МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри_____ Є. О. Давиденко
підпис
«___» _____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ІНТЕРНЕТ-МАГАЗИН СОНЦЕЗАХИСНИХ ОКУЛЯРІВ

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 408.22010824

Здобувачка ___К. С. Шкідіна
підпис
«__» _____ 2024 р.

Керівник канд. техн. наук, доцент _____ Є. О. Давиденко
підпис
«__» _____ 2024 р.

Консультант канд. техн. наук, доцент _____ А. О. Алексєєва
підпис
«__» _____ 2024 р.

м. Миколаїв – 2024 рік

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Зав. кафедри _____ Є. О. Давиденко

«06» грудня 2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи бакалавра

Видано студенту групи 408 факультету комп'ютерних наук

Шкідіній Катерині Сергіївні
(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи

Інтернет-магазин сонцезахисних окулярів

Затверджена наказом по ЧНУ від «22» грудня 2023 р. № 269

2. Строк представлення кваліфікаційної роботи «__» _____ 20___ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Очікуваним результатом роботи є готовий вебзастосунок продажу сонцезахисних окулярів

4. Перелік питань, що підлягають розробці Аналіз діяльності інтернет-магазинів. Аналіз вимог користувачів до інтернет-магазинів. Розробка дизайну інтернет-магазину. Проєктування бази даних. Розробка функціоналу вебзастосунку.

5. Перелік графічних матеріалів

Презентація

6. Завдання до спеціальної частини

Дослідження питань охорони праці, які безпосередньо пов'язані з діяльністю розробника програмного забезпечення

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Алексєєва А. О.	Кафедра екології Медичного інституту ЧНУ ім. Петра Могили	Спеціальна частина з охорони праці

Керівник роботи канд. техн. наук, доцент Давиденко Євген Олександрович

(посада, прізвище, ім'я, по батькові)

(підпис)

Завдання прийнято до виконання

Шкідіна Катерина Сергіївна

(прізвище, ім'я, по батькові студента)

(підпис)

Дата видачі завдання «06» грудня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: Інтернет-магазин сонцезахисних окулярів

№	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КРБ	05.12.2023	06.12.2023	виконано
2	Огляд літератури за темою роботи	20.01.2024	22.01.2024	виконано
3	Складання календарного плану КРБ	16.02.2024	17.02.2024	виконано
4	Аналіз предметної області	01.02.2024	18.02.2024	виконано
5	Розробка проєктних рішень	11.03.2024	24.03.2024	виконано
6	Моделювання та конструювання ПЗ	08.04.2024	29.04.2024	виконано
7	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	01.04.2024	02.06.2024	виконано
8	Розробка спеціальної частини з охорони праці	03.05.2024	10.05.2024	виконано
9	Відгук керівника КРБ	13.06.2024	13.06.2024	виконано
10	Оформлення КРБ та презентації	15.05.2024	09.06.2024	виконано
11	Попередній захист	05.06.2024	05.06.2024	виконано
12	Рецензування	13.06.2024	13.06.2024	виконано
13	Завершення оформлення КРБ та презентації	17.06.2024	17.06.2024	виконано
14	Захист кваліфікаційної роботи	25.06.2024	25.06.2024	виконано

Розробила здобувачка Шкідіна К. С. _____

(*прізвище, ім'я, по батькові)* *(підпис)*

«17» лютого 2024 р

Керівник роботи канд. техн. наук, доцент Давиденко Є. О. _____

(*посада, прізвище, ім'я, по батькові)* *(підпис)*

«17» лютого 2024 р

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра

«Інтернет-магазин сонцезахисних окулярів»

Здобувачка 408 гр.: Шкідіна Катерина Сергіївна

Керівник: канд. техн. наук, доцент Давиденко Євген Олександрович

Кваліфікаційна робота присвячена розробці програмного забезпечення для продажу сонцезахисних окулярів.

Тема роботи є актуальною, оскільки все більше людей обирають покупки в Інтернет через зручність онлайн-покупок, зокрема можливість оформити замовлення з будь-якого місця, економія часу на поїздки до фізичних магазинів, можливість порівняння цін та характеристик, доступ до повної інформаці про товар. Інтернет-магазини є ефективним рішенням для бізнесу, вони надають інструменти для ефективного управління асортиментом, аналізу даних та створення персоналізованих стратегій маркетингу, сприяючи підвищенню конкурентоспроможності.

Об’єкт роботи – процес розробки вебзастосунку для інтернет-магазину сонцезахисних окулярів.

Предмет роботи – програмні засоби та методи реалізації функціоналу та інтерфейсу інтернет-магазину.

Метою кваліфікаційної роботи є створення інтернет-магазину для продажу сонцезахисних окулярів шляхом реалізації всіх необхідних функцій для діяльності інтернет-магазину, розробки зручного інтерфейсу та шляхом використання сучасних технологій веброзробки.

Кваліфікаційна робота складається з вступу, 4 розділів, висновків та переліку джерел посилань.

У вступі визначається актуальність теми, мета, предмет та об’єкт дослідження.

У першому розділі проводиться аналіз існуючих вебзастосунків-аналогів, формується постановка задачі та специфікація вимог до програмного забезпечення.

Другий розділ присвячений розробці сценаріїв використання та моделюванню програмного забезпечення.

У третьому розділі подано огляд використаних технологій розробки, архітектура та проєктування програмного забезпечення.

У четвертому розділі описано процес розробки прграмного забезпечення та продемонтровано результати роботи.

У висновках проводиться аналіз виконаних робіт та отриманих результатів.

Кваліфікаційна робота бакалавра викладена на 68 сторінок, вона містить 4 розділи, 44 ілюстрації, 4 таблиці, 21 джерело в переліку посилань.

Ключові слова: розробка вебзастосунку, інтернет-магазин, електронна комерція, функціонал магазину, технології розробки, фреймворк Laravel.

ABSTRACT

of the Bachelor's Thesis

“Sunglasses webshop”

Student of group 408: Shkidina Kateryna Serhiivna

Supervisor: Candidate of Technical Sciences (PhD.), Associate Professor Davydenko Yevhen Oleksandrovych

The qualification work is devoted to the development of software for the sale of sunglasses.

The topic of the work is relevant because more and more people choose to shop online due to the convenience of online shopping, including the ability to place an order from anywhere, saving time on trips to physical stores, the ability to compare prices and characteristics, and access to complete product information. Online stores are an effective solution for businesses, they provide tools for effective inventory management, data analysis and creation of personalized marketing strategies, contributing to increased competitiveness. The object of the work is the process of developing a web application for an online sunglasses store.

The subject of the work is software tools and methods of implementing the functionality and interface of the online store.

The purpose of the qualification work is to create an online store for the sale of sunglasses by implementing all the necessary functions for the online store, developing a convenient interface and using modern web development technologies.

The qualification work consists of an introduction, 4 sections, conclusions and a list of reference sources.

The introduction defines the relevance of the topic, the purpose, subject and object of the research.

In the first section, an analysis of existing analogue web applications is carried out, a problem statement and a specification of software requirements are formed.

The second section is devoted to the development of usage scenarios and modeling of the software.

The third chapter provides an overview of the used development technologies, software architecture and design.

The fourth chapter describes the software development process and demonstrates the work results.

The conclusions analyze the work performed and the results obtained.

The bachelor's qualification work is laid out on 68 pages, it contains 4 sections, 44 illustrations, 4 tables, 21 sources in the list of references.

Keywords: web application development, online store, e-commerce, store functionality, development technologies, Laravel framework.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – База даних

ПЗ – Програмне забезпечення

API – Application Programming Interface

CRUD – Create, Read, Update, Delete

CSS – Cascading Style Sheet

DOM – Document Object Model

ERD – Entity Relationship Diagram

JS – JavaScript

IDE – Integrated Development Environment

PHP – Hypertext Preprocessor

MVC – Model-View-Controller

UML – Unified Modeling Language

ORM – Object-Relational Mapping

WYSIWYG – What You See Is What You Get

ВСТУП

За останні десятиліття інтернет-магазини стали невід'ємною частиною життя сучасного суспільства та економіки. Розвиток технологій, широке поширення інтернету та зміна споживчої поведінки сприяли стрімкому зростанню електронної комерції. Інтернет-магазини стали ключовим інструментом для бізнесів у плані розширення аудиторії та оптимізації продажів, електронна комерція дозволяє магазинам пропонувати широкий асортимент товарів та послуг, оскільки вони не обмежені фізичним простором. Онлайн-торгівля дозволяє навіть найменшим магазинам та брендам займати своє місце на ринку, забезпечуючи можливість конкурувати з іншими. Інтернет-магазини також значно скорочують витрати на оренду приміщень і оплату праці, що робить їх більш економічно вигідними для бізнесу. Вони дозволяють підприємствам швидко реагувати на зміну попиту та адаптувати свій асортимент у реальному часі. Автоматизація багатьох процесів, таких як управління запасами та обробка замовлень дозволяє підприємствам працювати більш ефективно, а впровадження реклам та спеціальних пропозицій допомагає утримувати існуючих клієнтів та збільшувати частоту покупок.

Сонцезахисні окуляри є не лише засобом захисту від ультрафіолетового випромінювання, але й важливим модним аксесуаром. Їх різноманітність у формах, кольорах та стилях дозволяє підкреслити індивідуальність кожної людини, що створює постійний попит на цей продукт. Цей попит стимулює розвиток онлайн-продажів, адже інтернет-магазини можуть запропонувати ширший асортимент та зручність вибору, ніж фізичні магазини.

Тема роботи є **актуальною**, оскільки все більше людей обирають покупки в Інтернет через їх зручність. Зокрема, інтернет-магазини сонцезахисних окулярів надають споживачам широкий спектр можливостей, а саме – перегляд характеристик товару, відгуків, порівняння цін, що дає змогу знайти ідеальні окуляри, які відповідають індивідуальним уподобанням. Також варто сказати про зручність процесу покупки, адже інтернет-магазини надають можливість замовити товар з будь-якого місця у будь-який час. Інтернет-магазини є ефективним

рішенням для бізнесу, відкриваючи нові перспективи для розвитку та розширення клієнтської бази, вони надають інструменти для ефективного управління асортиментом, аналізу даних та створення персоналізованих стратегій маркетингу, сприяючи підвищенню конкурентоспроможності. Усі перераховані аспекти вказують на актуальність теми кваліфікаційної роботи, оскільки вона відповідає сучасним тенденціям у розвитку ринку та враховує потреби споживачів.

Об’єкт роботи – процес розробки вебзастосунку для інтернет-магазину сонцезахисних окулярів.

Предмет роботи – програмні засоби та методи реалізації функціоналу та інтерфейсу інтернет-магазину.

Метою кваліфікаційної роботи є створення інтернет-магазину для продажу сонцезахисних окулярів шляхом реалізації всіх необхідних функцій для діяльності інтернет-магазину, розробки зручного інтерфейсу та шляхом використання сучасних технологій веброзробки.

Для досягнення поставленої мети необхідно виконати наступні завдання:

- проаналізувати специфіку діяльності інтернет-магазинів;
- проаналізувати вимоги користувачів до інтернет-магазинів;
- розробити дизайн інтернет-магазину;
- спроєктувати базу даних;
- розробити інтерфейс та функціонал вебзастосунку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Для успішної реалізації проєкту інтернет-магазину сонцезахисних окулярів необхідно провести аналіз предметної області. Це необхідно для з’ясування ключових аспектів, що впливають на вибір магазину та покупку сонцезахисних окулярів, визначення потреб та очікувань цільової аудиторії, а також виявлення переваг та недоліків існуючих рішень на ринку. Крім того, аналіз предметної області дозволить чітко визначити перелік функцій користувачів, основні вимоги до інтернет-магазинів та ключові характеристики продукції, що буде пропонуватися в створеному магазині. З цими даними можна створити конкурентоспроможний інтернет-магазин, який задовільнить потреби користувачів та забезпечить успішну реалізацію даного бізнесу.

1.1 Опис предметної області

В сучасну цифрову епоху електронна комерція стала не лише популярним, але й важливим елементом економіки. Інтернет-магазини, завдяки своїй доступності та широкому асортименту товарів, здобувають все більшу популярність серед споживачів.

Інтернет-магазин – це вебсайт або цифрова платформа, яка дозволяє компаніям або окремим особам продавати товари чи послуги через Інтернет. Клієнти можуть переглядати онлайн-каталог, вибирати товари, які вони бажають придбати, і завершувати транзакції в електронному вигляді. Інтернет-магазини стають все більш популярними завдяки зручності здійснення покупок з дому та глобальному охопленню, яке вони забезпечують [1].

Інтернет-магазини також є ефективним рішенням для бізнесу, відкриваючи нові перспективи для розвитку та розширення клієнтської бази, вони автоматизують процеси управління запасами та обробки замовлень, допомагають збільшувати клієнтську базу, зменшують витрати на оренду приміщень.

Нижче наведено деякі з найважливіших частин інтернет-магазину. Коли справа доходить до створення вебсайту електронної комерції, важливо враховувати все наведене нижче:

- списки товарів;
- кошик;
- процес оформлення замовлення;
- варіанти оплати;
- інфраструктура безпеки;
- управління замовленнями;
- облікові записи клієнтів;
- пошук і навігація [1].

Сонцезахисні окуляри є популярною сферою по декількох причинах. По-перше, вони забезпечують захист очей від шкідливого впливу ультрафіолетових променів, що сприяє збереженню здоров’я очей. По-друге, вони є модним аксесуаром, який доповнює образ. Люди використовують їх як частину свого образу, щоб виглядати стильно. Отже, потенційна аудиторія інтернет-магазинів сонцезахисних окулярів включає в себе широкий спектр споживачів.

