ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-сайту інтернет-магазину відеогрових консолей "BiTZone" з повним циклом функціонування»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Владислав КИСІЛЬ
(підпис)

Виконав: здобувач вищої освіти групи ПД-42
Владислав КИСІЛЬ

Керівник: Данило КОВАЛЕНКО
доктор філософії (PhD)

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кисілю Владиславу Олександровичу

1. Тема кваліфікаційної роботи: «Розробка web-сайту інтернет-магазину відеогрових консолей "BiTZone" з повним циклом функціонування»
керівник кваліфікаційної роботи доктор філософії PhD Данило КОВАЛЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про електронну комерцію, опис підходів до розробки web-сайтів інтернет-магазинів, технічна документація з описом React, Node.js, Express, MongoDB, Nova Poshta, Monobank та RoApp.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної галузі онлайн-продажу відеогрових консолей та існуючих програмних рішень електронної комерції.
2. Проектування web-сайту інтернет-магазину відеогрових консолей "BiTZone".

3. Програмна реалізація та опис функціонування web-сайту інтернет-магазину відеогрових консолей "BiTZone".
4. Тестування web-сайту.

5. Перелік графічного матеріалу: *презентація*
1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма розгортання.
6. Діаграма класів.
7. Екранні форми.
8. Апробація результатів дослідження.

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.11-28.11.2025	
2	Аналіз та дослідження існуючих аналогів	05.12-10.12.2025	
3	Аналіз предметної галузі онлайн-продажу відеогрових консолей та супутніх товарів	18.12-22.12.2025	
4	Проектування web-сайту інтернет-магазину відеогрових консолей "BiTZone"	15.12-24.12.2025	
5	Програмна реалізація web-сайту	26.12-21.02.2026	
6	Тестування web-сайту	22.02-25.02.2026	
7	Оформлення роботи: вступ, висновки, реферат	01.03-4.04.2026	
8	Розробка демонстраційних матеріалів	10.04-24.04.2026	
9	Попередній захист роботи	11.05-22.05.2026	

Здобувач вищої освіти _____ Владислав КИСІЛЬ
(підпис)

Керівник
кваліфікаційної роботи _____ Данило КОВАЛЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 72 стор., 13 табл., 20 рис., 27 джерел.

Мета роботи - підвищення зручності та ефективності онлайн-замовлення відеогрових консолей і супутніх товарів шляхом розробки web-сайту інтернет-магазину з повним циклом функціонування та інтеграцією зовнішніх сервісів.

Об'єкт дослідження - процес онлайн-продажу відеогрових консолей і супутніх товарів.

Предмет дослідження - програмні засоби автоматизації керування каталогом товарів, замовленнями, доставкою, оплатою та взаємодією користувача з web-сайтом інтернет-магазину.

Короткий зміст роботи: У роботі проаналізовано предметну галузь онлайн-продажу відеогрових консолей і супутніх товарів, розглянуто особливості повного циклу замовлення та виконано аналіз програмних рішень електронної комерції. Сформовано функціональні й нефункціональні вимоги до системи, спроєктовано архітектуру, структуру бази даних, сценарії взаємодії користувача та інтерфейс. Реалізовано клієнтську частину на React, серверну частину на Node.js та Express, збереження даних у MongoDB, а також інтеграції з Nova Poshta, Monobank і RoApp. Проведено тестування основних функціональних модулів системи.

Сферою використання розробленої системи є організація онлайн-продажу відеогрових консолей, аксесуарів та супутньої продукції в межах спеціалізованого інтернет-магазину.

КЛЮЧОВІ СЛОВА: ЕЛЕКТРОННА КОМЕРЦІЯ, ІНТЕРНЕТ-МАГАЗИН, ВІДЕОГРОВІ КОНСОЛІ, REACT, NODE.JS, EXPRESS, MONGODB, JWT, NOVA POSHTA, MONOBANK, ROAPP.

ЗМІСТ

ВСТУП

Сучасний розвиток електронної комерції зумовлює підвищення вимог до якості програмних рішень, що забезпечують продаж товарів у мережі Інтернет. Для інтернет-магазинів, які спеціалізуються на відеогрових консолях і супутніх товарах, важливими є не лише наявність каталогу, а й зручний пошук, зрозуміла навігація, інформативна картка товару, швидке оформлення замовлення, інтеграція із сервісами доставки та оплати, а також можливість перегляду історії покупок у особистому кабінеті. За таких умов розробка web-сайту інтернет-магазину з повним циклом функціонування є актуальним завданням у сфері програмної інженерії [1,2,3].

Наявні платформи електронної комерції часто потребують адаптації до специфіки продажу відеогрових консолей, аксесуарів та супутньої продукції. Частина рішень орієнтована на універсальне використання, однак не забезпечує необхідної комбінації функціональних можливостей у межах єдиної системи або вимагає значного доопрацювання для побудови повного користувацького циклу. Це зумовлює доцільність створення спеціалізованого web-рішення, орієнтованого саме на предметну галузь онлайн-продажу відеогрових консолей [4,5,6,7].

Об’єктом дослідження є процес онлайн-продажу відеогрових консолей і супутніх товарів.

Предметом дослідження є програмні засоби автоматизації керування каталогом товарів, замовленнями, доставкою, оплатою та взаємодією користувача з web-сайтом інтернет-магазину.

Метою роботи є підвищення зручності та ефективності онлайн-замовлення відеогрових консолей і супутніх товарів шляхом розробки web-сайту інтернет-магазину з повним циклом функціонування та інтеграцією зовнішніх сервісів.

Для досягнення поставленої мети в роботі визначено такі завдання:

1. дослідити особливості предметної галузі онлайн-продажу відеогрових консолей і супутніх товарів;

2. виконати аналіз існуючих програмних рішень для електронної комерції та встановити їх сильні й слабкі сторони;

3. обґрунтувати вибір технологій і засобів, доцільних для розробки системи;

4. сформувати функціональні та нефункціональні вимоги до web-сайту;

5. спроєктувати архітектуру системи, структуру даних та інтерфейс користувача;

6. реалізувати основні програмні модулі клієнтської та серверної частин;

7. забезпечити інтеграцію із сервісами доставки, оплати та синхронізації товарних даних;

8. провести тестування функціональності розробленої системи.

У роботі використано методи аналізу, порівняння, узагальнення та моделювання. Метод аналізу застосовано під час дослідження предметної галузі та наявних аналогів. Метод порівняння використано для оцінювання можливостей існуючих рішень і формування вимог до власної системи. Методи моделювання використано при проєктуванні архітектури, бази даних і сценаріїв взаємодії користувача з web-сайтом. Під час реалізації та перевірки працездатності системи застосовано методи проєктування програмного забезпечення і функціонального тестування [8,9,10,11].

Наукова новизна роботи полягає у розробці спеціалізованого web-сайту інтернет-магазину відеогрових консолей, у межах якого поєднано каталог товарів, пошук, фільтрацію, картку товару, кошик, список бажань, особистий кабінет, оформлення замовлення, інтеграцію зі службою доставки Nova Poshta, оплату через Monobank, а також синхронізацію товарів і категорій із зовнішньою системою RoApp. Практична цінність роботи полягає в можливості використання створеного рішення як основи для функціонування та подальшого розвитку спеціалізованого інтернет-магазину [12,13,14].

Структура роботи визначається поставленою метою та логікою дослідження. Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, переліку посилань і додатків. У першому розділі розглянуто предметну галузь, аналоги та засоби реалізації. Другий розділ присвячено проєктуванню системи. У третьому розділі подано опис реалізації основних модулів web-сайту. У четвертому розділі наведено підходи до тестування, тест-кейси та результати перевірки системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ІСНУЮЧИХ ЗАСОБІВ РЕАЛІЗАЦІЇ

Онлайн-продаж відеогрових консолей і супутніх товарів належить до тих напрямів електронної комерції, у яких якість програмного рішення безпосередньо впливає на готовність користувача завершити покупку. На відміну від товарів повсякденного попиту, вибір консолі зазвичай пов’язаний із попереднім порівнянням моделей, аналізом технічних характеристик, оцінюванням комплектації, сумісності з аксесуарами та доступних способів отримання замовлення. З цієї причини інтернет-магазин у такій сфері повинен виконувати не лише інформаційну функцію, а й забезпечувати логічно побудований, зручний і безперервний сценарій взаємодії користувача з системою.

Для спеціалізованого web-сайту важливими стають кілька взаємопов’язаних складових. По-перше, система має надавати структурований каталог із можливістю швидко знайти потрібний товар, відфільтрувати позиції за категоріями або характеристиками та перейти до детальної картки товару. По-друге, необхідно забезпечити зручні механізми формування покупки: додавання товару до кошика, збереження позицій у списку бажань, оформлення замовлення, вибір доставки та оплати. По-третє, інтернет-магазин повинен підтримувати післяпродажну взаємодію з користувачем, зокрема через особистий кабінет, перегляд замовлень та доступ до актуальної інформації про їх стан. Якщо хоча б одна з цих частин реалізована незручно або непослідовно, це негативно впливає на сприйняття сервісу в цілому.

Окрему складність становить те, що робота такого магазину не обмежується лише відображенням сторінок у браузері. За зовнішньо простим інтерфейсом стоїть система, яка працює з даними про товари, категорії, користувачів і замовлення, а також взаємодіє із зовнішніми сервісами доставки, оплати та джерелами товарної інформації. Саме тому розробка інтернет-

магазину відеогрових консолей є комплексною задачею програмної інженерії, де поєднуються вимоги до інтерфейсу, серверної логіки, збереження даних, інтеграцій та загальної узгодженості роботи всіх модулів.

Необхідність створення спеціалізованого web-сайту для цієї предметної галузі пояснюється також тим, що універсальні платформи електронної комерції не завжди враховують специфіку продажу відеогрових консолей у потрібній комбінації можливостей. Частина рішень пропонує базовий набір функцій для каталогу та замовлень, однак потребує значної адаптації під конкретну структуру товарів і бізнес-процеси. Інші системи є функціонально насиченими, але складними в налаштуванні або надмірними для задач спеціалізованого магазину. Тому перед розробкою web-сайту «BiTZone» необхідно послідовно дослідити особливості предметної галузі, користувацькі потреби, наявні програмні аналоги та технологічні підходи, що можуть бути використані для реалізації системи з повним циклом функціонування.

1.1 Особливості предметної галузі онлайн-продажу відеогрових консолей та супутніх товарів

Продаж відеогрових консолей через web-сайт має свою специфіку, оскільки в даному випадку користувач купує не просто окремий товар, а фактично обирає цілу платформу для подальшого використання. Рішення про покупку часто залежить від сукупності параметрів: покоління консолі, технічних характеристик, обсягу пам’яті, комплектації, наявності додаткових контролерів, сумісності з іграми та аксесуарами. Саме тому для такого інтернет-магазину недостатньо лише відобразити назву товару, ціну і кнопку оформлення замовлення. Система повинна надавати користувачу достатньо інформації для усвідомленого вибору і водночас не перевантажувати його зайвими деталями.

Ще однією особливістю цієї предметної галузі є різнорідність асортименту. Окрім самих консолей, магазин може містити ігри, геймпади,

зарядні станції, гарнітури, кабелі, підписки, тематичні набори та інші супутні товари. Усі ці позиції мають різні характеристики, належать до різних категорій і часто пов’язані між собою. Наприклад, покупець може шукати конкретну консоль, але одночасно очікувати, що сайт запропонує сумісний контролер, гру або аксесуар. Через це система повинна підтримувати чітку категоризацію, гнучку структуру карток товарів і зручну навігацію між пов’язаними позиціями.

Для покупця у цій сфері особливо важливим є етап попереднього вибору. Людина зазвичай не переходить одразу до оформлення замовлення, а спочатку переглядає кілька варіантів, звіряє характеристики, оцінює зовнішній вигляд товару, читає опис, перевіряє наявність, умови доставки та оплати. Це означає, що якість каталогу, пошуку, сортування і фільтрації безпосередньо впливає на ефективність роботи магазину. Якщо користувач не може швидко знайти потрібну модель або не розуміє відмінностей між кількома товарами, імовірність завершення покупки помітно знижується.

Окрему роль у предметній галузі відіграє довіра до продавця. Для технічно складних і відносно дорогих товарів користувач звертає увагу не лише на сам продукт, а й на ознаки надійності магазину: наявність повного опису, чітко сформульовані умови доставки й повернення, зрозумілий механізм оплати, особистий кабінет, можливість переглянути історію замовлень, зберегти обрані позиції та повернутися до них пізніше. З програмної точки зору це означає, що web-сайт має забезпечувати не тільки вітринну частину, а й стабільну роботу облікового запису користувача, кошика, списку бажань, оформлення замовлення та механізмів збереження даних.

Предметна галузь онлайн-продажу відеогрових консолей тісно пов’язана і з зовнішніми сервісами. Реальна робота магазину неможлива без актуальних даних про товари, обробки замовлень, організації доставки та приймання оплати. Через це програмне забезпечення такого типу зазвичай не є повністю автономним: воно повинно взаємодіяти з поштовими сервісами, платіжними системами та джерелами товарної інформації. Від якості цієї взаємодії залежить

коректність оформлення замовлення, точність адресної інформації, статус оплати та узгодженість даних між різними модулями системи.

З позиції програмної інженерії дана предметна галузь вимагає поєднання кількох рівнів логіки в одному рішенні. Перший рівень — інформаційний, тобто подання товарів, категорій, характеристик і описів. Другий — користувацький, пов’язаний із навігацією по сайту, авторизацією, списком бажань, кошиком і кабінетом. Третій — транзакційний, який охоплює оформлення замовлення, доставку, оплату та збереження історії покупок. Якщо ці частини розроблено окремо одна від одної без узгодженого підходу, сайт може виглядати завершеним лише зовні, але працювати непослідовно на етапах реальної взаємодії користувача із системою.

Таким чином, предметна галузь онлайн-продажу відеогрових консолей характеризується поєднанням інформаційної насиченості, комерційної логіки та високих вимог до зручності користування. Для ефективної роботи інтернет-магазину недостатньо створити окремі сторінки каталогу чи реалізувати базову форму замовлення. Необхідно побудувати цілісну систему, у якій усі етапи — від вибору товару до завершення покупки — логічно пов’язані між собою. Саме така специфіка предметної галузі і формує основу для розробки web-сайту інтернет-магазину відеогрових консолей «BiTZone» з повним циклом функціонування.

1.2 Аналіз бізнес-процесу функціонування

Робота інтернет-магазину не обмежується показом товарів на сторінках сайту. Якщо розглядати її не з позиції інтерфейсу, а як послідовність дій, стає видно, що магазин фактично обслуговує цілий бізнес-процес: від появи товару в каталозі до завершення оплати і подальшого перегляду замовлення користувачем. Для спеціалізованого магазину відеогрових консолей цей процес є особливо важливим, оскільки покупець зазвичай проходить кілька етапів вибору і не приймає рішення миттєво.

Перший етап пов’язаний з підготовкою товарних даних. До того як користувач відкриє сайт, система вже повинна містити коректну інформацію про моделі консолей, аксесуари, ігри, комплектацію, ціну, категорії, зображення та наявність. Якщо на цьому рівні виникають помилки, вони переходять далі по всьому ланцюгу: користувач бачить неточний опис, додає не той товар у кошик або отримує хибне уявлення про можливості товару. Тому підтримка актуального каталогу є не допоміжною задачею, а початковою умовою нормальної роботи всього магазину.

Після цього починається етап взаємодії користувача з каталогом. Людина заходить на сайт не для того, щоб переглядати всі товари підряд, а щоб знайти потрібну позицію або хоча б звузити вибір. У випадку з відеогровими консолями покупець часто орієнтується на конкретну платформу, серію пристрою, обсяг пам’яті, тип комплектації або бренд аксесуара. Через це бізнес-процес на цьому етапі залежить від якості пошуку, фільтрації, сортування і структури категорій. Якщо сайт не дозволяє швидко перейти до потрібного товару, користувач або витрачає зайвий час, або взагалі припиняє взаємодію із системою.

Коли потрібний товар знайдено, наступним кроком стає ознайомлення з карткою товару. Саме тут користувач приймає попереднє рішення, чи переходити до покупки. Для цього йому потрібні не лише назва і ціна, а й технічні характеристики, опис, фотографії, інформація про комплектацію, наявність та пов’язані товари. У цій частині бізнес-процесу сайт виконує роль консультанта: він має дати достатньо інформації, щоб зменшити невизначеність і підвести користувача до дії. Якщо сторінка товару не виконує цієї функції, кошик і оформлення замовлення втрачають сенс, бо користувач не доходить до них.

Далі починається етап формування покупки. На практиці це відбувається через додавання товару до кошика або збереження його у списку бажань. Такі дії вже означають, що користувач не просто переглядає асортимент, а переходить до більш конкретного сценарію. Для системи в цей момент важливо

зберегти обрані позиції, правильно відобразити їх у кошику, дозволити змінити кількість, видалити непотрібні товари або повернутися до каталогу без втрати вже сформованого набору. У багатьох магазинах саме на цьому етапі виникає найбільше проблем для користувача: неочікуване очищення кошика, дублювання товарів, незрозуміла логіка оновлення кількості. Тому кошик є одним із центральних елементів повного циклу замовлення.

Наступний крок — оформлення замовлення. Тут бізнес-процес переходить у більш формалізовану частину, оскільки система починає працювати не просто з товарами, а вже з конкретним замовленням. Користувач вводить або підтверджує свої дані, вказує спосіб отримання товару, обирає місто, відділення чи інші параметри доставки. У цей момент сайт повинен не лише зібрати інформацію, а й зробити це послідовно та зрозуміло. Якщо людина не бачить складу свого замовлення, не розуміє, які дані є обов’язковими, або не може швидко перейти до наступного кроку, процес оформлення стає джерелом помилок і відмов.

Після введення даних настає етап доставки й оплати. Саме тут система починає активно взаємодіяти із зовнішніми сервісами. Для магазину типу BiTZone це означає роботу з даними служб доставки та платіжних інструментів. На цьому етапі особливо важлива узгодженість дій: користувач повинен отримати доступні варіанти доставки, коректно обрати точку отримання замовлення, перейти до оплати і після виконання операції повернутися до системи без втрати інформації. Якщо інтеграції працюють нестабільно, користувач починає сумніватися не лише в оплаті, а в надійності всього магазину.

Після підтвердження замовлення процес не завершується. З погляду користувача важливо мати можливість знову відкрити сайт і побачити, що замовлення збережене, які товари в нього входять, на яку адресу воно оформлене і в якому стані перебуває. Тому завершальним етапом бізнес-процесу є збереження історії замовлень і робота особистого кабінету. Це вже не етап разової покупки, а елемент довшої взаємодії між клієнтом і магазином.

Саме завдяки цьому сайт стає повноцінним сервісом, а не одноразовою формою для оформлення товару.

Якщо зібрати все разом, повний цикл роботи інтернет-магазину включає кілька послідовних частин: актуалізацію каталогу, пошук товару, перегляд картки, формування кошика, оформлення замовлення, вибір доставки, оплату та подальший перегляд замовлення в кабінеті. Кожна з цих частин окремо важлива, але реальну цінність система отримує лише тоді, коли вони працюють як єдиний безперервний процес. Саме тому для web-сайту «BiTZone» доцільно розглядати повний цикл замовлення як основу всієї архітектури системи, а не як набір незалежних модулів.

1.3 Аналіз існуючих програмних рішень для організації електронної комерції

Перед розробкою власного web-сайту інтернет-магазину доцільно розглянути програмні рішення, які вже застосовуються в електронній комерції. Це потрібно не для того, щоб повторити готову платформу, а для розуміння того, як у різних системах реалізовано каталог товарів, сторінки продуктів, кошик, оформлення замовлення, адміністративну частину та інтеграцію з іншими сервісами. Для магазину відеогрових консолей та супутніх товарів такий аналіз особливо важливий, оскільки в цій сфері значення має не тільки сам факт продажу, а й зручність вибору моделі, навігація по асортименту, логіка переходу між етапами покупки та стабільність роботи всіх пов’язаних модулів.

Одним із найбільш відомих рішень є Shopify. Його популярність пояснюється тим, що ця платформа дозволяє досить швидко запустити інтернет-магазин без окремого проєктування всієї базової інфраструктури. У системі вже передбачено каталог товарів, картки продуктів, кошик, оформлення замовлення та адміністративні засоби для керування магазином. Для багатьох типових магазинів цього достатньо. Проте у випадку спеціалізованого магазину

такий підхід не завжди виявляється оптимальним. Платформа добре працює там, де бізнес-процес не потребує глибокої адаптації, але коли виникає потреба перебудувати логіку каталогу, інтегрувати нестандартні зовнішні сервіси або змінити послідовність взаємодії користувача із системою, гнучкість рішення вже не здається такою очевидною. [4].

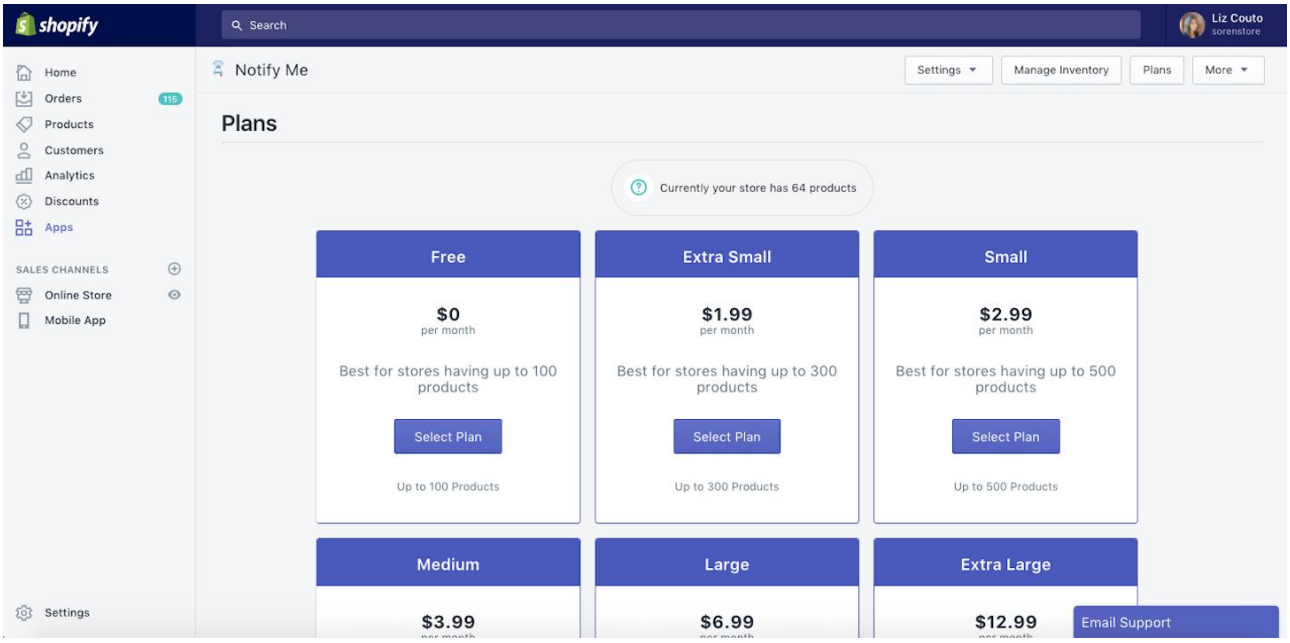


Рис. 1.1 Інтерфейс платформи Shopify

Інший поширений варіант - WooCommerce. На практиці він часто використовується там, де магазин створюється на базі WordPress і повинен поєднувати комерційну частину з інформаційним вмістом сайту. Його перевагою можна вважати доступність, велику кількість розширень і можливість поступово розширювати функціональність. Однак саме залежність від плагінів є і його слабким місцем. Коли ключові можливості магазину будуються з окремих модулів, виникає ризик того, що система працюватиме не як цілісний продукт, а як набір частково узгоджених елементів. Для спеціалізованого магазину це небажано, оскільки порушення логіки хоча б на одному етапі - наприклад, у кошику або при оформленні замовлення — впливає на загальне враження від сервісу [5].

