. Також перевірялася коректність відображення повідомлень про відсутність результатів пошуку та робота станів завантаження та помилок під час отримання даних із сервера.

Для сторінок товарів виконувалася перевірка відображення характеристик продукції, згрупованих атрибутів товару, опису товару, списку відгуків і питань користувачів. Окремо аналізувалася робота механізму додавання товарів до кошика, списків бажань та модуля порівняння.

Тестування модуля кошика включало перевірку додавання товарів, зміни кількості позицій, видалення товарів та оновлення загальної вартості замовлення. Досліджувалося також коректне оновлення стану кошика без перезавантаження сторінки та робота висувної панелі кошика після додавання товару.

Окремо аналізувалася коректність відображення акційних цін товарів у каталозі, картці товару, кошику та checkout-модулі. Під час тестування

перевірялося відображення перекресленої попередньої вартості товару, формування інформації про знижку та коректність оновлення підсумкової вартості замовлення після додавання акційних товарів до кошика.

Значну увагу приділено процесу оформлення замовлення. Аналізувалася коректність заповнення форми оформлення замовлення, робота механізмів вибору доставки, завантаження списку міст і відділень Нової Пошти, а також формування даних замовлення перед передачею на сервер. Для невалідних даних перевірялося відображення повідомлень про помилки та блокування створення замовлення.

Під час перевірки особистого кабінету користувача тестувалися перегляд історії замовлень, можливість скасування замовлень у дозволених статусах new або confirmed, створення відгуків, робота списків бажань та модулів взаємодії з каталогом. Також аналізувалося збереження токенів і стану авторизації у localStorage та синхронізація користувацьких даних із серверною частиною системи.

Для адміністративної панелі виконувалося тестування CRUD-операцій із товарами та категоріями, зміни статусів замовлень, фільтрації списків, пагінації, модерації контенту та роботи dashboard-компонентів. Окремо аналізувалася коректність рольової маршрутизації та обмеження доступу для ролей admin і moderator.

Результати функціонального тестування основних модулів веб-додатка наведено в таблиці 3.2.

Окремим напрямом тестування стала перевірка REST API та механізмів обмеження доступу. Для цього використовувався інструмент Postman, за допомогою якого аналізувалися основні HTTP-запити до серверної частини, коректність відповідей API та робота JWT-автентифікації.

Під час перевірки авторизації оцінювалася коректність створення JWT-токенів після входу користувача до системи, а також робота заголовка Authorization із Bearer-токеном під час виконання захищених запитів. Додатково аналізувалася робота middleware-функції authenticateToken, яка відповідає за

перевірку валідності токена та ідентифікацію користувача.

Таблиця 3.2

**Функціональне тестування основних модулів**

| Модуль | Тестовий сценарій | Очікуваний результат | Фактичний результат |
|---|---|---|---|
| Авторизація | Вхід користувача до системи | Успішна авторизація та отримання JWT | Працює коректно |
| Каталог товарів | Пошук і фільтрація товарів | Оновлення каталогу відповідно до параметрів | Працює коректно |
| Каталог товарів | Відображення акційних цін | Коректне відображення compareAtPrice та знижки | Працює коректно |
| Кошик | Додавання та видалення товарів | Оновлення стану кошика без перезавантаження сторінки | Працює коректно |
| Оформлення замовлення | Створення замовлення | Формування та збереження замовлення | Працює коректно |
| Списки бажань | Додавання товарів до списків бажань | Синхронізація списків із сервером | Працює коректно |
| Модерація | Схвалення відгуків і питань | Зміна статусів контенту | Працює коректно |
| Адміністративна панель | Редагування товарів і категорій | Оновлення даних у системі | Працює коректно |

Примітка. Джерело: розроблено автором на основі результатів функціонального тестування веб-додатка Nexora.

Для перевірки рольової моделі доступу використовувалася middleware-функція requireRole, яка обмежує доступ до окремих маршрутів залежно від ролі користувача. У процесі тестування аналізувалися сценарії доступу користувачів із ролями admin, moderator та user до захищених маршрутів адміністративної частини.