Інтернет-магазини сонцезахисних окулярів пропонують широкий вибір товарів різних брендів, стилів і цінових категорій. Покупці можуть легко порівнювати різні моделі, користуючись фільтрами за брендом, кольором, формою оправи, матеріалом і ціною. Окрім того, інтернет-магазини зазвичай надають детальні описи товарів та фотографії, що дозволяє отримати максимально повну інформацію про товар перед покупкою. Важливою перевагою є також можливість зручної доставки товару до дверей або в пункт самовивозу, що зекономить час і зусилля клієнта.

1.2 Аналіз аналогів

Було розглянуто декілька інтернет-магазинів сонцезахисних окулярів, а саме: The Sunglasses Shop [2], Polaroid [3], Multioptic [4], для визначення основних

функцій вебзастосунку, виявлення переваг та недоліків для покращення застосунку, що розробляється. Цей етап дозволив зосередитися на важливих аспектах, які формують користувацький досвід та впливають на успішність інтернет-магазину.

Далі у цьому підрозділі представлено результати дослідження.

The Sunglasses Shop

The Sunglasses Shop є британським інтернет-магазином, спеціалізованим на продажу сонцезахисних окулярів. Він пропонує широкий вибір моделей окулярів, включаючи такі відомі бренди, як Ray-Ban, Oakley, Maui Jim, Persol та інші. Забезпечує доставку в різні країни та надає можливість повернення або обміну товару у разі необхідності.

Розробник: недоступно.

Архітектура: web application.

Мова реалізації: PHP, JavaScript.

Перелік функцій та характеристик:

– перегляд галереї товарів;
– перегляд інформації про товари;
– оформлення замовлення;
– пошук по каталогу;
– створення та редагування профілю.

Переваги:

– зручний та зрозумілий інтерфейс.

Недоліки:

– повільне завантаження сторінок;
– пошук товарів тільки за назвою, не за іншими характеристиками;
– замало атрибутів для пошуку товарів у каталозі.

Посилання: https://www.thesunglassesshop.co.uk

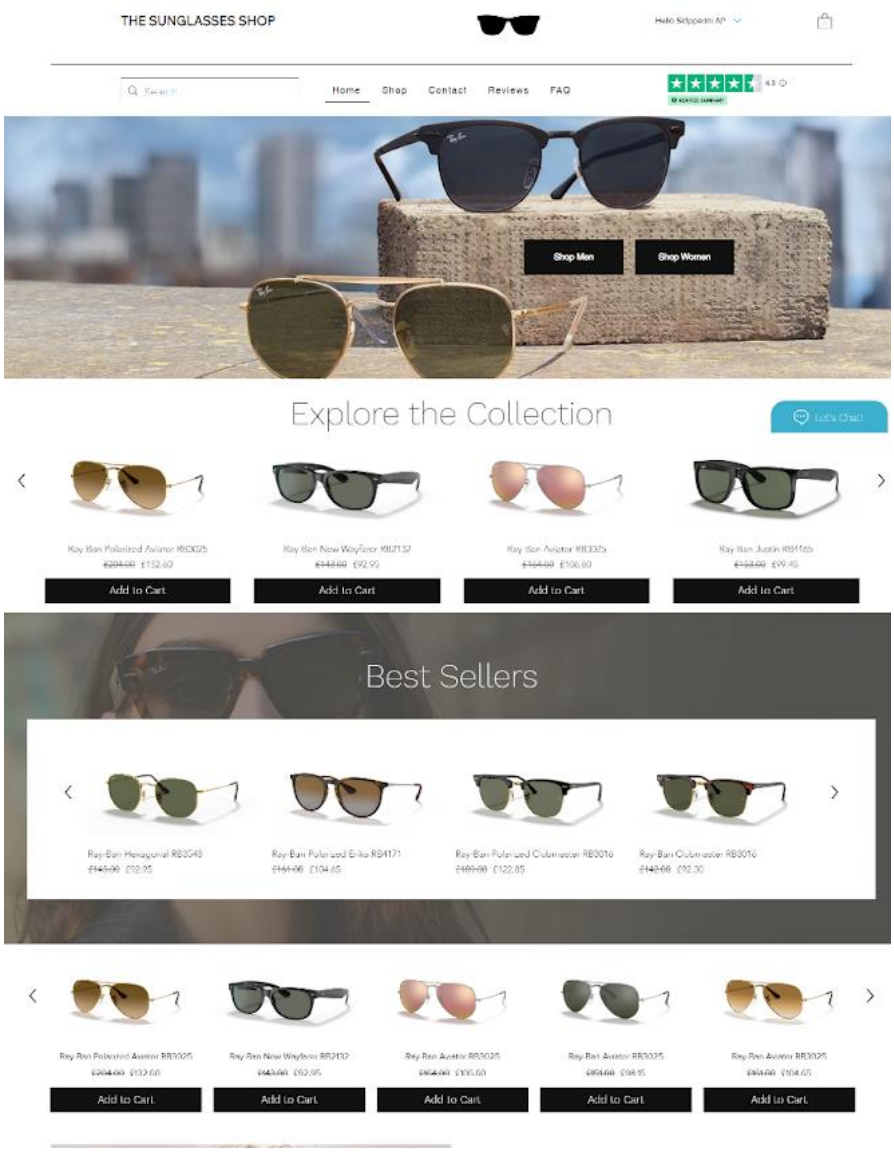


Рисунок 1.1 – Головна сторінка магазину The Sunglasses Shop

Polaroid

polaroid.in.ua – це офіційний дистриб'ютор сонцезахисних окулярів Polaroid в Україні. Polaroid – відомий бренд, який спеціалізується на виробництві сонцезахисних окулярів та інших продуктів оптики.

Розробник: недоступно.

Архітектура: web application.

Мова реалізації: PHP, JavaScript.

Перелік функцій та характеристик:

– перегляд галереї товарів;

– перегляд інформації про товари;

– формування списку бажань;
– написання відгуків;
– оформлення замовлення;
– пошук по каталогу;
– створення та редагування профілю.

Переваги:

– зручний повноцінний пошук по каталогу;
– онлайн-примірка.

Недоліки:

– невдало підібрані шрифти та кольори.

Посилання: https://polaroid.in.ua/

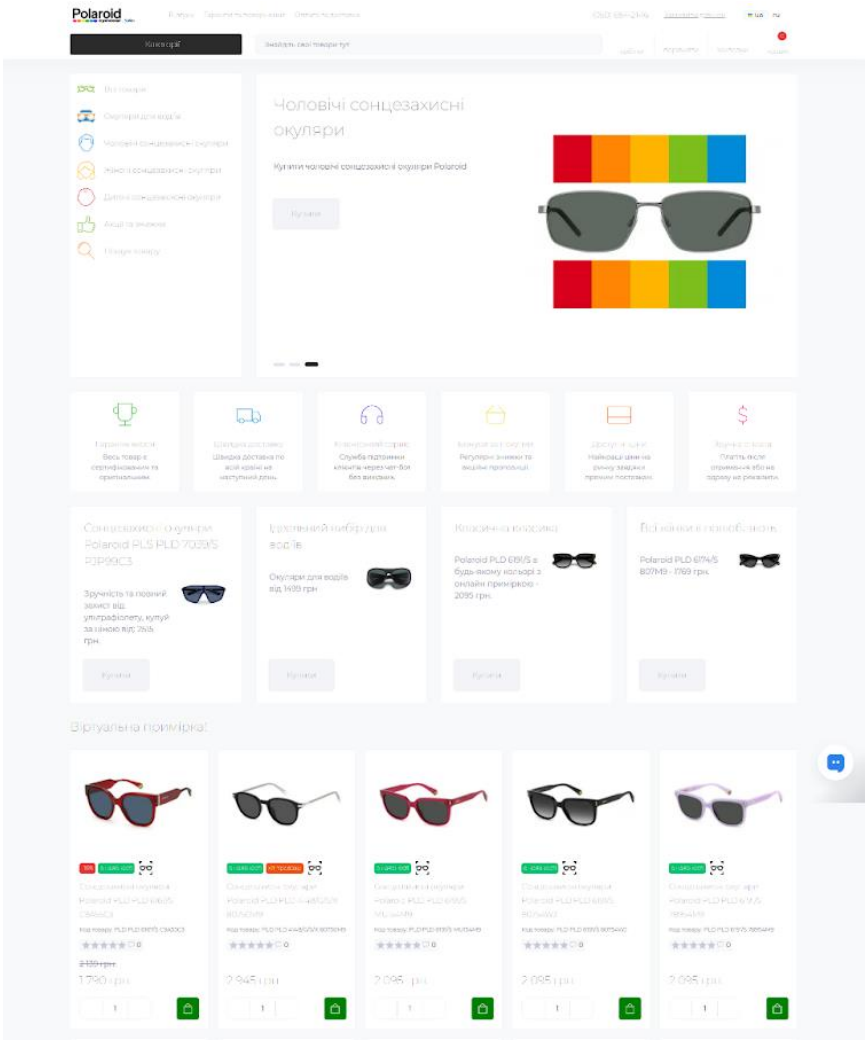


Рисунок 1.2 – Головна сторінка магазину Polaroid

Multioptic

Multioptic – це український інтернет-магазин, спеціалізований на продажі сонцезахисних окулярів. На їхньому сайті представлена тільки оригінальна продукція світових брендів, а також повна та актуальна інформація про окуляри.

Розробник: недоступно.

Архітектура: web application.

Мова реалізації: PHP, JavaScript.

Перелік функцій та характеристик:

– перегляд галереї товарів;
– перегляд інформації про товари;
– оформлення замовлення;
– пошук по каталогу;
– створення та редагування профілю.

Переваги:

– прибавливий інтерфейс.

Недоліки:

– фільтрація товарів не працює;
– відсутнє сортування товарів;
– відсутність валідації при оформленні замовлення.

Посилання: https://multioptic.com.ua

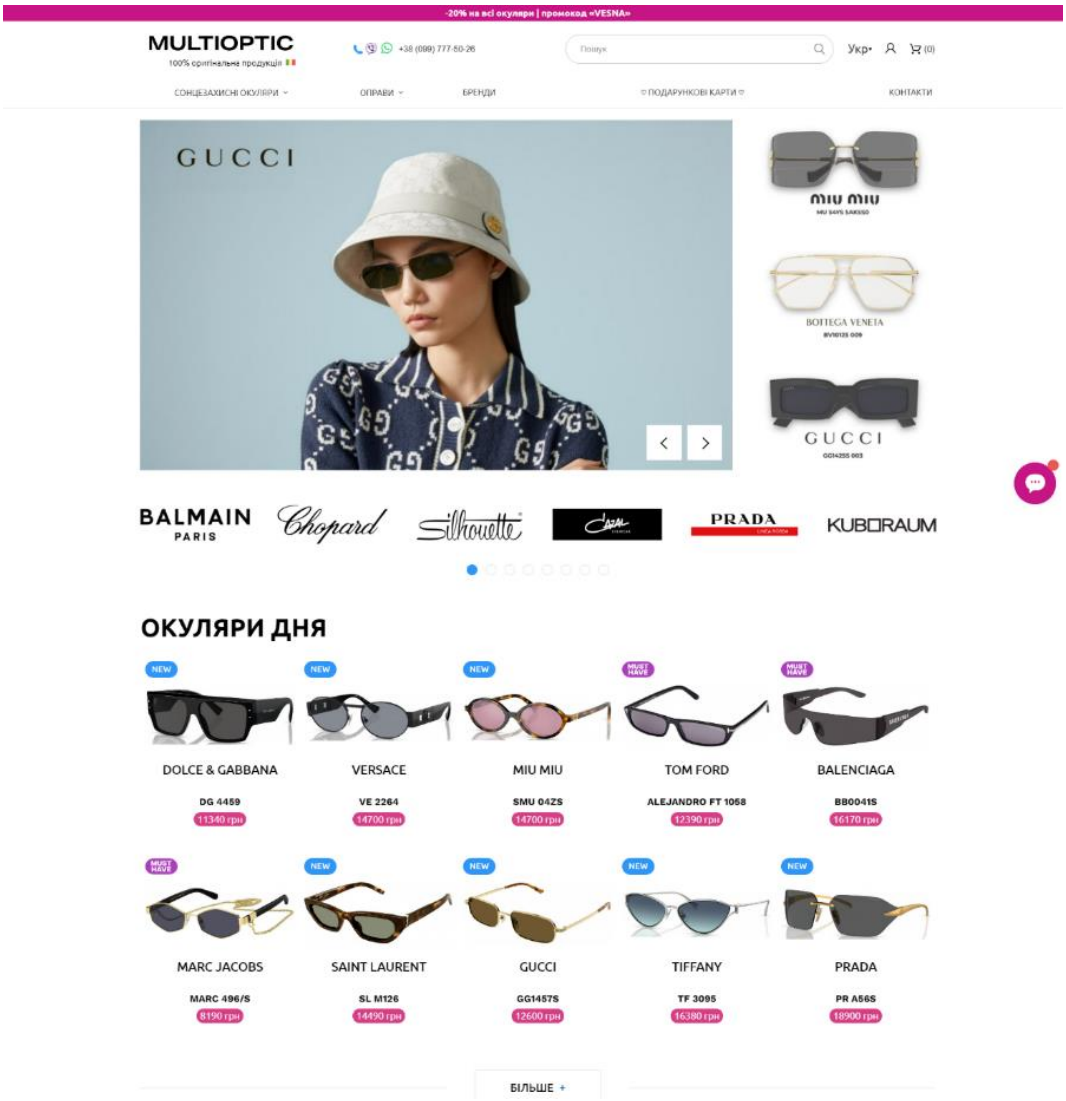


Рисунок 1.3 – Головна сторінка магазину Multioptic

1.3 Специфікація вимог

ПРИЗНАЧЕННЯ ТА МЕЖІ ПРОЄКТУ

Призначенням застосунку є вдосконалення процесу продаж сонцезахисних окулярів.

