МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ІНТЕРНЕТ МАГАЗИНА З ПРОДАЖУ ВІЙСЬКОВОГО СПОРЯДЖЕННЯ»**

Виконала: студентка 4 курсу, групи 6.1211-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
А.О. Куксенко
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Мухін В.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Куксенко Анні Олегівні
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка інтернет магазина з продажу військового спорядження

керівник роботи Мухін Віталій Вікторович, к.т.н. доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Теоретичні основи розробки вебзастосунків.
3. Розробка вебзастосунку «Магазин військового спорядження».

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 26.12.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.01.2025	
2.	Збір вихідних даних.	23.01.2025	
3.	Обробка методичних та теоретичних	11.02.2025	
	джерел.		
4.	Розробка першого та другого розділу.	10.04.2025	
5.	Розробка третього розділу.	19.05.2025	
6.	Оформлення та нормоконтроль	25.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	17.06.2025	

Студент _____ А.О. Куксенко
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.В. Мухін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка інтернет магазина з продажу військового спорядження»: 67 с., 38 рис., 1 табл., 9 джерел, 1 додаток.

ВЕБЗАСТОСУНОК, ВІЙСЬКОВЕ СПОРЯДЖЕННЯ, ЕЛЕКТРОННА КОМЕРЦІЯ, ІНТЕРНЕТ-МАГАЗИН, ФРЕЙМВОРК, LARAVEL, REST API, SQLITE, VUE.JS.

Об’єкт дослідження – процес розробки сучасного вебзастосунку для електронної комерції, що забезпечує взаємодію користувача з каталогом товарів та їх замовлення.

Мета роботи – створення повноцінного клієнт-серверного вебзастосунку «Магазин військового спорядження» з зручнім та ефективним інтерфейсом для користувача, функціоналом для управління каталогом товарів та оформлення замовлень.

Метод дослідження – аналіз предметної області, методи об’єктно-орієнтованого проєктування, прототипування інтерфейсу, використання фреймворків Vue.js та Laravel, організація взаємодії між клієнтом і сервером через REST API, тестування функціональності вебзастосунку.

В ході роботи було розроблено односторінковий вебзастосунок «Магазин військового спорядження». Для реалізації клієнтської частини використовувався Vue.js, для розробки серверної логіки – Laravel. Ключовий функціонал вебзастосунку включає в себе: перегляд доступних товарів, фільтрацію за категоріями та тегами, додавання товарів до кошика, оформлення замовлень. Весь функціонал реалізовано без потреби оновлювати сторінку.

SUMMARY

Bachelor’s qualifying paper “Development of an Online Store for Military Equipment Sales”: 67 pages, 38 figures, 1 table, 9 references, 1 supplement.

E-COMMERCE, EQUIPMENT, FRAMEWORK, LARAVEL, MILITARY ONLINE STORE, REST API, SQLITE, VUE.JS, WEB APPLICATION.

The object of the study is the process of developing a modern e-commerce web application that allows users to interact with a catalog of goods and order them.

The aim of the study is creation of a full-fledged client-server web application “Military Equipment Store” with a convenient and efficient user interface, functionality for managing the product catalog and placing orders.

The methods of research are analysis of the subject area, object-oriented design methods, interface prototyping, use of Vue.js and Laravel frameworks, organization of interaction between the client and server via Rest Api, testing the functionality of a web application.

During the project, a single-page web application called “Military Equipment Store” was developed. Vue.js was used for the client-side implementation, while Laravel was used for developing the server-side logic. The core functionality of the application includes viewing available products, filtering by categories and tags, adding products to the cart, and placing orders. All features are implemented in a way that eliminates the need to reload the page.

ЗМІСТ

Завдання на кваліфікаційну роботу ...……..2
Реферат 4
Summary 5
Вступ..... 8
1 Аналіз предметної області та вибір засобів розробки..... 10
1.1 Сучасні підходи до створення вебдодатків для електронної комерції.10
1.2 Односторінкові вебзастосунки (SPA) 11
1.3 Огляд фреймворку клієнтської частини: Vue.js 3.5..... 11
1.4 Огляд фреймворку серверної частини: Laravel 12..... 12
1.5 Реалізація та особливості REST API..... 13
2 Проєктування архітектури вебзастосунку..... 15
2.1 Структура вебзастосунку 15
2.1.1 Система маршрутизації та хуку..... 17
2.1.2 ER-діаграма сутностей 17
2.1.3 Архітектурний шаблон MVC..... 19
2.2 Висновки до 2 розділу 21
3 Реалізація вебзастосунку..... 22
3.1 Реалізація клієнтської частини 22
3.1.1 Відображення головної сторінки та обробка даних..... 23
3.1.2 Реалізація каталогу з фільтрацією..... 24
3.1.3 Реалізація управління кошиком 24
3.1.4 Використання стилів..... 27
3.1.5 Взаємодія компонентів через props та події у Vue..... 28
3.1.6 Використання шрифтів..... 29
3.2 Реалізація серверної частини..... 30
3.2.1 Реєстрація маршрутів API у Laravel 30
3.2.2 Реалізація моделі товару ShopItem..... 32

3.3 Візуалізація функціоналу вебзастосунку 35
3.4 Висновки до 3 розділу 45
Висновки 47
Перелік посилань 48
Додаток А Код основних елементів вебзастосунку 49

ВСТУП

У сучасних умовах розвитку інформаційних технологій інтернет-магазини стали невід'ємною частиною глобального ринку, забезпечуючи зручний та швидкий спосіб продажу товарів та послуг. Зокрема, в Україні стала актуальною потреба на військове спорядження, починаючи від військових та волонтерів, закінчуючи цивільним населенням. Саму тому набуває актуальності створення функціонального інтернет-магазину військового спорядження.

Інтернет-магазини сьогодні активно розвиваються і їх ефективність напряму залежить від використаних технологій. Тому для того, щоб забезпечити сайт повним функціоналом, використовувались такі вебтехнології, як Vue.js для Front-end та Laravel для Back-end розробки.

Vue.js – це сучасний JavaScript-фреймворк, який чудово підходить для інтерактивних SPA («додаток однієї сторінки»). Він дозволить створити швидкий, інтерактивний та масштабований інтерфейс. А Laravel, як один із самих найпопулярніших PHP-фреймворків, забезпечить інтернет-магазину надійний бекенд із зручною роботою з даними, безпекою та API.

Метою дослідження є створення повноцінного клієнт-серверного вебзастосунку «Магазин військового спорядження», який забезпечить користувачів зручним інтерфейсом, функціональним управлінням каталогом товарів та оформленням замовлень.

Під час розробки вебзастосунку були поставленні наступні задачі:

– проаналізувати вимоги до систем та типові рішенні у галузі електронної комерції;
– дослідити можливості фреймворків Vue.js та Laravel для реалізації SPA та REST API;
– розробити структуру бази даних для зберігання інформації про товари;

– реалізувати взаємодію фронтенда з бекендом через Axios;
– спроєктувати інтерфейс користувача та адмін-панелі;
– протестувати роботу системи на валідність даних, коректність запитів і адаптивність.

Об’єкт дослідження – процес розробки сучасного вебзастосунку для електронної комерції.

Предмет дослідження – реалізація клієнтської та серверної частини вебзастосунку інтернет магазину військового спорядження з використанням технологій Vue.js та Laravel.

Методи дослідження – аналіз предметної області, методи об’єктно-орієнтованого проєктування, прототипування інтерфейсу, використання фреймворків Vue.js та Laravel, організація взаємодії між клієнтом і сервером через REST API, тестування функціональності вебзастосунку.

Для досягнення поставленої мети в дипломному проєкті було використано сучасний технологічний стек. Клієнтська частина розроблена за допомогою фреймворку Vue.js версії 3.5, який дозволяє створювати реактивний і адаптивний інтерфейс користувача у вигляді односторінкового застосунку (SPA). Сервісну логіку реалізовано на основі Laravel 12 – одного з найпопулярніших PHP-фреймворків, який у поєднанні з PHP 8.2 забезпечує безпечну та ефективну побудову REST API для обміну даними з клієнтською частиною. Для зберігання інформації про товари, замовлення та користувачів використано реляційну СКБД SQLite.

В результаті реалізації було створено повноцінний вебзастосунок із функціоналом перегляду, пошуку та сортування товарів, додавання до кошику, оформлення замовлень та управлінням товарною базою.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ

1.1 Сучасні підходи до створення вебдодатків для електронної комерції

Однією з найбільш революційних галузей нової економіки є електронна комерція. Електронна комерція (е-комерція) – різновид бізнес-активності, в якій взаємодія суб'єктів бізнесу з купівлі-продажу товарів і послуг (як матеріальних, так й інформаційних) здійснюється з допомогою глобальної комп'ютерної мережі Internet або будь-якої іншої інформаційної мережі [1]. Активний зріст використання інтернету, розвиток цифрової економіки та зміна споживацьких звичок спричинили стрімке зростання популярності інтернет-магазинів та онлайн-серверів. Сучасні користувачі прагнуть отримати доступ до товарів та послуг швидко та без усіляких зайвих зусиль, а бізнес – скоротити витрати на оренду приміщень, наймання персоналу тощо.

Одним із ключових елементів успішного онлайн-бізнесу є наявність ефективного, зручного та функціонального вебзастосунку. Вебзастосунок – це програма, яка працює у веббраузері та не потребує встановлення на пристрій користувача. Це забезпечує найкращу доступність та гнучкість платформи.

Розробка таких вебзастосунків включає побудову клієнтської частини (фронтенду), серверної частину (бекенду) та бази даних, які разом забезпечують повноцінну взаємодію між користувачем та бізнес-логікою системи.

Сучасні вебтехнології дозволяють інтегрувати просунуті інтерфейси (наприклад, користувацькі екрани, кошики для покупок), зручні можливості для взаємодії з користувачами, онлайн-платежі, управління каталогом товарів, управління замовленнями та зворотній зв'язок. Також з їх допомогою відкриваються широкі можливості для реалізації SPA –

односторінкових застосунків, які динамічно оновлюють дані без необхідності перезавантаження сторінки. Такий підхід дозволяє підвищити зручність та швидкість взаємодії користувача з вебзастосунком.

1.2 Односторінкові вебзастосунки (SPA)

SPA (Single Page Application) – це тип вебзастосунку, в якому основні ресурси (HTML, CSS та JavaScript) завантажуються одноразово, після чого здійснюється динамічне оновлення контенту без перезавантаження сторінки. Такий метод покращує взаємодію між користувачем та системою, підвищує швидкість та зручність користування.

SPA дає такі переваги для вебзастосунку:

– створення інтерактивного каталогу товарів;
– динамічне фільтрування асортименту;
– швидке оформлення замовлень через API;
– оновлення інтерфейсу в реальному часі.

Серед недоліків SPA – це складність у пошуковій оптимізації (SEO), оскільки пошукові системи не завжди коректно індексують JavaScript-контент, а також збільшений початковий розмір завантажуваних файлів і потреба в клієнтському маршрутизаторі. Однак таких проблем можна уникнути, якщо застосувати підтримку технологій SSR (рендеринг на стороні сервера) або SSG (статична генерація сторінок). У нашому випадку SPA є доречним вибором, оскільки вебзастосунок орієнтований на активну взаємодію користувача з інтерфейсом у реальному часі.

1.3 Огляд фреймворку клієнтської частини: Vue.js 3.5

Серед найпопулярніших фреймворків для розробки вебдодатків є React, Angular та Vue.js. У даній роботі було обрано Vue.js версії 3.5, оскільки цей

фреймворк має баланс простоти та потужної функціональності: реактивність, компонентна архітектура, підтримка TypeScript, зручна екосистема (Vue Router, Pinia, Vite) та висока продуктивність.

Vue – це фреймворк JavaScript для створення користувацьких інтерфейсів. Він базується на стандартних HTML, CSS та JavaScript і надає декларативну компонентну модель програмування, яка допомагає ефективно розробляти користувацькі інтерфейси будь-якої складності [2].

Vue.js дозволяє створити односторінковий додаток (SPA), який може реагувати та взаємодіяти з користувачем надзвичайно швидко, не оновлюючи сторінку при кожному оновленні стану. Це набагато підвищує зручність користування та дозволяє реалізувати динамічну логіку інтерфейсу.

Основні переваги Vue.js у розробці проєкту:

– модульна архітектура – система компонентів дозволяє розбивати інтерфейс на незалежні блоки, що значно підвищує можливість повторного використання коду;
– реактивна модель даних – автоматична синхронізація стану додатку з відображенням, що усуває необхідність ручного оновлення DOM;
– гнучкість інтеграції – можливість як поступового впровадження в існуючі проєкти, так і побудови повноцінних SPA-додатків;
– Composition API – інноваційний підхід до організації коду, який особливо ефективний для складних компонентів;
– доступність вивчення – чітка документація та логічна структура роблять фреймворк зручним для освоєння розробниками різного рівня.

1.4 Огляд фреймворку серверної частини: Laravel 12

Laravel – це сучасний фреймворк для розробки вебзастосунків на мові PHP, вирізняється чистою архітектурою, високим рівнем безпеки, широким набором вбудованих інструментів. Фреймворк надає можливість створювати

RESTful API, реалізовувати маршрутизацію, контролери, автентифікацію, авторизацію, валідацію форм, черги завдань та багато інших функцій без необхідності використовувати сторонні рішення. Його структура дозволяє розділити бізнес-логіку, обробку запитів та взаємодію з базою даних, що сприяє підтримці й масштабованості застосунків.

Laravel неймовірно масштабований. Завдяки дружній до масштабування природі PHP та вбудованій підтримці Laravel швидких розподілених систем кешування, таких як Redis, горизонтальне масштабування з Laravel – це легка справа. Насправді, додатки на Laravel легко масштабуються для обробки сотень мільйонів запитів на місяць [3].

Однією з головних переваг Laravel є його видатна здатність каталізувати процес розробки. Це дозволяє компаніям швидко виводити свої продукти та послуги на ринок.

