**Міністерство освіти і науки України**
**Луцький національний технічний університет**
**Факультет комп'ютерних та інформаційних технологій**
**Кафедра інженерії програмного забезпечення**

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

# РОЗРОБКА ТА ДОСЛІДЖЕННЯ ІНТЕРНЕТ-МАГАЗИНУ З ІНТЕГРАЦІЄЮ ПЛАТІЖНИХ СИСТЕМ MONOBANK ТА WAYFORPAY ЗА ДОПОМОГОЮ ФРЕЙМВОРКУ LARAVEL

# DEVELOPMENT AND RESEARCH OF AN ONLINE STORE WITH THE INTEGRATION OF MONOBANK AND WAYFORPAY PAYMENT SYSTEMS USING THE LARAVEL FRAMEWORK

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗм-21
Киричук О. О.
Керівник:
к.т.н., доцент
Суринович О. М.

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент Суринович О. М.
_____

Луцьк – 2025 року

**ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ**

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *магістр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«___» _____ 202__ р.

**ЗАВДАННЯ**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА**
**ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ**
**Киричуку Олександру Олександровичу**

1. Тема кваліфікаційної роботи: *Розробка та дослідження інтернет-магазину з інтеграцією платіжних систем Monobank та WayForPay за допомогою фреймворку Laravel*
Керівник роботи: *Суринович Олена Миколаївна, доцент, к.т.н.*
_____
затверджені наказом закладу вищої освіти від «29» березня 2025 року № 190/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи: 4 грудня 2025 р.
3. Вихідні дані до роботи *технічне та програмне забезпечення ЕОМ*
4. Зміст розрахунково-пояснювальної записки: *Аналіз проблематики розробки інтернет-магазинів та вимог до сучасних платіжних сервісів, обґрунтування вибору технологій і фреймворку Laravel, проєктування та реалізація вебсистеми з інтеграцією платіжних систем Monobank і WayForPay, експериментальне дослідження стабільності, безпеки та результативності розробленого програмного забезпечення.*
5. Перелік графічного матеріалу 25 рисунків, 6 таблиць, 4 лістинги коду

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| *Аналіз проблеми за темою роботи та постановка завдань дослідження* | *Суринович О. М.* | | |
| *Теоретичне дослідження та практична реалізація* | *Суринович О. М.* | | |
| *Експериментальне дослідження системи* | *Суринович О. М.* | | |
| *Нормоконтроль* | *Повстяна Ю. С.* | | |
| *Гарант ОП* | *Андрущак І. Є.* | | |
| *Показник запозичень тексту* | ___% | | |
| *Академічна доброчесність* | *Суринович О. М.* | | |

7. Дата видачі завдання «02» квітня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи магістра | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Провести огляд літературних джерел по темі кваліфікаційної роботи | 02.05.2025 | |
| 2 | Провести аналіз загальної проблеми і вибір напрямків дослідження | 24.09.2025 | |
| 3 | Розробити функціональну модель та архітектуру системи | 01.11.2025 | |
| 4 | Описати засоби розробки об'єкта проектування | 19.11.2025 | |
| 5 | Практична реалізація об'єкта проектування | 26.11.2025 | |
| 6 | Розробити методику для проведення експерименту | 05.11.2025 | |
| 7 | Провести аналіз результатів експерименту | 15.11.2025 | |
| 8 | Здача чистового варіанту кваліфікаційної роботи на кафедру | 04.12.2025 | |

Здобувач вищої освіти _____ Киричук О. О.

Керівник кваліфікаційної роботи _____ Суринович О. М.

# АНОТАЦІЯ

Киричук О. О. Розробка та дослідження інтернет-магазину з інтеграцією платіжних систем Monobank та WayForPay за допомогою фреймворку Laravel. Рукопис.

Кваліфікаційна робота ОП «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається зі вступу, трьох розділів, висновків та списку використаних джерел.

У першому розділі проведено аналіз електронної комерції, сучасних технологій веброзробки та платіжних систем, обґрунтовано вибір стеку: Laravel 12, Nuxt 3 і MySQL. Визначено мету та завдання роботи: розробка бекенду, адміністративної панелі, клієнтського інтерфейсу, інтеграція платіжних систем і створення REST API. У другому розділі обґрунтовано вибір архітектури та технологій, показано практичну реалізацію вебсайту з інтегрованими Monobank і WayForPay. Наведено скриншоти ключових сторінок: головна, каталог, сторінка товару. У третьому розділі описано методику тестування, аналіз результатів інтеграції та роботи системи, оцінено стабільність і коректність функціоналу адміністративної панелі та користувацького інтерфейсу.

Розробка демонструє можливості безпечного і стабільного інтернет-магазину з адаптивним інтерфейсом та інтегрованими платіжними сервісами.

Ключові слова: Laravel, Nuxt 3, REST API, Monobank, WayForPay, інтеграція платежів, інтернет-магазин, Docker, Laradock.

# ABSTRACT

Kyrychuk O. O. Development and Research of an Online Store With the Integration of Monobank and WayForPay Payment Systems Using the Laravel Framework. Manuscript.

Qualification work of the educational program «Software Engineering». Lutsk National Technical University. Lutsk, 2025.

The master's qualification work consists of an introduction, three chapters, conclusions, and a list of references.

The first chapter analyzes e-commerce, modern web development technologies, and payment systems, and substantiates the choice of the technology stack: Laravel 12, Nuxt 3, and MySQL. The goal and tasks of the work are defined: backend development, creation of an administrative panel, client interface, payment system integration, and REST API implementation. The second chapter substantiates the choice of architecture and technologies and demonstrates the practical implementation of the website with integrated Monobank and WayForPay. Screenshots of the key pages are provided: homepage, catalog, and product page. The third chapter describes the testing methodology, analysis of integration results and system operation, and evaluation of the stability and correctness of the functionality of the administrative panel and user interface.

The development demonstrates the capabilities of a secure and stable online store with an adaptive interface and integrated payment services.



Keywords: Laravel, Nuxt 3, REST API, Monobank, WayForPay, Payment integration, Online store, Docker, Laradock.

# ЗМІСТ

# ВСТУП

Актуальність теми. Стрімкий розвиток електронної комерції в Україні та світі формує зростаючий попит на безпечні, стійкі та зручні онлайн-платформи для продажу товарів. Користувачі очікують швидку навігацію, простий інтерфейс, коректну обробку замовлень та можливість миттєво здійснювати онлайн-оплати. Одним з ключових факторів комерційного успіху інтернет-магазину є якісна інтеграція платіжних систем, що забезпечує безпеку транзакцій, надійність та стабільність роботи сервісу. В умовах розвитку українського фінтеху та популярності платіжних платформ Monobank і WayForPay особливо важливим є дослідження особливостей їх інтеграції у вебзастосунки на базі сучасних фреймворків.

Мета роботи полягає у розробці інтернет-магазину з інтеграцією платіжних систем Monobank та WayForPay на основі Laravel і Nuxt 3, а також дослідженні особливостей їх роботи в реальних умовах експлуатації.

Завдання дослідження:

— здійснити аналіз вимог до сучасних інтернет-магазинів, включно з функціональністю, безпекою та підтримкою онлайн-оплат;

— розробити серверну частину вебсистеми на основі фреймворку Laravel та налаштувати базову архітектуру проєкту;

— створити адміністративну панель для управління товарами, категоріями, замовленнями та користувачами;

— розробити клієнтську частину інтернет-магазину з використанням Nuxt 3 та забезпечити її адаптивність для різних пристроїв;

— інтегрувати платіжні системи Monobank та WayForPay за допомогою API та механізму Webhook;

— реалізувати повноцінний REST API для взаємодії клієнтської частини з бекендом;

— виконати підготовку технічної документації щодо структури, реалізації та експлуатації розробленої системи.

Об'єкт дослідження – процес створення та функціонування інтернет-магазину з інтегрованими платіжними системами.

Предмет дослідження – методи, алгоритми та технології розробки вебсистеми з безпечними онлайн-платежами, REST API та адаптивним інтерфейсом користувача.

Новизна роботи полягає у комплексному впровадженні інтернет-магазину з повноцінним REST API, клієнтською частиною на Nuxt 3, серверною логікою на Laravel 12 та інтеграцією сучасних українських платіжних сервісів Monobank і WayForPay через API та Webhook. Це забезпечує надійність обробки транзакцій, зручність адміністрування та високу продуктивність системи.

Практична цінність роботи полягає у створенні готового до використання вебзастосунку, який може бути розгорнутий у виробничих умовах та адаптований під реальні бізнес-потреби. Розроблена система може бути розширена модулем аналітики, інтеграціями з зовнішніми сервісами та додатковими платіжними інструментами.

Апробація результатів дослідження. Киричук О. О., Суринович О. М. Електронні оплати у системі сучасної економіки. Тези доповідей X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві» (ІТОНВ-2025) (23-24 травня 2025 року). Луцьк: ЛНТУ, 2025. С. 336-341 [1].

# РОЗДІЛ 1
# АНАЛІЗ ПРОБЛЕМАТИКИ ЗА ТЕМОЮ РОБОТИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

## 1.1 Огляд і аналіз предметної області проблеми, результатів існуючих теоретичних та експериментальних досліджень

Сьогоднішній розвиток технологій відбувається дуже швидко. Здається, що всього кілька десятків років тому була створена перша електронна обчислювальна машина, яку згодом назвали «комп'ютером». Сьогодні ж практично будь-хто, хто вміє користуватися комп'ютером на базовому рівні, може створити свій перший простий сайт. Для цього не потрібно витрачати багато часу, достатньо виділити кілька годин чи днів і розібратися з основними інструментами.

Раніше розробка сайтів здійснювалася виключно за допомогою мов розмітки і програмування, таких як HTML, CSS, JavaScript і PHP. Розробники постійно шукали способи спростити процес створення вебресурсів і зробити його більш ефективним. В результаті з'явилися системи управління контентом – CMS (Content Management System), які значно полегшили роботу над сайтом для користувачів без глибоких знань програмування [2]. CMS дозволяють швидко створювати структуру сайту, додавати сторінки та контент, а також змінювати дизайн за допомогою готових шаблонів.

Проте такі системи мають і обмеження. Наприклад, часто необхідно купувати додаткові розширення для реалізації складного функціоналу. Код CMS не завжди гнучкий і легко модифікується, а підключення великої кількості плагінів може призвести до конфліктів та проблем із сумісністю. Тому CMS добре підходять для невеликих проєктів із простим дизайном та обмеженим функціоналом, але для більш складних систем, де потрібна індивідуальна бізнес-логіка, їх можливостей недостатньо. Популярні CMS, такі як WordPress, Wix, Shopify, BigCommerce або Drupal, часто використовуються у комерційних

проєктах, але їх обмеження роблять неможливим реалізацію деяких специфічних вимог.

У таких випадках розробка повертається до класичних мов програмування та фреймворків. Для front-end частини використовуються інструменти на кшталт Gulp, Webpack, Parcel або Grunt, які дозволяють мінімізувати CSS та JavaScript код і оптимізувати зображення. Front-end виглядає на перший погляд легким, але для професійної роботи необхідно знати значно більше: HTML, CSS, JavaScript, роботу з API, JSON, а також фреймворки та бібліотеки Angular, Vue.js, React та Node.js. Для цього потрібна практика та вміння орієнтуватися в документації, адже правильне застосування всіх технологій прямо впливає на якість сайту.

Що стосується back-end, тут головне обрати стек технологій. Найпопулярніші варіанти – PHP і Python. Python має свій фреймворк Django, який дозволяє швидко створювати вебзастосунки, проте на ринку України більше затребуваний PHP і фреймворк Laravel [3]. Laravel поєднує простоту використання, зрозумілу структуру коду і велику кількість готових рішень, що значно прискорює розробку. Його перевага також у великій спільноті та наявності великої кількості пакетів для різних завдань, включно з інтеграцією платіжних систем.

Для створення інтернет-магазину особливо важливою є інтеграція з платіжними системами. В Україні популярними сервісами є Monobank та WayForPay. Monobank надає зручне API для створення платіжних посилань і перевірки статусів платежів, а WayForPay підтримує різні валюти, callback-сповіщення та роботу як із фізичними, так і з юридичними особами. Використання цих платіжних систем у поєднанні з Laravel дозволяє створити надійну платформу для онлайн-продажів, яка відповідає сучасним вимогам безпеки та зручності для користувачів.

Таким чином, поєднання Laravel як основи розробки і інтеграції Monobank та WayForPay забезпечує оптимальний баланс між гнучкістю,

масштабованістю та простотою підтримки інтернет-магазину. Це дозволяє реалізувати складні бізнес-процеси, налаштувати зручний інтерфейс для користувача та забезпечити безпечну обробку платежів, що робить платформу готовою для подальшого розвитку та використання в реальних комерційних проєктах.

Чому саме платіжні системи? Тому що для створення інтернет-магазину важливо, щоб користувач міг швидко та безпечно оплачувати товари або послуги. Звісно, будь-який сучасний магазин має мати можливість здійснювати платежі онлайн, але не всі платіжні системи однаково зручні та гнучкі. Під «гнучкістю» мається на увазі можливість швидко інтегрувати платіжний сервіс, отримувати детальну інформацію про платежі, підтримувати різні способи оплати та валюти.

Також варто зазначити, що вибір платіжної системи залежить не лише від технічних можливостей API, а й від потреб конкретного бізнесу. Для інтернет-магазину важливі такі критерії, як швидкість обробки платежів, стабільність роботи сервісу, підтримка різних способів оплати та можливість інтеграції з іншими сервісами магазину, наприклад з обліком замовлень або системою управління товарами. Усе це дозволяє забезпечити комфортний користувацький досвід і зменшити ймовірність виникнення проблем під час оплати.

На ринку існує кілька популярних платіжних сервісів, які використовують для інтернет-магазинів. Розглянемо основні з них.

Monobank API – сучасна українська платіжна система, що дозволяє створювати платіжні посилання, отримувати сповіщення про статуси оплат і інтегруватися з вебзастосунками через простий REST API. Monobank зручний для малого та середнього бізнесу, його можна підключити без складних налаштувань. На рисунку 1.1 зображено інтерфейс головної сторінки документації «Monobank API».

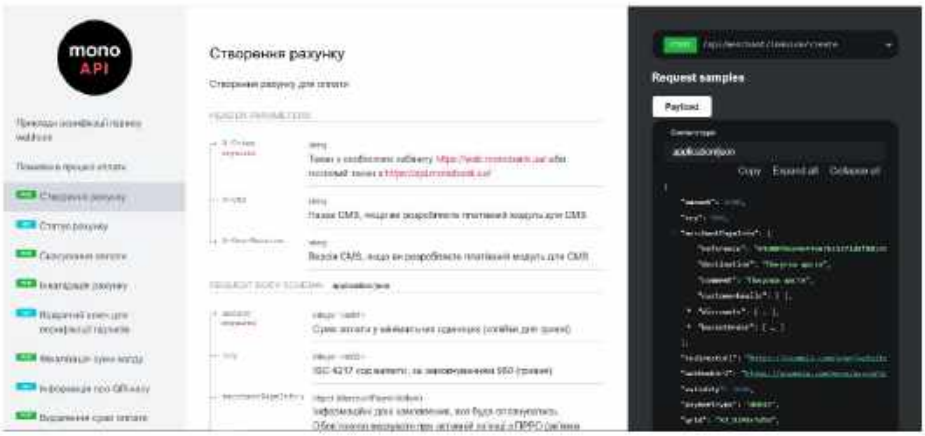


Рисунок 1.1 – Вигляд сторінки документації «Monobank API» [4]

«WayForPay» – сервіс, який підтримує оплату в різних валютах, автоматичне оброблення транзакцій та роботу як із фізичними, так і з юридичними особами. WayForPay надає callback-сповіщення, що дозволяє отримувати актуальний статус платежу і інтегрувати його в бізнес-логіку сайту. На рисунку 1.2 зображено інтерфейс головної сторінки документації «WayForPay API».

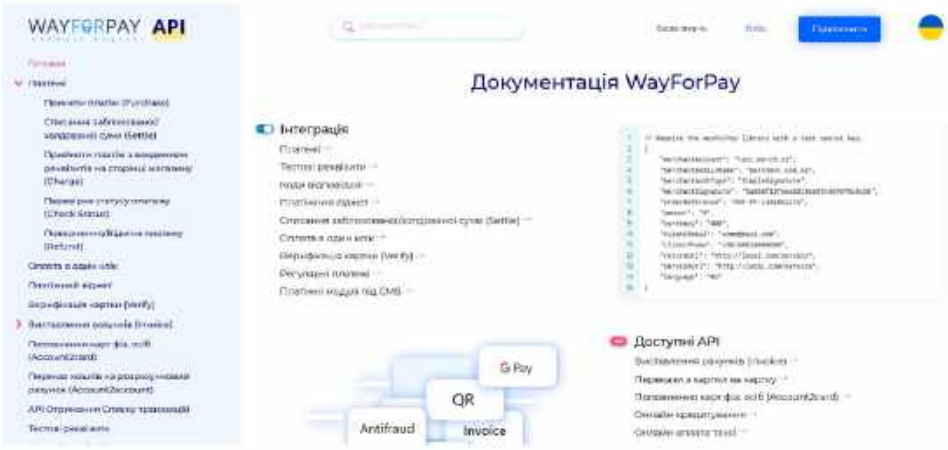


Рисунок 1.2 – Вигляд сторінки документації «WayForPay API» [5]

«LiqPay» – популярний український платіжний сервіс від ПриватБанку, який дозволяє приймати оплату картками «Visa» та «MasterCard», а також через «Apple Pay» та «Google Pay». Має зручний API і готові модулі для популярних

CMS та фреймворків. На рисунку 1.3 зображено інтерфейс головної сторінки документації «LiqPay API».