Межі проєкту ПЗ

Крайня дата завершення роботи над ПЗ – 00.00.2024

ЗАГАЛЬНИЙ ОПИС

Сфера застосування:

Представлений вебзастосунок може бути використаний у сфері продаж сонцезахисних окулярів.

Характеристика користувачів

Основні характеристики користувачів: наявність пристрою (смартфон, ноутбук, комп’ютер тощо) з доступом до Інтернет.

Загальна структура і склад системи:

а. клієнтська частина (Front-end):
- дизайн та інтерфейс вебсайту;
- клієнтська програмна логіка (Vuejs, JavaScript, jQuery).

б. серверна частина (Back-end):
- взаємодія з базою даних;
- серверна логіка (Laravel, PHP);

в. база даних:
- взаємодія з базою даних через ORM;
- збереження даних.

Загальні обмеження

Вимоги для роботи з програмним забезпеченням – наявність браузеру та інтернет-з’єднання.

ФУНКЦІЇ СИСТЕМИ

Для клієнта:

- реєстрація та авторизація;
- пошук та перегляд товарів;
- додавання товару до кошика та оформлення замовлення;
- перегляд особистого кабінету та історії замовлень;
- редагування особистої інформації.

Для менеджера та адміністратора:

- перегляд статистики продаж;
- управління товарами та атрибутами товарів, знижками, замовленнями, контактною інформацією.

Тільки для адміністратора:

- управління акаунтами користувачів;
- редагування контенту на сторінках.

ВИМОГИ ДО ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

Значних технічних обмежень для розробки даного програмного забезпечення немає. Бажано мати комп’ютер або ноутбук з мінімальною місткістю оперативної пам’яті 8 Гбайт.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Архітектура програмної системи

Система складається з клієнтської частини, серверної частини та бази даних.

Системне програмне забезпечення

Для написання клієнтської частини були використані: Blade, Vuejs, jQuery, Tailwind CSS. Для серверної: Laravel, MariaDB. В якості БД для застосунку обрано MariaDB.

Мережеве програмне забезпечення

Під час розробки було використано ОС Windows 10, для написання коду використано IDE PhpStorm, для перегляду та тестування роботи застосунку браузер Firefox.

ВИМОГИ ДО ЗОВНІШНІХ ІНТЕРФЕЙСІВ

Інтерфейс користувача

Інтерфейс системи має бути привабливим, простим та інтуїтивно зрозумілим для користувача.

Апаратний інтерфейс

Вебзастосунок має бути доступним на будь-якому пристрої з доступом до Інтернет та браузером, тому інтерфейс має бути адаптивним для різних розширень екрану.

ВЛАСТИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Доступність

Вебзастосунок повинен працювати на будь-якою пристрої з браузером та доступом до Інтернет.

Надійність:

– стабільність роботи;

– вбудовані механізми обробки можливих помилок;

– ПЗ має виключати зловживання рівнем доступу.

Безпека:

Пароль користувача хешований;

Супроводжуваність

Застосунок має бути відкритим для можливого розширення у майбутнтьому.

Переносимість

Програмне забезпечення може працювати на будь-яких операційних системах, що підтримують сучасні браузери.

Продуктивність

Продуктивність програмного забезпечення залежить від якості інтернет-з’єднання.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи бакалавра було проведено аналіз предметної області системи, що розробляється. Розглянуто вебзастосунки-аналоги, виділено їх переваги та недоліки, визначено основні функції інтернет-магазинів сонцезахисних окулярів. На основі аналізу даних аналогів було розроблено власний вебзастосунок, що поєднує найкращі риси цих платформ, а також враховано їхні недоліки.

Написано специфікацію вимог до програмного забезпечення, у якій зазначено призначення та межі проєкту, загальний опис ПЗ, функції системи, властивості ПЗ, вимоги до технічного та програмного ПЗ та вимоги до зовнішніх інтерфейсів. Це є основою для подальшої розробки та впровадження вебзастосунку.

2 МОДЕЛЮВАННЯ ВЕБЗАСТОСУНКУ

Моделювання вебзастосунку – це процес розробки архітектури, функціональності та інтерфейсу вебзастосунку перед його реалізацією. Це допомагає розробникам краще розуміти, як буде працювати вебзастосунок та як він буде взаємодіяти з користувачем.

2.1 Варіанти використання

Для демонстрації функціональності системи було розроблено діаграму варіантів використання.

Діаграма варіантів використання (Use Case Diagram) – це тип діаграми уніфікованої мови моделювання (UML), яка представляє взаємодію між акторами (користувачами або зовнішніми системами) і системою, що розглядається, для досягнення певних цілей. Вона забезпечує загальне уявлення про функціональність системи, ілюструючи різні способи взаємодії користувачів з нею [5].

Актори – це зовнішні суб'єкти, які взаємодіють із системою. Це можуть бути користувачі, інші системи або апаратні пристрої. У контексті діаграми варіантів використання актори ініціюють варіанти використання та отримують результати [5].

Варіанти використання схожі на сцени у виставі. Вони представляють конкретні речі, які може робити ваша система. У системі онлайн-покупок прикладами використання можуть бути «Розмістити замовлення», «Відстежити доставку» або «Оновити інформацію про продукт». Варіанти використання представлені овалами [5].

На рисунку 2.1 зображено діаграму використання вебзастосунку з акторами та доступними їм функціями.

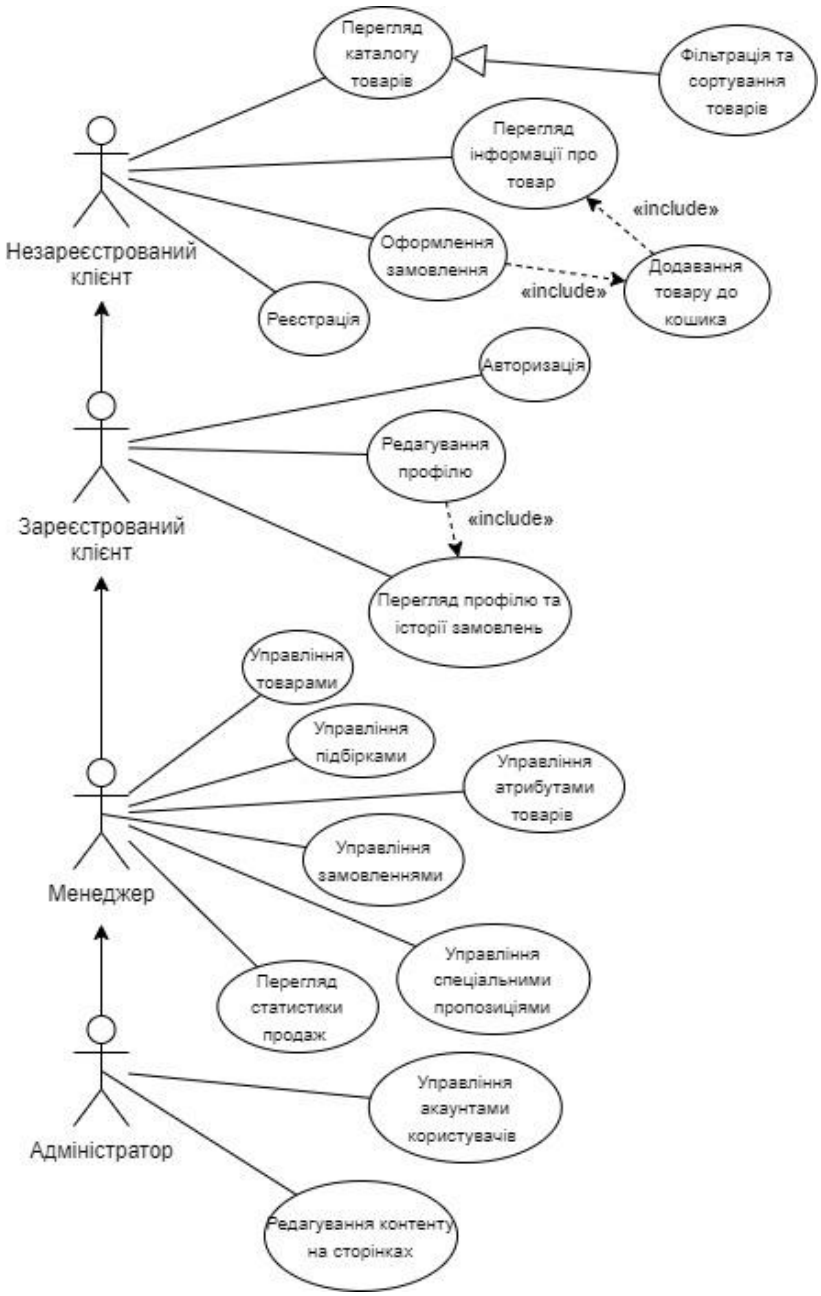


Рисунок 2.1 – Діаграма варіантів використання системи

На діаграмі зображено 4 ролі користувачів:

– незареєстрований клієнт (гість): може зареєструватись, переглядати каталог товарів та оформити замовлення;

– зареєстрований клієнт: може виконувати всі функції гостя, а також авторизуватись, редагувати власний профіль, переглядати історію замовлень;

– менеджер: може виконувати всі функції зареєстрованого клієнта, а також управляти товарами, їх атрибутами, замовленнями, контактами, спеціальними пропозиціями та переглядати статистику продаж;

– адміністратор: може виконувати всі функції менеджера, а також управляти акаунтами користувачів та редагувати контент на деяких сторінках.

2.2 Опис варіантів використання

Сценарій використання – це опис взаємодії між користувачем і системою для реалізації функції або задачі. Він використовується для аналізу вимог до системи та для визначення деталей функціональності. Сценарії використання можуть бути представлені у короткій, поверхневій та повній формах.

Сценарій №1 (коротка форма): Управління контентом на сторінках.

Авторизований адміністратор відкриває панель адміністрування, там обирає розділ управління сторінками. Якщо йому потрібно створити якусь сторінку, наприклад «Про нас», адміністратор натискає кнопку створення, після чого з’являється форма створення сторінки. Адміністратор заповнює її та зберігає дані. Після успішної перевірки система створює нову сторінку та відображає повідомлення про це. Якщо потрібно відредагувати контент на певній сторінці, адміністратор обирає її у переліку, натискає кнопку редагування, після чого змінює дані у формі та зберігає. Після успішної перевірки система оновлює дані та відображає повідомлення про це. Якщо необхідно видалити сторінку, користувач обирає її у переліку та натискає кнопку видалення і підтверджує дію. Система видаляє сторінку та відображає повідомлення про успішне видалення.

Сценарій №2 (коротка форма): Перегляд статистики.

Авторизований менеджер або адміністратор відкриває панель адміністрування. Користувач вибирає період, за який він хоче переглянути статистику продаж (наприклад, останній тиждень, місяць, рік тощо). Система відображає різні показники статистики продаж за обраний період, такі як кількість нових клієнтів, загальний оборот, кількість замовлень, прибуток, витрати, середній чек, найпопулярніші товари. Інформація представлена у вигляді графіків, таблиць та інших візуальних елементів.

Сценарій №3 (поверхнева форма): Авторизація користувача.

Головний сценарій.

На головній сторінці магазину користувач натискає кнопку «Увійти». Система перенаправляє користувача на сторінку авторизації. Користувач вводить електронну пошту та пароль у відповідні поля та натискає кнопку «Увійти». Система перевіряє коректність даних і перенаправляє користувача на сторінку особистого кабінету та надає доступ до його облікових даних.

Альтернативні сценарії:

1) користувач не має облікового запису. Система пропонує створити його;

2) користувач ввів неправильний логін чи пароль. Система сповіщає про некоректність введених даних;

3) користувач не пам’ятає пароль та натискає кнопку «Забули пароль?». Після чого йому на електронну пошту приходить лист з посиланням на форму відновлення паролю;

4) обліковий запис користувача заблокований адміністратором. Система надсилає повідомлення про те, що аккаунт заблоковано;

5) технічний збій роботи системи. Користувач отримує повідомлення про недоступність сервісу.

Сценарій №4 (поверхнева форма): Реєстрація користувача.

Головний сценарій.

Користувач натискає на кнопку «Реєстрація», після чого опиняється на сторінці реєстрації з формою. Після чого заповнює поля: Ім’я, email, пароль та підтвердження паролю. Потім натискає на кнопку «Зареєструватися». Система перенаправляє користувача на головну сторінку та надсилає на його електронну пошту верифікаційний лист.

Альтернативні сценарії:

1) користувач ввів невалідні дані. Система повідомляє про помилку;

2) введений email вже використовується. Система повідомляє, що електронна адреса вже зайнята;

3) технічний збій роботи системи. Користувач отримує повідомлення про недоступність сервісу.

Сценарій №5 (поверхнева форма): Редагування профілю.

Головний сценарій.

Авторизований користувач переходить до розділу редагування профілю. Користувач вводить нові дані у необхідні поля, після чого підтверджує зміни. Після успішної перевірки система оновлює дані профілю користувача. Користувач отримує повідомлення про успішну зміну даних.