Особливу увагу приділено перевірці реакції системи на невалідні або відсутні JWT-токени. У разі відсутності токена серверна частина повертала HTTP-відповідь зі статусом 401, тоді як для користувачів із недостатніми правами доступу поверталася відповідь 403. Це дозволило підтвердити коректність реалізації механізмів захисту серверних маршрутів.

Також виконувалася перевірка API-запитів, пов’язаних зі зміною статусів

замовлень, модерацією контенту, створенням товарів та завантаженням інформації про міста й відділення доставки (рис. 3.10).

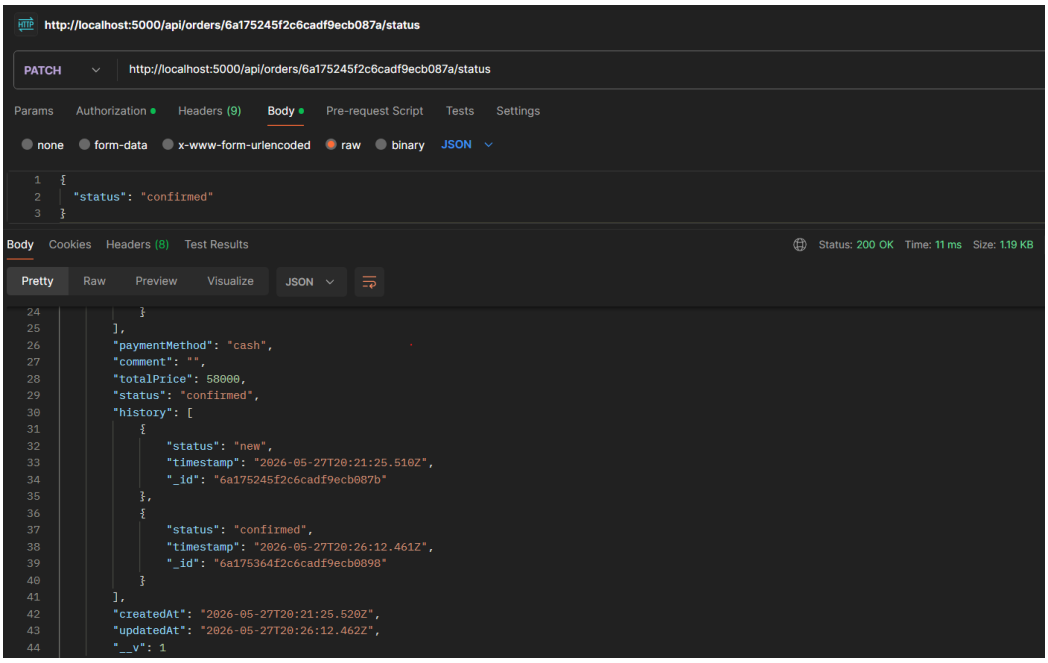


**Рис. 3.10. Приклад тестування REST API для зміни статусу замовлення у Postman**

Примітка. Джерело: розроблено автором.

Результати перевірки REST API наведено в таблиці 3.3.

Під час тестування окремо аналізувалася коректність валідації даних та поведінка інтерфейсу у випадку помилок введення або некоректних дій користувача. Для клієнтської частини перевірялися механізми клієнтської валідації у формах авторизації, реєстрації, оформлення замовлення, створення товарів і категорій.

У формах авторизації та реєстрації оцінювалася коректність введення електронної пошти, паролів та обов’язкових полів. Для checkout-форми тестувалася валідація контактних даних, міста доставки, номера телефону та вибору відділення доставки залежно від обраного способу отримання замовлення.

Таблиця 3.3

**Перевірка REST API**

| Endpoint | Метод | Умови перевірки | Очікуваний результат | Фактичний результат |
|---|---|---|---|---|
| /api/auth/admin/login | POST | Валідні дані адміністратора | Повернення JWT-токена | Працює коректно |
| /api/orders/:id/status | PATCH | JWT-токен адміністратора та валідний статус | Оновлення статусу замовлення | Працює коректно |
| /api/reviews/:id/status | PATCH | JWT-токен модератора або адміністратора | Зміна статусу відгуку | Працює коректно |
| /api/delivery/cities | GET | Передача параметрів пошуку | Отримання списку міст | Працює коректно |
| /api/customers | GET | JWT-токен користувача без прав admin/moderator | Повернення 403 | Працює коректно |

Примітка. Джерело: розроблено автором на основі результатів перевірки REST API у середовищі Postman.

