ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЇ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система управління контентом інтернет-магазину з функціоналом адміністрування каталогу, замовлень та користувачів»

на здобуття освітнього ступеня бакалавра (магістра)
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми Технології цифрового розвитку

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

_____ Роман КІЧ
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. ТЦР-41
Роман КІЧ

Керівник:
канд.техн.наук., доц.каф. Андрій АРОНОВ

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Технології цифрового розвитку

Ступінь вищої освіти бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедрою
Технології цифрового розвитку
_____Вікторія ЖЕБКА
«___» _____2026р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кіч Роман Олександрович

1. Тема кваліфікаційної роботи: «Система управління контентом інтернет-магазину з функціоналом адміністрування каталогу, замовлень та користувачів»

керівник кваліфікаційної роботи Андрій АРОНОВ канд. техн. наук, доцент кафедри ТЦР затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» 02 2026р. № 51.

2. Строк подання кваліфікаційної роботи «18» 05 2026р.

3. Вихідні дані до кваліфікаційної роботи: розробка CMS інтернет-магазину (REST API, SPA) на базі PHP 8.2, MySQL 8.0 та Vue.js 3; адміністрування каталогу, замовлень і користувачів; забезпечення безпеки та цілісності даних (JWT, JSON Snapshot).

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
    1. Аналіз предметної області та існуючих систем управління контентом (CMS) для систем електронної комерції
    2. Проектування архітектури системи, реляційної бази даних та алгоритмів клієнт-серверної взаємодії
    3. Програмна реалізація серверної (Backend) та клієнтської (Frontend) частин системи, розробка модулів управління і тестування платформи.

5. Перелік ілюстративного матеріалу: *презентація*

1. Мета, об’єкт та предмет дослідження.
2. Задачі кваліфікаційної роботи.
3. Аналіз існуючих аналогів систем електронної комерції.
4. Програмні засоби реалізації та вимоги до системи.
5. Структура бази даних (ER-діаграма).
6. Архітектурний потік MVC та діаграма класів.
7. Практичний результат (екранні форми адмін-панелі).
8. Апробація результатів роботи.
9. Висновки.

6. Дата видачі завдання «20» лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області, огляд літератури та існуючих систем електронної комерції.	20.02.2026 - 10.03.2026	
2	Проєктування архітектури системи (REST API) та реляційної бази даних (ER-діаграма).	11.03.2026 - 25.03.2026	
3	Розробка серверної частини (Backend) на PHP: налаштування PDO, маршрутизації та бази даних MySQL.	26.03.2026 - 10.04.2026	
4	Впровадження системи безпеки: JWT-авторизація, хешування паролів та розмежування прав доступу.	11.04.2026 - 18.04.2026	
5	Створення клієнтського інтерфейсу (Frontend) на Vue.js у форматі SPA (каталог, замовлення, Dashboard).	19.04.2026 - 25.04.2026	
6	Інтеграція Frontend та Backend частин, тестування функціоналу та виправлення помилок.	26.04.2026 - 01.05.2026	
7	Оформлення роботи: вступ, висновки, реферат	02.05.2026 - 08.05.2026	
8	Попередніх захист	18.05.2026	

Здобувач(ка) вищої освіти _____ Роман КІЧ
(підпис)

Керівник
кваліфікаційної роботи _____ Андрій АРОНОВ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра : 69 стор., 11 рис., 8 табл., 16 джерел.

Мета роботи – удосконалення процесів адміністрування електронної комерції та підвищення ефективності управління товарним контентом і замовленнями шляхом розробки та впровадження спеціалізованої кастомної системи управління контентом (CMS).

Об’єкт дослідження – процеси управління інформаційним наповненням, замовленнями та клієнтською базою в системах електронної комерції.

Предмет дослідження – методи, алгоритми та програмні засоби розробки адмін-панелі CMS з використанням технологій PHP, MySQL та JavaScript.

Короткий зміст роботи: У кваліфікаційній роботі проаналізовано архітектурні особливості та недоліки монолітних систем управління контентом, обґрунтовано доцільність переходу до розділеної (Headless) архітектури. Спроєктовано фізичну модель реляційної бази даних MySQL із застосуванням патерну денормалізації (JSON Snapshot) для забезпечення цілісності фінансової історії замовлень. Розроблено серверну логіку (Backend) на мові PHP з архітектурою REST API та інтегровано систему захисту сесій за допомогою JWT-авторизації. Створено реактивний та адаптивний користувацький інтерфейс адмін-панелі (Frontend) на JavaScript з використанням фреймворку Vue.js у форматі Single Page Application (SPA). Проведено комплексне функціональне тестування розроблених модулів управління каталогом, замовленнями та користувачами.

Галузь використання: Електронна комерція, веб-розробка, автоматизація процесів адміністрування інтернет-магазинів та управління цифровим комтентом.

КЛЮЧОВІ СЛОВА: СИСТЕМА УПРАВЛІННЯ КОНТЕНТОМ, REST API, ЕЛЕКТРОННА КОМЕРЦІЯ, PHP, MYSQL, VUE.JS, SINGLE PAGE APPLICATION, JWT-АВТОРИЗАЦІЯ, JSON SNAPSHOT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – База даних

СУБД – Система управління базами даних

ТЗ – Технічне завдання

API – Application Programming Interface (Інтерфейс програмування застосунків)

CMS – Content Management System (Система управління контентом)

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

ER-діаграма – Entity-Relationship diagram (Діаграма «сутність-зв'язок»)

JSON – JavaScript Object Notation (Текстовий формат обміну даними)

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

MVC – Model-View-Controller (Архітектурний патерн «Модель-Вигляд-Контролер») REST – Representational State Transfer (Архітектурний стиль для розподілених систем)

SKU – Stock Keeping Unit (Артикул / Ідентифікатор товарної позиції)

SPA – Single Page Application (Односторінковий веб-застосунок)

UML – Unified Modeling Language (Уніфікована мова моделювання

ВСТУП

Актуальність теми. Сучасний етап розвитку світової економіки характеризується стрімким масштабуванням та цифровізацією сектору електронної комерції (e-commerce). Зростання обсягів онлайн-продажів, розширення товарних асортиментів інтернет-магазинів та підвищення вимог користувачів до швидкодії веб-ресурсів створюють нові виклики для розробників програмного забезпечення. Ефективність функціонування торговельного майданчика в інтернеті безпосередньо залежить від гнучкості та надійності систем управління контентом (CMS), які використовуються для адміністрування каталогів, обробки замовлень та взаємодії зі співробітниками й клієнтами.

Традиційні монолітні платформи (наприклад, WordPress із плагіном WooCommerce або OpenCart) мають низку архітектурних обмежень. Вони перевантажені надлишковим кодом, жорстко пов'язують користувацький інтерфейс із серверною логікою, мають високі вимоги до апаратних ресурсів серверів та створюють додаткові ризики у сфері інформаційної безпеки. Крім того, у більшості стандартних систем існує проблема порушення цілісності даних при зміні цін або видаленні товарів, що призводить до викривлення фінансової історії вже оформлених замовлень.

Ефективним розв’язанням цих проблем є перехід до розділеної (Headless) архітектури програмних систем, де клієнтська частина (Frontend) повністю відокремлена від серверної логіки (Backend) і взаємодіє з нею виключно через захищений інтерфейс програмування застосунків (REST API). Застосування підходу Single Page Application (SPA) на базі сучасних JavaScript-фреймворків дозволяє мінімізувати мережевий трафік та досягти миттєвого відклику інтерфейсу, що є критично важливим для операційної діяльності менеджерів та адміністраторів інтернет-магазинів.

З огляду на це, розробка спеціалізованої легковагової CMS із кастомною архітектурою на базі мови PHP, що використовує сучасні підходи до захисту сесій (JWT) та збереження цілісності транзакційних даних (патерн Snapshot), є

актуальним науково-практичним завданням, спрямованим на оптимізацію процесів цифрової комерції.

Мета роботи – удосконалення процесів адміністрування електронної комерції та підвищення ефективності управління товарним контентом і замовленнями шляхом розробки та впровадження спеціалізованої кастомної системи управління контентом (CMS).

Для досягнення поставленої мети визначено та розв'язано такі задачі:

1. Проаналізувати архітектурні особливості та недоліки існуючих монолітних систем управління контентом інтернет-магазинів.
2. Обґрунтувати вибір технологічного стеку та розділеної (Headless) архітектури для проєктування платформи.
3. Спроєктувати структуру реляційної бази даних MySQL та розробити алгоритм захисту фінансової історії транзакцій за допомогою патерну денормалізації (JSON Snapshot).
4. Програмно реалізувати серверну логіку (Backend REST API) на мові PHP з інтеграцією механізмів безпеки (PDO, BCrypt, JWT).
5. Розробити реактивний клієнтський інтерфейс адмін-панелі (SPA Frontend) на базі фреймворку Vue.js.
6. Провести функціональне тестування розроблених модулів управління каталогом, замовленнями та користувачами.

Об'єкт дослідження – процеси управління інформаційним наповненням, замовленнями та клієнтською базою в системах електронної комерції.

Предмет дослідження – методи, алгоритми та програмні засоби розробки адмін-панелі CMS з використанням технологій PHP, MySQL та JavaScript.

Методи дослідження. Для розв'язання поставлених задач використано методи системного аналізу предметної області (при формуванні технічних вимог до системи); принципи об'єктно-орієнтованого проєктування та UML-моделювання (при побудові архітектури класів та діаграм послідовності); методи реляційного моделювання баз даних (при оптимізації СУБД MySQL); архітектурний стиль REST та технології асинхронного обміну даними Fetch API

(при реалізації взаємодії між Backend та Frontend).

Практичне значення одержаних результатів. Створено повністю готову до впровадження легковагову систему управління контентом інтернет-магазину, яка функціонує за принципом Headless CMS. Реалізовані програмні модулі дозволяють оптимізувати роботу контент-менеджерів та операторів, забезпечують високу швидкість обробки великих масивів товарів та ведення прозорого журналу аудиту статусів замовлень. Розроблена архітектурна модель збереження складу кошика у форматі JSON унеможливлює викривлення фінансової звітності підприємства. Програмний код системи є гнучким для подальшого масштабування (наприклад, для підключення мобільних застосунків до розробленого REST API).

Галузь застосування та апробація результатів. Сфера електронної комерції, автоматизація процесів адміністрування торговельних веб-майданчиків, веб-розробка.

Основні теоретичні положення та практичні результати кваліфікаційної роботи пройшли успішну апробацію у вигляді доповідей на двох науково-технічних конференціях:

1. Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація» (Тема: «Забезпечення цілісності даних під час обробки транзакцій у веб-системах електронної комерції»).

2. VI Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті» (Тема: «Застосування інтелектуальних методів обробки даних у кастомних системах управління контентом для електронної комерції»).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА СИСТЕМ УПРАВЛІННЯ КОНТЕНТОМ ДЛЯ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

1.1 Сучасний стан та тенденції розвитку систем електронної комерції

З огляду на безпекову ситуацію в країні на фоні тривалої агресії російської федерації проти України, з розвитком ІТ-технологій постійно зростає кількість компаній та суб'єктів господарювання, які мають потребу у власному вебсайті та системі управління контентом для продажу товарів та послуг он-лайн. Інтернет-торгівля представляє собою динамічний процес продажу послуг і товарів за допомогою електронних майданчиків, які дають можливість дистанційно оформити замовлення. В Україні цей процес розвивається досить успішно, обсяги продажів та кількість інтернет-магазинів постійно збільшуються (Рис. 1.1).

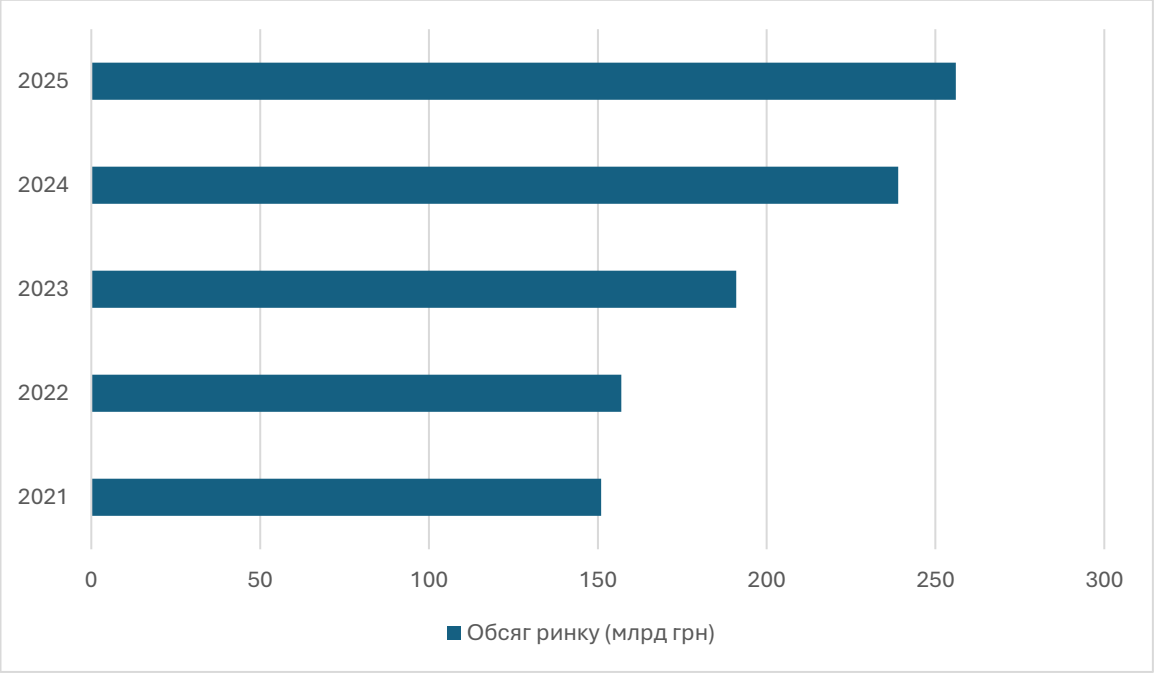


Рис. 1.1 Динаміка обсягів ринку електронної комерції в Україні за 2021–2025 рр

Порівняно високі витрати на традиційну торгівлю з прилавка у фізичній крамниці, що включають орендну плату, заробітну плату персоналу та комунальні послуги, створили ситуацію, коли більшість споживачів перед здійсненням

покупки шукають дешевші товари через мережу Інтернет. Онлайн–магазини не здатні повністю замінити традиційні крамниці, але вони суттєво розширюють загальний ринок та сферу застосування комерційних інструментів. Сайти інтернет-магазинів дозволяють представляти користувачам необхідну, актуальну та переважно правдиву інформацію про товари та послуги. Практично будь-який сучасний суб'єкт господарювання має свій сайт або веб-додаток з можливістю контекстної чи таргетованої реклами в Інтернеті. В умовах зростання конкуренції та розвитку ІТ-технологій, крамницям необхідно будувати інформаційні системи з набором необхідних засобів для успішного ведення бізнесу.