Альтернативні сценарії:

1) користувач ввів невалідні дані. Система повідомляє про помилку;

2) новий введений email вже використовується. Система повідомляє, що електронна адреса вже зайнята;

3) технічний збій роботи системи. Користувач отримує повідомлення про недоступність сервісу.

Таблиця 2.1 – **Сценарій №6 (повна форма): Оформлення замовлення.**

Usecase section	Comment
Use Case Name	Оформити замовлення
Scope	Інтернет-магазин сонцезахисних окулярів
Level	Оформлення замовлення в інтернет-магазині
Primary Actor	Клієнт
Stakeholders and interests	– Клієнт – замовити товар; – адміністратор та менеджер – успішне оформлення та обробка замовлення;
Preconditions	Користувач має бути зареєстрований, його профіль має бути активований.
Success guarantee	– Користувач використовує стабільне підключення до мережі Інтернет; – профіль користувача активований; – користувач вводить коректні дані;

Продовження таблиці 2.1

Main Success Scenario	1) Користувач знайшов один або декілька необхідних товарів; 2) користувач натискає на зображення кошика на картці товару; 3) користувач переходить до кошика; 4) за необхідності користувач коригує кількість товарів. 5) система підраховує загальну вартість товарів у кошику; 6) користувач натискає кнопку «Оформити замовлення». 7) система перенаправляє користувача на форму оформлення замовлення; 8) користувач заповнює усі поля необхідними даними та натискає кнопку «Оформити»; 9) система відправляє користувачу повідомлення про успішне оформлення замовлення; 10) дані про нове замовлення відображаються на панелі адміністратора та менеджера.
Extensions	а) Користувач не авторизований або не має облікового запису: 1) користувач переходить до кошика та натискає кнопку «Оформити замовлення»; 2) користувач оформлює замовлення та заповнює додаткові поля з особистими даними та натискає кнопку «Оформити»; 3) Система повідомляє про успішне оформлення замовлення. б) користувач бажає зареєструватись, заповнює додаткові поля з особистою інформацією та вводить пароль. Система створює замовлення та акаунт; в) помилка вводу даних у форму. Система сповіщає про некоректні дані; г) технічний збій роботи системи. Користувач отримує повідомлення про недоступність сервісу.

Кінець таблиці 2.1

Special Requirements	– Система повинна надавати швидкий та ефективний процес обробки замовлення; – інтерфейс для оформлення замовлення повинен бути оптимізований для мобільних пристроїв; – забезпечити надсилання сповіщень про статус замовлення на електронну пошту.
Technology and Data Variations List	– Інтегрувати систему оформлення замовлення з платформою надсилання повідомлень (електронна пошта); – інтегрувати систему оформлення замовлення з поштовим сервісом для отримання списку областей, населених пунктів та відділень; – використовувати техніки адаптивного дизайну для забезпечення оптимального відображення інтерфейсу на різних пристроях та розмірах екранів.
Frequency of Occurrence	Регулярно, в залежності від активності користувачів.
Miscellaneous	-

Таблиця 2.2 – **Сценарій №7 (повна форма): Управління атрибутами товарів.**

Usecase section	Comment
Use Case Name	Створити, відредагувати або видалити атрибут товару
Scope	Інтернет-магазин сонцезахисних окулярів
Level	Управління атрибутами товарів у інтернет-магазині
Primary Actor	Адміністратор або менеджер
Stakeholders and interests	– адміністратор або менеджер – успішне створення, редагування або видалення атрибуту товару.
Preconditions	Адміністратор або менеджер має бути авторизований.

Продовження таблиці 2.2

Success guarantee	– Користувач використовує стабільне підключення до мережі Інтернет; – профіль користувача активований; – користувач вводить коректні дані;
Main Success Scenario	а) Створення атрибуту: 1) менеджер увійшов у адміністративну панель застосунку; 2) менеджер обрав розділ управління атрибутами; 3) менеджер обирає опцію створення атрибуту; 4) менеджер вводить необхідні дані та зберігає новий атрибут; 5) система зберігає дані та відображає повідомлення про успішне створення атрибуту; б) редагування атрибуту: 1) менеджер увійшов у адміністративну панель застосунку; 2) менеджер обрав розділ управління атрибутами; 3) менеджер обирає опцію редагування атрибуту; 4) менеджер вводить нові необхідні дані та зберігає зміни; 5) система зберігає дані та відображає повідомлення про успішне редагування атрибуту; в) видалення атрибуту: 1) менеджер увійшов у адміністративну панель застосунку; 2) менеджер обрав розділ управління атрибутами; 3) менеджер натискає кнопку видалення атрибуту; 4) система видаляє атрибут товару та відображає повідомлення про успішне видалення.

Кінець таблиці 2.2

Extensions	– Помилка вводу даних у форму. Система сповіщає про некоректні дані. – Технічний збій роботи системи. Користувач отримує повідомлення про недоступність сервісу.
Special Requirements	– Система повинна надавати швидкий та ефективний процес обробки даних.
Frequency of Occurrence	Рідко, в залежності від потреб магазину
Miscellaneous	-

Управління категоріями, брендами, товарами та акаунтами користувачів мають схожий сценарій з управлінням атрибутами товарів, тому окремо сценарії для них не розписано.

Таблиця 2.3 **– Сценарій №8 (повна форма): Управління замовленнями.**

Usecase section	Comment
Use Case Name	Створити або відредагувати замовлення
Scope	Інтернет-магазин сонцезахисних окулярів
Level	Створення або редагування замовлення
Primary Actor	Адміністратор або менеджер
Stakeholders and interests	– Адміністратор або менеджер – успішне створення або редагування замовлення.
Preconditions	Користувач має бути зареєстрований, його профіль має бути активований.
Success guarantee	– Користувач використовує стабільне підключення до мережі Інтернет; – профіль користувача активований; – користувач вводить коректні дані;

Продовження таблиці 2.3

Main Success Scenario	а) Створення замовлення: 1) менеджер відкриває панель адміністрування та обирає розділ замовлень; 2) менеджер вводить дані клієнта, обирає товари та їх кількість; 3) менеджер підтверджує замовлення; 4) система коригує кількість товарів на складі; 5) система сповіщає про успішне створення замовлення та надсилає сповіщення на електронну адресу клієнта; б) редагування замовлення: 1) менеджер відкриває панель адміністрування та обирає розділ замовлень; 2) менеджер переходить до необхідного замовлення та натискає кнопку редагування; 3) менеджер вводить нові дані, за необхідності додає товари та редагує їх кількість; 4) менеджер зберігає зміни; 5) система коригує кількість товарів на складі; 6) система сповіщає про успішне редагування замовлення.
Extensions	– Необхідних товарів немає на складі. У менеджера немає можливості додати їх; – Помилка вводу даних у форму. Система сповіщає про некоректні дані. – Технічний збій роботи системи. Користувач отримує повідомлення про недоступність сервісу.
Special Requirements	– Система повинна надавати швидкий та ефективний процес обробки замовлення; – забезпечити надсилання сповіщень про створення замовлення на електронну пошту клієнта.

Кінець таблиці 2.3

Technology and Data Variations List	– Інтегрувати систему оформлення замовлення з платформою надсилання повідомлень (електронна пошта); – інтегрувати систему оформлення замовлення з поштовим сервісом для отримання списку областей, населених пунктів та відділень; – реалізувати сортування та фільтрацію замовлень.
Frequency of Occurrence	Регулярно, в залежності від потреб магазину та активності клієнтів.
Miscellaneous	-

Сценарій №9 (поверхнева форма): Зміна статусу замовлень.

Головний сценарій.

Авторизований адміністратор або менеджер входить у систему, переходить до розділу управління замовленнями. Менеджер шукає потрібне замовлення, за необхідності користується фільтрацією та сортуванням замовлень. Після знаходження потрібного замовлення, менеджер переходить на сторінку цього замовлення. В залежності від потреб, змінює статус самого замовлення, статус доставки або статус оплати, використовуючи відповідні опції випадаючого списку.. Система оновлює статус замовлення та сповіщає про збережені зміни. Якщо замовлення вже доставлене або відмінене, система надсилає повідомлення про це на електронну пошту клієнта.

Альтернативні сценарії:

1) статус замовлення «Виконано», при спробі змінити статус доставки чи оплати, система блокує цю дію та сповіщає, що замовлення вже виконане;

2) статус замовлення «Відмінено», при спробі змінити статус доставки чи оплати, система блокує цю дію та сповіщає, що замовлення відмінене;

3) обрана опція оплати на карту, але замовлення ще не оплачене, у такому випадку неможливо змінити статус доставки на «Доставляється» та «Доставлено»;

4) обрана опція оплати при отриманні, але замовлення ще не оплачене частково. У такому випадку неможливо змінити статус доставки на «Доставляється» та «Доставлено»;

5) технічний збій роботи системи. Користувач отримує повідомлення про недоступність сервісу.

Таблиця 2.4 **– Сценарій №10 (повна форма): Управління спеціальними пропозиціями.**

Usecase section	Comment
Use Case Name	Створити, відредагувати або видалити спеціальну пропозицію (акції)
Scope	Інтернет-магазин сонцезахисних окулярів
Level	Створення, редагування або видалення спеціальної пропозиції (акції)
Primary Actor	Адміністратор або менеджер
Stakeholders and interests	– Адміністратор або менеджер – успішне створення, редагування або видалення спеціальної пропозиції.
Preconditions	Користувач має бути зареєстрований, його профіль має бути активований. Користувач повинен мати доступ до адмін панелі та права на редагування акції.
Success guarantee	– Користувач використовує стабільне підключення до мережі Інтернет; – профіль користувача активований; – користувач вводить коректні дані;

Продовження таблиці 2.4

Main Success Scenario	а) Створення спеціальної пропозиції: 1) менеджер відкриває панель адміністрування та обирає розділ спеціальних пропозицій; 2) менеджер вводить дані спеціальної пропозиції, обирає банер, обирає товари, що беруть участь у акції; 3) менеджер підтверджує створення пропозиції; 4) система змінює вартість обраних товарів за введеним процентом знижки; 5) система сповіщає про успішне створення пропозиції; 6) система надсилає клієнтам повідомлення про акцію на електронну пошту; б) редагування спеціальної пропозиції: 1) менеджер відкриває панель адміністрування та обирає розділ спеціальних пропозицій; 2) менеджер переходить до необхідної пропозиції та натискає кнопку редагування; 3) менеджер вводить нові дані, за необхідності змінює товари; 4) менеджер зберігає зміни; 5) система змінює вартість товарів за введеним процентом знижки; 6) система сповіщає про успішне редагування спеціальної пропозиції; в) видалення спеціальної пропозиції: 1) менеджер відкриває панель адміністрування та обирає розділ спеціальних пропозицій; 2) менеджер переходить до необхідної пропозиції та натискає кнопку видалення та підтверджує дію; 3) система повертає товарам їх основну вартість та зберігає зміни; 4) система повідомляє про успішне видалення пропозиції.

Кінець таблиці 2.4

Extensions	– Помилка вводу даних у форму. Система сповіщає про некоректні дані. – Технічний збій роботи системи. Користувач отримує повідомлення про недоступність сервісу.
Special Requirements	– Система повинна надавати швидкий та ефективний процес обробки даних; – забезпечити надсилання сповіщень про нову спеціальну пропозицію.
Technology and Data Variations List	– Інтегрувати систему з платформою надсилання повідомлень (електронна пошта);
Frequency of Occurrence	Регулярно, в залежності від потреб магазину.
Miscellaneous	-

У таблиці 2.4 було представлено сценарій управління спеціальними пропозиціями у повній формі. Спеціальні пропозиції – це різноманітні знижки чи акції, які магазин пропонує своїм клієнтам для стимулювання продажів і привертання уваги до певних товарів чи послуг.

2.3 Побудова діаграм станів та переходів

Діаграма станів є однією з п’яти діаграм UML, які використовуються для моделювання динамічної природи системи. Вони визначають різні стани об'єкта протягом його життя, і ці стани змінюються подіями. Діаграми станів корисні для моделювання реактивних систем. Реактивні системи можна визначити як систему, яка реагує на зовнішні або внутрішні події.

Діаграма станів описує потік керування від одного стану до іншого. Стани визначаються як умови, в яких існує об’єкт і він змінюється, коли запускається якась подія. Найважливіша мета діаграми станів – моделювати тривалість життя об’єкта від створення до припинення [6].

На рисунку 2.2 представлено діаграма станів та переходів для системи в цілому (лише для авторизованого клієнта). Варто зазначити, що перехід до головної сторінки, каталогу, кошика та профілю користувача може відбуватись з будь-якої сторінки сайту, однак на діаграмі цього не показано, щоб діаграма виглядала простіше та менш заплутаною.