У модулях створення товарів і категорій аналізувалися обов’язкові поля, значення кількості товарів, коректність згрупованих атрибутів товару та структура defaultAttributes. Для невалідних значень система відображала повідомлення про помилки та блокувала збереження даних.

На серверній стороні виконувалася перевірка валідації на рівні Mongoose-схем, перевірок на рівні маршрутів і контролерів та middleware-функцій авторизації. Аналізувалася коректність створення замовлень, допустимість зміни статусів, валідація модераційних статусів pending, approved та rejected, а також перевірка параметрів для API доставки.

Додатково оцінювалася адаптивність інтерфейсу та адаптивна поведінка веб-додатка. Перевірялася робота sidebar-навігації адміністративної панелі, mobile drawer-компонентів, фільтрів каталогу та елементів навігації на різних розмірах екрана. Також аналізувалися стани завантаження, повідомлення про помилки та коректність динамічного оновлення стану інтерфейсу без

перезавантаження сторінки.

Під час перевірки механізмів обмеження доступу тестувалися права користувачів для різних ролей системи. Результати перевірки доступу користувачів відповідно до рольової моделі системи наведено в таблиці 3.4.

Таблиця 3.4

**Перевірка доступу за ролями користувачів**

| Роль | Доступні модулі | Обмеження | Результат перевірки |
|---|---|---|---|
| user | Каталог, checkout, orders, wishlist | Відсутній доступ до адміністративних маршрутів | Працює коректно |
| moderator | Центр модерації, клієнти, інформаційна панель | Відсутній доступ до керування товарами та замовленнями | Працює коректно |
| admin | Повна адміністративна панель | Має повний доступ до адміністративних модулів | Працює коректно |

Примітка. Джерело: розроблено автором на основі результатів перевірки рольової моделі доступу веб-додатка Nexora.

У результаті проведеного тестування та верифікації встановлено, що основні функціональні сценарії веб-додатка Nexora працюють коректно та забезпечують стабільну взаємодію між клієнтською та серверною частинами системи. Перевірка REST API підтвердила коректність обробки HTTP-запитів, механізмів JWT-автентифікації та рольової моделі доступу.

У межах виконаної функціональної перевірки критичних помилок не виявлено. Механізми валідації даних, модерації контенту, зміни статусів замовлень та роботи адміністративної панелі функціонують відповідно до очікуваної логіки. Проведена верифікація підтвердила працездатність реалізованих модулів і коректність взаємодії основних компонентів веб-додатка.

## Висновки до розділу 3

У висновках до третього розділу було підсумовано результати програмної реалізації веб-додатка Nexora, побудованого на основі клієнт-серверної архітектури із використанням стеку MERN. У процесі розробки організовано

взаємодію між клієнтською SPA-частиною на основі React та серверною логікою, реалізованою за допомогою Express.js і Node.js. Для зберігання даних використано MongoDB у поєднанні з Mongoose, що дозволило організувати роботу зі схемами даних, підтримати гнучку структуру товарів і забезпечити взаємодію серверної частини з базою даних через REST API.

У межах реалізації клієнтської частини було побудовано SPA-інтерфейс із динамічним оновленням стану без перезавантаження сторінок. Для керування станом компонентів використано Context API та React hooks, а модульна структура компонентів, сервісів і стилів дозволила впорядкувати логіку роботи окремих функціональних блоків. Серверна частина забезпечує обробку HTTP-запитів, роботу механізмів авторизації, перевірку прав доступу та взаємодію з моделями даних.

Під час реалізації веб-додатка було впроваджено ключові функціональні модулі інтернет-магазину, зокрема каталог товарів, механізми оформлення замовлень, особистий кабінет користувача, списки бажань, модуль порівняння товарів, систему відгуків і питань, а також інтеграцію доставки через API Нової Пошти. Окрему увагу приділено адміністративній панелі, яка забезпечує керування товарами, категоріями, замовленнями, клієнтами та процесами модерації контенту. Реалізована рольова модель доступу дозволяє розмежовувати функції користувачів із ролями user, moderator та admin.