Фреймворк також підтримує модульне програмування за допомогою менеджера пакетів Composer, що полегшує повторне використання коду, а також створення вже існуючих рішень. Це призводить до більш ефективного та швидкого циклу розробки, що мінімізує час виходу додатків на ринок.

Laravel пропонує різноманітні потужні функції безпеки. Він має вбудований механізм захисту від різних поширених загроз, включаючи міжсайтовий скриптинг (XSS), SQL-ін’єкції та підробку міжсайтових запитів.

Він має модульну архітектуру, яка забезпечує організацію коду за допомогою патернів MVC і пропонує вбудовану підтримку модульної розробки. Це дозволяє розробникам організовувати свій код в окремі модулі. Кожен модуль відповідає за певну функціональність і дозволяє зручно керувати складністю [4].

1.5 Реалізація та особливості REST API

REST API (Application Programming Interface) – це один з найпопулярніших архітектурних стилів, що використовується для взаємодії

між клієнтом та сервером, який став сучасним стандартом у веброзробці. Цей стиль базується на стандартних HTTP-запитах і передбачає роботу з ресурсами (наприклад, категоріями, товарами і тегами) за допомогою використання таких методів як: GET, POST, PUT, DELETE.

Однією з ключових переваг цього стилю є незалежність клієнта та сервера, яка дозволяє різним командам працювати паралельно над окремими частинами проєкту. Стандартизація API сприяє повторному використанню компонентів, масштабованості та забезпеченню безпеки за допомогою сучасних протоколів автентифікації.

У контексті електронної комерції REST API надає можливість будувати високоефективні й надійні системи, в яких запити на отримання або обробку інформації про товари, замовлення, категорії або теги виконується швидко, структуровано та відповідно до стандартів.

Отже, REST API є не лише технічним інструментом, а й основним архітектурним підходом, що забезпечує гнучкість, модульність та незалежність компонентів вебзастосунку.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ВЕБЗАСТОСУНКУ

Для успішної реалізації вебзастосунку необхідне чітке визначення його архітектурної моделі. Архітектура програмного забезпечення – це структурна модель, яка визначає компоненти системи, їх взаємодію, як будуть структуровані дані, а також засоби масштабування і підтримку системи. Правильний вибір архітектури є критично важливим фактором, що гарантує гнучкість, продуктивність та ремонтопридатність системи.

Для створення вебзастосунку «Магазин військового спорядження» було обрано клієнт-серверну архітектуру з REST API як протоколом зв’язку між фронтендом (на Vue.js) та бекендом (на Laravel). Саме такий підхід дозволяє розділити відповідальність між інтерфейсною та логічною частинами системи, полегшити розробку та забезпечити подальше масштабування.

2.1 Структура вебзастосунку

Структура вебзастосунку розроблена на основі класичної клієнт-серверної архітектури з чітким поділом на фронтенд та бекенд частину, які взаємодіють через REST API. Такий підхід забезпечує масштабованість, модульність та гнучкість при реалізації нових функцій та також дозволяє незалежно розвивати інтерфейс користувача, так і логіку обробки даних.

Клієнтська частина написана на фреймворку Vue.js 3.5 з використанням Vite який збирає модулі та Tailwind CSS для стилізації. Основні структурні компоненти:

– pages (сторінки) – основні розділи інтерфейсу (головна сторінка, категорії, корзина);
– components (компоненти) – багаторазові частини інтерфейсу (карточки товарів, тегові фільтри, навігаційні панелі);

– store (Pinia) – централізоване управління станом для збереження інформації про товари, контент корзини, теги;
– API-сервіси – функції, які обробляють запити до бекенду.

Серверна частина розроблена на Laravel 12:

– controllers – обробка HTTP-запитів, фільтрація даних;
– models – об’єктно-реляційне представлення сутностей товарів, категорій, тегів та атрибутів;
– migrations – опис структури бази даних у вигляді коду з підтримкою контролю версій;
– routes (API) – реєстрація публічних маршрутів /api для взаємодії з клієнтом.

Для зберігання інформації використовується SQLite. SQLite – це вбудована бібліотека, яка реалізує автономний, безсерверний, транзакційний механізм баз даних SQL без налаштування. На відміну від більшості інших баз даних SQL, SQLite не має окремого серверного процесу. SQLite читає та записує безпосередньо у звичайні файли на диску [5].

Основні таблиці:

– shop_items – товари;
– shop_categories – категорії;
– shop_tags – теги;
– shop_item_attributes – атрибути;
– реалізація зв’язків «багато-до-багатьох».

Частини фронтенду та бекенду виконують свою взаємодію через REST API з використання методів GET, POST, DELETE. На рисунку 2.1 зображено схему структури вебзастосунку.

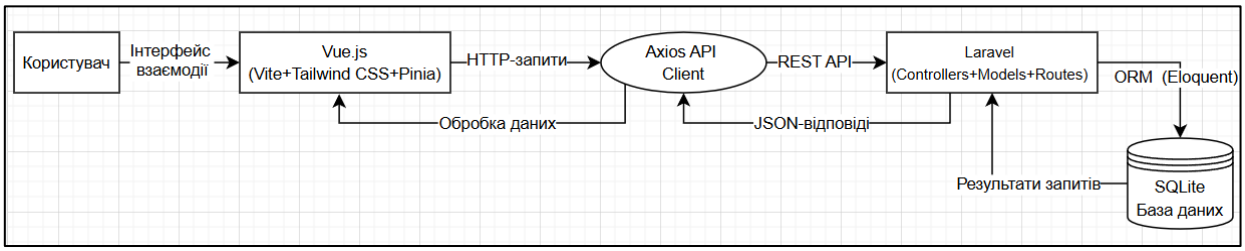


Рисунок 2.1 – Схема структури вебзастосунку

2.1.1 Система маршрутизації та хуку

Система маршрутизації відіграє важливу роль у зручній навігації між сторінками інтерфейсу без перезавантаження сторінки. Щоб забезпечити навігацією вебзастосунок було використано офіційну бібліотеку Vue Router, яка організовує маршрути у вигляді об’єкта конфігурації з точним описом всіх шляхів і відповідного йому компонента.

Конфігурація маршрутизатора базується на використанні createRouter та createWebHistory, яка забезпечить чисті URL-адреси.

У вебзастосунку реалізовано такі маршрути:

– / – головна сторінка (MainPage.vue);
– /cart – сторінка з товарами у кошику (CartView.vue);
– /catalog – каталог товарів (CatalogView.vue);
– /about – сторінка інформації про магазин (AboutUs.vue);
– /:pathMatch(.)* – маршрут для обробки помилок 404 ().

Окрім маршрутів, також було реалізовано глобальний навігаційний хук beforeEach, який записує попередні дані про маршрут у властивість router.previousRouter. Завдяки цьому в застосунку реалізована розширена навігаційна логіка, що дозволяє, наприклад, повернення на попередню сторінку сайту.

Відображення змісту маршруту на сторінці виконується завдяки тегу <router-view>, який слугує місцем вставки активного компонента залежно від поточного шляху. Перехід між сторінками реалізується через компонент <router-link> або програмно за допомогою router.push().

2.1.2 ER-діаграма сутностей

Діаграма «сутність-зв’язок», також відома як ERD, ER-діаграма або ER-модель, – це тип структурної діаграми, що використовується при проєктуванні баз даних. ER містить різні символи та з’єднувачі, які

візуалізують дві важливі інформації: Основні сутності в межах системи та взаємозв’язки між цими сутностями [6]. Така модель допомагає зрозуміти взаємозв’язки між даними.

У вебзастосунку «Магазин військового спорядження» до структури бази даних включені такі основні сутності:

– ShopItems – основна таблиця товарів з назвою, описом, цінами та зображенням;
– ShopCategories – таблиця категорій товарів;
– ShopTags – таблиця тегів, яка описує певні характеристики або особливості товару;
– ShopItemAttributes – таблиця атрибутів товарів, яка розширює опис товару специфічними даними;
– Ads – таблиця з рекламними блоками;
– Pivot-таблиці – реалізують зв’язки між сутностями багато-до-багатьох: shop_item_shop_tag, shop_category_shop_item, shop_item_shop_item_attribute, shop_catogory_shop_item_attribute.

У системі електронної комерції між сутностями бази даних встановлюється кілька типів зв’язків, щоб забезпечити ефективне зберігання, пошук та взаємодію даних. Найпоширенішими з них є зв’язки «багато-до-багатьох», реалізовані через проміжні (pivot) таблиці, де можна зберігати додаткову інформацію про зв’язки між сутностями. У таблиці 2.1 показано основні зв’язки між сутностями, які використовуються у вебзастосунку.

Таблиця 2.1 – Зв’язки між сутностями

Сутність 1	**Сутність 2**	**Pivot-таблиця**	**Тип зв’язку**	**Додаткові поля у Pivot-таблиця**
ShopItem	ShopTag	shop_item_shop_tag	Багато-до-багатьох	is_displayed
ShopItem	ShopCategory	shop_category_shop_item	Багато-до-багатьох	is_main
ShopItem	ShopItemAttribute	shop_item_shop_item_attribute	Багато-до-багатьох	value
ShopCategory	ShopItemAttribute	shop_category_shop_item_attribute	Багато-до-багатьох	is_important

З метою ефективного проєктування структури бази даних для вебзастосунку було розроблено ER-діаграму (див. рис. 2.2, ст. 20).

2.1.3 Архітектурний шаблон MVC

Одним з головних етапів розробки вебзастосунку є вибір ефективного архітектурного шаблону.

У вебзастосунку «Магазин військового спорядження» був використаний шаблон MVC (Model-View-Controller).

MVC – це шаблон проєктування, який використовується для розділення інтерфейсу користувача (представлення), даних (модель) і логіки програми (контролер). Цей шаблон допомагає досягти розділення завдань [7].

У рамках вебзастосунку «Магазин військового спорядження»:

– model – відповідає за роботу з базою даних, яка реалізується через Eloquent ORM у Laravel. Моделі взаємодіють з таблицями, встановлюють зв’язки між сутностями;
– view – реалізуються через компоненти Vue.js, які формують візуальний інтерфейс, адаптивний до потреб користувача. Кожен маршрут Vue містить відповідний візуальний компонент;
– controller – обробляє запити з клієнтської частини, взаємодія з моделями та передає дані у вигляді відповідей API. У Laravel контролери, такі як IndexItemsController, MainPageController, реалізують REST-логіку.

Використання шаблону MVC дозволило зменшити зв’язність між компонентами, забезпечило масштабованість системи та пришвидшило процес розробки.

Такі особливості досить важливі для динамічних проєктів, таких як, інтернет-магазин, де постійно змінюється логіка обробка даних, інтерфейс користувача або представлення контенту.

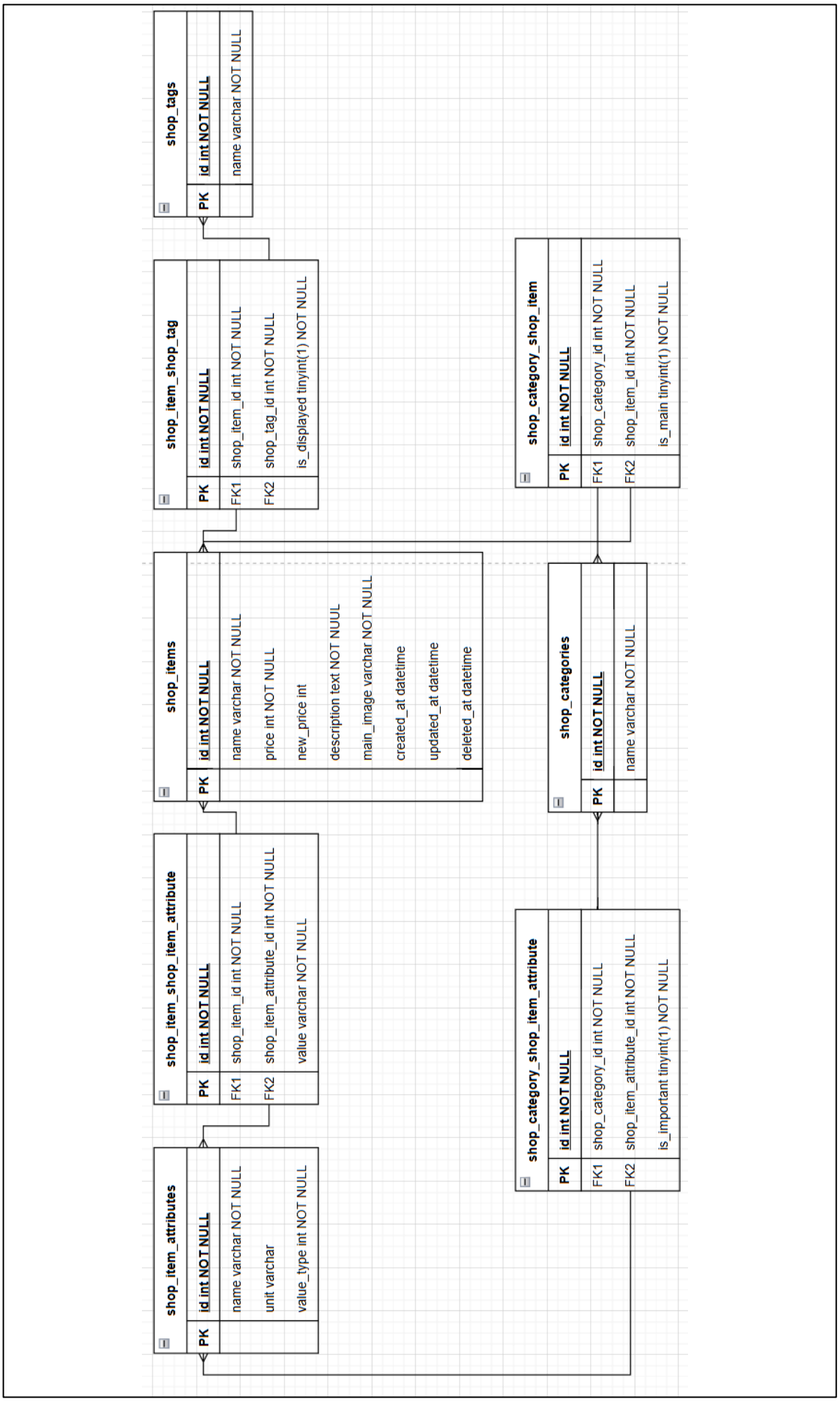


Рисунок 2.2 – ER-діаграма сутностей

2.2 Висновки до 2 розділу

У другому розділі дипломної роботи було проведено повне проєктування архітектури вебзастосунку «Магазин військового спорядження». На основі аналізу вимог було обрано клієнт-серверну архітектуру з REST API, яка забезпечує чіткий розподіл обов’язків між фронтендом (Vue.js) та бекендом (Laravel). За допомогою такої архітектури можна досягти більшої гнучкості, масштабованості та спрощення обслуговування системи.