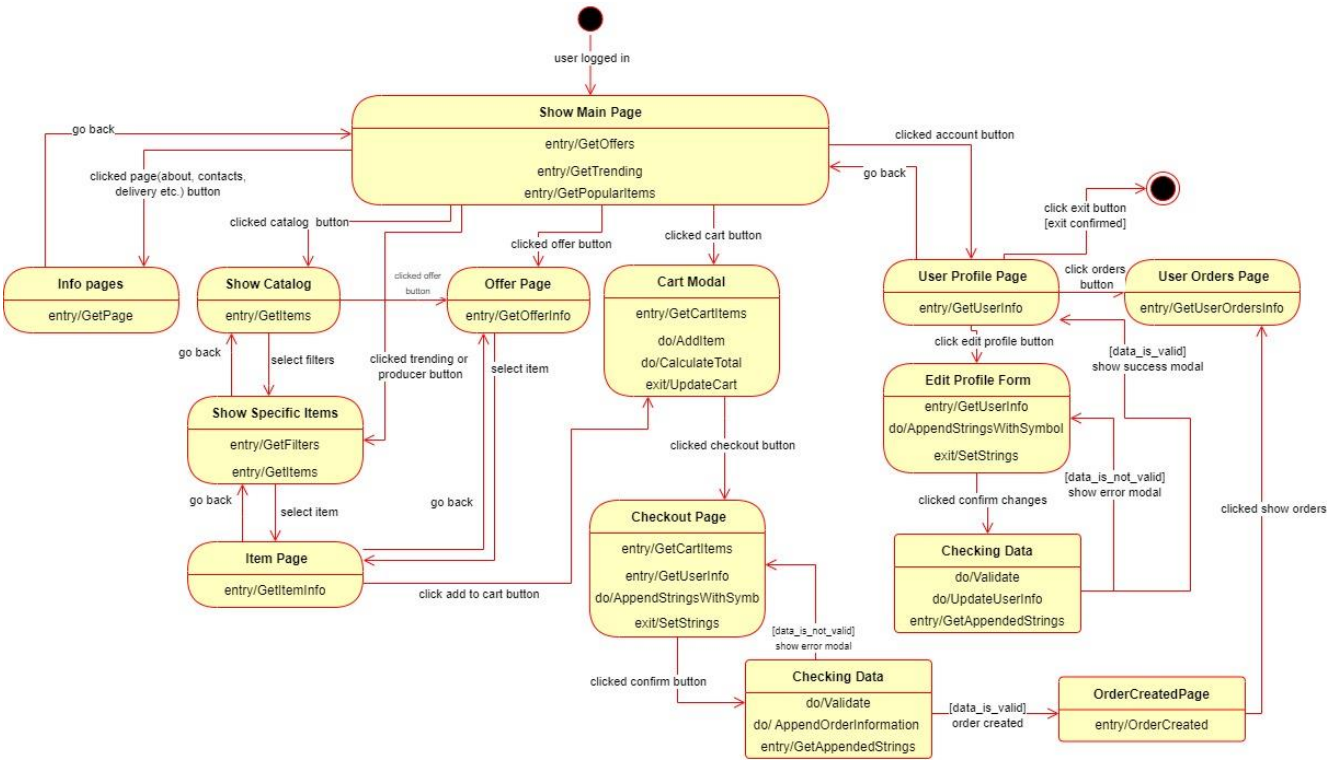


Рисунок 2.2 – Діаграма станів та переходів для систпеми в цілому для авторизованого клієнта

Основною задачею адміністратора та менеджера є управління об’єктами магазину (товари, атрибути товарів, спеціальні пропозиції, підбірки, знижки, сторінки, замовлення) та користувачами. Тому було створено узагальнену діаграму станів системи для CRUD-операцій (рис. 2.3).

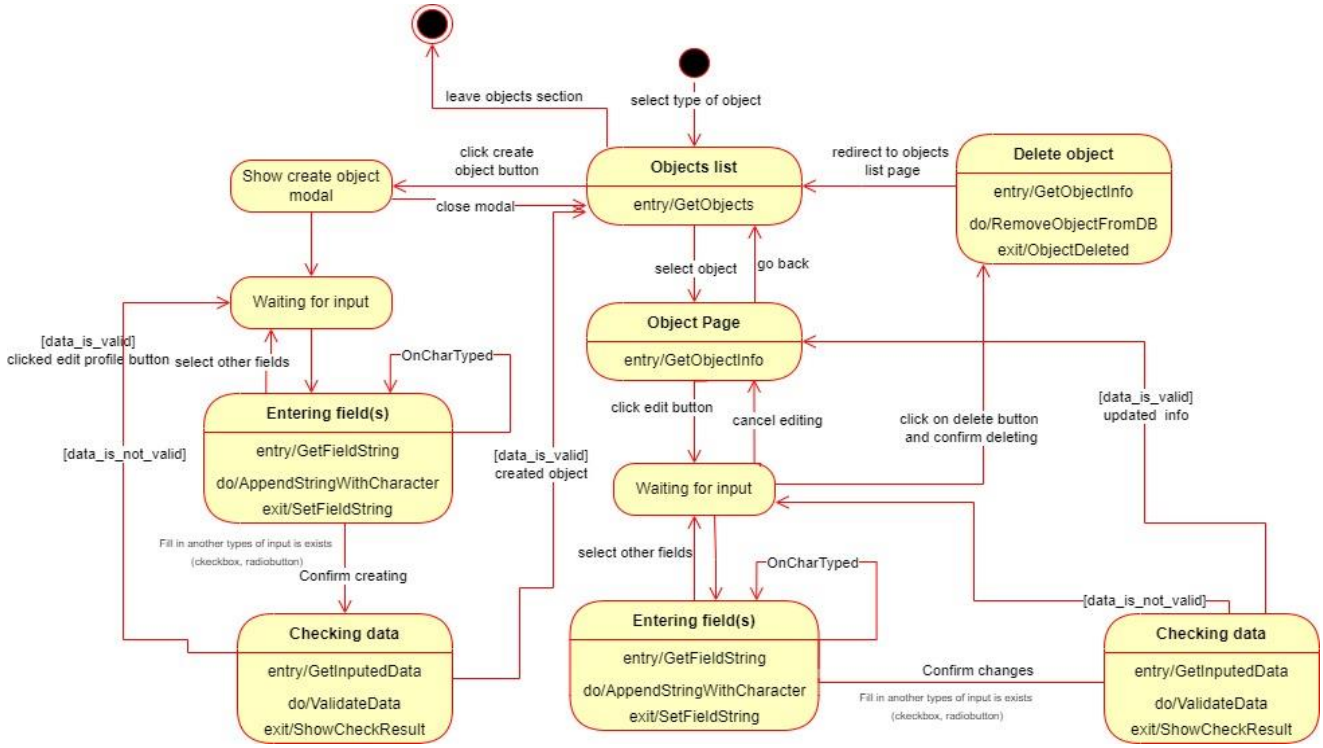


Рисунок 2.3 – Діаграма станів та переходів для CRUD-операцій

На даній діаграмі вказано, що вхідною точкою є вибір розділу (тип об’єкта), а вихідною – вихід з цього розділу після внесення необхідних змін за потреби.

Висновки до розділу 2

У другому розділі роботи було розписано етап моделювання програмного забезпечення.

Проведено аналіз потреб користувачів та створено діаграму використання, вказано її призначення у проєктуванні програмного забезпечення. Розписано сценарії використання системи для клієнта, менеджера та адміністратора у трьох формах: короткій, поверхневій та повній.

Наступним етапом було створення діаграм станів та переходів для системи в цілому для відображення сценаріїв використання авторизованим клієнтом та діаграму для CRUD-операцій.

Завдяки описаним крокам отримано розуміння того, як система має працювати, які функції вона має виконувати та як користувачі будуть з нею взаємодіяти та закріплено навички моделювання ПЗ.

3 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ТА ОГЛЯД СТЕКУ ТЕХНОЛОГІЙ

У цьому розділі буде розглянуто процес проєктування вебзастосунку та роглянуто стек технологій, які було використано для його розробки. Проєктування вебзастосунку є важливим етапом, який визначає архітектуру системи. Огляд стеку технологій включає опис використаних технологій для розробки програмного забезпечення: мова програмування, фреймворки, бібліотеки та бази даних.

3.1 Огляд технологій

Для розробки вебзастосунку «Інтернет-магазин сонцезахисних окулярів» було задіяно наступні технології:

- мови програмування – PHP, JavaScript;
- фреймворки – Laravel, Vue.js;
- система управління базами данних – MariaDB;
- вебсервер – nginx;
- бібліотеки – jQuery, TailwindCSS, pinia.

PHP

PHP – це популярна мова сценаріїв загального призначення, яка особливо підходить для веб-розробки. Швидкий, гнучкий і прагматичний PHP забезпечує все, від вашого блогу до найпопулярніших вебсайтів у світі [7]. Переваги використання PHP для вебзастосунків:

- висока продуктивність та швидкодія;
- підтримка сучасних фреймворків, таких як Laravel, Symfony, та Zend Framework;
- широка екосистема – PHP має велику кількість готових бібліотек та пакетів, доступних через Composer;
- гнучкість та розширюваність – PHP легко інтегрується з іншими мовами та технологіями, такими як JavaScript, що дозволяє створювати динамічні та інтерактивні вебзастосунки. Крім того, PHP підтримує розширення через власні модулі та бібліотеки;

– PHP стабільний і надійний для довготривалих проектів.

JavaScript та **jQuery**

JavaScript (JS) – це полегшена інтерпретована (або своєчасно скомпільована) мова програмування з першокласними функціями. Хоча вона більш відома як мова сценаріїв для веб-сторінок, багато середовищ без браузерів також використовують її, наприклад Node.js, Apache CouchDB і Adobe Acrobat. JavaScript – це багатопарадигмальна, однопотокова, динамічна мова на основі прототипу, яка підтримує об’єктно-орієнтований, імперативний і декларативний стилі (наприклад, функціональне програмування) [8].

Було підключено бібліотеку jQuery для спрощення роботи з JS за допомогою зручного синтаксису та використання селекторів, використання готових методів jQuery і функцій для різних завдань, таких як обробка подій, анімація. Загалом, ця бібліотека дозволяє писати менше коду для досягнення тих же результатів, що й з використанням «чистого» JavaScript.

Laravel

Laravel – це фреймворк вебзастосунків із виразним елегантним синтаксисом. Веб-фреймворк забезпечує структуру та відправну точку для створення вашої програми, дозволяючи зосередитися на створенні чогось дивовижного, поки він працює над деталями [9]. Переваги Laravel для створення вебзастосунків:

– чистий та зрозумілий синтаксис;

– великий набір вбудованих функцій (роутинг, автентифікація, сесії, кешування та інші);

– Eloquent, вбудована ORM (Object-Relational Mapping) система Laravel, надає простий і інтуїтивно зрозумілий спосіб взаємодії з базами даних;

– система міграцій, яка дозволяє легко керувати змінами у структурі бази даних;

– обширна документація;

– шаблонізація Blade, це простий, але потужний механізм створення шаблонів, який входить до складу Laravel [10].

Vue.js

Vue.js – це фреймворк JavaScript для створення інтерфейсів користувача. Він створений на основі стандартних HTML, CSS і JavaScript і забезпечує декларативну модель програмування на основі компонентів, яка допомагає вам ефективно розробляти інтерфейси користувача будь-якої складності [11]. Його обуло використано для створення деяких модальних вікон, кошика, сторінки оформлення замовлення та адмін-панелі. Vue.js має кілька переваг, які роблять його популярним фреймворком для розробки вебзастосунків:

– реактивність: Vue.js використовує реактивний підхід, тобто інтерфейс автоматично оновлюється при зміні даних;

– компонентна архітектура;

– проста інтеграція, використання Vue у вже існуючому коді відбувається з мінімальними зусиллями та складнощами;

– компактний розмір та висока продуктивність;

– екосистема, у Vue.js існує широка екосистема плагінів та інструментів, що допомагає розширити його функціональність та прискорити розробку.

У якості state-менеджера для Vue було використано бібліотеку pinia. **Pinia** – це store-бібліотека для Vue, вона дозволяє ділитися станом між компонентами/сторінками [12].

MariaDB

MariaDB – це система управління базами даних, яка виникла як відгалуження MySQL. MariaDB існує як альтернатива MySQL, оскільки пропонує нові функції, покращення продуктивності, краще тестування та виправлення помилок, яких немає в MySQL [13]. Основні переваги MariaDB у порівнянні з MySQL:

– більша швидкодія та продуктивність;

– нові функції: MariaDB надає ряд нових функцій, які розширюють можливості бази даних, включаючи підтримку нових типів даних, операцій та функціональність;

– безпека: MariaDB має додаткові заходи безпеки, такі як захист від SQL-ін'єкцій та інші методи захисту даних.

TailwindCSS

TailwindCSS – CSS-фреймворк, який надає набір готових класів для створення користувацьких інтерфейсів. Перший утилітарний фреймворк CSS, наповнений такими класами, як flex, pt-4, text-center і rotate-90, які можна скомпонувати для створення будь-якого дизайну безпосередньо у вашій розмітці [14]. Основна відмінність Tailwind CSS полягає в його концепції «utility-first». Замість того, щоб пропонувати готові компоненти, Tailwind CSS надає набір атомарних класів, які використовуються для стилізації окремих елементів. Це дозволяє швидко створювати та налаштовувати стилі, не виходячи за межі HTML файлу. Tailwind CSS має вбудовану підтримку адаптивного дизайну, що дозволяє легко змінювати вигляд елементів на різних розмірах екрану, використовуючи префікси, наприклад, sm:, md:, lg:.

3.2 Діаграма сутність-зв’язок

Моделювання бази даних є важливою складовою проєктування системи для інтернет-магазину сонцезахисних окулярів, оскільки воно дозволяє ефективно зберігати та управляти даними. Визначення сутностей, їх атрибутів та взаємозв'язків допомагає створити структуру, яка відображає сутність бізнес-процесів та потреб користувачів. Це дозволяє оптимізувати доступ до даних та забезпечити швидкий пошук та обробку інформації.