Розвиток Інтернету привів до необхідності створення різнопланових веб-ресурсів для надання послуг та реалізації товарів. Існують як невеликі сайти з базовою інформацією про компанію та послуги, так і великі портали з детальною характеристикою товарів, їхньою ілюстрацією та вартістю, які створені для того, щоб користувач міг знайти детальний опис. Вебсайти електронної комерції виконують функції прилавку, обробки замовлень, каси, продажу товарів та організації їхнього доставлення. Це ефективне поєднання маркетингу з традиційним магазином, де головною відмінною рисою є можливість представлення великої кількості товарів і послуг, надання покупцям значних відомостей для прийняття рішення про укладення договору купівлі-продажу.

Недоліки онлайн-платформ полягають лише в тому, що у звичайному магазині покупці можуть побачити та тактильно оцінити товар, тоді як на вебсайті вони позбавлені цього. Проте, онлайн-продажі стрімко зростають через зручність для покупців, економію часу та грошових коштів. Вебсайт працює цілодобово, а товари можна продавати в автоматичному режимі. У звичайному магазині зона продажу обмежена фізичним простором, а охоплення вебсайту практично не має територіальних меж. Для досягнення успіху необхідно, щоб вебсайт мав практичну навігацію, розширений опис товарів, актуальну контактну інформацію, вчасне оновлення даних про асортимент, сучасний дизайн та високу швидкість обробки замовлень. Будування системи управління контентом є важливим етапом для кожного суб'єкта, який бажає забезпечити ефективне використання своїх

матеріальних ресурсів та надати своїм клієнтам якісну інформацію (Рис. 1.2).

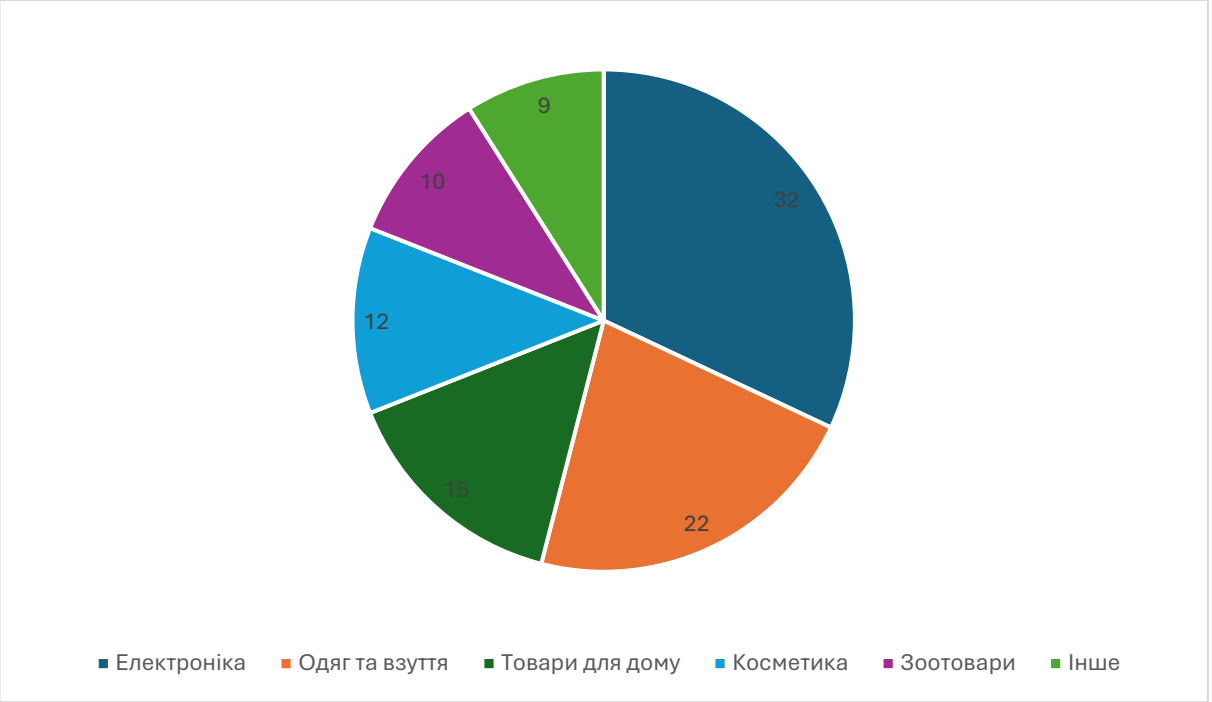


Рис. 1.2 Структура популярності категорій товарів в українському сегменті e-commerce

1.2 Порівняльний аналіз існуючих монолітних та хмарних рішень (CMS)

Система управління наповненням сайту (Content Management System) – це спеціалізоване програмне забезпечення, яке дозволяє автоматизувати процеси створення, редагування та публікації контенту. Вона надає можливість різним користувачам оновлювати вебсайт без наявних спеціалізованих знань програмування та веб-дизайну. Основними перевагами традиційних CMS є:

1. Забезпечення єдиного стилю та форматування контенту на сайті для його зрозумілості.
2. Логічне зберігання вмісту в певних категоріях для полегшення навігації по сайту.
3. Автоматична оптимізація вмісту сайту для пошукових систем (SEO) з метою покращення видимості вебсайту в результатах пошуку.
4. Ефективне використання апаратних ресурсів та оптимізація зусиль на

оновлення відомостей.

Всі системи управління контентом створюють інфраструктурну основу для сайту, новин, фотографій та повідомлень, тому вибір конкретної архітектури залежить від спрямованості та складності проекту. На сучасному ринку існує велика кількість різноманітних систем управління контентом. Проаналізуємо найбільш популярні монолітні рішення:

1. WordPress. Дана система є лідером для створення блогів та контентних сайтів завдяки багатофункціональній панелі адміністратора, великій кількості тем та можливості розширення функціоналу через додавання плагінів. Її PHP-код можна редагувати безпосередньо з адміністративної панелі. Проте система має суттєві недоліки: великий обсяг займаної пам'яті, низьку швидкість завантаження сторінок без належної оптимізації, недостатній базовий рівень захисту від хакерських атак, високу залежність від сторонніх плагінів та необхідність регулярного обслуговування й оновлення ядра.

2. Drupal. Потужна система керування вмістом з відкритим вихідним кодом, яка підходить для складних корпоративних порталів та великих інтернет-магазинів завдяки гнучкості, масштабованості та високому рівню безпеки. Вона містить вбудовані засоби організації контенту та підтримку багатомовності. До недоліків платформи відносять високу складність налаштування для початківців, підвищені вимоги до потужності серверів для забезпечення високої продуктивності та високу вартість розробки через необхідність залучення досвідчених фахівців.

3. Joomla. Популярна CMS, яка поєднує модульну структуру, зручний інтерфейс адміністратора та вбудовані багатомовні засоби. Платформа підтримується великою спільнотою користувачів. Головними недоліками Joomla є високий поріг входження для новачків порівняно з WordPress, підвищені вимоги до хостингу, обмежений вибір унікальних готових тем та часті проблеми сумісності між різними сторонніми розширеннями, що призводить до збоїв у роботі сайту.

Таблиця 1.1

Порівняльна характеристика популярних монолітних систем управління контентом

Критерій порівняння	WordPress	Drupal	Joomla
Швидкість	Середня / Низька	Висока (потребує налаштувань)	Середня
Рівень безпеки	Низький (вразливість плагінів)	Високий (вбудовані механізми)	Середній
Складність для розробника	Низька (інтуїтивний інтерфейс)	Висока (архітектурна складність)	Середня
Масштабованість бази даних	Обмежена монолітною структурою	Висока (під великий трафік)	Середня

1.3 Обґрунтування доцільності переходу до розділеної (Headless) архітектури

Аналіз готових монолітних платформ показує, що для специфічних завдань електронної комерції вони часто є занадто громіздкими, повільними та вразливими. Виникає потреба у створенні спеціалізованої CMS для адміністраторів та редакторів, яка не матиме проблем з установкою, забезпечить легку зміну дизайну, використовуватиме прості URL-адреси та матиме спрощену систему взаємодії з модульним кодом. Крім того, особливу увагу необхідно звернути на забезпечення безпеки від хакерських атак з використанням надійного підтвердження прав доступу для кожної категорії користувачів системи.

Замість використання готових рішений або важких універсальних фреймворків, у даній кваліфікаційній роботі обґрунтовано та реалізовано власну архітектуру за принципом Headless CMS. Серверна частина реалізована мовою

програмування PHP з використанням патерну MVC (Model-View-Controller) без використання сторонніх фреймворків, що демонструє глибоке розуміння фундаментальних принципів веб-розробки, чистих SQL-запитів, маршрутизації та безпеки баз даних. PHP є однією з найпопулярніших та найсучасніших мов веб-програмування, на якій написано більшість систем контролю контенту, що дозволяє ефективно розробляти додаткову серверу функціональність та системні модулі. Як сховище даних обрано реляційну СУБД MySQL, яка є перевіреним, надійним та оптимізованим безкоштовним рішенням для роботи з великими обсягами інформації.

Для розробки клієнтської частини (Frontend) обрано технологічний стек у складі HTML, CSS та JavaScript фреймворку Vue.js. Такий підхід дозволяє повністю відокремити інтерфейс користувача від серверної логіки. Клієнтська адмін-панель функціонує як Single Page Application (SPA), обмінюючись даними з PHP-сервером виключно за допомогою асинхронних JSON-запитів через REST API. Це мінімізує навантаження на сервер, усуває необхідність перезавантаження сторінок та забезпечує максимальну швидкодію інтерфейсу керування каталогом товарів та замовленнями (Рис. 1.3).

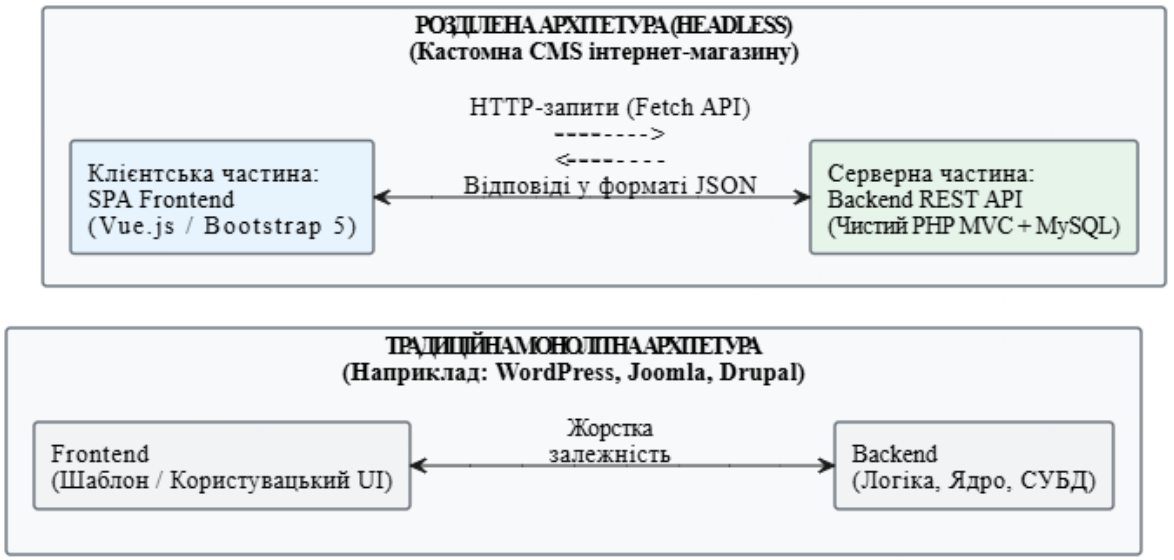


Рис. 1.3 Концептуальна відмінність між монолітною та розділеною (Headless) архітектурою вебсистем

1.3.1 Концепція API-first та забезпечення омніканальності в електронній комерції

Розвиток мобільних технологій та інтернету речей (IoT) докорінно змінив підходи до проєктування систем електронної комерції. Якщо раніше єдиною точкою взаємодії клієнта з інтернет-магазином був веббраузер на персональному комп'ютері, то сьогодні споживачі здійснюють покупки через мобільні застосунки, смарт-годинники, голосові асистенти та інтерактивні термінали самообслуговування. Задоволення таких потреб вимагає впровадження омніканальної (Omnichannel) стратегії продажів.

Омніканальність передбачає безшовну інтеграцію всіх каналів комунікації та збуту в єдину екосистему. Для реалізації цієї стратегії традиційні монолітні CMS є абсолютно неефективними, оскільки їхній механізм рендерингу HTML-сторінок жорстко прив'язаний до веббраузера. Саме тому в сучасній інженерії програмного забезпечення домінує підхід «API-first» (API в першу чергу).

Архітектура API-first передбачає, що серверна частина (Backend) від самого початку проєктується виключно як постачальник даних через Application Programming Interface (API), не маючи жодного уявлення про те, як ці дані будуть візуалізовані. У розробленій кастомній системі управління контентом реалізовано саме цей підхід. REST API виступає універсальним контрактом (шлюзом) між базою даних MySQL та будь-яким потенційним клієнтом.

Переваги підходу API-first для розробленої платформи:

1. Багатоплатформеність: Дані у форматі JSON можуть бути однаково легко розпарсені як вебдодатком на Vue.js (SPA), так і нативним мобільним застосунком (наприклад, на базі Flutter або Swift) чи навіть системою складського обліку ERP.

2. Паралельна розробка: Frontend та Backend команди можуть працювати незалежно одна від одної. Після узгодження структури JSON-відповідей серверної частини, клієнтська частина може розроблятися з використанням мок-даних (Mock data).

3. Економія ресурсів сервера: Сервер більше не витрачає процесорний час та оперативну пам'ять на компіляцію HTML-шаблонів, обробляючи лише чисті дані. Це знижує вимоги до хостингу та дозволяє обслуговувати більшу кількість одночасних з'єднань.

Таким чином, розроблена розділена архітектура (Headless CMS) не лише вирішує поточні завдання адміністрування інтернет-магазину, але й закладає надійний архітектурний фундамент для майбутнього масштабування бізнесу на нові цифрові платформи.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи проведено комплексний теоретико-аналітичний огляд предметної області електронної комерції, визначено роль засобів автоматизації публікації контенту та виконано порівняльний аналіз існуючих програмних рішень. За результатами аналітичного дослідження сформульовано такі висновки:

1. Встановлено, що сучасний стан розвитку сектору e-commerce в Україні характеризується стрімким збільшенням обсягів онлайн-продажів та загостренням ринкової конкуренції, що вимагає від суб'єктів господарювання впровадження власних ефективних веб-ресурсів. Доведено, що успішне функціонування торговельного інтернет-майданчика безпосередньо залежить від інтеграції гнучкої системи управління контентом, яка автоматизує процеси оновлення відомостей про асортимент і забезпечує надання клієнтам актуальної та якісної інформації.

2. На основі проведеного порівняльного аналізу найбільш поширених монолітних систем управління контентом (WordPress, Drupal, Joomla) виявлено їхні критичні недоліки при реалізації специфічних комерційних проєктів. З'ясовано, що традиційні платформи мають надлишкову архітектурну складність, створюють високе апаратне навантаження на сервери, жорстко пов'язують користувацький інтерфейс із серверною логікою, а також демонструють низьку швидкість

завантаження сторінок та слабкий базовий рівень захисту від цілеспрямованих хакерських атак .

