Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Кваліфікаційна робота магістра

з теми**: “Кастомізація методів створення вебплатформи для ефективного функціонування інтернет-магазинів”**

Виконав: студент KN1-M24 групи
спеціальності 122 Комп’ютерні науки
Мозолюк Микола Ігорович
Керівник:
Пилипюк Тетяна Михайлівна
доцент кафедри комп’ютерних наук, кандидат фізико-математичних наук, доцент
Рецензент:
Шумиляк Лілія Михайлівна
кандидат технічних наук, доцент кафедри програмного забезпечення комп’ютерних систем Чернівецького національного університету імені Юрія Федьковича

Кам’янець-Подільський, 2025 р.

ЗМІСТ

АНОТАЦІЯ

У даній роботі представлено теоретичні та практичні аспекти розробки вебплатформи для створення інтернет-магазинів із можливістю кастомізації. Дослідження охоплює аналіз вимог до такої системи, порівняльний аналіз існуючих рішень, а також проєктування та розробку адміністративної панелі для управління магазинами та маркетплейсом.

Метою роботи є створення гнучкої вебплатформи, що дозволяє користувачам швидко налаштовувати власний інтернет-магазин, змінювати його дизайн, керувати товарами та інтегруватися в загальний маркетплейс. Під час розробки використовувалися такі технології як: Angular для фронтенду, Node.js та Express.js для бекенду, MongoDB для зберігання даних, а також шардінг та індекси для швидкого пошуку. Для стилізації застосовано Angular Material і Tailwind CSS, що забезпечує адаптивний та зручний інтерфейс.

Результати роботи підтверджують ефективність використання розробленої платформи для швидкого розгортання та налаштування інтернет-магазинів, що спрощує процес входу бізнесу в онлайн режим.

Ключові слова: вебплатформа, інтернет-магазин, адміністративна панель, Angular, MEAN, Node.js, MongoDB, Sharding, кастомізація, маркетплейс, управління товарами, адаптивний інтерфейс.

ABSTRACT

This work presents the theoretical and practical aspects of developing a web platform for creating customizable online stores. The study covers the analysis of system requirements, a comparative review of existing solutions, as well as the design and development of an administrative panel for managing individual stores and the marketplace.

The aim of the work is to create a flexible web platform that enables users to quickly configure their own online store, adjust its design, manage products, and integrate it into a unified marketplace. The development utilizes the following technologies: Angular for the frontend, Node.js and Express.js for the backend, MongoDB for data storage, along with sharding and indexing for fast search. Angular Material and Tailwind CSS were applied for styling, ensuring a responsive and user-friendly interface.

The results confirm the efficiency of the developed platform for rapid deployment and customization of online stores, significantly simplifying the process of bringing businesses online.

Keywords: web platform, online store, administrative panel, Angular, MEAN, Node.js, MongoDB, sharding, customization, marketplace, product management, responsive interface.

ВСТУП

Сучасний ринок електронної комерції розвивається стрімко, і все більше компаній переходять в онлайн-середовище. Але власникам інтернет-магазинів важливо не лише створювати платформу для продажів, але й мати прості інструменти для керування. Підприємці, що працюють у сфері e-commerce, дуже хочуть автоматизувати процеси управління товарами, замовленнями та спілкування з клієнтами.

Актуальність. Можливість адаптувати інтернет-магазин під потреби власника, що дозволяє відрізнятися від конкурентів, є важливим компонентом успішного функціонування інтернет-магазину. Більшість сучасних платформ пропонують лише базові налаштування, обмежуючи кількість можливостей для налаштування. Це робить створення веб-платформи актуальним, що дозволяє власникам магазинів швидко запускати проєкти та гнучко налаштовувати їхній дизайн і функції.

Мета. Основною метою кваліфікаційної роботи є створення вебплатформи для створення та кастомізації інтернет-магазинів, яка забезпечить взаємодію з маркетплейсом, керування товарами, замовлення та гнучкі налаштування дизайну.

Для досягнення мети виконувалися завдання:

1. Провести аналіз існуючих варіантів створення онлайн-магазинів і їх кастомізації.
2. Розробити архітектуру вебплатформи, що дозволяє користувачам створювати власні магазини.
3. Реалізувати адміністративну панель для керування магазином, товарами та замовленнями.
4. Забезпечити можливість кастомізації інтерфейсу магазину (шаблони, кольори, шрифти, логотип тощо).
5. Реалізувати інтеграцію магазину з загальним маркетплейсом, де користувачі можуть розміщувати свої товари.

6. Виконати тестування вебплатформи та оцінити її ефективність.

Об’єктом дослідження є процес створення та функціонування вебплатформ для інтернет-магазинів, а **предметом** – методи розробки, кастомізації та інтеграції такої платформи з маркетплейсом.

Практичне значення. Вебплатформа, яка надає підприємцям широкий вибір налаштувань, значно спрощує процес створення та управління онлайн-магазинами. Використання сучасного технологічного стеку MEAN [2], який включає MongoDB, Express.js, Angular і Node.js, у поєднанні з MongoDB гарантує високу продуктивність, масштабованість і ефективність роботи системи.

Апробація результатів. Основні результати кваліфікаційного дослідження були оприлюднені під час виступу на студентській науково-практичній конференції Кам’янець-Подільського національного університету імені Івана Огієнка (2025 р.). У доповіді на тему «Ефективність індивідуалізації інтернет-магазинів для підвищення конверсії та залучення клієнтів» було представлено ключові теоретичні та практичні напрацювання роботи, зокрема:

- аналіз сучасних підходів до кастомізації вебплатформ та методів підвищення конверсії;
- результати порівняння моделей індивідуалізації на маркетплейсах (Amazon, eBay);
- огляд ефективності застосування алгоритмів машинного навчання для персоналізації контенту;
- представлено авторську концепцію побудови кастомізованої вебплатформи з використанням MEAN-стеку та мікросервісної архітектури.

Після виступу результати дослідження отримали позитивні відгуки учасників та були опубліковані у Віснику фізико-математичних наук Кам’янець-Подільського університету імені Івана Огієнка 2024 року [14].

Структура роботи. Кваліфікаційна робота магістра складається з вступу, трьох розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1 ОЗНАЙОМЛЕННЯ З ВИМОГАМИ ДО АДМІНІСТРАТИВНОЇ ПАНЕЛІ

Успішна розробка будь-якої програмної системи починається з правильної постановки задачі, аналізу потреб користувачів і визначення ключових вимог до функціональності. Важливо узгодити всі нюанси проєкту, щоб створити ефективне та зручне рішення для кінцевих користувачів.

1.1 Постановка задачі

Для успішної розробки вебплатформи для інтернет-магазинів необхідні чіткі вимоги, визначення функціональних можливостей та розуміння потреб кінцевих користувачів. Щоб гарантувати розробку унікального інтернет-магазину, його налаштування та подальше управління всіма бізнес-процесами, спочатку необхідно розробити детальну концепцію системи. Метою платформи є надання інструментів управління магазином та адміністративної панелі, щоб підприємці могли легко розпочати власний бізнес в сфері електронної комерції без залучення експертів з розробки. Основними цілями проекту є:

- забезпечення можливості автоматичного створення інтернет-магазинів за допомогою готових шаблонів;
- створення зручної адміністративної панелі для управління замовленнями, клієнтами, категоріями, товарами та аналітикою;
- надання ресурсів для налаштування магазину (зміна структури інтерфейсу, колірної гами, шрифтів та логотипу);
- розробка архітектури, що полегшує інтеграцію, масштабування та розширення додаткової функціональності;
- можливість об'єднання роздрібних продавців в єдиний ринок, де клієнти можуть розміщувати товари для спільного продажу.

Подамо характеристику об’єкта автоматизації.

Об'єктом автоматизації є веб-платформа для створення інтернет-магазинів під білою маркою, яка дозволяє користувачам проектувати магазини, змінювати їхній інтерфейс та контролювати бізнес-процеси без необхідності програмування.

Важливі особливості продукту:

- Прикладами типів користувачів є власники магазинів, адміністратори та кінцеві споживачі.
- Основними функціями є створення магазину, управління товарами та замовленнями, редагування дизайну та інтеграція з ринком.
- Технологічна основа: фронтенд - Angular; бекенд - Node.js [7] та Express; база даних - MongoDB; API - REST.
- Основною вимогою до інтерфейсу це адаптивний веб-додаток, оптимізований для різних типів пристроїв.

Визначимо вимоги до програми.

Для забезпечення ефективної роботи системи необхідно дотримуватися таких вимог:

- Функціональні вимоги**:**
    1. Реєстрація та автентифікація власників магазинів.
    2. Автоматичне створення інтернет-магазину на основі обраного шаблону.
    3. Редагування каталогу: додавання/видалення товарів, створення категорій, керування відгуками.
    4. Управління замовленнями та клієнтськими даними.
    5. Налаштування дизайну магазину (кольори, шрифти, логотип, компонування).
- Нефункціональні вимоги**:**
    1. Масштабованість: підтримка великої кількості одночасно активних магазинів.

2. Продуктивність: мінімальна затримка відповіді API, оптимізовані процеси пошуку та фільтрації.
3. Безпека: автентифікація, авторизація, обмеження доступу, шифрування ключових даних.
4. Надійність: можливість відновлення після збоїв, резервне копіювання даних.
5. Кросплатформність: коректна робота на ПК, планшетах і смартфонах.