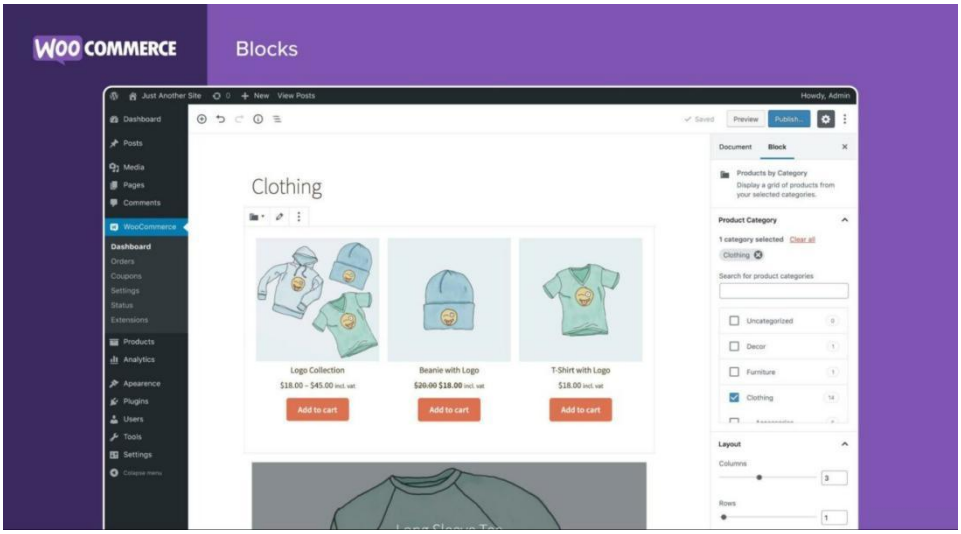


Рис. 1.2 Інтерфейс платформи WooCommerce

Adobe Commerce розглядається як більш складне і функціонально насичене рішення. Його часто пов’язують із великими проєктами, де потрібне розширене адміністрування, гнучке керування товарами, замовленнями, клієнтами та маркетинговими механізмами. З технічної точки зору це сильна платформа, але для вузькоорієнтованого інтернет-магазину вона може бути надмірною. Велика кількість можливостей не завжди означає, що рішення буде зручним саме в конкретній предметній галузі. У деяких випадках надлишкова складність системи не спрощує роботу, а навпаки, ускладнює адаптацію, підтримку та подальший розвиток проєкту [6].

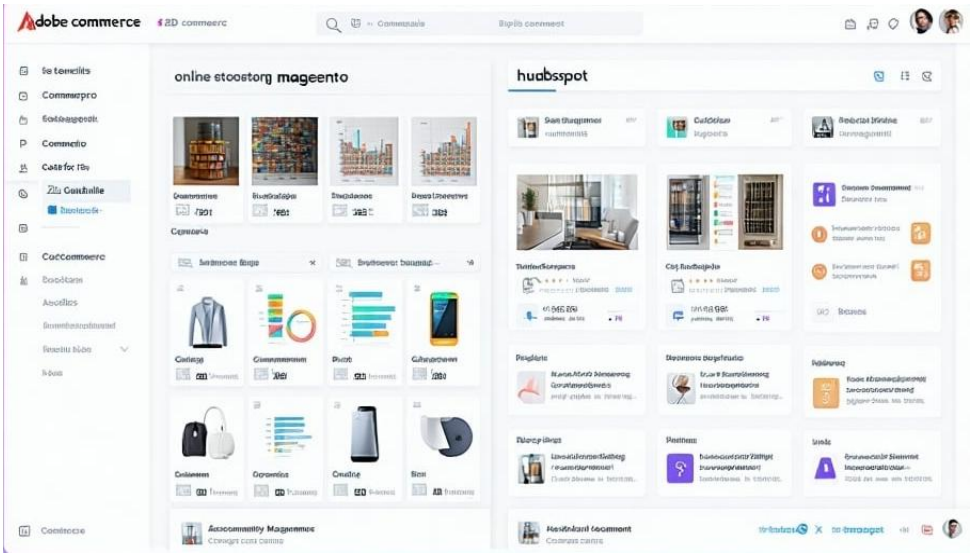


Рис. 1.3 Інтерфейс платформи Adobe Commerce

Ще одним відомим рішенням є OpenCart. Воно тривалий час застосовується для створення інтернет-магазинів різного призначення і приваблює порівняно простою структурою. Для базового запуску електронної комерції цього часто достатньо: є категорії, товари, кошик, оформлення замовлення та адміністративна частина. Проте така простота має межі. Якщо магазин потребує більш складної логіки роботи, персоналізованої взаємодії з користувачем, розширеної інтеграції або точного підлаштування під вузьку категорію товарів, стандартних засобів стає недостатньо. У такому разі доводиться змінювати архітектуру рішення або суттєво розширювати його додатковими модулями [7].

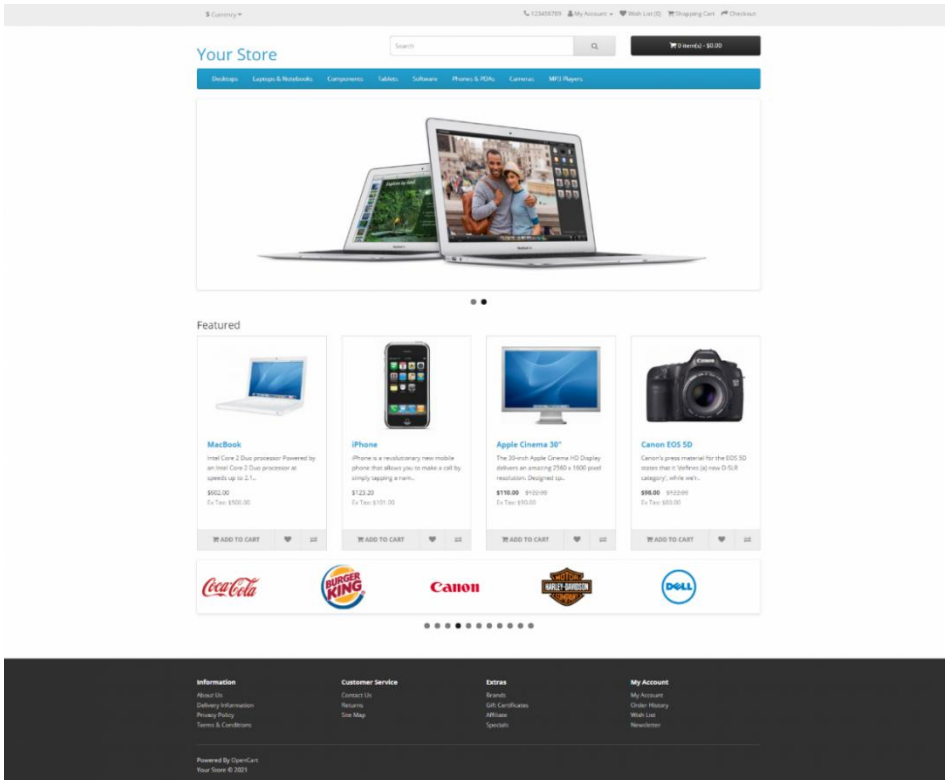


Рис. 1.4 Інтерфейс платформи OpenCart

Якщо оцінювати всі ці системи разом, можна помітити спільну рису: вони створювалися як універсальні інструменти для електронної комерції. Саме в цьому їхня сила, але одночасно і обмеження. Магазин відеогрових консолей має справу не просто з продажем товарів загального призначення. Тут важливі

технічні характеристики, сумісність платформ, різні типи комплектацій, зв’язок між основними товарами та аксесуарами, а також послідовність дій користувача від вибору товару до завершення оплати. Якщо система добре реалізує загальні функції магазину, але не дозволяє гнучко вибудувати саме цей сценарій, її придатність для конкретного проєкту стає обмеженою.

Універсальна платформа може прискорити старт, але не завжди дає змогу побудувати сервіс саме так, як цього вимагає конкретний бізнес-процес. Для одних проєктів це не є проблемою, але для спеціалізованого магазину, де важливо забезпечити повний цикл взаємодії користувача із системою, така відмінність є принциповою. Користувач повинен не просто переглянути товар і залишити заявку, а пройти весь шлях: знайти потрібну модель, оцінити її характеристики, додати до кошика, оформити замовлення, обрати доставку, здійснити оплату і пізніше переглянути інформацію про свою покупку. Саме в межах такого повного циклу й потрібно оцінювати придатність аналогічних рішень.

Отже, існуючі програмні рішення для організації електронної комерції демонструють різні підходи до побудови інтернет-магазину, але не усувають потреби у створенні власного спеціалізованого web-сайту. Для теми даної роботи важливим є не саме використання популярної платформи, а побудова системи, у якій усі етапи онлайн-покупки логічно пов’язані між собою і підпорядковані потребам користувача. Саме тому подальший аналіз доцільно зосередити не лише на переліку функцій окремих платформ, а й на їх сильних і слабких сторонах з позиції розробки web-сайту інтернет-магазину відеогрових консолей «BiTZone» з повним циклом функціонування.

1.4 Порівняльний аналіз аналогів та обґрунтування актуальності розробки web-сайту BiTZone

Після розгляду наявних платформ можна зробити висновок, що всі вони вирішують задачу онлайн-продажу, але роблять це по-різному і з різним

ступенем придатності для вузькоспеціалізованого магазину. Для звичайного інтернет-магазину універсальних можливостей часто достатньо. Проте у випадку продажу відеогрових консолей ситуація складніша. Тут користувач не просто обирає товар із загального списку, а проходить послідовний шлях: шукає потрібну платформу, порівнює моделі, оцінює комплектацію, звертає увагу на сумісність з аксесуарами, додає товари до кошика або списку бажань, після чого оформлює доставку і оплату. Саме така логіка взаємодії висуває до системи вищі вимоги, ніж типова схема “товар — кошик — замовлення”.

Shopify показує сильний результат у тих випадках, коли головною перевагою є швидкість запуску. Система вже містить основні елементи інтернет-магазину і дозволяє відносно швидко перейти до комерційного використання. Однак ця зручність досягається за рахунок обмеженої свободи при глибокому налаштуванні. Якщо магазин повинен працювати не за стандартною моделлю, а з урахуванням окремої логіки товарів, користувацьких сценаріїв та інтеграцій, платформа починає нав’язувати власний підхід до організації процесів [4].

WooCommerce, у свою чергу, є більш гнучким у налаштуванні, особливо якщо магазин створюється як частина сайту на WordPress. Це корисно в тих випадках, коли важливо поєднати торгову частину з контентом, новинами, статтями чи оглядами. Проте на практиці така гнучкість часто залежить від набору сторонніх розширень. Чим складнішою стає система, тим сильніше вона прив’язується до модулів, що можуть відрізнятися за якістю, сумісністю і стабільністю. Для магазину, де весь процес покупки має працювати без розривів, така залежність не завжди є бажаною [5].

Adobe Commerce виглядає найбільш потужним рішенням серед розглянутих. Воно розраховане на масштабніші проєкти, складніші бізнес-процеси і глибше адміністрування. Саме тому ця платформа добре підходить для великих комерційних систем, де є потреба в розширеному керуванні товарами, замовленнями, клієнтами та маркетинговими інструментами. Але для спеціалізованого магазину середнього масштабу така система може виявитися

надто важкою. Частина її можливостей просто не буде використана повною мірою, а загальна складність підтримки зростатиме [6].

OpenCart займає більш просту нішу. Його часто обирають саме через відносну легкість структури й зрозумілу організацію основних модулів. Для невеликих магазинів це справді зручний варіант. Але щойно з'являється потреба в більш глибокій адаптації, персоналізації інтерфейсу, складніших інтеграціях або нетиповому сценарії покупки, базові можливості системи вже не виглядають достатніми. У результаті магазин або зупиняється на простому рівні, або поступово обростає додатковими рішеннями, які не завжди утворюють цілісну систему [7].

Якщо порівнювати ці платформи саме з позиції web-сайту BiTZone, стає помітно, що жодна з них не дає ідеального поєднання всіх потрібних характеристик. Одні рішення сильні у швидкому розгортанні, другі — у розширюваності, треті — у масштабі, четверті — у простоті. Але для даної теми важливо інше: не окрема сильна сторона, а збалансоване поєднання кількох функціональних можливостей у межах одного спеціалізованого продукту. Магазин відеогрових консолей повинен поєднувати зручний каталог, інформативну картку товару, пошук, фільтрацію, список бажань, кошик, особистий кабінет, логіку оформлення замовлення, роботу з доставкою і оплатою, а також актуалізацію товарних даних. Саме в такій комбінації й виникає реальна складність розробки.

Актуальність створення «BiTZone» пояснюється тим, що власне рішення дозволяє будувати систему не навколо універсальної платформи, а навколо конкретного користувацького шляху. Це означає, що структура каталогу, спосіб подання товару, логіка переходу між сторінками, поведінка кошика, оформлення замовлення й інтеграція з зовнішніми сервісами можуть проектуватися одразу під потреби магазину відеогрових консолей. У такому підході немає необхідності підлаштовувати предметну область під уже готову платформу. Навпаки, сама система створюється виходячи з особливостей товарів, очікувань покупця і реального сценарію онлайн-покупки.

Таким чином, порівняльний аналіз аналогів показує, що готові платформи електронної комерції є корисним орієнтиром, але не розв’язують задачу повністю. Для розробки web-сайту «BiTZone» доцільним є створення окремого програмного рішення, у якому основні функції інтернет-магазину будуть об’єднані в єдину, узгоджену систему з повним циклом функціонування.

1.5 Аналіз засобів і технологій

Вибір технологічної основи для інтернет-магазину не зводиться до пошуку найпопулярнішого фреймворку чи найвідомішої бази даних. Для системи типу BiTZone важливо, щоб усі складові працювали узгоджено: інтерфейс повинен швидко реагувати на дії користувача, серверна частина — коректно обробляти запити, база даних — зберігати інформацію про товари, користувачів і замовлення, а інтеграційні модулі — підтримувати доставку, оплату та синхронізацію даних. Саме тому під час аналізу засобів і технологій доцільно оцінювати не окремі інструменти самі по собі, а їх придатність до побудови цілісного web-рішення.

Для клієнтської частини інтернет-магазину можуть використовуватися різні підходи. Найпростішим є формування сторінок переважно на сервері з мінімальною кількістю динамічної логіки в браузері. Такий варіант є придатним для невеликих інформаційних сайтів, однак для магазину з каталогом, фільтрацією, кошиком, списком бажань і кабінетом користувача він менш зручний. У подібній системі користувач постійно взаємодіє зі сторінкою, тому важливими стають швидке оновлення даних, збереження стану інтерфейсу та можливість змінювати вміст сторінки без повного перезавантаження. Через це більш доречним є застосування клієнтських бібліотек і фреймворків, які підтримують компонентну побудову інтерфейсу.

Серед поширених рішень для фронтенду доцільно розглядати React, Angular і Vue. Angular зазвичай використовують у великих системах із більш жорсткою структурою проєкту, де команда працює за чітко визначеними

правилами. Vue відомий своєю простотою і зручністю для порівняно швидкого входження в розробку. React посідає проміжне, але дуже практичне місце: він не нав’язує надмірно жорстку структуру, водночас має зрілу екосистему, велику кількість бібліотек і добре підходить для побудови динамічних web-інтерфейсів. Для BiTZone такий вибір є виправданим, оскільки магазин містить багато взаємопов’язаних елементів — картки товарів, каталог, кошик, форму оформлення замовлення, авторизацію, кабінет користувача — і вся ця взаємодія зручно реалізується саме в компонентному підході [15].

Окрім бібліотеки для побудови інтерфейсу, важливими є допоміжні фронтенд-засоби. Маршрутизація дає можливість логічно організувати переходи між основними сторінками магазину. Керування станом потрібне там, де система одночасно працює з даними кошика, списку бажань, авторизації, замовлень і поточного стану користувацької сесії. Засоби стилізації відповідають не лише за зовнішній вигляд, а й за повторне використання інтерфейсних елементів, зручність підтримки та цілісність дизайну. Для магазину, у якому значна частина взаємодії відбувається на клієнтському рівні, ці засоби мають не допоміжне, а структурне значення [16].

Серверна частина інтернет-магазину повинна виконувати іншу групу завдань. Вона відповідає за доступ до товарних даних, обробку користувацьких запитів, авторизацію, роботу із замовленнями, збереження відгуків, підключення доставки, оплати та взаємодію із зовнішніми сервісами. Для реалізації такої логіки можуть використовуватися різні технологічні стеки: Python з Django або FastAPI, Java з відповідними серверними фреймворками, C# з платформою .NET, а також JavaScript-сервер на базі Node.js. Кожен варіант має свої переваги, однак для системи, де клієнтська частина вже реалізована засобами JavaScript, практичним є використання Node.js, оскільки це спрощує загальний стек і дозволяє будувати API в межах тієї самої мовної екосистеми[17, 18].

Для побудови серверної логіки доцільно використовувати Express або подібні фреймворки, які дають змогу керувати маршрутизацією, обробкою

запитів, middleware та структурою API. У випадку BiTZone це є виправданим, оскільки серверна частина повинна залишатися достатньо гнучкою: система працює з товарами, категоріями, користувачами, замовленнями, платіжними запитами, сервісами доставки та зовнішніми webhook-подіями. Така логіка вимагає не надмірно жорсткої серверної архітектури, а зручного інструментарію для побудови окремих функціональних модулів і їх взаємодії[18].

Під час вибору засобу зберігання даних також необхідно враховувати особливості структури майбутньої системи. У магазині зберігаються різні типи сутностей: товари, категорії, замовлення, облікові записи користувачів, відгуки, адресні дані, параметри оплати. Для таких задач можна застосовувати як реляційні системи керування базами даних, так і документно-орієнтовані рішення. Реляційні бази є доцільними там, де домінують суворо формалізовані зв’язки і складні транзакційні сценарії. Документно-орієнтовані бази даних є зручнішими у випадках, коли структура сутностей повинна залишатися більш гнучкою, а дані часто подаються у вигляді вкладених об’єктів. Для BiTZone виправданим є використання MongoDB, оскільки в межах такого магазину важливо зручно працювати з JSON-подібними структурами товарів, замовлень та користувацьких записів [19].

Для взаємодії з базою даних потрібен також засіб моделювання сутностей на рівні програмного коду. У JavaScript-проєктах цю роль часто виконує Mongoose, який дозволяє визначати схеми документів, перевіряти структуру даних, працювати зі зв’язками між сутностями та зберігати логіку моделі без переходу до ручної побудови кожного запиту. Для магазину, у якому одночасно існують товари, користувачі, замовлення та відгуки, така модель є зручною, оскільки спрощує підтримку серверної частини й робить структуру даних більш передбачуваною [20].

Окремий напрям становлять інтеграційні технології. Сучасний інтернет-магазин не існує відокремлено: він взаємодіє з платіжними системами, службами доставки, зовнішніми джерелами товарної інформації та сервісами

синхронізації. Через це серверна частина повинна підтримувати роботу з REST API, обробку зовнішніх відповідей, збереження статусів та отримання webhook-повідомлень. Для BiTZone це має безпосереднє значення, оскільки логіка магазину охоплює доставку через Nova Poshta, оплату через Monobank і синхронізацію товарів та категорій із RoApp. Тому аналіз технологій не може обмежуватися лише внутрішніми модулями — у нього обов'язково повинні входити засоби безпечної та стабільної роботи із зовнішніми сервісами.

Не менш важливими є засоби, що забезпечують безпеку та якість роботи системи. Для інтернет-магазину критичне значення мають автентифікація користувачів, перевірка вхідних даних, захист API, обмеження підозрілої активності, коректна робота із зображеннями товарів і, за потреби, кешування часто використовуваних даних. У проєкті типу BiTZone такі засоби не можна вважати додатковими, оскільки вони впливають і на безпеку персональних даних, і на швидкодію, і на загальне сприйняття системи користувачем.

Отже, аналіз засобів і технологій показує, що для створення спеціалізованого web-сайту інтернет-магазину відеогрових консолей потрібне не окреме «модне» рішення, а узгоджене поєднання фронтенд-, бекенд-, базових і інтеграційних інструментів. Для BiTZone найбільш доцільним є використання fullstack-підходу, у якому клієнтська частина будується на React, серверна логіка — на Node.js та Express, робота з даними — на MongoDB і Mongoose, а зовнішня взаємодія — через API служб доставки, оплати та синхронізації товарної інформації. Саме така технологічна основа дає змогу перейти від загального аналізу до безпосереднього визначення об'єкта, предмета, мети та завдань бакалаврської роботи.

1.6 Об'єкт, предмет, мета та завдання бакалаврської роботи

Після розгляду особливостей предметної галузі, логіки повного циклу замовлення, наявних аналогів та технологічних підходів можна перейти до формулювання основних положень бакалаврської роботи.

Об’єктом дослідження є процес онлайн-продажу відеогрових консолей і супутніх товарів.

Предметом дослідження є програмні засоби автоматизації керування каталогом товарів, замовленнями, доставкою, оплатою та взаємодією користувача з web-сайтом інтернет-магазину.

Метою бакалаврської роботи є підвищення зручності та ефективності процесу онлайн-замовлення відеогрових консолей і супутніх товарів шляхом розробки web-сайту інтернет-магазину з повним циклом функціонування та інтеграцією зовнішніх сервісів.

Для досягнення поставленої мети в роботі необхідно розв’язати такі завдання:

1. дослідити особливості предметної галузі онлайн-продажу відеогрових консолей і супутніх товарів;

2. проаналізувати існуючі програмні рішення для організації електронної комерції та визначити їх сильні і слабкі сторони;

3. обґрунтувати вибір засобів і технологій, доцільних для реалізації системи;

4. сформувати функціональні та нефункціональні вимоги до web-сайту;

5. спроєктувати архітектуру системи, структуру даних та основні сценарії взаємодії користувача із сайтом;

6. реалізувати клієнтську та серверну частини web-сайту;

7. забезпечити інтеграцію системи із зовнішніми сервісами доставки, оплати та синхронізації товарних даних;

8. провести тестування основної функціональності розробленого web-сайту та оцінити його відповідність поставленим вимогам.

Таким чином, бакалаврська робота спрямована на створення спеціалізованого програмного рішення, яке враховує особливості продажу відеогрових консолей, підтримує повний користувацький маршрут від вибору товару до завершення замовлення та може бути використане як основа для практичної реалізації інтернет-магазину «BiTZone».

Висновки до розділу 1.

У першому розділі досліджено предметну галузь онлайн-продажу відеогрових консолей і супутніх товарів та встановлено, що для такого типу інтернет-магазину важливим є не лише відображення асортименту, а й підтримка повного користувацького маршруту: пошук товару, перегляд характеристик, формування кошика, оформлення замовлення, вибір доставки, оплата та подальший перегляд інформації в особистому кабінеті.

Також у розділі проаналізовано існуючі рішення електронної комерції та визначено, що їхніми перевагами є наявність готових засобів для каталогу, кошика, оформлення замовлення й адміністрування. Водночас їхніми обмеженнями є складність адаптації під спеціалізований бізнес-процес, залежність від сторонніх модулів або надмірна складність для вузькоорієнтованого магазину. На основі цього обґрунтовано доцільність розробки власного web-сайту BiTZone та вибрано технологічну основу для подальшого проєктування і реалізації системи.