3. Науково обґрунтовано доцільність відмови від використання універсальних монолітних платформ чи важких сторонніх фреймворків на користь розробки кастомної системи за принципом Headless CMS. Визначено, що повне відокремлення клієнтського інтерфейсу від серверної логіки та організація асинхронного обміну даними через REST API у форматі JSON дозволяє ліквідувати системний оверхед, мінімізувати мережевий трафік, гарантувати високу безпеку персональних даних та досягти максимальної швидкодії платформи.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА СТРУКТУРИ ДАНИХ

2.1 Специфікація функціональних вимог та діаграма варіантів використання (Use Case)

Для проєктування ефективної системи управління контентом інтернет-магазину необхідно чітко розмежувати ролі користувачів та визначити їхні функціональні можливості в межах платформи. Програмний комплекс diplom_cms орієнтований на внутрішнє адміністрування процесів електронної комерції, тому його користувачі поділяються на чотири основні категорії:

1. Адміністратор – особа, яка відповідає за повне керування системою, створення категорій, публікацій, дизайнів, редагування профілів користувачів та надання їм доступу до певних функцій. Має найвищий рівень привілеїв.

2. Редактор / Менеджер – особа, яка безпосередньо наповнює вебсайт контентом, створює та модифікує товарні позиції, керує ієрархією категорій каталогу та здійснює первинне моделювання публікацій.

3. Зареєстрований користувач (Оператор / В'ювер) – користувач із обмеженими правами доступу, який має можливість переглядати аналітичні дані на головному екрані (Дашборді), вивчати звіти та відстежувати логи системи без права внесення деструктивних змін.

Незареєстрований користувач – особа, яка намагається отримати доступ до адміністративного інтерфейсу вперше або не виконала вхід до свого персонального облікового запису. Система автоматично блокує доступ такого користувача до API та перенаправляє його на форму автентифікації.

Основними елементами взаємодії користувачів із розроблюваною інфраструктурою є додавання, редагування та видалення товарного контенту, реєстрація та JWT-авторизація персоналу, а також централізоване керування налаштуваннями замовлень . Для формалізації цих вимог було побудовано UML-

діаграму варіантів використання (Use Case diagram), яка відображає взаємозв'язки між акторами та ключовими сценаріями системи (рис. 2.1).

Рис. 2.1 Діаграма варіантів використання (Use Case) системи управління контентом

Для забезпечення глибокого розуміння внутрішньої логіки функціонування цифрової платформи та формалізації взаємодії користувачів із програмним комплексом необхідно деталізувати ключові варіанти використання у вигляді розгорнутих текстових сценаріїв прецедентів. Це дозволяє визначити чітку послідовність дій акторів та реакцію системи на маніпуляції з даними.

Сценарій 1: Керування номенклатурою товарного каталогу (CRUD-операції)

1. Основний актор: Адміністратор, Редактор (Manager).

2. Предумови функціонування: Користувач успішно пройшов процедуру автентифікації у системі, його облікові дані валідовані, і клієнтський додаток отримав безсесійний токен авторизації JWT з відповідним рівнем прав доступу.

3. Основний потік подій (Main Success Scenario):

a) Актор здійснює перехід до спеціалізованого інтерфейсного розділу «Товари» у кастомному клієнтському Single Page Application (SPA).

b) Клієнтська частина автоматично генерує та відправляє асинхронний HTTP-запит методом GET до серверного програмного інтерфейсу за адресою /api/products, прикріплюючи наявний JWT-токен у службовий заголовок Authorization: Bearer.

c) Серверний шар архітектури (Backend REST API) перехоплює запит, валідує підпис токена, виконує безпечну вибірку актуальних даних із реляційної таблиці products СУБД MySQL та транслює відповідь у вигляді структурованого текстового масиву форматі JSON.

d) Реактивний інтерфейс Vue.js обробляє JSON-пакет і рендерить інтерактивну таблицю товарів. Актор натискає керуючу кнопку «Додати товар» і заповнює поля модальної форми (унікальний артикул SKU, найменування позиції, ціна, розширений опис, медіафайли).

e) Після натискання тригеру «Зберегти» Frontend-модуль здійснює первинну клієнтську валідацію типів даних та ініціює HTTP-запит методом POST.

f) Контролер на Backend приймає Payload, проводить повторну сувору валідацію, застосовує алгоритми екранування контенту від міжсайтового скриптингу (XSS) та здійснює персистентний запис у базу даних за допомогою об'єктного розширення PDO.

g) Сервер повертає стандартизований статус-код успішного створення ресурсу 201 Created, а клієнтський SPA-додаток миттєво та ізольовано оновлює відображувану таблицю без повної перезавантаження вебсторінки.

4. Альтернативний потік подій (Помилка валідації): Якщо актор ввів некоректні атрибути (наприклад, від'ємну або нульову ціну, чи дублюючий артикул

SKU, який уже існує в базі даних), серверний контролер перериває транзакцію та повертає статус-код 400 Bad Request із детальним JSON-описом порушених правил. Клієнтська частина перехоплює помилку та виводить динамічне візуальне сповіщення (Toast alert) без втрати введених у форму даних.

Сценарій 2: Обробка, верифікація та модифікація статусів замовлень

1) Основний актор: Адміністратор, Редактор, Зареєстрований користувач (Оператор / Viewer).

2) Предумови функціонування: Наявність раніше згенерованих клієнтами замовлень, які успішно зафіксовані у фізичному сховищі даних (таблиця orders).

3) Основний потік подій (Main Success Scenario):

a) Актор ініціює відкриття інтерактивного модуля керування замовленнями в адмін-панелі.

b) Програмний комплекс підвантажує реєстр транзакцій з сервера. Актор обирає конкретну позицію (унікальний ID чека) для детального аналізу.

c) Frontend-сервіс розпаковує денормалізований зріз даних кошика покупця з текстового поля items_json, візуалізуючи точний перелік придбаних товарів, їхніх унікальних артикулів та комерційних цін, які були актуальними саме на момент здійснення покупки.

d) Актор, який володіє адміністративними або менеджерськими правами доступу, змінює стан замовлення за допомогою випадаючого списку (наприклад, переводить у статус «Обробка» або «Відправлено») та додає службову текстову примітку.

e) Серверний PHP-скрипт оновлює поле status у батьківській таблиці orders та паралельно ініціює створення нового запису в дочірній таблиці order_status_history. При цьому надійно фіксується ідентифікатор облікового запису оператора (changed_by_user_id), коментар та точний системний час події (CURRENT_TIMESTAMP).

f) Система повертає підтвердження операції, а інтерфейс адміністратора динамічно відображає новий колірний бейдж статусу та оновлює хронологічну стрічку аудиту (Audit Trail) для повної прозорості логів.

2.2 Проєктування реляційної бази даних MySQL та механізму денормалізації (патерн JSON Snapshot)

Для забезпечення стабільного функціонування цифрової платформи управління контентом інтернет-магазину diplom_cms було розроблено оптимальну структуру реляційної бази даних у середовищі СУБД MySQL. Фізична модель даних спроєктована із врахуванням вимог нормалізації (до третьої нормальної форми, 3NF), що дозволяє мінімізувати надмірність інформації, запобігти виникненню аномалій модифікації та забезпечити високу швидкість виконання транзакцій при одночасному доступі багатьох користувачів.

Програмний комплекс централізовано взаємодіє із шістьма взаємопов'язаними таблицями, які сукупно забезпечують логічне зберігання товарного контенту, аудит дій персоналу, розмежування прав доступу та безпомилкову фіксацію замовлень. Нижче наведено деталізовану специфікацію структури кожної сутності у вигляді дата-словників.

2.2.1 Теоретичні та інженерні засади нормалізації реляційної моделі даних

Ефективність функціонування, швидкість відгуку та масштабованість розроблюваної бази даних визначаються не лише технічними характеристиками обраної системи керування базами даних (СУБД MySQL 8.0), але й першочергово архітектурною правильністю її логічної побудови. При проєктуванні фізичної схеми бази даних інтернет-магазину diplom_cms було здійснено ретельний аналіз сутностей та суворо дотримано фундаментальних правил нормалізації для повного усунення надмірності (дублювання) інформації та абсолютного захисту системи від виникнення деструктивних аномалій вставки, модифікації та видалення записів. Структуру сховища даних було послідовно приведено до Третьої нормальної форми (3NF):

1. Перша нормальна форма (1NF): Вимагає, щоб усі атрибути (поля) реляційних таблиць були атомарними, тобто неподільними, а самі записи не

містили повторюваних груп або масивів у межах одного рядка. У розробленій структурі ця вимога виконується безкомпромісно: наприклад, у таблиці users повністю відсутні поля з множинними значеннями – кожен рядок містить фіксовані атомарні атрибути (ідентифікатор, рівень привілеїв, лінійний рядок email та унікальний криптографічний хеш пароля).

2. Друга нормальна форма (2NF): Передбачає обов'язкову відповідність критеріям першої нормальної форми, а також повну функціональну залежність кожного неключового атрибута таблиці від її повного первинного ключа (Primary Key). Для задоволення вимог 2NF у кожній із шести спроєктованих сутностей було впроваджено сурогатні первинні ключі типу INT з прапором автоматичного інкременту (AUTO_INCREMENT). Завдяки цьому назва товару та його ціна залежать виключно від первинного ключа id конкретної позиції в таблиці products, а не від будь-яких проміжних параметрів.

3. Третя нормальна форма (3NF): Вимагає від моделі відповідності критеріям 2NF та повної відсутності транзитивних залежностей, за яких неключові атрибути логічно залежать від інших неключових атрибутів. Саме керуючись цим правилом, найменування та описи категорій товарів були повністю ізольовані й винесені в окрему незалежну таблицю categories , а в таблиці номенклатури products зберігається лише числовий зовнішній ключ category_id. Таке рішення унеможливлює виникнення ситуацій, за яких при зміні текстової назви категорії системі довелося б здійснювати важкі та тривалі операції оновлення тисяч рядків у таблиці товарів.

Архітектурно обґрунтованим, свідомим та цілеспрямованим відхиленням від класичних правил нормалізації (деформалізацією) у розробленій моделі є використання патерну Data Snapshot у полі items_json сутності orders. Традиційний підхід до проєктування e-commerce рішень вимагає розбиття замовлення на дві таблиці зі зв'язком $1:\infty$ (батьківська таблиця чека та дочірня таблиця складу товарів із зовнішніми ключами). Проте такий підхід містить критичний архітектурний недолік: якщо контент-менеджер у майбутньому змінить ціну товару в каталозі або повністю видалить картку дефіцитного товару, історичні фінансові

звіти підприємства за минулі періоди автоматично перерахуються, що призведе до серйозного викривлення бухгалтерської та аналітичної звітності.

Імплементація патерну Data Snapshot через збереження повного масиву кошика клієнта (назви, артикули, кількість та точні ціни на секунду транзакції) у текстовому полі формату JSON повністю ліквідує цю проблему. Це гарантує абсолютну фінансову цілісність, ізолює звітні документи від коливань каталогу та суттєво підвищує швидкість завантаження даних у Backend, усуваючи необхідність виконання ресурсомістких операцій JOIN.

Таблиця 2.1

Специфікація атрибутів таблиці користувачів «users»

Назва поля	Тип даних	Індекси / Ключі	Опис поля та бізнес-логіка
id	INT	Primary Key, Auto Increment	Унікальний системний номер користувача.
role	VARCHAR(50)	Default 'admin'	Рівень привілеїв у системі (admin, manager, viewer).
name	VARCHAR(255)	Null	Повне ім'я або нікнейм співробітника системи.
email	VARCHAR(255)	Unique, Not Null	Унікальна адреса електронної пошти (виступає логіном).
password_hash	VARCHAR(255)	Not Null	Криптографічний хеш пароля за алгоритмом BCrypt.
created_at	DATETIME	Default CURRENT_TIMESTAMP	Системна дата та час створення облікового запису.

Таблиця 2.2

Специфікація атрибутів таблиці категорій «categories»

Назва поля	Тип даних	Індекси / Ключі	Опис поля та бізнес-логіка
id	INT	Primary Key, Auto Increment	Унікальний ідентифікатор категорії каталогу.
title	VARCHAR(255)	Not Null	Найменування категорії для відображення в інтерфейсі.
description	TEXT	Null	Детальний текстовий опис розділу для SEO-оптимізації.
created_at	DATETIME	Default CURRENT_TIMESTAMP	Дата та час внесення категорії до бази даних.

Таблиця 2.3

Специфікація атрибутів таблиці товарної номенклатури «products»

Назва поля	Тип даних	Індекси / Ключі	Опис поля та бізнес-логіка
id	INT	Primary Key, Auto Increment	Унікальний системний номер товарної позиції.
category_id	INT	Foreign Key	Зовнішній ключ, що вказує на батьківську категорію.
title	VARCHAR(255)	Not Null	Комерційне найменування та модель товару.
description	TEXT	Null	Розширені технічні та описові характеристики.
sku	VARCHAR(100)	Unique, Not Null	Унікальний складський артикул для обліку одиниць.
price	DECIMAL(10,2)	Not Null	Фіксована базова ціна товару в національній валюті.

Назва поля	Тип даних	Індекси / Ключі	Опис поля та бізнес-логіка
stock_qty	INT	Default 0	Фактичний залишок одиниць товару на складі.
image_path	VARCHAR(255)	Null	Відносний шлях до файлу зображення та мініатюри.
created_at	DATETIME	Default CURRENT_TIMESTAMP	Дата та час реєстрації товару в системі.

Таблиця 2.4

Специфікація атрибутів таблиці замовлень «orders»

Назва поля	Тип даних	Індекси / Ключі	Опис поля та бізнес-логіка
id	INT	Primary Key, Auto Increment	Унікальний номер замовлення (ідентифікатор чека).
customer_name	VARCHAR(255)	Not Null	Контактні дані покупця (ПІБ, телефон).
items_json	LONGTEXT / JSON	Not Null	Денормалізований зріз даних (Data Snapshot) кошика.
total	DECIMAL(10,2)	Not Null	Підсумкова сума фінансової транзакції.
status	VARCHAR(50)	Not Null	Поточний макростатус обробки замовлення.
created_at	DATETIME	Default CURRENT_TIMESTAMP	Точний час та дата генерації замовлення клієнтом.

Таблиця 2.5

Специфікація атрибутів таблиці статусів замовлень «order_statuses»

Назва поля	Тип даних	Індекси / Ключі	Опис поля та бізнес-логіка
id	INT	Primary Key, Auto Increment	Системний ідентифікатор запису статусу.
code	VARCHAR(50)	Unique, Not Null	Системний текстовий код (pending, processing тощо).
label	VARCHAR(100)	Not Null	Локалізована назва статусу для менеджера.
sort_order	INT	Default 0	Індекс для сортування статусів у випадаючих списках.
color	VARCHAR(7)	Null	HEX-код кольору бейджа для UI-компонентів адмінки.