1.2 Аналіз підходів для створення платформи

Вибір відповідних технологій, які гарантують високу продуктивність, масштабованість, безпеку та зручність використання, необхідний для розробки веб-платформи для інтернет-магазинів. Щоб обґрунтувати вибір найкращих технологій для конкретного проекту, у цьому підрозділі буде розглянуто стек MEAN, який включає MongoDB, Express.js, Angular і Node.js, а також альтернативи.

Розглянемо стек MEAN, який використовується для розробки вебплатформи.

MEAN – це популярна група технологій, яка використовується для розробки динамічних веб-додатків, яка включає MongoDB, Express.js, Angular і Node.js. Цей стек дозволяє виконувати повну роботу з JavaScript на клієнтській і серверній частинах додатка.

MongoDB – це база даних NoSQL, орієнтована на документи, яка дозволяє зберігати дані у вигляді документів JSON. Вона пропонує масштабованість, гнучкість у моделюванні даних і високу продуктивність при обробці великих обсягів даних, таких як дані користувачів, замовлення та каталоги товарів.

Express.js – це невеликий фреймворк Node.js, який полегшує створення додатків і API на сервері. Він пропонує просте управління маршрутами та

обробку HTTP-запитів, що є важливим для обробки даних замовлень, авторизації користувачів і інтеграції з іншими сервісами.

Angular – потужний фреймворк для фронтенд-розробки на основі TypeScript. Він дозволяє створювати динамічні та змінні інтерфейси, наприклад панель адміністратора та магазин для покупців. Крім того, Angular дозволяє використовувати перевикористання компонентів і модульну архітектуру, що прискорює процес розробки.

Node.js – серверне середовище виконання JavaScript, яке може обробляти велику кількість запитів одночасно з низькою затримкою, що особливо корисно для роботи з інтернет-магазинами. Node.js дозволяє швидко обробляти запити до бази даних, обробку платежів та інші серверні функції завдяки своїй подійно-орієнтованій архітектурі.

Переваги стека MEAN:

- Повна робота з JavaScript на всіх рівнях додатка.
- Висока продуктивність і масштабованість.
- Швидкість розробки завдяки перевикористанню коду.
- Велика спільнота розробників і підтримка.

Недоліки стека MEAN:

- MongoDB не підтримує складні транзакції, що може бути важливим для платіжних систем.
- Висока швидкість розробки може призвести до неструктурованого коду, якщо не дотримуватися правил організації проєкту.

Розглянемо альтернативи стеку MEAN, серед яких одним із найпоширеніших є стек LAMP (Linux, Apache, MySQL, PHP).

LAMP є класичним стеком веброзробки, який використовує MySQL як реляційну базу даних і PHP як серверну мову програмування. Цей стек добре працює з класичними CMS (наприклад, WordPress) і підходить для проектів з меншою динамічністю інтерфейсу.

Переваги стеку LAMP:

- Висока стабільність і перевірена часом архітектура.

- Підтримка великої кількості хостинг-провайдерів.

Недоліки стеку LAMP:

- Обмежені можливості для створення динамічних SPA (односторінкових додатків).
- PHP не настільки гнучкий, як Node.js, для роботи з асинхронними запитами.

Щодо стеку MERN (MongoDB, Express.js, React, Node.js), то MERN і MEAN схожі, але вони використовують React замість Angular. React покращує продуктивність і робить інтеграцію зі сторонніми бібліотеками простішою. Але для великих проектів React може знадобитися більше ручного налаштування управління станом. Це можна зробити, використовуючи Redux.

Переваги стеку MERN:

- Гнучкий компонентний підхід до розробки інтерфейсу.
- Висока продуктивність завдяки віртуальному DOM.

Недоліки стеку MERN:

- Більша складність в організації проєкту через відсутність суворої структури.
- Менше вбудованих рішень, що вимагає додаткових бібліотек.

Стек MEVN (MongoDB, Express.js, Vue.js, Node.js) використовує Vue.js на додаток до Angular або React. Vue.js відомий своєю високою продуктивністю та простотою у використанні. Він підходить для швидкої розробки прототипів і проєктів розміром від невеликого до середнього.

Переваги стеку MEVN:

- Легкий у вивченні, простий в інтеграції.
- Висока продуктивність і гнучкість.

Недоліки стеку MEVN:

- Менша екосистема у порівнянні з Angular або React.
- Менше готових рішень для складних корпоративних проєктів.

Обґрунтуємо вибір стека MEAN, який був обраний з таких причин:

- Angular пропонує широкий спектр інструментів для створення динамічних інтерфейсів, які ідеально підходять для магазину покупців і адміністративної панелі.
- Додаток можна легко масштабувати за допомогою Node.js, щоб підтримувати велику кількість користувачів.
- MongoDB надає гнучкість зберігання даних, що особливо корисно для динамічних каталогів товарів і даних про користувачів.
- За рахунок мінімалістичного підходу до маршрутизації та обробки запитів Express.js значно спрощує розробку серверної частини.

Таким чином, при розробці вебплатформ інтернет-магазинів використання стека MEAN дозволяє забезпечити високу продуктивність, масштабованість і гнучкість системи, а також значно скоротити час розробки завдяки уніфікованому використанню JavaScript на всіх рівнях додатка.

1.3 Вибір архітектури вебплатформи

Архітектура вебплатформи має вирішальне значення для продуктивності, масштабованості, безпеки та зручності подальшого обслуговування системи. Серед популярних архітектурних підходів для розробки веб-платформи інтернет-магазину є монолітна архітектура, мікросервіси та архітектура з розділеним фронтендом і бекендом (SPA + API).

Щодо монолітної архітектури, то вона дозволяє об’єднати всі частини додатка, включаючи фронтенд, бекенд і бізнес-логіку, в одному коді. Цей метод простий у використанні та підтримується на початкових етапах розробки.

Переваги монолітної архітектури:

- Простота в розробці та розгортанні.
- Менше накладних витрат на управління зв’язком між компонентами.
- Уніфікований кодовий простір, що спрощує налагодження.

Недоліки монолітної архітектури:

- Обмежена масштабованість: зміни в одній частині додатка можуть впливати на інші компоненти.
- Ускладнене обслуговування у великих проєктах, оскільки зміни можуть потребувати повторного розгортання всього додатка.
- Складність у впровадженні нових технологій, оскільки всі компоненти тісно пов’язані.

Монолітна архітектура не підходить для веб-платформи інтернет-магазину, яка повинна підтримувати високі навантаження та швидко адаптуватися до змін, які вимагають компанія.

Мікросервіси поділяють додаток на менші незалежні сервіси, кожен з яких відповідає за різні бізнес-логіки, такі як авторизація користувачів, обробка замовлень і управління товарами. Кожен сервіс може бути розроблений, тестований і розгорнутий індивідуально.

Переваги мікросервісної архітектури:

- Висока масштабованість: можна масштабувати окремі сервіси залежно від навантаження.
- Гнучкість у виборі технологій: кожен сервіс може бути реалізований з використанням оптимальної технології.
- Зручність у підтримці: зміни в одному сервісі не впливають на інші компоненти.

Недоліки мікросервісної архітектури:

- Складність в управлінні: необхідна організація комунікацій між сервісами через API.
- Вищі вимоги до DevOps практик: потреба в налаштуванні CI/CD та управлінні контейнерами.
- Збільшення затримок через мережеву взаємодію між сервісами.

Порівняння з монолітною архітектурою подамо в таблиці 1.1.

Таблиця 1.1

Порівняльна таблиця архітектурних підходів

Параметр	Мікросервіси	Моноліт
Масштабованість	Висока	Обмежена
Залежності	Мінімальні	Високі
Розгортання	Незалежне	Суцільне
Підтримка	Простіше	Складніше

У таблиці 1.1 можна побачити явні переваги мікросервісної архітектури, тому її обрано для забезпечення високої гнучкості, масштабованості та надійності.

Таким чином, поєднання Angular, Node.js з Express, MongoDB і мікросервісної архітектури забезпечить високу продуктивність, масштабованість і ефективність розробки платформи інтернет-магазинів.

1.4 Визначення структури вхідних даних

Інформація, яку користувачі або система надсилають на платформу для обробки, називається вхідними даними. Правильна робота функціональних модулів, створення каталогу товарів, обробка замовлень, управління налаштуваннями дизайну та взаємодія з ринком залежать від точного визначення структури вхідних даних.

Основні категорії вхідних даних складаються з:

- Інформації про користувача:
  1) ім'я користувача;
  2) номер телефону та адреса електронної пошти;
  3) хешований пароль;
  4) функція в системі (покупець, адміністратор та власник магазину).

- Деталі продукту:
  1) назва продукту;
  2) опис;
  3) ціна;
  4) кількість на складі;
  5) зображення (шлях до файлу або URL);
  6) стан продукту (активний/неактивний).
- Деталі замовлення:
  1) ідентифікатор покупця;
  2) перелік товарів у замовленні;
  3) кількість товарів у замовленні;
  4) місце доставки;
  5) статус (нове, в обробці, відправлено, доставляється).
- Деталі для персоналізації магазинів:
  1) логотип (зображення);
  2) колірна гама;
  3) види;
  4) формати сторінок;
  5) основні стилі.
- Конфігурації системи та дані інтеграції:
  1) ключі API зовнішніх служб;
  2) специфікації підключення домену;
  3) налаштування доставки та оплати.