Структура вебзастосунку була розроблена на основі компонентної архітектури, централізованого стану та ефектної маршрутизації. Особливу увагу приділено розробці структури даних, де зв’язки між сутностями були спроєктовані на основі ER-діаграми, а реалізовані за допомогою проміжних таблиць. Також обґрунтовано використання архітектурного шаблону MVC, що дозволило оптимізувати взаємодію логіки та представлення даних.

Загалом, архітектурне рішення забезпечує міцну базу для подальшої реалізації, функціонального збагачення та підтримки вебдодатку.

3 РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

У цьому розділі представлено опис практичної реалізації вебзастосунку «Магазин військового спорядження». Вебзастосунок реалізовано на основі клієнт-серверної моделі: frontend (Vue.js 3.5), backend (Laravel 12) та з використанням бази даних SQLite.

3.1 Реалізація клієнтської частини

Клієнтська частина вебзастосунку розроблена на фреймворку Vue.js 3.5. Для взаємодії клієнтської частини з сервером у вебзастосунку використано бібліотеку Axios. У проєкті було створено окремий екземпляр Axios, в якому реалізовано централізовану конфігурацію всіх запитів. Завдяки використанню перехоплювача interceptors, який автоматично додає базову адресу сервера до кожного запиту та використовуючи значення змінної середовища VITE_API_BASE_URL. Це дозволяє просто адаптувати вебзастосунок до різних середовищ розгортання, змінюючи при цьому лише одне налаштування без необхідності редагувати кожен окремий запит. На рисунку 3.1 зображено приклад коду конфігурації Axios, який використовується для централізованого управління HTTP-запитами у клієнтській частині вебзастосунку.

```
1   import axios from 'axios'
2
3   const api = axios.create()
4
5   api.interceptors.request.use((config) => {
6       config.baseURL = import.meta.env.VITE_API_BASE_URL;
7
8     return config;
9   })
10
11  export default api
```

Рисунок 3.1 – Приклад конфігурації Axios

3.1.1 Відображення головної сторінки та обробка даних

Однією з основних сторінок є MainPage.vue, яка є початковим інтерфейсом взаємодії користувача з магазином. На ній зображені товари зі знижками, товари місяця та інше. Відповідні дані надсилаються з бекенду за допомогою асинхронного запиту до бекенду при завантаженні сторінки.

У клієнтській частині Vue-компонент MainPage.vue відповідає за відображення цих даних.

Основна логіка виконується під час життєвого onMounted – коли компонент уже змонтовано, і можна безпечно звертатись до API. Дані, які було отримано, будуть збережені в реактивні змінній типу ref, потім Vue автоматично оновлює інтерфейс, показуючи користувачеві відповідні товари на головній сторінці.

На рисунку 3.2 представлено приклад завантаження даних для головної сторінки вебзастосунку за допомогою onMounted та ref.

```
<script setup>
import InlineAd from '@/components/ads/InlineAd.vue'
import ShopItemCard from '@/components/shop/ShopItemCard.vue'
import { useRouter } from 'vue-router'
import { onMounted, ref } from 'vue'
import api from '@/api.js'

const router = useRouter()

const monthlyItems = ref([]);
const discountedItems = ref([]);
const weapons = ref([]);
const others = ref([]);

onMounted(() => {
  api.get('landing').then(
    (response) => {
      monthlyItems.value = response.data.items_month;
      discountedItems.value = response.data.items_discounts;
      weapons.value = response.data.weapons;
      others.value = response.data.not_weapons;
    },
    (error) => {
      console.log(error)
    },
  )
})
```

Рисунок 3.2 – Фрагмент коду компонента MainPage.vue

3.1.2 Реалізація каталогу з фільтрацією

Головною задачею каталогу товарів є показ користувачеві всіх товарів з можливістю фільтрації за категоріями та тегами.

Компонент CatalogView.vue відповідає за виведення списку товарів, їх фільтрації за категоріями та тегами, а також оновлює вміст без перезавантаження сторінки. Рисунок 3.3 демонструє, як відбувається взаємодія з сервером та формування структури даних для подальшого відображення на сторінці, а саме як йде отримання товарів з API та оновлення масиву goods.

```
42    await api
43      .get('goods', {
44        params: {
45          category: selectedCategory.value,
46          tags: selectedTags.value,
47        },
48      })
49      .then(
50        (response) => {
51          goods.value = []
52
53          response.data.forEach((good) => {
54            goods.value.push({
55              name: good.name,
56              price: good.price,
57              new_price: good.new_price ?? null,
58              description: good.description,
59              image_source: good.main_image,
60              tags: good.tags,
61            })
62          })
63        },
```

Рисунок 3.3 – Фрагмент коду отримання товарів з API та оновлення масиву goods

3.1.3 Реалізація управління кошиком

У вебзастосунку «Магазин військового спорядження» реалізація управлінням кошика працює за допомогою бібліотеки Pinia. Pinia – це

офіційна система управління станом для Vue.js 3, яка прийшла на заміну Vuex. Вона базується на Composition API, проста у використанні, типобезпечна та інтегрується з Devtools.

Pinia дозволяє зручно зберігати та оновлювати глобальний стан вебзастосунку. На відміну від складної конфігурації Vuex, Pinia працює як звичайні JavaScript-функції з прозорою реактивністю.

Функція defineStore, використовується для створення store і містить три ключові частини:

– state: реактивні змінні (масив товарів у кошику);
– getters: обчислюванні властивості (загальна сума, кількість і т.д.);
– actions: методи для зміни стану (додавання чи видалення товарів).

Через геттер total розраховується загальна вартість усіх товарів у кошику, а itemsCount повертає загальну кількість одиниць товару (див. рис. 3.4). Це дозволяє швидко та динамічно відображати відповідну інформацію на інтерфейсі користувача.

```
getters: {
  total: (state) => state.items.reduce((acc, item) => acc + item.price * item.quantity, 0),

  allItems: (state) => state.items,

  itemsCount: (state) => state.items.reduce((acc, item) => acc + item.quantity, 0),

  itemByName: (state) => (name) => state.items.find((item) => item.name === name),

  itemCountByName: (state) => (name) => state.items.find((item) => item.name === name)?.quantity ?? 0,
},
```

Рисунок 3.4 – Геттери стану кошика у Pinia

Важливою частиною стало додавання механізму збереження стану кошика в LocalStorage браузера. Навіть якщо користувач закриє або оновить сторінку, то вміст кошика буде збереженим. Для цього застосовується метод updateLocalStorage, який оновлюється при кожній зміні (додаванні або видаленні товару, очищенні корзини).

Метод addItem (див. рис. 3.5) перевіряє, чи вже є товар у кошику. Якщо так – збільшує його кількість, якщо ні – то додає новий товар.

```
actions: {
  updateLocalStorage() {
    localStorage.setItem('cart-items', JSON.stringify(this.items))
  },

  /**
   * @param {CartItem|string} item
   */
  addItem(item) {
    const isJustName = typeof item === 'string'

    const itemName = isJustName ? item : item.name

    const existing = this.itemByName(itemName)

    if (existing) {
      existing.quantity += isJustName ? 1 : item.quantity
    } else {
      if (!isJustName) {
        this.items.push(item)
      }
    }

    this.updateLocalStorage()
  },
```

Рисунок 3.5 – Додавання товарів у кошику

Метод removeItem (див. рис. 3.6) видаляє товар з кошика за його назвою. А метод clear() очищує весь вміст кошика. Кожна з цих дій викликає updateLocalStorage, яка синхронізує стан з локальним сховищем браузера. Така логіка забезпечує вебзастосунок зручним управлінням списком покупок та динамічною зміною стану без перезавантаження сторінки.

```
  /**
   * @param {string} itemName
   */
  removeItem(itemName) {
    const existing = this.itemByName(itemName)

    if (existing && existing.quantity > 1) {
      existing.quantity--
    } else {
      this.items = this.items.filter((i) => i.name !== itemName)
    }

    this.updateLocalStorage()
  },

  clear() {
    this.items = []

    this.updateLocalStorage()
  },
},
})
```

Рисунок 3.6 – Видалення товарів з кошика

3.1.4 Використання стилів

Для стилізації клієнтської частини вебзастосунку було використано CSS-фреймворк Tailwind CSS. Це сучасний інструмент стилізації, який забезпечує практичний підхід для створення інтерфейсу застосунків. Tailwind CSS значно пришвидшує верстку, завдяки утилітарному підходу. Цей підхід виключає потребу у написанні зайвого CSS [8].

Завдяки Tailwind можна скоротити час, витрачений на написання окремих файлів стилів для більшості елементів, оскільки всі необхідні властивості – кольори, відступи, гнучкість, типографіка – доступні через класи. Це прискорює розробку і зменшує кількість повторюваного коду.

Для прикладу, як Tailwind CSS використовується для стилізації сторінки кошика (див. рис. 3.7).

```
<div class="max-w-4xl mx-auto">
  <div class="flex items-center justify-between mb-8">
    <h1 class="text-3xl text-main">Кошик</h1>

    <a @click="goBack" href="#" class="text-right">Повернутись</a>
  </div>
```

Рисунок 3.7 – Приклад використання Tailwind CSS

У прикладі:

– max-w-4xl встановлює максимальну ширину блоку;
– mx-auto централізує блок по горизонталі;
– flex, items-center, justify-between, використовуються для гнучкого вирівнювання елементів;
– text-3xl встановлює розмір шрифту.

Tailwind CSS дозволяє досить легко адаптувати інтерфейс для різних екранів користувачів за допомогою мобільних класів (sm:, md:, lg:).

3.1.5 Взаємодія компонентів через props та події у Vue

Одна з переваг Vue.js – це ефективна організація взаємодії між компонентами за допомогою механізмів props і events. Це дозволяє розробити інтерфейс із чіткою структурою, в якій кожен компонент виконує окрему роль та має обмежену відповідальність.

Для передачі даних від батьківського компонента до дочірнього використовуються вхідні параметри props. У компоненті ShopTag.vue через props передаються такі основні параметри:

– назва тегу;

– чи вибраний тег;

– кількість товарів за тегом.

А щоб дочірній компонент міг повідомити батьківському про певну дію користувача застосовується механізм генерації подій (emit). Наприклад, при кліку на тег компонент ShopTag.vue через emit('clicked') повідомляє батьківський компонент про взаємодію, що дозволяє реалізувати функціонал фільтрації товарів (див. рис. 3.8 – 3.9).

```
const props = defineProps({
  tag: {
    type: String,
    required: true,
  },

  isSelectable: {
    type: Boolean,
    default: false,
  },

  isClickable: {
    type: Boolean,
    default: false,
  },

  count: {
    type: [Number, String],
    default: "",
  },
```

Рисунок 3.8 – Передача props та генерація подій (частина 1)

```
  showCount: {
    type: Boolean,
    default: false,
  },

  isSelected: {
    type: Boolean,
    default: false,
  }
})

const emit = defineEmits(['clicked'])

const text = computed(() => {
  return props.showCount ? `${props.tag} (${props.count})` : props.tag
})

const handleClick = () => {
  props.isClickable && emit('clicked', { name: props.tag, isSelected: props.isSelected })
}
```

Рисунок 3.9 – Передача props та генерація подій (частина 2)

3.1.6 Використання шрифтів

Для того щоб вебзастосунок «Магазин військового спорядження» мав унікальне стилістичне оформлення інтерфейсу було використано кастомні шрифти, які підключаються за допомогою директиви @font-face. Це дозволило використати шрифти, які не входять до стандартного набору браузера, водночас зберігаючи контроль над їх виглядом та поведінкою.

У вебзастосунку було використано два основних шрифти: основний шрифт Main, що використовувався для описів, товарів і загального вмісту вебзастосунку, і другий декоративний шрифт Heads для заголовків і акцентних елементів інтерфейсу.

Щоб забезпечити кращий користувацький досвід було використано властивість font-display: swap (див. рис. 3.10). Поки шрифт ще не завантажився, браузер тимчасово показує системний шрифт, а після завантаження – автоматично підміняє на кастомний без особливих помітних візуальних збоїв.

```
@font-face {
  font-family: 'Main';
  src: url('./main.ttf') format('truetype');
  font-weight: normal;
  font-style: normal;
  font-display: swap;
}

@font-face {
  font-family: 'Heads';
  src: url('./heads.otf') format('opentype');
  font-weight: normal;
  font-style: normal;
  font-display: swap;
}
```

Рисунок 3.10 – Підключення кастомних шрифтів

3.2 Реалізація серверної частини

3.2.1 Реєстрація маршрутів API у Laravel

Маршрути API є ключовими точками входу для взаємодії клієнтської частиною з сервером. Усі маршрути знаходяться в файлі routes/api.php. Для покращення зручності та підтримки принципів REST маршрути поділено на групи відповідно до їх функціонального призначення: автентифікація, робота з каталогом, створення замовлень, виведення реклами.

Функції, які було реалізовано через API-маршрути:

– отримання та автентифікація користувача (Sanctum middleware);
– авторизація: реєстрація (sign-up), вхід (sign-in), вихід (sign-out);
– отримання даних про товари, категорії, теги та кремі товари;
– створення нового замовлення;
– отримання головного рекламного блоку.