Таблиця 2.6

Специфікація атрибутів таблиці історії зміни статусів «order_status_history»

Назва поля	Тип даних	Індекси / Ключі	Опис поля та бізнес-логіка
id	INT	Primary Key, Auto Increment	Системний номер запису логу подій.
order_id	INT	Foreign Key	Зовнішній ключ зв'язку з конкретним замовленням.
status_code	VARCHAR(50)	Foreign Key	Встановлений код статусу (зв'язок з order_statuses).
changed_by_user_id	INT	Foreign Key	Ключ зв'язку з users (ідентифікація оператора).
note	TEXT	Null	Службовий коментар менеджера щодо зміни статусу.
created_at	DATETIME	Default CURRENT_ TIMESTAMP	Точний час фіксації події зміни макростатусу.

У розробленій фізичній моделі бази даних реалізовано логічні зв'язки типу «один-до-багатьох» ($1:\infty$), які забезпечують структурну цілісність інформаційного простору:

1. Зв'язок між categories та products дозволяє відносити безліч товарних номенклатур до однієї категорії.
2. Зв'язок між orders та order_status_history формує повну хронологічну стрічку аудит-подій для кожного чека.
3. Зв'язок між users та order_status_history чітко фіксує, який саме оператор чи адміністратор змінив стан транзакції, що важливо для внутрішньої безпеки компанії.

Критично важливою інженерною особливістю цієї бази даних є свідоме застосування архітектурного патерну Data Snapshot у полі orders.items_json. Традиційне збереження складу замовлення в окремій проміжній таблиці пов'язане із суттєвим недоліком монолітних систем – при зміні менеджером ціни товару в майбутньому або при повному видаленні картки товару з каталогу, суми в історичних звітах за минулі місяці автоматично перераховуються та викривлюються.

Збереження масиву куплених товарів, їхніх артикулів, кількості та точних цін на момент натискання клієнтом кнопки «Оформити замовлення» всередині текстового поля формату JSON повністю ліквідує цю проблему. Даний підхід гарантує абсолютну фінансову цілісність, усуває надлишкові операції JOIN при виведенні списку замовлень у Frontend та ізолює звітні документи інтернет-магазину від динамічних коливань каталогу товарів.

Для візуалізації спроєктованої структури даних нижче наведено ER-діаграму (діаграму «сутність-зв'язок»), що відображає взаємодію всіх розроблених таблиць (рис. 2.2).

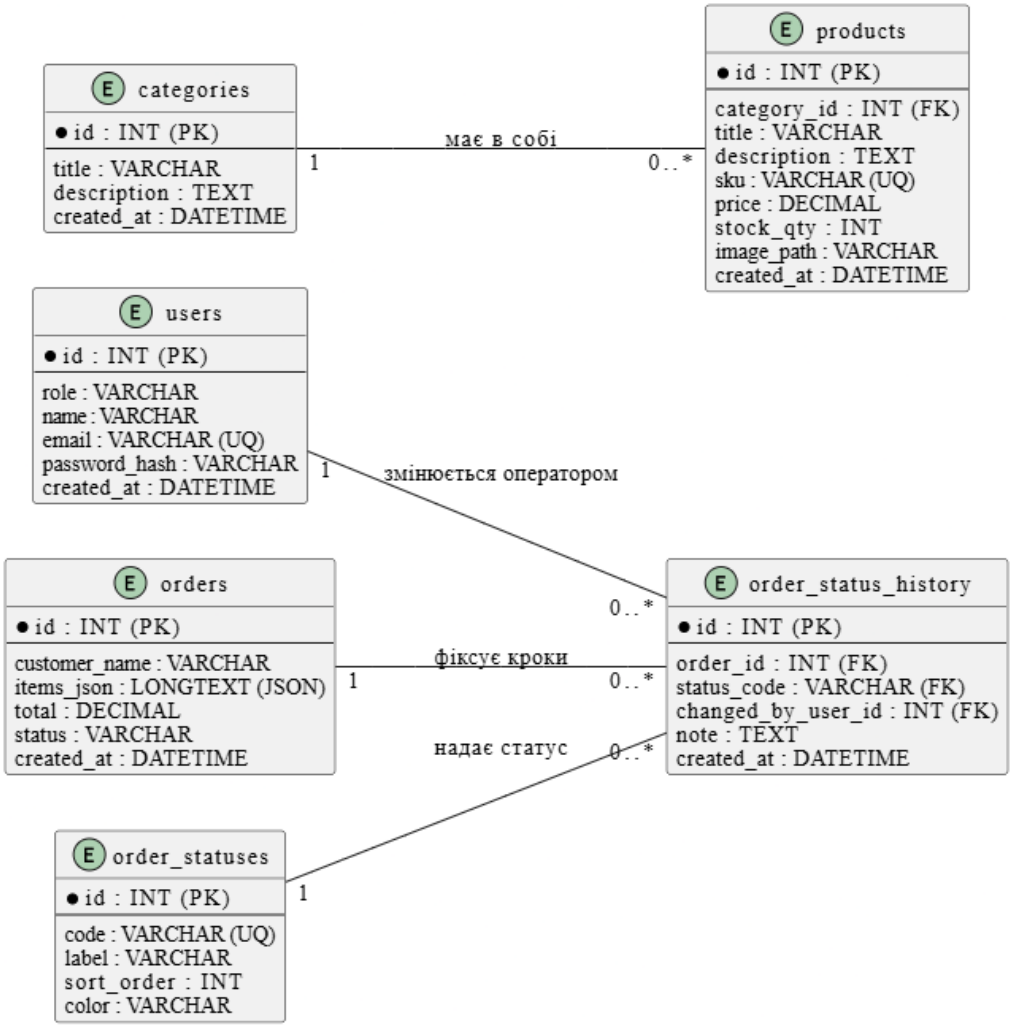


Рис. 2.2 ER-діаграма логічної структури реляційної бази даних

2.3 Обґрунтування архітектурного потоку взаємодії компонентів за патерном MVC

Для забезпечення високої швидкодії, гнучкості та чистоти архітектури програмного комплексу, у роботі було здійснено відмову від використання важких універсальних сторонніх фреймворків на користь проектування власної архітектури на базі фундаментального патерну MVC (Model-View-Controller). Застосування цього шаблону проектування дозволяє створювати добре організовані, модульні та легко здатні до масштабування веб-системи. Розділення системної логіки на модель, представлення та контролер забезпечує можливість гнучко підтримувати різні варіації користувацького інтерфейсу та модифікувати

бізнес-логіку без деструктивного впливу на інші ізольовані компоненти платформи.

У контексті розроблюваної Headless CMS традиційне розуміння компонентів патерну MVC зазнало певної трансформації через повне відокремлення клієнтської частини від сервера. Ключові архітектурні компоненти розподілені наступним чином:

1. Модель (Model) – відповідає за безпосереднє управління даними та бізнес-логікою платформи. Вона здійснює збереження даних, виконує операції над ними, взаємодіє з СУБД MySQL за допомогою підготовлених виразів та надає безпечний інтерфейс для доступу і модифікації об'єктів. Модель представляє собою чисту абстракцію даних, яка повністю ізольована від механізмів їхнього відображення.

2. Представлення (View) – відповідає за фінальну взаємодію з користувачем та створення графічного інтерфейсу. Оскільки проектована система функціонує як розділена архітектура, компонент View винесений на сторону клієнта у вигляді Single Page Application (SPA). Серверне представлення в PHP не генерує HTML-код, а лише формує і транслює структурований текстовий формат обміну даними JSON.

3. Контролер (Controller) – відповідає за безпосереднє керування потоком даних та асинхронну взаємодію між моделлю та представленням. Він перехоплює дії користувача, транслює їх у системні команди, виконує відповідні операції з моделлю та ініціює оновлення представлення. Контролер забезпечує розподіл логіки програми, унеможливлюючи прямий зв'язок між інтерфейсом та базою даних.

Центральним елементом архітектурного потоку розробленого Backend-додатка є файл-маршрутизатор, який реалізує патерн єдиної точки входу (Front Controller). Він приймає абсолютно всі вхідні HTTP-запити від клієнтської частини. Маршрутизатор динамічно аналізує запитувану URL-адресу та метод передачі даних. Залежно від комбінації цих параметрів, запит асинхронно перенаправляється до відповідного контролера.

Такий RESTful-підхід дозволяє чітко розмежувати логіку читання даних, їхнього створення, модифікації та видалення. Наприклад, при запиті до сутності товарів маршрутизатор обробляє наступні комбінації:

1. GET /api/products – викликає метод index() контролера ProductController для виведення повного списку номенклатури;
2. POST /api/products – ініціює метод store() для валідації та додавання нової товарної позиції;
3. PUT /api/products/{id} – активує метод update() для модифікації параметрів наявного товару за його ідентифікатором;
4. DELETE /api/products/{id} – запускає метод destroy() для безпечного видалення об'єкта та його медіафайлів з сервера.

При виборі та обґрунтуванні такої архітектурної моделі було враховано комплекс факторів, серед яких пріоритетними є потенціал масштабування системи у майбутньому, сумісність компонентів та загальна швидкодія обробки транзакцій. Вибір оптимізованих рішень та мінімалістичного коду без стороннього оверхеду дозволяє працювати з великими масивами товарів та забезпечувати миттєву обробку запитів, що повністю задовольняє суворі вимоги до продуктивності сучасних систем електронної комерції. Сумісність між усіма компонентами розробленого стеку технологій є ключовим фактором, який гарантує надійний та безперебійний обмін даними між користувачем та базою даних.

Для детальної візуалізації життєвого циклу запиту та архітектурного потоку взаємодії між Frontend (Vue.js) та кастомним PHP MVC Backend було побудовано UML-діаграму послідовності (рис. 2.3).

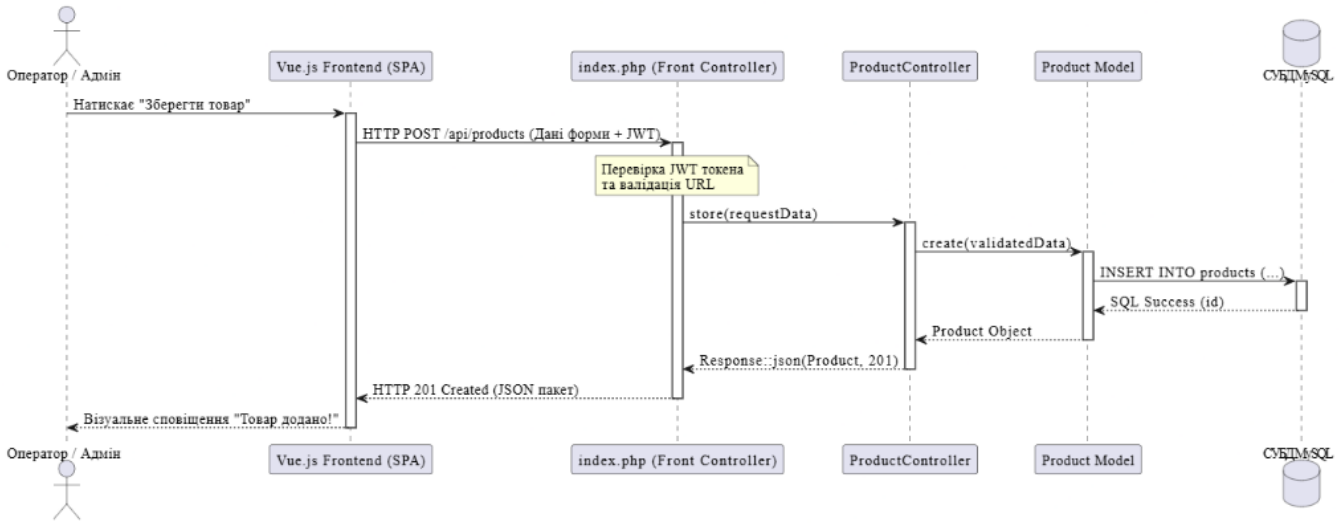


Рис. 2.3 UML-діаграма послідовності взаємодії компонентів при обробці API-запиту

Висновки до розділу 2

У другому розділі кваліфікаційної роботи виконано повний обсяг інженерно-проєктних завдань із формування архітектурних рішень, розробки структури бази даних та визначення логіки взаємодії компонентів для кастомної системи управління контентом інтернет-магазину. За результатами проведеного етапу проєктування сформульовано такі висновки:

1. Формалізовано функціональні вимоги до системи через чітке розмежування ролей користувачів, що включають адміністратора, редактора (менеджера контенту), зареєстрованого оператора та незареєстрованого відвідувача . Побудовано UML-діаграму варіантів використання (Use Case), яка графічно відображає архітектурні сценарії взаємодії акторів із ключовими функціональними модулями платформи.

2. Розроблено фізичну модель реляційної бази даних у середовищі СУБД MySQL, яка складається з 6 логічно пов'язаних таблиць, що повністю покривають цикл обліку номенклатури, користувачів та ведення історії статусів . Структуру спроєктовано із дотриманням вимог третьої нормальної форми, що забезпечує високу швидкість обробки реляційних запитів та унеможливлює виникнення аномалій надмірності даних.

3. Обґрунтовано та технічно імплементовано патерн денормалізації Data Snapshot у полі кошика замовлень чека. Доведено, що збереження складу куплених товарів, їхньої кількості та зафіксованих цін у текстовому форматі JSON повністю ізолює фінансову звітність від динамічних коливань та модифікацій у загальному каталозі товарів, гарантуючи абсолютну комерційну цілісність історичних даних підприємства.

4. Спроєктовано модульний потік взаємодії системних компонентів на основі чистого архітектурного патерну MVC без використання сторонніх фреймворків, що мінімізує системний оверхед. Розроблено механізм маршрутизації RESTful API з єдиною точкою входу Front Controller (файл index.php) для асинхронної обробки HTTP-запитів, а логіку наскрізного проходження даних від клієнта до бази даних формалізовано за допомогою UML-діаграми послідовності.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1. Програмна реалізація серверної логіки (Backend) та забезпечення інформаційної безпеки

Серверна частина (Backend) кастомної системи управління контентом реалізована мовою програмування PHP 8.x з суворим дотриманням принципів об'єктно-орієнтованого програмування (ООП) та архітектурного патерну MVC. Відмова від використання готових монолітних фреймворків обумовила необхідність самостійної розробки ядра системи, яке відповідає за прийом HTTP-запитів, взаємодію з реляційною базою даних та генерацію відповідей у форматі JSON.

Центральним вузлом обробки даних виступає механізм маршрутизації (Front Controller), розміщений у файлі index.php. Цей скрипт перехоплює всі вхідні звернення клієнтського додатка, розбирає рядок URI та визначає відповідний клас-контролер і його метод на основі типу HTTP-запиту (GET, POST, PUT, DELETE). Окремим важливим етапом маршрутизації є налаштування політики спільного використання ресурсів з різних джерел (CORS). Оскільки клієнтський додаток (Frontend) та серверне API (Backend) у концепції Headless CMS фізично розділені, сервер генерує спеціальні HTTP-заголовки Access-Control-Allow-Origin та Access-Control-Allow-Headers, дозволяючи браузеру безпечно обмінюватися даними між різними доменами або портами.