За допомогою REST API вхідні дані отримуються у вигляді об'єктів JSON. Схеми в MongoDB забезпечують типізацію, а на стороні сервера для виконання перевірки використовуються проміжне програмне забезпечення та відповідні бібліотеки.

1.5 Визначення структури вихідних даних

Усі відповіді, які надає вебплатформа після обробки запитів, називаються вихідними даними. Їхня структура визначає якість відображення інформації, правильність інтерфейсу та передбачуваність поведінки системи.

Основними категоріями вихідних даних є:

- Збереження інформації:
  1) ідентифікатор магазину;
  2) назву магазину;
  3) обрану колірну палітру та тему;
  4) активні налаштування та модулі;
  5) детальну інформацію про власника.
- Інформація про товар:
  1) вичерпна інформація про товар;
  2) перелік товарів за категоріями;
  3) відсортований або відфільтрований перелік товарів;
  4) інформація про наявність та вартість.
- Інформація про замовлення:
  1) список замовлень;
  2) деталі певного замовлення;
  3) історія змін та статуси замовлень.
- Інформація про користувача:
  1) роль користувача;
  2) повноваження;
  3) ім'я та контактну інформацію;
  4) історію взаємодій (для власників магазинів).
- Інформація про персоналізацію:
  1) специфікації теми;
  2) змінні CSS;
  3) макет сторінки;

1.6 База даних

MongoDB – це документоорієнтована база даних NoSQL, яка зберігає дані у форматі документів BSON, що є бінарним аналогом JSON. Це дозволяє гнучко керувати структурою даних, що особливо корисно при зберіганні налаштувань кастомізації, які можуть мати динамічну структуру.

Переваги MongoDB:

- Гнучка структура даних: дозволяє зберігати об'єкти з різними властивостями без попереднього визначення схеми, що підходить для збереження налаштувань кастомізації (кольори, шрифти, макети).
- Висока продуктивність та масштабованість: забезпечується горизонтальним шардінгом, що дозволяє обробляти великі обсяги запитів і даних.
- Простота інтеграції: MongoDB добре інтегрується з Node.js і TypeScript, що підходить для мікросервісної архітектури проєкту.
- Приклад використання: зберігання налаштувань кастомізації інтернет-магазинів, таких як стиль, макет, колірна схема, що може відрізнятися для кожного магазину.

Через свою гнучку структуру даних, легку інтеграцію з Node.js та масштабованість MongoDB ідеально підходить для зберігання налаштувань кастомізації. Це гарантує високу продуктивність платформи та дозволяє ефективно керувати динамічними даними.

Висновки до розділу 1

Перший розділ присвячений характеристиці об'єкта автоматизації, визначенню системних вимог та аналізу методів розробки вебплатформи для інтернет-магазинів. З метою забезпечення високої продуктивності та зручності використання було визначено, що платформа повинна пропонувати гнучкі налаштування дизайну, управління магазином та інтеграцію з ринком.

Використання мікросервісної архітектури для масштабованості та незалежності розгортання компонентів було підтримано аналізом технологічного підходу.

Проведено дослідження сучасного технологічного стеку MEAN, який спрощує підтримку та розробку платформи і дозволяє використовувати JavaScript/TypeScript на всіх рівнях розробки. База даних MongoDB забезпечує ефективне управління динамічними налаштуваннями магазину, а фронтенд побудований на Angular з окремими компонентами – для гарантування високої продуктивності. Запропоноване рішення поєднує масштабованість, гнучкість та ефективність, створюючи надійну основу для подальшої розробки платформи.

РОЗДІЛ 2 ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ ДЛЯ СТВОРЕННЯ ІНТЕРНЕТ-МАГАЗИНІВ

2.1 Створення логічної структури системи

Одним з найважливіших етапів є проектування логічної структури вебплатформи, оскільки саме на цьому етапі визначаються взаємодії основних компонентів, функції та потоки даних. Система повинна забезпечувати створення інтернет-магазинів, їх налаштування, управління замовленнями та товарами, а також інтеграцію з ринком.

Основними логічними модулями вебплатформи є:

- Модуль управління користувачами (User Management):
  1) аутентифікацію та реєстрацію користувачів;
  2) адміністрування ролей та прав доступу;
  3) зберігання даних профілю.
- Модуль створення та управління магазинами (Store Manager):
  1) створення магазину за допомогою шаблону;
  2) зміну налаштувань магазину;
  3) управління налаштуваннями та темами.
- Модуль управління товарами (Product Module):
  1) додавання, зміну та видалення товарів;
  2) створення категорій;
  3) пошук та сортування товарів.
- Модуль управління замовленнями (Order Processing):
  1) процедури створення замовлення;
  2) зміни статусу;
  3) повідомлення про нові замовлення;
  4) зберігання історії покупок.
- Модуль інтеграції з маркетплейсом:
  1) експорт товарів на ринок;

2) координації балансів;

3) виконання замовлень на ринковій платформі.

- Фронтенд-модуль (Angular Client):

1) дозволяє переглядати:

2) екран адміністратора;

3) інтерфейсу магазину;

4) налаштувань у режимі реального часу для персоналізації.

- API-модуль (Express + Node.js):

слугує посередником між користувачем і базою даних, обробляючи запити та надаючи відповіді.

- База даних (MongoDB):

1) таблиця Users;

2) таблиця Stores;

3) таблиця Products;

4) таблиця Orders;

5) таблиця Customizations;

6) таблиця MarketplaceExports.

2.2 Проєктування бази даних

На цьому етапі встановлюються взаємозв'язки та структура колекцій MongoDB. Структура є адаптивною і може бути налаштована відповідно до вимог кожного магазину завдяки архітектурі даних, орієнтованій на документи.

Розглянемо основні поля колекцій.

Колекція Users:

```
{
  _id: ObjectId,
  name: String,
  email: String,
```

passwordHash: String,
role: String,
createdAt: Date
}

Колекція Stores:

{
_id: ObjectId,
ownerId: ObjectId,
name: String,
theme: String,
colors: Object,
settings: Object,
createdAt: Date
}

Колекція Products:

{
_id: ObjectId,
storeId: ObjectId,
title: String,
price: Number,
description: String,
photos: [String],
category: String,
stock: Number,
isActive: Boolean
}

Колекція Orders:

{
_id: ObjectId,
storeId: ObjectId,

customer: Object,
products: [Object],
total: Number,
status: String,
createdAt: Date
}

Колекція Customizations:

{
_id: ObjectId,
storeId: ObjectId,
layout: Object,
colors: Object,
fonts: Object
}

2.3 Проєктування мікросервісної архітектури

Система може бути розділена на окремі частини завдяки концепції мікросервісів. Це спрощує масштабування, оновлення та управління платформою.

Проєкт містить в собі такі сервіси:

1. AuthService
   - Реєстрація
   - Логін
   - JWT-автентифікація
2. StoreService
   - Створення магазину
   - Редагування налаштувань
   - Обробка кастомізації
3. ProductService

- Управління товарами
- Категоризація
- Фото-менеджер

4. OrderService
   - Обробка замовлень
   - Оповіщення
   - Збереження статистики
5. MarketplaceService
   - Експорт товарів
   - Синхронізація залишків

2.4 Моделювання системи засобами UML

Під час створення вебплатформи було використано уніфіковану мову моделювання (UML), яка дозволяє описати структуру системи, логіку взаємодії між її компонентами та основні бізнес-операції. Діаграми UML спрощують процес переходу від теоретичної моделі до реалізації шляхом формалізації вимог.

Основні функції вебплатформи інтернет-магазину та взаємодію користувача з системою подано на рисунку 2.1.

Модель включає три суб'єкти:

1. Користувач – основний відвідувач платформи, який переглядає товари, додає їх до кошика, оформлює замовлення та здійснює оплату.
2. Адміністратор контролює вміст магазину та обробляє замовлення.
3. Платіжна система – зовнішня служба, що спрощує здійснення платежів.

Основні сценарії використання складаються з:

1. Створення акаунта та підтвердження email, необхідні для подальшої роботи з платформою (зв’язок include).

2. Авторизація для доступу до персоналізованих функцій.
3. Перегляд товарів, додавання до кошика, оформлення замовлення.
4. Оплата замовлення, що включає взаємодію з зовнішньою платіжною системою (include).
5. Перегляд статусу замовлення.

Адміністратор має окремий набір можливостей:

1. Авторизація адміністратора.
2. Додавання товарів, з можливістю опціонально додати зображення (extend).
3. Редагування товарів.
4. Керування замовленнями, що включає процедуру зміни їх статусів (include).