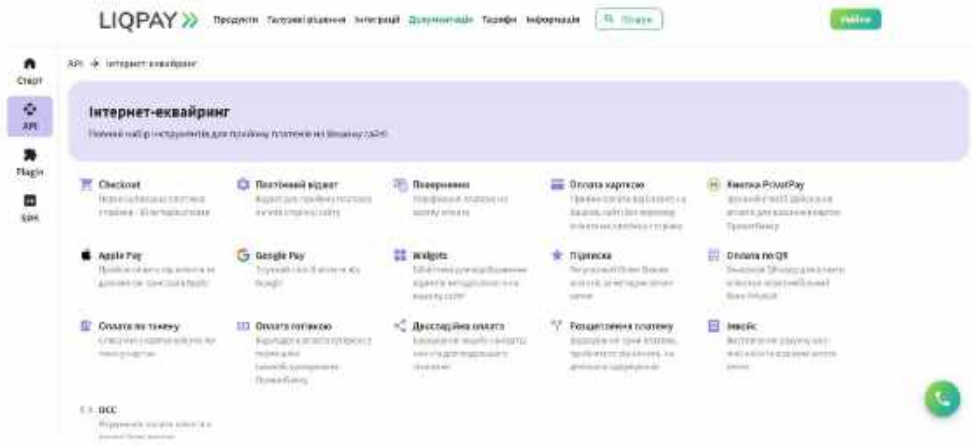


Рисунок 1.3 – Вигляд сторінки документації «LiqPay API» [6]

«Fondy» – міжнародна платіжна платформа, що підтримує прийом платежів у різних валютах та надає широкий функціонал для інтеграції з інтернет-магазинами, включно з регулярними платежами і підписками. На рисунку 1.4 зображено інтерфейс головної сторінки документації «API Fondy».


Рисунок 1.4 – Вигляд сторінки документації API Fondy [7]

«Stripe» – міжнародна платіжна платформа, що дозволяє приймати платежі картками, цифровими гаманцями та іншими способами, підтримує

підписки та регулярні платежі. Stripe має зручне API і добре документовану інтеграцію з популярними фреймворками, включаючи Laravel. На рисунку 1.5 зображено інтерфейс головної сторінки документації «Stripe API».

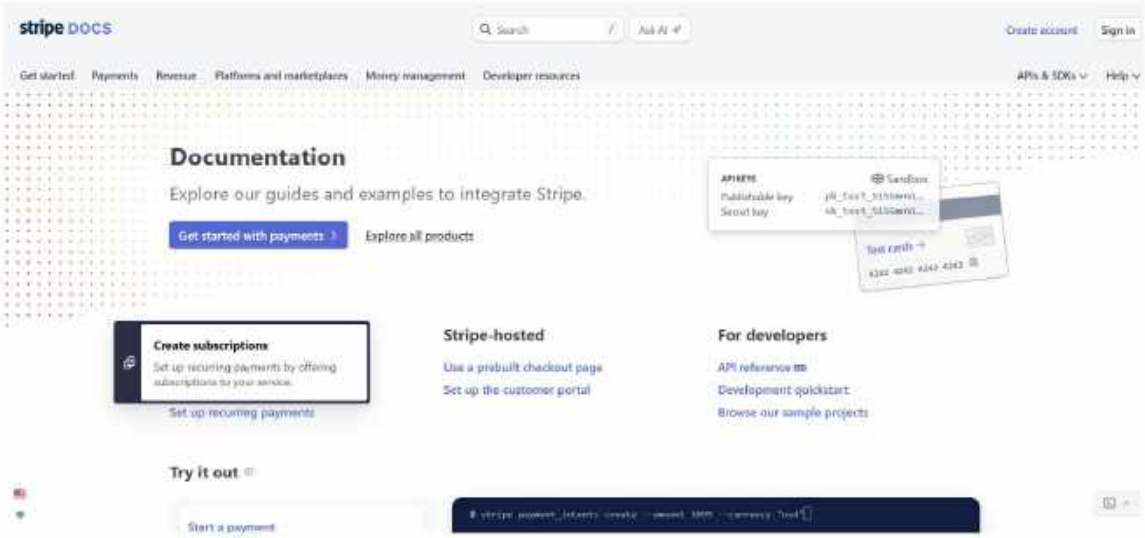


Рисунок 1.5 – Вигляд сторінки документації «Stripe API» [8]

Для мого інтернет-магазину я обрав Monobank та WayForPay, оскільки ці сервіси найкраще підходять для українського ринку. Вони підтримують оплату в гривнях, мають зрозуміле API та детальну документацію, завдяки чому інтеграція проходить швидко і без складнощів.

Також у лістингу 1.1 наведено приклад запиту до «Monobank API» для створення платежу та отримання підтвердження про його виконання. Даний приклад демонструє базову схему взаємодії між серверною частиною інтернет-магазину та платіжним сервісом: від формування запиту на стороні Laravel до отримання відповіді з інформацією про створену транзакцію. У фрагменті коду видно, як здійснюється ініціалізація платіжної сесії, передаються основні параметри платежу та формується структура запиту згідно з вимогами Monobank. Це дозволяє чітко зрозуміти, як налаштовується логіка проведення оплати, обробки відповіді та подальшої фіксації результату платежу в системі.

Лістинг 1.1 – Запит до Monobank API для створення рахунку.

```
curl -X POST https://api.monobank.ua/api/merchant/invoice/create
-H "X-Token: {токен}"
-d '{
  "amount": 10000,
  "currency": 980,
  "orderId": "335",
  "description": "Оплата товару",
  "redirectUrl": "https://mysite.com/webhook/payment/monobank"
}'
```

Кінець лістингу 1.1

WayForPay працює схожим чином, дозволяючи створювати платіжні кнопки, інтегрувати їх у сайт і отримувати callback-сповіщення про успішні та неуспішні транзакції.

Використання Laravel для інтеграції цих платіжних систем значно спрощує розробку. Laravel дозволяє організувати код структуровано, використовуючи MVC, легко працювати з API та обробляти транзакції через сервіси і контролери. Завдяки цьому інтеграція Monobank та WayForPay проходить швидко та без помилок, а користувач отримує надійний і зручний інтерфейс для оплати товарів.

Таким чином, поєднання Laravel і платіжних систем Monobank та WayForPay забезпечує стабільну та безпечну роботу інтернет-магазину, що важливо як для власника сайту, так і для його користувачів.

Сучасні інтернет-магазини перетворилися на невід'ємну частину цифрової економіки, забезпечуючи користувачам можливість придбання товарів та послуг у зручний спосіб, без необхідності фізичної присутності. Водночас із розвитком електронної комерції зростають вимоги до якості сервісу, швидкості обробки замовлень, безпеки персональних даних та зручності оплати. Користувачі очікують не лише широкий асортимент товарів, а й простий інтерфейс, надійність під час оплати та швидкий зворотний зв'язок. Це створює потребу у вдосконаленні технічних рішень, які лежать в основі таких систем.

Розробка інтернет-магазину вже не обмежується створенням сторінок із товарами – сьогодні це складна багаторівнева система, що включає управління базами даних, інтеграцію платіжних сервісів, обробку замовлень, аналітику та маркетингові інструменти. Для малого та середнього бізнесу особливо важливо мати рішення, яке дозволяє швидко масштабувати функціонал і при цьому залишатися фінансово доступним. Саме тому актуальним є пошук ефективних технологій і підходів до побудови інтернет-магазинів, що поєднують гнучкість, продуктивність і безпеку.

Отже, аналіз сучасного стану розвитку вебтехнологій та електронної комерції показує, що створення інтернет-магазинів залишається актуальним напрямом досліджень і практичної реалізації. З огляду на зростаючі вимоги користувачів та бізнесу, необхідно розробляти рішення, які забезпечуватимуть високу продуктивність, надійність і безпечну взаємодію між клієнтом та системою. Саме тому подальше дослідження методів, засобів та технологій розробки таких вебсистем є важливим кроком для підвищення ефективності електронної торгівлі та поліпшення користувацького досвіду.

## 1.2 Огляд і аналіз методів та засобів розробки інтернет-магазину з інтеграцією платіжних систем WayForPay та Monobank для розв'язання проблеми дослідження

Аналіз методів та засобів розробки інтернет-магазину й інтеграції платіжних систем є важливим етапом у процесі створення сучасного вебзастосунку. На цьому етапі визначаються оптимальні технології, архітектурні підходи та інструменти, які забезпечують ефективність, безпеку, масштабованість і зручність подальшої підтримки проєкту. Оскільки інтернет-магазин включає значну кількість взаємодіючих компонентів — від роботи з базою даних і формування REST API до реалізації клієнтського інтерфейсу та обробки онлайн-платежів — правильний вибір технологічного

стеку відіграє ключову роль. Від застосованих інструментів залежить стабільність роботи системи, швидкість обробки замовлень, надійність транзакцій та загальний користувацький досвід.

У межах цього розділу проаналізовано технологічний стек застосунку, внутрішню структуру проєкту, принципи побудови клієнтської та серверної частини, а також методи інтеграції платіжних систем WayForPay і Monobank у розроблений інтернет-магазин. Окрема увага приділена вибору інструментів для реалізації бізнес-логіки, роботі з даними, забезпеченню безпеки, а також відповідності архітектурних рішень сучасним вимогам веброзробки. Проведений аналіз дозволяє обґрунтувати вибір технологій та визначити їхню роль у досягненні поставленої мети.

Для розробки серверної частини інтернет-магазину було обрано PHP-фреймворк «Laravel», який є одним із найоптимальніших рішень для створення вебзастосунків комерційного рівня. Його популярність пояснюється високою продуктивністю, зручними інструментами для формування REST API, продуманою структурою та широкою екосистемою пакетів. Laravel підтримує чітке розділення відповідальностей, модульну архітектуру та сприяє написанню чистого й підтримуваного коду, що особливо важливо для масштабованих проєктів.

Фреймворк містить розвинуті механізми аутентифікації, авторизації, валідації даних, маршрутизації та логування, що істотно скорочує час розробки та зменшує кількість потенційних помилок. Завдяки широкій спільноті, активним оновленням та великій кількості готових рішень фреймворк залишається актуальним та стабільним інструментом для сучасної веброзробки.

У межах порівняння можливостей Laravel із іншими сучасними системами керування вебпроєктами проаналізовано продуктивність, складність інтеграції, зручність підтримки та гнучкість архітектури. Зведені характеристики подані в таблиці 2.1, що дозволяє обґрунтувати вибір фреймворку з точки зору практичних вимог до інтернет-магазину.

Таблиця 2.1 – Порівняльна характеристика популярних систем створення веб-сайтів

| Платформа / технологія | Тип системи | Переваги | Недоліки |
|---|---|---|---|
| WordPress | CMS | Простота використання, велика кількість плагінів і тем | Обмежена гнучкість, проблеми з безпекою при великій кількості плагінів |
| Wix | Конструктор сайтів | Не потребує знань програмування, швидкий старт | Закрита екосистема, обмеження у налаштуванні коду |
| Shopify | Saas платформа для e-commerce | Зручність керування товарами, інтеграція з платіжними системами | Щомісячна плата, складність глибокої кастомізації |
| Laravel | PHP-фреймворк | Висока гнучкість, модульність, масштабованість, активна спільнота | Потребує досвіду програмування |
| Django | Python-фреймворк | Надійність, швидка розробка, зручна ORM | Менш популярний на українському ринку, складніша хостинг-інтеграція |

Вибір Laravel також обґрунтований його активною спільнотою, детальною документацією та підтримкою сучасних стандартів програмування. Це дає можливість швидко реалізовувати REST API для взаємодії з фронтендом, розробленим на Nuxt 3, забезпечуючи стабільний і безпечний обмін даними між клієнтською та серверною частинами системи.

Для реалізації клієнтської частини інтернет-магазину було обрано Nuxt 3 [9]. Nuxt 3 – сучасний фронтенд-фреймворк на основі Vue.js, який поєднує зручність розробки, високу продуктивність та гнучкість у побудові інтерфейсу користувача [10]. Його ключовою перевагою є підтримка SSR (Server-Side Rendering) та SPA (Single Page Application) підходів, що

дозволяє оптимізувати швидкість завантаження сторінок і забезпечує кращу SEO-оптимізацію сайту.

Nuxt 3 надає зручну структуру проєкту, що сприяє розділенню логіки на компоненти, спрощує навігацію та повторне використання коду. Крім того, використання TypeScript [11] та Composition API [12] дає змогу створювати сучасний, адаптивний і масштабований інтерфейс користувача.

Завдяки вбудованим інструментам для роботи з API, таким як useFetch() та Axios, Nuxt 3 забезпечує ефективну взаємодію з бекендом на Laravel, що робить інтеграцію швидкою та стабільною. У результаті, Nuxt 3 став оптимальним вибором для створення зручного, динамічного та швидкодіючого інтерфейсу інтернет-магазину.

Для зберігання та обробки даних у проєкті використовується система керування базами даних MySQL [13]. Вибір саме цього рішення зумовлений його стабільністю, високою продуктивністю та широкою підтримкою у середовищі веброзробки. MySQL ефективно взаємодіє з фреймворком Laravel, забезпечуючи зручну роботу через ORM Eloquent, що дозволяє виконувати складні запити у простій та зрозумілій формі.

Крім того, MySQL характеризується хорошою масштабованістю, підтримкою транзакцій та надійними механізмами резервного копіювання. Завдяки своїй поширеності, вона має велику спільноту та розвинену екосистему інструментів для адміністрування, що спрощує налаштування та обслуговування бази даних у процесі розробки та розгортання інтернет-магазину.

У процесі розробки передбачається використання низки допоміжних інструментів, які забезпечать ефективність і зручність роботи над проєктом. Менеджери пакетів Composer (для PHP) та NPM [14] (для JavaScript) застосовуватимуться для встановлення й оновлення необхідних бібліотек та залежностей. Node.js використовуватиметься для виконання фронтенд-скриптів і збірки проєкту на Nuxt 3. Для тестування та налагодження API планується

використати Postman, що дозволить перевіряти коректність запитів і відповідей між клієнтською та серверною частинами. Система контролю версій Git допоможе відстежувати зміни в коді, працювати з гілками та підтримувати командну взаємодію. Розробку буде здійснено у середовищі Visual Studio Code, яке відзначається зручністю, швидкодією та широкими можливостями розширення функціоналу для ефективної роботи з кодом.

Для забезпечення ефективної взаємодії між клієнтською та серверною частинами проєкту буде використано принципи REST API (Representational State Transfer Application Programming Interface). Цей підхід дозволить організувати обмін даними у стандартизованому форматі, що забезпечить сумісність, простоту інтеграції та масштабованість системи.

REST API працюватиме на основі HTTP-запитів типів GET, POST, PUT, PATCH та DELETE, які відповідатимуть CRUD-операціям над даними. Усі відповіді від сервера повертатимуться у форматі JSON, що спрощує їх обробку на фронтенді.

Застосування REST-архітектури дасть змогу розділити відповідальність між клієнтом і сервером, забезпечити незалежність інтерфейсу від реалізації бекенду, а також полегшити тестування, оновлення та подальший розвиток застосунку. Такий підхід також стане основою для подальшої інтеграції зовнішніх сервісів, зокрема платіжних систем WayForPay та Monobank.

Під час розробки інтернет-магазину планується застосувати модульний підхід, який передбачає поділ системи на окремі логічні частини (модулі), кожна з яких відповідатиме за певну функціональність. Такий підхід забезпечить зручність у підтримці, тестуванні та подальшому розширенні проєкту без порушення роботи інших компонентів.

У межах архітектури буде передбачено створення окремих сервісів для роботи з користувачами, замовленнями, кошиком, товарами, категоріями та платежами. Кожен сервіс реалізовуватиме власну бізнес-логіку, що дозволить ізолювати відповідальність і підвищити надійність системи. Наприклад, сервіс

замовлень оброблятиме процес оформлення покупки, тоді як сервіс платежів – інтеграцію з платіжними системами та перевірку статусу транзакцій.

Такий підхід відповідає принципам SOLID і сприяє гнучкості архітектури, що є важливою умовою для подальшого масштабування інтернет-магазину та впровадження нових функцій [15].

У серверній частині інтернет-магазину, що буде реалізована на основі Laravel 12, планується чітко організувати структуру додатку згідно з сучасними принципами архітектури та кращими практиками фреймворку.

Роутінг здійснюватиметься через файли «routes/api.php» та «routes/web.php». Усі запити до REST API будуть визначені у файлі «api.php», що забезпечить логічне розділення між публічними та внутрішніми маршрутами. Для зручності обробки запитів планується використання груп маршрутів із префіксами, middleware та контролерами.