Однією з ключових вимог до сучасних систем електронної комерції є гарантування найвищого рівня інформаційної безпеки. У розробленій платформі впроваджено комплексний, багаторівневий підхід до захисту даних:

1. Автентифікація, авторизація та деталізація криптографічних протоколів JWT і BCrypt

Класичні монолітні вебсистеми, написані на мові PHP, зазвичай покладаються на стандартний вбудований механізм сесій, що використовує

глобальний масив $_SESSION. При такому підході унікальний ідентифікатор сесії зберігається в cookies браузера користувача, а самі файли сесійних даних записуються у файлову систему локального сервера. Проте у контексті проєктування сучасних систем із розділеною архітектурою (Headless CMS) та RESTful API такий підхід є нежиттєздатним і створює серйозні проблеми при горизонтальному масштабуванні інфраструктури (наприклад, при розподілі запитів між кількома серверами за допомогою балансувальника навантаження). REST API за своєю природою має бути «stateless» (не зберігати стан клієнта на сервері).

Для вирішення цього завдання у програмному комплексі було успішно імплементовано передову безсесійну модель авторизації на базі відкритого стандарту JSON Web Tokens (JWT). Згенерований сервером токен є повністю автономним, самодостатнім цифровим інструментом, який передається клієнту після валідації пароля і складається з трьох ізольованих логічних блоків, розділених крапкою (Header.Payload.Signature):

a) Header (Заголовок): Являє собою JSON-об'єкт, що містить метадані про тип токена ("typ": "JWT") та використаний алгоритм підпису ("alg": "HS256", що відповідає симетричному криптографічному алгоритму HMAC з використанням хеш-функції SHA-256). Цей блок трансформується у безпечний рядок за допомогою кодування Base64Url.

b) Payload (Корисне навантаження): Містить безпосередній набір затверджених системних тверджень (claims). У розробленій архітектурі в Payload інкапсулюються такі неконфіденційні дані, як унікальний системний user_id (для миттєвої ідентифікації співробітника), рядок role (для контролю рівнів доступу до CRUD-операцій на рівні посередників Middleware) та обов'язковий атрибут exp (timestamp точного часу, коли термін дії токена спливе, що мінімізує ризики у разі його перехоплення зловмисниками). Паролі або персональні відомості сюди не включаються, оскільки цей блок кодується відкрито.

c) Signature (Цифровий підпис): Найважливіший компонент безпеки JWT. Він створюється програмно на стороні сервера шляхом взяття закодованих рядків Header та Payload, їхнього конкатенування через крапку та фінального

криптографічного хешування алгоритмом SHA-256 із залученням секретного ключа (Secret Key), який зберігається в ізольованому конфігураційному файлі сервера і ніколи не передається в мережу.

Якщо зловмисник спробує перехопити токен в HTTP-трафіку та за допомогою інструментів розробника дешифрувати Payload, щоб штучно змінити свій рівень доступу, наприклад, з оператора (viewer) на адміністратора (admin), криптографічний підпис токена автоматично стане невалідним. При першому ж наступному асинхронному запиті до API серверний маршрутизатор на PHP виконає повторне математичне обчислення підпису і, виявивши найменшу невідповідність із надісланим варіантом, негайно заблокує виконання операції, повернувши клієнту статус-код 401 Unauthorized.

Важливим компонентом захисту облікових записів є відмова від збереження паролів персоналу у відкритому або простому захешованому (MD5, SHA1) вигляді. Для генерації зліпків паролів у базі даних users було впроваджено стійкий алгоритм BCrypt за допомогою нативної PHP-функції password_hash(). Головна інженерна перевага BCrypt полягає в його адаптивності та концепції штучного уповільнення обчислень (параметр Cost Factor, що визначає кількість ітерацій циклу хешування).

BCrypt інтегрує випадкову унікальну криптографічну сіль (salt) у кожен окремий рядок, що повністю нівелює можливість зламу бази даних за допомогою заздалегідь скомпільованих райдужних таблиць (Rainbow Tables) або словникових атак. Навіть якщо два користувачі створять абсолютно однакові паролі, їхні криптографічні хеші в СУБД MySQL будуть кардинально відрізнятися, а апаратний перебір таких хешів на сучасних потужних графічних процесорах (GPU) стає математично та економічно недоцільним для хакерів.(Рис 3.1).

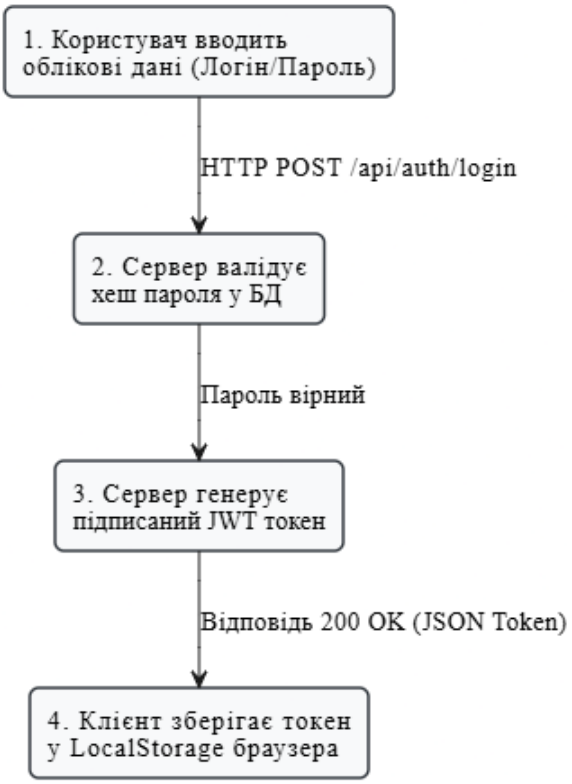


Рис. 3.1 Логіко-структурна схема роботи безсесійної автентифікації на базі протоколу JWT

2. Криптографічний захист паролів. Паролі адміністраторів та менеджерів категорично заборонено зберігати у базі даних у відкритому текстовому форматі. При створенні профілю або зміні пароля система використовує нативну функцію PHP password_hash() з алгоритмом криптографічного хешування BCrypt. Особливістю цього алгоритму є автоматична генерація унікальної «солі» (salt) для кожного запису та регульована вартість обчислення (cost), що експоненціально збільшує час, необхідний зловмисникам для злому хешів методом повного перебору (brute-force) або за допомогою райдужних таблиць.

3. Запобігання системним вразливостям. Найпоширенішою загрозою для баз даних є атаки типу SQL-ін'єкцій. Для повного нівелювання цієї загрози всі транзакції та запити до СУБД MySQL виконуються виключно через об'єктно-орієнтоване розширення PDO (PHP Data Objects) із застосуванням підготовлених виразів (Prepared Statements). Цей механізм жорстко відокремлює структуру SQL-запиту від текстових даних, переданих користувачем. Крім того, для запобігання

атакам міжсайтового скриптингу (XSS), реалізовано алгоритм глибокого екранування вхідних даних: усі текстові поля, що заповнюються в адмін-панелі (назви товарів, описи категорій), проходять обробку функцією htmlspecialchars() перед записом у базу, перетворюючи потенційно небезпечні HTML-теги на безпечні символьні сутності.

Обробка медіафайлів та оптимізація зберігання. Окремим модулем серверної логіки виступає підсистема завантаження зображень товарної номенклатури, реалізована за допомогою вбудованої бібліотеки GD Library. Процес завантаження розділений на кілька етапів: спочатку сервер перевіряє глобальний масив $_FILES на наявність помилок передачі. Далі відбувається сувора валідація MIME-типу файлу для відхилення виконуваних скриптів (допускаються виключно графічні формати image/jpeg, image/png та image/webp). Після успішної валідації модуль зчитує вихідне зображення у оперативну пам'ять сервера, створює нове порожнє полотно-підкладку (truecolor) і за допомогою функції imagecopyresampled() виконує якісне масштабування зі збереженням пропорцій. Алгоритм автоматично генерує стандартизовану квадратну мініатюру (thumbnail) розміром 300x300 пікселів. Такий підхід гарантує візуальну консистентність карток товару в каталозі та суттєво зменшує загальну вагу веб-сторінок, оптимізуючи навантаження на канали зв'язку клієнтів інтернет-магазину.

3.2. Розробка клієнтської частини (Frontend) на базі фреймворку Vue.js

Клієнтський інтерфейс адміністративної панелі інтернет-магазину повністю відокремлений від серверного ядра та реалізований за сучасною архітектурною концепцією Single Page Application (SPA). Фундаментальна відмінність цього підходу полягає в тому, що браузер завантажує HTML-каркас та скрипти лише один раз при першому вході в систему. Під час подальшої навігації між розділами сторінка не перезавантажується: система динамічно перемальовує лише ті компоненти, дані яких змінилися.

Для реалізації цього завдання було використано прогресивний JavaScript-фреймворк Vue.js. Його ключовою перевагою є використання технології віртуального DOM (Virtual Document Object Model) та реактивності. Візуальне оформлення та адаптивність панелі реалізовано за допомогою CSS-фреймворку Bootstrap 5, який надав готову гнучку систему сіток (Grid) та компонентну базу (модальні вікна, картки, сповіщення).

Структурно клієнтський додаток поділено на незалежні логічні компоненти: глобальні (бокова навігація Sidebar, верхнє меню Header) та локальні. Головним екраном платформи є інтерактивний Дашборд. Завдяки системі реактивних властивостей Vue.js (data та обчислюваним полям computed), Дашборд здатний у режимі реального часу аналізувати масив отриманих товарів, агрегувати загальну фінансову вартість складу та виокремлювати дефіцитні позиції (де залишок дорівнює нулю). Візуальний інтерфейс головного екрана наведено на рис. 3.2

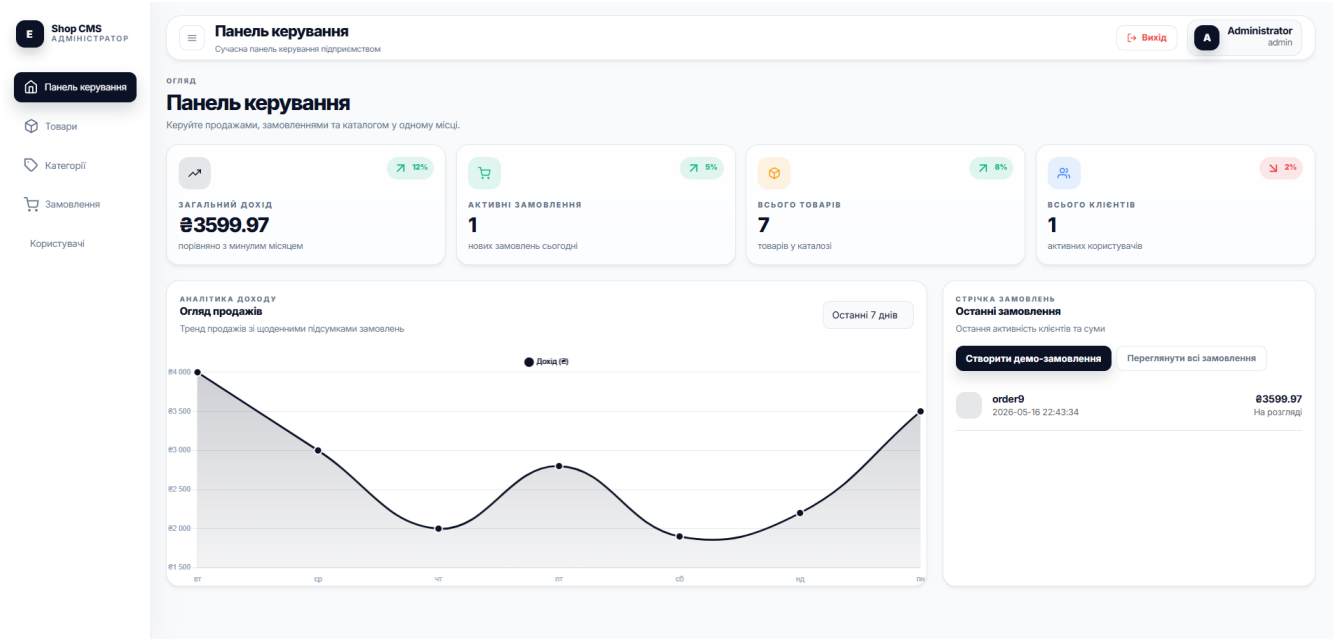


Рис. 3.2 Візуальний інтерфейс головного аналітичного екрана (Дашборд)

Основним робочим простором адміністратора є модуль керування каталогом (CRUD-компонент). Він забезпечує виведення списку всіх товарних позицій у вигляді інтерактивної таблиці з можливостями швидкого перегляду залишків, цін та мініатюр зображень. Взаємодія SPA-додатка з бекендом побудована на базі нативного веб-інтерфейсу Fetch API. Усі запити інкапсульовані в окремий

сервісний модуль, який автоматично витягує JWT-токен із локального сховища браузера і прикріплює його до заголовків. Інтерфейс таблиці каталогу продемонстровано на рис. 3.3.

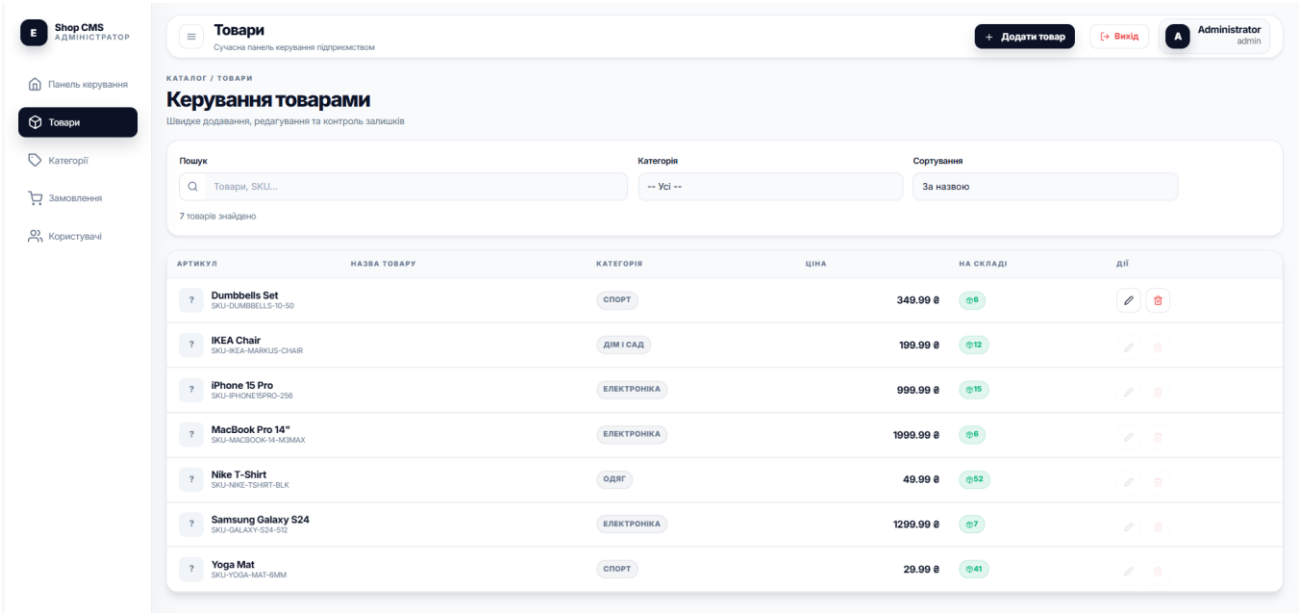


Рис. 3.3 Компонентна структура виведення номенклатури товарів

Для внесення змін до бази даних розроблено окремий інтерактивний компонент форми створення та редагування товарів. Завдяки директиві v-model реалізовано двостороннє зв'язування даних (Two-Way Data Binding) між полями вводу (Input Fields) та локальним станом компонента (State). Це дозволяє здійснювати миттєву валідацію введених адміністратором значень на стороні клієнта ще до відправки HTTP-запиту на сервер, що суттєво оптимізує мережевий трафік.