2 ПРОЄКТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ ВІДЕОГРОВИХ КОНСОЛЕЙ BITZONE

Після визначення особливостей предметної галузі, аналізу наявних рішень та формулювання мети роботи наступним етапом є проектування майбутньої системи. Саме на цьому рівні задається внутрішня логіка web-сайту, визначається склад основних модулів, спосіб їх взаємодії, структура даних і загальна організація користувацького маршруту. Для інтернет-магазину відеогрових консолей це особливо важливо, оскільки система повинна поєднати зручний каталог, інформативну картку товару, кошик, особистий кабінет, оформлення замовлення, доставку, оплату та взаємодію із зовнішніми сервісами в межах одного узгодженого рішення.

Проектування в даній роботі розглядається не як формальна схема побудови сторінок, а як основа для подальшої реалізації повного циклу функціонування магазину. На цьому етапі необхідно визначити функціональні та нефункціональні вимоги до системи, описати ролі користувачів, зафіксувати основні сценарії взаємодії, обрати підхід до побудови архітектури, а також спроектувати структуру даних, від якої залежить робота каталогу, замовлень, облікового запису користувача та інших модулів. Окреме значення має проектування інтерфейсу, оскільки саме через нього користувач послідовно проходить шлях від вибору товару до завершення покупки.

Для web-сайту «BiTZone» проектування повинно враховувати не тільки типові задачі електронної комерції, а й специфіку спеціалізованого магазину. У системі мають бути передбачені зручні засоби пошуку товарів, логічно вибудувана структура каталогу, цілісна робота кошика і списку бажань, коректне оформлення замовлення, а також надійний обмін даними із сервісами доставки, оплати та зовнішньою системою синхронізації товарних позицій. Саме тому у цьому розділі основну увагу буде зосереджено на вимогах до

системи, архітектурних рішеннях, моделі даних та проектуванні користувацького інтерфейсу.

2.1 Формування функціональних і нефункціональних вимог до системи

Побудова web-сайту інтернет-магазину починається не з вибору кольорової схеми чи розміщення кнопок, а з чіткого визначення того, що саме повинна робити система і якими характеристиками вона має володіти під час реальної експлуатації. Для проєкту BiTZone це особливо важливо, оскільки сайт повинен підтримувати повний цикл замовлення, а отже будь-яка помилка або прогалина у вимогах неминуче впливатиме на каталог, кошик, оформлення покупки, доставку чи оплату. Саме тому на етапі проєктування необхідно окремо визначити функціональні та нефункціональні вимоги до системи.

Функціональні вимоги описують ті можливості, які безпосередньо повинен отримати користувач або адміністратор даних у межах роботи з web-сайтом. Нефункціональні вимоги визначають не стільки сам перелік дій, скільки умови, за яких система має працювати: швидкодію, надійність, безпеку, зручність використання та адаптивність. Для інтернет-магазину відеогрових консолей ці дві групи вимог тісно пов’язані між собою, оскільки зручність користування безпосередньо впливає на завершення покупки, а технічна стабільність системи — на довіру до сервісу [10].

До основних функціональних вимог системи BiTZone доцільно віднести такі.

1. Робота з каталогом товарів. Система повинна надавати користувачу доступ до структурованого каталогу відеогрових консолей, ігор, аксесуарів та супутніх товарів. Каталог має підтримувати поділ за категоріями, відображення короткої інформації про товар, ціну, зображення та статус доступності.

2. Пошук, фільтрація та сортування. Користувач повинен мати можливість знаходити товари за назвою, категорією та іншими доступними

параметрами. Система також повинна підтримувати сортування результатів за ціною, новизною або іншими релевантними ознаками, щоб спростити вибір серед кількох подібних позицій.

3. Перегляд картки товару. Для кожної позиції система повинна формувати окрему сторінку з детальним описом товару, характеристиками, зображеннями, ціною, інформацією про наявність та можливістю додавання до кошика або списку бажань.

4. Робота зі списком бажань. Користувач повинен мати можливість зберігати окремі товари у списку бажань для подальшого перегляду. Це важливо для предметної галузі, у якій покупець часто не приймає рішення одразу, а повертається до вибору після порівняння кількох варіантів.

5. Робота з кошиком. Система повинна забезпечувати додавання товарів до кошика, зміну кількості позицій, видалення товарів, автоматичний перерахунок вартості та збереження поточного стану покупки в межах користувацької сесії.

6. Реєстрація та автентифікація користувача. Сайт повинен підтримувати створення облікового запису, вхід до системи, збереження персональних даних користувача та використання цих даних під час оформлення замовлення.

7. Особистий кабінет. Авторизований користувач повинен мати доступ до сторінки особистого кабінету, де зберігаються його основні дані, історія замовлень та результати попередніх дій у системі.

8. Оформлення замовлення. Система повинна дозволяти користувачу сформувати замовлення на основі вмісту кошика, вказати контактні дані, обрати місто, відділення або інші параметри доставки, перевірити склад замовлення і підтвердити його створення.

9. Інтеграція з сервісом доставки. Web-сайт повинен забезпечувати отримання актуальних даних про населені пункти та точки видачі через зовнішній сервіс доставки. Для BiTZone це означає підтримку взаємодії з Nova Poshta на етапі оформлення замовлення.

10. Інтеграція з платіжним сервісом. Система повинна підтримувати ініціацію онлайн-оплати та подальшу обробку результату такої операції. Для даного проєкту це передбачає інтеграцію з Monobank.

11. Робота із замовленнями. Після створення замовлення система повинна зберігати інформацію про нього в базі даних, пов’язувати замовлення з конкретним користувачем і надавати можливість перегляду цього замовлення в особистому кабінеті.

12. Підтримка відгуків. Користувач повинен мати можливість залишати відгуки до товарів, а система — зберігати та відображати їх у картці товару.

13. Синхронізація товарних даних. Для підтримання актуальності каталогу система повинна взаємодіяти із зовнішнім джерелом товарної інформації та категорій. У випадку BiTZone це означає обмін даними з RoApp і підтримку webhook-подій.

Наведені функціональні вимоги описують ядро системи і формують той набір можливостей, без якого web-сайт не зможе підтримувати повний цикл роботи інтернет-магазину. Проте для реальної експлуатації цього недостатньо. Не менш важливо визначити, якими характеристиками повинна володіти система під час використання.

До основних нефункціональних вимог для BiTZone доцільно віднести такі положення.

1. Швидкодія. Типові сторінки каталогу, картки товару, кошика та особистого кабінету повинні завантажуватися не довше ніж за 3 секунди за стандартного користувацького підключення. Запити на пошук, фільтрацію та сортування повинні повертати результат не довше ніж за 2 секунди за звичайного навантаження.

2. Адаптивність інтерфейсу. Система повинна коректно працювати на персональних комп’ютерах, планшетах і смартфонах. Інтерфейс має залишатися придатним до використання на екранах різної ширини без втрати основної функціональності.

3. Зручність використання. Основні дії користувача — перегляд каталогу, перехід до товару, додавання в кошик, перехід до оформлення та підтвердження замовлення — повинні виконуватися без зайвих переходів і без складних для сприйняття форм. Логіка розміщення елементів інтерфейсу має бути послідовною на всіх основних сторінках.

4. Надійність збереження даних. Відомості про користувачів, товари, замовлення та відгуки повинні зберігатися в базі даних таким чином, щоб після завершення операції вони були доступні для подальшого перегляду і не втрачалися під час переходу між етапами сценарію.

5. Безпека. Система повинна забезпечувати захист облікових записів користувачів, безпечну передачу токенів автентифікації, валідацію вхідних даних та базовий захист серверної частини від несанкціонованих звернень. Паролі користувачів не повинні зберігатися у відкритому вигляді.

6. Сумісність. Web-сайт повинен коректно працювати в актуальних версіях поширених браузерів, зокрема Google Chrome, Mozilla Firefox, Microsoft Edge та Safari.

7. Масштабованість. Архітектура системи повинна дозволяти розширення каталогу, додавання нових категорій, властивостей товарів, інтеграцій або сервісних модулів без повної перебудови вже реалізованої логіки.

8. Підтримуваність. Структура клієнтської та серверної частин повинна залишатися зрозумілою для подальшого супроводження, а програмні модулі — бути достатньо відокремленими для внесення змін без критичного впливу на інші частини системи.

Таким чином, сформовані вимоги задають рамки для подальшого проєктування web-сайту BiTZone. Функціональні вимоги визначають, які саме можливості повинні бути реалізовані в системі, а нефункціональні — якими мають бути умови їхньої роботи. Саме на основі цих вимог надалі доцільно будувати сценарії взаємодії користувача із системою, архітектуру застосунку, структуру даних та інтерфейсні рішення.

2.2 Моделювання вимог користувачів і сценаріїв взаємодії з системою

Після визначення загальних вимог до web-сайту необхідно уточнити, хто саме буде взаємодіяти із системою і які дії для цих користувачів є основними. Для інтернет-магазину це важливо не менше, ніж для будь-якого іншого програмного продукту, оскільки навіть за наявності повного набору функцій сайт може залишатися незручним, якщо логіка переходу між етапами покупки побудована непослідовно. Саме тому на етапі проектування вимоги доцільно розглядати через ролі користувачів і через конкретні сценарії їхньої взаємодії з системою [10, 11].

Для web-сайту BiTZone можна виділити кілька основних учасників взаємодії. Першим є неавторизований користувач, який заходить на сайт для перегляду асортименту, пошуку товарів, ознайомлення з картками позицій, додавання товарів до кошика або списку бажань. Другим є авторизований користувач, для якого, крім базових можливостей перегляду каталогу, відкривається оформлення замовлення, перегляд історії покупок, доступ до особистого кабінету та робота з власними даними. Окрему роль у роботі системи відіграють зовнішні сервіси, з якими сайт взаємодіє під час виконання частини сценаріїв: служба доставки Nova Poshta, платіжний сервіс Monobank та зовнішня система синхронізації товарних даних RoApp. Хоча ці сервіси не є користувачами в звичному розумінні, з погляду моделювання вимог вони виступають повноправними учасниками процесу, оскільки без них частина функціональності не може бути завершена.

Для наочного подання таких зв’язків у роботі доцільно використати діаграму варіантів використання, на якій відображаються основні ролі та дії, доступні кожній з них. На цій діаграмі для неавторизованого користувача доцільно передбачити перегляд каталогу, пошук і фільтрацію товарів, відкриття сторінки товару, додавання позиції до кошика і перехід до реєстрації або входу. Для авторизованого користувача — оформлення замовлення, перегляд

особистого кабінету, історії замовлень, роботу з відгуками та повторне звернення до збережених позицій. Для зовнішніх сервісів — отримання даних про доставку, ініціацію та підтвердження оплати, а також синхронізацію товарної інформації. Схематично ролі та основні варіанти використання BiTZone наведено на рисунку 2.1.

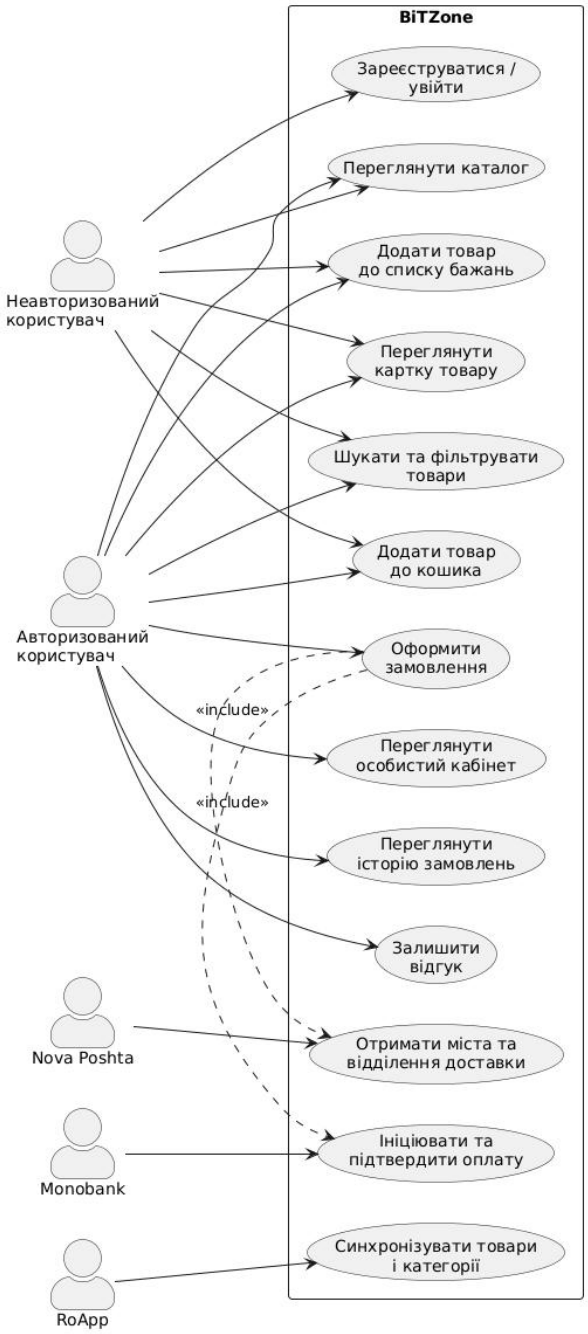


Рис. 2.1 Діаграма варіантів використання web-сайту BiTZone

Ключовим для BiTZone є сценарій повного циклу замовлення. Саме він найбільш повно відображає вимоги до системи й дозволяє оцінити, наскільки окремі модулі узгоджені між собою. У типовому варіанті користувач відкриває каталог, знаходить потрібний товар через категорії або пошук, переходить до картки товару, аналізує опис і характеристики, після чого додає позицію до кошика. Далі він переходить до оформлення покупки, за потреби проходить автентифікацію, вказує контактні дані, обирає місто та відділення доставки, перевіряє склад замовлення і переходить до оплати. Після взаємодії з платіжним сервісом система зберігає результат операції, а замовлення відображається в особистому кабінеті користувача. У межах цього сценарію задіяні майже всі ключові модулі сайту, тому саме він є центральним для подальшого проєктування архітектури системи.

Не менш важливим є сценарій відкладеної покупки. У практичному використанні користувач не завжди переходить до оформлення замовлення одразу після перегляду товару. Часто він відкладає товар у список бажань, повертається до нього пізніше, порівнює його з іншими позиціями і лише після цього приймає остаточне рішення. Для предметної галузі відеогрових консолей така поведінка є типовою, оскільки покупка подібної техніки рідко має імпульсивний характер. Тому при моделюванні вимог важливо врахувати не лише прямий сценарій «вибрав — купив», а й сценарій відкладеної взаємодії з товаром.

Окремо слід розглянути сценарій роботи з особистим кабінетом. Для авторизованого користувача кабінет не є другорядним елементом системи, а виконує роль точки повернення до попередніх дій. Через нього користувач переглядає замовлення, контролює власні дані, може знову звернутися до потрібних товарів і, за необхідності, залишити відгук. Якщо цей сценарій не враховано на етапі моделювання вимог, система фактично перетворюється на сайт одноразового продажу, що не відповідає ідеї повного циклу функціонування.

Ще один важливий напрям моделювання вимог пов’язаний не безпосередньо з користувачем, а з поведінкою системи відносно зовнішніх джерел даних. Для BiTZone це стосується синхронізації товарів і категорій із зовнішньою системою, а також обробки webhook-подій. У реальній роботі магазину ці процеси не видно покупцеві безпосередньо, але саме вони впливають на актуальність каталогу, коректність цін, наявність товарів і узгодженість даних між модулями. Тому при моделюванні вимог необхідно враховувати не лише класичні користувацькі сценарії, а й службові сценарії обміну даними.

Послідовність основних дій користувача під час оформлення замовлення доцільно подати окремою діаграмою діяльності. Вона дозволяє показати логіку переходу між етапами: від вибору товару до підтвердження оплати та відображення замовлення в особистому кабінеті. Така схема є важливою не тільки для опису користувацького маршруту, а й для подальшого проектування архітектури системи.

Таким чином, моделювання вимог для web-сайту BiTZone повинно охоплювати як мінімум три взаємопов’язані площини: ролі користувачів, основні сценарії покупки та службові сценарії інтеграції із зовнішніми сервісами. Такий підхід дає змогу перейти від загального переліку функцій до чіткішого уявлення про те, як саме система повинна поводитися в реальному використанні. Саме на цій основі надалі доцільно будувати архітектурну модель системи, структуру даних та деталізацію інтерфейсних рішень.

2.3 Проєктування архітектури системи

Під час проєктування web-сайту «BiTZone» важливо було визначити не лише перелік функцій, які повинна підтримувати система, а й спосіб організації цих функцій у межах єдиного програмного рішення. Для інтернет-магазину з повним циклом функціонування архітектура має забезпечувати узгоджену роботу каталогу, сторінок товарів, кошика, списку бажань, особистого кабінету,

оформлення замовлення, доставки, оплати та синхронізації товарних даних. Якщо ці частини проектуються без єдиної логіки, система швидко втрачає цілісність, а окремі модулі починають працювати як ізольовані елементи.

Для «BiTZone» доцільним є використання клієнт-серверної архітектури, у межах якої клієнтська частина відповідає за взаємодію з користувачем, а серверна — за обробку запитів, бізнес-логіку, збереження даних та інтеграцію із зовнішніми сервісами. Такий підхід дозволяє відокремити інтерфейсний рівень від рівня обробки даних і зробити систему більш зрозумілою для подальшого розвитку. Для інтернет-магазину це особливо важливо, оскільки користувач працює з динамічними сторінками, а сервер повинен паралельно обробляти товари, користувачів, замовлення, платежі, доставку та службові події [8, 9, 22].

На клієнтському рівні система повинна забезпечувати відображення каталогу, пошук, фільтрацію, відкриття картки товару, роботу з кошиком, списком бажань, реєстрацію, вхід, особистий кабінет та перехід між етапами оформлення замовлення. Цей рівень не повинен містити критичну бізнес-логіку, пов’язану із збереженням замовлень або зовнішніми інтеграціями. Його основним завданням є отримання даних із сервера, показ цих даних у зручному вигляді та передача назад результатів користувацьких дій.

Серверний рівень, у свою чергу, виступає центральною частиною системи. Саме тут обробляються запити, пов’язані з отриманням товарів, авторизацією, оформленням замовлення, збереженням відгуків, формуванням історії покупок та обміном даними із зовнішніми сервісами. Такий підхід дозволяє зосередити бізнес-логіку в одному місці, уникнути дублювання функцій і підтримувати цілісність роботи системи. Для «BiTZone» це означає, що всі критичні дії — створення замовлення, перевірка користувача, отримання параметрів доставки, ініціація оплати, синхронізація товарних даних — повинні проходити саме через серверну частину.

Окремим архітектурним компонентом є база даних, у якій зберігаються відомості про товари, категорії, користувачів, замовлення, відгуки та інші

сутності, необхідні для роботи магазину. З точки зору проектування важливо, щоб доступ до цих даних був організований через чітко визначені моделі та програмні рівні, а не хаотично через окремі запити з різних частин системи. Це спрощує супровід проєкту, дозволяє контролювати зв’язки між сутностями та робить систему більш передбачуваною при розширенні функціоналу.

В архітектурі «BiTZone» також необхідно врахувати зовнішні сервіси, без яких повний цикл роботи інтернет-магазину не може бути завершений. До них належать служба доставки Nova Poshta, платіжний сервіс Monobank та зовнішня система синхронізації товарів і категорій RoApp. Усі ці елементи не повинні бути включені в клієнтську частину безпосередньо. Взаємодія з ними має проходити через сервер, який виступає посередником між внутрішньою логікою магазину та зовнішнім цифровим середовищем. Саме такий підхід зменшує ризики, пов’язані з безпекою, спрощує контроль над даними та дає змогу коректно обробляти результати зовнішніх операцій [12-14].

На рисунку 2.3 доцільно подати загальну архітектурну діаграму web-сайту «BiTZone», яка відображає взаємодію між клієнтською частиною, серверною частиною, базою даних та зовнішніми сервісами.

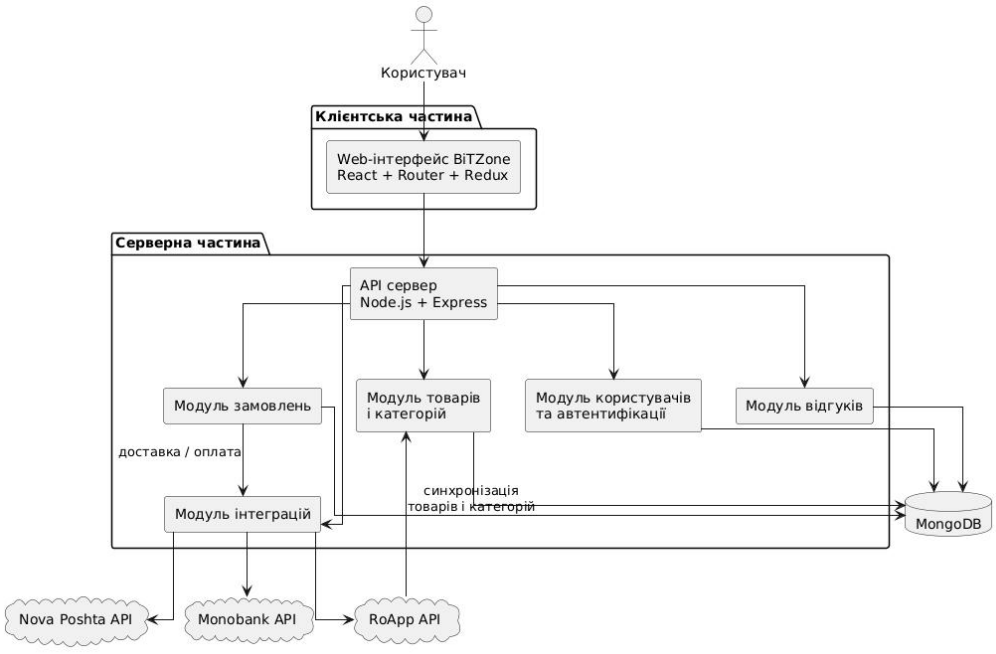


Рис 2.3 Архітектурна діаграма web-сайту BiTZone

Крім загальної архітектурної схеми, для даної системи важливим є і фізичний погляд на розгортання її компонентів. У реальній роботі користувач відкриває сайт через браузер, клієнтська частина взаємодіє із серверним API, сервер звертається до бази даних, а також обмінюється запитами з зовнішніми сервісами доставки, оплати та синхронізації. Така схема розгортання дає змогу побачити, де саме розміщуються основні частини системи і як між ними відбувається обмін даними.

Для «BiTZone» діаграма розгортання є корисною ще й тому, що вона показує систему не як абстрактний набір модулів, а як реально розподілене рішення, у якому браузер користувача, сервер застосунку, сховище даних і зовнішні сервіси взаємодіють у межах одного бізнес-процесу. Це особливо важливо для магазину з повним циклом замовлення, оскільки під час покупки система залежить не лише від власних внутрішніх компонентів, а й від стабільного зв’язку з доставкою та оплатою.

На рисунку 2.4 доцільно подати діаграму розгортання web-сайту «BiTZone», яка відобразить взаємодію між браузером користувача, клієнтською частиною, сервером застосунку, базою даних та зовнішніми API.