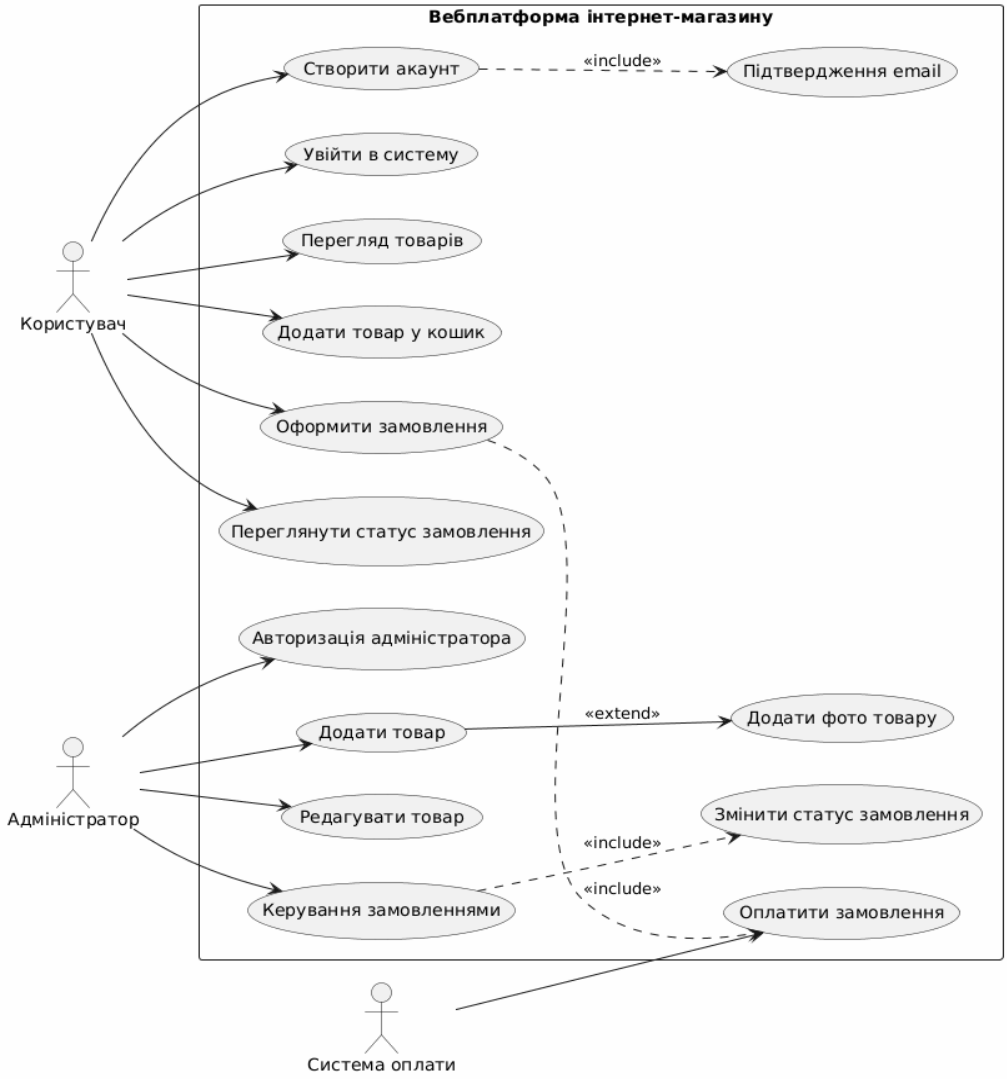

Рис. 2.1. Діаграма прецедентів

Структура даних та логічні зв'язки між основними об'єктами, що використовуються у вебплатформі на базі стеку MEAN, зображені на діаграмі класів (рисунок 2.2).

У моделі є п'ять основних класів:

1. User – містить інформацію про користувача: ім’я, email, пароль, роль та дату створення. Методи забезпечують реєстрацію, авторизацію та оновлення профілю.
2. Product – описує товар: назва, опис, ціна, зображення та належність до категорії. Підтримує додавання, редагування та видалення товарів.
3. Category – містить ідентифікатор та назву категорії. Використовується для групування товарів.
4. Order – відображає замовлення: список товарів, користувача, загальну ціну, статус та прив’язку до платіжного запису. Має методи для створення замовлення, обчислення загальної вартості та оновлення статусу.
5. Payment – містить інформацію про оплату: суму, статус, метод та посилання на пов’язане замовлення.

Зв’язки між класами:

1. Користувач – Замовлення: один користувач може мати багато замовлень (1 → N).
2. Замовлення – Платіж: кожне замовлення має один відповідний запис оплати (1 → 1).
3. Категорія – Товар: одна категорія може містити багато товарів (1 → N).
4. Замовлення – Товар: замовлення містить список товарів, що відображено як композиція.

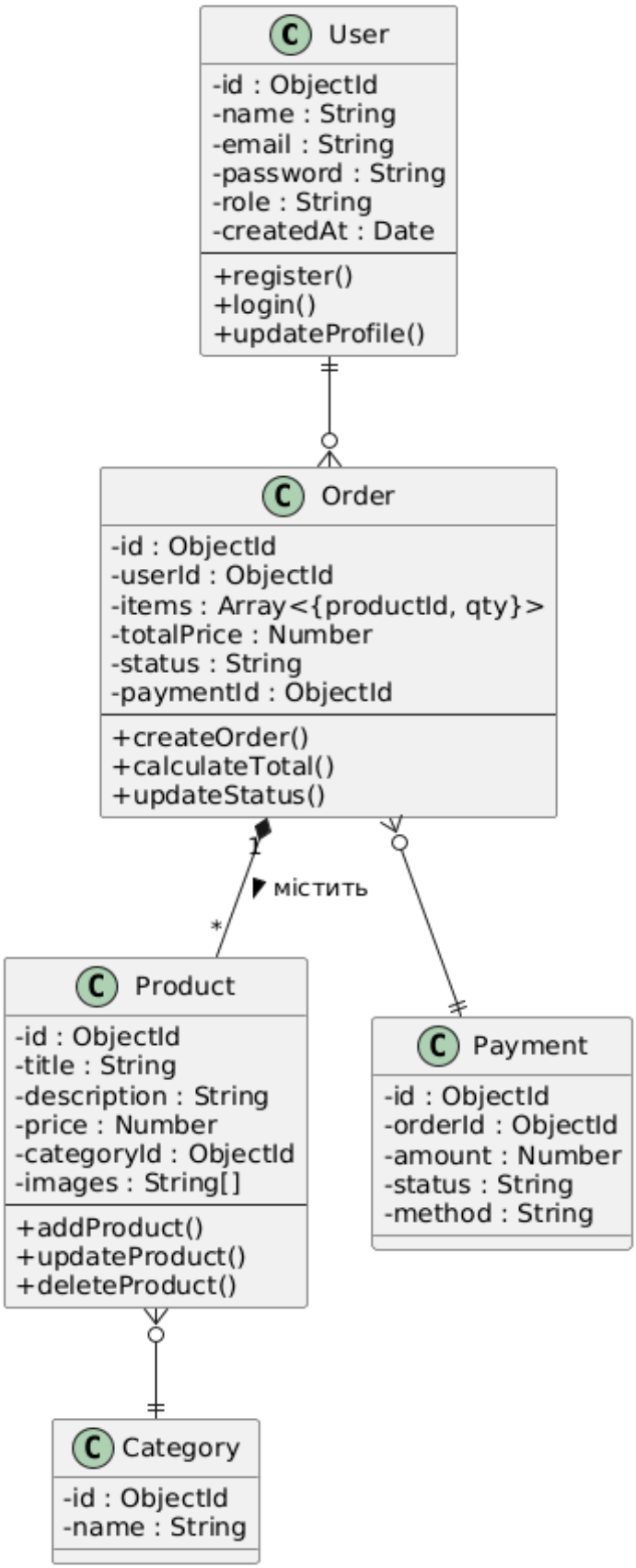

Рис. 2.2. Діаграми класів

Діаграма діяльності (рисунок 2.3) демонструє процес оформлення замовлення на платформі від першого кроку до завершення покупки.

Послідовність дій виглядає так:

1. Користувач відкриває сторінку товару і додає вибраний товар у кошик.
2. Переходить до кошика та натискає кнопку «Оформити замовлення».
3. Відбувається перевірка авторизації:

- якщо користувач не авторизований – система перенаправляє його на сторінку входу або реєстрації;
- якщо авторизований – процес триває.

4. Користувач вводить адресу доставки, обирає спосіб оплати.
5. Дані передаються у зовнішню платіжну систему.
6. Перевіряється результат оплати:

- у разі неуспіху – відображається повідомлення про помилку та пропонується повторний вибір способу оплати;
- при успіху – створюється запис у базі даних.

7. Система надсилає email-підтвердження користувачу.
8. Відображається фінальне повідомлення «Замовлення успішно оформлено».

Діаграма точно відображає повну бізнес-логіку оформлення покупки з усіма можливими переходами.

Рис. 2.3. Діаграма діяльності

Діаграма послідовності (рисунок 2.4) моделює взаємодію користувача, клієнтської частини (Angular), сервісів бекенду та бази даних у процесі оформлення замовлення.

Основні етапи:

1. Користувач на фронтенді натискає «Оформити замовлення».
2. Фронтенд передає список товарів і суму до OrderService.
3. OrderService створює попереднє замовлення та надсилає запит у MongoDB на вставку нового запису.
4. База даних повертає orderId.
5. OrderService звертається до PaymentService з параметрами для ініціалізації платежу.
6. PaymentService повертає посилання для оплати або статус транзакції.
7. Далі можливі два сценарії:

- Успішна оплата: OrderService оновлює статус замовлення в MongoDB → фронтенд показує повідомлення подяки.
- Помилка оплати: фронтенд отримує повідомлення про помилку і відображає його користувачу.