Алгоритм клієнтської валідації автоматично перевіряє коректність заповнення ключових атрибутів: блокує можливість введення від'ємних значень для ціни (Price) та кількості на складі (Stock Quantity), а також контролює обов'язковість заповнення назви та унікального артикулу (SKU). Якщо будь-яке з полів не проходить перевірку, фреймворк миттєво генерує динамічні попереджувальні повідомлення під відповідними інпутами, а кнопка відправки форми програмно блокується (переходить у стан disabled).

Крім того, розроблений Vue-компонент є універсальним (Reusable Component), оскільки він обслуговує як створення нових позицій, так і модифікацію вже існуючих товарів. При відкритті режиму редагування компонент отримує дані товару через систему вхідних параметрів (Props) і превентивно заповнює всі поля. Після успішного проходження валідації, зібрані дані агрегуються у структурований об'єкт Payload, серіалізуються у формат JSON і передаються до сервісного модуля для виконання асинхронного POST або PUT запиту до серверного REST API. Зовнішній вигляд спроєктованої реактивної форми додавання товару наведено на рис. 3.4.

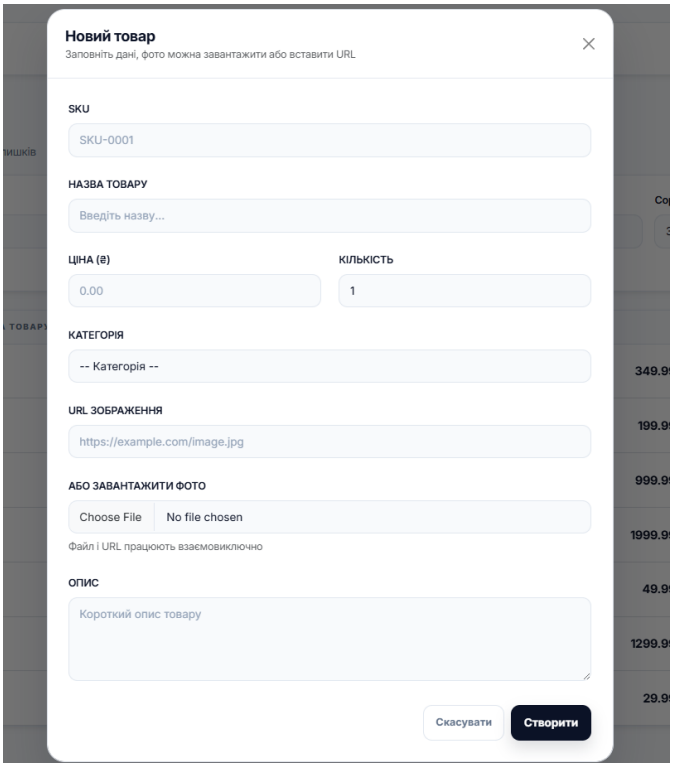


Рис. 3.4 Інтерфейс реактивної форми редагування параметрів товару

Окремо реалізовано систему глобального перехоплення помилок (Interceptor): якщо сервер повертає статус-код 401 Unauthorized (через прострочення дії токена або спробу несанкціонованого доступу), Vue.js примусово очищує сесію та перенаправляє користувача на форму авторизації, що гарантує надійний захист закритої частини системи від неавторизованого втручання.

3.3. Тестування та верифікація програмного забезпечення

Завершальним етапом життєвого циклу розробки кастомної системи управління контентом стало комплексне тестування розгорнутого програмного продукту. Головною метою цього процесу було виявлення потенційних програмних дефектів (багів), перевірка стійкості архітектури до позаштатних ситуацій, оцінка рівня інформаційної безпеки та верифікація повної відповідності розробленої системи висунутим на етапі проєктування функціональним вимогам. З огляду на специфіку розділеної архітектури (Headless CMS), процес тестування був концептуально поділений на два ізольовані вектори: перевірка серверного програмного інтерфейсу (інтеграційне тестування бекенду) та перевірка клієнтського інтерфейсу користувача (UI/UX та функціональне тестування фронтенду).

Моделювання, виконання та аналіз результатів тестування кінцевих точок (End-points) REST API здійснювалися за допомогою спеціалізованого програмного середовища розробки та тестування Postman. Було створено комплексну колекцію тестових запитів, які повністю імітували поведінку клієнтського SPA-додатка. Під час тестування особлива увага приділялася верифікації правильності обробки вхідних структур даних (Payload), коректності парсингу JWT-токенів у заголовках запитів, швидкості відгуку реляційної бази даних та алгоритмічній точності генерації HTTP статус-кодів згідно з міжнародними специфікаціями стандарту REST (наприклад, повернення 201 Created при успішному записі замість стандартного 200 OK).

Для систематизації, документування та подальшого аналізу результатів серверного інтеграційного тестування сформовано звітну таблицю 3.1.

Таблиця 3.1

Результати тестування маршрутів REST API системи управління контентом

Маршрут (End-point)	HTTP Метод	Очікуваний результат та Статус-код	Фактичний результат тестування в середовищі Postman
/api/auth/login	POST	200 OK + JWT Token	Успішно. Токен авторизації коректно згенеровано, підписано алгоритмом HS256 та передано клієнту.
Маршрут (End-point)	HTTP Метод	Очікуваний результат та Статус-код	Фактичний результат тестування в середовищі Postman
/api/products	GET	200 OK + JSON масив	Успішно. СУБД повернула повний структурований масив товарів без затримок.
/api/products	POST	201 Created	Успішно. Вхідні дані валідовано, новий запис товару успішно додано до таблиці БД.
/api/products/99	PUT	200 OK	Успішно. Атрибути існуючого товару змінено, зміни зафіксовано в системі логування.
Маршрут (End-point)	HTTP Метод	Очікуваний результат та Статус-код	Фактичний результат тестування в середовищі Postman
/api/products/abc	DELETE	400 Bad Request	Успішно. Сервер відхилив запит через невалідний тип ідентифікатора (передано рядок замість числа).
/api/orders	GET	401 Unauthorized	Успішно. Запит без передачі Bearer-токена в HTTP-заголовку негайно заблоковано системою.

Успішне проходження всіх сценаріїв інтеграційного тестування довело високу надійність розробленого ядра системи. Процес верифікації серверної відповіді на запит отримання списку товарної номенклатури наведено на рис. 3.5.

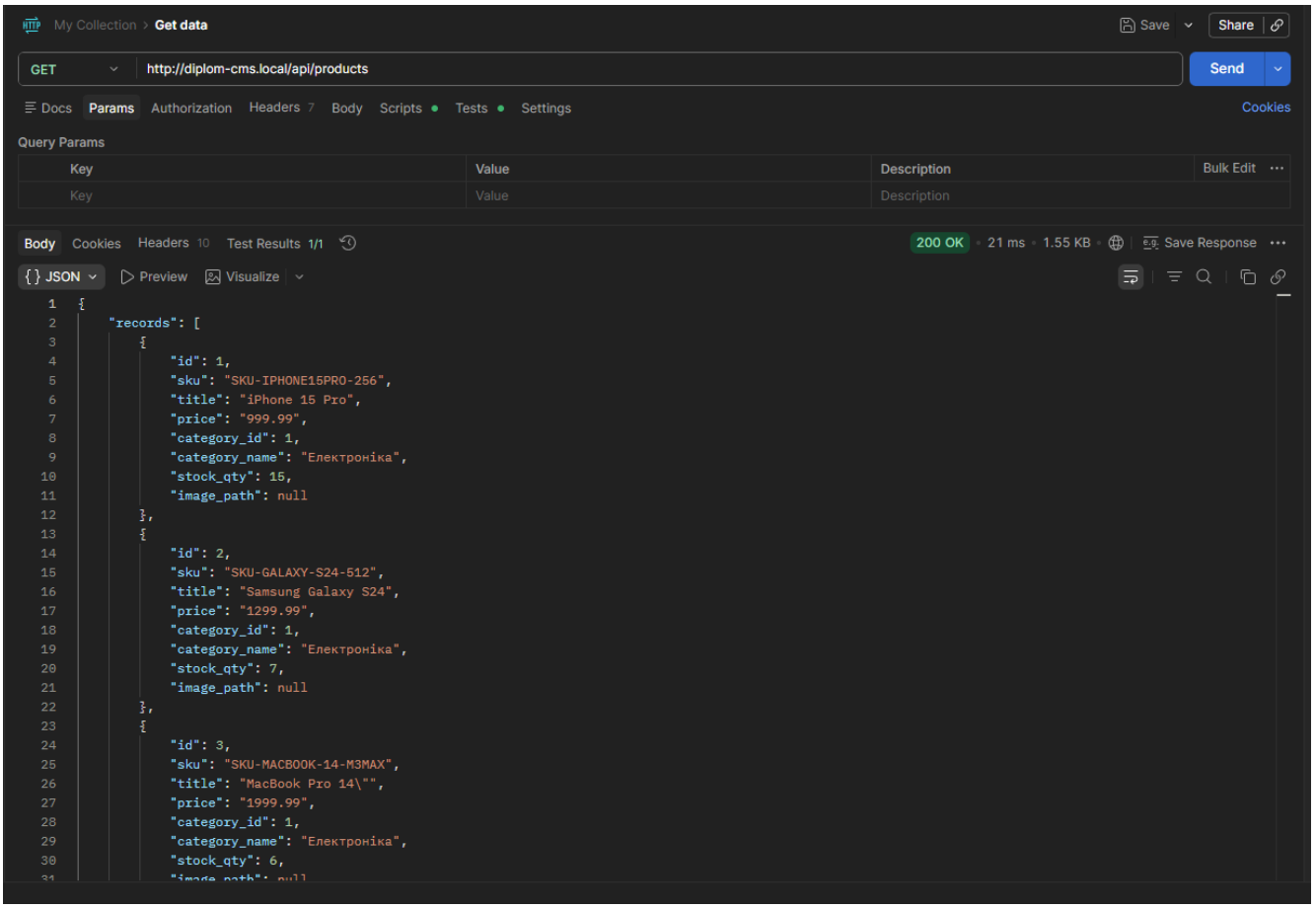


Рис. 3.5 Верифікація кінцевої точки отримання каталогу товарів у середовищі Postman

Паралельно з тестуванням бекенду було проведено кросбраузерне та адаптивне тестування клієнтського UI-інтерфейсу за допомогою інструментів розробника (Google Chrome DevTools та Safari Web Inspector). Симуляція перегляду адмін-панелі на екранах із різною роздільною здатністю (від широкоформатних десктопних моніторів до мобільних пристроїв із шириною екрана 320 пікселів) довела абсолютну гнучкість застосованої CSS-сітки Bootstrap 5. Усі таблиці, модальні вікна та елементи навігації автоматично трансформуються у зручний мобільний формат (Stacking), що дозволяє адміністратору повноцінно керувати інтернет-магазином зі смартфона.

Окремим критичним етапом стало стрес-тестування системи обробки помилок (Error Handling) на стороні клієнта. Моделювалися сценарії нетипової поведінки користувача: спроби відправки порожніх форм створення товару, завантаження непідтримуваних типів файлів у модуль обробки зображень

(наприклад, текстових документів .pdf або .docx замість графічних форматів) та введення некоректних фінансових значень (наприклад, від'ємна ціна товару або текстові символи у полі залишку на складі).

В усіх змодельованих випадках фреймворк Vue.js успішно перехоплював виключення ще на стадії валідації локального стану компонента (State). Система миттєво генерувала відповідні візуальні сповіщення (Toasts/Alerts) для адміністратора із чітким описом помилки, при цьому блокуючи відправку хибних даних на сервер. Додатково було перевірено механізм глобальних перехоплювачів (Interceptors): при штучному видаленні JWT-токена з LocalStorage, додаток миттєво ідентифікував втрату авторизації та перенаправляв користувача на захищений екран входу.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи здійснено безпосередню програмну реалізацію та комплексне тестування спроєктованої кастомної системи управління контентом інтернет-магазину. За результатами виконання практичної частини сформульовано такі висновки:

1. Здійснено архітектурну реалізацію захищеного серверного бекенду мовою PHP. Відмова від використання готових монолітних фреймворків дозволила створити оптимізований механізм маршрутизації (Front Controller), який забезпечує високошвидкісну обробку RESTful-запитів. Впроваджено багатокомпонентну підсистему інформаційної безпеки: безсесійну авторизацію на базі стандарту JWT, алгоритм криптографічного хешування паролів BCrypt та надійні механізми захисту від SQL-ін'єкцій за допомогою технології підготовлених виразів PDO.

2. Імплементовано серверний модуль автоматичної обробки медіаресурсів із застосуванням GD Library. Програмний механізм забезпечує сувору валідацію форматів завантажуваних файлів та здійснює алгоритмічне масштабування зображень товарів до стандартизованих мініатюр (300x300

пікселів), що суттєво оптимізує швидкість завантаження сторінок каталогу для кінцевих користувачів.

3. Програмно реалізовано клієнтську інфраструктуру панелі адміністратора за архітектурою Single Page Application (SPA). Використання реактивного фреймворку Vue.js спільно з адаптивною компонентною базою Bootstrap 5 дозволило створити сучасний інтерактивний інтерфейс. Застосування Fetch API для асинхронного обміну даними гарантує миттєве оновлення аналітичної інформації на Дашборді без блокування інтерфейсу чи перезавантаження сторінок.

4. Виконано всебічне функціональне та інтеграційне тестування розробленого програмного комплексу. Валідація API-шлюзів за допомогою середовища Postman підтвердила точність генерації статус-кодів та безпомилкову обробку JSON-пакетів при виконанні CRUD-операцій. Кросбраузерне тестування інтерфейсу довело його стовідсоткову адаптивність, стабільність роботи системи обробки помилок та загальну відповідність кастомної платформи всім висунутим на етапі проєктування технічним вимогам

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальне науково-практичне завдання щодо оптимізації процесів адміністрування електронної комерції, мінімізації серверного навантаження та підвищення ефективності управління товарним контентом інтернет-магазину. Мети було досягнуто шляхом проєктування, програмної реалізації та всебічного тестування кастомної системи управління контентом із розділеною архітектурою (Headless CMS) на базі прогресивних вебтехнологій.