Діаграма сутність-зв’язок (entity relationship diagram, ERD) – це візуальна форма реляційних баз даних. ERD можна використовувати для розробки нової бази даних або візуалізації та розуміння існуючої. Їх також часто називають моделями зв’язків сутностей або графіками зв’язків сутностей.

ERD – це модель даних, яка графічно представляє сутності організації та зв’язки між цими сутностями. Сутності – це цінності, які представляють речі, з якими працює організація, наприклад клієнти, продукти або постачальники. Зв'язки – це дескриптори, які визначають, як взаємодіють дві сутності, наприклад

покупки чи продажі. Діаграма сутність-зв’язок показує ці сутності та їхні зв’язки в графічному вигляді [15].

На рисунку 3.1 представлено діаграму сутність-зв'язок для інтернет-магазину сонцезахисних окулярів.

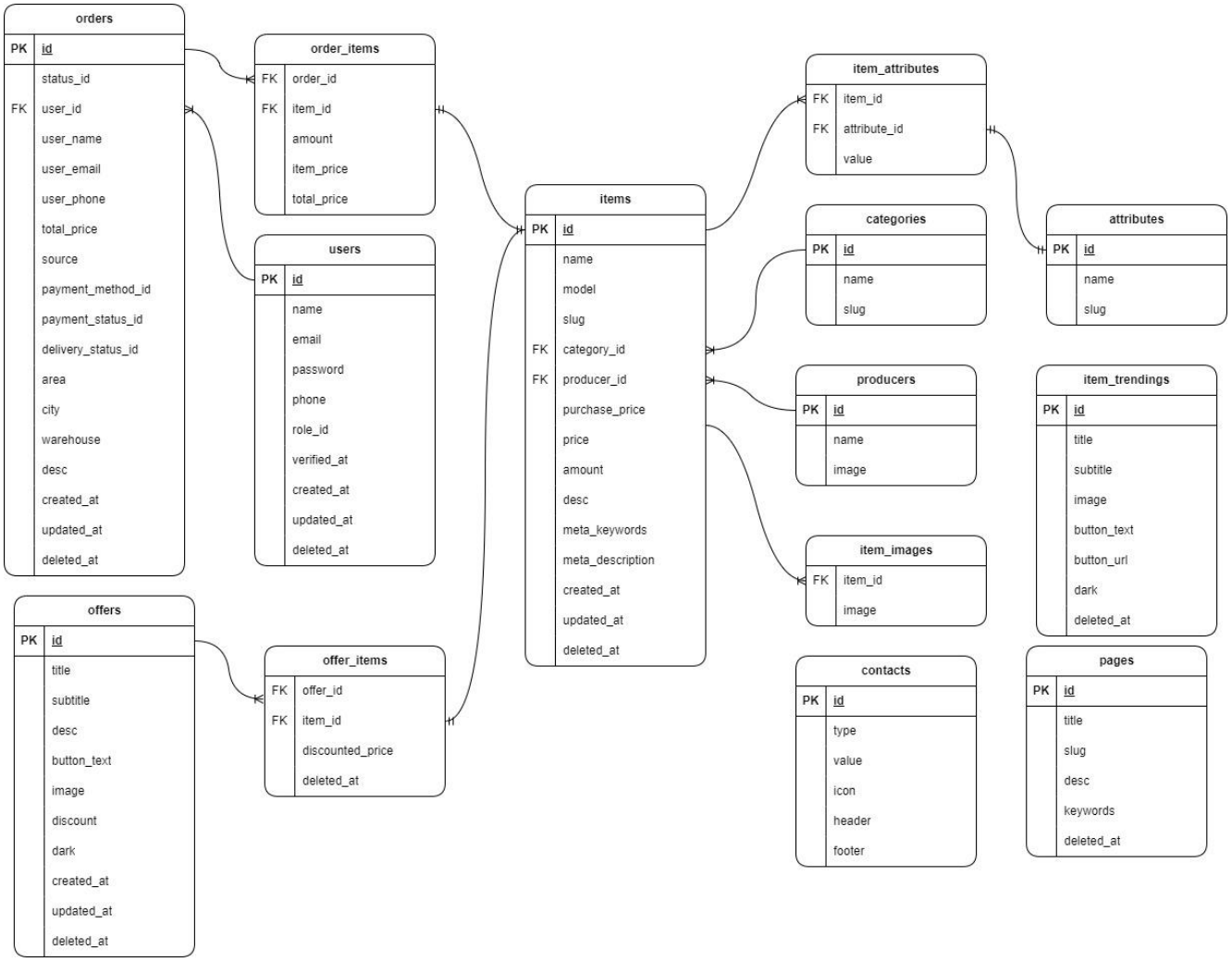


Рисунок 3.1 – Діаграма сутність-зв’язок інтернет-магазину сонцезахисних окулярів

На даній діаграмі представлено такі **основні таблиці**:

– items – таблиця товарів, включає у себе основні відомості про товар, кількість одиниць товару на складі, має зв’язок з таблицями категорій, виробників, зображень, коментарів та з проміжними таблицями для замовлень, спеціальних пропозицій, значень атрибутів;

– item_images – таблиця для зберігання зображень товарів;

– categories – таблиця категорій товарів;

– producers – таблиця виробників товарів;

– attributes – таблиця атрибутів;

– users – таблиця користувачів, містить загальну інформацію про користувача, хешований пароль, має зв’язки з таблицями замовлення та відгуків;

– orders – таблиця замовлень, містить дані про замовлення, а саме: дані замовника, адресу доставки, метод оплати, статуси замовлення, доставки та оплати, опис та джерело замовлення;

– offers – таблиця спеціальних пропозицій (акцій), містить дані про акцію, заголовок, опис, банер, процент знижки;

– item_trendings – таблиця підбірок. Містить такі поля: заголовок, підзаголовок, картинку, текст кнопки, посилання, яке складається з обраних атрибутів, яке передається для каталогу для фільтрації товарів; а також поле dark для вибору кольору тексту (світлий чи темний);

– pages – таблиця, де зберігаються дані деяких сторінок, які може редагувати адміністратор (наприклад, сторінки «Про нас», «Контакти», «Доставка» і тому подібні);

– contacts – таблиця, де зберігаються контактні дані (посилання на соціальні мережі, номери телефонів, геопозиція, пошта). Містить поля типу контакту, значення, іконки та чи буде відображатись у хедері/футері.

Проміжні таблиці:

– order_items – таблиця для зберігання товарів, що відносяться до певного замовлення. Вона пов’язана з таблицями товарів та замовлень. У ній зберігається кількість одиниць товару у замовленні, його вартість на момент створення замовлення та загальна вартість за усі одиниці даного товару у замовленні;

– offer_items – таблиця для зберігання товарів, що відносяться до певної спеціальної пропозиції. Вона пов’язана з таблицями товарів та спеціальних пропозицій. Містить також ціну товару зі знижкою;

– item_attributes – таблиця для зберігання значень атрибутів товарів, пов’язана з таблицями товарів та атрибутів.

3.3 Архітектура системи та діаграма класів

Для даного проєкту було використано архітектуру MVC (Model View Controller). Архітектура MVC є широко використовуваним шаблоном для розробки вебзастосунків.

Model View Controller широко відоме як архітектура MVC, складається з трьох частин: модель, представлення та контролер. MVC в основному використовується для розробки вебзастосунків, оскільки допомагає розробнику мати чітке розуміння всіх модулів. Архітектура MVC розділяє бізнес-рівень (логіка моделі), рівень відображення (логіка перегляду) і рівень керування (логіка контролера). MVC допомагає організувати дані та розділити код відповідно до вимог, щоб зменшити складність і працювати над одним файлом як бізнес-логікою. Повторне використання коду та гнучкість можливі за рахунок впровадження архітектури MVC [16].

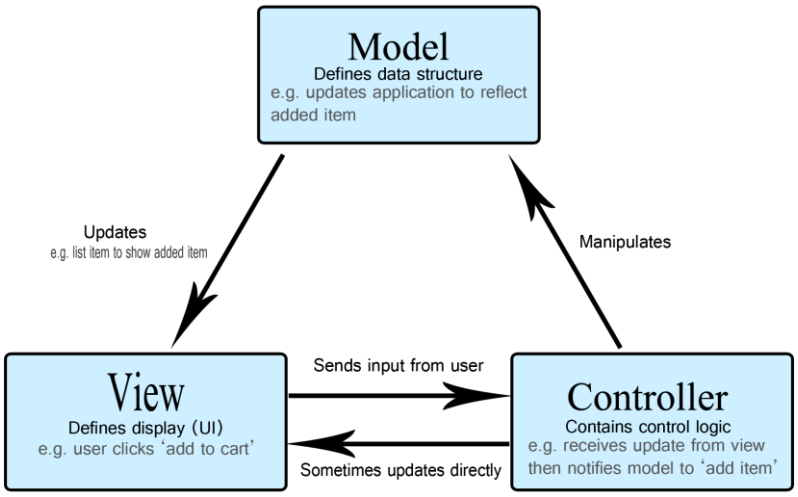


Рисунок 3.1 – Діаграма архітектури MVC [17]

Наступним етапом є створення діаграми класів. Діаграми класів – це тип діаграми UML (Unified Modeling Language), яка використовується в розробці програмного забезпечення для візуального представлення структури та зв’язків класів у системі, тобто для побудови та візуалізації об’єктно-орієнтованих систем [18].

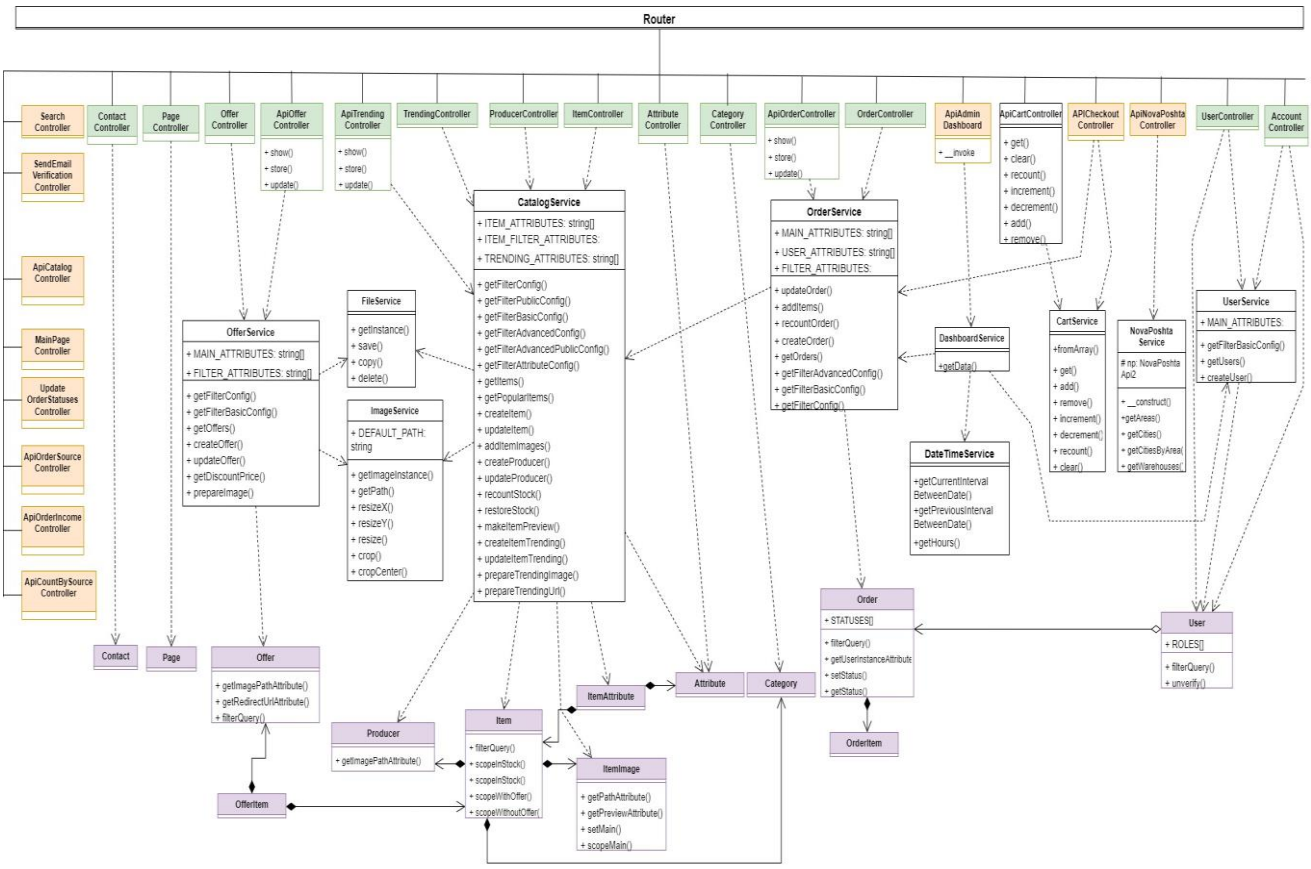


Рисунок 3.3 – Діаграма класів вебзастосунку

На діаграмі (рис.3.3) відображені не усі класи, поля та методи через їх велику кількість а також задля спрощення її розуміння. Відображено лише основні, ключові класи та їх методи Оскільки використовується архітектура MVC, усі класи можна розділити на такі групи:

– моделі (models, позначені фіолетовим) – класи, які представляють сутності та відповідають за взаємодію з базою даних;

– контролери (controllers) – класи, що відповідають за обробку запитів від користувачів, взаємодію з моделями та керуванням відображення у представленнях;

– сервіси (services) – класи, у яких зібрано бізнес-логіку.