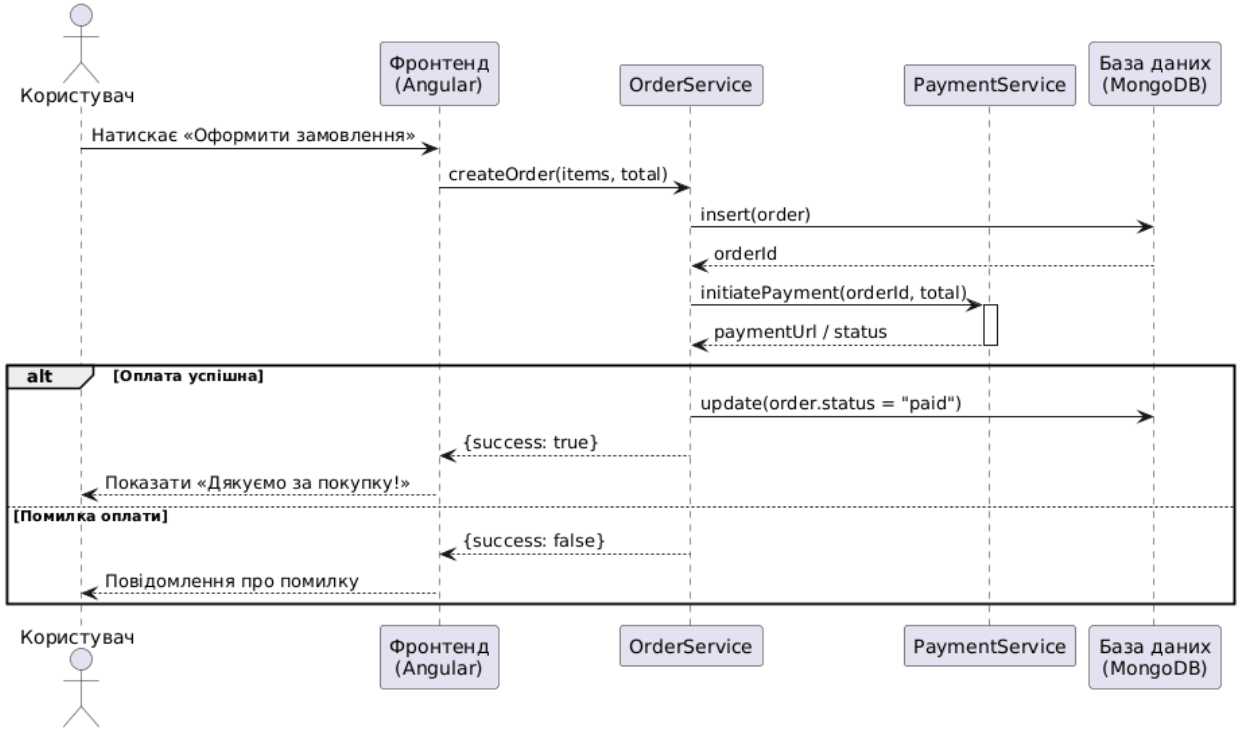

Рис. 2.4. Діаграма послідовності

2.5. Функціональні та нефункціональні вимоги

Функціональні вимоги визначають основні дії та сервіси, які платформа повинна надавати користувачам:

1. Управління магазинами:
   - Створення, редагування та видалення магазинів користувачем.
   - Можливість налаштування дизайну магазину (шаблони, кольори, логотип).
   - Управління товарами: додавання, редагування, видалення, категоризація, встановлення цін.
2. Користувачі та права доступу:
   - Реєстрація та авторизація користувачів.
   - Розмежування ролей: адміністратор платформи, власник магазину, покупець.
   - Контроль доступу до ресурсів відповідно до ролей.
3. Замовлення та оплата:
   - Прийом замовлень, облік статусу замовлення (нове, обробляється, доставлено).
   - Інтеграція з платіжними системами (LiqPay, Stripe, PayPal).
   - Генерація електронних чеків та історії замовлень.
4. Аналітика та звітність:
   - Відстеження продажів та відвідуваності магазину.
   - Формування звітів по товарах, замовленнях та користувачах.
5. Інтеграція з маркетплейсами:
   - Експорт та синхронізація товарів і замовлень.
   - Підтримка стандартних REST API для зовнішніх сервісів.

Нефункціональні вимоги визначають характеристики системи, що забезпечують її якість:

1. Продуктивність:

- Система повинна обробляти одночасно до 5000 активних користувачів без зниження швидкості відповіді.

2. Масштабованість:

- Архітектура повинна дозволяти легке додавання нових мікросервісів та горизонтальне масштабування серверів.

3. Безпека:

- Використання HTTPS для всіх з’єднань.
- Зберігання паролів у хешованому вигляді (bcrypt).
- Захист від NoSQL-ін’єкцій та XSS-атак.

4. Надійність та відновлення:

- Система повинна зберігати дані у резервних копіях.
- Відновлення роботи після збоїв упродовж 5 хвилин.

5. Юзабіліті:

- Інтуїтивно зрозумілий інтерфейс для користувачів різного рівня.
- Мобільна адаптивність та підтримка популярних браузерів.

6. Підтримка та розширюваність:

- Можливість легко додавати нові функції без зміни основної архітектури.
- Документовані API та інструкції для інтеграцій.

Висновки до розділу 2

Логічна структура системи, моделювання даних, проєктування архітектури мікросервісів, а також функціональні та нефункціональні вимоги викладені в розділі 2. Створено основні модулі платформи – управління користувачами, магазинами, продуктами та замовленнями, інтеграція з ринком, інтерфейс та API. Перехід від проєктування до реалізації спрощується

завдяки формалізації бізнес-процесів та взаємодії компонентів за допомогою UML-діаграм випадків використання, класів, дій та послідовностей.

Архітектура мікросервісів гарантує ізольоване виконання функцій, масштабованість та простоту обслуговування, а база даних на базі MongoDB була розроблена для забезпечення гнучкої обробки та зберігання даних користувачів, магазинів та продуктів.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ВЕБПЛАТФОРМИ ДЛЯ ІНТЕРНЕТ-МАГАЗИНІВ

3.1 Загальні принципи реалізації

Створена раніше архітектура та структура системи, описані в другому розділі, слугують основою для реалізації вебплатформи. Для збереження масштабованості, гнучкості та придатності системи для майбутнього зростання, на етапі реалізації програмного забезпечення надзвичайно важливо підтримувати узгодженість між логічною моделлю, вимогами та реальними програмними компонентами.

Основні функції вебплатформи включають інтеграцію з ринком та підтримку численних інтернет-магазинів з можливістю налаштування. Це вимагає відмінної продуктивності, однозначної комунікації між службами та підтримки одночасної роботи декількох користувачів.

Під час впровадження було дотримано таких рекомендацій:

1. Модульність і розділення відповідальностей.
   Кожен сервіс виконує вузько визначений набір функцій: аутентифікація, управління товарами, створення замовлень, оплата тощо. Це спрощує тестування та розширення функціоналу.
2. Використання REST API.
   Комунікація між frontend та backend здійснюється шляхом HTTP-запитів, що забезпечує простоту інтеграції та можливість підключення мобільних застосунків чи зовнішніх сервісів.
3. Асинхронна обробка даних.
   Завдяки Node.js система здатна обробляти велику кількість одночасних запитів без втрати продуктивності.
4. Зберігання даних у документній базі MongoDB.
   Гнучка структура документів дозволяє зберігати складні об’єкти (товари, замовлення, списки зображень) без надмірної нормалізації.

5. Безпека даних.

   Для роботи з особистою інформацією та оплатами застосовано JWT, валідацію даних, обмеження доступу та захист від основних типів атак (XSS, SQL/NoSQL injection, CSRF).

3.2 Технологічна реалізація системи

Вебплатформа була реалізована з використанням стеку MEAN (MongoDB, Express, Angular і Node.js). Кожна частина вносить свій унікальний вклад у загальну архітектуру і забезпечує найкращий баланс між масштабованістю, продуктивністю та простотою обслуговування (рисунок 3.1).

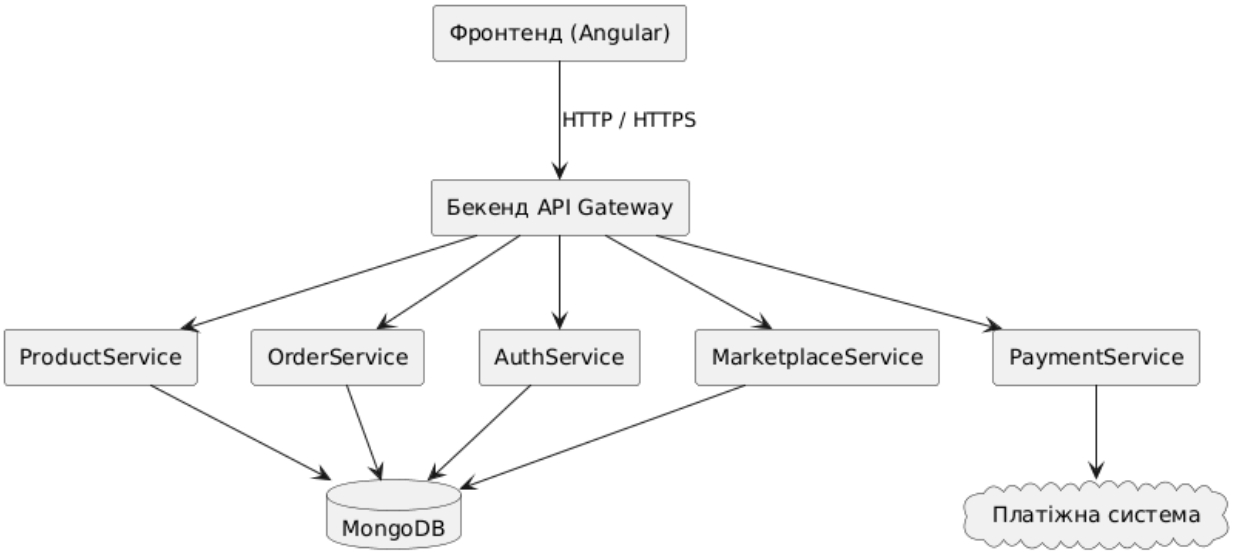

Рис. 3.1. Технології для створення користувацького інтерфейсу.