У процесі розробки також було організовано механізми JWT-автентифікації та захисту серверних маршрутів. Для підтримки взаємодії між компонентами системи застосовано REST API, а частина користувацьких даних та стану інтерфейсу зберігається у localStorage, що дозволяє підтримувати авторизацію та окремі елементи інтерфейсної логіки між сесіями браузера.

Значна увага приділялася перевірці працездатності системи. У межах тестування виконано аналіз основних користувацьких та адміністративних сценаріїв, перевірку роботи REST API, захищених маршрутів, механізмів авторизації та валідації даних. У межах проведеного тестування підтверджено коректність взаємодії між компонентами системи, стабільність роботи основних

модулів та відповідність реалізованого функціоналу поставленим вимогам.

Отже, у третьому розділі реалізовано основні компоненти веб-додатка Nexora, організовано взаємодію між клієнтською та серверною частинами, впроваджено адміністративні та користувацькі модулі, а також проведено тестування функціональних можливостей системи. Запропоновані архітектурні та програмні рішення забезпечують підтримку основних бізнес-процесів інтернет-магазину та створюють основу для подальшого розширення функціональних можливостей веб-додатка.

# ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було досліджено особливості функціонування систем електронної комерції та сучасні підходи до розробки SPA-веб-додатків. Проведений аналіз предметної області підтвердив актуальність використання клієнт-серверної архітектури, REST API та JavaScript-технологій під час створення веб-додатків інтернет-магазинів. Значну увагу приділено питанням організації інтерактивної взаємодії користувача із системою, побудови адміністративних модулів, підтримки персоналізованого функціоналу та інтеграції зовнішніх сервісів у межах єдиного програмного середовища.

Аналіз сучасних веб-додатків електронної комерції та існуючих платформ-аналогів дозволив визначити доцільність використання SPA-архітектури для побудови інтерфейсу інтернет-магазину. Використання підходу Single Page Application спрощує взаємодію користувача із системою за рахунок динамічного оновлення інтерфейсу без повторного перезавантаження сторінок. Проведене дослідження також підтвердило ефективність використання REST API для організації взаємодії між клієнтською та серверною частинами системи, а застосування MongoDB як документо-орієнтованої бази даних є доцільним для роботи з товарами, які можуть містити різні набори характеристик залежно від категорії продукції.

У межах роботи було виконано проєктування архітектури веб-додатка Nexora на основі стеку MERN. Побудовано клієнт-серверну модель взаємодії між React SPA та серверною частиною на Node.js і Express.js. Спроєктовано структуру документо-орієнтованої бази даних MongoDB, визначено основні сутності системи та логічні зв’язки між ними. Для опису взаємодії компонентів веб-додатка та основних бізнес-процесів використовувалися елементи UML- та BPMN-моделювання. У роботі також сформовано структуру REST API та організовано механізми обміну даними між компонентами системи через HTTP-запити у форматі JSON.

Під час проєктування значну увагу приділено організації модульної

структури клієнтської та серверної частин системи. Клієнтська частина побудована з використанням окремих сторінок, компонентів, React hooks, context-модулів і сервісів, що спрощує підтримку та подальше розширення функціоналу веб-додатка. Серверна частина містить маршрути, middleware-функції, моделі даних та окремі controllers для основних сутностей системи. Така організація дозволила розділити логіку обробки HTTP-запитів, взаємодію з базою даних та механізми авторизації користувачів.

Важливим компонентом під час проєктування стала база даних MongoDB, структура якої була адаптована до роботи з різними категоріями товарів та динамічними наборами характеристик. Використання документної моделі зберігання даних спростило організацію  згруповані характеристики, category стандартні атрибути категорій та вкладених структур даних для замовлень, відгуків і списків бажань. Застосування UML- та BPMN-схем дало можливість формалізувати основні процеси системи, описати сценарії взаємодії користувача із веб-додатком та структуру взаємодії між окремими компонентами архітектури.