Middleware використовуватимуться для виконання проміжних перевірок під час обробки запитів, зокрема для автентифікації користувачів, перевірки прав доступу, обробки CORS-запитів та захисту від несанкціонованого доступу.

Контролери будуть відповідати за реалізацію бізнес-логіки кожного модуля системи. Вони повертатимуть відповіді у форматі JSON, забезпечуючи коректну взаємодію з фронтендом на Nuxt 3. Для зручності та підтримки чистої архітектури буде застосовано розподіл на API-контролери та внутрішні контролери адміністративної частини.

Ресурси (Resources) використовуватимуться для форматування даних перед відправкою на клієнтську частину, що забезпечить структурованість та єдиний стиль відповіді API.

Крім того, планується реалізувати сервісний шар (Service Layer), який інкапсулюватиме бізнес-логіку, забезпечуючи гнучкість і можливість повторного використання коду. Такий підхід сприятиме підтримуваності коду, дотриманню принципів SOLID, а також покращить тестованість системи.

У процесі розробки клієнтської частини інтернет-магазину буде створено зручний та інтуїтивно зрозумілий інтерфейс користувача, що забезпечить легку навігацію та швидке оформлення покупок. Основними компонентами інтерфейсу стануть:

— головна сторінка, яка відображатиме основну інформацію про магазин, акції та популярні товари;

— каталог товарів, де користувачі зможуть переглядати асортимент і швидко переходити до перегляду детальної інформації про кожен товар;

— сторінка одного товару, що міститиме деталі продукту, опис, характеристики, доступні опції та кнопку додавання до кошика;

— кошик, де буде відображатися інформація про вибрані товари, їхню кількість та загальну суму замовлення;

— сторінка оформлення замовлення, що забезпечить введення даних користувача, вибір способу доставки та оплати, а також підтвердження покупки.

Для реалізації цих компонентів буде використано Nuxt 3 із застосуванням компонентного підходу, що дозволить повторно використовувати код і спростить подальшу підтримку інтерфейсу. Верстка буде здійснюватися з використанням SCSS, що дозволить створювати сучасний, адаптивний і структурований дизайн сайту.

Завдяки такій організації фронтенду користувачі зможуть легко переглядати товари, додавати їх до кошика та оформлювати замовлення, отримуючи стабільний і швидкий досвід взаємодії з інтернет-магазином.

Для забезпечення безпечного доступу користувачів до функціоналу інтернет-магазину планується реалізація автентифікації та авторизації за допомогою Bearer-токенів, згенерованих сервером на основі JWT (JSON Web Tokens). Такий підхід дозволить відокремити клієнтську та серверну частини, забезпечуючи надійний захист персональних даних і зручність у роботі з REST API.

Під час реєстрації та входу користувача сервер на Laravel 12 буде генерувати токен доступу, який фронтенд на Nuxt 3 включатиме у заголовок HTTP-запитів «Authorization: Bearer токен» для підтвердження прав доступу при взаємодії з API. Токен зберігатиметься у безпечному місці на клієнтській стороні, що гарантує безпеку сесії користувача.

Використання Bearer-токенів забезпечить гнучке керування сесіями, можливість легко впроваджувати різні рівні доступу до функцій системи та спростить інтеграцію фронтенду з бекендом, а також з іншими сервісами та модульними компонентами інтернет-магазину.

Для взаємодії фронтенду на Nuxt 3 з серверною частиною, реалізованою на Laravel 12, планується використання функцій «useAsyncData», «useFetch» та «fetch», які надає Nuxt 3 для роботи з HTTP-запитами. Ці інструменти дозволять здійснювати асинхронні запити до REST API та обробляти відповіді сервера у форматі JSON, забезпечуючи коректний обмін даними між клієнтською та серверною частинами.

Всі запити до захищених ресурсів будуть містити Bearer-токен у заголовку Authorization, що дозволить серверу підтверджувати права доступу користувача та захищати персональні дані. Використання useAsyncData, useFetch та стандартного fetch дає змогу централізовано налаштовувати базову URL-адресу API, передавати заголовки та обробляти помилки, а також ефективно працювати з даними на стороні клієнта у режимі SSR або SPA.

Завдяки такій інтеграції фронтенд зможе отримувати список товарів, деталі одного продукту, обробляти дії у кошику та оформляти замовлення, забезпечуючи швидку, стабільну та реактивну взаємодію користувача з інтернет-магазином.

У межах розробки серверної частини інтернет-магазину на Laravel 12 планується реалізація повного циклу CRUD-операцій (Create, Read, Update, Delete) для ключових сутностей системи – товарів, категорій, користувачів та замовлень. Важливо, що ці операції будуть реалізовані в адміністративній

панелі (адмінці), призначеній для управління даними та контролю функціоналу системи, а не для клієнтської частини.

Для товарів адміністративна панель дозволить додавати нові продукти, переглядати список і деталі, редагувати характеристики та видаляти товари. Для категорій буде забезпечено створення, редагування, перегляд і видалення, що дозволить структурувати каталог товарів і підтримувати його актуальність. Операції для користувачів включатимуть перегляд профілів, оновлення даних та, за потреби, видалення облікових записів. Для замовлень передбачено створення нових записів, оновлення статусів (наприклад, «оплачене», «у процесі доставки») та перегляд історії замовлень.

Всі CRUD операції будуть реалізовані через REST API із застосуванням ресурсів для форматування даних і middleware для автентифікації та авторизації. Це забезпечить безпеку, модульність і масштабованість бекенд-частини, а також полегшить інтеграцію з фронтендом на Nuxt 3.

Для реалізації інтерфейсу адміністративної панелі планується використання AdminLTE 3 – готового шаблону адмінки з сучасним дизайном та зручною структурою компонентів. AdminLTE 3 дозволить швидко створювати сторінки для управління товарами, категоріями, користувачами та замовленнями, забезпечить готові елементи таблиць, форм, панелей навігації та модальних вікон. Використання цього шаблону спростить верстку, прискорить розробку адмінки і забезпечить інтуїтивно зрозумілий інтерфейс для адміністратора системи.

Для забезпечення надійності та стабільності роботи інтернет-магазину планується впровадження системи логування та обробки помилок у бекенд-частині.

Логування буде здійснюватися за допомогою вбудованих механізмів Laravel, зокрема через фасад «Log», що дозволяє записувати інформацію про ключові події системи, помилки, запити користувачів та результати виконання бізнес-логіки. Логи будуть зберігатися у вигляді файлів або надсилатися до

зовнішніх сервісів моніторингу, що полегшить відстеження проблем та аналіз роботи системи. Планується логування таких подій, як створення, оновлення та видалення даних, неуспішні спроби авторизації, а також критичні помилки сервера.

Обробка помилок буде реалізована через глобальний обробник виключень Laravel (App\Exceptions\Handler), який дозволить перехоплювати як системні, так і користувацькі помилки, надавати стандартизовані відповіді API у форматі JSON та повертати користувачеві зрозумілі повідомлення про проблему. Планується застосовувати кастомні виключення для модулів, таких як товари, замовлення або платежі, що дозволить точніше визначати причину помилки та швидко реагувати на неї.

Завдяки логуванню та централізованій обробці помилок буде забезпечено стабільність, безпеку та прозорість роботи системи, а також спрощено підтримку та подальший розвиток інтернет-магазину.

У межах підготовчого етапу інтеграції платіжних систем планується ознайомлення з API платіжних сервісів WayForPay та Monobank. Це дозволить вибрати оптимальний підхід для реалізації безпечних і надійних платежів у інтернет-магазині та забезпечить коректну взаємодію фронтенду і бекенду з платіжними шлюзами.

Для WayForPay буде вивчено можливості API щодо обробки транзакцій, перевірки статусу платежів, створення платіжних форм і генерації безпечних підписів для підтвердження платежів. Особлива увага буде приділена методам інтеграції, що підтримують віддалену авторизацію платежів і забезпечують захист від шахрайства.

Для Monobank планується ознайомлення з REST API, яке дозволяє створювати платіжні запити, перевіряти статуси транзакцій, отримувати інформацію про проведені операції та інтегрувати платіжні посилання у процес оформлення замовлення. Вивчення документації Monobank також включатиме принципи автентифікації за допомогою Bearer-токенів, формат запитів і

відповідей, а також обмеження на кількість запитів та правила обробки помилок.

Ознайомлення з API обох сервісів дозволить побудувати надійну та безпечну систему оплати, визначити необхідні ендпоінти для бекенд-частини на Laravel 12 і підготувати фронтенд Nuxt 3 до коректної взаємодії з платіжними сервісами під час оформлення замовлення.

Для інтеграції платіжних систем у проєкт планується реалізація окремих драйверів або сервіс-класів для кожної платіжної системи: WayForPay та Monobank. Такий підхід забезпечить ізоляцію бізнес-логіки кожного платіжного сервісу, гнучкість у роботі та полегшить подальшу підтримку й масштабування системи.

Кожен сервіс-клас буде відповідати за:

— формування запитів до API платіжної системи;

— перевірку і валідацію даних, що надсилаються;

— обробку відповідей від сервера платіжної системи, включаючи статус транзакцій;

— логування дій та помилок для подальшого аналізу;

— генерацію підписів або токенів для безпечного проведення платежів.

Використання окремих драйверів дозволить легко додавати нові платіжні системи у майбутньому, не змінюючи наявну логіку інших модулів. Також це сприятиме дотриманню принципів SOLID, зокрема Single Responsibility, коли кожен клас відповідає лише за інтеграцію з одним сервісом.

Завдяки такій організації, бекенд на Laravel 12 буде готовим до безпечної та стабільної взаємодії з фронтендом на Nuxt 3 під час процесу оформлення замовлення та обробки платежів, а також до подальшого масштабування платіжної функціональності.

Для забезпечення стабільності, коректності роботи та швидкого виявлення помилок у проєкті планується використання Postman та інструментів Laravel Pint і Pest для автоматизованого тестування.

Postman дозволить тестувати REST API бекенд-частини на Laravel 12, перевіряти коректність запитів і відповідей, відпрацьовувати сценарії взаємодії фронтенду на Nuxt 3 із сервером, а також тестувати інтеграцію з платіжними системами WayForPay та Monobank. Використання колекцій і середовищ у Postman дасть змогу створювати набори тестів, що легко повторно виконуються та підтримуються.

Laravel Pint буде застосовуватися для автоматичного форматування коду відповідно до стандартів, що забезпечить чистоту та однорідність коду, спростить його читання та подальшу підтримку.

Pest дозволить створювати модульні тести для перевірки роботи окремих компонентів бекенду, включаючи моделі, контролери, сервіси та інтеграцію з платіжними системами. Завдяки простому синтаксису Pest тестування буде зрозумілим і зручним, що сприятиме швидкому виявленню помилок та підвищенню надійності системи.

Поєднання цих інструментів забезпечить ефективне автоматизоване тестування, стабільність роботи інтернет-магазину, контроль якості коду та швидке виявлення потенційних проблем під час розробки та інтеграції нових функцій.

Для забезпечення швидкої та стабільної роботи інтернет-магазину планується оптимізація запитів і продуктивності як на серверній, так і на клієнтській частинах. На стороні бекенду, реалізованого на Laravel 12, це передбачатиме ефективне використання Eloquent relationships із правильним застосуванням «lazy» та «eager loading» для уникнення надлишкових запитів до бази даних, а також індексацію колонок для прискорення пошуку та сортування. Часто запитувані дані будуть кешуватися за допомогою Redis або вбудованого кешу Laravel, наприклад, для списків товарів і категорій. Крім того, планується

оптимізація складних SQL-запитів і використання агрегатних функцій, що дозволить зменшити навантаження на базу даних. Профілювання запитів через інструменти, такі як Laravel Telescope, дозволить виявляти та усувати «вузькі місця» у продуктивності.

На стороні фронтенду, який буде реалізований на Nuxt 3, оптимізація передбачатиме використання SSR та механізму hydration для швидкого рендерингу сторінок, а також ефективну роботу з асинхронними запитами через «useAsyncData», «useFetch» та «fetch», що дозволить уникати повторних викликів API. Для зменшення обсягу передаваних даних буде застосовано пагінацію та обмеження кількості елементів у списках товарів, а кешування отриманих даних на клієнтській стороні допоможе знизити навантаження на сервер.

Завдяки цим заходам буде забезпечено швидке завантаження сторінок, стабільну роботу інтерфейсу та ефективне використання ресурсів серверної частини, що підвищить зручність користування інтернет-магазином і загальну продуктивність системи. Це також сприятиме кращій масштабованості застосунку, зменшенню часу відгуку при зростанні кількості користувачів та забезпечить стабільну роботу навіть у пікові періоди навантаження.

У цьому підрозділі проаналізовано методи та засоби розробки інтернет-магазину й інтеграції платіжних систем, що забезпечать ефективну, безпечну та стабільну роботу майбутнього проєкту. Передбачається використання Laravel 12 для бекенду та Nuxt 3 для фронтенду, що гарантує зручний REST API та швидкий адаптивний інтерфейс. Реалізація автентифікації через Bearer-токени, інтеграції з WayForPay і Monobank, а також створення адмінпанелі на базі AdminLTE 3 забезпечать повний функціонал системи. Завдяки логуванню, тестуванню та оптимізації продуктивності проєкт буде надійним, масштабованим і зручним для користувачів і адміністратора.

## 1.3 Постановка завдання на кваліфікаційну роботу магістра

Метою кваліфікаційної роботи є розробка інтернет-магазину з інтеграцією платіжних систем Monobank та WayForPay на основі фреймворку Laravel та клієнтської частини, реалізованої у Nuxt 3. Система повинна забезпечувати стабільну роботу, безпечну обробку платежів і зручність використання.

Для досягнення поставленої мети необхідно виконати такі завдання:

— здійснити аналіз вимог до сучасних інтернет-магазинів, включно з функціональністю, безпекою та підтримкою онлайн-оплат;

— розробити серверну частину вебсистеми на основі фреймворку Laravel та налаштувати базову архітектуру проєкту;

— створити адміністративну панель для управління товарами, категоріями, замовленнями та користувачами;

— розробити клієнтську частину інтернет-магазину з використанням Nuxt 3 та забезпечити її адаптивність для різних пристроїв;

— інтегрувати платіжні системи Monobank та WayForPay за допомогою API та механізму Webhook;

— реалізувати повноцінний REST API для взаємодії клієнтської частини з бекендом;

— виконати підготовку технічної документації щодо структури, реалізації та експлуатації розробленої системи.

Реалізація цих завдань забезпечить створення повноцінного інтернет-магазину, придатного до практичного використання та подальшого розширення. Отримані результати дадуть можливість продемонструвати ефективність застосування сучасних вебтехнологій, платіжних API та архітектурних підходів у реальному проєкті. Виконання роботи також підтвердить здатність розв'язувати комплексні задачі, пов'язані з розробкою, інтеграцією та забезпеченням стабільної роботи вебзастосунків.

# РОЗДІЛ 2
# ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ РОЗРОБКИ ІНТЕРНЕТ-МАГАЗИНУ З ІНТЕГРАЦІЄЮ ПЛАТІЖНИХ СИСТЕМ MONOBANK ТА WAYFORPAY

## 2.1 Обґрунтування вибору шляхів, технологій, алгоритмів і засобів вирішення поставленого завдання

У межах даної роботи розглядається предметна область електронної комерції, а саме створення інтернет-магазину з можливістю онлайн-оплат. Інтернет-магазини сьогодні є одним із найпоширеніших способів продажу товарів, тому важливо врахувати типові функції таких систем: каталог товарів, пошук, фільтри, кошик, оформлення замовлення та інтеграцію з платіжними сервісами. Під час аналізу було вивчено сучасні підходи до побудови вебзастосунків, їх архітектуру, вимоги користувачів та технічні особливості роботи з платежами. Це дозволило сформувати загальне бачення того, як має працювати майбутня система та які технології найкраще підходять для її реалізації.

Під час визначення вимог до системи було встановлено основні функції, які повинен виконувати інтернет-магазин. Система має забезпечувати зручний перегляд товарів, роботу з фільтрами, кошиком та можливість оформлення замовлення. Також важливо підтримувати реєстрацію й авторизацію користувачів, а для адміністратора – керування товарами, категоріями та замовленнями. Основною вимогою є інтеграція з платіжними сервісами Monobank і WayForPay, щоб користувач міг безпечно здійснювати онлайн-оплати. Крім того, система повинна бути стабільною, безпечною, адаптивною та здатною витримувати збільшення кількості користувачів у майбутньому.

Для кращого розуміння, чому для проєкту був обраний саме Laravel, я вирішив порівняти його з іншими популярними технологіями. Спочатку я

переглянув можливі варіанти, які зазвичай використовуються для веброзробки, і звернув увагу на їх сильні та слабкі сторони. Таке порівняння допомогло оцінити, наскільки кожен фреймворк підходить для задач мого проєкту, зокрема по продуктивності, зручності розробки та доступності інструментів.

Нижче наведена таблиця 2.1 з коротким порівнянням технологій, яку я використовував для обґрунтування вибору Laravel. Це дозволило більш об'єктивно оцінити функціональність та визначити, який інструмент найкраще підходить для реалізації системи.