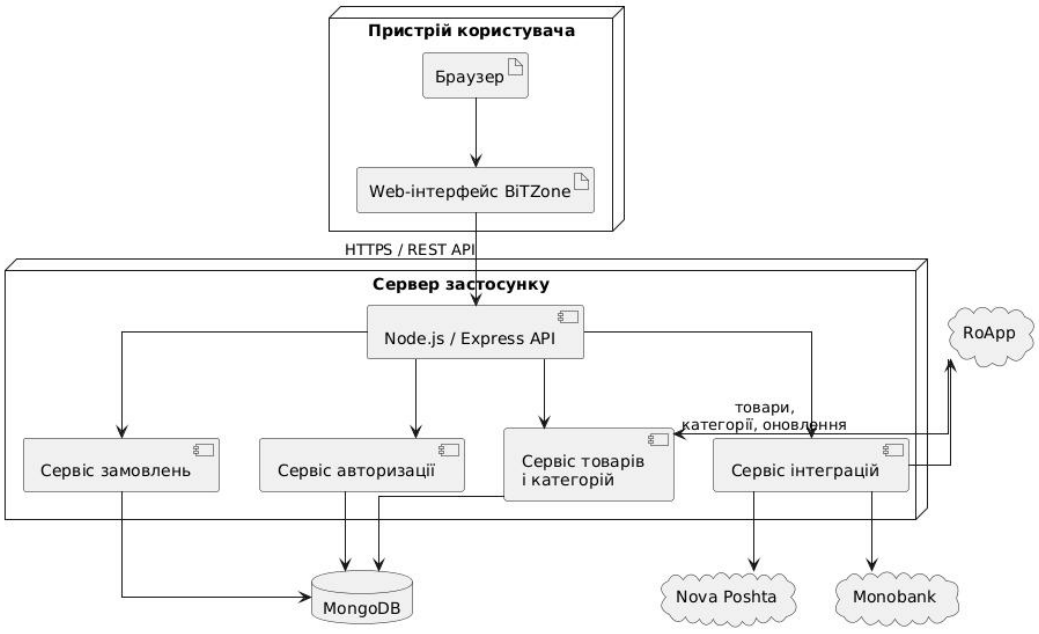


Рис 2.4 Діаграма розгортання web-сайту BiTZone

Отже, під час проектування архітектури системи «BiTZone» доцільно було обрати клієнт-серверний підхід із чітким розподілом ролей між інтерфейсом, серверною логікою, базою даних і зовнішніми сервісами. Таке рішення відповідає особливостям інтернет-магазину відеогрових консолей, підтримує повний цикл

2.4 Проєктування інформаційного забезпечення та структури бази даних

Для інтернет-магазину, у якому користувач проходить повний шлях від вибору товару до оформлення й перегляду замовлення, структура даних має принципове значення. Якщо зв’язки між товарами, категоріями, користувачами, замовленнями та відгуками організовані невдало, це одразу позначається на роботі каталогу, кошика, особистого кабінету та інших модулів. Саме тому на етапі проектування потрібно було визначити не лише перелік сутностей, а й логіку їх взаємодії між собою.

У межах розроблюваної системи інформаційне забезпечення охоплює кілька основних груп даних. До першої належать відомості про товари та категорії, які формують каталог і забезпечують навігацію по асортименту. Друга група пов’язана з користувачами: реєстраційними даними, обліковими записами, історією покупок і результатами взаємодії з магазином. Третя група охоплює замовлення, включаючи склад покупки, дані доставки, оплату та поточний статус. Окремо слід враховувати інформацію, яка надходить із зовнішніх джерел, зокрема під час синхронізації товарних позицій.

Однією з центральних сутностей є товар. Саме з ним пов’язана значна частина функціональності системи: відображення каталогу, пошук, фільтрація, сторінка товару, кошик, список бажань та відгуки. Для кожної товарної позиції необхідно зберігати назву, короткий і повний опис, вартість, набір зображень, категорію, технічні характеристики, ознаки наявності, а також службові дані,

потрібні для синхронізації із зовнішнім джерелом. Через те, що різні типи товарів можуть мати різну деталізацію параметрів, модель зберігання повинна залишатися достатньо гнучкою.

Наступною важливою сутністю є категорія. Вона потрібна не лише для формального групування товарів, а й для побудови логічної структури каталогу. Саме через категорії користувач переходить до потрібного сегмента асортименту: консолей, ігор, контролерів, гарнітур, кабелів, док-станцій чи інших аксесуарів. Категорія повинна бути пов’язана з товарами таким чином, щоб система могла підтримувати навігацію, формування сторінок каталогу, фільтрацію та синхронізацію даних.

Окремий блок моделі даних стосується користувача. Для зареєстрованого покупця система повинна зберігати ім’я, електронну адресу, пароль у захищеному вигляді, а також додаткові дані, що використовуються в особистому кабінеті та під час оформлення замовлення. На цьому рівні важливо забезпечити не лише доступ до функціональності сайту, а й безпеку збережених відомостей. Тому дані користувача мають бути логічно відокремлені від інших сутностей, але при цьому пов’язані із замовленнями, відгуками та іншими діями, які виконуються після входу в систему.

Найбільш значущою транзакційною сутністю виступає замовлення. Воно поєднує результати роботи відразу кількох модулів: каталогу, кошика, авторизації, доставки й оплати. У структурі замовлення мають зберігатися товари, обрані користувачем, їх кількість, сума покупки, контактні дані, спосіб доставки, параметри отримання, інформація про оплату та поточний статус. Інакше кажучи, замовлення є точкою, у якій з’єднуються всі основні сценарії роботи магазину. Саме тому його структура повинна бути цілісною і достатньо детальною для подальшого відображення в особистому кабінеті та використання в бізнес-логіці.

Ще однією важливою сутністю є відгук. Він прив’язується до конкретного товару і, як правило, пов’язується з користувачем, який його залишив. Для системи це означає необхідність зберігати текст відгуку, оцінку

або інші пов’язані поля, а також забезпечувати зв’язок між сторінкою товару та автором відгуку. Попри те, що цей модуль не є центральним для самого факту продажу, він має значення для довіри до магазину та змістовного наповнення карток товарів.

Для зберігання таких даних доцільно використовувати MongoDB, оскільки документно-орієнтований підхід дає змогу гнучко працювати з товарними характеристиками, замовленнями та користувацькими записами. У межах проєкту це дозволяє уникнути надмірно жорсткої структури там, де різні сутності можуть мати неоднаковий склад полів. Для роботи з моделями доцільно застосовувати Mongoose, що спрощує опис сутностей, валідацію та побудову зв’язків на рівні серверної логіки [19, 20].

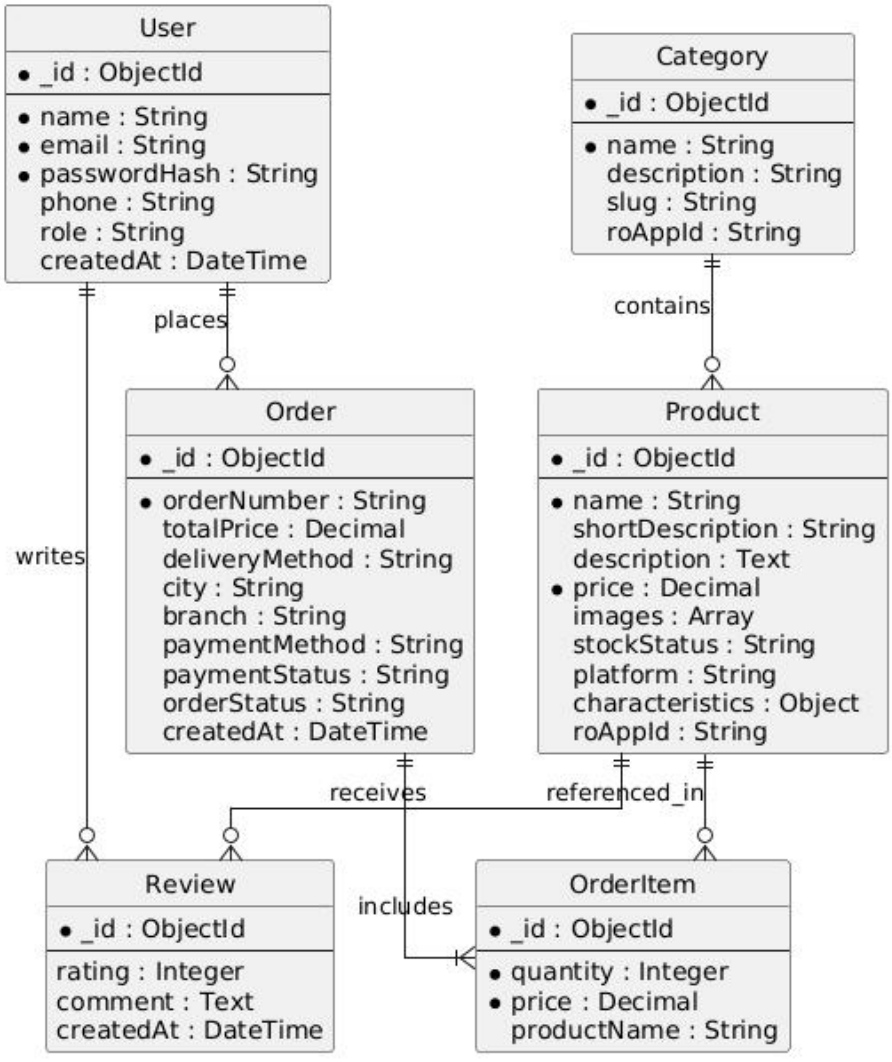


Рис 2.5 ER-діаграма бази даних інтернет-магазину

Якщо розглядати зв’язки між сутностями, то одна категорія може містити багато товарів, тоді як кожний товар належить до певної категорії. Один користувач може мати багато замовлень, але кожне замовлення пов’язується лише з одним користувачем. Замовлення, у свою чергу, включає одну або кілька товарних позицій. Відгук також має двосторонній зв’язок: він стосується конкретного товару і водночас належить користувачу, який його залишив. Саме така схема дає змогу забезпечити логічну цілісність збережених даних і надалі коректно відображати їх у різних частинах системи.

Окрім ER-моделі, доцільно окремо показати спрощену структуру основних інформаційних блоків, які використовуються під час роботи каталогу, оформлення замовлення та особистого кабінету. Така схема дає змогу наочно відобразити, як саме дані про товари, користувача і замовлення використовуються в межах одного сценарію.

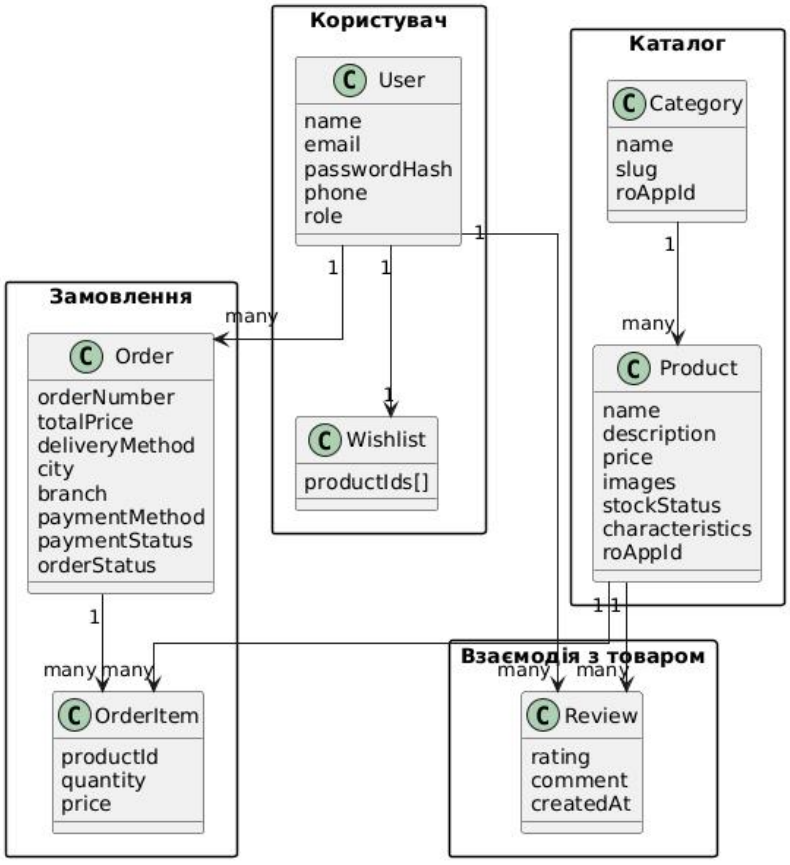


Рис 2.6 Структура основних даних системи

При проектуванні інформаційного забезпечення важливо було врахувати і перспективу подальшого розвитку системи. Структура бази даних не повинна бути надто жорсткою, оскільки в майбутньому можуть з’являтися нові типи товарів, додаткові характеристики, нові поля для доставки чи оплати, а також нові інтеграції. Саме тому модель даних повинна підтримувати розширення без повного перегляду всієї вже реалізованої логіки.

Отже, на етапі проектування було визначено основні сутності системи, їх призначення та зв’язки між ними. Такий підхід дозволяє створити цілісну основу для подальшої реалізації каталогу, особистого кабінету, кошика, оформлення замовлення, відгуків і взаємодії із зовнішніми сервісами. Правильно побудована структура даних у даному випадку виступає не окремим технічним елементом, а фундаментом усієї системи.

2.5 Проєктування інтерфейсу користувача

Для інтернет-магазину з повним циклом замовлення інтерфейс не можна розглядати лише як візуальне оформлення сторінок. У цьому випадку він виконує значно важливішу функцію: допомагає користувачеві послідовно пройти всі етапи взаємодії із системою без зайвих дій, непорозумінь і повернень до попередніх кроків. Якщо структура сторінок побудована нелогічно, навіть технічно справна система сприймається як незручна. Саме тому під час проектування особливу увагу потрібно приділити не декоративним елементам, а зрозумілості навігації, послідовності користувацького маршруту та передбачуваній поведінці основних елементів інтерфейсу [23, 24].

У межах даного проєкту інтерфейс мав забезпечити роботу з кількома типами сторінок, які відрізняються за призначенням, але повинні залишатися частинами однієї цілісної системи. До них належать головна сторінка, сторінка каталогу, картка товару, кошик, сторінки авторизації та реєстрації, сторінка оформлення замовлення, особистий кабінет, сторінка деталей замовлення, а також допоміжні інформаційні сторінки. Користувач не повинен щоразу

“вчитись заново”, як працює сайт після переходу на іншу сторінку. Тому базові принципи навігації, розміщення головних елементів і спосіб подання ключової інформації мають бути однаковими в межах усієї системи.

Однією з основних вимог до інтерфейсу є швидке орієнтування в каталозі. Для магазину відеогрових консолей це має особливе значення, оскільки користувач зазвичай працює не з кількома випадковими товарами, а з певною групою схожих позицій. Через це сторінка каталогу повинна відображати товари так, щоб між ними було легко перемикатися, а також містити зручні засоби пошуку, фільтрації та сортування. У користувача має бути можливість без зайвих переходів звузити вибір, залишивши тільки ті товари, які відповідають його очікуванням. У проектуванні це означає, що елементи навігації не повинні конкурувати між собою, а сторінка не має бути перевантажена другорядними деталями.

Картка товару повинна проектуватися інакше, ніж сторінка каталогу. Якщо каталог допомагає знайти товар, то картка товару має допомогти прийняти рішення про покупку. Тому в центрі уваги тут знаходяться назва, зображення, вартість, ключові характеристики, опис, інформація про наявність та основні дії користувача: додавання до кошика або списку бажань. Для даної предметної галузі важливо, щоб сторінка не приховувала суттєві параметри консолі чи аксесуара, але й не змушувала користувача перечитувати великий масив другорядного тексту. Іншими словами, проектування картки товару має базуватися на балансі між інформативністю і простотою сприйняття.

Не менш важливим елементом є кошик. У багатьох випадках саме на цьому етапі користувач остаточно оцінює, наскільки зручною є система. Якщо кошик показує лише перелік позицій без можливості швидко змінити кількість або видалити зайвий товар, це ускладнює подальший перехід до оформлення замовлення. Тому під час проектування потрібно передбачити, щоб кошик не був пасивним списком, а залишався керованим інтерфейсним блоком, у межах якого користувач одразу бачить основні параметри покупки, проміжну суму та логічний перехід до наступного кроку.

Оформлення замовлення є найчутливішим етапом взаємодії із системою, тому інтерфейс цієї частини повинен бути максимально прямим і зрозумілим. На цьому етапі користувач не повинен шукати, куди саме вводити дані, в якій послідовності заповнювати поля та що відбудеться після підтвердження замовлення. Для цього сторінка оформлення повинна проектуватися за принципом послідовного сценарію: спочатку дані користувача, потім параметри доставки, далі перевірка складу замовлення і перехід до оплати. Саме така структура дозволяє зменшити кількість помилок і знизити ризик того, що користувач покине сайт до завершення покупки.

Окремо потрібно враховувати особистий кабінет. Його роль не зводиться лише до зберігання імені або електронної адреси користувача. Для системи з повним циклом замовлення кабінет є точкою повернення до всіх важливих дій після покупки. Саме тут повинні відображатися історія замовлень, коротка інформація про попередні покупки, доступ до деталей окремого замовлення та, за потреби, зв’язок з іншими модулями системи. Через це інтерфейс кабінету повинен бути стриманим, упорядкованим і орієнтованим на швидкий доступ до потрібної інформації, а не на декоративне ускладнення.

Під час проектування інтерфейсу також важливо було врахувати адаптивність. Частина користувачів буде працювати із сайтом з ноутбука або персонального комп’ютера, інші — зі смартфона. Через це сторінки повинні зберігати зрозумілу структуру на різних розмірах екрана. На практиці це означає, що головна навігація, блоки каталогу, картка товару, кошик та оформлення замовлення не повинні втрачати логіку після зміни ширини екрана. Адаптивність у даному випадку не є окремою “додатковою перевагою”, а належить до базових умов придатності інтерфейсу до реального використання.

Для наочного подання логіки побудови сторінок у роботі доцільно показати схему основних розділів і переходів між ними. Це дозволяє побачити систему не як набір незалежних екранів, а як впорядковану структуру, у якій кожна сторінка має власне призначення і водночас є частиною загального користувацького маршруту.

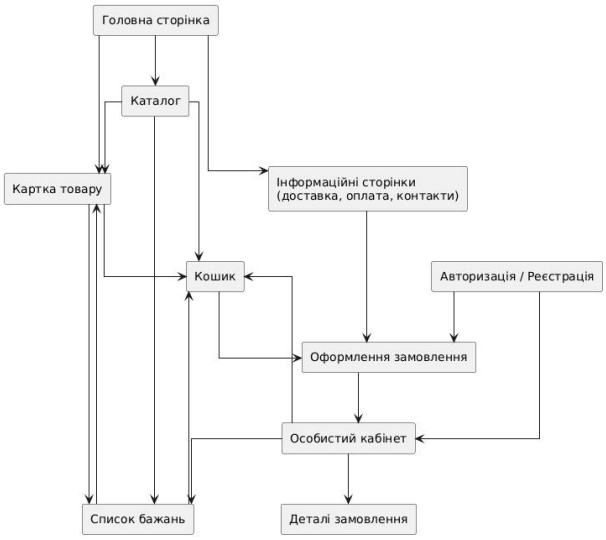


Рис 2.7 Схема структури основних сторінок системи

Отже, проектування інтерфейсу в даній роботі було спрямоване не лише на формування візуально впорядкованих сторінок, а передусім на побудову зрозумілого й послідовного способу взаємодії користувача із системою. Саме така логіка дозволяє поєднати каталог, картку товару, кошик, оформлення замовлення та особистий кабінет у межах одного узгодженого сценарію роботи магазину.

Висновки до розділу 2.

У другому розділі сформовано функціональні та нефункціональні вимоги до web-сайту BiTZone. До ключових функціональних вимог віднесено роботу з каталогом товарів, пошук, фільтрацію, сортування, картку товару, кошик, список бажань, реєстрацію, авторизацію, особистий кабінет, оформлення замовлення, відгуки, доставку, оплату та синхронізацію товарних даних. Серед основних нефункціональних вимог визначено швидкодію, адаптивність, зручність використання, безпеку, надійність збереження даних, сумісність, масштабованість і підтримуваність системи.

На основі сформованих вимог спроєктовано клієнт-серверну архітектуру web-сайту, сценарії взаємодії користувача, структуру даних і користувацький інтерфейс. Як основу інформаційного забезпечення передбачено використання документно-орієнтованої СУБД MongoDB, у якій зберігаються дані про користувачів, товари, категорії, замовлення та відгуки. Отримані проєктні рішення стали основою для подальшої програмної реалізації системи.

3 РЕАЛІЗАЦІЯ СИСТЕМИ

Реалізація інтернет-магазину такого типу не зводиться до створення окремих сторінок або API-запитів. Важливим є те, наскільки узгоджено між собою працюють клієнтська частина, серверна логіка, база даних і зовнішні сервіси. Якщо ці частини розроблені ізольовано, користувач може бачити працездатний інтерфейс, але при цьому стикатися з помилками під час оформлення покупки, втрати даних кошика, неправильним відображенням замовлень або неузгодженістю статусів доставки й оплати. Саме тому під час реалізації системи особливу увагу необхідно приділити не лише окремим програмним модулям, а й зв’язкам між ними.

У межах даного розділу розглядається практична реалізація клієнтської та серверної частин, інтеграція із зовнішніми сервісами, структура основних моделей і модулів, а також логіка функціонування ключових екранів. Це дає змогу показати, як саме спроєктоване рішення було втілене у вигляді цілісної системи, орієнтованої на повний цикл онлайн-замовлення відеогрових консолей і супутніх товарів.

3.1 Реалізація клієнтської частини web-сайту

Клієнтська частина системи реалізована як односторінковий web-застосунок, побудований на основі React. Такий підхід дозволяє організувати взаємодію користувача із сайтом без постійного повного перезавантаження сторінок, що є важливим для каталогу, кошика, особистого кабінету та оформлення замовлення. Для користувача це означає більш плавну навігацію між розділами, швидше оновлення інтерфейсу та зручніший перехід між етапами покупки [15].

Структура клієнтської частини побудована за компонентним принципом. Окремі функціональні блоки винесені в самостійні компоненти, що спрощує

повторне використання елементів інтерфейсу та підтримку коду. У складі застосунку реалізовано головну сторінку, каталог товарів, сторінку окремого товару, кошик, список бажань, сторінки входу та реєстрації, особистий кабінет, сторінку деталей замовлення, сторінку результату оплати, а також інформаційні розділи магазину. Такий поділ є доцільним, оскільки кожна сторінка виконує чітко визначену роль у загальному користувацькому маршруті.

Для організації переходів між сторінками використано React Router. Завдяки цьому навігація по сайту реалізована як система маршрутів, де кожний розділ має власну адресу і може бути відкритий окремо. Це важливо не лише з точки зору зручності користувача, а й для побудови передбачуваної логіки застосунку. Каталог, картка товару, кошик, кабінет користувача та інші сторінки не існують ізольовано, а поєднані в єдину структуру переходів.

Для роботи зі станом застосунку використано Redux Toolkit. Такий вибір є виправданим, оскільки в системі існує кілька типів даних, які повинні бути доступні в різних частинах інтерфейсу: вміст кошика, товари зі списку бажань, інформація про авторизованого користувача, дані про замовлення та окремі результати запитів до сервера. Централізоване керування станом дозволяє уникнути дублювання логіки в компонентах і робить поведінку інтерфейсу більш узгодженою [16].

Одним із найважливіших модулів клієнтської частини є каталог товарів. У ньому реалізовано відображення списку позицій, пошук, фільтрацію та сортування. Користувач може переглядати асортимент, звужувати результати за потрібними параметрами і переходити до детальнішого перегляду конкретного товару. Для завантаження даних використовується axios, через який клієнтська частина звертається до серверного API. Це дозволяє одержувати актуальні дані про товари без жорсткого вбудовування їх у фронтенд.

Сторінка товару реалізує інший рівень взаємодії. Якщо каталог потрібен для первинного пошуку, то картка товару вже орієнтована на прийняття рішення про покупку. У ній відображаються основні відомості про позицію: назва, вартість, опис, зображення, характеристики, а також елементи дії —

додавання до кошика і до списку бажань. У цій частині інтерфейсу також передбачено відображення відгуків, що робить сторінку більш змістовною для користувача.

Окремо реалізовано модулі кошика і списку бажань. Їх наявність є важливою для предметної галузі, оскільки покупка відеогрової консолі або комплекту супутніх товарів часто не є миттєвим рішенням. Користувач може відкласти певні позиції, повернутися до них пізніше, змінити кількість у кошику або видалити зайві товари. На рівні інтерфейсу це означає, що система повинна постійно синхронізувати стан цих модулів із загальною логікою застосунку та відображати актуальну суму й склад покупки.