За результатами виконання теоретичних, аналітичних та практичних етапів дослідження одержано такі основні висновки та результати:

1. На основі системного аналізу предметної області електронної комерції та порівняльного огляду існуючих монолітних рішень (WordPress, Drupal, Joomla) визначено їхні ключові архітектурні недоліки при масштабуванні під специфічні комерційні проєкти. Встановлено, що традиційні платформи створюють надлишкове апаратне навантаження на сервери через системний оверхед, жорстко пов'язують інтерфейс користувача із серверною логікою та мають високу вразливість до кібератак. Обґрунтовано доцільність переходу до концепції Headless CMS, яка забезпечує повну ізоляцію компонентів та суттєво підвищує швидкість завантаження вебсторінок.

2. Спроєктовано та функціонально обґрунтовано легковаговий, але високопродуктивний технологічний стек платформи, що базується на використанні мови програмування PHP та реляційної СУБД MySQL на стороні сервера, а також інструментів JavaScript-фреймворку Vue.js та бібліотеки стилів Bootstrap 5 на стороні клієнта. Розроблено архітектурну модель взаємодії за фундаментальним патерном MVC із реалізацією єдиної точки входу (Front Controller) та RESTful-маршрутизацією асинхронних запитів.

3. Розроблено та оптимізовано фізичну структуру бази даних diplom_cms, яка складається з 6 логічно пов'язаних таблиць, що забезпечують повний цикл

обліку номенклатури, користувачів та логування аудит-подій. Структуру спроєктовано із дотриманням вимог третьої нормальної форми (3NF), що гарантує відсутність аномалій модифікації та надмірності даних. Для забезпечення абсолютної фінансової цілісності транзакцій імплементовано архітектурний патерн денормалізації Data Snapshot. Збереження складу кошика покупця у форматі JSON безпосередньо в реєстрі замовлень унеможливило викривлення історичної звітності підприємства при подальших динамічних змінах цін чи видаленні товарів із каталогу.

4. Програмно реалізовано серверну логіку (Backend REST API) мовою PHP 8 без залучення сторонніх важких фреймворків, що дозволило досягти максимальної швидкості виконання операцій CRUD та мінімального часу відгуку сервера. Інтегровано спеціалізований модуль обробки медіафайлів на базі GD Library, який здійснює сувору валідацію завантажень за MIME-типами та автоматичну генерацію оптимізованих графічних мініатюр (thumbnails) розміром 300x300 пікселів, що суттєво заощаджує мережевий трафік користувачів.

5. Петудовано реактивний інтерфейс адміністративної панелі інтернет-магазину за концепцією Single Page Application (SPA) на базі фреймворку Vue.js. Завдяки використанню технології двостороннього зв'язування даних та обчислюваних властивостей (computed) реалізовано інтерактивний аналітичний дашборд статистики, який миттєво відображає фінансову вартість складу та дефіцитні позиції в режимі реального часу без надлишкових запитів до бази даних, забезпечуючи високу ергономіку системи.

6. Впроваджено багаторівневу підсистему забезпечення інформаційної безпеки платформи. Захист від найпоширеніших уразливостей реалізовано шляхом запобігання SQL-ін'єкціям через використання підготовлених виразів PDO, автоматичного екранування HTML-контенту від XSS-атак, а також криптографічного одностороннього хешування паролів алгоритмом BCrypt із динамічною сіллю. Безпеку сесій та захист від CSRF-атак гарантовано впровадженням безсесійних підписаних токенів JSON Web Tokens (JWT).

7. Проведено комплексне функціональне та інтеграційне тестування розробленого програмного продукту. Верифікація кінцевих точок серверного API за допомогою інструментарію Postman підтвердила 100% коректність обробки HTTP-методіів та повернення валідних статус-кодів і JSON-пакетів. Кросбраузерне тестування SPA-інтерфейсу довело стабільність асинхронного обміну даними через Fetch API, повну адаптивність графічних компонентів під мобільні пристрої та готовність системи до розгортання в реальних умовах операційної діяльності підприємств цифрової комерції.

ПЕРЕЛІК ПОСИЛАНЬ

1. Галатенко, В. Б. Основи веб-розробки та проектування клієнт-серверних архітектур. Київ: Політехніка, 2022.
2. Ковальов, Ю. М., та ін. Проєктування інформаційних систем та баз даних. Львів: Новий Світ-2000, 2023.
3. Сидоренко, О. В., Петренко, Д. М. Інженерія програмного забезпечення: об'єктно-орієнтований підхід та UML-моделювання. Харків: ХНУРЕ, 2021.
4. Мельник, А. О. Архітектурні патерни та чиста архітектура у сучасній веб-розробці. Вчені записки ТНУ імені В. І. Вернадського. Серія: Технічні науки, 2023.
5. Шевченко, О. І., Бондар, В. М. Порівняльний аналіз монолітних та headless-систем управління контентом для платформ електронної комерції. Сучасні комп'ютерні системи та мережі, 2024.
6. PHP Documentation. PHP Manual. PHP, 2024. [Електронний ресурс]. URL: https://www.php.net/docs.php
7. Oracle Corporation. MySQL Reference Manual. Oracle, 2024. [Електронний ресурс]. URL: https://dev.mysql.com/doc/
8. Vue.js Foundation. Vue.js 3 Documentation. Vue.js, 2024. [Електронний ресурс]. URL: https://vuejs.org/guide/introduction.html
9. Fielding, R. T. Architectural Styles and the Design of Network-based Software Architectures. University of California, 2000.
10. JWT.io. Introduction to JSON Web Tokens. Auth0, 2024. [Електронний ресурс]. URL: https://jwt.io/introduction
11. Кнут, Д. Е. Мистецтво програмування. Том 3. Сортування та пошук. Львів: БаК, 2020.
12. Річардсон, К. Патерни розробки мікросервісів та розділених веб-додатків. Київ: Діалектика, 2022.

13. Смирнов, В. А. Методи захисту веб-орієнтованих інформаційних систем від уразливостей SQL-ін'єкцій та XSS-атак. Кібербезпека та захист інформації, 2023.

14. Postman API Platform. Postman Learning Center. Postman, 2024. [Електронний ресурс]. URL: https://learning.postman.com/

15. Кіч Р. О. Забезпечення цілісності даних під час обробки транзакцій у веб-системах електронної комерції // Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. – К.: ДУІКТ, 2026.

16. Кіч Р. О. Застосування інтелектуальних методів обробки даних у кастомних системах управління контентом для електронної комерції // Всеукраїнській науково-практичній конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті»

ДОДАТОК А. ДЕМОНСТРАТИВНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Технологій цифрового розвитку

Система управління контентом інтернет-магазину з функціоналом адміністрування каталогу, замовлень та користувачів

Виконав студент 4 курсу
Групи ТЦР-41
Кіч Роман Олександрович
Керівник роботи
кандидат технічних наук, доцент кафедри ТЦР
Аронов Андрій Олексійович

Київ-2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи - удосконалення процесів адміністрування електронної комерції та підвищення ефективності управління товарним контентом і замовленнями шляхом розробки та впровадження спеціалізованої системи управління контентом (CMS).

Об'єкт дослідження - процеси управління інформаційнимнаповненням, замовленнямита клієнтською базою в електроннійкомерції.

Предмет дослідження - методи, алгоритмита програмнізасобирозробкиадмін-панелі CMS з використаннямPHP та MySQL.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Аналіз предметної області та існуючих рішень для e-commerce.
2. Проектування архітектури бази даних для зберігання товарів, категорій та замовлень.
3. Розробка серверної логіки (Backend) на мові PHP
4. Створення інтуїтивно зрозумілого інтерфейсу (Frontend) на JS
5. Інтеграція модулів управління правами користувачів та статусами замовлень.

АНАЛІЗ АНАЛОГІВ

Категорія	WordPress (woo)	Shopify	OpenCart	Розроблена CMS
Вартість	Безкоштовно (плагіни платні)	Щомісячна підписка	Безкоштовно (модулі платні)	Безкоштовно
Гнучкість	Середня	Обмежена платформою	Середня	Максимальна (Custom)
Швидкість роботи	Залежить від плагінів	Висока	Середня	Оптимізована під завдання
Складність налаштування	Висока	Низька	Середня	Відповідно до ТЗ

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

1. **PHP 8.2** - Серверна частина, обробка запитівта взасмодія з БД.
2. **MySQL** - Реляційна база данихдля структурованого зберігання контенту.
3. **JavaScript** - Динамічне оновлення інтерфейсута клієнтська валідація.
4. **CSS/HTML5** - Адаптивна верстка адмін-панелі для різнихпристроїв.

ФУНКЦІОНАЛЬНІ ВИМОГИ

1. Авторизація та чітке розмежування прав доступу користувачів (Адміністратор, Менеджер, Глядач).
2. Реалізація повного циклу управління (CRUD) товарним каталогом, включаючи завантаження медіафайлів, та ієрархією категорій.
3. Обробка замовлень клієнтів із можливістю динамічного управління їхніми статусами.
4. Ведення автоматичного журналу аудиту (історії) змін статусів замовлень із фіксацією відповідального оператора та часу події.
5. Генерація агрегованих аналітичних даних для головної панелі (Dashboard) у режимі реального часу.

НЕФУНКЦІОНАЛЬНІ ВИМОГИ

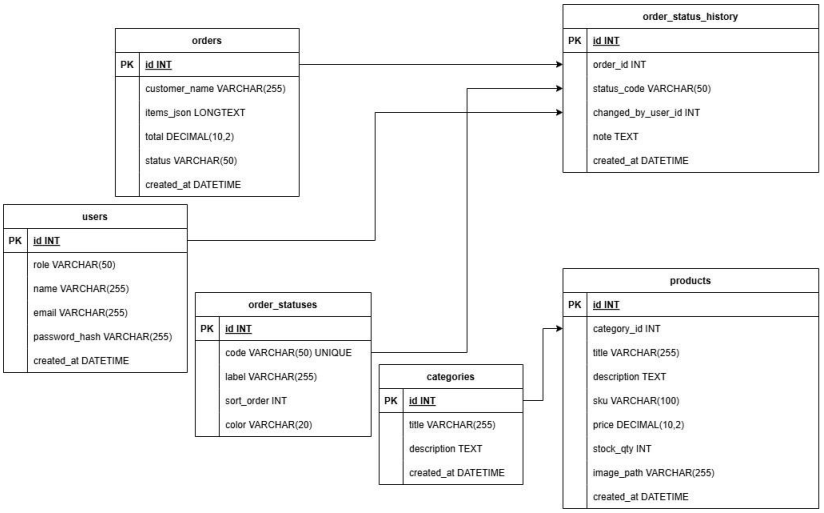
1. **Архітектура:** Використання розділеної (Headless) архітектури на базі REST API для повної незалежності клієнтської (Frontend) та серверної (Backend) частин.

2. **Продуктивність:** Забезпечення миттєвої реактивності інтерфейсу та мінімізація мережевого трафіку завдяки підходу Single Page Application (Vue.js).

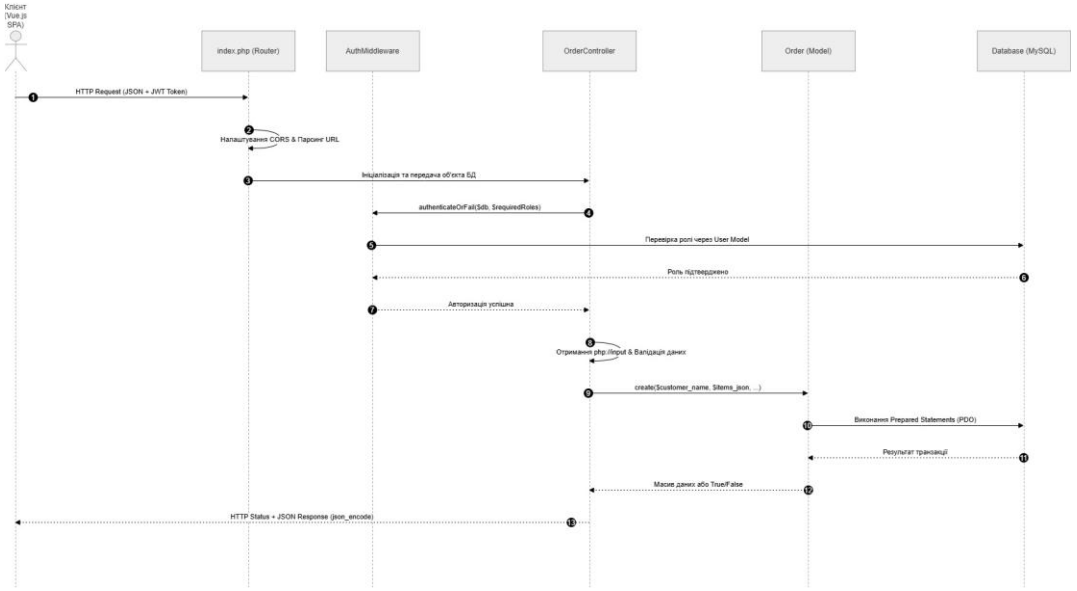
3. **Безпека:** Захист бази даних від SQL-ін'єкцій за допомогою об'єктів PDO, безпечне хешування паролів (BCrypt) та криптографічний захист сесій (JWT).

4. **Цілісність даних:** Гарантування незмінності фінансової історії замовлень (патерн JSON Snapshot) незалежно від майбутніх змін цін у каталозі товарів.

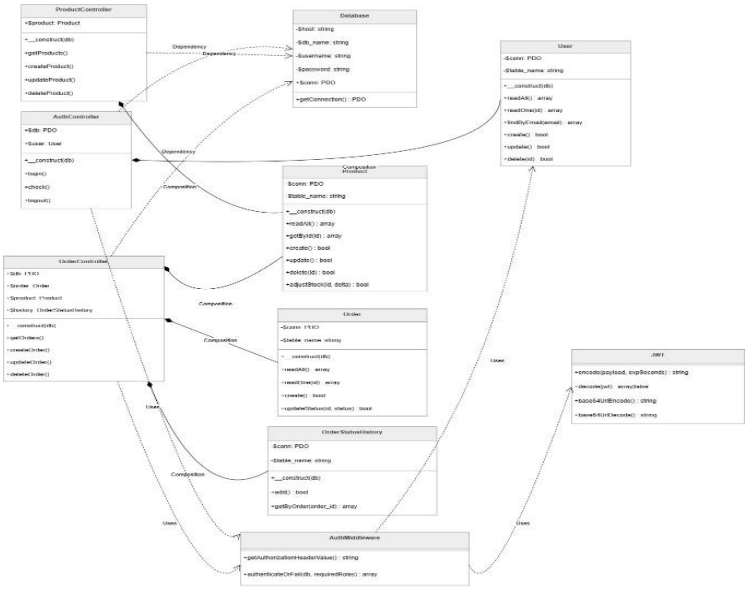
СТРУКТУРА БАЗИ ДАНИХ (ER-ДІАГРАМА)



АРХІТЕКТУРНИЙ ПОТІК MVC (ДІАГРАМА ПОСЛІДОВНОСТІ)



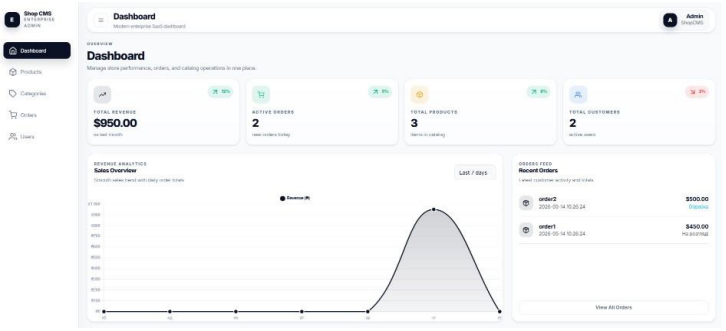
ДІАГРАМА КЛАСІВ (CLASS DIAGRAM)



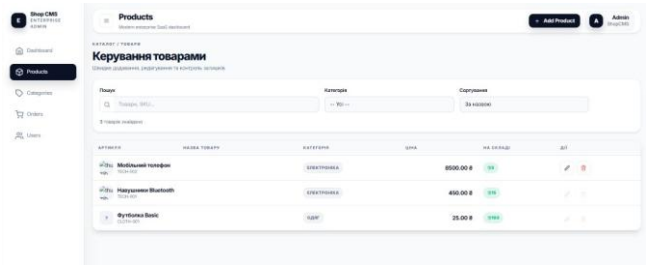
ГОЛОВНА СТОРІНКА

Панель Dashboard забезпечує швидкий перегляд ключових показників:

1. Загальний дохід за період.
2. Кількість активних замовлень.
3. Статистика продажів (Revenue Analytics)
4. Останні операції у системі.