На рисунку 3.11 показано реалізацію логіки автентифікації користувача. Для цього було використано Laravel Sanctum, який забезпечує безпечну роботу з токенами. Маршрут /user отримує інформацію про поточного авторизованого користувача.

```php
use App\Http\Controllers\CreateOrderController;
use App\Http\Controllers\GetUserController;
use App\Http\Controllers\IndexCategoriesController;
use App\Http\Controllers\IndexItemsController;
use App\Http\Controllers\IndexTagsController;
use App\Http\Controllers\MainPageController;
use App\Http\Controllers\ShowAdController;
use App\Http\Controllers\ShowItemController;
use App\Http\Controllers\SignInController;
use App\Http\Controllers\SignOutController;
use App\Http\Controllers\SignUpController;
use Illuminate\Support\Facades\Route;
use Laravel\Sanctum\Http\Controllers\CsrfCookieController;

Route::get(uri: '/user', action: GetUserController::class)->middleware(middleware: 'auth:sanctum');

Route::get(uri: 'csrf-cookie', action: [CsrfCookieController::class, 'show']);

Route::post(uri: '/sign-up', action: SignUpController::class);

Route::post(uri: '/sign-in', action: SignInController::class);

Route::delete(uri: '/sign-out', action: SignOutController::class);
```

Рисунок 3.11 – Реєстрація маршрутів

Додатково, для забезпечення захисту CSRF-токенів у вебзастосунку застосовується маршрут csrf-cookie. Такий маршрут ініціалізує встановлення відповідного cookie на стороні клієнта. Реєстрація нового користувача здійснюється через маршрут /sign-up, а вхід в систему – sign-in, які обробляються відповідними контролерами з валідацією даних. Після входу всі наступні запити автоматично авторизуються через middleware auth:sanctum, що зчитує токен з cookie. А для виходу передбачено маршрут /sign-out, який видаляє токен доступу.

Інші маршрути (див. рис. 3.12) використовують патерн контролера-однієї дії для обробки запитів на отримання даних:

– /landing – головна сторінка, обробляється MainPageController;
– /goods – повертає перелік товарів з фільтрами, обробляється IndexItemsCntroller;
– /categories – список категорій, обробляється IndexCategoriesController;
– /tags – список тегів, обробляється IndexTagsController;
– /ad – повертає один випадковий рекламний тег, обробляється ShowAdController.

Ці маршрути – це основні канали отримання даних клієнтською частиною через Axios. Вони реалізують стандартну архітектуру з використанням методу GET для читання інформації. Централізація реєстрації маршрутів у цьому коді робить систему легкою в обслуговуванні, масштабованою та зрозумілою.

```
27   Route::get(uri: '/landing', action: MainPageController::class);
28
29   Route::get(uri: '/goods', action: IndexItemsController::class);
30
31   Route::get(uri: '/goods/{item}', action: ShowItemController::class);
32
33   Route::post(uri: '/order', action: CreateOrderController::class);
34
35   Route::get(uri: '/categories', action: IndexCategoriesController::class);
36
37   Route::get(uri: '/tags', action: IndexTagsController::class);
38
39   Route::get(uri: '/ad', action: ShowAdController::class);
```

Рисунок 3.12 – Маршрути для роботи з контентом та створенням замовлень

3.2.2 Реалізація моделі товару ShopItem

У серверній частині вебзастосунку важливу роль відіграє модель ShopItem. В цій моделі описується основна сутність – товар у магазині. Вона реалізована як клас у фреймворку Laravel з використанням Eloquent ORM. Використання ORM дозволяє зменшити кількість SQL-запитів та підвищити безпеку й спростити взаємодію з базою даних [9]. Завдяки цьому модель може не лише зберігати й читати дані з бази даних, а ще встановлювати логіку фільтрації, сортування, доступ до пов’язаних даних та багато чого іншого.

Основні властивості моделі описуються на рисунку 3.13. У коді застосовується масив $fillable, в якому вказані поля, які дозволено масово заповнювати через форми або API. Таке обмеження потрібне для захисту від масового присвоєння полів. Властивість $with визначає зв’язки, які завжди

автоматично завантажуються разом з моделлю – у коді це атрибути товару shopAttributes та його основні теги (mainTags).

```
0 references
15      protected $fillable = [
16          'name',
17          'price',
18          'new_price',
19          'description',
20          'main_image',
21      ];
22
0 references
23      public $with = [
24          'shopAttributes',
25          'mainTags',
26      ];
27
```

Рисунок 3.13 – Властивості моделей

Наступна важлива частина проєкту є реалізація зв’язків між сутностями.

Метод tags() описує зв’язок «багато-до-багатьох» із моделлю тегів ShopTag (див. рис. 3.14).

```
public function tags(): BelongsToMany
{
    return $this
        ->belongsToMany(related: ShopTag::class)
        ->withPivot(columns: 'is_displayed');
}

1 reference | 0 overrides
public function mainTags(): BelongsToMany
{
    return $this
        ->tags()
        ->wherePivot(column: 'is_displayed', operator: true);
}
```

Рисунок 3.14 – Зв’язки з тегами

Додатково використовується поле is_displayed, яке зберігається в проміжній таблиці shop_item_shop_tag. Метод mainTags() дозволяє отримати лише ті теги, які мають активний прапорець is_displayed.

Категорії так само як теги реалізовані зв’язком багато-до-багатьох (див. рис. 3.15).

```php
public function categories(): BelongsToMany
{
    return $this
        ->belongsToMany(related: ShopCategory::class)
        ->withPivot(columns: 'is_main');
}

0 references | 0 overrides
public function mainCategory(): BelongsToMany
{
    return $this
        ->categories()
        ->wherePivot(column: 'is_main', operator: true)
        ->limit(value: 1);
}
```

Рисунок 3.15 – Зв’язки з категоріями

Атрибут товару (вага, розмір, колір тощо) реалізовані через зв’язок «багато-до-багатьох» (див. рис. 3.16). Значення кожного атрибуту для відповідного товару зберігається у проміжній таблиці shop_item_shop_item_attribute.

```php
public function shopAttributes(): BelongsToMany
{
    return $this
        ->belongsToMany(related: ShopItemAttribute::class)
        ->withPivot(columns: 'value');
}
```

Рисунок 3.16 – Зв’язки з атрибутами

Метод getValue(string|ShopItemAttribute $attribute): mixed – отримує конкретне значення атрибута за його назвою або об’єктом атрибута (див. рис. 3.17). Наприклад, можна швидко дізнатись вагу певного товару.

А метод public function getValues(): array – повертає всі атрибути товару (див. рис. 3.18), як асоціативний масив (назва атрибута => значення). Такий метод досить корисний для відображення технічних характеристик у шаблоні товару.

```php
public function getValue(string|ShopItemAttribute $attribute): mixed
{
    if ($attribute instanceof ShopItemAttribute) {
        $attribute = $attribute->name;
    }

    return $this
        ->shopAttributes
        ->firstWhere('name', $attribute)
        ?->value;
}
```

Рисунок 3.17 – Метод доступу до атрибутів

```php
public function getValues(): array
{
    return $this
        ->shopAttributes
        ->mapWithKeys(function (ShopItemAttribute $attribute): array {
            return [$attribute->name => $attribute->value];
        })
        ->toArray();
}
```

Рисунок 3.18 – Метод доступу до атрибута

3.3 Візуалізація функціоналу вебзастосунку

На головній сторінці (див. рис. 3.19) вебзастосунку вгорі розташоване основне меню з логотипом, справа іконка кошика, а зліва розташоване меню для переходу до інших сторінок. А після основного меню йде яскравий заголовок «SLAVA SHOP – Озбройся на славу!». Вебзастосунок має темно тло з контрастними червоними елементами.

Нижче – динамічна рекламна вставка. Наступним користувач бачить є блок «Товари місяця», який відображає рекомендовані товари. Кожен товар виводиться у вигляді окремої картки з фото, назвою, ціною та тегами.

Користувач може прокручувати цей список, щоб продивитись інші товари. Для цього був реалізовано горизонтальний скрол.

Головна сторінка не просто інформаційна, а й динамічна – на ній зручно взаємодіяти з вмістом, швидко знайти актуальні пропозиції та одразу перейти до покупки.

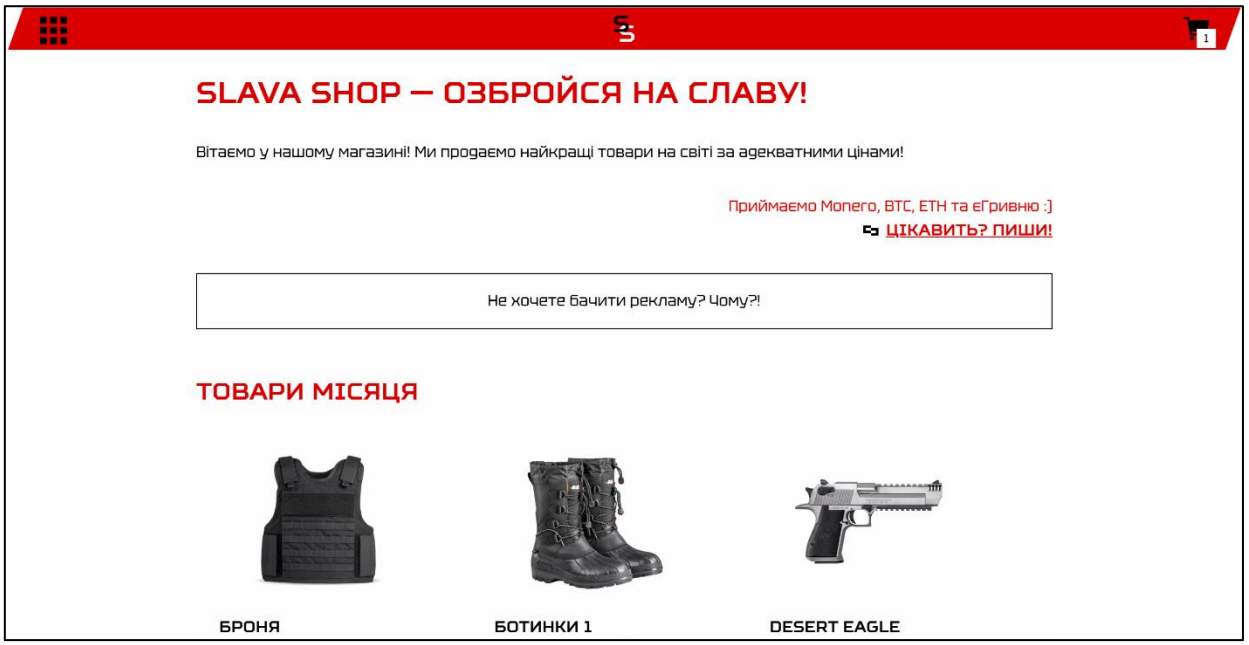


Рисунок 3.19 – Головна сторінка вебзсатосунку

Прокрутивши вниз користувач бачить наступний блок з товарами зі знижкою (див. рис. 3.20). У цьому блоці візуально зображені товари, які мають знижку. Для зацікавлення користувача під кожним товаром відображається як стара ціна, так і нова з урахуванням знижки. Це підкреслює вигоду для покупця та стимулює швидше прийняти рішення щодо покупки. Кожний товар містить також зображення, короткий опис та теги.

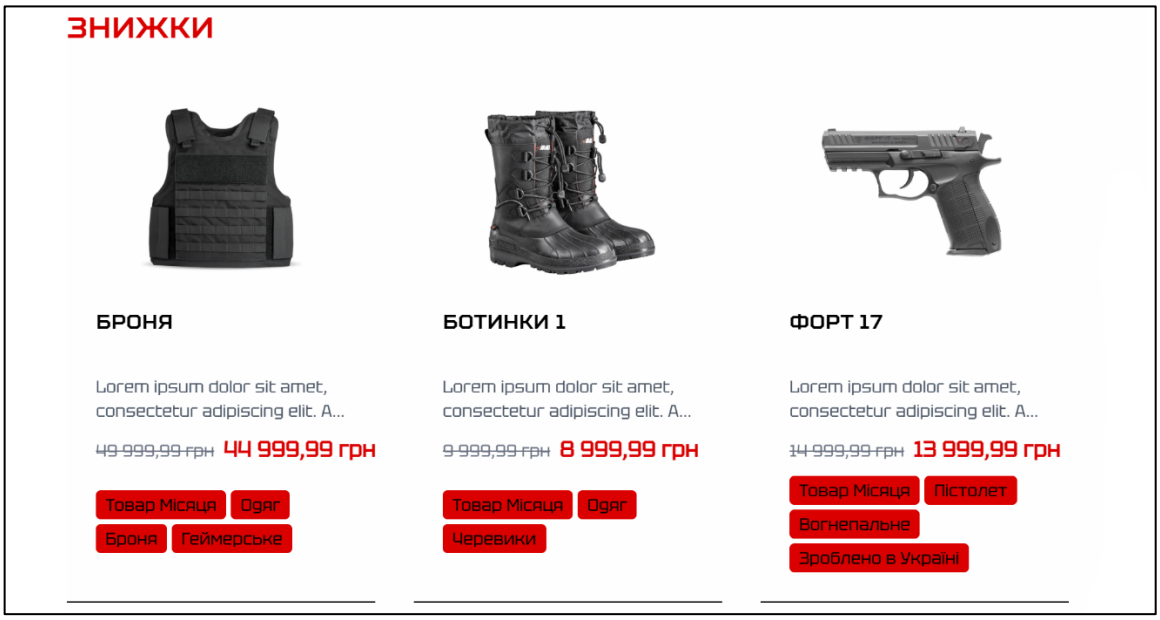


Рисунок 3.20 – Товари зі знижкою

Після блоку зі знижками на головній сторінці розташовано два окремих списки товарів. Перший «Зброя» (див. рис. 3.21), а другий «Не зброя» (див. рис. 3.22).

У блоці «Зброя» показано товари, які відповідають відповідній категорії – пістолети, ножі, тощо. А у блоці «Не зброя» елементи екіпірування, аксесуари, рюкзаки, тощо.