Frontend – Angular – основна технологія, яка використовується для створення користувацького інтерфейсуr. Завдяки її модульній конструкції та підтримці TypeScript можна реалізувати складні інтерфейси, такі як:

- авторизацію;
- перегляд каталогу;
- управління товарами (для адміністратора);

- оформлення замовлень;
- роботу з кошиком у режимі реального часу.

Фреймворк забезпечує реактивність, чіткий поділ компонентів та інструменти для оптимізації швидкодії.

Backend – Node.js + Express

Мікросервіси, побудовані на Express, складають бек-енд. Основні маршрути – AuthService, ProductService, OrderService, PaymentService та MarketplaceService - розділені на основі логіки сервісу.

Express забезпечує:

- обробку REST-запитів;
- middleware-архітектуру;
- гнучку маршрутизацію;
- можливість додавання власної бізнес-логіки без надмірної складності.

База даних – MongoDB використовується як основне сховище:

- товари;
- замовлення;
- користувачі;
- категорії;
- платежі.

Документна модель дозволяє швидко змінювати структуру даних без необхідності переробки всієї бази.

Також використовуються зовнішні сервіси та інструменти:

- JWT – для авторизації та управління сесіями.
- Multer / Cloudinary – для завантаження та зберігання зображень товарів.
- LiqPay / Stripe API – для інтеграції оплати.
- PM2 – для керування процесами на сервері.
- Docker – для контейнеризації сервісів.
- Postman – для тестування API.

Проєкт структуровано таким чином, щоб кожен модуль був незалежним. Це спрощує масштабування і дозволяє окремо розгортати сервіси.

Поверхнева структура backend:

/auth-service

- /controllers
- /routes
- /models
- /middlewares

/product-service

- /controllers
- /routes
- /models

/order-service

- /controllers
- /routes
- /models

/payment-service

- /controllers
- /routes
- /models

Поверхнева структура frontend:

/src

- /app
  - /components
  - /services
  - /pages
  - /guards
  - /models

Такий поділ дозволяє чітко розмежувати відповідальність, забезпечити чисту архітектуру та спростити майбутнє розширення.

3.3 Реалізація функціональних модулів

При реалізації кожної служби дотримувалась концепція модульності та розподілу обов'язків. Кожна служба є окремим логічним блоком, який обробляє частину бізнес-логіки та надає доступ до відповідних даних через REST API. Цей метод дозволяє додавати нові функції, розширювати систему та оптимізувати окремі модулі без необхідності повного перепроектування платформи.

Нижче детально описано основні функціональні компоненти вебплатформи, які забезпечують її працездатність.

AuthService відповідає за реєстрацію, авторизацію, перевірку прав доступу та управління користувацькими токенами. Реалізація передбачає використання JWT як основного механізму підтвердження особи користувача (рисунок 3.3).

На рисунку 3.2 представлено головну сторінку адмін. панелі.

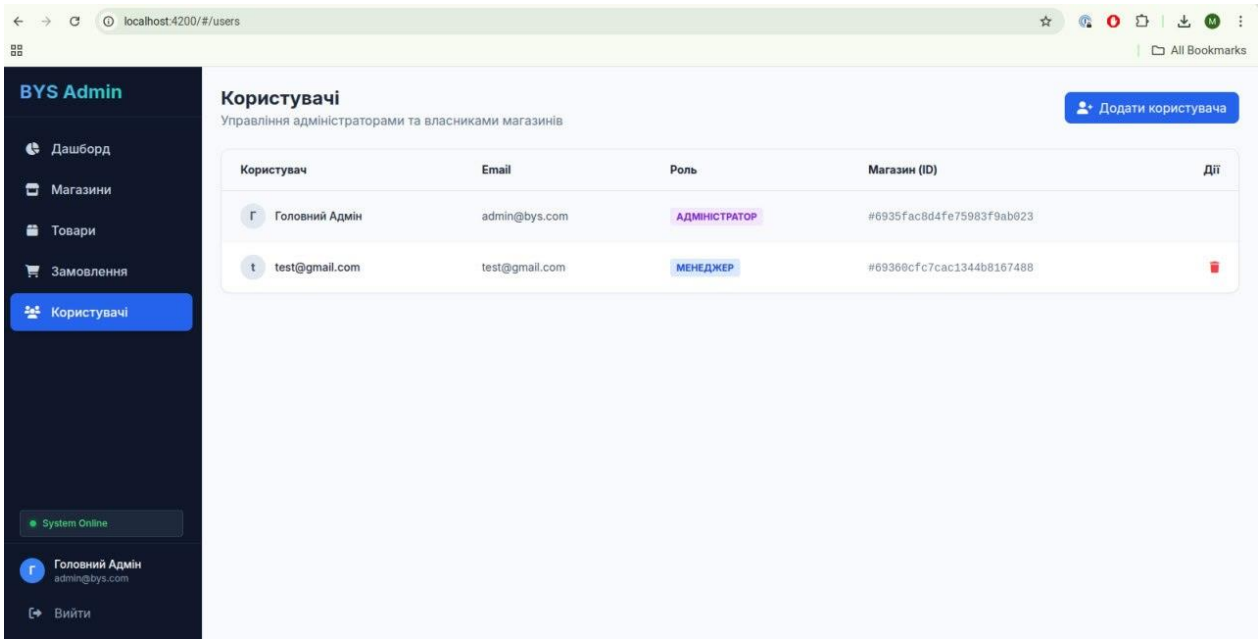

Рис. 3.2. Управління користувачами та правами доступу

Рис. 3.3. Алгоритм роботи AuthService

Основні функції AuthService:

1. Реєстрація нового користувача.

   Дані проходять валідацію, пароль хешується, email перевіряється на унікальність.

2. Авторизація.

   Перевіряються email та пароль, після чого генерується JWT-токен.

3. Перевірка доступу.

   Middleware аналізує токен у заголовку Authorization та визначає роль користувача.

4. Оновлення користувацьких даних.

   Користувач може змінювати особисту інформацію, пароль, аватар.

Структура бізнес-логіки AuthService:

- auth.controller.js (обробка запитів)
- auth.routes.js (маршрутизація)
- auth.model.js (схема користувача)
- auth.middleware.js (перевірка JWT)

Розгянемо ProductService – модуль управління товарами (рисунок 3.4).

Рис. 3.4. Алгоритм ProductService

ProductService відповідає за створення, редагування, видалення та отримання товарів, а також роботу з їх зображеннями (рисунок 3.5). Це один із центральних модулів платформи.

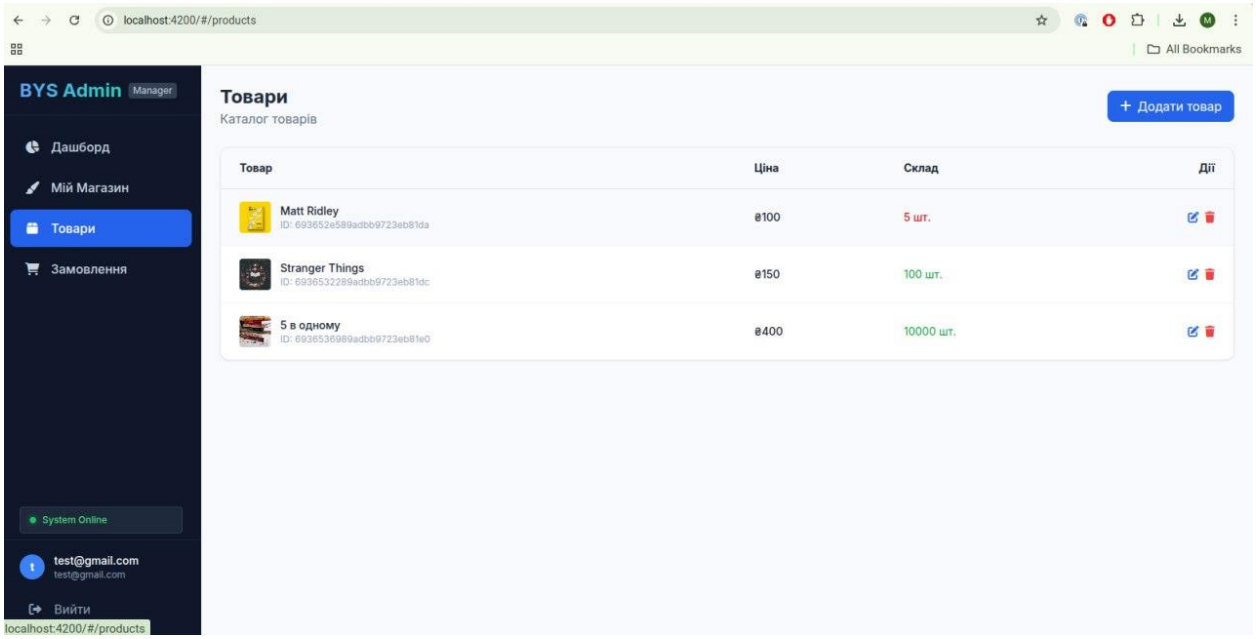

Рис. 3.5. Сторінка управління товарами

Функціонал модуля (рис. 3.4):

1. Створення товару.