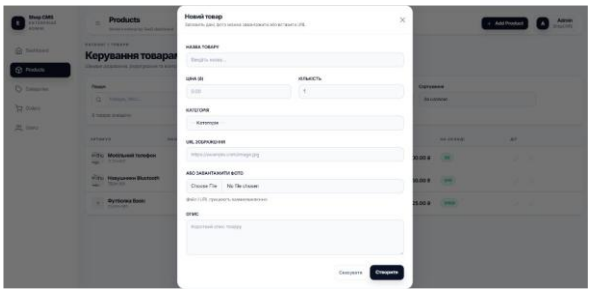
КЕРУВАННЯ ТОВАРАМИ



Централізований каталог товарів:

1. Фільтрація за категоріями та пошук за артикулом (SKU)
2. Відображення залишків на складі в режимі реального часу.
3. Швидке редагування цін та описів.
4. Масове управління статусами наявності.

ДОДАВАННЯ ТОВАРУ



Інтерфейс наповнення каталогу:

1. Завантаження фото.
2. Вибір категорії та атрибутів товару.
3. Встановлення ціни та складських лімітів.
4. Валідація полів форми на рівні Backend та Frontend.

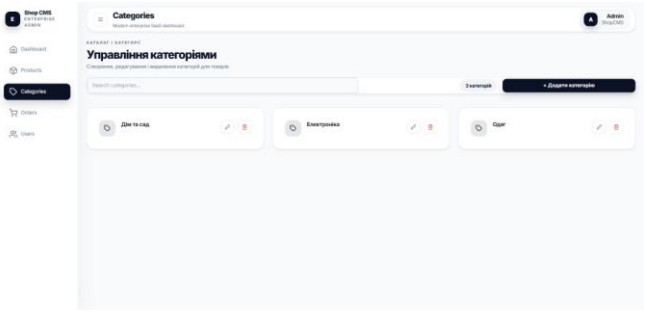
УПРАВЛІННЯ КАТЕГОРІЯМИ

Гнучка структура каталогу:

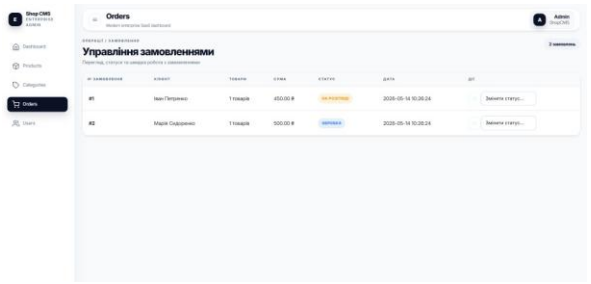
1. Створення ієрархіччної структури розділів.

2. Швидка деактивація цілих груп товарів.

3. Візуальний менеджер категорій.



УПРАВЛІННЯ ЗАМОВЛЕННЯМИ



Рабоче місце менеджера замовлень:

1. Моніторинг нових замовлень без оновлення сторінки.

2. Зміна статусів (На розгляді, Обробка, Завершено, Скасовано).

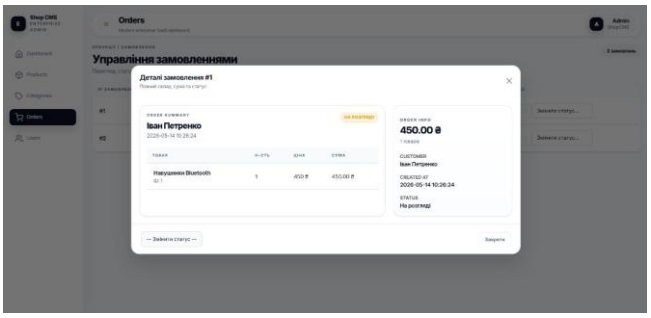
3. Прив’язка замовлення до клієнтського профілю.

4. Формування звітної документації

ДЕТАЛІ ЗАМОВЛЕННЯ

Повний огляд конкретної транзакції:

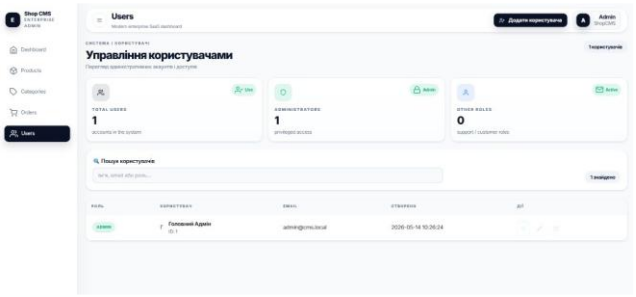
1. Інформація про доставку та оплату.
2. Детальний перелік товарів у кошику з цінами на момент замовлення.
3. Історія зміни статусів та примітки менеджера.
4. Контактні дані покупця для швидкого зв’язку.



УПРАВЛІННЯ КОРИСТУВАЧАМИ

Модуль адміністрування облікових засобів дозволяє:

1. Переглядати повний список зареєстрованих користувачів.
2. Керувати ролями (Адміністратор, Менеджер, Клієнт).
3. Здійснювати швидкий пошук за Іменем, email та роллю.



АПРОБАЦІЇ

1. Кіч Р.О. Забезбечення цілісності даних під час обробки тразакцій у веб-системах електронної комерції. // Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація»

2. Кіч Р.О. Застосування інтелектуальних методів обробки даних у кастомних системах управління контентом для електронної комерції. // VI Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті»

ВИСНОВОК

1. Проаналізовано архітектурні особливості та недоліки існуючих монолітних систем управління контентом, обґрунтовано доцільність переходу до розділеної (Headless) архітектури.

2. Спроєктовано архітектуру реляційної бази даних MySQL для надійного зберігання товарів, категорій та замовлень із застосуванням сучасних підходів до цілісності даних.

3. Розроблено серверну логіку (Backend) на мові PHP з використанням архітектури REST API для забезпечення безпечної та швидкої обробки запитів.

4. Створено інтуїтивно зрозумілий інтерфейс (Frontend) на JavaScript у форматі Single Page Application, що гарантує миттєву взаємодію без перезавантаження сторінок.

5. Інтегровано надійні модулі управління правами користувачів (через JWT-авторизацію) та впроваджено гнучку систему керування статусами замовлень.

6. Таким чином, поставлену мету успішно досягнуто: розроблено спеціалізовану систему управління контентом (CMS), яка дозволила удосконалити процеси адміністрування електронної комерції та суттєво підвищити ефективність управління товарним каталогом і замовленнями.

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
// index.php
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS");

require_once 'config/Database.php';
require_once 'controllers/ProductController.php';

$database = new Database();
$db = $database->getConnection();

$uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
$uri_parts = explode('/', trim($uri, '/'));
$resource = end($uri_parts);
$requestMethod = $_SERVER["REQUEST_METHOD"];

switch ($resource) {
    case 'products':
        $controller = new ProductController($db);
        if ($requestMethod === 'GET') {
            $controller->getProducts();
        } elseif ($requestMethod === 'POST') {
            $controller->createProduct();
        } elseif ($requestMethod === 'PUT') {
            $controller->updateProduct();
        } elseif ($requestMethod === 'DELETE') {
            $controller->deleteProduct();
        }
        break;

    default:
        http_response_code(404);
        echo json_encode(["message" => "Ендпоінт не знайдено."],
JSON_UNESCAPED_UNICODE);
        break;
}

// helpers/AuthMiddleware.php
require_once __DIR__ . '/JWT.php';
require_once __DIR__ . '/../models/User.php';

function authenticateOrFail($db, $requiredRoles = []) {
    $authHeader = getAuthorizationHeaderValue();

    if (!$authHeader || stripos($authHeader, 'Bearer ') !== 0) {
        http_response_code(401);
        echo json_encode(['message' => 'Токен не надано.'],
JSON_UNESCAPED_UNICODE);
        exit();
    }

    $token = trim(substr($authHeader, 7));
    $payload = JWT::decode($token);

    if (!$payload || !isset($payload['sub'])) {
        http_response_code(401);
        echo json_encode(['message' => 'Невірний або прострочений токен.'],
JSON_UNESCAPED_UNICODE);
        exit();
    }

    $userModel = new User($db);
```

```
    $user = $userModel->readOne($payload['sub']);

    if (!empty($requiredRoles) && !in_array($user['role'], $requiredRoles)) {
        http_response_code(403);
        echo json_encode(['message' => 'Недостатньо прав для виконання
операції.'], JSON_UNESCAPED_UNICODE);
        exit();
    }

    return $user;
}

// controllers/OrderController.php
public function createOrder() {
    $data = json_decode(file_get_contents('php://input'));

    try {
        $this->db->beginTransaction();

        $finalItems = [];
        $total = 0;

        foreach ($normalizedItems as $productId => $qty) {
            $product = $this->product->getById($productId);
            $availableQty = isset($product['stock_qty']) ?
(int)$product['stock_qty'] : 0;

            if ($availableQty >= $qty) {
                $this->product->adjustStock($productId, -$qty);
            }

            $price = isset($product['price']) ? (float)$product['price'] : 0;
            $total += $price * $qty;

            $finalItems[] = [
                'product_id' => $productId,
                'title' => $product['title'],
                'qty' => $qty,
                'price' => $price
            ];
        }

        $items_json = json_encode($finalItems, JSON_UNESCAPED_UNICODE);
        $status = isset($data->status) ? $data->status : 'pending';

        $insertedId = $this->order->create($data->customer_name, $items_json,
round($total, 2), $status);

        if ($insertedId !== false) {
            $history = new OrderStatusHistory($this->db);
            $history->add($insertedId, $status, $createdBy, 'created');

            $this->db->commit();
            http_response_code(201);
            echo json_encode(['message' => 'Замовлення створено.', 'id' =>
$insertedId], JSON_UNESCAPED_UNICODE);
        } else {
            $this->db->rollBack();
            http_response_code(503);
            echo json_encode(['message' => 'Не вдалося створити замовлення.'],
JSON_UNESCAPED_UNICODE);
        }
    } catch (Exception $e) {
        if ($this->db->inTransaction()) {
```

```
            $this->db->rollBack(); // Відкат у разі системної помилки
        }
        http_response_code(500);
        echo json_encode(['message' => 'Помилка створення замовлення: ' . $e-
>getMessage()]);
    }
}

// models/Product.php
public function create($category_id, $sku, $title, $price, $stock_qty, $image_path
= null) {
    $query = "INSERT INTO " . $this->table_name . "
              (category_id, sku, title, price, stock_qty, image_path)
              VALUES (:category_id, :sku, :title, :price, :stock_qty,
:image_path)";

    $stmt = $this->conn->prepare($query);

    $title = htmlspecialchars(strip_tags($title));

    $stmt->bindParam(":category_id", $category_id);
    $stmt->bindParam(":sku", $sku);
    $stmt->bindParam(":title", $title);
    $stmt->bindParam(":price", $price);
    $stmt->bindParam(":stock_qty", $stock_qty);
    $stmt->bindParam(":image_path", $image_path);

    return $stmt->execute();
}

public function readAll() {
    $query = "SELECT p.*, c.title as category_name
              FROM " . $this->table_name . " p
              LEFT JOIN categories c ON p.category_id = c.id
              ORDER BY p.created_at DESC";

    $stmt = $this->conn->prepare($query);
    $stmt->execute();
    return $stmt;
}
computed: {
    filteredProducts() {
        let filtered = this.products.filter(product => {
            const title = (product.title || '').toLowerCase();
            const sku = (product.sku || '').toLowerCase();
            const matchesSearch = title.includes(this.searchQuery.toLowerCase())
||
                                  sku.includes(this.searchQuery.toLowerCase());
            const matchesCategory = !this.selectedCategory ||
String(product.category_id) === String(this.selectedCategory);

            return matchesSearch && matchesCategory;
        });

        if (this.sortBy === 'name') {
            filtered.sort((a, b) => (a.title || '').localeCompare((b.title ||
'')));
        } else if (this.sortBy === 'price-asc') {
            filtered.sort((a, b) => parseFloat(a.price || 0) - parseFloat(b.price
|| 0));
        } else if (this.sortBy === 'price-desc') {
            filtered.sort((a, b) => parseFloat(b.price || 0) - parseFloat(a.price
|| 0));
        } else if (this.sortBy === 'stock') {
```

```
            filtered.sort((a, b) => parseInt(b.stock_qty || 0) -
parseInt(a.stock_qty || 0));
        }

        return filtered;
    }
},
methods: {
    async fetchProducts() {
        try {
            const response = await fetch('/api/products', {
                cache: 'no-store'
            });

            if (!response.ok) {
                this.products = [];
                return;
            }

            const data = await response.json();
            this.products = data.records ? data.records : [];

        } catch (error) {
            console.error("Помилка завантаження даних:", error);
        }
    }
}
-- Таблиця замовлень, яка використовує JSON-поле для зберігання масиву товарів
CREATE TABLE `orders` (
  `id` int NOT NULL AUTO_INCREMENT,
  `customer_name` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `items_json` json DEFAULT NULL,
  `total` decimal(10,2) NOT NULL,
  `status` varchar(64) COLLATE utf8mb4_unicode_ci DEFAULT 'pending',
  `created_at` datetime DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- Таблиця історії зміни статусів (зв'язана з замовленнями та користувачами)
CREATE TABLE `order_status_history` (
  `id` int UNSIGNED NOT NULL AUTO_INCREMENT,
  `order_id` int NOT NULL,
  `status_code` varchar(64) COLLATE utf8mb4_unicode_ci NOT NULL,
  `changed_by_user_id` int DEFAULT NULL,
  `note` text COLLATE utf8mb4_unicode_ci,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  KEY `order_id` (`order_id`),
  KEY `changed_by_user_id` (`changed_by_user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```