Таблиця 2.1 – Порівняння фреймворків для створення вебзастосунку

| Критерій | Laravel (PHP) | Symfony (PHP) | Node.js (Express) |
|---|---|---|---|
| Крива навчання | Низька, легкий старт | Вища, багато конфігурацій | Середня |
| Швидкість розробки | Висока завдяки вбудованим інструментам | Нижча через більшу кількість налаштувань | Висока, але більше ручної роботи |
| Архітектура | MVC, чітка структура | Дуже строгий, enterprise-рівень | Мінімалістична, залежить від розробника |
| Документація | Зрозуміла, доступна | Дуже детальна, але складніша | Залежить від бібліотек |
| Робота з БД | Eloquent ORM (проста й зручна) | Doctrine ORM (потужна, але складна) | Немає стандартного ORM |
| Готові рішення | Багато пакетів, Laravel Breeze, Cashier, Horizon | Менше готових рішень «із коробки» | Все через сторонні бібліотеки |
| Спільнота | Дуже велика та активна | Активна, але більш вузька | Дуже велика й різноманітна |
| Продуктивність | Висока для веб-проєктів | Висока, підходить для великих систем | Висока, особливо для реального часу |
| Інтеграція платежів | Простий підхід через HTTP-клієнт + пакети | Також можлива, але складніша | Можлива, але залежить від бібліотек |

На основі проведеного порівняння було обрано Laravel, оскільки він забезпечує швидку розробку, має зручну архітектуру, простий ORM та велику кількість готових рішень. Це дозволяє ефективно реалізувати інтернет-магазин із підтримкою онлайн-оплат і спростити інтеграцію з API платіжних сервісів.

Для цього проєкту я планую використовувати архітектурний підхід, що поєднує MVC та REST. Зараз я розглядаю ці два підходи як найбільш зручні для побудови сучасного вебзастосунка. MVC дає змогу структуровано розділяти логіку, дані та відображення, що в майбутньому повинно спростити розробку і підтримку. REST, у свою чергу, допоможе організувати зрозумілу взаємодію між клієнтською та серверною частиною через API. Я виходжу з того, що таке поєднання дозволить зробити систему гнучкішою і логічно впорядкованою під час подальшої реалізації.

Для розробки веб-проєкту я розглядаю декілька можливих систем керування базами даних. Для інтернет-магазину важливо, щоб СУБД була швидкою, стабільною та могла працювати під навантаженням. Також важливим є питання простоти інтеграції з Laravel, оскільки це напряму впливає на швидкість розробки та підтримку у майбутньому. Для порівняння я вибрав найбільш популярні СУБД, що представлені у таблиці 2.2.

Таблиця 2.2 – Порівняння популярних систем керування базами даних

| СУБД | Тип | Переваги | Недоліки |
|---|---|---|---|
| MySQL | Реляційна | Висока продуктивність, стабільність у роботі під навантаженням, проста інтеграція з Laravel, велика кількість готових інструментів та документації, зручний у налаштуванні | Для дуже складних аналітичних задач може знадобитися додаткове оптимізування |
| PostgreSQL | Реляційна | Розширені можливості для складних SQL запитів, підтримка спеціальних типів даних | Вища вимогливість до ресурсів, складніше налаштовувати |

Продовження таблиці 2.2

| СУБД | Тип | Переваги | Недоліки |
|---|---|---|---|
| SQLite | Вбудована реляційна | Простота у використанні, не потребує розгортання сервера | Погано підходить для систем з великою кількістю одночасних користувачів, обмежена масштабованість |

На основі порівняння я бачу, що MySQL найбільше підходить під вимоги майбутньої системи. Вона поєднує в собі високу продуктивність, простоту інтеграції і зручність роботи з Laravel. Також MySQL має широку документацію та підтримку, що допоможе у процесі подальшої розробки, тестування і масштабування вебзастосунка. Саме тому я планую використовувати MySQL як основну СУБД у цьому проєкті.

Для роботи з базою даних у Laravel я планую використовувати вбудовану ORM Eloquent. Основною причиною є те, що Eloquent значно спрощує взаємодію з даними. Робота з таблицями відбувається через моделі, а не через складні SQL-запити, що робить код більш зрозумілим і структурованим. Також Eloquent підтримує зв'язки між таблицями, фільтрацію, пагінацію, транзакції та інші інструменти, які будуть потрібні у процесі розробки інтернет-магазину.

Ще однією важливою перевагою є те, що ORM повністю інтегрована у Laravel, тому немає потреби підключати сторонні бібліотеки або додатково налаштовувати доступ до бази даних. Це дозволяє швидше розробляти функціонал і уникнути зайвих помилок.

У моєму проєкті Eloquent забезпечує оптимальний варіант для роботи з даними, оскільки він поєднує простоту використання, хорошу продуктивність і зручність подальшої підтримки коду.

У розробці проєкту я планую використовувати REST-підхід для створення API. Основною причиною є те, що REST забезпечує просту і зрозумілу

структуру взаємодії між клієнтом і сервером. Дані передаються у стандартному форматі JSON, який легко обробляється як на фронтенді, так і на бекенді. REST також не прив'язаний до конкретних технологій, тому в майбутньому його можна буде використовувати для мобільного застосунку або інших сервісів.

Ще однією перевагою цього підходу є те, що REST дозволяє розділяти відповідальність між частинами системи. Кожен маршрут виконує окрему дію, а доступ до ресурсів будується за принципом зрозумілих ендпоінтів. Така структура полегшує тестування, масштабування та подальшу підтримку проєкту.

Для розробки клієнтської частини планується використовувати NuxtJS. Однією з основних причин вибору є те, що NuxtJS значно спрощує роботу з Vue та структурує проєкт. Фреймворк має готову маршрутизацію, SSR-можливості та зручну організацію компонентів, що дозволяє швидше розробляти функціонал. Також важливо, що NuxtJS добре підходить для SPA та інтернет-магазинів, оскільки забезпечує високу швидкість завантаження сторінок та зручну роботу з API.

Крім цього, NuxtJS має багато додаткових модулів для роботи з авторизацією, SEO та кешуванням. Це допоможе у майбутньому масштабувати проєкт і покращувати продуктивність без значних змін у структурі додатку. Тому NuxtJS виглядає оптимальним варіантом для створення сучасного та адаптивного інтерфейсу.

Для вибору технології для клієнтської частини розглянуто декілька сучасних рішень, які використовуються для створення вебінтерфейсів. Основними критеріями були простота розробки, швидкість завантаження сторінок, підтримка SPA/SSR та можливість гнучко працювати з API (табл. 2.3). Також важливо враховувати, наскільки легко проєкт можна масштабувати в майбутньому та наскільки велика спільнота підтримує інструмент.

Таблиця 2.3 – Порівняння фронтенд-технологій

| Технологія | Особливості | Переваги | Недоліки |
|---|---|---|---|
| NuxtJS | Фреймворк на Vue | Готова структура проєкту, SSR, SEO, просте налаштування, швидка розробка | Потребує базових знань Vue |
| React | Бібліотека | Великий екосистемний вибір, популярність | Потрібно багато інструментів вручну (роутер, управління станом, SSR) |
| Angular | Повний фреймворк | Готові рішення під великі системи, типізація | Високий поріг входу, складність |
| VueJS | Фреймворк | Простий синтаксис, легка адаптація | Доведеться самостійно налаштовувати routing, SSR та структуру |

На основі порівняння я використовуватиму NuxtJS, оскільки він оптимально підходить під задачі проєкту. NuxtJS дозволяє швидше організувати структуру, має вбудовані інструменти для роботи з API та SSR, а також спрощує подальший розвиток системи. Цей фреймворк виглядає найбільш збалансованим варіантом за функціональністю, складністю та продуктивністю.

Під час розробки інтернет-магазину важливо правильно обрати платіжні системи, які забезпечать безпечні та швидкі фінансові операції для користувачів. Сучасний ринок пропонує великий вибір сервісів, кожен із яких має свої переваги та недоліки. Для проєкту важливо оцінити простоту інтеграції, доступність API, підтримку популярних платіжних методів та надійність сервісу.

У ході аналізу було вирішено порівняти найбільш популярні платіжні системи, що підходять для інтернет-магазинів на українському ринку та міжнародному середовищі. Це дозволить обґрунтувати вибір сервісів для

інтеграції та забезпечить зручність подальшої роботи як розробнику, так і користувачам. У таблиці 2.4 наведено порівняння основних характеристик платіжних систем, які розглядаються для реалізації проєкту.

Таблиця 2.4 – Порівняння платіжних систем

| Платіжна система | Переваги | Недоліки | Кому підходить |
|---|---|---|---|
| Monobank | Просте REST API; швидка інтеграція; популярність в Україні; прозора документація | Немає готових рішень "із коробки" для інтернет-магазинів | Для швидких кастомних інтеграцій |
| WayForPay | Підтримка Apple Pay/Google Pay; готові рішення для e-commerce; хороший інструментарій для платежів | Потрібні обов'язкові налаштування магазину та перевірка | Для комерційних проєктів та маркетплейсів |
| LiqPay | Популярність; підтримка кас і чеків; зручні модулі | Трохи складніша документація; менш гнучкий API | Для магазинів з класичною структурою |
| Fondy | Підтримка мультивалютних платежів; простий API; готові віджети | Менш популярний в Україні; обмежена документація | Для магазинів з міжнародними клієнтами |
| Stripe | Потужний API; глобальна підтримка; багато готових SDK; підтримка підписок | Складніша інтеграція для новачків; комісії вищі | Для стартапів і міжнародних магазинів |

Як видно з порівняння, кожна платіжна система має свої сильні та слабкі сторони. Вибір конкретного сервісу залежить від потреб проєкту: Monobank і WayForPay добре підходять для українського ринку та швидкої інтеграції, тоді як Fondy і Stripe забезпечують ширші можливості для міжнародних клієнтів. На основі цього аналізу планується інтегрувати у розроблюваний інтернет-магазин Monobank та WayForPay, що дозволить забезпечити надійну роботу платежів та зручність для користувачів.

Одним із ключових механізмів у взаємодії з платіжними системами буде використання «Webhook». Цей підхід дозволяє платіжному сервісу автоматично надсилати дані на сервер після того, як відбулася відповідна подія, наприклад підтвердження транзакції. Завдяки цьому немає потреби постійно виконувати опитування API, а система одразу отримує інформацію про результати оплати. «Webhook» дає можливість швидше обробляти статуси замовлень і забезпечує більш гнучку інтеграцію з платіжними сервісами. Під час реалізації я також врахую перевірку повідомлень та безпечне приймання даних.

Алгоритм взаємодії з API платіжних сервісів передбачає кілька етапів обміну даними між системами. Спочатку сервер надсилає запит із параметрами платежу до API, після чого платіжний сервіс повертає відповідь із попереднім статусом транзакції. Далі здійснюється перенаправлення або автоматична обробка відповіді залежно від результату операції. На завершальному етапі система має отримати підтвердження платежу, перевірити цифровий підпис та оновити статус замовлення. Такий підхід дозволяє налаштувати повний цикл взаємодії з платіжним сервісом і гарантує правильність виконання операцій.

Питання безпеки займає важливе місце під час створення інтернет-магазину, тому окремо розглядаються підходи до захисту даних користувачів та платіжних операцій. Для авторизації і керування доступом передбачено використання вбудованих механізмів Laravel, які забезпечують розмежування ролей і перевірку прав користувача. Для запобігання CSRF-атак планується застосування токенів, що дозволяють перевірити достовірність запиту та гарантують, що він був надісланий саме з довіреного джерела. Також буде враховано захист від XSS-вразливостей за допомогою фільтрації та екранування даних на стороні сервера. Сукупне застосування цих підходів має підвищити надійність системи та мінімізувати ризики несанкціонованого доступу.

Організація валідації даних є важливою частиною майбутньої системи, адже саме вона забезпечує коректність введеної користувачами інформації та

стабільність роботи сервісу. Для перевірки даних я розглядаю використання вбудованих механізмів Laravel, оскільки вони пропонують зручний набір правил і дозволяють легко контролювати правильність запитів як на стороні сервера, так і під час роботи з формами. Окрему увагу планується приділити перевірці платіжних даних та інформації про замовлення, щоб запобігти помилкам та некоректним операціям. Завдяки такому підходу можна буде забезпечити високу якість обробки даних і зменшити кількість можливих помилок у процесі оплати та оформлення замовлення.

Під час розробки інтернет-магазину важливо також продумати методи оптимізації продуктивності, оскільки від цього залежить швидкість роботи системи та зручність для користувача. Особливу увагу варто звернути на кешування, оскільки воно дозволяє зменшити кількість повторних запитів до бази даних та прискорює завантаження сторінок. Також планується використовувати оптимізацію запитів і індексування таблиць у MySQL, щоб знизити навантаження на сервер. У майбутньому можуть застосовуватись додаткові засоби оптимізації, такі як стиснення ресурсів, оптимізація зображень і черги для фонових операцій. Комплексний підхід дасть змогу покращити швидкодію системи та забезпечити стабільну роботу при збільшенні кількості користувачів.

На бекенд частині проєкту будуть використані різні бібліотеки, що полегшують розробку та додають необхідний функціонал. Наприклад, «kalnoy/nestedset» допомагає ефективно працювати зі вкладеними категоріями, а «laravel/sanctum» забезпечує безпечну автентифікацію користувачів. «Lorisleiva/laravel-actions» спрощує організацію бізнес-логіки через Actions, а «spatie/laravel-permission» дозволяє гнучко налаштовувати права доступу [16]. Для роботи з медіафайлами будуть застосовані «unisharp/laravel-filemanager» і «intervention/image-laravel» [17]. Інші пакети забезпечують зручну роботу з аватарами, інтерактивне тестування та розширення можливостей «Tinker».

На фронтенді планується використання модулів Nuxt, що дозволяють створювати сучасний, адаптивний і динамічний інтерфейс. @vueuse/nuxt забезпечує корисні утиліти для роботи з Vue, @pinia/nuxt відповідає за управління станом додатка, а nuxt-typed-router спрощує роботу з маршрутизацією. Для роботи з датами застосовується dayjs-nuxt, а @nuxt/image дозволяє ефективно оптимізувати та обробляти зображення. Додатково будуть використані модулі для роботи з шрифтами та інтерактивними компонентами, як-от nuxt-easy-lightbox, що покращує візуальне сприйняття контенту.

Після проведеного аналізу технологій для розробки інтернет-магазину можна зробити висновок, що обраний стек є оптимальним для поставлених завдань. Використання Laravel 12 для серверної частини забезпечує гнучку архітектуру, безпечну роботу з даними та зручне створення REST API. Nuxt 3 дозволяє реалізувати швидкий і адаптивний інтерфейс користувача, підтримує SSR та SPA і добре інтегрується з бекендом. Для зберігання даних MySQL демонструє стабільність і продуктивність, а система пакетів і бібліотек, яку планується використовувати, спрощує роботу з правами доступу, медіа, автентифікацією та іншими функціями. Таким чином, обрані технології дозволяють створити надійну, масштабовану і сучасну систему, яка відповідає вимогам проєкту.

## 2.2 Практична реалізація об’єкта проєктування

Архітектура інтернет-магазину побудована на розподілі системи на декілька незалежних логічних частин: клієнтську (фронтенд), серверну (бекенд) та адміністративну панель. Такий підхід дозволяє чітко розмежувати функціональність, спростити підтримку коду та забезпечити масштабованість проєкту в майбутньому. Усі частини взаємодіють між собою через REST API, що дає змогу розвивати їх окремо та оновлювати без порушення роботи інших компонентів.

Фронтендна частина розроблена на основі Nuxt 3, який працює як SPA-додаток та отримує дані з Laravel API. На рисунку 2.1 показано структуру каталогу, яка містить ключові директорії, відповідальні за маршрутизацію, компоненти інтерфейсу, бізнес-логіку та налаштування застосунку. Така організація забезпечує модульність та простоту розширення.