Для роботи з обліковим записом користувача реалізовано сторінки реєстрації, входу та особистого кабінету. Після авторизації користувач отримує доступ до власних даних, історії замовлень та окремих функцій, недоступних для неавторизованого відвідувача. Така організація клієнтської частини дозволяє відокремити публічний режим перегляду магазину від персоналізованої взаємодії, яка починається після входу до системи.

Особлива увага була приділена інтерфейсу оформлення замовлення. На цій сторінці користувач послідовно переходить від перевірки вмісту кошика до введення контактних даних, вибору параметрів доставки та переходу до оплати. Саме в цій частині система повинна поводитися максимально передбачувано, оскільки будь-яка незрозуміла дія або порушення логіки може призвести до переривання покупки. Тому інтерфейс оформлення замовлення побудований як послідовний сценарій, де користувач крок за кроком заповнює необхідні дані та переходить до наступного етапу.

Для стилізації інтерфейсу використано styled-components, а для окремих візуальних переходів та динамічних ефектів — framer-motion. Це дало змогу не лише оформити сторінки в єдиному стилі, а й зробити інтерфейс більш цілісним візуально. Водночас декоративна частина не є основною метою реалізації: усі інтерфейсні рішення підпорядковані зручності навігації, читабельності контенту та логіці виконання користувацьких дій.

Таким чином, клієнтська частина системи реалізована як цілісний динамічний застосунок, у якому окремі сторінки та компоненти об’єднані спільною логікою маршрутизації, централізованим керуванням станом і взаємодією із серверним API. Саме така організація фронтенду дозволяє підтримувати повний користувацький маршрут — від перегляду каталогу до оформлення замовлення та перегляду його результату.

3.2 Реалізація серверної частини системи

Серверна частина забезпечує виконання тих операцій, які не можуть бути покладені на клієнтський інтерфейс. Саме тут відбувається обробка запитів до каталогу, перевірка користувача під час входу в систему, формування замовлень, збереження відгуків, передавання даних для доставки та оплати, а також взаємодія із зовнішніми сервісами. У межах розроблюваної системи цей рівень побудовано на основі Node.js та Express, що дозволило реалізувати серверне API, необхідне для роботи всіх основних модулів магазину [17, 18].

На практиці серверна частина не є одним суцільним програмним блоком. Її реалізація організована через окремі маршрути, контролери, моделі даних, middleware та сервіси. Такий підхід є виправданим для інтернет-магазину, оскільки логіка каталогу, авторизації, замовлень і зовнішніх інтеграцій суттєво відрізняється за призначенням. Якщо об’єднати ці частини без чіткого поділу, код швидко ускладнюється, а внесення змін в одну ділянку починає впливати на інші модулі.

Основою серверної реалізації є набір API-маршрутів. Частина з них використовується для отримання каталогу товарів, окремих категорій і карток товарів. Інша група маршрутів відповідає за автентифікацію та роботу з профілем користувача. Окремо виділено маршрути для замовлень, відгуків, доставки, оплати, обробки зображень та службових інтеграцій. Завдяки цьому система працює не як набір випадкових запитів, а як структуроване серверне середовище, у якому кожен маршрут має чітке призначення.

Для зберігання даних використано MongoDB, а роботу з моделями організовано через Mongoose. На рівні реалізації це дозволило описати основні сутності магазину у вигляді окремих моделей і надалі звертатися до них у контролерах та сервісах. Такий підхід є зручним з кількох причин. По-перше, він дає змогу централізовано керувати структурою даних. По-друге, полегшує перевірку вхідної інформації. По-третє, спрощує подальше розширення системи, якщо виникає потреба додати нові поля або службові властивості [19, 20].

У серверній частині реалізовано кілька ключових напрямів логіки. Перший із них стосується каталогу. Він охоплює отримання товарів, роботу з категоріями, фільтрацію, сортування, пошук та видачу окремої картки товару. На цьому рівні сервер приймає параметри запиту від клієнтської частини, обробляє їх, виконує звернення до бази даних і формує результат у вигляді відповіді, придатної для відображення в інтерфейсі. Для магазину відеогрових консолей це має особливе значення, оскільки асортимент містить кілька типів товарів і вимагає достатньо гнучкої обробки.

Другий важливий блок пов’язаний з користувачами та автентифікацією. У системі реалізовано реєстрацію, вхід, отримання даних профілю та доступ до захищених маршрутів. Для цього використовується механізм JWT-автентифікації, за якого після успішного входу користувач отримує токен доступу. Надалі цей токен перевіряється на сервері за допомогою middleware, і лише після цього користувач отримує доступ до персоналізованих дій, наприклад до перегляду власних замовлень або оформлення покупки. Такий підхід дає змогу відокремити публічну частину магазину від тих операцій, які повинні виконуватися тільки після підтвердження облікового запису.

Третій, один із найважливіших, блок стосується замовлень. Саме тут зводяться результати роботи каталогу, кошика, особистого кабінету, доставки й оплати. Під час створення замовлення сервер приймає інформацію про вибрані товари, контактні дані користувача, параметри доставки та спосіб оплати, після чого формує новий запис у базі даних. На цьому етапі важливо забезпечити цілісність обробки: замовлення не повинно створюватися частково або з

неперевіреними даними. Тому логіка цього блоку реалізована так, щоб перед збереженням виконувалась перевірка потрібних полів, а результат операції повертався у зрозумілому для фронтенду форматі.

Окремо реалізовано роботу з відгуками. Хоча цей модуль не є визначальним для самого факту продажу, він суттєво впливає на наповнення картки товару та довіру до магазину. На серверному рівні це означає збереження відгуку, зв’язування його з конкретним товаром і користувачем, а також повернення оновленого результату для подальшого відображення в інтерфейсі.

У межах реалізації серверної частини важливу роль відіграють і службові компоненти. До них належать middleware для перевірки токенів, валідації запитів, централізованої обробки помилок, а також засоби роботи із зображеннями та допоміжними запитами. Для інтернет-магазину ці елементи мають практичне значення, оскільки без них система або залишалася б вразливою, або повертала б нестабільні результати при некоректних зверненнях до API.

На рисунку 3.1 доцільно подати діаграму класів серверної частини, яка відображає основні моделі, контролери та логічні зв’язки між ними.

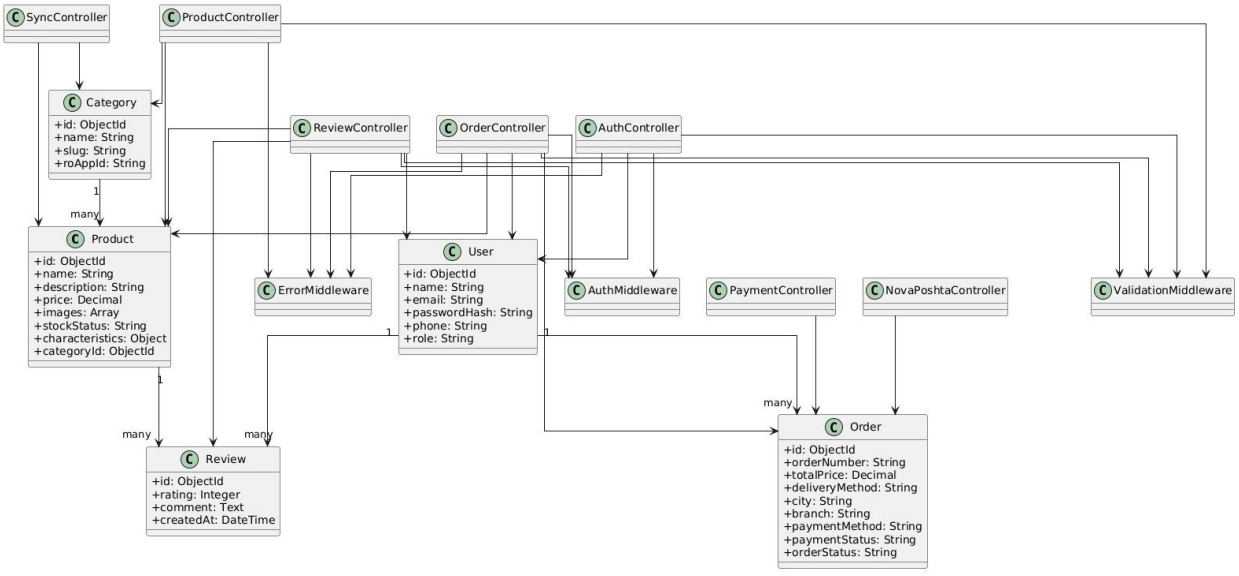


Рис 3.1 Діаграма класів серверної частини системи

Окрім загальної структури модулів, у серверній частині важливо показати і послідовність виконання однієї з ключових операцій. Для цього доцільно окремо подати сценарій створення замовлення, оскільки саме він поєднує кілька критичних дій: приймання даних із клієнтської частини, перевірку користувача, формування запису в базі даних і підготовку відповіді для подальшої оплати або відображення в особистому кабінеті.

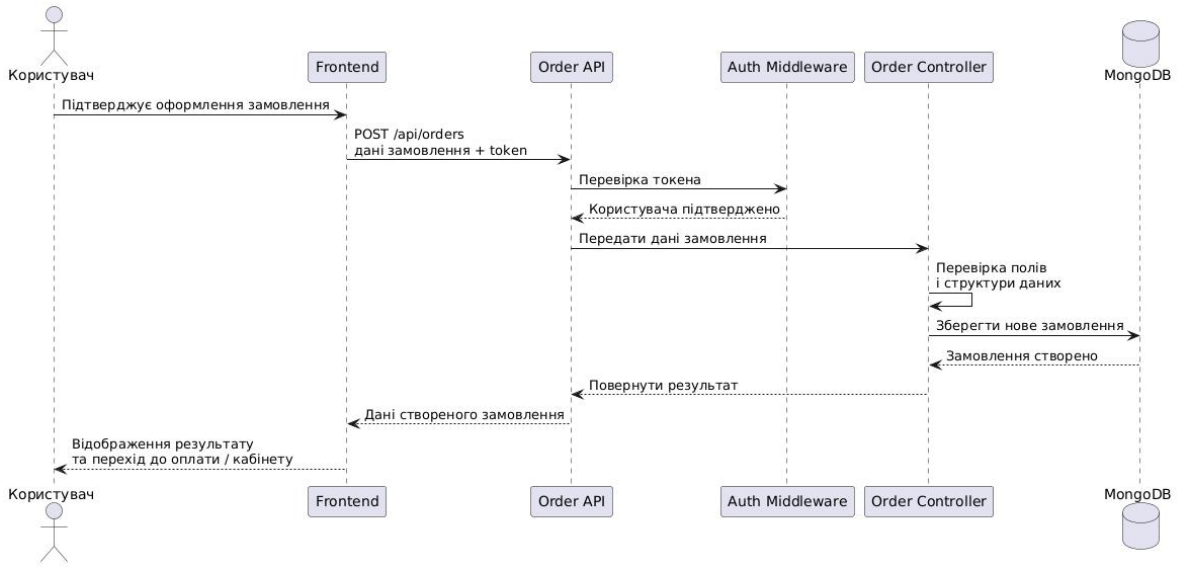


Рис 3.2 Діаграма послідовності створення замовлення

Отже, серверна частина реалізована як структурована система модулів, у якій окремо організовано роботу з каталогом, користувачами, замовленнями, відгуками та службовими компонентами. Такий підхід дозволив побудувати узгоджене API, яке забезпечує взаємодію між клієнтською частиною, базою даних та зовнішніми сервісами й підтримує повний цикл функціонування магазину.

3.3 Реалізація інтеграції із зовнішніми сервісами

Для інтернет-магазину повний цикл роботи не завершується на рівні власного каталогу, кошика і сторінок оформлення замовлення. У реальному використанні система повинна взаємодіяти із зовнішніми сервісами, без яких

частина функціональності залишалася б неповною. У даному проєкті така взаємодія реалізована для трьох основних напрямів: доставки, оплати та синхронізації товарних даних. Йдеться про інтеграцію з Nova Poshta, Monobank і RoApp.

Усі зовнішні звернення в системі організовано через серверну частину. Такий підхід був обраний свідомо. По-перше, він дозволяє не виносити службові ключі та параметри доступу в клієнтський інтерфейс. По-друге, саме сервер може перевірити вхідні дані, перетворити їх у потрібний формат, зберегти результат запиту та передати у фронтенд уже підготовлену відповідь. Для магазину це важливо, оскільки користувач повинен бачити простий і зрозумілий інтерфейс, а не деталі роботи сторонніх API.

Інтеграція з Nova Poshta потрібна на етапі оформлення замовлення. Користувачеві необхідно вибрати населений пункт і відділення, куди буде доставлено товар. Щоб не підтримувати таку довідкову інформацію вручну всередині системи, доцільніше одержувати її безпосередньо із сервісу доставки. На практиці це означає, що сервер приймає запит від клієнтської частини, звертається до API служби доставки, отримує список міст або відділень і повертає його в інтерфейс оформлення замовлення. Завдяки цьому дані залишаються актуальними, а користувач працює з реальною інформацією, а не з фіксованим локальним списком [12].

Інтеграція з Monobank відповідає за оплату замовлення. Після того як користувач перевірив склад покупки, заповнив контактні дані та обрав доставку, система повинна підготувати платіжний сценарій. Для цього сервер формує необхідні параметри замовлення, ініціює створення платіжної сесії й повертає клієнтській частині результат, потрібний для подальшого переходу до оплати. Після виконання платіжної операції система повинна отримати або перевірити її результат і відобразити користувачеві підсумок у зрозумілому вигляді. Саме тому логіка роботи з оплатою не зводиться до одного запиту, а охоплює створення операції, повернення користувача до магазину й оновлення статусу замовлення [13].

Окреме місце в проєкті займає інтеграція з RoApp, оскільки вона стосується не етапу покупки, а актуальності даних каталогу. У магазині з постійно змінюваним асортиментом незручно покладатися тільки на ручне внесення товарів. Значно практичніше синхронізувати товари й категорії із зовнішнім джерелом даних. У межах цієї реалізації сервер отримує або оновлює інформацію про позиції, категорії та службові ідентифікатори, після чого зберігає її у власній базі даних у формі, придатній для подальшого використання в каталозі. Завдяки цьому асортимент магазину не існує окремо від зовнішньої системи, а підтримується в більш узгодженому стані [14].

Крім прямих запитів, важливу роль відіграє обробка webhook-подій. Такий механізм дозволяє системі не тільки самостійно звертатися до стороннього сервісу, а й приймати повідомлення від нього про зміну стану даних. У контексті магазину це корисно насамперед для синхронізації товарної інформації. Якщо зовнішня система повідомляє про зміну товару або категорії, сервер може прийняти цю подію, перевірити її коректність і оновити локальні записи без окремої ручної дії з боку користувача або адміністратора. Це робить підтримку каталогу більш керованою і зменшує ризик застарілих даних на сторінках магазину [14].

Під час реалізації інтеграцій особливу увагу потрібно було приділити перевірці даних і обробці помилок. У випадку доставки система повинна коректно реагувати на ситуації, коли сервіс не повертає потрібний результат або коли користувач ввів неповні дані. Для оплати критично важливо не створювати невизначений стан, коли користувач уже вважає замовлення завершеним, а сервер ще не отримав підтвердження про результат операції. Для синхронізації товарів принципове значення має перевірка вхідних даних, щоб зовнішнє оновлення не призвело до пошкодження локальної структури каталогу. Саме тому інтеграційний рівень у системі реалізований не як простий міст до сторонніх API, а як окремий шар серверної логіки з власними перевірками..

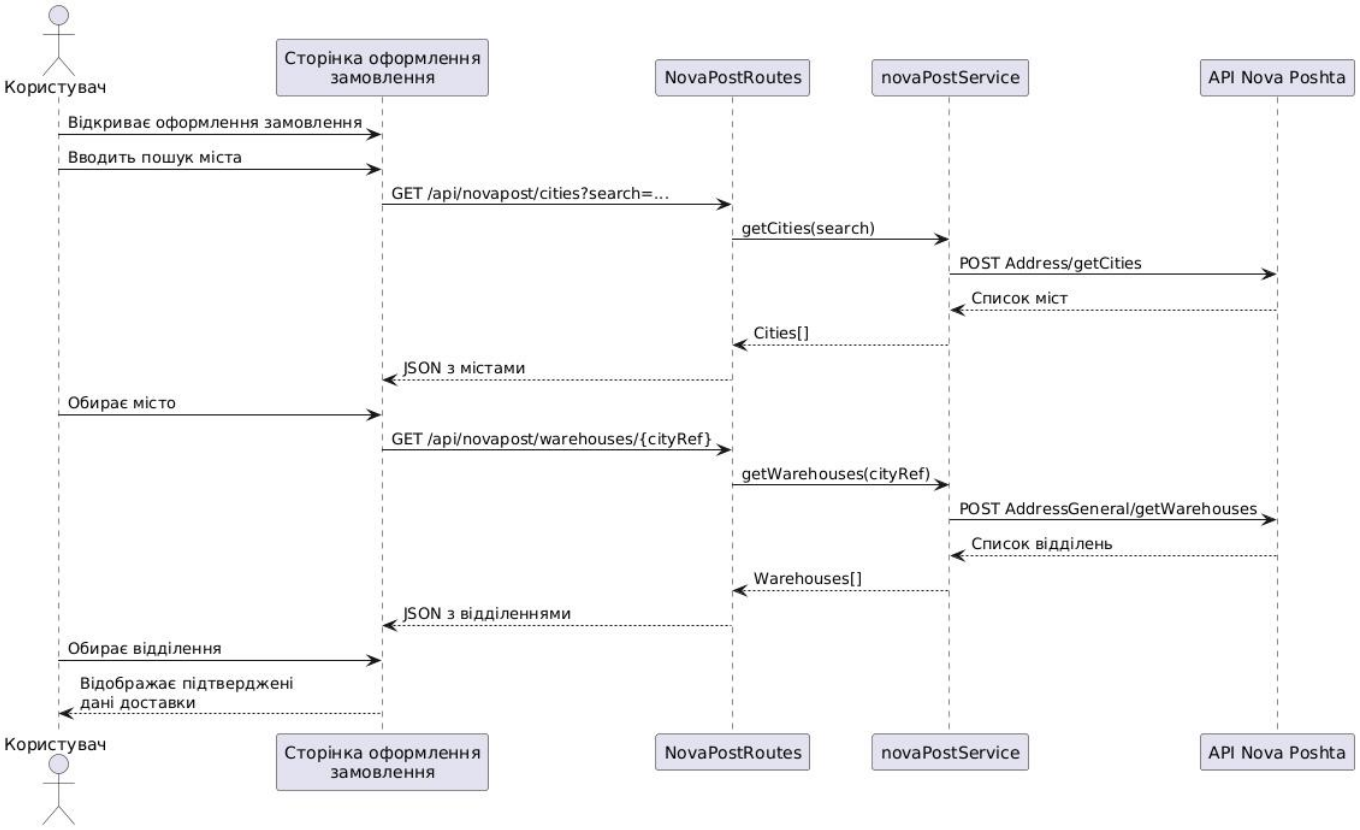


Рис 3.3 Діаграма послідовності інтеграції з Nova Poshta

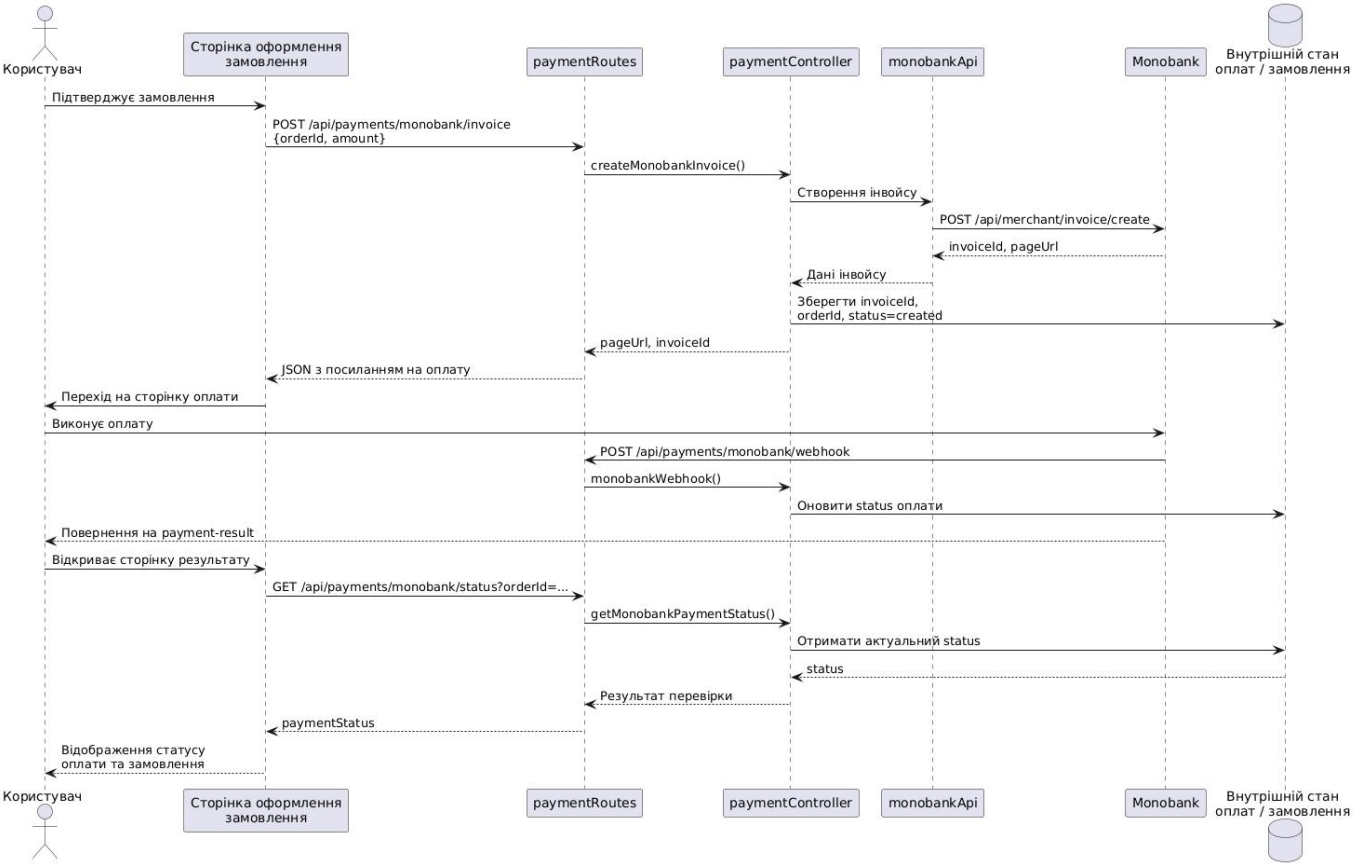


Рис 3.4 Діаграма послідовності інтеграції з Monobank

У практичному плані реалізація зовнішніх інтеграцій дала можливість зробити систему ближчою до реального інтернет-магазину, а не лише до демонстраційного проєкту. Nova Poshta забезпечує реалістичний сценарій вибору доставки, Monobank додає завершений етап оплати, а RoApp підтримує актуальність товарного каталогу. Разом ці інтеграції формують ту частину функціональності, яка відрізняє повноцінний магазин від інформаційного сайту з товарами.

Отже, зовнішні сервіси в даній системі виступають не допоміжним доповненням, а важливою складовою всієї логіки роботи. Їх інтеграція через серверну частину дозволила поєднати інтерфейс, внутрішню бізнес-логіку, каталог, доставку, оплату та синхронізацію даних у межах одного узгодженого рішення.