   Обробка заголовка, опису, ціни, категорії та зображень.

2. Редагування товару.

   Часткове або повне оновлення полів товару.

3. Видалення товару.

   Видаляються також відповідні зображення.

4. Отримання товарів.

   Можливість фільтрації за категоріями, пошуку за ключовими словами.

5. Завантаження зображень.

   Використання Multer та інтеграції з хмарним CDN для оптимізації.

Структура модуля:

- product.controller.js (CRUD-операції
- product.routes.js (маршрути)
- product.model.js (схема товару)
- upload.middleware.js (обробка файлів)

OrderService відповідає за формування замовлень, перевірку даних, підрахунок вартості та оновлення статусів замовлень. Це ключова частина, що пов’язує роботу каталогу, користувачів і платіжного сервісу.

Основні можливості:

1. Створення замовлення.

   Система перевіряє наявність товарів, підраховує загальну суму, формує структуру замовлення.

2. Зміна статусів.

   Адміністратор може оновлювати статус:

   нове → в обробці → відправлено → доставлено.

3. Отримання історії замовлень користувача.
4. Інтеграція з PaymentService.

   Для оплати потрібно передати orderId, суму, спосіб оплати.

Структура модуля:

- order.controller.js
- order.routes.js
- order.model.js

ShopModule, зображений на рисунках 3.6 – 3.7, відповідає за повний цикл роботи з інтернет-магазинами на платформі: створення, редагування, налаштування дизайну, керування доступами та інтеграцію магазину з маркетплейсом. Модуль забезпечує гнучкість, що дозволяє власникам створювати власний бренд усередині платформи та керувати його роботою без участі адміністратора системи.

Основні можливості:

1. Створення магазину

   Користувач із відповідними правами може створити магазин, вказавши базові дані: назву, опис, контактну інформацію.

   Під час створення формується унікальний shopId та ініціалізується структура магазину.

2. Редагування інформації про магазин

Доступні зміни назви, опису, банера, логотипу, теми оформлення та інших атрибутів, що визначають зовнішній вигляд магазину.

3. Керування параметрами магазину
   - налаштування категорій і товарних блоків;
   - вибір валюти;
   - активація/деактивація магазину;
   - встановлення правил доставки й оплати.
4. Управління користувачами магазину

   Власник може додавати менеджерів, призначати ролі та контролювати доступ до функціональних частин магазину.
5. Інтеграція з MarketplaceService

   Магазин може бути опублікований у маркетплейсі; для цього модуль передає інформацію про магазин, його статус та базові метадані.
6. Отримання списку магазинів та їх деталей

   Адміністратор або власник можуть переглядати всі магазини чи конкретний shopId для подальших дій.

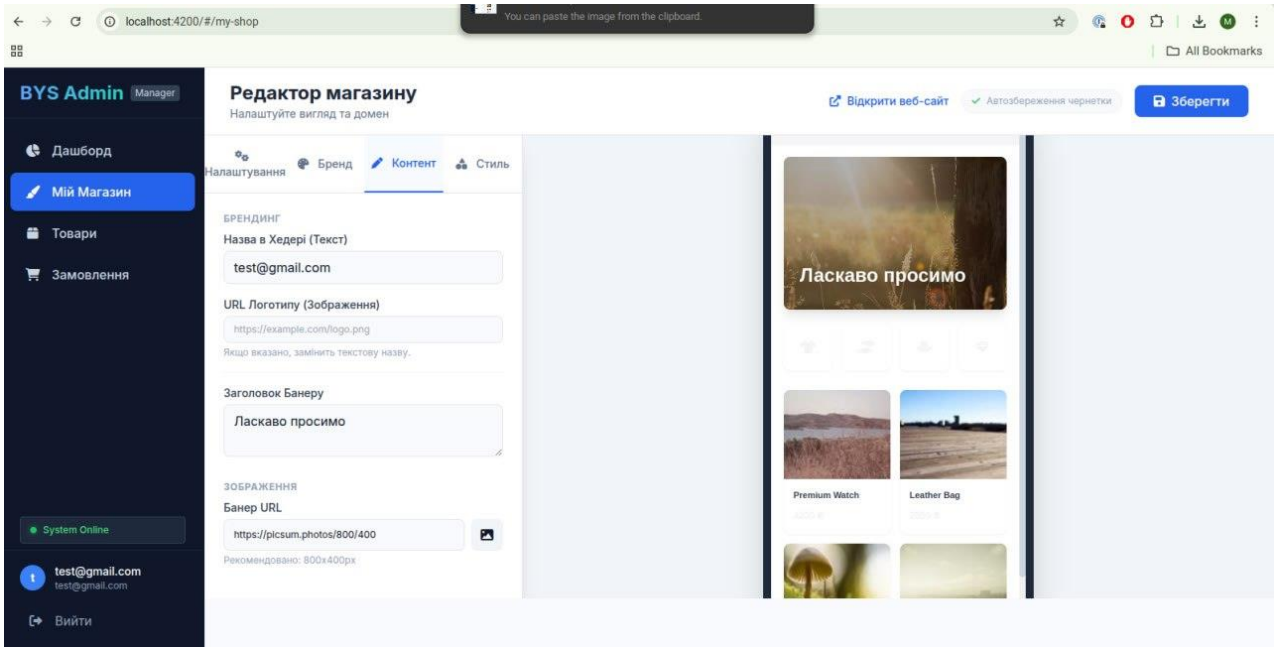

Рис. 3.6. Сторінка створення/редагування магазину

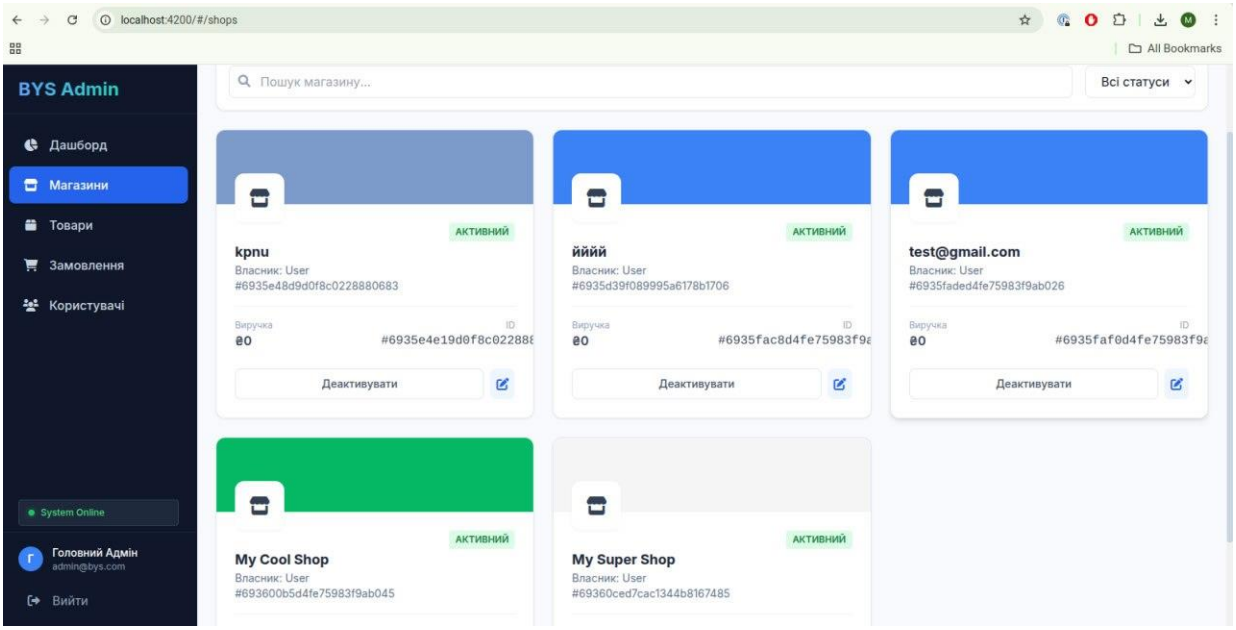

Рис. 3.7. Сторінка перегляду магазинів від адміністратора

Структура модуля:

- shop.controller.js – логіка створення, оновлення, налаштування та отримання магазинів.
- shop.routes.js – REST-маршрути (GET, POST, PUT, DELETE) для роботи з магазинами.
- shop.model.js – схема магазину в MongoDB (назва, опис, логотип, дизайн, статус, власник, список менеджерів тощо).

Висновки до розділу 3

У розділі 3 розглянуто практичне використання вебплатформи, побудованої на стеку MEAN і архітектурі мікросервісів для створення та управління інтернет-магазинами. Масштабована і корисна платформа електронної комерції, що сприяє модульності, повторному використанню компонентів і чіткому розподілу обов'язків між сервісами, стала можливою завдяки впровадженню ключових компонентів системи. Завдяки синхронізації даних та управлінню замовленнями і магазинами, розроблена система забезпечує стабільну взаємодію між фронтендом і бекендом, а також

легко розширюється, оновлюється та адаптується до індивідуальних потреб користувачів.