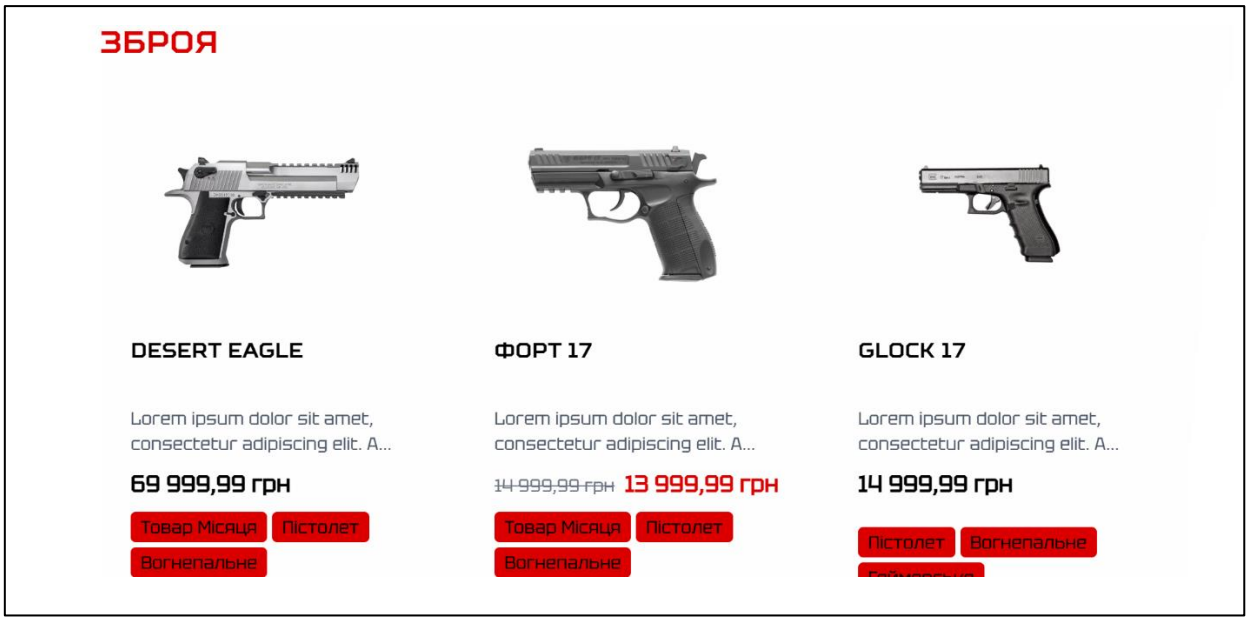


Рисунок 3.21 – Блок товарів категорії «Зброя»

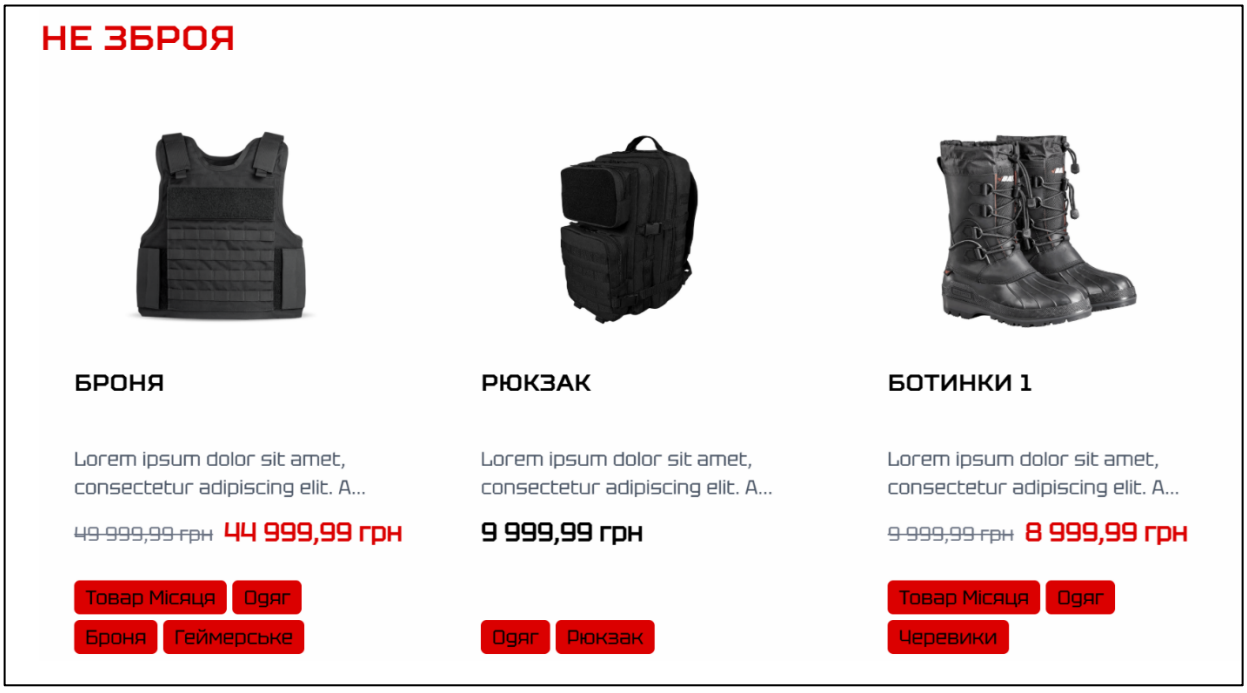


Рисунок 3.22 – Блок товарів категорії «Не зброя»

Вибірка здійснюється за допомогою категоризації товарів у базі даних, а дані передаються через відповідний REST-запит. Як і минулі блоки вони реалізовані у вигляді горизонтальної прокрутки товарних карток. Кожен товар містить зображення, назву, опис, ціну, а також відповідні теги.

Меню навігації розташоване у верхній частині вебзастосунку. У меню знаходяться такі елементи (див. рис. 3.23):

– пошук;
– каталог;
– кабінет (особистий кабінет користувача);
– про нас.

Кожен з цих елементів відкривається в окремій сторінці.

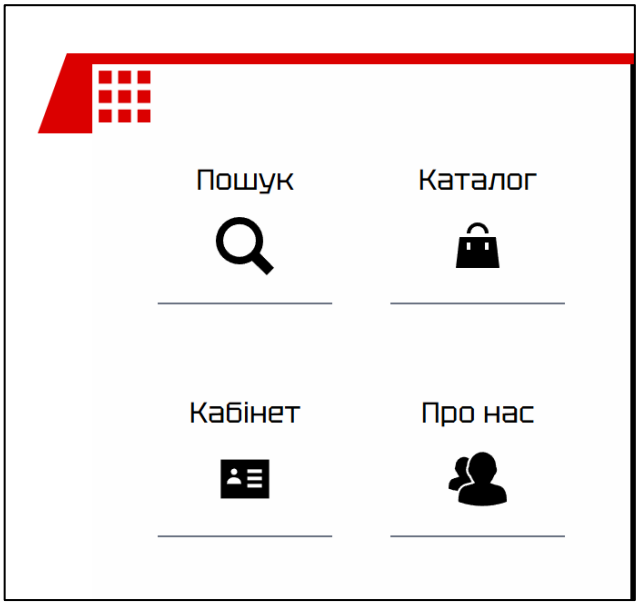


Рисунок 3.23 – Блок меню

Вебзастосунок має окрему сторінку для пошуку товарів за категоріями та тегами (див. рис. 3.24). Це дозволяє користувачеві швидше знайти потрібний йому товар серед широкого асортименту магазину без необхідності передивлятись увесь каталог.

На сторінці пошуку реалізовано дві основні фільтрації:

– за категоріями;
– за тегами.

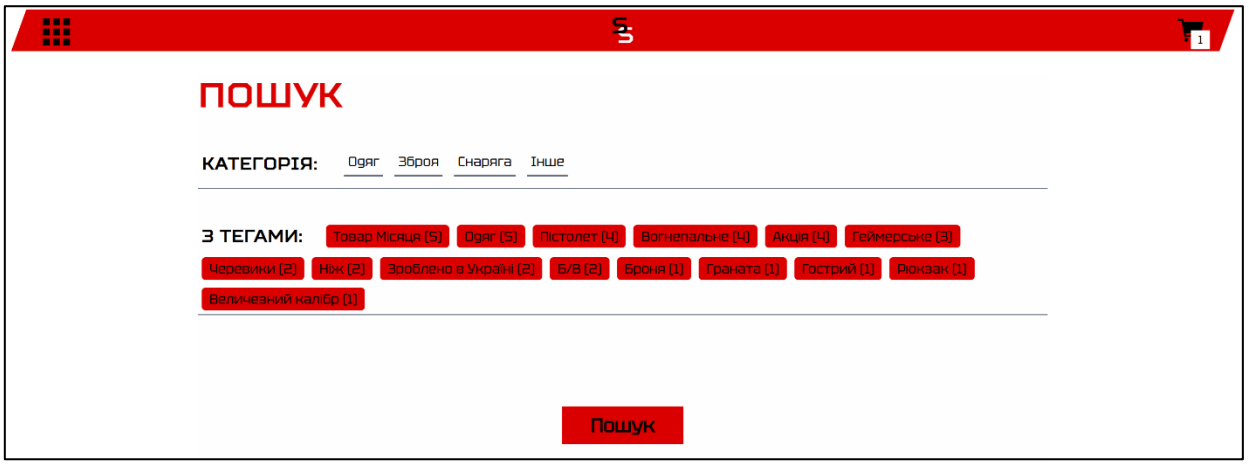


Рисунок 3.24 – Сторінка пошуку

На рисунку 3.25 зображено приклад пошуку за категорією «Зброя» та тегом «Пістолет». Як бачимо, пошук виконався без перезавантаження сторінки. Усі фільтри оновлюються динамічно. Це було реалізовано за допомогою Vue Router та реактивності Vue, що зберігає обрані фільтри в URL та полегшує повторне відкриття конкретного результату пошуку. Також реактивні змінні забезпечують миттєву відповідь інтерфейсу без додаткових запитів або перезавантажень сторінки.

Вся логіка фільтрації виконується на стороні клієнта з динамічним підвантаженням лише необхідних даних через API-запити.

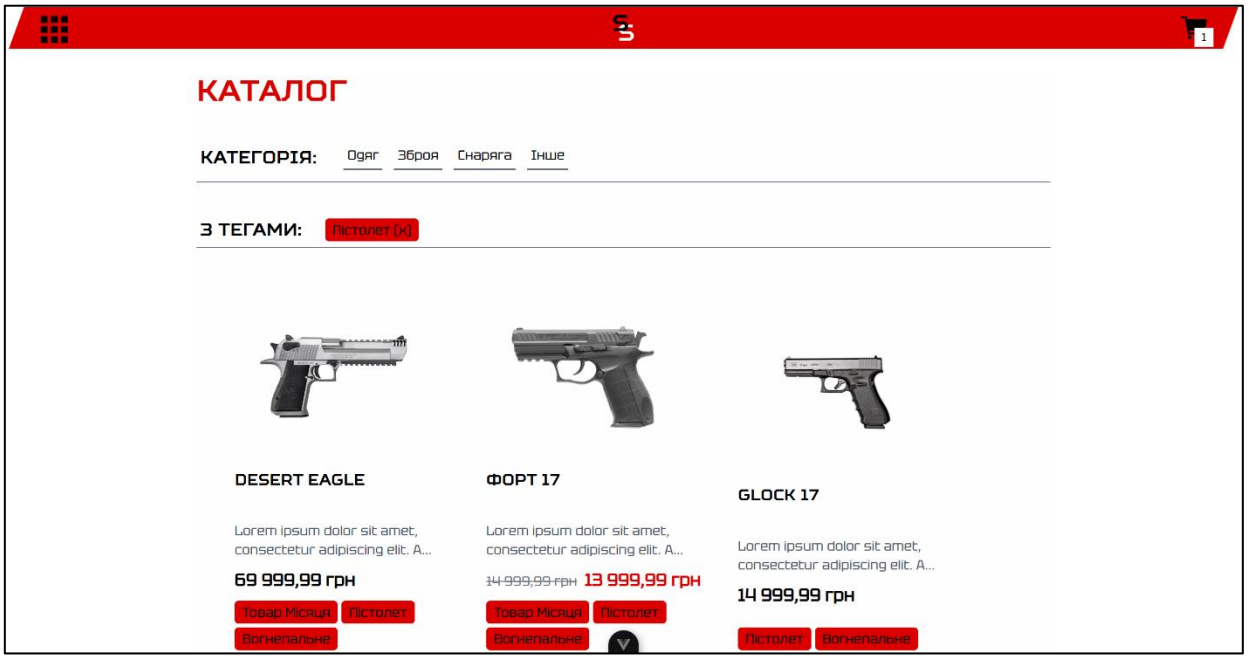


Рисунок 3.25 – Приклад пошуку за категорією та тегом

На сторінці «Каталогу» показані всі доступні користувачеві товари (див. рис. 3.26). Також користувач може обрати певну категорію товарів, наприклад «Зброя» чи «Одяг».

Товари виводяться у вигляді окремих карток з назвою, описом, цінами, зображенням та тегами. У випадку, якщо є активна знижка – нова ціна візуально відрізняється.