Під час програмної реалізації веб-додатка розроблено клієнтську частину із використанням React, Context API, React Router DOM та модульної SCSS-структури. Серверну логіку побудовано на основі Node.js та Express.js із використанням MongoDB і Mongoose для роботи з базою даних. У межах системи організовано JWT-автентифікацію користувачів та рольову модель доступу для ролей user, moderator та admin. У межах адміністративної частини реалізовано механізми керування товарами, категоріями, замовленнями, клієнтами та модерації користувацького контенту.

У веб-додатку реалізовано основні функціональні модулі інтернет-магазину, необхідні для взаємодії користувача із системою. Користувацька частина підтримує роботу каталогу товарів, сторінок товарів, кошика, процесу оформлення замовлення, особистого кабінету користувача та історії замовлень. Додатково впроваджено механізми wishlist multi-lists, модуль порівняння товарів, блок recently viewed, систему відгуків і питань користувачів, а також

інтерактивні елементи інтерфейсу для динамічної роботи з каталогом продукції.

В адміністративній частині веб-додатка організовано модулі керування товарами, категоріями, брендами, замовленнями та клієнтами. Передбачено систему модерації відгуків і питань користувачів, що дозволяє контролювати публікацію контенту та підтримувати коректну роботу механізмів взаємодії між користувачами і системою. Для розмежування доступу до окремих компонентів веб-додатка використано role-based access control, що забезпечує окремі рівні доступу для користувачів із ролями user, moderator та admin.

У роботі також реалізовано інтеграцію із сервісами доставки Нової Пошти, що включає отримання списків міст і відділень через API служби доставки та формування параметрів доставки під час оформлення замовлення. Для локального розгортання та ізоляції середовища розробки використовувалися засоби контейнеризації Docker, що спрощує запуск веб-додатка та підтримку стабільного середовища виконання.

Під час тестування веб-додатка виконано перевірку основних функціональних сценаріїв користувацької та адміністративної частин системи. Проведено тестування REST API, механізмів JWT-автентифікації, захищених маршрутів та рольової моделі доступу. Окремо перевірялася коректність взаємодії між компонентами системи, робота модулів оформлення замовлень, модерації контенту, пошуку та фільтрації товарів, а також стабільність оновлення інтерфейсу без перезавантаження сторінок.

У межах виконаної функціональної перевірки також аналізувалася коректність роботи SPA-сценаріїв, механізмів валідації даних, обробки помилок та динамічного оновлення стану інтерфейсу. Перевірку REST API виконано за допомогою Postman із використанням JWT-токенів та тестуванням захищених маршрутів для різних ролей користувачів. Результати функціональної перевірки підтвердили коректність взаємодії між клієнтською та серверною частинами системи, стабільність роботи основних функціональних модулів та відповідність реалізованого функціоналу поставленим вимогам.

У результаті виконання кваліфікаційної роботи досягнуто поставленої

мети та виконано всі визначені завдання. Створений веб-додаток Nexora підтримує основні бізнес-процеси інтернет-магазину та поєднує користувацькі й адміністративні модулі у межах єдиної клієнт-серверної системи. Запропоновані архітектурні та програмні рішення можуть бути використані як основа для подальшого розвитку систем електронної комерції та модернізації окремих компонентів веб-додатків подібного типу. Архітектура системи допускає подальше розширення функціональних можливостей, зокрема інтеграцію платіжного шлюзу, розширення аналітичних можливостей адміністративної панелі та оптимізацію серверної фільтрації у разі збільшення обсягу каталогу товарів.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Глобальна електронна комерція: ключові цифри та тренди e-commerce 2024. URL: https://rau.ua/novyni/trendi-e-com-2024/ (дата звернення: 10.04.2026).

2. Український e-commerce 2025: дослідження, тренди та прогнози. URL: https://www.promodo.ua/ukrayinskiy-ecommerce-2025-1 (дата звернення: 10.04.2026).

3. eCommerce Statistics (2025): Sales & User Growth Trends / Capital One Shopping Research. 2025. URL: https://capitaloneshopping.com/research/ecommerce-statistics/ (дата звернення: 10.04.2026).

4. Баран С. В. Основи web-програмування: навч. посібник. Кривий Ріг : Державний університет економіки і технологій, 2023. 316 с.