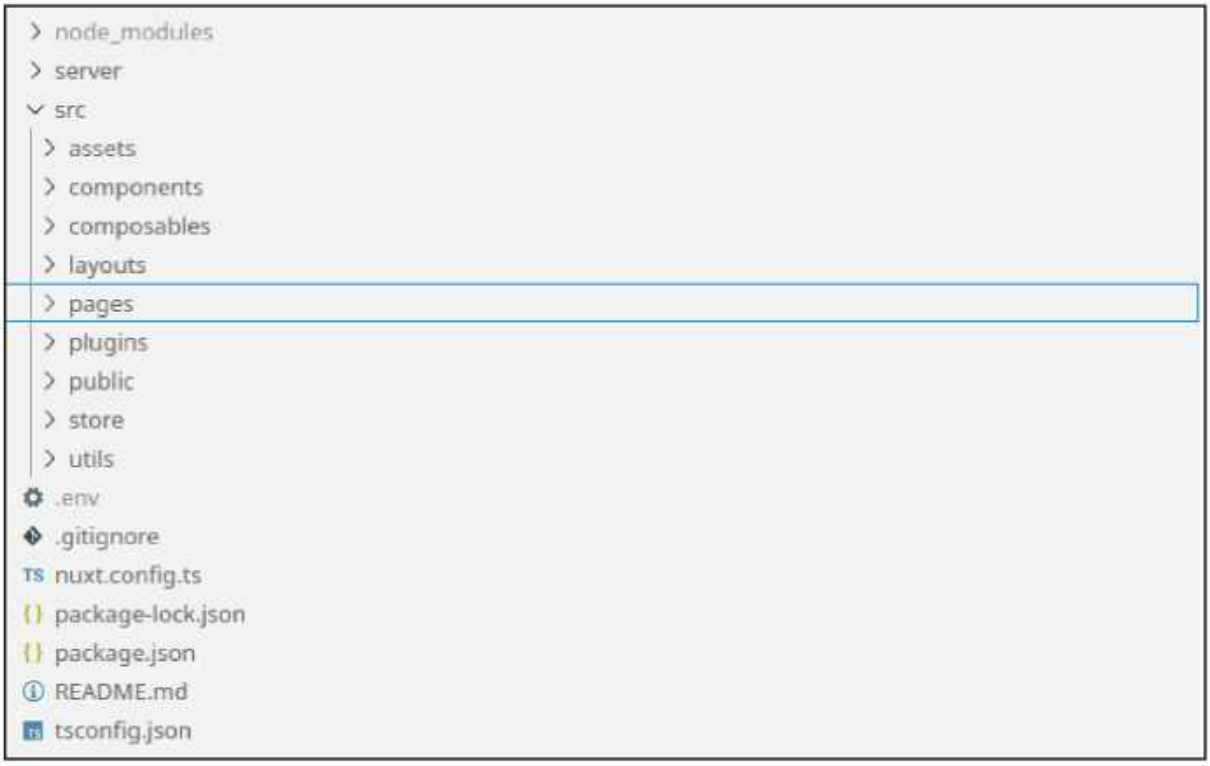


Рисунок 2.1 – Структура фронтендної частини проєкту (Nuxt 3)

Показана структура демонструє чітке розділення відповідальностей:

- «src/pages/» – сторінки, які формують маршрути;
- «src/components/» – багаторазові Vue-компоненти;
- «src/composables/» – локальна логіка роботи з API;
- «src/plugins/» – підключення зовнішніх бібліотек;
- «src/public/» – статичні ресурси.

Така організація спрощує роботу над інтерфейсом та забезпечує масштабованість майбутніх розробок.

Серверна частина реалізована на Laravel, який відповідає за бізнес-логіку, роботу з базою даних, авторизацію, платіжні інтеграції та обробку API-запитів. Структура проєкту розширена додатковими каталогами, що забезпечують більш чисту та зрозумілу архітектуру: Actions, Services, DTO та інші (рис. 2.2).

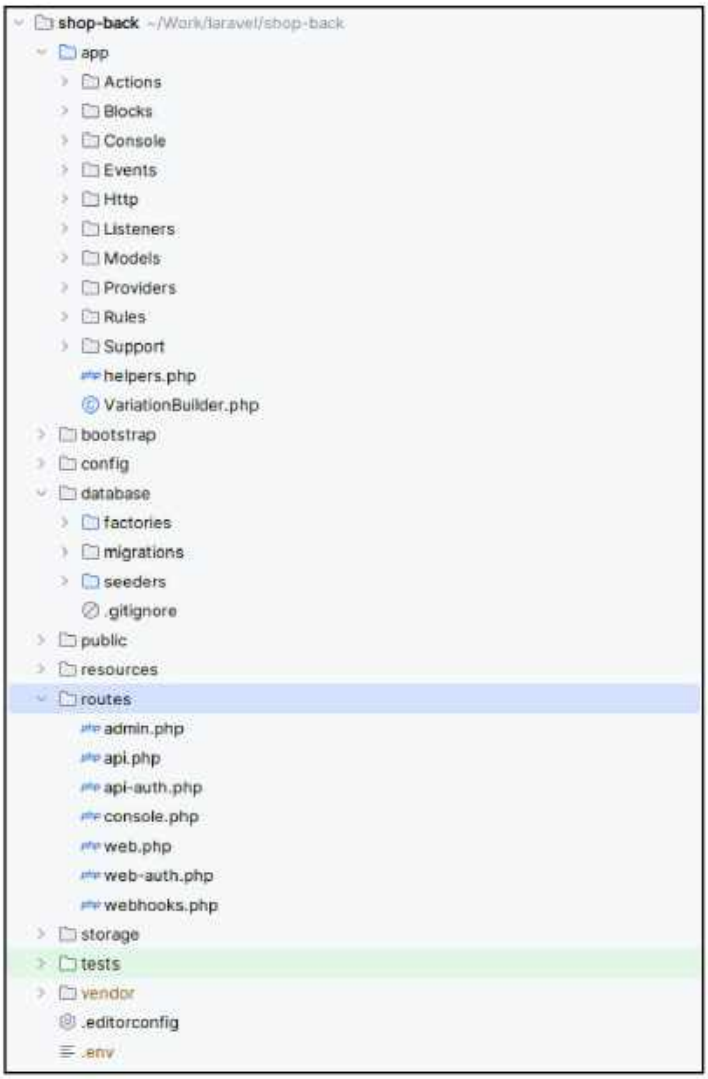


Рисунок 2.2 – Структура серверної частини проєкту (Laravel)

На рисунку видно основні каталоги:

- «app/Models» – сутності бази даних;
- «app/Actions» – окремі бізнес-операції;
- «app/Services» – логіка роботи з API та платіжними системами;

— «routes/api.php» – API маршрути для фронтенду;

— «database/migrations/» – структура таблиць;

— «resources/» – шаблони Blade для адмінпанелі.

Така структура забезпечує зручність підтримки та відповідає принципам SOLID.

Адмінпанель створена на Laravel з використанням Blade-шаблонів та теми AdminLTE 3 [18]. Її структура містить розділи для управління товарами, категоріями, замовленнями та користувачами (рис. 2.3). Усі елементи розміщені у відповідних директоріях, що дозволяє легко орієнтуватися в коді.

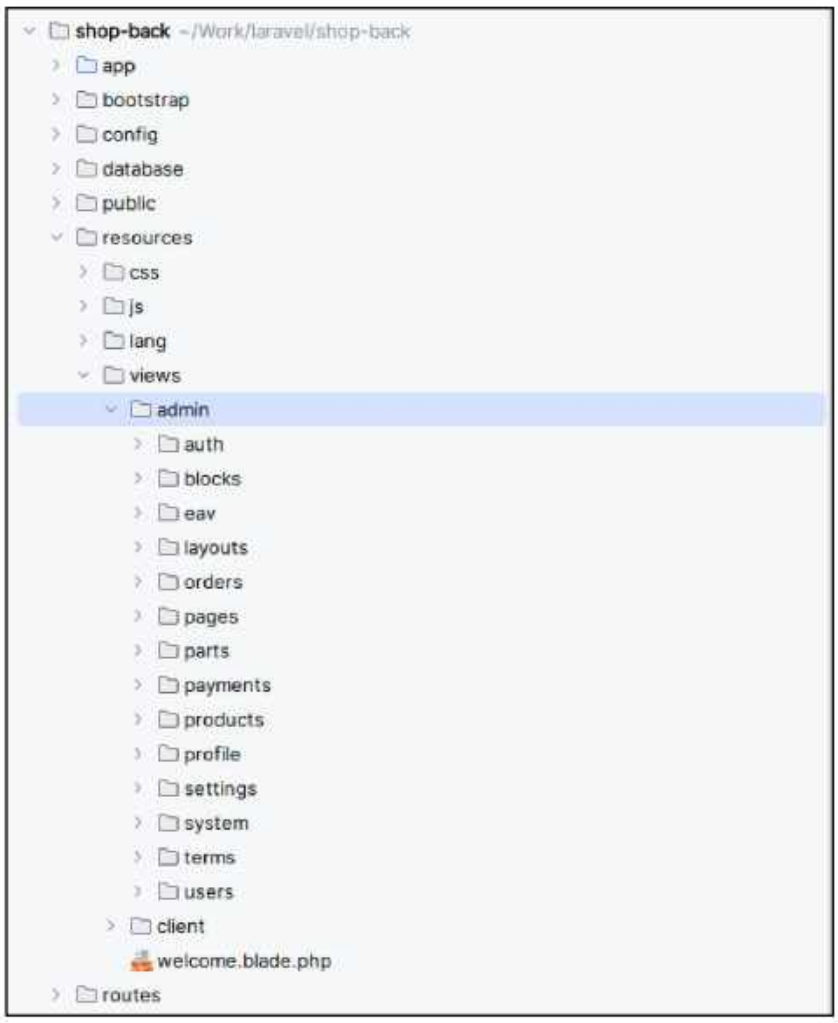


Рисунок 2.3 – Структура адміністративної панелі

Ця структура складається з трьох основних блоків:

– «views/admin/» – шаблони сторінок адмінки;
– «Controllers/Admin/» – логіка керування розділами панелі;
– «public/admin/ – стилі, скрипти та ресурси AdminLTE.

Таке розділення дозволяє ефективно керувати інтерфейсом, спростити оновлення стилів і забезпечує логічне групування файлів.

Таким чином, структура проєкту демонструє чітку та логічну організацію елементів системи. Кожен із компонентів – фронтенд, бекенд і адмінпанель має власну ізольовану архітектуру, але при цьому всі вони взаємодіють між собою за єдиними правилами через REST API. Це забезпечує гнучкість, надійність та можливість подальшого масштабування інтернет-магазину.

Головна сторінка інтернет-магазину, що продемострована на рисунку 2.4, є ключовим елементом взаємодії користувача з системою та виконує роль основної навігаційної точки, з якої відвідувач отримує доступ до каталогу, популярних товарів і ключових функцій сервісу. На етапі проєктування було важливо створити таку структуру сторінки, яка забезпечувала б водночас естетичну привабливість, швидкодію та інтуїтивність використання. Враховуючи це, інтерфейс головної сторінки був розроблений у Nuxt 3, що дало змогу сформувати високу продуктивність та зручну інтеграцію з API, реалізованим на Laravel.

Після першого завантаження користувач бачить навігаційну панель, що містить посилання на каталог, пошук, кошик та особистий кабінет. Це стандартні елементи сучасних інтернет-магазинів, які спрощують початкове ознайомлення з функціоналом платформи. Нижче розміщений слайдер, який реалізований як окремий Vue-компонент і може змінюватися відповідно до маркетингових потреб магазину. Завдяки SSR-рендерингу Nuxt слайдер завантажується без затримок, що позитивно впливає на перше враження користувача.

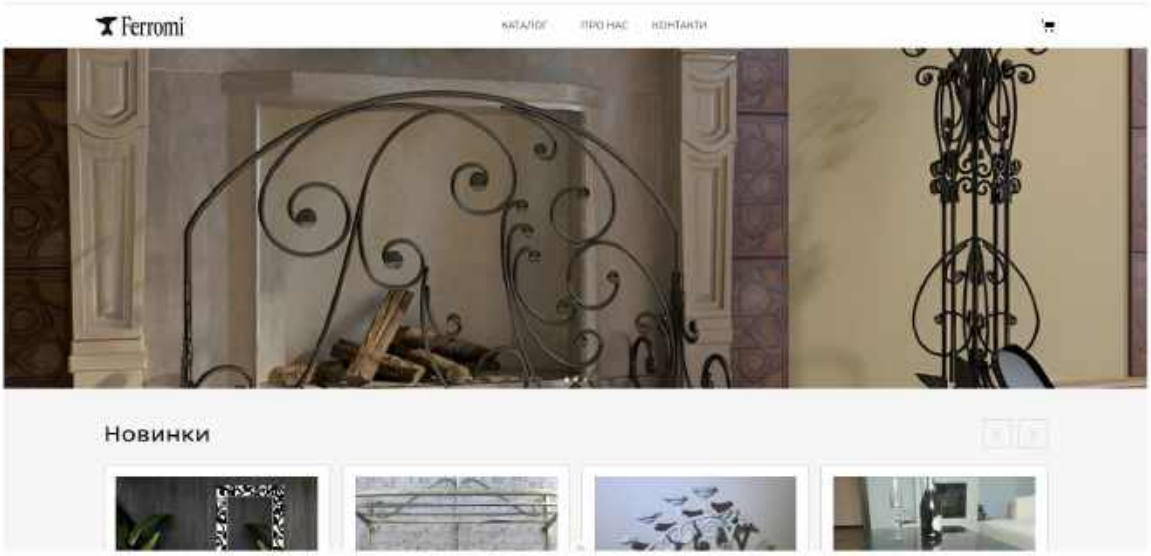


Рисунок 2.4 – Головна сторінка вебзастосунок

У серединній частині сторінки розташовано блоки популярних або рекомендованих товарів. Цей функціонал реалізується через API-ендпойнт, що повертає список товарів із кешу або з бази даних, залежно від налаштувань. Після отримання даних Nuxt динамічно відтворює карточки популярних товарів (рис. 2.5). Кожна карточка містить назву, фото (яке повертається через медіаконверсії), ціну, а також посилання на сторінку товару.

Крім того, на сторінці передбачено інтерактивні елементи навігації, що дозволяють користувачам швидко знаходити потрібні товари та адаптувати відображення контенту під власні потреби. Всі взаємодії реалізовані асинхронно через REST API, що забезпечує плавну роботу інтерфейсу без перезавантаження сторінки. Це дозволяє підвищити швидкодію застосунку та покращити користувацький досвід, особливо на мобільних пристроях та при великих об’ємах даних.

Важливо, що механізм відображення цих товарів працює асинхронно, без перезавантаження сторінки. Це дозволяє пришвидшити взаємодію та не перевантажувати сервер зайвими запитами. Крім того, застосунок враховує можливість відсутності фотографії – у таких випадках відображається

заглушка, згенерована на бекенді, що забезпечує візуальну цілісність інтерфейсу.

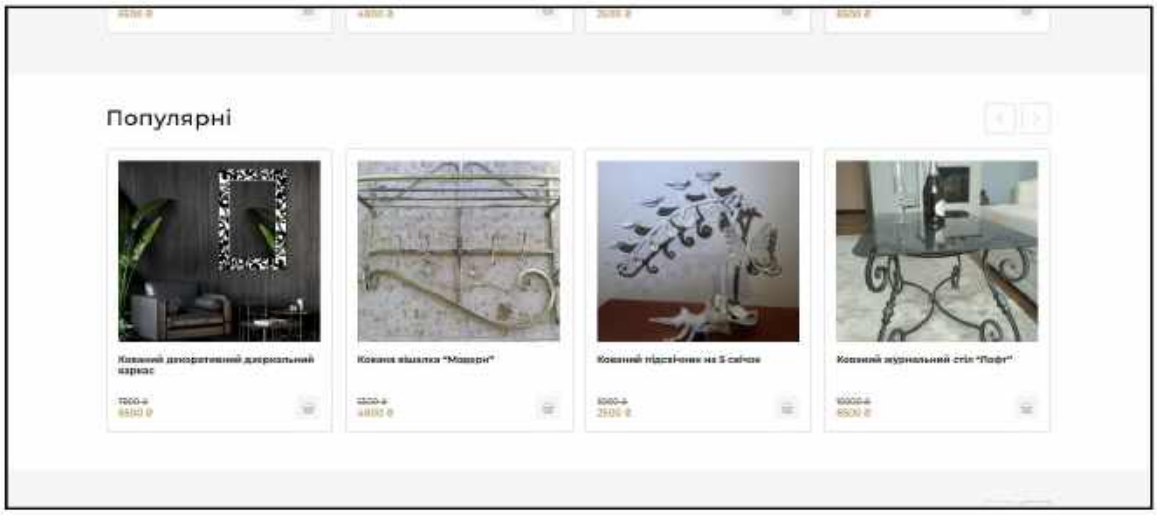


Рисунок 2.5 – Відображення блоку популярних товарів на головній сторінці

Нижня частина сторінки містить службову інформацію, посилання на політику конфіденційності, контакти та інші важливі розділи. Цей блок також винесений у окремий компонент, що спрощує подальше оновлення контенту й підтримку проєкту. Усі статичні елементи кешуються браузером, що забезпечує швидкість повторних завантажень.

Завдяки компонентному підходу у Nuxt 3, зміни у нижньому блоці можна вносити централізовано, що спрощує підтримку і оновлення контенту без втручання в інші частини інтерфейсу. Додатково застосовано «lazy-loading» для графічних елементів, що мінімізує час первинного завантаження сторінки та знижує навантаження на сервер.

Окрему увагу приділено адаптивності сторінки (рис. 2.6). За допомогою SCSS реалізована гнучка система вирівнювання та масштабування компонентів, що дозволяє коректно відображати сторінку на мобільних телефонах, планшетах та широкоформатних моніторах. Це особливо важливо, адже значна частина користувачів здійснює покупки зі смартфонів.