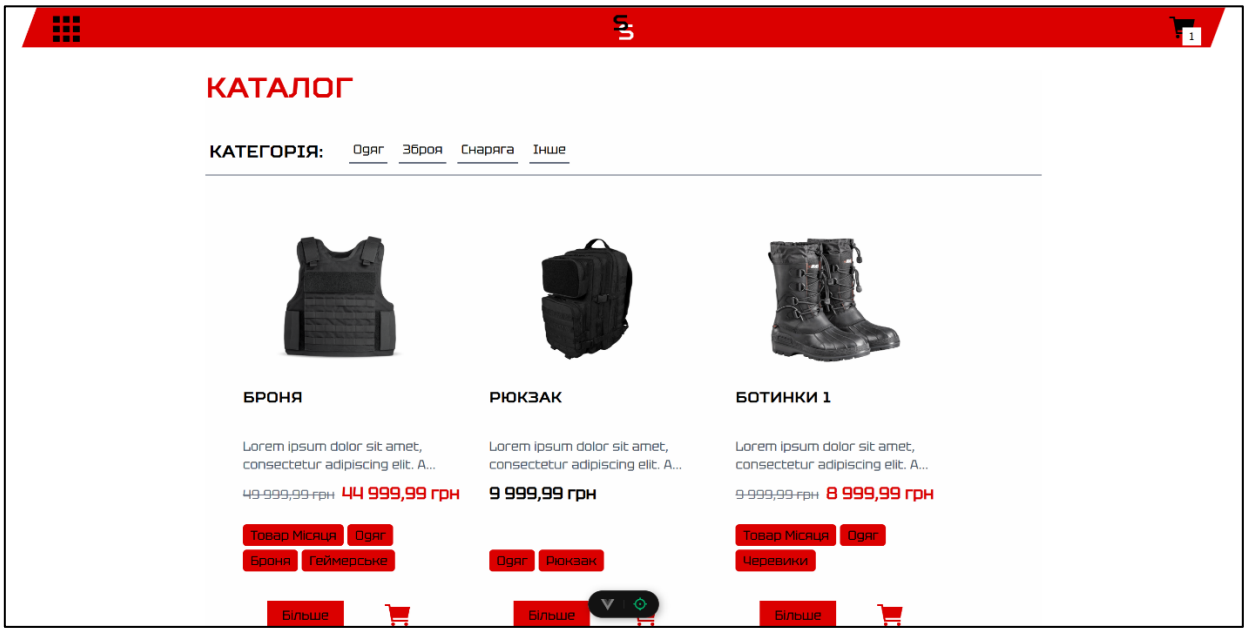


Рисунок 3.26 – Інтерфейс сторінки «Каталог»

Сторінка «Особистого кабінету» має вигляд простої та інтуїтивної форми авторизації (див. рис. 3.27). Вона містить два основних поля: введення електронної пошти та пароля. У зареєстрованих користувачів є доступ до перегляду своїх замовлень.



Рисунок 3.27 – Інтерфейс реєстрації

У нижній частині форма є посилання «Ви мене ще не знаєте». Це посилання переходить на сторінку з формою реєстрації, яка швидко створює новий обліковий запис. (див. рис. 3.28).

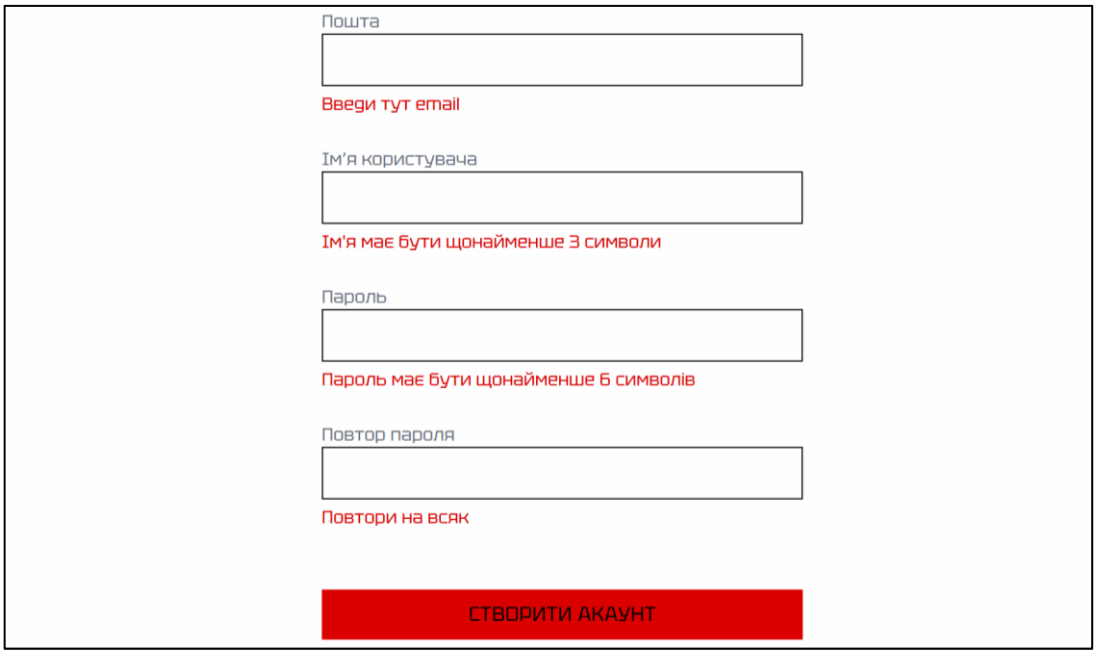


Рисунок 3.28 – Форма реєстрації нового користувача

Після успішної реєстрації, користувач потрапляє у свій кабінет (див. рис. 3.29). В кабінеті користувач може переглянути свої замовлення та їх статус.

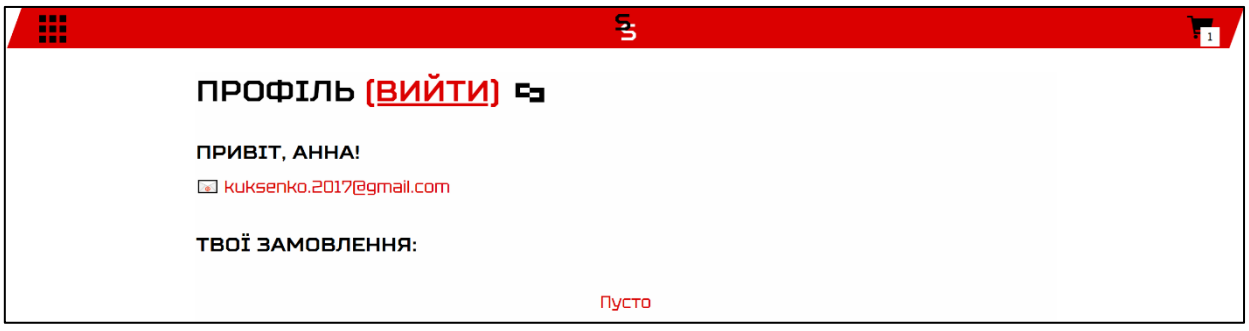


Рисунок 3.29 – Інтерфейс сторінки «Кабінету»

Після додавання товару до кошика, інтерфейс миттєво реєструє зміни – біля іконки кошика в шапці сайту з’являється лічильник із числом, який показує загальну кількість товару (див. рис. 3.30).



Рисунок 3.30 – Відображання кількості товару у кошику

Сторінка «Кошик» розроблена з досить простим та функціональним інтерфейсом (див. рис. 3.31). Користувач одразу бачить перелік з усіма доданими товарами: кожна позиція містить зображення, назву, ціну, кількість та також кнопки для збільшення або зменшення одиниць і всю суму за цей товар.

Нижче відображається загальна сума до сплати за всі товари. Якщо кошик порожній, то користувачу відображається текст із закликом повернутись до каталогу та продовжити покупки.

Якщо користувач випадково оновить сторінку кошика, то завдяки бібліотеці Pinia, в якій зміни у кошику синхронізуються з локальним сховищем браузера, дані будуть збережені.

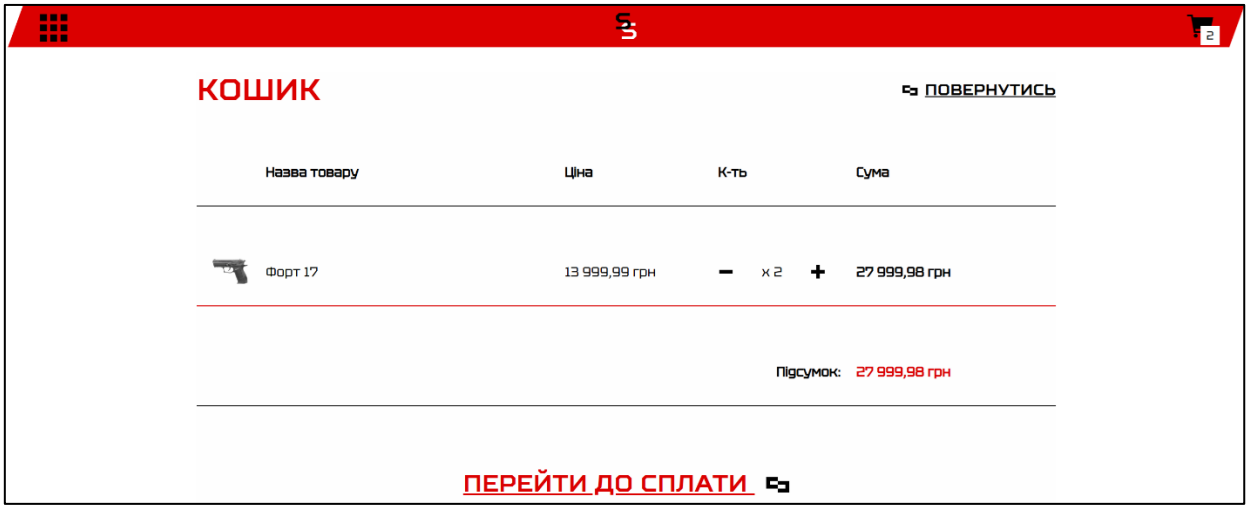


Рисунок 3.31 – Інтерфейс сторінки «Кошика»

Після того як користувач на сторінці кошика перейде до сплати, його перенаправляє на останній етап оформлення замовлення. На цій сторінці користувач бачить підтвердження, що замовлення було прийнято, і у користувача є можливість залишити додаткову контактну інформацію (див. рис. 3.32).

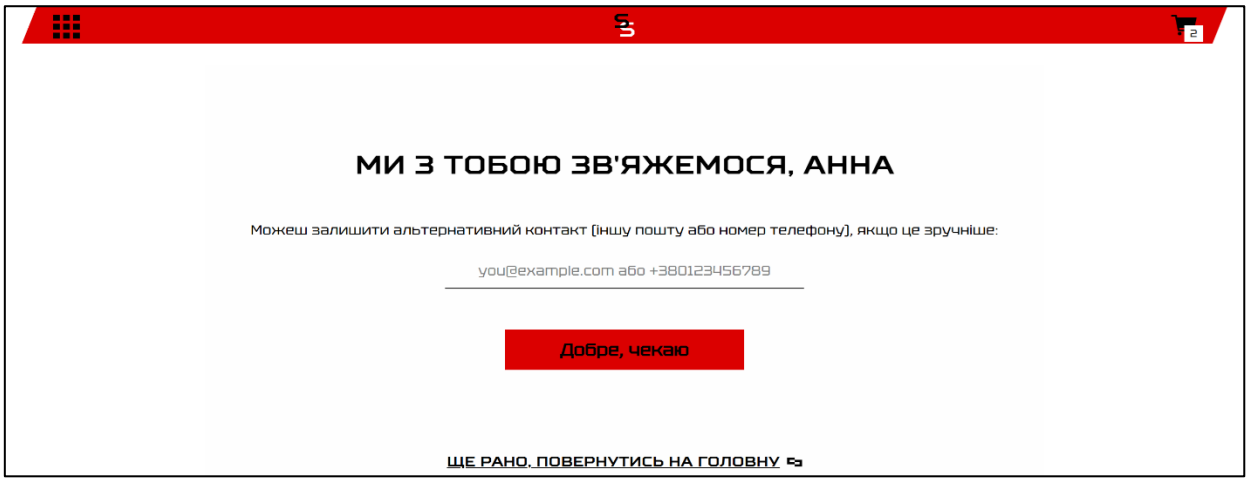


Рисунок 3.32 – Підтвердження замовлення

Після оформлення замовлення в особистому кабінеті з’являється щойно створене замовлення. У цьому блоці відображається номер замовлення, дата та час його створення, загальна сума, а також статус самого замовлення (див. рис. 3.33). Сторінка «Про нас» несе в собі інформативний контент про сам магазин (див. рис. 3.34).

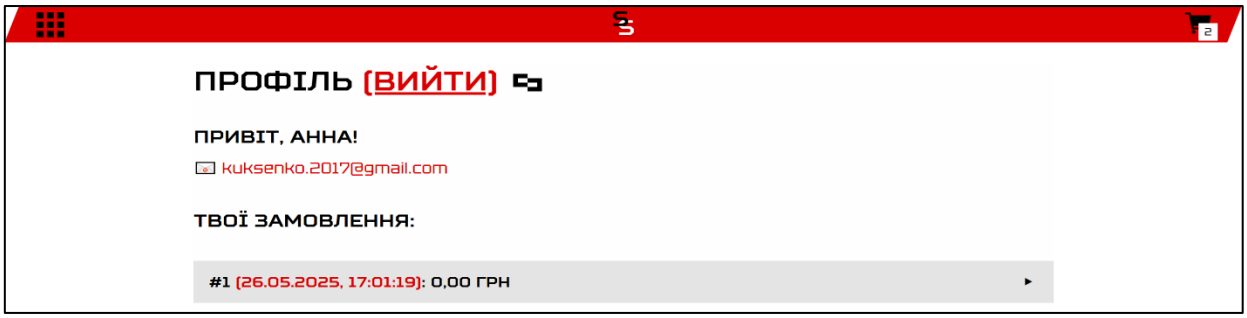


Рисунок 3.33 – Інтерфейс особистого кабінету після оформлення замовлення



Рисунок 3.34 – Інтерфейс сторінки «Про нас»

На сторінці окремного товару показано повноцінні характеристики продукту, користувач може детально ознайомитись з зовнішнім виглядом, описом та технічними параметрами. Як показано на рисунку 3.35, інтерфейс поділено на два основних блока: ліворуч – текстовий опис, а праворуч – зображення товару та відповідні теги. Також на сторінці товару є копка «Додати в кошик», яка додає цей товар до кошика.

Кожна картка товару містить в собі:

– назву товару, категорію та пов’язану з цим товаром основну інформацію;
– зображення товару;
– список тегів, яким відповідає товар;
– основні характеристику товару (наприклад, довжина, колір, вага, тощо);
– актуальну ціну;
– кнопку додати в кошик.

Такий інтерфейс допоможе користувачеві легко оцінити характеристики товару та прийняти рішення про додавання його до кошика.