Контроллери також поділяються на такі групи:

а) ресурсні контроллери (позначені зеленим) – тип контролера, який автоматично надає набір стандартних маршрутів для виконання CRUD операцій, включає наступні методи:

1) index() – відображає список ресурсів;
2) show($id) – відображає один ресурс за ідентифікатором;
3) create() — показує форму для створення нового ресурсу;
4) store(Request $request) — зберігає новий ресурс у базі даних;
5) edit($id) — показує форму для редагування існуючого ресурсу;
6) update(Request $request, $id) — оновлює ресурс у базі даних;
7) destroy($id) — видаляє ресурс з бази даних;

б) Single Action контроллери (позначені помаранчевим) – контролери, які містять лише один метод дії (__invoke), використовуються у випадках, коли контролер повинен виконувати тільки одну задачу;

в) API-контроллери (з префіксом «API» у назві) – контролери, спеціально призначені для обробки запитів до API, зазвичай повертають дані у форматі JSON.

г) кастомні контроллери – контроллери, які створюються відповідно до потреб, без обмеження стандартними CRUD операціями або концепцією одного методу.

Кастомні контроллери краще уникати, адже вони порушують принципи SOLID і можуть вважатися поганим тоном.

Оскільки деякі компоненти фронтенду є Vue-компонентами, використовуються також API-контроллери для обробки запитів з фронтенду.

Моделі реалізують можливості ORM (Eloquent) і зазвичай пов'язані з відповідними контролерами, що використовують ці моделі для взаємодії з базою

даних. Це відношення на діаграмі класів можна відобразити у вигляді асоціації, зокрема у вигляді використання. Контролери використовують моделі для доступу до даних і виконання CRUD-операцій. Контроллери також пов’язані з сервісами зв’язком використання. У сервісах прописана бізнес-логіка та деякі операції для створення чи редагування сутностей, щоб не розповсюджувати логіку між моделями та контроллерами. Сервіси зазвичай використовують моделі для отримання або збереження даних. Це відношення також відображається на діаграмі класів як використання.

Між моделями присутнє відношення агрегації. Агрегація може відображати відношення, коли одна модель містить зв'язок з іншою моделлю, але ці моделі можуть існувати і працювати незалежно одна від одної. Це відношення використовується між товаром та його картинками, між замовленням та користувачем. Відношення композиції є сильнішим відношенням, ніж агрегація. Воно вказує на те, що один об'єкт є частиною іншого об'єкта і ця залежність є повною – частина не може існувати без цілого. Даний тип відношення використовується між атрибутом товару та атрибутом і товаром, товаром та категорією, виробником; товаром у замовленні та товаром і замовленням; товаром у офері та товаром і офером.

3.4 Діаграми компонентів та розгортання

Діаграма компонентів є ключовим інструментом у проєктуванні програмного забезпечення, оскільки вона надає візуальне відображення структури системи та допомагає чітко розділити функціональні частини програми. Вона сприяє зрозумінню взаємозв'язків між компонентами.

Мета діаграми компонентів – показати взаємозв’язок між різними компонентами в системі. Для цілей UML 2.0 термін «компонент» відноситься до модуля класів, які представляють незалежні системи або підсистеми з можливістю взаємодії з рештою системи [19].

На рисунку 3.4 представлено діаграму компонентів бекенду для показу взаємодії деяких компонентів системи.

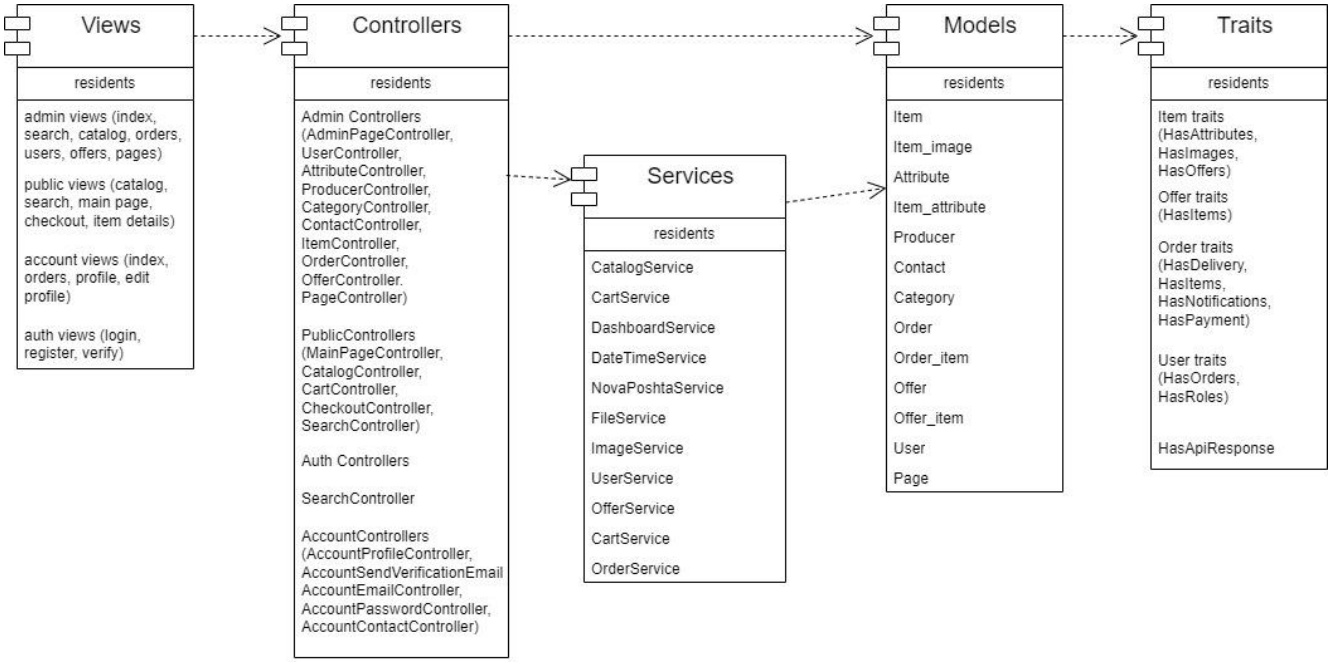


Рисунок 3.4 – Діаграма компонентів бекенду

На даній діаграмі можна побачити, що представлення (views) використовують контролери для отримання необхідних даних та обробки запитів користувачів. Контролери (controllers), у свою чергу, взаємодіють із моделями та сервісами, в яких прописана бізнес-логіка та деякі операції для створення чи редагування сутностей. Сервіси використовують моделі, що реалізують можливості ORM, для роботи з базою даних. Моделі використовують трейти, що містять додаткові методи. Трейти зручні тим, що розширюють функціональність класів без наслідування а також надають можливість ділитися спільною логікою між різними класами.

Для створення модальних вікон для створення та редагування замовлень, підбірок та спеціальних пропозицій, кошика та сторінки оформлення замовлення, адмін-панелі було використано фрейморк Vuejs, тому також застосовувалась архітектура HTTP API. Для відображення цієї частини було створено окрему діаграму компонентів (рис. 3.5).

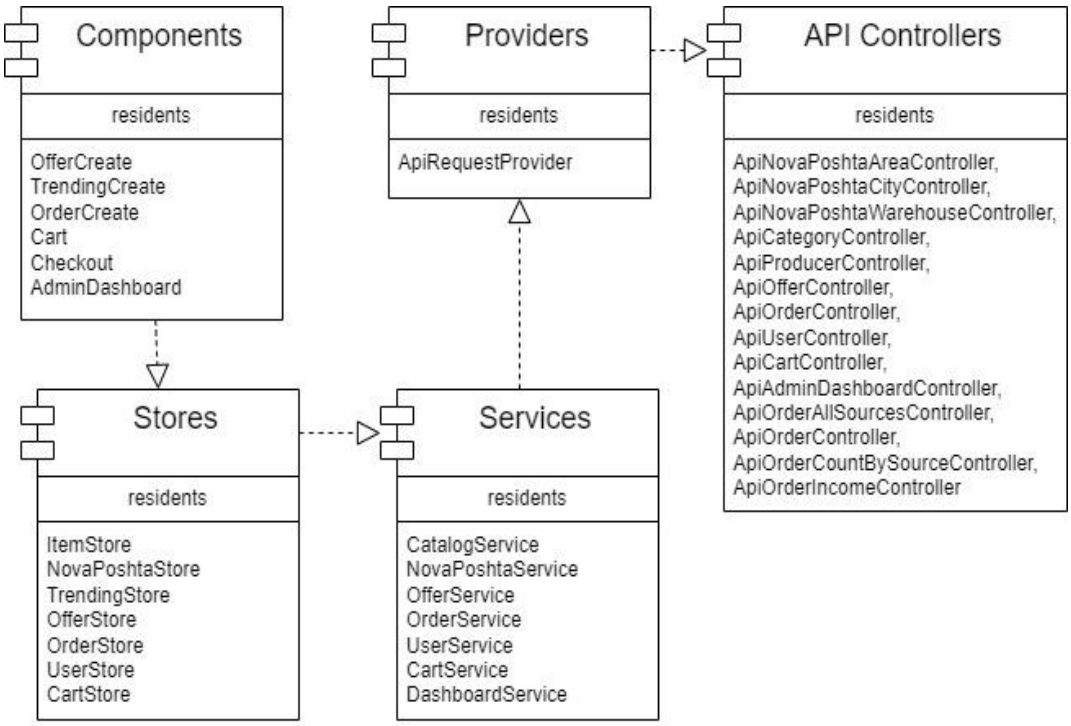


Рисунок 3.5 – Діаграма компонентів

На даній діаграмі компонентів представлено:

– components – vue-компоненти, що відповідають за візуальне представлення даних та інтерфейс взаємодії, у них містяться методи з компонентів stores;

– stores – стори, які зберігають дані та надають методи для їх отримання та оновлення;

– services – сервіси для обробки бізнес-логіки, включають методи для створення та оновлення у базі даних за допомогою providers;

– providers – провайдери для відправлення запитів до API;

– API controllers – контролери для обробки запитів.

На діаграмі розгортання (рис. 3.6) зображено архітектуру системи, яка складається з трьох основних компонентів: сервер з базою даних, сервер хостингу застосунку та обробку запитів та пристрій користувача з браузером.

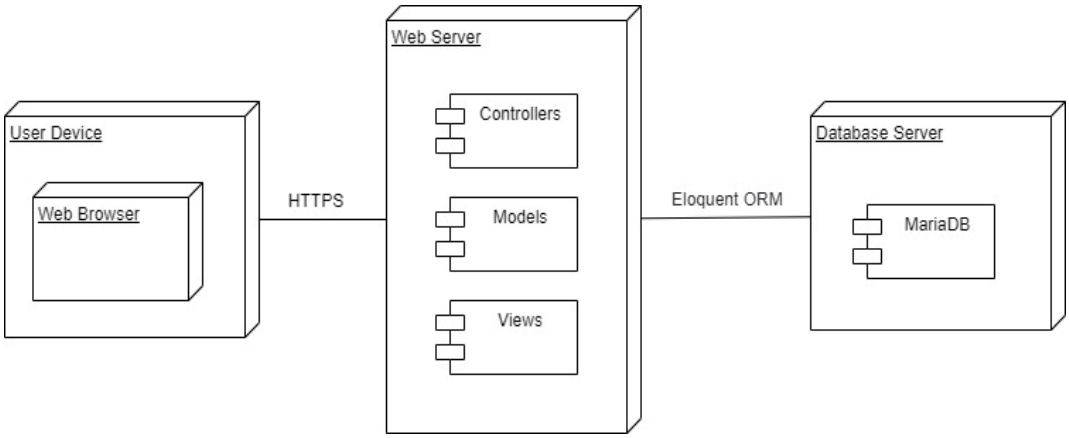


Рисунок 3.6 – Діаграма розгортання

Взаємодія між браузером на пристрої користувача та сервером хостингу застосунку відбувається через HTTP-запити та відповіді. Сервер хостингу застосунку використовує Eloquent для взаємодії з базою даних, який генерує SQL-запити для отримання, зберігання та обробки даних, які надсилаються до сервера бази даних.

Висновки до розділу 3

У третьому розділі роботи було розглянуто використані технології розробки, що включає опис технологій, їх переваги та причини їх використання.

Далі було складено діаграму сутність-зв’язок, яка є важливим етапом для проєктування бази даних. ER-діаграма допомагає визначити сутності, їх атрибути та взаємозв’язки.

Було створено діаграму класів, що відображає структуру системи, визначає взаємозв'язки між класами та ілюструє їхню функціональність, діаграми компонентів, які відображають архітектуру ПЗ з використанням патерну проектування MVC. Вони дозволяють визначити, як система буде розбита на окремі компоненти та як вони будуть взаємодіяти між собою. Також було створено діаграму розгортання для моделювання фізичного розміщення компонентів системи.

4 РОЗРОБКА ВЕБЗАСТОСУНКУ ТА РЕЗУЛЬТАТИ

4.1 Установка необхідних пакетів та бібліотек

Крім самого Laravel, були встановлені додаткові пакети та бібліотеки для розширення функціональних можливостей проєкту. Файл composer.json містить інформацію про всі PHP-бібліотеки та пакети, необхідні для роботи проєкту. На рисунку 4.1 наведено перелік основних пакетів та бібліотек, які було використано, а також їх призначення.