Висока продуктивність, стабільність та відсутність критичних збоїв платформи були підтверджені модульним, інтеграційним, комплексним та навантажувальним тестуванням. Надійність, масштабованість та гнучкість поєднані в реалізованому рішенні, що створює основу для майбутнього розвитку та впровадження нових функцій.

ВИСНОВКИ

Під час роботи над темою кваліфікаційного дослідження проведено ретельний аналіз сучасних методів створення вебплатформ для електронної комерції. Описано об'єкт автоматизації, здійснено порівняння сучасних технологічних рішень та аналіз системних вимог. Концепція нашого власного програмного рішення була сформована на основі виявлення важливих тенденцій в онлайн-бізнесі та електронній комерції, а також проблем, пов'язаних із масштабованістю, налаштуванням та інтеграцією різних сервісів в рамках єдиної платформи.

Створена платформа поєднує мікросервісну архітектуру із стеком MEAN (MongoDB, Express.js, Angular, Node.js), щоб гарантувати масштабованість, адаптивність і модульність системи. Поряд із підключенням до торгових майданчиків та платіжних систем, впроваджено панель адміністратора та функціональні модулі для управління користувачами, товарами, замовленнями та магазинами. Така архітектура спрощує додавання нових функцій та адаптацію системи до різних сценаріїв використання. Вона також дозволяє розробляти індивідуальні магазини, ефективно обробляти замовлення та забезпечувати стандартизовану комунікацію між компонентами через REST API.

Особлива увага приділялася стабільності та надійності платформи. Здатність системи обробляти кілька запитів одночасно без зниження продуктивності або катастрофічних збоїв була перевірена за допомогою модульних, інтеграційних, ретельних та навантажувальних тестів. Ефективна синхронізація даних, управління бізнес-процесами та персоналізація магазину на основі вимог користувачів стали можливими завдяки вбудованій структурі бази даних на основі MongoDB та встановленню необхідних служб (AuthService, ProductService, OrderService, PaymentService, MarketplaceService).

Основною новизною роботи є створення кастомізованої платформи для одночасного управління багатьма інтернет-магазинами, що поєднує гнучкість налаштувань, масштабованість і високий рівень інтеграції з зовнішніми сервісами. Запропоноване рішення відкриває можливості для подальшого розвитку електронної комерції, впровадження індивідуальних бізнес-моделей та оптимізації процесів управління онлайн-бізнесом. Крім того, реалізовані підходи до модульності, мікросервісної архітектури та синхронізації даних можуть бути використані як методологічна база для подальших досліджень у сфері розробки вебплатформ і цифрової трансформації бізнесу.

Таким чином, виконана робота забезпечує цілісний підхід до створення сучасної, гнучкої та ефективної системи для електронної комерції, демонструючи як теоретичну обґрунтованість, так і практичну реалізацію, що відповідає сучасним вимогам ринку та технологічним стандартам, а також відкриває перспективи для подальшого вдосконалення та масштабування платформи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мозолюк М. Ефективність індивідуалізації інтернет-магазинів для підвищення конверсії ти залучення клієнтів. Вісник Кам'янець-Подільського національного університету імені Івана Огієнка. Фізико-математичні науки. Кам'янець-Подільський : К-ПНУ ім. І. Огієнка, 2024. № 17. С. 100-105. URL: https://fizmat.kpnu.edu.ua/wp-content/uploads/2024/12/visnyk-17-2024-17-12-2024.pdf

2. Мозолюк М. White label-рішення в електронній комерції: технології та перспективи розвитку. *Збірник матеріалів наукової конференції за підсумками науково-дослідної роботи здобувачів вищої освіти фізико-математичного факультету Кам'янець-Подільського національного університету імені Івана Огієнка у 2024-2025 н.р.*, 9-10 квітня 2025 року. Кам'янець-Подільський : К-ПНУ ім. І. Огієнка, 2025. С. 52-55. URL: https://fizmat.kpnu.edu.ua/wp-content/uploads/2025/05/zbirnyk-stud-i-mahistrpravky-27-05-2025.docx.pdf

3. Mastering Express.js: The Complete Guide for MEAN Stack Developers (2025 Edition). MeanStackGuides. URL: https://www.meanstackguides.com/2025/11/mastering-expressjs-complete-guide-for.html?

4. MEAN Stack in 2025: Modern Approaches and Best Practices. Medium. URL: https://medium.com/%40vipulagg2015/mean-stack-in-2025-modern-approaches-and-best-practices-e44a5adf5a30?

5. Latest Trends in MEAN Stack Development for 2024. MoldStud. URL: https://moldstud.com/articles/p-whats-new-in-mean-stack-development-2024-trends-and-technologies?

6. Top Tools to Use for Full Stack Developers in 2025. OSIZ Labs. URL: https://www.osizlabs.com/news/top-tools-to-use-for-full-stack-developers-in-2025?

7. Курс “Node.js + Express + MongoDB: побудова REST API” (HILLEL MAX, 2025). URL: https://max.ithillel.ua/courses/node-js?

8. “Найкращі фреймворки веб-розробки на 2025 рік” — огляд Dizz Agency. URL: https://dizz.in.ua/uk/najkrashhi-frejmvorki-veb-rozrobki-na-2025-rik

9. “Що повинен знати Node.js розробник у 2025-Q1” — форум DOU. URL: https://dou.ua/forums/topic/52865

10. “Search-Based Fuzzing For RESTful APIs That Use MongoDB” (2025). URL: https://arxiv.org/abs/2507.20848

11. “Towards a Standard for JSON Document Databases” (2025) — дослідження формалізації JSON-документних БД. URL: https://arxiv.org/abs/2509.12189

12. “GraphQLer: Enhancing GraphQL Security with Context-Aware API Testing” (2025) — стаття про безпеку API. URL: https://arxiv.org/abs/2504.13358?

13. “Serverless architecture + MEAN Stack: сучасні тренди 2025” — огляд SquarespaceBlog. URL: https://squarespaceblog.com/mean-stack-development-company-trends-to-watch-in-2025/?

14. “Express.js still лідер: огляд 2025” — Reddit / r/node дискусія про актуальність Express.js. URL: https://www.reddit.com//r/node/comments/1jd1e8c?

15. Огляд реальних проєктів на MERN/MEAN-стеках 2025 — Reddit / r/webdevelopment. URL: https://www.reddit.com/r/javascript/comments/1dbbszk?

16. Обговорення популярності стеків MERN/MEAN та Node.js на ринку праці 2025 — DOU форум. URL: https://dou.ua/forums/topic/52865/?

17. Angular v18 Release Overview (2025).

18. Google Angular Team. URL: https://angular.dev/updates

19. State of JavaScript 2024–2025 Report — офіційний звіт про тренди у веб-розробці. URL: https://stateofjs.com

20. Node.js 22 Release Notes (2025) — офіційні оновлення ядра. URL: https://nodejs.org/en/blog

21. MongoDB 8.0 — What’s New (2025) — нові можливості кластерів та індексів. URL: https://www.mongodb.com/press

22. The Future of REST APIs in 2025 — Best Practices & Trends.Nordic APIs. URL: https://nordicapis.com

23. Microservices in 2025: Architecture, Orchestration, Security. InfoQ. URL: https://www.infoq.com/microservices

24. Testing Modern JavaScript Applications in 2025.freeCodeCamp. URL: https://www.freecodecamp.org/news

25. Angular Standalone Components & Signals: Deep Dive (2025) — технічний огляд. URL: https://blog.angular.io

26. Express.js Performance Optimization Techniques (2025).Medium. URL: https://medium.com

27. Serverless Backends with Node.js in 2025 — Architecture Guide. AWS Blog. URL: https://aws.amazon.com/blogs

28. Reactive Programming in Angular 18 Using RxJS 8 (2025). Angular Architects. URL: https://www.angulararchitects.io

29. Top 10 Trends in Full Stack Development 2025. JetBrains Survey. URL: https://www.jetbrains.com/lp/devecosystem-2024

30. Building Scalable Marketplace Backends with Node.js (2025). DigitalOcean Tutorials. URL: https://www.digitalocean.com/community

31. API Load Testing with k6: Best Practices 2025 Edition. Grafana k6 Docs. URL: https://k6.io/docs

32. CI/CD for MEAN Stack Applications in 2025 — GitHub Actions Pipeline Guide. GitHub Engineering Blog. URL: https://github.blog

33. Modern Microservices Architecture Patterns (2025). URL: https://www.infoq.com/articles/modern-microservices-architecture-2025

34. Scalable Web Architecture 2025: Best Practices — архітектурні рішення для високонавантажених платформ. URL: https://www.nginx.com/blog/scalable-web-architecture-best-practices-2025/

35. Designing Cloud-Native Applications with Node.js & Angular (2025) — принципи побудови cloud-native рішень, кейси. URL: https://cloud.google.com/architecture

36. Event-Driven Architecture in Modern Web Apps (2025). URL: https://martinfowler.com/articles/2025-event-driven-architecture.html

37. The State of API Architecture 2025 — REST, GraphQL, Grpc. URL: https://nordicapis.com/state-of-api-architecture-2025