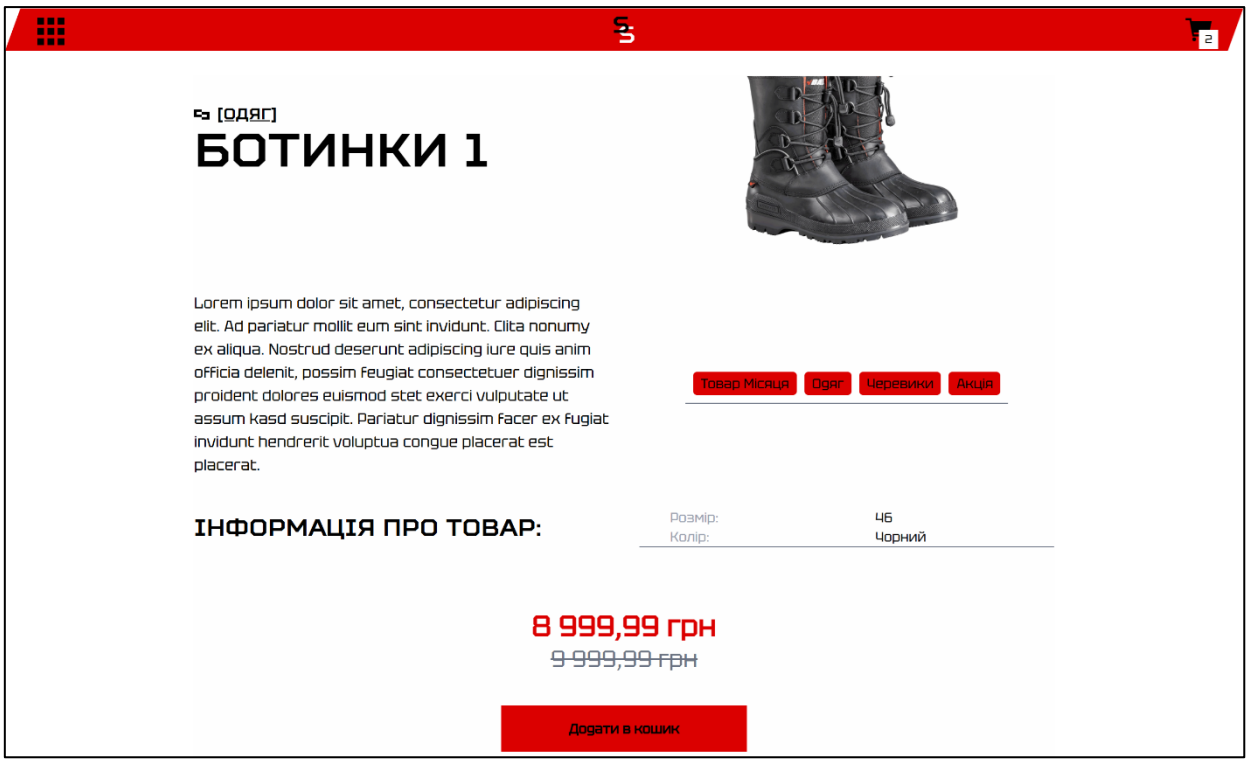


Рисунок 3.35 – Картка товару

Для того, щоб користувачеві було зручно користуватись сайтом як на комп’ютері, так і на планшеті або телефоні, інтерфейс розроблено з урахуванням адаптивного дизайну. Адаптивність було досягнена завдяки використанню Tailwind CSS.

На рисунку 3.36 зображено приклад інтерфейсу на мобільному пристрої.



Рисунок 3.36 – Відображення інтерфейсу на мобільному пристрої

3.4 Висновки до 3 розділу

У третьому розділі було розглянуто процес розробки «Магазин військового спорядження» відповідно до описаної архітектури. Показані фрагменти коду з поясненням для клієнтської частини (Vue.js) та серверної частини (Laravel), а також продемонстровано функціонал вебзастосунку.

Користувач може ознайомитись з товарами на головній сторінці, переглянути популярні товари, товари зі знижкою, категорії зброї та не зброї. Також реалізовано каталог із фільтрацією, кошиком із збереженням даних у локальному сховищі, сторінки пошуку та особистого кабінету із авторизацією. Laravel забезпечує стабільну роботу API, авторизацію, обробку замовлень та взаємодію з базою даних.

Особливу увагу приділено адаптивності сайту. Відвідування сайту однаково зручно реалізовано на мобільних та десктопних пристроях. Передача даних реалізована через props, обробку подій у компонентах, збереження стану кошика, а також взаємодію з бекендом через Axios.

Вебзастосунок «Магазин військового спорядження» є чудовим прикладом того, як за допомогою сучасних технологій можна створити зручний, швидкий і адаптивний вебсайт, який легко підтримувати та розвивати в майбутньому.

ВИСНОВКИ

Для кваліфікаційної роботи було реалізовано повноцінний вебзастосунок «Магазин військового спорядження». Вебзастосунок чітко відповідає вимогам систем електронної комерції. Проєкт охоплює весь процес розробки програмного забезпечення: від аналізу предметної області та проєктування архітектури до саме конкретної реалізації та візуального представлення вебзастосунку.

Під час написання роботи було проведено аналіз сучасних підходів до розробки вебдодатків для онлайн-торгівлі. На основі цього аналізу було обґрунтовано вибір архітектури типу «клієнт-сервер» з сегментацією на фронтенд та бекенд, в яких взаємодію відбувається через REST API. Завдяки цьому система забезпечена масштабованістю, модульністю та зручністю в її супроводі.

Було обрано технології Vue.js 3.5 на стороні клієнта, Laravel 12 (PHP 8.2) на стороні сервера та найлегшу з доступних баз даних SQLite. Tailwind CSS забезпечив вебзастосунку адаптивний та сучасний дизайн, а бібліотека Pinia – централізоване управління станом (зокрема, кошиком користувача).

На основі аналізу вимог було створено архітектурну систему, ER-діаграму сутностей, функціональну структуру застосунку, опис сутностей, модулі та маршрути та також шаблон архітектури MVC. Всі основні функції – перегляд товарів, фільтрація, кошик, інтеграція з API були реалізовані та протестовані. Код розроблено з урахуванням оптимізаційної логіки роботи з товарами, категоріями, тегами та атрибутами, а також дотримано принципи повторного використання коду та чистої архітектури.

Таким чином, мета дослідження досягнута, а всі поставлені завдання в вступі – виконані. Готовий вебзастосунок «Магазин військового спорядження» вийшов зручним, сучасним та легко масштабованим, що дає змогу у майбутньому доповнювати його новими функціям.

ПЕРЕЛІК ПОСИЛАНЬ

1. Карпенко М. Ю. Електронна комерція. Харків: ХНУМГ ім. О. М. Бекетова, 2021. 146 с.

2. Vue.js Team. Introduction – Vue.js Guide. *Vue.js*. URL: https://vuejs.org/guide/introduction.html (дата звернення: 25.04.2025).

3. Laravel documentation. *Laravel*. URL: https://laravel.com/docs/12.x (дата звернення: 25.04.2025).

4. Top 10 Benefits of Laravel Framework. *Benefits Laravel*. URL: https://www.moontechnolabs.com/blog/laravel-benefits/ (дата звернення: 29.04.2025).

5. SQLite. *About SQLite*. URL: https://sqlite.org/about.html (дата звернення: 26.04.2025).

6. What is Entity Relationship Diagram (ERD)? *ERD*. URL: https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/ (дата звернення: 29.04.2025).

7. ASP.NET MVC overview. *MVC*. URL: https://dotnet.microsoft.com/en-us/apps/aspnet/mvc (дата звернення: 02.04.2025).

8. Tailwind CSS documentation. *Tailwind*. URL: https://tailwindcss.com/docs/styling-with-utility-classes (дата звернення: 11.05.2025).

9. Laravel Eloquent ORM. *Laravel*. URL: https://laravel.com/docs/12.x/eloquent (дата звернення: 06.05.2025).

ДОДАТОК А

Код основних елементів вебзастосунку

A.1 Api.js

```
import axios from 'axios'

const api = axios.create()

api.defaults.withCredentials = true
api.defaults.withXSRFToken = true

api.interceptors.request.use((config) => {
  config.baseURL = import.meta.env.VITE_API_BASE_URL

  return config
})

api.interceptors.response.use(
  (response) => response,

  async error => {
    const config = error.config;

    if (error.response?.status === 419 && config && !config._retry) {
      config._retry = true;

      try {
        await api.get('csrf-cookie');

        return api.request(config);
      } catch (e) {
        return Promise.reject(e);
      }
    }

    return Promise.reject(error);
  }
);


export default api
```

A.2 App.vue

```
<script setup>
import { RouterView } from 'vue-router'
import TopMenu from '@/components/nav/TopMenu.vue'
import { useAuthStore } from '@/stores/auth.js'
import { onMounted, watch } from 'vue'
import ws from '@/ws.js'

const auth = useAuthStore()

onMounted(async () => {
  auth.isAuthenticated &&
    ws.channel('user-' + auth.id).listen('order.updated', (event) => {
      auth.pushOrder(event.order)
    })
})

watch(() => auth.isAuthenticated, (val) => {
  val
    ? ws.channel('user-' + auth.id).listen('.order.updated', (event) => {
      auth.pushOrder(event.order)
      })
    : ws.disconnect()
})
</script>

<template>
  <div class="max-h-screen flex flex-col">
    <top-menu class="shrink-0" />

    <RouterView class="bg-black text-white h-full w-full overflow-y-auto flex-1 no-scrollbar" />
  </div>
</template>

<style>
body {
  background-color: black;
  color: white;
}

.no-scrollbar::-webkit-scrollbar {
  display: none;
}

.no-scrollbar {
  -ms-overflow-style: none;
  scrollbar-width: none;
}
</style>
```

A.3 CartView.vue

```
<script setup>
import { useRouter } from 'vue-router'
import CartItem from '@/components/shop/Cart/CartItem.vue'
import CartHead from '@/components/shop/Cart/CartHead.vue'
import CartFoot from '@/components/shop/Cart/CartFoot.vue'
import { useCartStore } from '@/stores/cart.js'

const router = useRouter()
const cart = useCartStore()

const goBack = () => {
  router.previousRoute && router.previousRoute !== router.currentRoute
    ? router.go(-1)
    : router.push('/')
}
</script>

<template>
  <div class="max-w-4xl mx-auto">
    <div class="flex items-center justify-between mb-8">
      <h1 class="text-3xl text-main">Кошик</h1>

      <a @click="goBack" href="#" class="text-right">Повернутись</a>
    </div>

    <div v-if="cart.itemsCount > 0" class="w-full">
      <cart-head />

      <cart-item
        v-for="item in cart.allItems"
        :key="item.name"
        :quantity="item.quantity"
        :image-source="item.imageSource"
        :price="item.price"
        :name="item.name"
        @add="cart.addItem"
        @remove="cart.removeItem"
      />

      <cart-foot :total="cart.total" />

      <div class="my-16 flex justify-center">
        <router-link
          to="/checkout"
          class="active:opacity-75 cursor-pointer underline text-main font-heads text-xl
sm:text-2xl"
        >
          Перейти до сплати
        </router-link>
```

```
      </div>
    </div>

    <div v-else class="flex flex-col justify-center">
      <p class="text-2xl mb-8">Порожньо :/ Додай товари до кошику та повертайся</p>

      <router-link to="/catalog">Закупитися на славу</router-link>
    </div>
  </div>
</template>

<style scoped></style>
```

A.4 MainPage.vue

```
<script setup>
import InlineAd from '@/components/ads/InlineAd.vue'
import ShopItemCard from '@/components/shop/ShopItemCard.vue'
import { useRouter } from 'vue-router'
import { onMounted, ref } from 'vue'
import api from '@/api.js'

const router = useRouter()

const monthlyItems = ref([]);
const discountedItems = ref([]);
const weapons = ref([]);
const others = ref([]);

onMounted(() => {
  api.get('landing').then(
    (response) => {
      monthlyItems.value = response.data.items_month;
      discountedItems.value = response.data.items_discounts;
      weapons.value = response.data.weapons;
      others.value = response.data.not_weapons;
    },
    (error) => {
      console.log(error)
    },
  )
})

const goToTag = (tag) => router.push({ path: '/catalog', query: { tags: [tag.name] } })
</script>

<template>
```

```
<div class="max-w-4xl mx-auto">
  <h1 class="text-3xl mb-8 text-main">SLAVA SHOP — Озбройся на славу!</h1>

  <p>
    Вітаємо у нашому магазині! Ми продаємо найкращі товари на світі за адекватними цінами!
  </p>

  <div class="text-main text-right py-8">
    <p>Приймаємо Monero, BTC, ETH та єГривню :)</p>

    <a href="#" class="text-right">Цікавить? Пиши!</a>
  </div>

  <inline-ad />

  <h2 class="text-2xl mt-12 mb-8 text-main">Товари місяця</h2>

  <div class="flex gap-8 mb-8 overflow-x-scroll no-scrollbar">
    <shop-item-card
      v-for="item in monthlyItems"
      :key="item.name"
      :id="item.id"
      :image-source="item.main_image"
      :name="item.name"
      :description="item.description"
      :price="item.price"
      :new-price="item.new_price"
      :tags="item.tags"
      @tag-click="goToTag"
    />
  </div>

  <router-link :to="{ path: '/catalog', query: { tags: ['Товар Місяця'] } }">Дивитися всі акційні пропозиції</router-link>

  <h2 class="text-2xl mt-12 mb-8 text-main">Знижки</h2>

  <div class="flex gap-8 mb-8 overflow-x-scroll no-scrollbar">
    <shop-item-card
      v-for="item in discountedItems"
      :key="item.name"
      :id="item.id"
      :image-source="item.main_image"
      :name="item.name"
      :description="item.description"
      :price="item.price"
      :new-price="item.new_price"
      :tags="item.tags"
      @tag-click="goToTag"
    />
  </div>
```

```
    <router-link :to="{ path: '/catalog', query: { tags: ['Акція'] } }">Дивитися всі акційні пропозиції</router-link>

    <inline-ad class="my-12" />

    <h2 class="text-2xl mt-12 mb-8 text-main">Зброя</h2>
    <div class="flex gap-8 mb-8 overflow-x-scroll no-scrollbar">
      <shop-item-card
        v-for="item in weapons"
        :key="item.name"
        :id="item.id"
        :image-source="item.main_image"
        :name="item.name"
        :description="item.description"
        :price="item.price"
        :new-price="item.new_price"
        :tags="item.tags"
        @tag-click="goToTag"
      />
    </div>
```