3.4 Опис ключових класів, моделей і програмних модулів системи

У структурі реалізованої системи можна виділити кілька груп програмних елементів, які безпосередньо забезпечують роботу магазину. До них належать моделі даних, серверні контролери, маршрути API, клієнтські сторінки, компоненти інтерфейсу, а також модулі керування станом. Такий поділ не є формальним: кожна з цих частин виконує окрему роль, а разом вони утворюють узгоджене середовище, у якому користувач може переглядати товари, формувати кошик, оформлювати замовлення та взаємодіяти з особистим кабінетом.

Серед моделей даних центральне місце займає модель Product. Вона описує товарну позицію, яка відображається в каталозі та картці товару. У межах цієї моделі зберігаються назва, опис, ціна, зображення, характеристики, службові ідентифікатори та інші поля, потрібні для показу товару на фронтенді і для синхронізації із зовнішніми джерелами. Саме ця модель є базовою для каталогу, пошуку, відгуків і багатьох операцій, пов’язаних із замовленням.

Поруч із нею використовується модель Category, яка потрібна для побудови структури каталогу. Вона дозволяє групувати товари за логічними розділами і підтримує навігацію по асортименту. Завдяки їй каталог не перетворюється на суцільний список позицій, а зберігає зрозумілу ієрархію. Для системи такого типу це особливо важливо, оскільки користувач зазвичай починає перегляд не з конкретного товару, а з певної групи товарів.

Модель User відповідає за обліковий запис користувача. У ній зберігаються персональні дані, електронна адреса, пароль у захищеному вигляді та інші поля, пов’язані з автентифікацією і роботою особистого кабінету. З практичної точки зору саме ця модель забезпечує перехід від публічного режиму роботи магазину до персоналізованого. Після входу в систему користувач уже працює не просто з каталогом, а з власними замовленнями, збереженими даними та індивідуальною історією взаємодії.

Однією з найважливіших моделей є Order. Вона зберігає інформацію про оформлену покупку: склад замовлення, контактні дані, параметри доставки, спосіб оплати, статус та зв’язок із конкретним користувачем. У межах серверної логіки ця модель фактично є центральною транзакційною сутністю. Через неї проходить підсумок роботи кошика, авторизації, доставки і платежу. Саме тому її роль у системі значно ширша, ніж просте збереження переліку придбаних товарів.

Для підтримки зворотного зв’язку реалізовано модель Review, яка використовується для зберігання відгуків до товарів. Вона пов’язує товар із користувачем, який залишив оцінку або коментар. Хоча цей модуль не визначає сам факт продажу, він має значення для наповнення картки товару та створення більш живого інформаційного середовища магазину.

На серверному рівні ключову роль відіграють контролери. Саме вони приймають запити від клієнтської частини, звертаються до моделей і формують відповідь у потрібному форматі. Для роботи з каталогом використовуються контролери, що відповідають за товари та категорії. Через них реалізовано отримання списків товарів, окремих позицій, фільтрацію та пов’язані операції.

Для облікового запису і входу в систему застосовуються контролери авторизації, а для оформлення покупки — контролери замовлень. Окремо виділено логіку для платежів, доставки та синхронізації.

Важливе місце посідають і серверні сервіси. Якщо контролер є точкою входу для запиту, то сервіс зазвичай виконує більш прикладну або зовнішню логіку. Саме тому робота з Nova Poshta, Monobank і RoApp доцільно винесена в окремі програмні модулі. Це дає змогу не перевантажувати контролери деталями зовнішньої інтеграції та зберігати код у більш структурованому вигляді. Такий поділ спрощує супровід системи, особливо в тих випадках, коли змінюються умови стороннього API або потрібно оновити логіку взаємодії з ним.

На клієнтському рівні одними з головних елементів є сторінки Products, ProductDetail, Cart, Wishlist, Login, Register, Account, OrderDetail та PaymentResult. Кожна з них пов’язана з окремим етапом взаємодії користувача із системою. Наприклад, сторінка каталогу відповідає за пошук і вибір товару, сторінка картки товару — за детальний перегляд і прийняття рішення про покупку, кошик — за формування набору товарів перед замовленням, а сторінка оформлення або результату оплати — за завершальний етап сценарію. Така структура дозволяє розглядати клієнтську частину не як набір випадкових екранів, а як послідовну систему переходів.

Окремо слід виділити модулі керування станом, реалізовані через Redux Toolkit. Для інтернет-магазину це особливо важливо, оскільки частина даних повинна бути доступною відразу в кількох місцях інтерфейсу. Йдеться насамперед про кошик, список бажань, стан авторизації та окремі дані замовлення. Якщо подібна логіка розподіляється хаотично по компонентах, система швидко стає важкою для підтримки. Централізоване керування станом дозволяє уникнути дублювання й забезпечує більш передбачувану поведінку інтерфейсу.

З технічної точки зору важливими є також допоміжні модулі, які не завжди помітні користувачеві безпосередньо. Це middleware для перевірки

токенів доступу, модулі валідації, обробники помилок, логіка роботи із зображеннями та службові частини інтеграцій. Саме ці елементи забезпечують стабільність системи в реальному використанні. Без них навіть добре реалізовані моделі й сторінки могли б працювати непослідовно або вразливо.

Для наочного подання взаємозв’язку між основними класами і модулями в цьому підпункті доцільно використати діаграму класів прикладної логіки системи або схему взаємодії основних програмних модулів. Така діаграма дозволяє показати, як саме пов’язані між собою моделі даних, контролери, сервіси та клієнтські компоненти, не зводячи опис лише до текстового переліку.

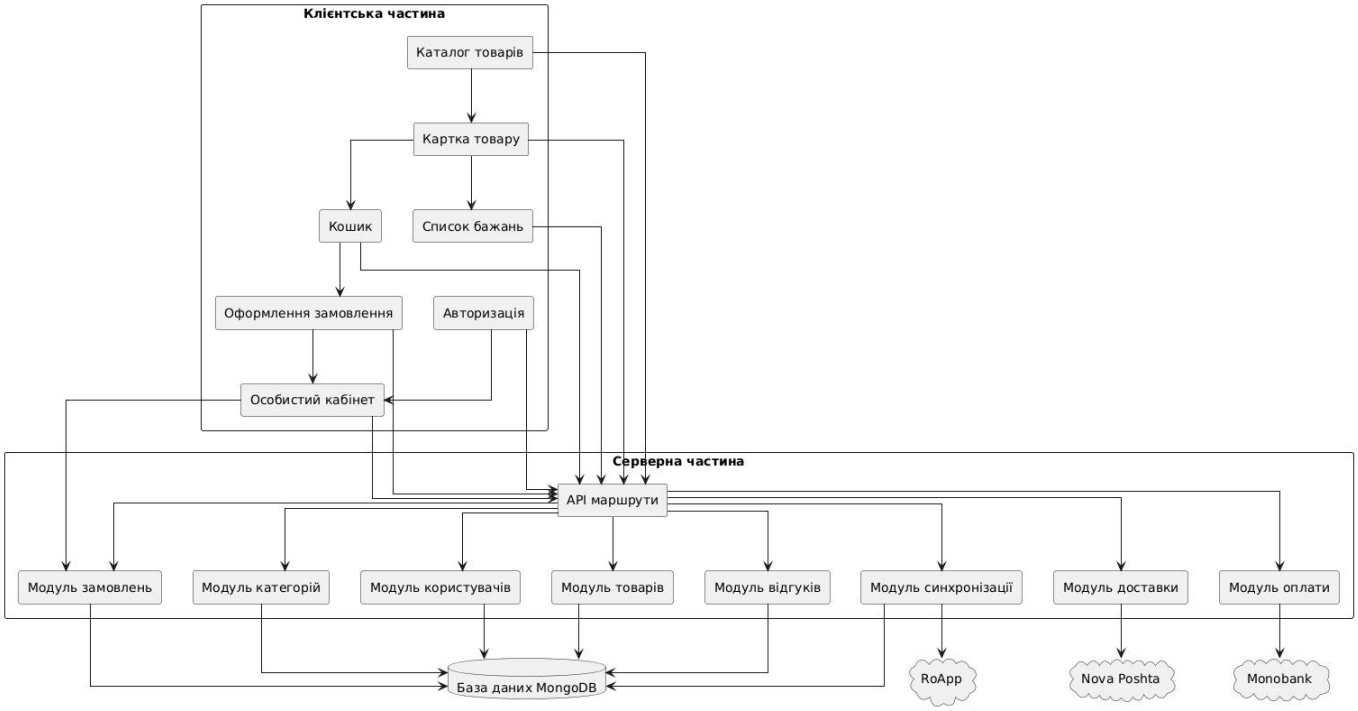


Рис. 3.5 Діаграма взаємодії ключових програмних модулів системи

Отже, реалізована система складається з набору взаємопов’язаних моделей, контролерів, сервісів, сторінок і модулів стану, кожен з яких відповідає за свою частину функціональності. Саме таке розділення дало змогу побудувати магазин не як сукупність окремих сторінок, а як цілісне програмне рішення, у якому каталог, користувач, замовлення, доставка, оплата та допоміжні сервіси працюють узгоджено.

3.5 Опис функціонування системи та екранних форм

Функціонування системи побудовано як послідовний користувацький сценарій, у якому окремі модулі не існують ізольовано, а переходять один в інший без порушення загальної логіки роботи магазину. Для користувача це виглядає як звичний процес взаємодії з інтернет-магазином: перегляд асортименту, вибір товару, формування кошика, оформлення покупки, оплата та подальший перегляд замовлення. Проте на рівні реалізації кожний із цих етапів пов’язаний із конкретними сторінками, маршрутами, API-запитами та зміною стану системи.

Початком взаємодії є головна сторінка, з якої користувач може перейти до каталогу товарів, тематичних розділів або інформаційних сторінок. На цьому етапі система виконує не лише роль точки входу, а й роль навігаційного центру, через який формується подальший маршрут. Якщо користувач уже має конкретний запит, він переходить до каталогу; якщо ж ні, система дає змогу ознайомитися з основними категоріями й актуальними пропозиціями.

Наступний етап пов’язаний із роботою каталогу. Тут користувач переглядає товари, застосовує пошук, фільтрацію та сортування, а далі переходить до сторінки окремої позиції. У межах цього сценарію система має швидко реагувати на зміну параметрів перегляду, оновлювати список товарів і не вимагати повторного завантаження всієї сторінки. Саме каталог є тим модулем, де користувач найчастіше звужує вибір і переходить від загального перегляду до конкретного рішення.

На сторінці товару користувач отримує розширену інформацію про вибрану позицію: назву, ціну, опис, зображення, характеристики та доступні дії. У цій частині інтерфейсу система фактично переводить користувача від етапу ознайомлення до етапу прийняття рішення. Саме тут він додає товар до кошика або до списку бажань. Якщо рішення про покупку ще не остаточне, товар може

бути збережений для подальшого повернення. Якщо ж рішення прийнято, користувач переходить до кошика.

Кошик виконує роль проміжного контрольного етапу перед оформленням замовлення. У ньому користувач бачить перелік вибраних товарів, може змінити їх кількість, видалити зайві позиції та перевірити поточну суму покупки. Важливо, що на цьому етапі система не лише відображає список товарів, а й зберігає логіку переходу до наступного кроку. Саме з кошика користувач переходить до оформлення замовлення, тому зрозумілість і стабільність роботи цього модуля безпосередньо впливають на завершення покупки.

Оформлення замовлення побудоване як послідовний сценарій введення та підтвердження даних. Користувач вказує контактну інформацію, обирає місто і відділення доставки, перевіряє склад покупки та переходить до оплати. На цьому етапі система звертається до зовнішнього сервісу доставки для отримання актуальних параметрів, а після підтвердження замовлення ініціює взаємодію з платіжним сервісом. У результаті оформлення не є окремою формою, а виступає зв’язувальною ланкою між каталогом, користувацьким профілем, доставкою та оплатою.

Після завершення платіжного сценарію користувач повертається до системи й отримує результат оплати. Далі інформація про замовлення відображається в особистому кабінеті, де можна переглянути історію покупок і деталі конкретного замовлення. Саме це завершує повний цикл взаємодії: від першого перегляду товару до фіксації результату покупки в системі.

Головна сторінка системи виконує роль стартового екрану, з якого користувач може перейти до каталогу, інформаційних сторінок або окремих пропозицій. Її структура побудована так, щоб одразу дати доступ до ключових розділів і не перевантажувати інтерфейс другорядною інформацією.

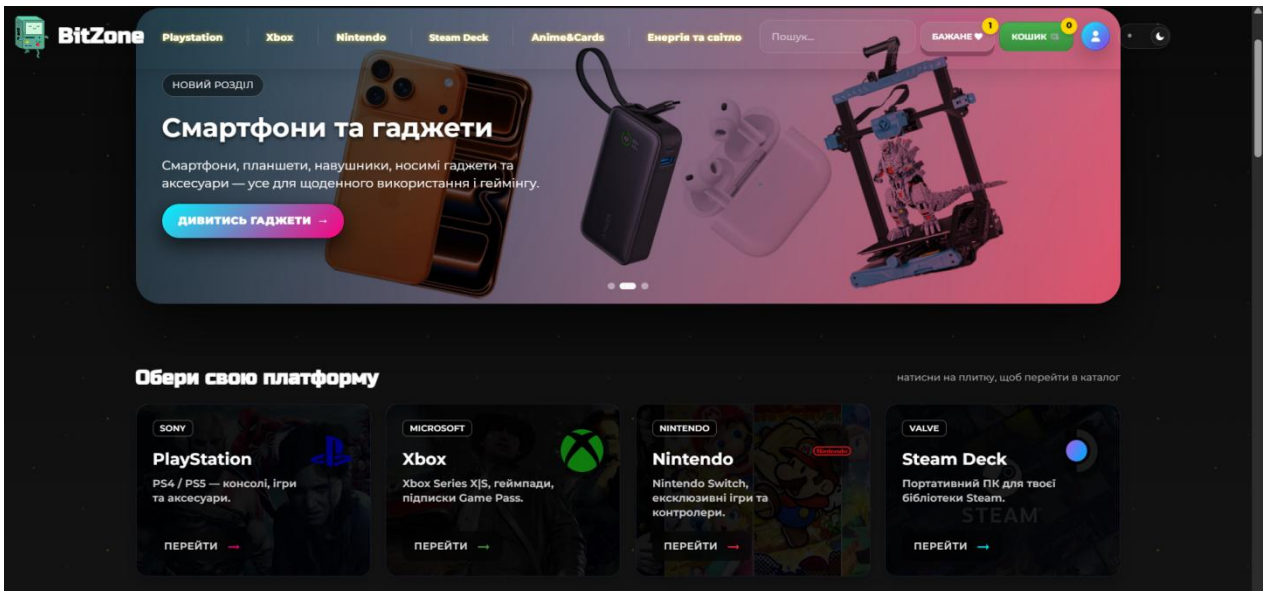


Рис. 3.6 Головна сторінка системи

Сторінка каталогу відображає асортимент товарів і засоби навігації по ньому. Саме тут реалізовано пошук, фільтрацію та сортування. Для користувача це одна з найбільш функціонально насичених сторінок, оскільки вона забезпечує перехід від загального перегляду до вибору конкретної позиції.

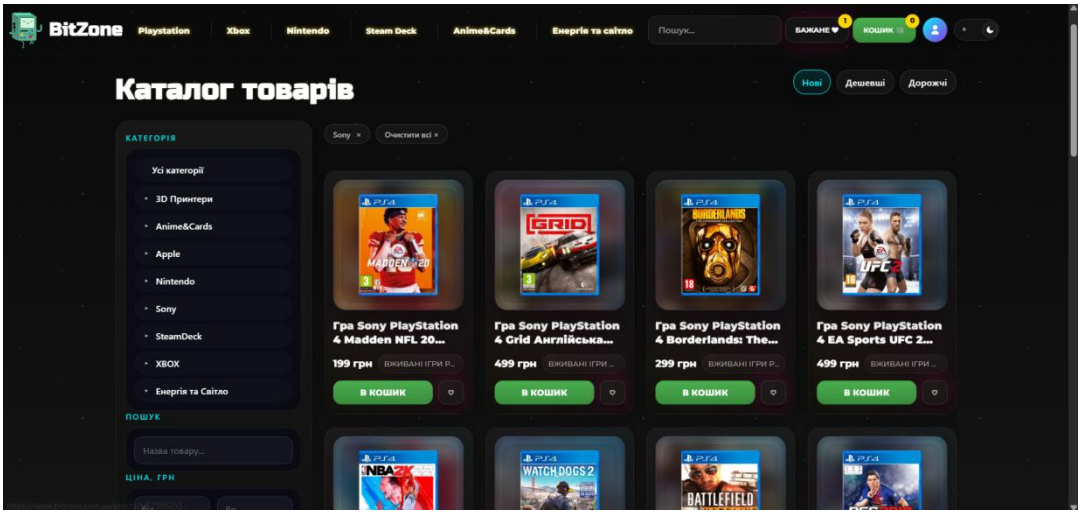


Рис. 3.7 Сторінка каталогу товарів

Картка товару містить опис конкретної позиції. На ній зібрано основні характеристики, ціну, візуальні матеріали, відгуки та кнопки дії. Саме ця сторінка є найбільш важливою з точки зору прийняття рішення про покупку.

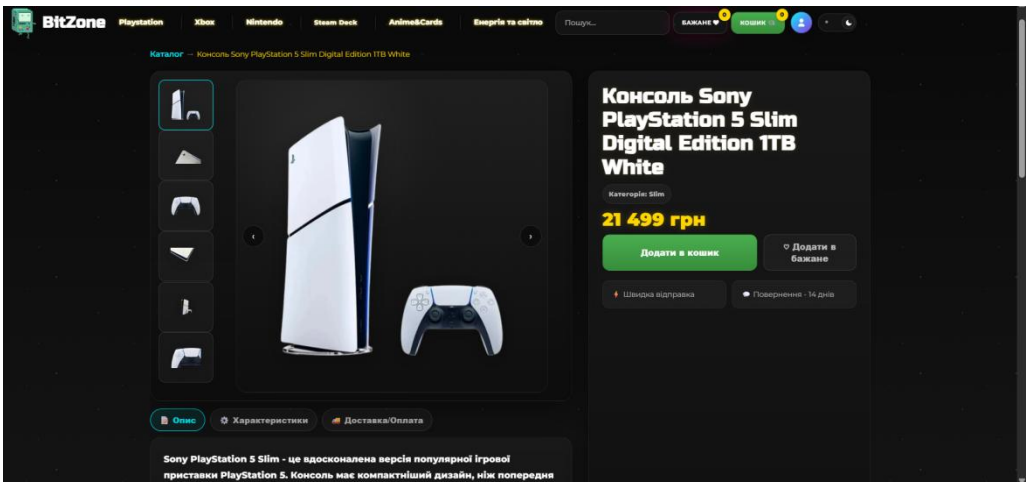


Рис. 3.8 Сторінка окремого товару

Екран кошика та сторінка оформлення замовлення відображають перехід до завершального етапу покупки — ввести необхідні дані, вибрати доставку і перейти до оплати. З позиції функціонування системи саме ці сторінки об’єднують інтерфейсну, серверну та інтеграційну логіку.

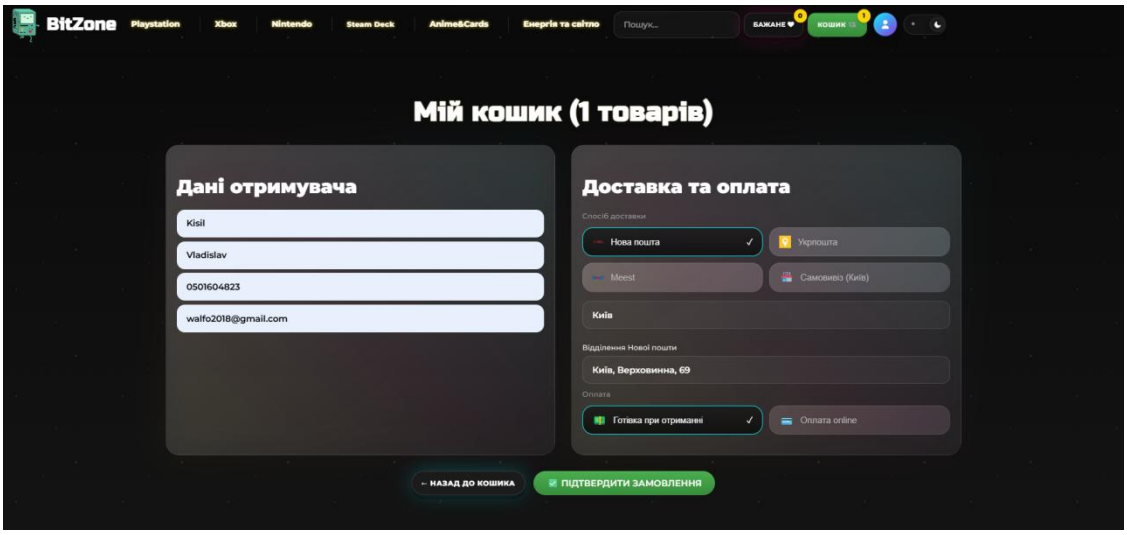


Рис. 3.9 Екран кошика та оформлення замовлення

Таким чином, функціонування системи реалізоване як послідовний ланцюг взаємопов’язаних екранів, у межах якого користувач проходить повний маршрут від пошуку товару до перегляду результату покупки.

Висновки до розділу 3.

У третьому розділі описано програмну реалізацію web-сайту BiTZone. Клієнтську частину реалізовано на основі React із використанням маршрутизації, компонентного підходу та централізованого керування станом для кошика, списку бажань і авторизації. На рівні інтерфейсу реалізовано головну сторінку, каталог, картку товару, кошик, сторінку оформлення замовлення, сторінки реєстрації та входу, особистий кабінет і сторінку списку бажань.

Серверну частину реалізовано на Node.js та Express у вигляді REST API з окремими маршрутами, контролерами, middleware, Mongoose-моделями та сервісами бізнес-логіки. Для збереження даних використано MongoDB, а для роботи з моделями — Mongoose. Окремо реалізовано інтеграцію з Nova Poshta для отримання міст і відділень, Monobank для створення та перевірки платежів, а також RoApp для синхронізації товарів, категорій, залишків, клієнтів і замовлень.

4 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

Після завершення реалізації основних модулів системи було проведено тестування її працездатності та перевірку коректності виконання ключових функцій. Для інтернет-магазину цей етап є важливим, оскільки стабільність роботи впливає не лише на зручність користувача, а й на правильність оформлення замовлень, обробки платежів і взаємодії з доставкою.

У межах роботи основна увага приділялась функціональному тестуванню. Перевірялась робота каталогу товарів, сторінок окремих позицій, кошика, авторизації користувача, оформлення замовлення та особистого кабінету. Окремо тестувалась взаємодія із зовнішніми сервісами, зокрема отримання даних доставки через Nova Poshta та створення платіжних операцій через Monobank.

Перевірка проводилась як для окремих модулів, так і для повного сценарію роботи системи. Під час тестування оцінювалась правильність переходів між сторінками, збереження даних користувача, передача інформації між клієнтською та серверною частинами, а також обробка помилкових запитів. Для серверної частини додатково перевірялась робота API-маршрутів і взаємодія з базою даних MongoDB.

Особливу увагу було приділено роботі кошика та оформленню замовлення, оскільки саме ці модулі поєднують основну логіку системи. Під час перевірки тестувались додавання та видалення товарів, зміна кількості позицій, передача контактних даних, вибір доставки та створення платіжної сесії.

У результаті проведеного тестування було встановлено, що система коректно виконує основні функції інтернет-магазину. Реалізовані модулі стабільно взаємодіють між собою, а інтеграція із зовнішніми сервісами забезпечує виконання необхідних сценаріїв оформлення замовлення та оплати.

У наступних підпунктах наведено тест-кейси та результати перевірки основних функціональних можливостей системи.