5. Global Ecommerce Forecast 2024 / eMarketer Insider Intelligence. URL: https://capitaloneshopping.com/research/ecommerce-statistics/ (дата звернення: 10.04.2026).

6. Каплун В. А., Ціхоцький М. С., Лукічов В. В. Основи web-програмування. Теорія і практика: електрон. навч. посібник. Вінниця : ВНТУ, 2023. 128 с.

7. Хроленко В. М. Web-програмування: навч. матеріал. Київ : КНУБА, 2024.

8. React Documentation. URL: https://react.dev/ (дата звернення: 10.04.2026).

9. Material UI Documentation. URL: https://mui.com/ (дата звернення: 10.04.2026).

10. Node.js Documentation. URL: https://nodejs.org/en/docs/ (дата звернення: 10.04.2026).

11. MongoDB Documentation. URL: https://www.mongodb.com/docs/ (дата звернення: 10.04.2026).

12. JSON Web Token Introduction. URL: https://jwt.io/introduction (дата звернення: 10.04.2026).

13. OWASP Top 10:2021 / The Open Web Application Security Project. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 10.04.2026).

14. Amazon Web Services (AWS) Architecture / Amazon. URL: https://aws.amazon.com/architecture/ (дата звернення: 10.04.2026).

15. Офіційний сайт інтернет-магазину Rozetka. URL: https://rozetka.com.ua/ (дата звернення: 10.04.2026).

16. WooCommerce 10.5 Release Notes / WooCommerce Documentation. URL: https://developer.woocommerce.com/ (дата звернення: 10.04.2026).

17. Vue.js Documentation. URL: https://vuejs.org/ (дата звернення: 10.04.2026).

18. Bootstrap Documentation. URL: https://getbootstrap.com/ (дата звернення: 10.04.2026).

19. Google Material Design Guidelines. URL: https://m3.material.io/ (дата звернення: 10.04.2026).

20. Авраменко В. С., Авраменко А. С. Проектування інформаційних систем : навч. посібник. Черкаси : ЧНУ ім. Б. Хмельницького, 2017. 434 с. URL: https://files.znu.edu.ua/files/Bibliobooks/Inshi72/0053479.pdf (дата звернення: 07.05.2026).

21. Єлфімов Д., Воротнікова З. Приклади проектування документоорієнтованої бази даних в MongoDB. Вісник Приазовського Державного Технічного Університету. Серія: Технічні науки. 2025. № 50. С. 50–60. URL: https://doi.org/10.31498/2225-6733.50.2025.336268 (дата звернення: 07.05.2026).

22. Журик І. В. Аналіз ефективності впровадження MERN стеку у розробці веб-застосунків : магістерська робота / Тернопільський національний технічний університет імені Івана Пулюя. Тернопіль, 2025. 81 с. URL: http://elartu.tntu.edu.ua/handle/lib/48657 (дата звернення: 07.05.2026).

23. Крилов Є. В. Технології проєктування NoSQL баз даних : навч. посіб. Київ : НТУУ «КПІ», 2023. 142 с. URL:

https://ela.kpi.ua/server/api/core/bitstreams/b89d8289-4c26-42f8-8fe8-ae0d72d4eb2c/content (дата звернення: 07.05.2026).

24. Щербаков Є. В., Щербакова М. Є. Дослідження ефективності використання бібліотеки React. Наукові вісті Далівського університету. 2022. № 23. URL: https://doi.org/10.33216/2222-3428-2022-23-5 (дата звернення: 07.05.2026).

25. Chalmers E. UML Class Diagram Feedback Tool : 4th Year Project Report. Edinburgh : University of Edinburgh, 2024. 65 p. URL: https://project-archive.inf.ed.ac.uk/ug4/20244089/ug4_proj.pdf (дата звернення: 07.05.2026).

26. Express.js: Fast, unopinionated, minimalist web framework for Node.js. URL: https://expressjs.com/ (дата звернення: 07.05.2026).

27. Mongoose ODM v7.0.0 Documentation. URL: https://mongoosejs.com/docs/7.x/docs/guide.html (дата звернення: 07.05.2026).