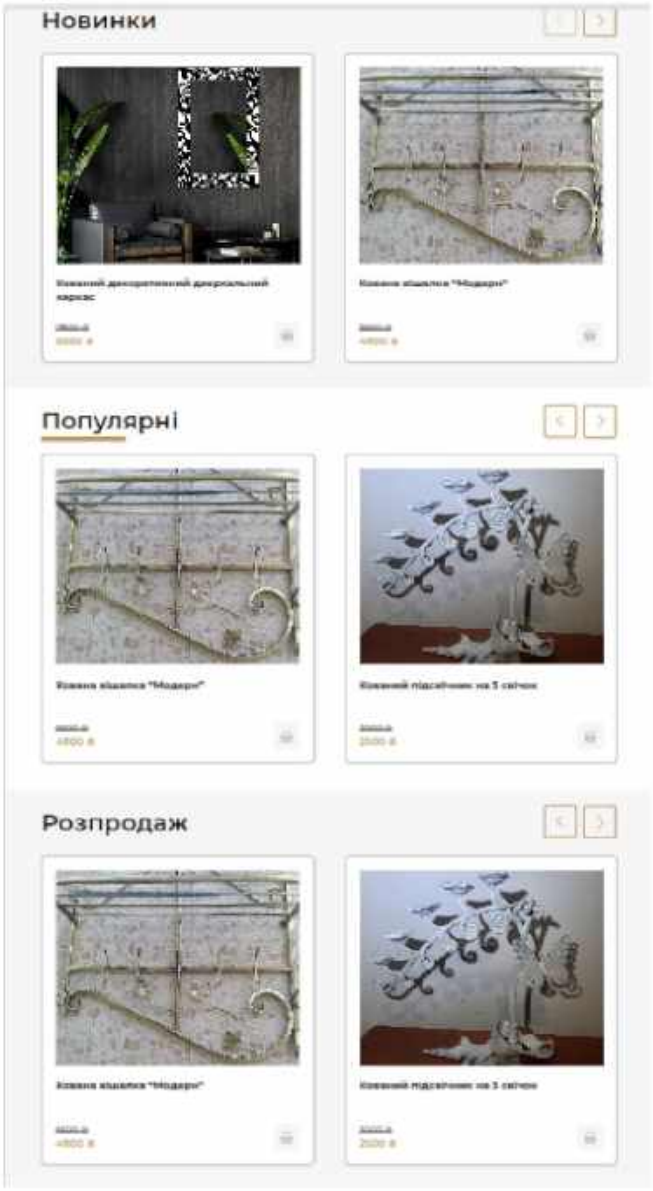


Рисунок 2.6 – Мобільна версія головної сторінки

Загалом головна сторінка забезпечує комфортну початкову взаємодію з платформою та містить основні точки входу до функціональних модулів магазину. Вона побудована на сучасних підходах до проєктування інтерфейсів та ефективно взаємодіє з бекенд-архітектурою, що забезпечує стабільність роботи та добру масштабованість у майбутньому.

Сторінка каталогу є одним із ключових елементів клієнтської частини інтернет-магазину, оскільки вона формує перше уявлення користувача про

структуру товарів та надає зручний спосіб навігації по розділах. На цьому етапі розробки важливо було забезпечити чітку ієрархію категорій, інтуїтивний інтерфейс та логічний перехід до списку товарів кожної категорії.

Після переходу до каталогу користувач бачить перелік усіх доступних категорій (рис. 2.7). Вони представлені у вигляді адаптивних блоків, які містять назву категорії, зображення, та посилання на сторінку з товарами. Такий підхід дозволяє швидко орієнтуватися в асортименті магазину й одразу перейти до потрібного розділу.

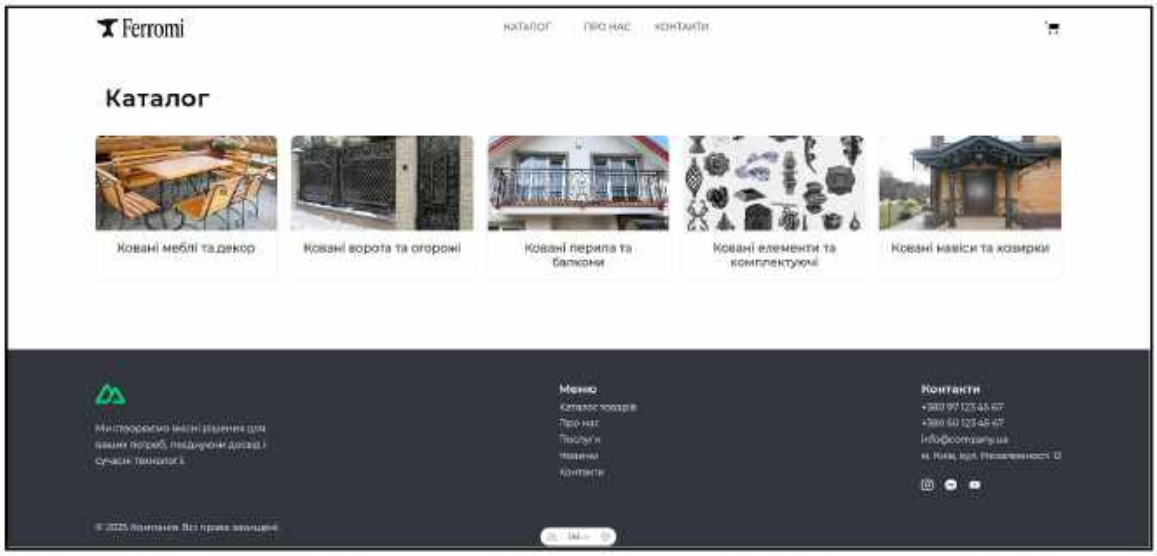


Рисунок 2.7 – Інтерфейс сторінки каталогу з переліком категорій

Після вибору категорії користувач переходить на сторінку з товарами, які належать саме до цього розділу (рис. 2.8). Навігація між категоріями реалізована таким чином, щоб мінімізувати кількість кліків для пошуку потрібного товару.

Для цього переходу використовуються динамічні маршрути Nuxt (/catalog/[slug]), які формуються через API Laravel. Це забезпечує стабільний процес завантаження даних та дозволяє зручно будувати SEO-оптимізовану структуру сайту.

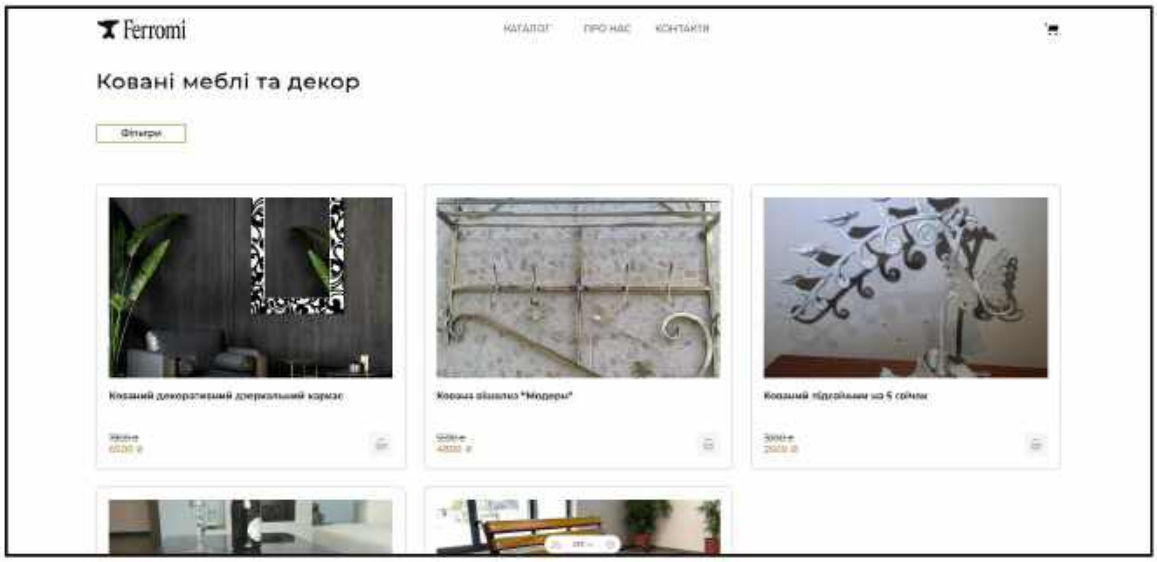


Рисунок 2.8 – Перехід зі сторінки каталогу на сторінку товарів категорії

Таким чином, сторінка каталогу виступає не лише візуальним елементом, а й логічною точкою входу до товарної частини магазину. Вона визначає перше враження користувача та формує зручний маршрут взаємодії з інтернет-магазином. На лістингу 2.1 наведено фрагмент серверного коду, що відповідає за отримання, групування та передачу варіацій товару до API.

Лістинг 2.1 – Приклад бекенд-логіки отримання варіацій товарів.

```
public function variations(Request $request)
{
    $variations = ProductVariation::query()
        ->with([
            'media.model',
            'product.media.model',
            'product.category',
            'properties.media',
            'properties.attribute',
        ])
        ->byGropedType($request->groped_type)
        ->filterable(
            $request->all(),
            [
                'status' => ProductVariation::STATUS_PUBLISHED,
                'sort_reality' => 1,
            ]
        )
```

```
        ->paginate();

    return ProductVariationListResource::collection($variations);
}
```

Кінець лістингу 2.1

На рисунку 2.9 зображено сторінку одного товару, яка є ключовим елементом взаємодії користувача з інтернет-магазином, оскільки саме на ній відбувається остаточне формування рішення про покупку. Тому під час реалізації важливо було забезпечити максимально зрозумілий інтерфейс, швидке завантаження даних та логічне розміщення інформації.

На сторінку товару користувач потрапляє зі списку товарів певної категорії або через пошук. Після завантаження він бачить головне зображення товару, заголовок, актуальну ціну та доступні варіації (наприклад: колір, розмір або інші вибіркові модифікації). Всі ці дані отримуються з API Laravel через окремий ендпоінт, який повертає структуру товару разом із медіафайлами, характеристиками та наявністю.

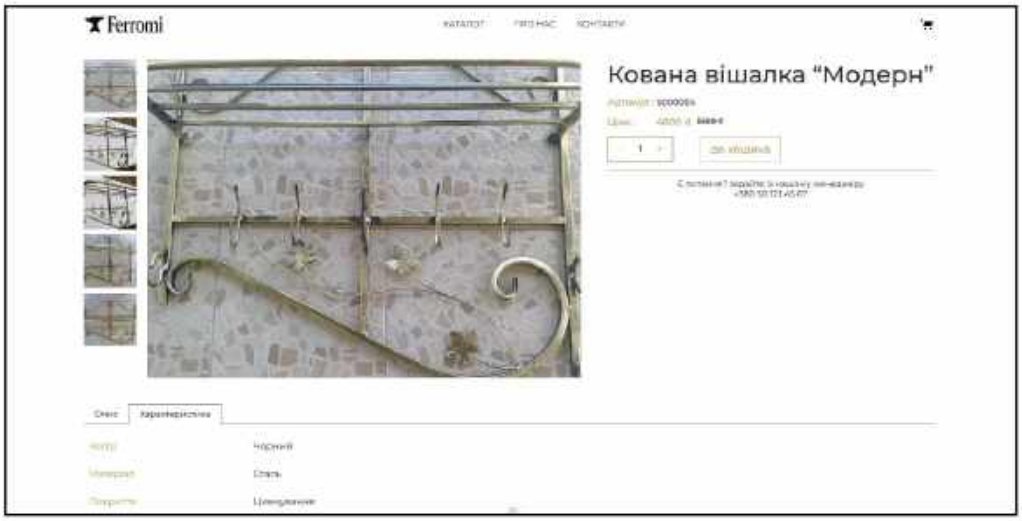


Рисунок 2.9 – Сторінка одного товару

Нижче основної інформації розташована кнопка «до кошика». При натисканні формується запит до API, де передаються дані про вибрану варіацію. На стороні Laravel формується позиція кошика, і далі оновлений стан повертається на фронтенд. Це дозволяє користувачу одразу побачити, що товар додано, і продовжити покупки.

Також на сторінці відображаються додаткові характеристики товару: матеріали, розміри, технічні властивості або інші дані, залежно від типу товару. На рисунку 2.10 відображається відповідний блок, винесений окремо, щоб не перевантажувати головну частину сторінки.


Рисунок 2.10 – Характеристика товару

Таким чином, сторінка одного товару виконує не лише інформаційну роль, а й формує основний користувацький шлях у процесі покупки. Завдяки інтеграції Nuxt 3 та Laravel API вдалося забезпечити швидке завантаження, динамічну зміну даних і зручний інтерфейс для покупця, що позитивно впливає на загальний досвід користувача та підвищує ефективність роботи інтернет-магазину.

Сторінка кошика дозволяє користувачеві переглядати всі вибрані товари перед оформленням замовлення. Тут відображаються назви продуктів, їхні зображення, ціни та кількість, яку можна змінити безпосередньо у кошику.

Також відображається загальна сума замовлення та можливість видаляти непотрібні позиції, що забезпечує користувачу повний контроль над своїм замовленням. Для переходу до наступного етапу – оформлення замовлення – передбачена кнопка, що перенаправляє користувача на відповідну сторінку. На рисунку 2.11 наведено приклад інтерфейсу кошика, де показано розташування товарів, доступні дії та загальну суму замовлення.


Рисунок 2.11 – Сторінка кошика покупця

Сторінка оформлення замовлення є завершальним етапом процесу покупки, де користувач вводить усі необхідні дані для обробки замовлення та оплати. Тут передбачено заповнення контактної інформації, таких як ім'я, прізвище, телефон та електронна пошта, а також адреси доставки з уточненням вулиці та міста. Користувач має можливість обрати зручний метод оплати, після чого підтвердити замовлення. На рисунку 2.12 показано приклад інтерфейсу сторінки оформлення замовлення, де користувач може зручно вводити дані та переглядати підсумок покупки перед підтвердженням.

Також реалізовано динамічне відображення підсумкової суми замовлення з урахуванням обраних товарів та кількості, що допомагає користувачу чітко бачити фінальний результат перед підтвердженням покупки.

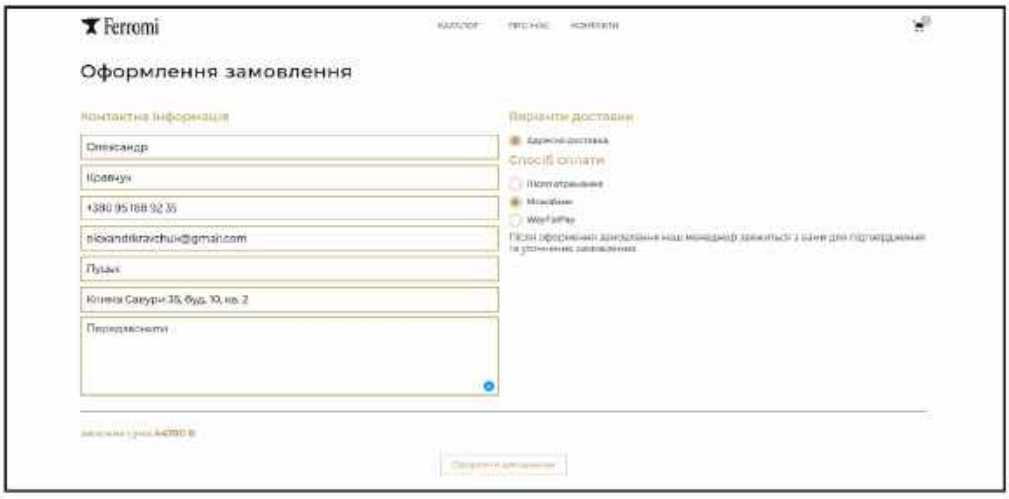


Рисунок 2.12 – Сторінка оформлення замовлення

Після підтвердження даних замовлення користувач переходить на сторінку оплати, де ініціалізується платіжна сесія. Система взаємодіє з платіжними сервісами для швидкої та безпечної операції. Інтерфейс відображає суму замовлення, обраний спосіб оплати та реквізити транзакції. На першому етапі доступний Monobank із платіжним посиланням та кнопкою підтвердження (рис. 2.13).

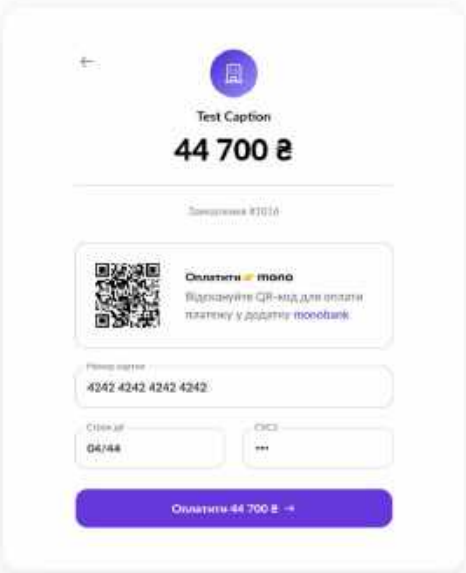


Рисунок 2.13 – Сторінка оплати через Monobank

Оплата формується через створення платіжної сесії на бекенді. У лістингу 2.2 показано приклад для Monobank: формуються дані платежу, відправляється запит до API, зберігається ідентифікатор транзакції, і користувач перенаправляється на сторінку Monobank для завершення оплати.