4.1 Методика та організація тестування

Тестування програмної системи проводилось після реалізації основних модулів і було спрямоване на перевірку працездатності інтернет-магазину в межах типових користувацьких сценаріїв. Основну увагу приділено діям, без яких неможливий повний цикл роботи системи: перегляду каталогу, вибору товару, роботі з кошиком, авторизації, оформленню замовлення, вибору доставки та переходу до оплати.

Для перевірки було використано функціональне та інтеграційне тестування. Функціональне тестування дало змогу оцінити правильність роботи окремих модулів системи. Інтеграційне тестування застосовувалось для перевірки обміну даними між клієнтською частиною, сервером, базою даних MongoDB та зовнішніми сервісами Nova Poshta, Monobank і RoApp.

Тестування виконувалось у локальному середовищі розробки з послідовною перевіркою основних модулів. Спочатку перевірялись окремі сторінки та API-маршрути, після чого тестувався повний сценарій роботи користувача: відкриття каталогу, вибір товару, додавання до кошика, оформлення замовлення, вибір доставки та створення платіжної сесії.

Окремо перевірялась поведінка системи у випадку некоректних дій користувача: введення неповних даних, спроба доступу до захищених сторінок без авторизації, помилки під час отримання даних від зовнішніх сервісів. Це дозволило оцінити не лише позитивні сценарії, а й здатність системи коректно реагувати на помилкові ситуації.

Узагальнену методику тестування наведено в таблиці 4.1.

Таблиця 4.1

Методика тестування основних модулів системи

№ з/п	Об’єкт тестування	Вид тестування	Зміст перевірки	Очікуваний результат
1	Каталог і картка товару	Функціональне	Перевірка завантаження товарів, пошуку, фільтрації, сортування та переходу до сторінки товару	Товари відображаються коректно, сторінка товару відкривається без помилок
2	Кошик і список бажань	Функціональне	Додавання, видалення, зміна кількості товарів, перерахунок вартості та збереження стану	Склад покупки оновлюється правильно, підсумкова сума відповідає вибраним товарам
3	Авторизація і кабінет	Функціональне	Реєстрація, вхід, вихід, доступ до захищених сторінок і перегляд даних користувача	Персональні функції доступні лише після успішної авторизації
4	Оформлення замовлення	Функціональне	Передача контактних даних, перевірка складу покупки, створення запису замовлення	Замовлення створюється та пов’язується з відповідним користувачем
5	Доставка Nova Poshta	Інтеграційне	Отримання списку міст і відділень через серверний API	Користувач отримує актуальні варіанти доставки під час оформлення
6	Оплата Monobank	Інтеграційне	Створення платіжної сесії, перехід до оплати, перевірка статусу платежу	Система формує платіжний сценарій і оновлює статус оплати
7	Помилкові сценарії	Негативне тестування	Неповні форми, неправильні дані авторизації, недоступність зовнішнього сервісу	Система не завершує роботу аварійно та повідомляє користувача про помилку

Наведена методика дозволяє послідовно перевірити окремі функціональні модулі та їхню взаємодію між собою. Такий підхід є достатнім для підтвердження працездатності основних сценаріїв інтернет-магазину та підготовки результатів тестування в наступних підпунктах роботи.

4.2 Тестування основних функціональних модулів

Після визначення загального підходу до перевірки було виконано тестування основних модулів системи. Основну увагу приділено тим частинам, які безпосередньо формують користувацький сценарій інтернет-магазину: каталогу, картці товару, кошику, авторизації, оформленню замовлення, доставці, оплаті та особистому кабінету.

Першим перевірявся модуль каталогу товарів. У процесі тестування відкривалась сторінка з товарами, виконувалась перевірка завантаження асортименту, пошуку, фільтрації та сортування. Також перевірявся перехід із каталогу до сторінки окремого товару. У результаті встановлено, що каталог коректно отримує дані із серверної частини та відображає їх у клієнтському інтерфейсі.

Наступним етапом було протестовано сторінку товару. Перевірялась наявність основної інформації про товар: назви, ціни, опису, зображень, характеристик, а також можливість додати товар до кошика або списку бажань. Окремо перевірялась робота блоку відгуків, оскільки він пов’язаний із даними користувача та конкретною товарною позицією.

Окрему увагу приділено модулю кошика. Під час тестування перевірялось додавання товарів, зміна кількості, видалення позицій, перерахунок загальної вартості та збереження вибраних товарів під час переходу між сторінками. Цей модуль є одним із найважливіших, оскільки саме через нього користувач переходить до оформлення замовлення.

Модуль авторизації перевірявся через сценарії реєстрації, входу до системи та виходу з облікового запису. Також тестувалась поведінка системи при введенні неправильних даних. У результаті перевірки було встановлено, що доступ до персоналізованих сторінок надається лише після успішної автентифікації.

Таблиця 4.2

Тестування основних функціональних модулів системи

№	Модуль	Дія під час тестування	Очікуваний результат	Фактичний результат
1	Каталог товарів	Відкрити сторінку каталогу	Система відображає список товарів	Список товарів відображено коректно
2	Пошук товарів	Ввести назву товару в поле пошуку	Відображаються товари, що відповідають запиту	Пошук працює коректно
3	Фільтрація	Обрати категорію або параметр фільтра	Каталог оновлює перелік товарів	Результати фільтрації відображаються правильно
4	Картка товару	Відкрити сторінку окремого товару	Відображається опис, ціна, зображення та характеристики	Дані товару завантажено коректно
5	Список бажань	Додати товар до wishlist	Товар зберігається у списку бажань	Функція працює коректно
6	Кошик	Додати товар до кошика	Товар з’являється в кошику	Товар успішно додано
7	Кошик	Змінити кількість товару	Загальна сума перераховується автоматично	Сума оновлюється правильно
8	Авторизація	Виконати вхід із правильними даними	Користувач отримує доступ до особистого кабінету	Вхід виконано успішно
9	Авторизація	Виконати вхід із неправильними даними	Система повідомляє про помилку	Помилка відображається коректно
10	Оформлення замовлення	Заповнити дані та підтвердити замовлення	Замовлення створюється і зберігається в системі	Замовлення створено коректно
11	Доставка	Обрати місто та відділення Nova Poshta	Система повертає доступні варіанти доставки	Дані доставки завантажуються правильно
12	Оплата	Створити платіж через Monobank	Система формує платіжну сесію	Платіжна сесія створюється
13	Особистий кабінет	Відкрити історію замовлень	Користувач бачить власні замовлення	Історія замовлень відображається
14	Відгуки	Додати відгук до товару	Відгук зберігається та відображається в картці товару	Відгук додається коректно

Під час тестування також перевірялись ситуації з помилковими або неповними даними. Зокрема, було розглянуто спробу оформлення замовлення без авторизації, введення некоректних даних у формах, відкриття захищених сторінок без токена доступу та можливі помилки під час звернення до зовнішніх сервісів. У таких випадках система не повинна завершувати роботу аварійно, а має повертати користувачу зрозуміле повідомлення або блокувати виконання некоректної дії.

Отже, тестування основних функціональних модулів показало, що реалізовані частини системи працюють узгоджено та забезпечують виконання ключових сценаріїв інтернет-магазину. Перевірка каталогу, кошика, авторизації, оформлення замовлення, доставки, оплати та особистого кабінету підтвердила працездатність основної логіки програмної системи.

4.3 Результати тестування системи

За результатами перевірки встановлено, що реалізована система виконує основні функції інтернет-магазину та підтримує головний користувацький сценарій: перегляд товарів, вибір позиції, додавання до кошика, оформлення замовлення, вибір доставки, перехід до оплати та перегляд інформації в особистому кабінеті.

Під час тестування оцінювалася не лише робота окремих сторінок, а й їх взаємодія між собою. Для інтернет-магазину важливо, щоб каталог, кошик, авторизація, оформлення замовлення і зовнішні сервіси працювали як послідовний ланцюг, без втрати даних під час переходів між етапами покупки.

Перевірка показала, що каталог коректно завантажує дані із серверної частини, сторінка товару відображає основну інформацію про позицію, а кошик правильно реагує на додавання, видалення та зміну кількості товарів. Також підтверджено працездатність авторизації користувача і доступу до персоналізованих сторінок після входу в систему.

Оформлення замовлення перевірялося як один із ключових сценаріїв. У процесі тестування система передавала дані з кошика, дозволяла заповнити контактну інформацію, обрати параметри доставки та створити замовлення. Інтеграція з Nova Poshta забезпечувала отримання даних для вибору міста й відділення, а інтеграція з Monobank давала можливість сформувати платіжну операцію.

У таблиці 4.3 наведено узагальнені результати тестування системи.

Таблиця 4.3

Узагальнені результати тестування системи

№	Напрям перевірки	Результат тестування	Висновок
1	Каталог товарів	Товари завантажуються та відображаються коректно	Вимогу виконано
2	Пошук і фільтрація	Список товарів оновлюється відповідно до заданих параметрів	Вимогу виконано
3	Картка товару	Відображаються назва, ціна, опис, зображення та характеристики	Вимогу виконано
4	Кошик	Додавання, видалення та зміна кількості товарів працюють правильно	Вимогу виконано
5	Авторизація	Вхід до системи відкриває доступ до персоналізованих сторінок	Вимогу виконано
6	Особистий кабінет	Користувач може переглядати власні дані та замовлення	Вимогу виконано
7	Оформлення замовлення	Дані з кошика передаються до форми, запис створюється в системі	Вимогу виконано
8	Доставка	Дані Nova Poshta використовуються під час вибору міста та відділення	Вимогу виконано
9	Оплата	Система формує платіжну операцію через Monobank	Вимогу виконано
10	Обробка помилкових дій	Некоректні або неповні дані не призводять до аварійного завершення	Вимогу виконано
11	Синхронізація товарних даних	Передбачено оновлення товарів і категорій через зовнішню систему	Вимогу виконано

Під час тестування було виявлено, що основні сценарії працюють стабільно, однак окремі частини системи потребують подальшого вдосконалення. Насамперед це стосується розширення обробки помилок під час взаємодії із зовнішніми сервісами, уніфікації окремих структур даних між клієнтською та серверною частинами, а також подальшого покращення адміністративних можливостей системи.

Ці зауваження не скасовують працездатності основного користувацького сценарію, але можуть бути враховані під час подальшого розвитку проєкту. Проведене тестування підтвердило, що система загалом відповідає поставленим функціональним вимогам і забезпечує повний цикл роботи інтернет-магазину: від перегляду товару до оформлення й оплати замовлення.

Висновки до розділу 4.

У четвертому розділі розроблено методику тестування та перевірено основні функціональні модулі web-сайту BiTZone. Тестування охопило перегляд головної сторінки, роботу каталогу, пошук, фільтрацію, відкриття картки товару, додавання товарів до кошика і списку бажань, реєстрацію, авторизацію, оформлення замовлення, вибір доставки, оплату, роботу особистого кабінету, відгуки та взаємодію із зовнішніми сервісами.

Результати тестування показали, що основні сценарії функціонують коректно і система забезпечує повний цикл роботи інтернет-магазину: від пошуку та вибору товару до створення замовлення, вибору способу доставки, виконання оплати та подальшого перегляду інформації користувачем. Виявлені зауваження стосуються переважно подальшого вдосконалення обробки помилок і розширення окремих інтеграцій, але не порушують працездатність основної функціональності системи.

ВИСНОВКИ

У кваліфікаційній роботі було розглянуто процес розробки web-сайту інтернет-магазину відеогрових консолей BiTZone з повним циклом функціонування. У межах роботи досліджено предметну галузь онлайн-продажу відеогрових консолей і супутніх товарів, визначено особливості взаємодії користувача з таким типом інтернет-магазину та обґрунтовано доцільність створення спеціалізованого програмного рішення.

У першому розділі було проаналізовано предметну галузь, бізнес-процес функціонування інтернет-магазину та існуючі програмні рішення для організації електронної комерції. Розгляд таких платформ, як Shopify, WooCommerce, Adobe Commerce та OpenCart, дав змогу визначити їх переваги й обмеження. Було встановлено, що готові рішення забезпечують базові можливості електронної комерції, проте не завжди дають змогу зручно поєднати каталог, кошик, список бажань, особистий кабінет, оформлення замовлення, доставку, оплату та синхронізацію товарних даних у межах одного спеціалізованого сценарію. Саме це підтвердило актуальність розробки власного web-сайту для магазину відеогрових консолей.

У другому розділі виконано проєктування системи. Було сформовано функціональні та нефункціональні вимоги, змодельовано основні сценарії взаємодії користувача із системою, спроєктовано клієнт-серверну архітектуру, структуру бази даних та користувацький інтерфейс. Окрему увагу приділено повному циклу замовлення: від перегляду товару в каталозі до оформлення покупки, вибору доставки, переходу до оплати та перегляду замовлення в особистому кабінеті. Побудовані діаграми дозволили наочно відобразити ролі користувачів, структуру системи, зв’язки між основними сутностями та логіку користувацького маршруту.

У третьому розділі описано реалізацію системи. Клієнтську частину реалізовано з використанням React, React Router, Redux Toolkit, styled-

components, framer-motion та axios. Серверну частину побудовано на основі Node.js, Express, MongoDB та Mongoose. У системі реалізовано каталог товарів, пошук, фільтрацію, картку товару, кошик, список бажань, реєстрацію та авторизацію користувача, особистий кабінет, роботу із замовленнями та відгуками. Також описано інтеграцію із зовнішніми сервісами: Nova Poshta для вибору доставки, Monobank для організації оплати та RoApp для синхронізації товарів і категорій.

У четвертому розділі проведено тестування програмної системи. Було перевірено основні функціональні модулі: каталог, пошук, фільтрацію, картку товару, кошик, список бажань, авторизацію, оформлення замовлення, доставку, оплату та особистий кабінет. Результати тестування показали, що система коректно виконує основні функції інтернет-магазину та забезпечує проходження ключового користувацького сценарію. Окремо було враховано ситуації з некоректними або неповними даними, а також перевірено взаємодію із зовнішніми сервісами.

У результаті виконання роботи було досягнуто поставленої мети - підвищення зручності та ефективності процесу онлайн-замовлення відеогрових консолей і супутніх товарів за рахунок розробки web-сайту інтернет-магазину з повним циклом функціонування. Практична цінність роботи полягає в тому, що створене рішення може бути використане як основа для подальшого розвитку спеціалізованого інтернет-магазину, орієнтованого на продаж відеогрових консолей, аксесуарів та супутньої продукції.

Розроблена система поєднує основні етапи роботи інтернет-магазину в одному програмному рішенні: перегляд асортименту, вибір товару, формування кошика, оформлення замовлення, доставку, оплату та подальший перегляд інформації в кабінеті користувача.

Подальший розвиток проєкту може бути пов’язаний із розширенням адміністративної частини, покращенням аналітики замовлень, удосконаленням обробки помилок під час роботи із зовнішніми сервісами. Незважаючи на це, реалізована система вже демонструє працездатність основних модулів і може

розглядатися як завершений програмний прототип спеціалізованого інтернет-магазину.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Кисіль В.О., Коваленко Д.С. Визначення вимог інтернет магазину для комерційного використання. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.468 -471.

2. Кисіль В.О., Коваленко Д.С. Забеспечення конфіденціальності та захису данних у WEB-застосунку для онлайн-торгівлі. Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2026, С.287 -289.

ПЕРЕЛІК ПОСИЛАНЬ

1. Краус К. М., Краус Н. М., Манжура О. В. Електронна комерція та Інтернет-торгівля: навч.-метод. посіб. Київ: Аграр Медіа Груп, 2021. 454 с.

2. Швиденко М. З., Касаткіна О. М., Швиденко О. М. Електронна комерція: підручник. Київ: ФОП Ямчинський О. В., 2020. 478 с.

3. Laudon K. C., Traver C. G. E-commerce 2023-2024: Business, Technology, Society. 18th ed. Harlow: Pearson, 2023. 912 p.

4. Chaffey D., Hemphill T., Edmundson-Bird D. Digital Business and E-Commerce Management. 7th ed. Harlow: Pearson, 2019. 680 p.

5. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner’s Approach. 9th ed. New York: McGraw-Hill Education, 2020. 672 p.

6. Sommerville I. Software Engineering. 10th ed. Boston: Pearson, 2016. 816 p.

7. ISO/IEC/IEEE 29148:2018. Systems and software engineering - Life cycle processes - Requirements engineering. Geneva: ISO, 2018.

8. ISO/IEC/IEEE 12207:2017. Systems and software engineering - Software life cycle processes. Geneva: ISO, 2017.

9. ISO/IEC 25010:2023. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model. Geneva: ISO, 2023.

10. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Hoboken: John Wiley & Sons, 2012. 256 p.

11. Krug S. Don’t Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3rd ed. Berkeley: New Riders, 2014. 216 p.

12. Garrett J. J. The Elements of User Experience: User-Centered Design for the Web and Beyond. 2nd ed. Berkeley: New Riders, 2011. 192 p.

13. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2002. 560 p.

14. Richardson L., Amundsen M., Ruby S. RESTful Web APIs. Sebastopol: O’Reilly Media, 2013. 406 p.
15. React. Quick Start. URL: https://react.dev/learn (дата звернення: 11.05.2026).
16. Redux Toolkit. Redux Toolkit Documentation. URL: https://redux-toolkit.js.org/ (дата звернення: 11.05.2026).
17. Node.js. Node.js Documentation. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 11.05.2026).
18. Express. Express - Node.js web application framework. URL: https://expressjs.com/ (дата звернення: 11.05.2026).
19. MongoDB. MongoDB Manual. URL: https://www.mongodb.com/docs/manual/ (дата звернення: 11.05.2026).
20. Mongoose. Mongoose ODM Documentation. URL: https://mongoosejs.com/docs/ (дата звернення: 11.05.2026).
21. Shopify Help Center. URL: https://help.shopify.com/en (дата звернення: 11.05.2026).
22. WooCommerce Documentation. URL: https://woocommerce.com/documentation/woocommerce/.
23. Adobe Commerce Documentation. URL: https://experienceleague.adobe.com/en/docs/commerce (дата звернення: 11.05.2026).
24. OpenCart Documentation. URL: https://docs.opencart.com/ (дата звернення: 11.05.2026).
25. Nova Post. Automating and integrating your business’s logistics processes. URL: https://novaposhta.ua/en/for-business/cooperation/integration/ (дата звернення: 11.05.2026).
26. monobank. Acquiring API Documentation. URL: https://api.monobank.ua/docs/acquiring.html (дата звернення: 11.05.2026).
27. RO App. API Reference: Public API RO App. URL: https://roappua.readme.io/reference/getting-started-with-your-api (дата звернення: 11.05.2026).

ДОДАТОК А

ПРЕЗЕНТАЦІЯ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-сайту інтернет-магазину відеогрових консолей “BiTZone” з повним циклом функціонування

Виконав студент 4 курсу
групи ПД-42
Кисіль Владислав Олександрович
Керівник роботи
старш. викл. каф., доктор філософії, Коваленко Данило Сергійович

Київ – 2026

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – підвищення ефективності онлайн-замовлення відеогрових консолей і супутніх товарів шляхом розробки web-сайту інтернет-магазину з повним циклом функціонування та інтеграцією зовнішніх сервісів.

Об’єкт дослідження – процес онлайн-продажу відеогрових консолей і супутніх товарів.

Предмет дослідження – програмні засоби автоматизації керування каталогом товарів, замовленнями, доставкою, оплатою та взаємодією користувача з web-сайтом інтернет-магазину.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Дослідити особливості онлайн-продажу відеогрових консолей і супутніх товарів.
2. Проаналізувати існуючі рішення електронної комерції та визначити їх сильні й слабкі сторони.
3. Обґрунтувати вибір засобів і технологій для реалізації системи.
4. Сформувати функціональні та нефункціональні вимоги до web-сайту.
5. Спроєктувати архітектуру, структуру даних та сценарії взаємодії користувача з web-сайтом.
6. Реалізувати клієнтську та серверну частини web-сайту.
7. Забезпечити інтеграцію з службою доставки “Нова Пошта”, платіжною системою “MonoPay” та CRM “RoApp”.
8. Провести тестування каталогу, пошуку, кошика, wishlist, авторизації, замовлення, доставки, оплати, відгуків та синхронізації.

3

АНАЛІЗ АНАЛОГІВ

Характеристика	Shopify	WooCommerce	Adobe Commerce	OpenCart	BiTZone
Каталог товарів, кошик та оформлення замовлення	+	+	+	+	+
Можливість глибокої зміни серверної логіки під конкретний бізнес-процес	–	+	+	+	+
SPA-інтерфейс із централізованим керуванням станом кошика, wishlist та авторизації	–	–	–	–	+
Інтеграція доставки та оплати безпосередньо в логіку оформлення замовлення	+	+	+	+	+
Синхронізація товарів і категорій із зовнішньою системою обліку через API та webhook	–	–	–	–	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1. Перегляд структурованого каталогу товарів.
2. Пошук, фільтрація та сортування товару.
3. Перегляд картки товару з можливістю дізнатись характеристики.
4. Наповнення кошику та оформлення списку бажань.
5. Реєстрація та авторизація користувача.
6. Оформлення замовлення.
7. Здійснення оплати замовлення
8. Підтвердження способу та місця доставки товару.