28. React Router Documentation. URL: https://reactrouter.com/ (дата звернення: 07.05.2026).

29. Vite: Next Generation Frontend Tooling. URL: https://vite.dev/ (дата звернення: 07.05.2026).

30. Максютенко І. Є. Аналіз тенденцій розвитку електронної комерції. Економіка та суспільство. 2024. № 64. URL: https://doi.org/10.32782/2524-0072/2024-64-54 (дата звернення: 10.05.2026).

# ДОДАТКИ

**Додаток А**

## Структура моделі Product

```
const productSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    trim: true
  },
  price: {
    type: Number,
    required: true,
    min: 0
  },
  compareAtPrice: {
    type: Number,
    default: null,
    min: 0,
    validate: {
      validator(value) {
        const price = typeof this.get === 'function' ?
this.get('price') : this.price;
        return value == null || price == null || value > price;
      },
      message: 'Compare-at price must be greater than product
price'
    }
  },
  stock: {
    type: Number,
    default: 0
  },
  description: {
    type: String,
    trim: true
},
  image: String,
  category: {
    type: String,
    default: 'phones'
  },
  brand: {
    type: String,
    trim: true
  },
  attributes: [{
    groupName: {
      type: String,
      required: true,
      trim: true
```

```
},
    items: [{
      key: {
        type: String,
        required: true,
        trim: true
      },
      value: {
        type: String,
        default: ''
      }
    }]
  }]
}, {
  timestamps: true
});
```

**Додаток Б**

## JWT middleware та перевірка ролей

```
const jwt = require('jsonwebtoken');

const JWT_SECRET = process.env.JWT_SECRET;

const authenticateToken = (req, res, next) => {
  const authHeader = req.headers.authorization;
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({
      success: false,
      message: 'Токен відсутній'
    });
  }

  jwt.verify(token, JWT_SECRET, (err, user) => {
    if (err) {
      return res.status(403).json({
        success: false,
        message: 'Невалідний токен'
      });
    }

    req.user = user;
    next();
  });
};

const requireRole = (...roles) => {
  return (req, res, next) => {
    if (req.user && roles.includes(req.user.role)) {
      return next();
 }
    return res.status(403).json({
      success: false,
      message: 'Доступ заборонено. Потрібні права адміністратора.'
    });
  };
}
```

**Додаток В**

## Основна структура docker-compose.yml

```
services:
  database:
    image: mongo:latest
    restart: unless-stopped
    ports:
      - "27018:27017"
    volumes:
      - mongo-data:/data/db

  backend:
    build:
      context: ./server
    restart: unless-stopped
    depends_on:
      - database
    ports:
      - "5000:5000"
    environment:
      PORT: 5000
      MONGODB_URI: mongodb://database:27017/eshop-admin
      MONGO_URI: mongodb://database:27017/eshop-admin
      JWT_SECRET: ${JWT_SECRET}
      NOVA_POSHTA_API_KEY: ${NOVA_POSHTA_API_KEY:-}

  frontend:
    build:
      context: .
      dockerfile: client/Dockerfile
      args:
        VITE_API_URL: /api
    restart: unless-stopped
   depends_on:
      - backend
    ports:
      - "80:80"

volumes:
  mongo-data:
```

**ЗГОДА**
**здобувача вищої освіти**
Державного університету економіки і технологій
про перевірку кваліфікаційної роботи на прояви академічного плагіату та розміщення в Репозитарії Університету

Я, **Акбар Фаяд Фарідович**,

підтримую політику Державного університету економіки і технологій з академічної доброчесності і відкритого доступу.

Засвідчую, що кваліфікаційна бакалаврська робота

**«Розробка та вдосконалення веб-додатка інтернет-магазину з адміністративною панеллю та розширеним функціоналом взаємодії з користувачем»**

виконана самостійно та не містить академічного плагіату. Я не надавав і не одержував недозволену допомогу під час підготовки цієї роботи. Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Державного університету економіки і технологій ознайомлений. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення норм академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Також я поінформований, що відповідно до «Положення про Репозитарій (електронну базу даних) Державного університету економіки і технологій» зазначена робота буде розміщена в Електронному архіві Університету (Репозитарії ДУЕТ). З умовами такого розміщення ознайомлений.

16.06.26 р  Акбар Ф. Ф.