Лістинг 2.2 – Створення платіжної сесії Monobank на стороні бекенду

```
public function pay(Payment $payment)
{
    $order = $payment->model;
    $this->initApi();
    $data = [
        'amount' => intval($payment->amount * 100),
        'ccy' => 980, // UAH
        'merchantPaymInfo' => [
            'reference' => $order->number,
            'destination' => "Замовлення #{$order->number}",
        ],
        'webHookUrl' => route('webhooks.payment.notify', ['monobank']),
                  'redirectUrl'  =>  route('payment.info',  ['progress',
'order_number' => $order->number]),
    ];
    $headers = ['X-Token' => $this->paymentToken];
    $response = Http::withHeaders($headers)->post($this->apiUrl, $data);
    $responseData = $response->json();
                       if     (!empty($responseData['invoiceId'])     &&
!empty($responseData['pageUrl'])) {
        $payment->extern_id = $responseData['invoiceId'];
        $payment->save();
        return $responseData['pageUrl'];
    }

    return false;
}
```

Кінець лістингу 2.2

Поданий фрагмент коду демонструє створення платіжної сесії Monobank на стороні бекенду.

Іншим доступним варіантом оплати є сервіс WayForPay, який підтримує оплату картками, електронними гаманцями та різними валютами. На рисунку 2.14 наведено приклад інтерфейсу WayForPay, де користувач може

обрати зручний метод оплати, переглянути підсумкову суму замовлення та підтвердити транзакцію. Такий підхід дозволяє гнучко підлаштовувати процес оплати під потреби користувача та забезпечує високий рівень безпеки завдяки механізмам автентифікації та шифрування даних.

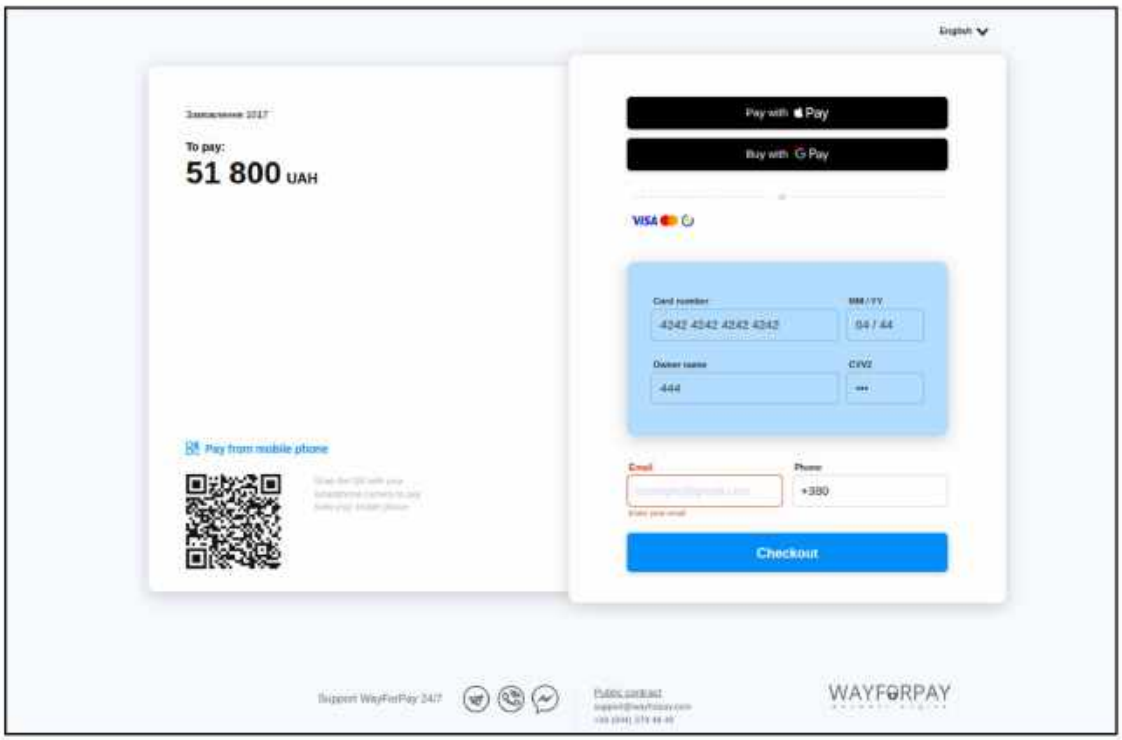


Рисунок 2.14 – Сторінка оплати через WayForPay

Оплата здійснюється через створення платіжної сесії на стороні бекенду. У лістингу 2.3 наведено приклад для WayForPay: формуються основні дані платежу, такі як сума, валюта, інформація про замовлення та callback URL для отримання повідомлень про статус транзакції. Після відправки запиту до API користувач автоматично перенаправляється на сторінку WayForPay для завершення оплати. Такий підхід забезпечує контроль за процесом платежу з боку сервера, дозволяє відстежувати успішність транзакцій і гарантує безпечну обробку фінансових даних, зберігаючи високий рівень захисту користувацької інформації.

Лістинг 2.3 – Створення платіжної сесії WayForPay на стороні бекенду

```
public function pay(Payment $payment)
{
    $order = $payment->model;
    $this->initApi();
    $time = time();
    $amount = round($payment->amount, 2);
    $orderNumber = $order->number . ':' . Str::uuid();
    $productName = 'Замовлення ' . $order->number;
                                $signature       =       hash_hmac("md5",
"{$this->account};{$this->domain_merchant};{$orderNumber};{$time};{$amou
nt};{$payment->currency_code};{$productName};1;{$amount}",
        $this->secret_key
    );
    $response = Http::post(self::API_URL, [
        'merchantAccount' => $this->account,
        'merchantDomainName' => $this->domain_merchant,
        'orderReference' => $orderNumber,
        'orderDate' => $time,
        'amount' => $amount,
        'currency' => $payment->currency_code,
        'productName' => [$productName],
        'productCount' => [1],
        'productPrice' => [$amount],
        'merchantSignature' => $signature,
         'returnUrl' => route('payment.info', ['progress', 'order_number'
=> $order->number]),
                                                        'serviceUrl'      =>
"https://model-dassie-comic.ngrok-free.app/webhooks/payment/notify/wayfo
rpay",
    ]);
    $payment->extern_id = $orderNumber;
    $payment->save();
    return $response->json()['url'];
}
```

Кінець лістингу 2.3

Завдяки використанню двох платіжних систем забезпечується вибір для користувача та надійність проведення платежів. Інтерфейс адаптивний і зручний як на десктопних, так і на мобільних пристроях, що робить процес оплати максимально комфортним і безпечним.

# РОЗДІЛ 3
# ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТИВНОСТІ РОЗРОБЛЕНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

## 3.1 Методика проведення дослідження

Метою експериментального дослідження є перевірка того, наскільки створена інформаційна система відповідає поставленим вимогам та чи може вона стабільно працювати в умовах реального використання. У межах цього розділу оцінюється коректність роботи REST API, поведінка системи при оформленні замовлень, а також правильність взаємодії з платіжними сервісами Monobank і WayForPay. Окрему увагу зосереджено на роботі механізмів webhook, оскільки від них залежить оновлення статусів платежів і замовлень.

Також важливо з'ясувати, як система поводиться під різним навантаженням, чи немає затримок у відповідях, та наскільки точно обробляються дані, що передаються через API. Дослідження покликане продемонструвати, чи здатний розроблений функціонал забезпечити безперебійну обробку замовлень, коректну оплату й надійне логування кожного етапу взаємодії. Результати цього тестування стануть основою для подальшого аналізу та підтвердження ефективності запропонованих рішень.

Тестування функціоналу інтернет-магазину проводилося в локальному середовищі, розгорнутому за допомогою Docker-контейнерів через Laradock [19]. Такий підхід дозволяє отримати стабільну та ізольовану інфраструктуру, де всі сервіси – вебсервер, база даних, кеш і PHP-процес – працюють у прогнозованих умовах. Основою є контейнер із PHP 8.2, що забезпечує сумісність з вимогами Laravel 12, а також окремий контейнер з MySQL, у якому зберігаються всі дані магазину.

Конфігурація проєкту здійснювалася через файл .env, де були вказані параметри підключення до бази даних, локальний домен shop-back.test, налаштування логування та режим роботи застосунку. Завдяки цьому

середовище максимально наближене до реального, що дозволяє оцінити поведінку системи у виробничих умовах.

Для моніторингу внутрішніх процесів під час тестування застосовувався стандартний логер Laravel, а також пакет «ka4ivan/laravel-api-debugger», який особливо зручний для виявлення N+1 проблем та надто довгих SQL-запитів (рис. 3.1). Цей інструмент дозволив швидко визначати місця, де ORM формує зайві запити до бази даних, а також аналізувати тривалість кожної операції. Завдяки цьому вдалося оптимізувати моделі та запити ще на етапі тестування, що позитивно вплинуло на продуктивність системи. Усі перевірки проводилися в однаковому середовищі, що забезпечило достовірність отриманих даних і мінімізувало вплив сторонніх факторів.


Рисунок 3.1 – Представлення даних API Debugger у форматі JSON

Для проведення експериментального дослідження було застосовано набір інструментів, які дозволили перевірити роботу системи на різних рівнях, від окремих API-запитів до внутрішніх механізмів застосунку. Основним інструментом став Postman, за допомогою якого виконувалися всі запити до REST API (рис. 3.2). Це дало можливість перевірити правильність маршрутів,

відповідність структури JSON-відповідей очікуваному формату, а також поведінку системи при передачі некоректних даних. Postman також дозволив зручно повторювати однакові сценарії, що особливо корисно під час тестування оплати й webhook-викликів.

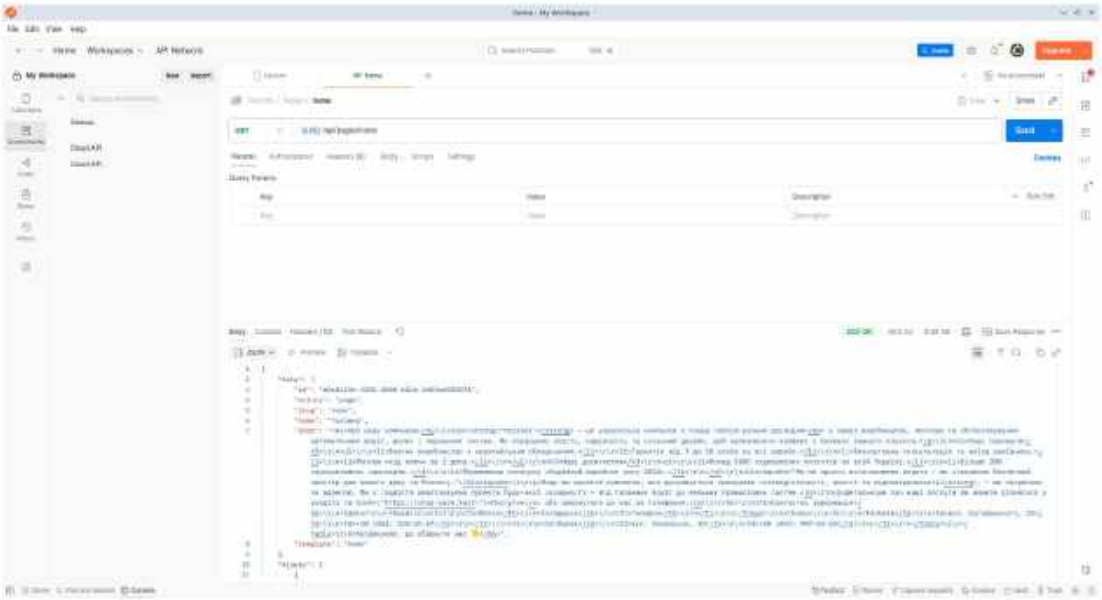

Рисунок 3.2 – Графічний інтерфейс інструменту Postman

Для контролю внутрішніх процесів Laravel використовувався пакет ka4ivan/laravel-logger, який надає розширені можливості перегляду інформації про виконані запити, SQL-операції, помилки та інші важливі події у застосунку (рис. 3.3). Інструмент дозволяє відстежувати всі запити до бази даних, перевіряти виконання бізнес-логіки та контролювати послідовність обробки даних у різних модулях системи. Завдяки цьому вдалося швидко виявляти проблемні місця, аналізувати фактичні дані, що надходять від платіжних систем, та перевіряти коректність обробки транзакцій. Особливо ефективним пакет виявився під час тестування callback-повідомлень від Monobank і WayForPay, оскільки дозволяє фіксувати помилки та відтворювати сценарії повторних викликів API, що підвищує надійність роботи інтернет-магазину.

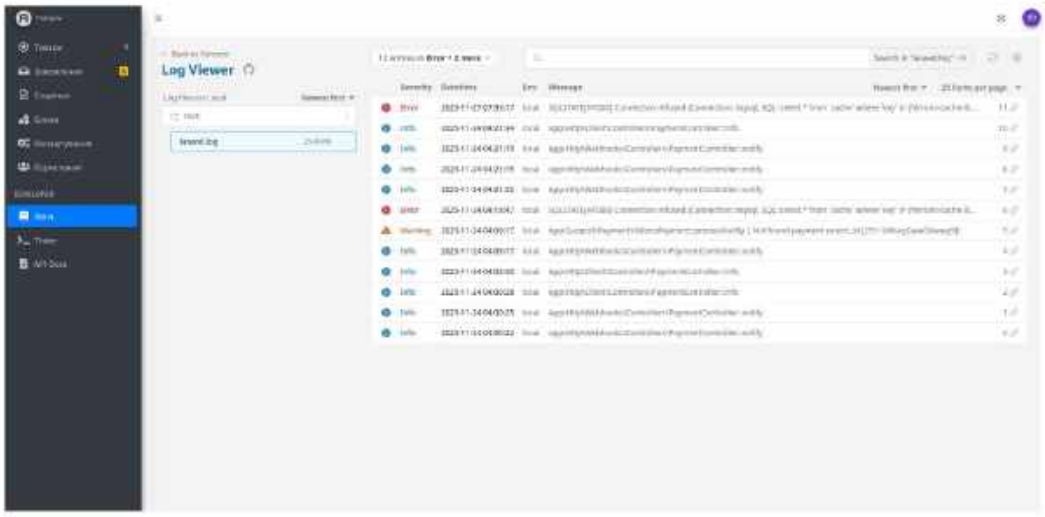


Рисунок 3.3 – Вигляд сторінки логування

Окрім цього, під час дослідження застосовувався Spatie Web Tinker, що дозволяє виконувати PHP-код у робочому середовищі застосунку без створення окремих скриптів. За допомогою цього інструмента було зручно перевіряти роботу окремих сервісів, тестувати функції драйверів та аналізувати поведінку моделей. У сукупності такі інструменти забезпечили комплексний підхід до перевірки системи та дозволили отримати максимально точні результати

Під час дослідження велика увага приділялася перевірці REST API, оскільки саме через нього взаємодіє клієнтська частина та адміністративна панель з сервером. Тестування виконувалося послідовно, починаючи з найпростіших запитів і поступово переходячи до більш складних сценаріїв. Основна увага зосереджувалася на правильності роботи маршрутів POST, GET, PUT та DELETE, адже ці операції відповідають за створення, отримання, оновлення та видалення даних.

Під час перевірки кожного маршруту контролювався статус-код відповіді, щоб переконатися, що сервер коректно реагує на запити. Наприклад, для успішних операцій очікувалися коди 200 або 201, а у випадку помилкових запитів мала повертатися відповідна помилка (400, 401, 403, 404) (рис. 3.4). Це

дозволило переконатися, що система правильно відпрацьовує стандартні ситуації та коректно реагує на не валідні дані.

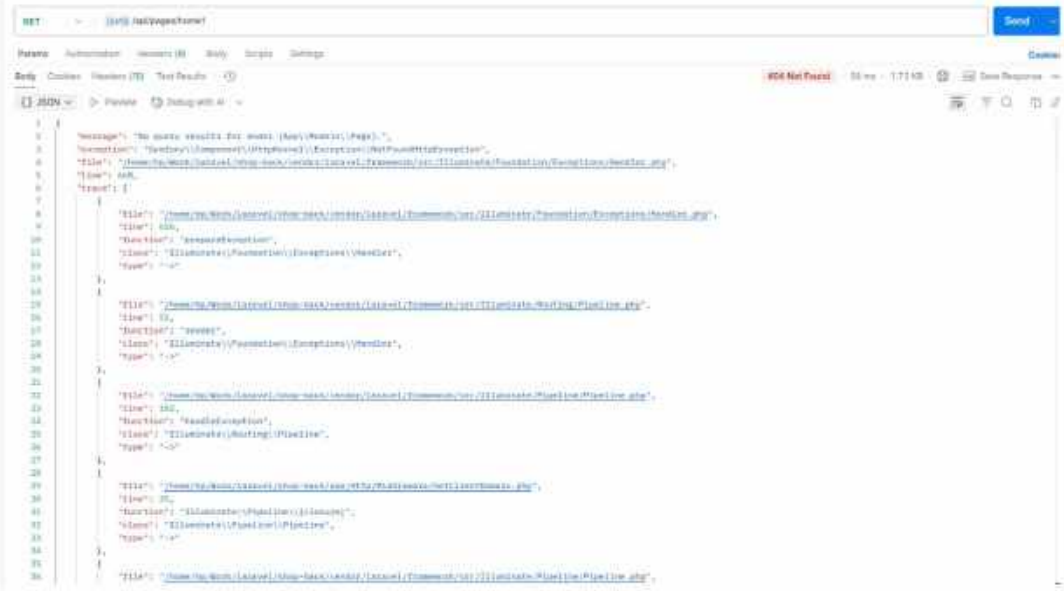

Рисунок 3.4 – Приклад відповіді з кодом 404 у випадку, коли запис не знайдено в базі даних

Окрему увагу приділено перевірці структури JSON-відповідей. Перевірялося, щоб усі поля відповідали заявленому формату, а сервер повертав тільки ті дані, які необхідні клієнту. Також тестувалися сценарії з навмисно некоректним введенням, що дозволило оцінити якість обробки помилок. Завдяки такому підходу вдалося переконатися, що API працює стабільно, а логіка валідації та обробки запитів реалізована коректно.