A.5 ProfileView.vue

```
<script setup>
import { useAuthStore } from '@/stores/auth.js'
import { onMounted } from 'vue'
import router from '@/router/index.js'
import OrderComponent from '@/components/orders/OrderComponent.vue'

const auth = useAuthStore()

onMounted(async () => {
  await auth.requestUser(true)

  if (!auth.isAuthenticated) {
    await router.push('/auth')
  }
})
</script>

<template>
  <div class="max-w-4xl mx-auto">
    <div v-if="auth.isAuthenticated">
      <h1 class="text-3xl mb-8 text-white">
        Профіль <a href="#" class="text-main" @click="auth.logout">(вийти)</a>
      </h1>

      <h2 class="text-xl mb-2">
```

```
            {{ 'Привіт, ' + (auth.username || 'Ноунейм') + '!' }}
          </h2>

          <p class="mb-8 text-lg text-main">✉ {{ auth.email || 'Пошта' }}</p>

          <h2 class="text-xl mb-8">Твої замовлення:</h2>

          <div class="flex flex-col gap-8 mb-8 overflow-x-scroll no-scrollbar">
            <div v-if="auth.orders.length === 0">
              <p class="text-lg text-main self-center justify-self-center">Пусто</p>
            </div>

            <order-component v-for="order in auth.sortedOrders" :key="order.id" :order="order" />
          </div>
        </div>
      </div>
    </template>

    <style scoped></style>


        <router-link :to="{ path: '/catalog', query: { category: 'Зброя' } }">Дивитися всі товари</router-link>

        <h2 class="text-2xl mt-12 mb-8 text-main">Не зброя</h2>
        <div class="flex gap-8 mb-8 overflow-x-scroll no-scrollbar">
          <shop-item-card
            v-for="item in others"
            :key="item.name"
            :id="item.id"
            :image-source="item.main_image"
            :name="item.name"
            :description="item.description"
            :price="item.price"
            :new-price="item.new_price"
            :tags="item.tags"
            @tag-click="goToTag"
          />
        </div>

        <router-link :to="{ path: '/catalog' }">Дивитися всі товари</router-link>

        <inline-ad class="my-12" />

        <h2 class="text-2xl mt-12 mb-8 text-main">Корисні посилання</h2>

        <inline-ad class="my-12" />
        <inline-ad class="my-12" />
        <inline-ad class="my-12" />
        <inline-ad class="my-12" />
      </div>
```

```
</template>

<style scoped></style>
```

A.6 SearchView.vue

```
<script setup>
import ShopTag from '@/components/shop/ShopTag.vue'
import api from '@/api.js'
import { computed, onMounted, ref, watch } from 'vue'
import SelectOne from '@/components/elements/SelectOne.vue'
import { useRouter } from 'vue-router'

const router = useRouter()

const tags = ref([])
const selectedTags = ref([])
const categories = ref([])
const selectedCategory = ref(null)
const parameters = ref([])

const fetchTags = async () => {
  tags.value = []

  return api
    .get('tags', {
      params: selectedCategory.value ? { category: selectedCategory.value } : {},
    })
    .then(
      (response) => {
        response.data.forEach((tag) => {
          tags.value.push(tag)
        })
      },

      (error) => {
        console.log(error)
      },
    )
}

onMounted(async () => {
  api.get('categories').then(
    (response) => {
      response.data.forEach((category) => {
        categories.value.push(category.name)

        category.parameters?.forEach((parameter) => {
```

```
            parameters.value.push({
              name: parameter.name,
              category: category.name,
              min: parameter.min,
              max: parameter.max,
              unit: parameter.unit,
            })
          })
        })
      },

      (error) => {
        console.log(error)
      },
    )

    await fetchTags()
  })

  const sortedTags = computed(() => {
    return [...tags.value].sort((a, b) => b.count - a.count)
  })

  const updateTagsOnCategoryChange = async () => {
    await fetchTags()

    selectedTags.value = selectedTags.value.filter((selectedTag) =>
      tags.value.some((tag) => tag.name === selectedTag),
    )
  }

  const selectTag = (tag) => {
    tag.isSelected
      ? selectedTags.value.splice(selectedTags.value.indexOf(tag.name), 1)
      : selectedTags.value.push(tag.name)
  }

  const isTagSelected = (tag) => selectedTags.value.includes(tag)

  const parametersForCategory = computed(() => {
    return parameters.value.filter((parameter) => parameter.category ===
selectedCategory.value)
  })

  watch(selectedCategory, updateTagsOnCategoryChange)

  const search = () => {
    router.push({
      path: '/catalog',
      query: {
        category: selectedCategory.value,
        tags: selectedTags.value,
```

```
    },
   })
  }
 </script>

 <template>
  <div class="max-w-4xl mx-auto flex flex-col">
   <h1 class="text-4xl mb-8 text-main">Пошук</h1>

   <select-one name="Категорія" :values="categories" v-model="selectedCategory" />

   <div v-if="selectedCategory" class="mb-8 border-b border-gray-500">
    <h2 class="text-lg mb-4">Параметри:</h2>

    <div v-for="parameter in parametersForCategory" :key="parameter.name" class="text-sm">
     {{
      parameter.name + ': ' + (
       parameter.unit
       ? (parameter.min + ' - ' + parameter.max + ' ' + parameter.unit)
       : 'DROPDOWN'
      )
     }}
    </div>
   </div>

   <div class="mb-8 p-1 flex flex-wrap items-center gap-2 border-b border-gray-500">
    <h2 class="text-lg mr-4">3 тегами:</h2>

    <shop-tag
     v-for="tag in sortedTags"
     :key="tag.name"
     :tag="tag.name"
     :count="tag.count"
     show-count
     is-selectable
     :is-selected="isTagSelected(tag.name)"
     is-clickable
     @clicked="selectTag"
    />
   </div>

   <button
    class="active:opacity-60 cursor-pointer w-32 h-10 my-16 text-xl bg-main font-extrabold mx-auto"
    @click="search"
   >
    Пошук
   </button>
  </div>
 </template>

 <style scoped></style>
```

A.7 Main.js

```
import './assets/main.css'

import { createApp } from 'vue'
import { createPinia } from 'pinia'

import App from './App.vue'
import router from './router'

const app = createApp(App)

app.use(createPinia())
app.use(router)

app.mount('#app')
```

A.8 Api.php

```
<?php

use App\Http\Controllers\CreateOrderController;
use App\Http\Controllers\GetUserController;
use App\Http\Controllers\IndexCategoriesController;
use App\Http\Controllers\IndexItemsController;
use App\Http\Controllers\IndexTagsController;
use App\Http\Controllers\MainPageController;
use App\Http\Controllers\ShowAdController;
use App\Http\Controllers\ShowItemController;
use App\Http\Controllers\SignInController;
use App\Http\Controllers\SignOutController;
use App\Http\Controllers\SignUpController;
use Illuminate\Support\Facades\Route;
use Laravel\Sanctum\Http\Controllers\CsrfCookieController;

Route::get('/user', GetUserController::class)->middleware('auth:sanctum');

Route::get('csrf-cookie', [CsrfCookieController::class, 'show']);

Route::post('/sign-up', SignUpController::class);

Route::post('/sign-in', SignInController::class);

Route::delete('/sign-out', SignOutController::class);

Route::get('/landing', MainPageController::class);
```

```
Route::get('/goods', IndexItemsController::class);

Route::get('/goods/{item}', ShowItemController::class);

Route::post('/order', CreateOrderController::class);

Route::get('/categories', IndexCategoriesController::class);

Route::get('/tags', IndexTagsController::class);

Route::get('/ad', ShowAdController::class);
```

A.9 ShopItem.php

```
<?php

namespace App\Models;

use Closure;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;
use Illuminate\Database\Eloquent\SoftDeletes;
use Illuminate\Database\Eloquent\Builder;

class ShopItem extends Model
{
   use HasFactory;

   use SoftDeletes;

    protected $fillable = [
       'name',
     'price',
     'new_price',
   'description',
 'main_image',
  ];

public $with = [
    'shopAttributes',
    'mainTags',
 ];

 public function tags(): BelongsToMany
 {
     return $this
      ->belongsToMany(ShopTag::class)
```

```
        ->withPivot('is_displayed');
}

  public function mainTags(): BelongsToMany
{
 return $this
    ->tags()
    ->wherePivot('is_displayed', true);
    }

   public function categories(): BelongsToMany
 {
 return $this
   ->belongsToMany(ShopCategory::class)
 ->withPivot('is_main');
    }

  public function mainCategory(): BelongsToMany
 {
   return $this
     ->categories()
    ->wherePivot('is_main', true)
   ->limit(1);
 }

 public function shopAttributes(): BelongsToMany
{
  return $this
    ->belongsToMany(ShopItemAttribute::class)
    ->withPivot('value');
}

public function getValue(string|ShopItemAttribute $attribute): mixed
{
  if ($attribute instanceof ShopItemAttribute) {
    $attribute = $attribute->name;
  }

 return $this
    ->shopAttributes
    ->firstWhere('name', $attribute)
    ?->value;
  }

 public function getValues(): array
{
  return $this
 ->shopAttributes
 ->mapWithKeys(function (ShopItemAttribute $attribute) {
     return [$attribute->name => $attribute->value];
 })
 ->toArray();
```

```
 }

  public function scopeWhereAttribute(
     Builder                $query,
 string|ShopItemAttribute $attribute,
 mixed                    $operator,
 mixed                    $value = null,
): Builder
{
  if ($attribute instanceof ShopItemAttribute) {
          $attribute = $attribute->name;
   }

    if (is_null($value)) {
  $value = $operator;

   $operator = '=';
  }

  $value = (string)$value;

  return $query
            ->whereHas('shopAttributes', function (Builder $query) use ($attribute,
$operator, $value) {
        $query
           ->where('name', $attribute)
            ->wherePivot('value', $operator, $value);
     });
 }

 public function scopeWhereAttributeBetween(
      Builder                $query,
     string|ShopItemAttribute $attribute,
   mixed                    $start,
    mixed                    $end,
 ): Builder
{
 if ($attribute instanceof ShopItemAttribute) {
    $attribute = $attribute->name;
 }

 $start = (string)$start;
 $end = (string)$end;

 return $query
            ->whereHas('shopAttributes', function (Builder $query) use ($attribute, $start,
$end) {
      $query
            ->where('name', $attribute)
            ->wherePivotBetween('value', $start, $end);
```

```
});
  }

public function scopeByCategory(Builder $query, ShopCategory|string|null $category): Builder
{
if (is_null($category)) {
   return $query;
}

  if ($category instanceof ShopCategory) {
    $category = $category->name;
  }

            return $query->whereHas('categories', function (Builder $query) use ($category) {
    $query->whereName($category);
  });
 }

 public function scopeByTags(Builder $query, array|ShopTag|null $tags): Builder
 {
    if (empty($tags)) {
       return $query;
    }

 if ($tags instanceof ShopTag) {
    $tags = [$tags->name];
 }

return $query->whereHas('tags', function ($query) use ($tags) {
  $query->whereIn('name', $tags);
}, '=', count($tags));
}

public static function cardMapper(): Closure
{
  return function (self $item): array {
          return [
  'id' => $item->id,
  'name' => $item->name,
  'price' => $item->price,
  'new_price' => $item->new_price,
  'description' => $item->description,
  'main_image' => $item->main_image,
  'tags' => $item->mainTags->pluck('name'),
 ];
};
}
}
```

A.10 ShopItemAttribute.php

```
<?php

namespace App\Models;

use App\Enum\AttributeValueType;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;

class ShopItemAttribute extends Model
{
    use HasFactory;

    public $timestamps = false;

    protected $fillable = [
        'name',
        'value_type',
    ];

    protected function casts(): array
    {
        return [
            'name' => 'string',
            'value_type' => AttributeValueType::class,
        ];
    }

    public function getValueAttribute(): mixed
    {
        if (is_null($value = $this->pivot?->value)) {
            return null;
        }

        return match ($this->value_type) {
            AttributeValueType::STRING => $value,
            AttributeValueType::INTEGER => (int) $value,
            AttributeValueType::FLOAT => (float) $value,
            AttributeValueType::BOOLEAN => (bool) $value,
            default => null,
        };
    }

    public function categories(): BelongsToMany
    {
        return $this->belongsToMany(ShopCategory::class);
    }

    public function items(): BelongsToMany
```

```
    {
        return $this
            ->belongsToMany(ShopItem::class)
            ->withPivot('value');
    }
}
```

A.11 ShopTag.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;

class ShopTag extends Model
{
    use HasFactory;

    public $timestamps = false;

    protected $fillable = [
        'name',
    ];

    public function items(): BelongsToMany
    {
        return $this->belongsToMany(ShopItem::class);
    }
}
```

A.12 ShopCategory.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;

class ShopCategory extends Model
```

```
{
    use HasFactory;

    public $timestamps = false;

    protected $fillable = [
        'name',
        'unit',
    ];

    public function items(): BelongsToMany
    {
        return $this->belongsToMany(ShopItem::class);
    }

    public function shopAttributes(): BelongsToMany
    {
        return $this
            ->belongsToMany(ShopItemAttribute::class)
            ->withPivot('is_important');
    }

    public function importantshopAttributes(): BelongsToMany
    {
        return $this
            ->shopAttributes()
            ->wherePivot('is_important', true);
    }
}
```

A.13 OrderItem.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class OrderItem extends Model
{
    use HasFactory;

    public $timestamps = false;

    protected $fillable = [
        'name',
```

```
        'price',
        'quantity',
        'order_id',
    ];

    public function order(): BelongsTo
    {
        return $this->belongsTo(Order::class);
    }
}
```