Нефункціональні вимоги

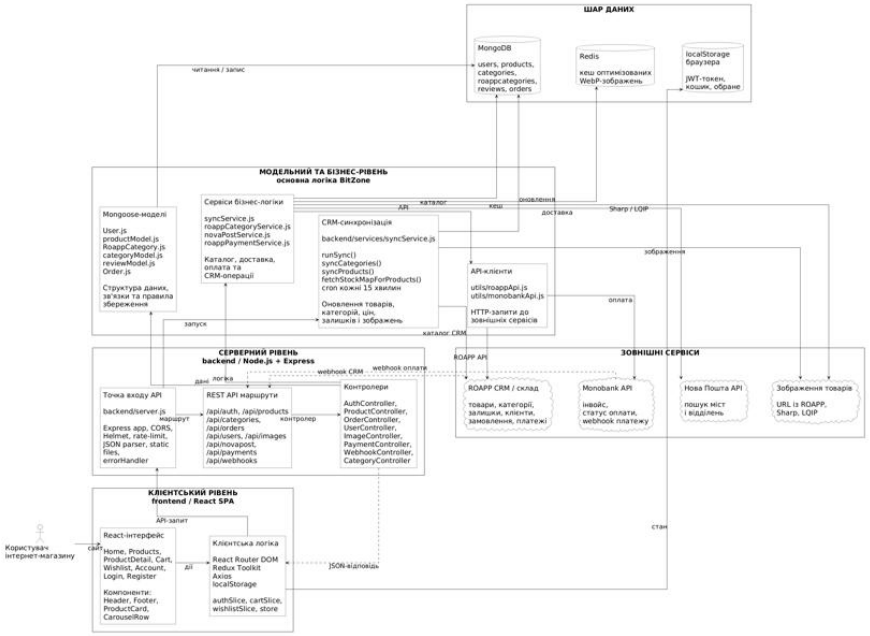
1. Інтерфейс має коректно працювати на екранах від 360 px до 1920 px по ширині.
2. JWT-токен має мати строк дії до 24 годин, паролі зберігаються лише у хешованому вигляді.
3. Повідомлення про помилку має з'являтися не пізніше ніж за 1 секунду.
4. API-запити мають виконуватися не довше ніж за 2 секунди.
5. Дані на сторінці мають оновлюватися без перезавантаження за 1–2 секунди.
6. Основний сценарій користувача має виконуватися не більше ніж за 5 кроків.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

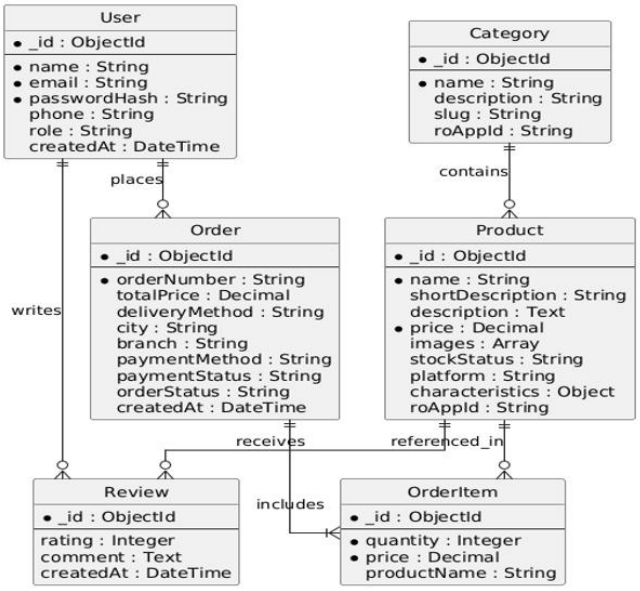
6

АРХІТЕКТУРА КЛІЄНТ-СЕРВЕРНОЇ ВЗАЄМОДІЇ



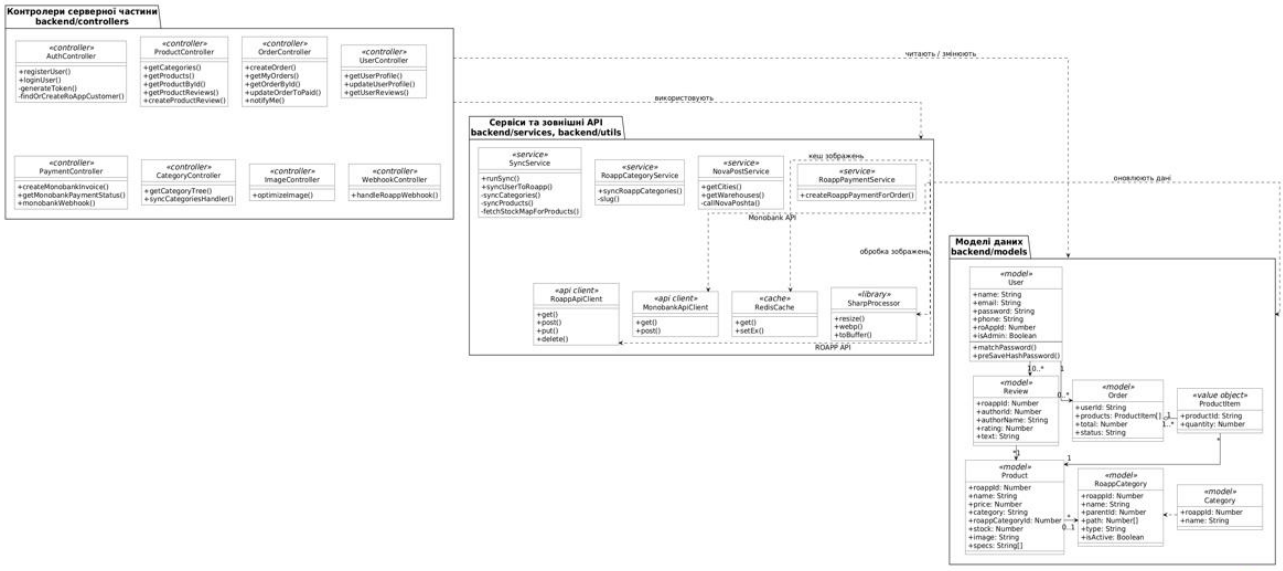
7

ОСНОВНІ СУТНОСТІ ТА ЗВ'ЯЗКИ БАЗИ ДАНИХ



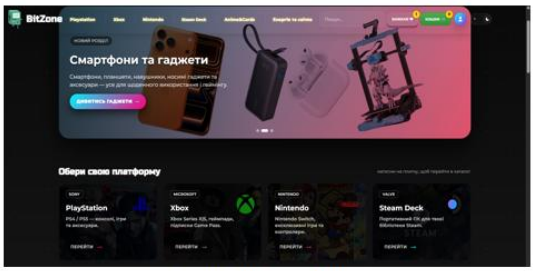
8

ДІАГРАМА КЛАСІВ СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ

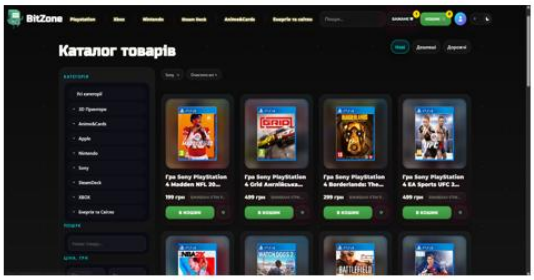


9

ЕКРАННІ ФОРМИ

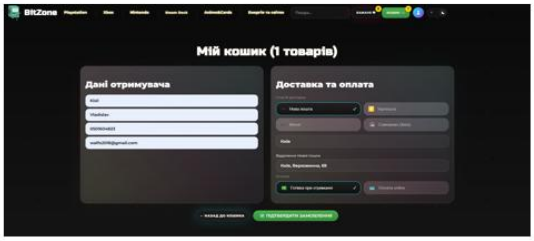


Головна сторінка



Каталог товарів

Картка товару



Кошик та оформлення замовлення

10

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Кисіль В.О., Коваленко Д.С. Визначення вимог інтернет магазину для комерційного використання. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.468 -471.

2. Кисіль В.О., Коваленко Д.С. Забезпечення конфіденціальності та захису данних у WEB-застосунку для онлайн-торгівлі. Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез. *(подано до друку).*

ВИСНОВКИ

1. Досліджено предметну галузь онлайн-продажу відеогрових консолей. Виявлено, що для такого магазину критичними є зручний каталог, картка товару, кошик, wishlist, оформлення замовлення, доставка, оплата та особистий кабінет.
2. Проаналізовано Shopify, WooCommerce, Adobe Commerce та OpenCart. Їх перевагами є готові e-commerce-модулі, а недоліками — обмежена адаптація під спеціалізований бізнес-процес, залежність від сторонніх розширень і недостатня гнучкість інтеграцій.
3. Для реалізації обрано React, Redux Toolkit, Node.js, Express, MongoDB, Mongoose, JWT, API «Нової Пошти», MonoPay / Monobank API та RoApp, що дозволило побудувати повноцінну клієнт-серверну систему.
4. Сформовано вимоги до системи: каталог, пошук, фільтрація, картка товару, кошик, wishlist, авторизація, особистий кабінет, замовлення, відгуки, доставка, оплата, синхронізація товарів, адаптивність, безпека й стабільність API.
5. Спроєктовано клієнт-серверну архітектуру, структуру даних на основі MongoDB та сценарії взаємодії користувача з web-сайтом: від вибору товару до завершення замовлення.
6. Реалізовано frontend на React і backend на Node.js/Express. Серверна частина містить REST API, контролери, Mongoose-моделі, middleware, сервіси бізнес-логіки та API-клієнти зовнішніх сервісів.
7. Інтегровано «Нову Пошту» для доставки, MonoPay / Monobank API для оплати та RoApp для синхронізації товарів, категорій, залишків, клієнтів і замовлень.
8. Проведено тестування каталогу, пошуку, кошика, wishlist, авторизації, замовлення, доставки, оплати, відгуків та синхронізації. Перевірка підтвердила працездатність системи й відповідність основним вимогам.

ДОДАТОК Б
ЛІСТИНГИ ОСНОВНИХ ПРОГРАМНИХ МОДУЛІВ СИСТЕМИ BITZONE

Лістинг А.1 - налаштування серверного застосунку та підключення API-маршрутів

Файл проєкту: backend/server.js

```
const path = require('path');
const express = require('express');
const cors = require('cors');
const dotenv = require('dotenv');
const helmet = require('helmet');
const connectDB = require('./config/db');
const { errorHandler } =
require('./middleware/errorMiddleware');
const rateLimit = require('express-rate-limit');

dotenv.config();
connectDB();
require('./services/syncService');

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

app.set('trust proxy', 1);
app.use(helmet({ crossOriginResourcePolicy: false }));
app.use(cors());
app.use(express.json({ limit: '10kb' }));
app.use(express.urlencoded({ extended: true, limit: '10kb' }));

const authLimiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 10,
  standardHeaders: true,
  legacyHeaders: false,
  message: {
    message: 'Забагато спроб входу з цієї IP-адреси, будь
ласка, спробуйте знову через 15 хвилин'
  }
});

app.use('/api/categories', require('./routes/categoryRoutes'));
app.use('/api/novapost', require('./routes/novaPostRoutes'));
app.use('/api/auth', authLimiter, require('./routes/authRoutes'));
app.use('/api/products', require('./routes/productRoutes'));
app.use('/api/users', require('./routes/userRoutes'));
app.use('/api/orders', require('./routes/orderRoutes'));
app.use('/api/images', require('./routes/imageRoutes'));
app.use('/api/webhooks', require('./routes/webhookRoutes'));
app.use('/api/payments', require('./routes/paymentRoutes'));

app.use(errorHandler);

app.listen(PORT, () => {
  console.log(`Сервер успішно запущено на порту
http://localhost:${PORT}`);
});
```

Лістинг А.2 - модель товару в базі даних

Файл проєкту: backend/models/productModel.js

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

const productSchema = new mongoose.Schema(
  {
    roappId: {
      type: Number,
      required: true,
      unique: true,
      index: true,
    },
    name: {
      type: String,
      required: true,
      trim: true,
    },
    price: {
      type: Number,
      default: 0,
    },
    category: {
      type: String,
      default: 'Різне',
    },
    roappCategoryId: {
      type: Number,
      index: true,
      default: null,
    },
    description: {
      type: String,
      default: '',
    },
    image: {
      type: String,
      default: null,
    },
    images: {
      type: [String],
      default: [],
    },
    stock: {
      type: Number,
      default: 0,
    },
    createdAtRoapp: {
      type: Date,
    },
    lqip: {
      type: String,
      default: null,
    },
    specs: {
      type: [String],
      default: [],
    },
  },
  { timestamps: true }
);

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

Лістинг А.3 - захист маршрутів через JWT-автентифікацію

Файл проєкту: backend/middleware/authMiddleware.js

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

```
const asyncHandler = require('express-async-handler');
const User = require('../models/User');

const authMiddleware = asyncHandler(async (req, res, next) =>
{
  let token;

  if (req.headers.authorization &&
req.headers.authorization.startsWith('Bearer')) {
    try {
      token = req.headers.authorization.split(' ')[1];
      const decoded = jwt.verify(token,
process.env.JWT_SECRET);
      req.user = await User.findOne({ roAppId:
decoded.id }).select('-password');

      if (!req.user) {
        res.status(401);
        throw new Error('Not authorized, token failed (user not
found)');
      }
      next();
    } catch (error) {
      console.error(error);
      res.status(401);
      throw new Error('Not authorized, token failed');
    }
  }

  if (!token) {
    res.status(401);
    throw new Error('Not authorized, no token');
  }
});

const optionalAuthMiddleware = asyncHandler(async (req, res,
next) => {
  let token;
  if (req.headers.authorization &&
req.headers.authorization.startsWith('Bearer')) {
    try {
      token = req.headers.authorization.split(' ')[1];
      const decoded = jwt.verify(token,
process.env.JWT_SECRET);
      req.user = await User.findOne({ roAppId:
decoded.id }).select('-password');
    } catch (error) {
      req.user = null;
    }
  } else {
    req.user = null;
  }
  next();
});

module.exports = { authMiddleware,
optionalAuthMiddleware };
```

Лістинг А.4 - отримання каталогу товарів із фільтрацією та сортуванням

Файл проєкту: backend/controllers/productController.js

```
const getProducts = asyncHandler(async (req, res) => {
  const { category: categoryId, search, page = 1, sort, minPrice,
maxPrice, types, platforms } = req.query;

  const limit = 20;
  const pageNum = Number(page) || 1;
  const skip = (pageNum - 1) * limit;
  const queryConditions = [];

  const priceFilter = {};
  if (minPrice && !isNaN(parseFloat(minPrice)))
priceFilter.$gte = parseFloat(minPrice);
  if (maxPrice && !isNaN(parseFloat(maxPrice)))
priceFilter.$lte = parseFloat(maxPrice);
  if (Object.keys(priceFilter).length > 0)
queryConditions.push({ price: priceFilter });

  if (search) {
    queryConditions.push({
      $or: [
        { name: new RegExp(search, 'i') },
        { description: new RegExp(search, 'i') },
        { category: new RegExp(search, 'i') },
      ],
    });
  }

  const match = queryConditions.length > 0 ? { $and:
queryConditions } : {};
  const total = await Product.countDocuments(match);

  const sortStage = { isOutOfStock: 1 };
  switch (sort) {
    case 'price-asc': sortStage.price = 1; break;
    case 'price-desc': sortStage.price = -1; break;
    case 'name-asc': sortStage.name = 1; break;
    case 'name-desc': sortStage.name = -1; break;
    default: sortStage.createdAtRoapp = -1; break;
  }
  sortStage._id = 1;

  const products = await Product.aggregate([
    { $match: match },
    { $addFields: { isOutOfStock: { $cond: [{ $lte: [{ $ifNull:
['$stock', 0] }, 0] }, 1, 0] } } },
    { $sort: sortStage },
    { $skip: skip },
    { $limit: limit },
  ]);

  res.json({
    products: products.map((p) => ({ ...p, _id: p.roappId })),
    total,
  });
});
```

Лістинг А.5 - створення замовлення і передача товарних позицій до RO App

Файл проєкту: backend/controllers/orderController.js

```
const createOrder = asyncHandler(async (req, res) => {
  const { customerData, cartItems } = req.body;

  if (!cartItems || cartItems.length === 0) {
    res.status(400);
    throw new Error('Неможливо створити замовлення без
товарів');
  }
  if (!customerData || !customerData.phone) {
    res.status(400);
    throw new Error('Відсутні дані клієнта або телефон');
  }

  let customerId;
  if (req.user && typeof req.user.roAppId === 'number') {
    customerId = req.user.roAppId;
  } else {
    const searchResponse = await roappApi.get('contacts/people',
{
      params: { 'phones[]': customerData.phone },
    });
```

```
    if (searchResponse.data?.data?.length > 0) {
      customerId = searchResponse.data.data[0].id;
    } else {
      const newCustomerPayload = {
        first_name: customerData.firstName,
        last_name: customerData.lastName,
        phones: [{ title: 'Основний', phone: customerData.phone,
notify: false }],
        emails: customerData.email ? [{ title: 'Основний', email:
customerData.email, notify: false }] : [],
      };
      const createCustomerResponse = await
roappApi.post('contacts/people', newCustomerPayload);
      customerId = createCustomerResponse.data.id;
    }
  }

  const roappOrderPayload = {
    client_id: customerId,
    branch_id: MY_BRANCH_ID,
    order_type_id: MY_ORDER_TYPE_ID,
    assignee_id: MY_ASSIGNEE_ID,
    due_date: new Date().toISOString(),
    malfunction: 'Замовлення з сайту BitZone',
  };

  const { data } = await roappApi.post('orders',
roappOrderPayload);
  const orderId = data.id;
  const orderNumber = data.number || data.order_number ||
data.id;

  for (const item of cartItems) {
    await roappApi.post(`orders/${orderId}/items`, {
      entity_id: item.id,
      quantity: item.qty,
      price: item.price,
      assignee_id: MY_ASSIGNEE_ID,
      cost: 0,
      discount: { type: 'value', percentage: 0, amount: 0, sponsor:
'staff' },
      warranty: { period: '0', periodUnits: 'months' },
    });
  }

  res.status(201).json({ id: orderId, orderId, orderNumber });
});
```

Лістинг А.6 - створення платіжної сесії Monobank

Файл проєкту: backend/controllers/paymentController.js

```
const createMonobankInvoice = asyncHandler(async (req, res)
=> {
  const { orderId, amount } = req.body;

  if (!orderId || !amount) {
    return res.status(400).json({ ok: false, message: 'Потрібно
передати orderId та amount.' });
  }

  const orderIdStr = String(orderId);
  const FRONTEND_URL = process.env.FRONTEND_URL ||
process.env.BASE_CLIENT_URL || 'https://bitzone.com.ua';
  const redirectUrl = `${FRONTEND_URL.replace(/\/$/,
'')}/payment-
result?orderId=${encodeURIComponent(orderIdStr)}`;

  let webHookUrl =
process.env.MONOBANK_WEBHOOK_URL?.trim() ||
undefined;
  if (!webHookUrl &&
process.env.BACKEND_PUBLIC_URL) {
    webHookUrl =
`${process.env.BACKEND_PUBLIC_URL.replace(/\/$/,
'')}/api/payments/monobank/webhook`;
  }

  const payload = {
    amount,
    ccy: 980,
    merchantPaymInfo: {
      reference: orderIdStr,
      destination: `Оплата замовлення №${orderIdStr} на
BitZone`,
      comment: `Оплата замовлення №${orderIdStr} на
BitZone`,
    },
    redirectUrl,
    ...(webHookUrl ? { webHookUrl } : {}),
  };

  const { data } = await monobankApi.post('invoice/create',
payload);
  if (!data || !data.invoiceId || !data.pageUrl) {
    return res.status(502).json({ ok: false, message: 'Monobank
повернув неочікувану відповідь.' });
  }

  monobankInvoices.set(orderIdStr, {
    orderId: orderIdStr,
    invoiceId: data.invoiceId,
    amount,
    ccy: 980,
    status: 'created',
    roappPaymentCreated: false,
    lastUpdate: new Date(),
  });

  return res.json({ ok: true, invoiceId: data.invoiceId, pageUrl:
data.pageUrl });
});
```

Лістинг А.7 - маршрутизація сторінок клієнтської частини

Файл проєкту: frontend/src/App.js

```
import React, { Suspense } from 'react';
import { Routes, Route } from 'react-router-dom';
import Header from './components/Header';
import Footer from './components/Footer';
import PaymentResult from './pages/PaymentResult';

const Home = React.lazy(() => import('./pages/Home'));
const Products = React.lazy(() => import('./pages/Products'));
const ProductDetail = React.lazy(() =>
import('./pages/ProductDetail'));
const Cart = React.lazy(() => import('./pages/Cart'));
const Login = React.lazy(() => import('./pages/Login'));
const Register = React.lazy(() => import('./pages/Register'));
const Wishlist = React.lazy(() => import('./pages/Wishlist'));
const Account = React.lazy(() => import('./pages/Account'));
const OrderDetail = React.lazy(() =>
import('./pages/OrderDetail'));

export default function App() {
  return (
    <div className="app">
      <Header />
      <main className="container" style={{ minHeight:
'70vh' }}>
        <Suspense fallback={<p className="p">Завантаження
сторінки...</p>}>
          <Routes>
```

```
          <Route path="/payment-result"
element={<PaymentResult />} />
          <Route path="/" element={<Home />} />
          <Route path="/products" element={<Products />} />
          <Route path="/product/:id" element={<ProductDetail
/>} />
          <Route path="/cart" element={<Cart />} />
          <Route path="/wishlist" element={<Wishlist />} />
          <Route path="/login" element={<Login />} />
          <Route path="/register" element={<Register />} />
          <Route path="/account" element={<Account />} />
          <Route path="/account/orders/:orderId"
element={<OrderDetail />} />
        </Routes>
       </Suspense>
     </main>
     <Footer />
   </div>
  );
}
```

Лістинг А.8 - керування станом кошика через Redux Toolkit

Файл проєкту: frontend/src/redux/cartSlice.js

```
import { createSlice } from '@reduxjs/toolkit';

const CART_KEY = 'bitzone_cart_v1';
const loadCart = () => {
  try { return JSON.parse(localStorage.getItem(CART_KEY)) ||
{ items: [] }; }
  catch { return { items: [] }; }
};
const saveCart = (state) => localStorage.setItem(CART_KEY,
JSON.stringify({ items: state.items }));

const cartSlice = createSlice({
  name: 'cart',
  initialState: loadCart(),
  reducers: {
    addToCart: (state, { payload }) => {
      const { id, name, price, image } = payload;
      const item = state.items.find((i) => i.id === id);
      if (item) item.qty += 1;
      else state.items.push({ id, name, price, image: image || '',
qty: 1 });
      saveCart(state);
    },
    removeFromCart: (state, { payload }) => {
      state.items = state.items.filter((i) => i.id !== payload);
      saveCart(state);
    },
    increaseQty: (state, { payload }) => {
      const item = state.items.find((i) => i.id === payload);
      if (item) item.qty += 1;
      saveCart(state);
    },
    decreaseQty: (state, { payload }) => {
      const item = state.items.find((i) => i.id === payload);
      if (item && item.qty > 1) item.qty -= 1;
      else state.items = state.items.filter((i) => i.id !== payload);
      saveCart(state);
    },
    clearCart: (state) => {
      state.items = [];
      saveCart(state);
    },
  },
});

export const { addToCart, removeFromCart, increaseQty,
decreaseQty, clearCart } = cartSlice.actions;
export const selectCartItems = (state) => state.cart.items;
export const selectCartTotal = (state) =>
state.cart.items.reduce((sum, i) => sum + i.price * i.qty, 0);
export default cartSlice.reducer;
```

Лістинг А.9 - оформлення замовлення на клієнтській частині

Файл проєкту: frontend/src/pages/Cart.js

```
const handleCheckout = async () => {
  if (checkoutStep === 'cart') {
    setCheckoutStep('form');
    return;
  }

  if (!validateForm() || isSubmitting) return;
  setIsSubmitting(true);

  const normalizedFormData = {
    ...formData,
    phone: normalizePhoneNumber(formData.phone),
    payment: mapPaymentForBackend(formData.payment),
  };

  const orderPayload = {
    customerData: normalizedFormData,
    cartItems: items.map((item) => ({
      id: item.id,
      name: item.name,
      price: item.price,
      qty: item.qty,
      image: item.image,
    })),
  };

  try {
    const config = { headers: {} };
    if (token) config.headers.Authorization = `Bearer ${token}`;

    const res = await axios.post('/api/orders', orderPayload,
config);
    const backendOrderId = res?.data?.orderId || res?.data?.id ||
res?.data?.orderNumber;

    if (normalizedFormData.payment === 'card' &&
backendOrderId) {
      const invoiceRes = await axios.post(
        '/api/payments/monobank/invoice',
        { orderId: backendOrderId, amount: Math.round(total *
100) },
        config
      );

      if (invoiceRes?.data?.ok && invoiceRes?.data?.pageUrl) {
        window.location.href = invoiceRes.data.pageUrl;
        return;
      }
    }

    dispatch(clearCart());
    setCheckoutStep('success');
  } catch (err) {
    const errorMessage = err.response?.data?.message || 'Не
вдалося оформити замовлення.';
    alert(errorMessage);
  } finally {
    setIsSubmitting(false);
  }
};
```

Лістинг А.10 - перевірка результату оплати після повернення користувача

Файл проєкту: frontend/src/pages/PaymentResult.js

```
useEffect(() => {
  const params = new URLSearchParams(location.search);
  const orderId = params.get('orderId');

  if (!orderId) {
    setState({
      loading: false,
      success: false,
      status: null,
      amount: null,
      error: 'Не передано номер замовлення.',
      orderId: null,
    });
    return;
  }

  const checkPayment = async () => {
    try {
      const { data } = await
axios.get('/api/payments/monobank/status', {
        params: { orderId },
      });
      if (data.ok && data.paid && data.status === 'success') {
        dispatch(clearCart());
        setState({ loading: false, success: true, status: data.status,
amount: data.amount, error: null, orderId });
      } else {
        setState({ loading: false, success: false, status: data.status
|| 'not_found', amount: data.amount || null, error: null, orderId });
      }
    } catch (err) {
      setState({
        loading: false,
        success: false,
        status: null,
        amount: null,
        error: 'Не вдалося перевірити оплату. Спробуйте
оновити сторінку або зв'язатися з нами.',
        orderId,
      });
    }
  };

  checkPayment();
}, [location.search, dispatch]);
```