Після перевірки роботи API окремо тестувалися ключові модулі інтернет-магазину, такі як управління товарами, категоріями, користувачами та замовленнями. Основний акцент робився на коректній роботі CRUD-операцій, оскільки саме вони забезпечують можливість повноцінного адміністрування контенту. Для кожного модуля послідовно перевірялося створення запису, його редагування, перегляд і видалення. Це дозволило переконатися, що база даних

правильно обробляє зміни, а інтерфейс адміністративної панелі коректно відображає оновлену інформацію.

Окремо тестувалася серверна валідація, щоб уникнути ситуацій, коли в систему можуть потрапити некоректні або неповні дані (рис. 3.5). Для цього вручну вводилися неправильні значення, пропускалися обов'язкові поля та перевірялися реакції системи. У випадку з модулями товарів і категорій додатково оцінювалася робота пов'язаних даних – наприклад, перевірялася коректність прив'язки товарів до категорій та їх відображення у списках.

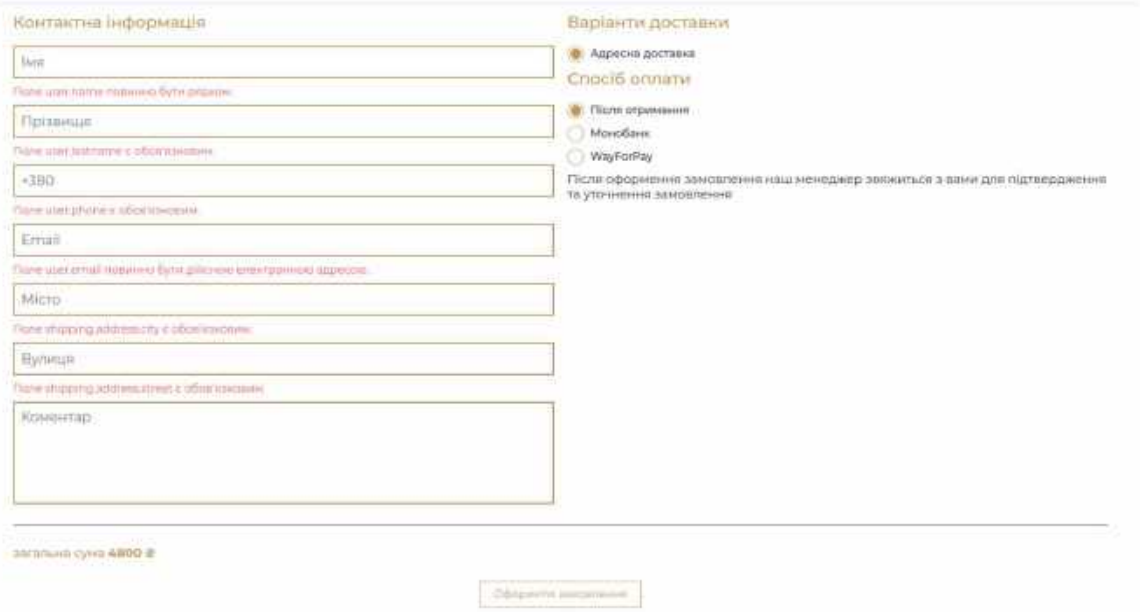


Рисунок 3.5 – Демонстрація результату валідації при відсутності обов'язкових даних

Модуль управління замовленнями проходив розширене тестування, оскільки він тісно пов'язаний із платіжними сервісами та взаємодією з користувачами.

Під час перевірки взаємодії з платіжними сервісами використовувалися тестові токени, надані Monobank та WayForPay. Завдяки цьому вдалося безпечно виконати повний цикл оплати: сформувати платіжний запит, створити

тестову транзакцію та отримати callback-повідомлення (рис. 3.6). Такий підхід дозволив переконатися, що логіка обробки платежів працює коректно і система правильно реагує на успішні та неуспішні операції.

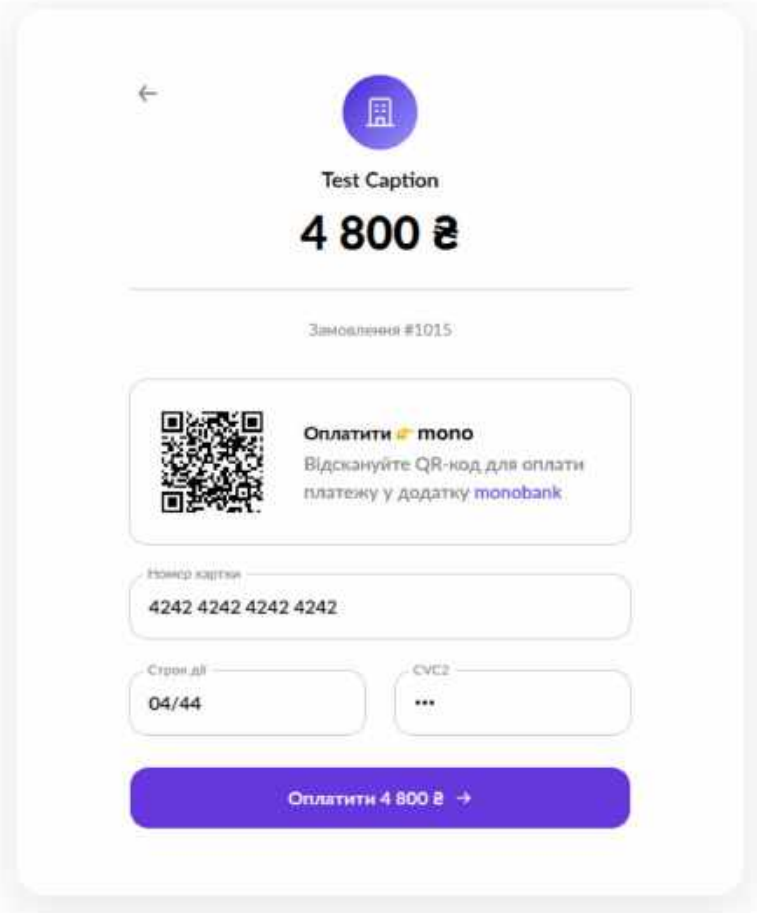


Рисунок 3.6 – Демонстрація формування тестової оплати замовлення через Monobank

У межах проведеного дослідження була сформована цілісна методика тестування функціоналу системи, що включала використання спеціалізованих інструментів та єдиних умов запуску. Основними засобами взаємодії з API слугував Postman, який забезпечував зручне формування HTTP-запитів та аналіз отриманих відповідей. Для контролю внутрішніх процесів Laravel було застосовано вбудоване логування та пакет «ka4ivan/laravel-api-debugger», що

дозволив відстежувати N+1-запити, тривалі SQL-операції та інші потенційні проблеми продуктивності.

Завдяки тестовим токенам платіжних систем було перевірено коректність роботи механізмів оплати – створення платежу, отримання callback-повідомлень, обробка статусів та реагування системи на різні сценарії транзакцій. Усі тести виконувалися в стандартизованих умовах, що гарантує об'єктивність результатів і дозволяє зробити висновок про стабільну та передбачувану роботу реалізованого функціоналу.

### 3.2 Обробка та аналіз отриманих результатів

Взаємодія з платіжними системами Monobank та WayForPay здійснюється за допомогою Webhook – механізму, коли платіжний сервіс автоматично надсилає інформацію на сервер про зміну статусу транзакції. Це дозволяє отримувати актуальні дані без постійного опитування API і підвищує продуктивність системи.

Для перевірки коректності роботи інтегрованих платіжних систем Monobank та WayForPay проведено серію тестових оплат. Основною метою експерименту було оцінити стабільність роботи API, швидкість обробки запитів та правильність відображення статусів транзакцій у системі.

Як видно з таблиці 3.1, обидві платіжні системи продемонстрували високу стабільність роботи. Monobank зареєстрував один випадок неуспішної оплати, що, ймовірно, пов'язано з умовами тестування або некоректними даними картки, тоді як WayForPay обробила всі транзакції успішно. Середній час обробки запиту склав 1,2 секунди для Monobank та 1,5 секунди для WayForPay, що свідчить про швидку та ефективну взаємодію з API обох сервісів.

Таблиця 3.1 – Результати тестування платіжних систем Monobank та WayForPay.

| Платіжна система | Кількість тестів | Успішні оплати | Неуспішні оплати | Середній час обробки запиту |
|---|---|---|---|---|
| Monobank | 20 | 19 | 1 | 1.2 секунд |
| WayForPay | 20 | 20 | 0 | 1.5 секунд |

Результати експерименту показують, що розроблена система коректно обробляє платежі, оновлює статуси замовлень та забезпечує своєчасне надходження інформації про транзакції. Крім того, дані демонструють ефективність використання Webhook для автоматичного отримання статусів платежів без додаткових опитувань API, що значно підвищує продуктивність системи.

Таким чином, на підставі результатів, наведених у таблиці 3.1, можна зробити висновок про стабільність і надійність інтеграції платіжних систем у розроблений інтернет-магазин, що підтверджує доцільність обраного підходу та готовність системи до використання у реальних умовах.

Під час тестування перевіряються основні аспекти:

— надходження подій з різними статусами платежів (successful, failed, pending) і правильне оновлення статусів замовлень у базі даних;

— всі події Webhook зберігаються для подальшого аналізу та відновлення стану системи, а також для оцінки часу обробки та стабільності під навантаженням;

— у разі тимчасової недоступності сервера перевіряється, що повторна відправка подій обробляється коректно і не створює дублікати замовлень;

— тестові сценарії Webhook можуть відтворюватися за допомогою Postman або скриптів, що імітують різні статуси транзакцій без реальних платежів.

Тестування через Webhook забезпечує надійну обробку статусів платежів, коректне оновлення замовлень і захист системи від підроблених запитів, що підвищує стабільність і готовність інтернет-магазину до реальної експлуатації.

Результати проведеного експериментального дослідження та розробки інтернет-магазину можуть бути використані для практичної реалізації онлайн-продажів у реальних умовах. Основні особливості впровадження:

— для роботи системи необхідне середовище з Laravel 12, база даних MySQL, налаштовані сервіси Redis і Webhook для отримання статусів платежів;

— підключення до платіжних систем потребує наявності токенів API і належної конфігурації платіжних шлюзів;

— для тестування і перевірки роботи системи можна використовувати тестові картки і тестові події Webhook, що дозволяє оцінювати стабільність і швидкість обробки платежів без фінансових витрат.

Практична цінність результатів:

— впроваджена інтеграція забезпечує швидку і безпечну обробку платежів, своєчасне оновлення статусів замовлень і зменшення навантаження на сервер завдяки Webhook;

— розроблені сервіси для роботи з платежами і модульна структура коду дозволяють легко масштабувати систему та підключати нові платіжні сервіси без зміни наявної логіки;

— автоматизоване логування та обробка помилок забезпечують надійність і прозорість роботи системи, що підвищує ефективність адміністрування.

Таким чином, результати кваліфікаційної роботи можуть бути безпосередньо впроваджені у комерційні проєкти, забезпечуючи стабільну, безпечну та масштабовану платформу для електронної комерції.

# ВИСНОВКИ

У кваліфікаційній роботі було поставлено за мету розробити інтернет-магазин із повноцінною інтеграцією платіжних систем Monobank та WayForPay за допомогою фреймворку Laravel. У ході дослідження виконано всі поставлені завдання, що дозволило комплексно реалізувати вебсистему, яка відповідає сучасним вимогам електронної комерції.

На першому етапі проєкту здійснено аналіз вимог до інтернет-магазинів, що включав вивчення особливостей побудови архітектури, безпеки онлайн-платежів, вимог до користувацького інтерфейсу та функціоналу адміністративної панелі. Це дало змогу сформувати чіткі технічні та функціональні вимоги до майбутньої системи та визначити обґрунтований вибір технологій.

Для реалізації серверної частини було використано Laravel 12, що забезпечило гнучку структуру проєкту, розділення бізнес-логіки та стабільну роботу REST API. Архітектура побудована з використанням сервісного шару та пакету laravel-actions, що дозволило підвищити модульність, читабельність та тестованість програмного коду. Розроблено та впроваджено структуру бази даних з необхідними зв'язками між товарами, категоріями, замовленнями та користувачами.

Створено повноцінну адміністративну панель, яка забезпечує можливість керування товарами, категоріями, користувачами, замовленнями та іншими елементами системи. Реалізація CRUD-функціоналу дала змогу забезпечити адміністрування на рівні реального комерційного застосунку.

Клієнтська частина створена на основі Nuxt 3 та підтримує SSR і SPA режими, що забезпечує швидкість завантаження сторінок, адаптивний інтерфейс та зручність взаємодії користувача з сайтом. Структура фронтенду орієнтована на масштабованість і подальше розширення функціоналу.

Одним із ключових завдань була інтеграція платіжних систем Monobank та WayForPay. У роботі реалізовано взаємодію через API та механізм Webhook, що забезпечило автоматичне оновлення статусів транзакцій, коректне створення та завершення замовлень, а також захист від дублювання подій. Ретельно опрацьовано обробку помилок, валідацію запитів і логування платіжних подій, що гарантує надійність і безпечність фінансових операцій.

Для забезпечення якості та стабільності роботи проведено комплексне тестування системи. Використано Postman для перевірки REST API, Laravel Pint для аналізу стилю коду, Pest для модульного тестування та додаткові сценарії для перевірки роботи Webhook. Проведене експериментальне дослідження підтвердило правильність інтеграції, стабільність обробки платежів, відсутність критичних помилок та готовність системи до реального використання.

Таким чином, у результаті виконаної роботи створено сучасний інтернет-магазин, який відповідає вимогам щодо безпеки, продуктивності та зручності користування. Реалізована система демонструє правильність обраної архітектури, технологічного стека та підходів до інтеграції платіжних сервісів. Розроблений вебзастосунок може бути використаний як основа для розширення функціоналу, впровадження додаткових сервісів або запуску комерційного продукту в реальному середовищі.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Киричук О. О. Суринович О. М. Електронні оплати у системі сучасної економіки. Тези доповідей X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025) (23-24 травня 2025 року). Луцьк: ЛНТУ, 2025. С. 336-341, https://itonv.lntu.edu.ua/files/2025/zbirnyk_itonv_2025.pdf.

2. CMS – що це таке і як працює, види та приклади | HOSTiQ Wiki. URL: https://hostiq.ua/wiki/ukr/cms/ (дата звернення: 13.09.2025).

3. Laravel – PHP Фреймворк. Laravel – The PHP Framework : вебсайт. URL: https://laravel.com/ (дата звернення: 15.09.2025).

4. Acquiring. Mono API. URL: https://api.monobank.ua/docs/acquiring.html (дата звернення: 21.09.2025).

5. Документація. Wayforpay API. URL: https://wiki.wayforpay.com/ (дата звернення: 29.09.2025).

6. Документація APIs. LiqPay. URL: https://www.liqpay.ua/doc/api (дата звернення: 01.10.2025).

7. Документація FONDY. FONDY. URL: https://docs.fondy.eu/uk (дата звернення: 05.10.2025).

8. Documentation. Stripe API. URL: https://docs.stripe.com/ (дата звернення: 07.10.2025).

9. Introduction – Nuxt v3. *Nuxt JS*. URL: https://nuxt.com/docs/3.x/getting-started/introduction (дата звернення: 10.10.2025).

10. Vue.js – Framework. Vue.js. URL: https://vuejs.org/guide/introduction (дата звернення: 15.10.2025).

11. TypeScript – Documentation. TypeScript. URL: https://www.typescriptlang.org/docs/ (дата звернення: 22.10.2025).

12. Поширені запитання по Композиційному API. Vue.js. Vue.js. URL: https://ua.vuejs.org/guide/extras/composition-api-faq.html (дата звернення: 25.10.2025).

13. MySQL – Documentation. MySQL. URL: https://dev.mysql.com/doc/ (дата звернення: 02.11.2025).

14. NPM Docs. NPM. URL: https://docs.npmjs.com/ (дата звернення: 05.11.2025).

15. Oloruntoba S. SOLID Design Principles Explained: Building Better Software Architecture. DigitalOcean. URL: https://tinyurl.com/y6mntjy9 (дата звернення: 07.11.2025).

16. Laravel Actions. Actions. URL: https://www.laravelactions.com/ (дата звернення: 10.11.2025).

17. Laravel File Manager. UniSharp Open-Source. URL: https://unisharp.github.io/laravel-filemanager/ (дата звернення: 12.11.2025).

18. AdminLTE 3. Dashboard. Free Bootstrap Admin Template - AdminLTE.IO. URL: https://adminlte.io/themes/v3/index.html (дата звернення: 15.11.2025).

19. Laradock. Use Docker First. URL: https://laradock.io/docs/usage/ (дата звернення: 17.11.2025).