Рисунок 4.1 – Список встановлених пакетів

Отже, встановлено такі додаткові пакети:

– laravel/ui – пакет для генерації базових інтерфейсів автентифікації;

– fedeisas/laravel-mail-css-inliner – метою цього пакета є автоматизація процесу вбудовування CSS перед надсиланням електронних листів [20];

– gumlet/php-image-resize – бібліотека для зміни розміну розбражень у PHP;

– guzzlehttp/guzzle – клієнт для відправки HTTP-запитів;

– laravel/sanctum – пакет для аутентифікації API через токени;

– lis-dev/nova-poshta-api-2 – пакет для роботи з API Нової Пошти.

Файл package.json містить інформацію про всі JavaScript-бібліотеки та пакети, необхідні для роботи фронтенд-частини проекту. На рисунку 4.2 наведено вміст файлу, а саме установлені пакети та бібліотеки.

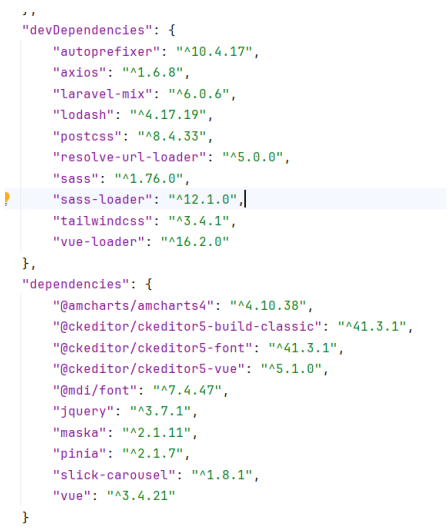


Рисунок 4.2 – Частина вмісту файлу package.json

Опис встановлених пакетів:

– @amcharts/amcharts4 – це бібліотека для створення інтерактивних графіків та діаграм;

– tailwindcss – CSS-фреймворк;

– vue-loader – завантажувач для Vue-компонентів у Webpack;

– @ckeditor/ckeditor5-build-classic – класична збірка CKEditor5, редактору тексту;

– @ckeditor/ckeditor5-font – плагін для управління шрифтами в CKEditor5;

– @ckeditor/ckeditor5-vue – інтеграція CKEditor5 з Vue;

– @mdi/font – файл, який містить в собі набір іконок згідно з Material Design Icons;

– jquery – бібліотека для спрощення роботи з DOM;

– pinia – state-менеджер для vue;

– slick-carousel – jQuery плагін для створення каруселей;

– vue – фреймворк для побудови інтерфейсів користувача.

4.2 Створення бази даних

Наступним етапом розробки вебзастосунку було створення бази даних. Для створення, редагування та видалення таблиць у Laravel використовуються міграції. Міграції – це як контроль версій бази даних, що дозволяє команді визначати та надавати спільний доступ до визначення схеми бази даних програми [21]. На рисунку 4.1 наведено приклад міграції, а саме – код для створення таблиці товарів.

```
class CreateItemsTable extends Migration
{

    Kateryna Shkidina
    public function up()
    {
        Schema::create( table: 'items', function (Blueprint $table) {
            $table->id();
            $table->string( column: 'name',  length: 100);
            $table->string( column: 'model',  length: 100);
            $table->string( column: 'slug',  length: 100);
            $table->foreignId( column: 'category_id')
                ->references( column: 'id')->on( table: 'categories');
            $table->foreignId( column: 'producer_id')
                ->references( column: 'id')->on( table: 'producers');
            $table->float( column: 'purchase_price')->default( value: 0);
            $table->float( column: 'price')->default( value: 0);
            $table->unsignedMediumInteger( column: 'amount')->default( value: 0);
            $table->text( column: 'desc')->default( value: null);
            $table->string( column: 'meta_keywords')->nullable()->default( value: null);
            $table->string( column: 'meta_description')->nullable()->default( value: null);
            $table->timestamps();
            $table->softDeletes();
        });
    }

    Kateryna Shkidina
    public function down()
    {
        Schema::dropIfExists( table: 'items');
```

Рисунок 4.3 – Код міграції для таблиці товарів

Після запуску міграцій було автоматично створено таблиці бази даних.

4.3 Реалізація CRUD-операцій

Після створення моделей сутностей для реалізації **CRUD** необхідно створити контроллер з ресурсними методами, які було описано у 3 розділі (index(), show($id), create(), store(Request $request), edit($id), update(Request $request, $id), destroy($id)). Приклад **CRUD** буде продемонстровано на сутності товарів. На рисунку 4.4 продемонстровано частину коду ItemController, а саме метод index для відображення списку товарів. У ньому робиться запит до таблиці, яка належить моделі Item. Товари відображаються за фільтрами, якщо вони обрані (к-ть товарів на сторінку, категорія, виробник, атрибут) і також сортуються (за замовчуванням –

за датою створення). Метод фільтрації прописано у моделі Item. У додатку А наведено повний код моделі Item. Метод index() повертає представлення index.blade.php (рис. 4.5) і передає йому товари. Слід зазначити, що для кожного методу було створено кастомні запити з налаштованими правилами валідації.

```
Kateryna Shkidina
public function index(IndexItemRequest $request)
{
    $page = config( key: 'nav.admin')[Route::currentRouteName()];

    $filters['data'] = $request->only( keys: 'page', 'perPage', 'sort', 'category', 'producer', 'attribute');
    $filters['config'] = CatalogService::getFilterConfig();

    return view( view: 'admin.catalog.items.index', [
        'page' => $page,
        'filters' => $filters,
        'data' => CatalogService::getItems($filters['data'])->withTrashed()
            ->paginate($filters['data']['perPage'] ?? config( key: 'catalog.perPage')[0]),
    ]);
}
```

Рисунок 4.4 – Метод index()

```
@extends('layouts.admin')

@section('title', $page['title'])

@section('count', $data->total())

@section('route', route(Route::currentRouteName()))

@section('nav')
    @include('admin.catalog.items.partials.nav')
@endsection

@section('filters')
    @include('admin.catalog.items.partials.filters.attribute-filter')
    @include('admin.catalog.items.partials.filters.category-filter')
    @include('admin.catalog.items.partials.filters.producer-filter')
@endsection

@section('buttons')
    <x-button :modal="true" target="#AdminCatalogItemCreateModal" icon="plus"/>
@endsection

@section('content')
    <div class="px-6 md:container md:mx-auto">
        <div class="w-full grid grid-cols-2 md:grid-cols-3 lg:grid-cols-5 gap-4">
            @foreach($data as $item)
                <x-catalog.admin-item :item="$item"/>
            @endforeach
        </div>
    </div>

    @include('admin.catalog.items.create')
@endsection
```

Рисунок 4.5 – Вміст index.blade.php

Наступним кроком є створення товару. Для відображення сторінки створення існує метод create(), але у даному контроллері він відсутній, оскільки створення товару відбувається у модальному вікні, а не на окремій сторінці. У додатку Б наведено вміст модального вікна для створення товару.

Метод store() обробляє дані з форми create і зберігає новий запис у базі даних (рис 4.6) і, за успішного створення, робить перенаправлення на сторінку index.

```
Kateryna Shkidina
public function store(StoreItemRequest $request)
{
    CatalogService::createItem(
        $request->only( keys: CatalogService::ITEM_ATTRIBUTES),
        $request->file( key: 'images'),
    );

    return redirect()->route( route: 'admin.catalog.items.index')->withSuccess('Створено');
}
```

Рисунок 4.6 – Метод store()

Щоб не нагромаджувати контроллер логікою створення товару, її винесено в окремий сервісний клас CatalogService. На рисунку 4.7 наведено метод створення товару.

```
public static function createItem(array $attributes, iterable $images)
{
    $attributes['slug'] = Str::slug($attributes['name']);
    $item = Item::create($attributes);

    self::addItemImages($item, $images);
```

Рисунок 4.7 – Метод createItem()

Наступним методом контроллеру є метод show(), який відображає конкретний товар та повертає представлення show.blade.php. Код цього методу представлено на рисунку 4.8. У додатку В наведено код представлення show.blade.php.

```
Kateryna Shkidina
public function show(Item $item)
{
    $page = config( key: 'nav.admin')[Route::currentRouteName()];

    return view( view: 'admin.catalog.items.show', [
        'page' => $page,
        'data' => $item,
    ]);
}
```

Рисунок 4.8 – Метод show()

Метод edit() повертає представлення edit.blade.php, у якому знаходиться форма редагування. А метод update обробляє дані форми і оновлює запис у базі

даних. На рисунку 4.9 показано код цих методів. У додатку Г наведено код сторінки edit.

```
Kateryna Shkidina
public function edit(Item $item)
{
    $page = config( key: 'nav.admin')[Route::currentRouteName()];
    $filters['config'] = CatalogService::getFilterAdvancedConfig();
    $filters['config']['current_attributes'] = ($item->attributes ?? collect([]))
        ->pluck( value: 'value', key: 'attribute_id');

    return view( view: 'admin.catalog.items.edit', [
        'page' => $page,
        'filters' => $filters,
        'data' => $item,
    ]);
}

Kateryna Shkidina
public function update(UpdateItemRequest $request, Item $item)
{
    CatalogService::updateItem(
        $item,
        $request->only( keys: CatalogService::ITEM_ATTRIBUTES),
        $request->collect( key: 'attributes'),
        $request->file( key: 'images'),
    );

    return redirect()->route( route: 'admin.catalog.items.index')->withSuccess('Збережено');
}
```

Рисунок 4.9 – Методи edit() та update()

Логіку оновлення товару також було винесено у сервіс CatalogService, метод updateItem() (рис 4.10).

```
1 usage   Kateryna Shkidina
public static function updateItem(Item $item, array $attributes, iterable $attrs = [], ?iterable $images = null)
{
    $attributes['slug'] = Str::slug( title: $attributes['name'] . ' ' . $attributes['model']);
    $item->update($attributes);

    $item->syncAttributes($attrs
        ->filter(function ($attr) {
            return $attr['value'];
        })
        ->toArray());

    if ($images) {
        self::addItemImages($item, $images);
    }
}
```

Рисунок 4.10 – Метод updateItem()

Останній метод контроллера, destroy(), видаляє конкретний запис (рис. 4.11)

```
public function destroy(Item $item)
{
    $item->delete();

    return redirect()->route( route: 'admin.catalog.items.index')->withSuccess('Видалено');
}
```

Рисунок 4.11 – Метод destroy()

4.4 Сервіси для роботи з файлами та для обробки зображень

Для полегшення роботи з файловою системою було створено клас FileService. У ньому містяться наступні методи:

– getInstance() – повертає об'єкт класу, що реалізує інтерфейс Filesystem;

– save() – метод для збереження файлу у вказаному шляху;

– copy() – метод копіювання файлу з одного місця у інше;

– delete() – метод видалення файлу за вказаним шляхом.

Клас ImageService надає функціональність для роботи з зображеннями, зокрема зміну їх розмірів та обрізання, використовуючи бібліотеку «Gumlet\ImageResize». Його методи:

– getImageInstance() – повертає інстанс класу ImageResize для роботи з зображенням за вказаним шляхом;

– getPath() – повертає повний шлях до файлу;

– resizeX() – змінює розмір зображення за шириною;

– resizeY() – змінює розмір зображення за висотою;

– resize() – змінює розмір зображення до заданих ширини та висоти;

– crop() – обрізає зображення до заданих ширини та висоти;

– cropCenter() – обрізає зображення до заданих ширини та висоти по центру.

Ці сервіси використовуються для роботи з зображеннями товарів, логотипами виробників, банерами спеціальних пропозицій та підбірок.

Повний код цих сервісів подано у додатку Д.

4.5 Використання Vue.js

Як вже зазначалось, для створення модальних вікон створення та редагування замовлень, спеціальних пропозицій та підбірок, панелі адміністратора, кошика та форми оформлення замовлення, використовувався фреймворк Vue.js замість шаблонізатора blade через його здатність забезпечувати динамічний рендеринг та

реактивність даних. Приклад використання цього фреймворка буде наведено на створенні замовлення з адмін-панелі.

Перш за все, було створено Vue-компонент. Він складається з шаблону (template), що включає розмітку HTML та скрипта, у якому імпортуються необхідні бібліотеки та компоненти, такі як mapActions і mapState з Pinia для управління станом, а також додаткові компоненти. Скрипт складається з наступного:

– data() – Визначає стан компоненту, включаючи дані про замовлення (користувач, товари, область, місто, відділення, спосіб оплати, опис, джерело) та змінні для обробки обраних товарів та їх кількості;

– computed – використовує mapState для доступу до даних з Pinia store. Також тут визначається чи знаходиться компонент у режимі редагування;

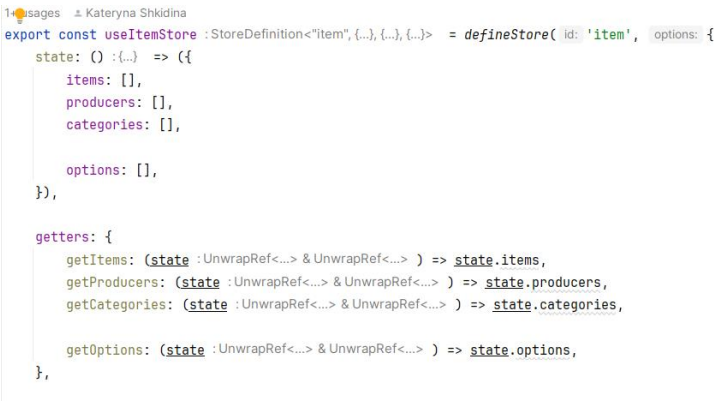
– methods – використовує mapActions для виклику методів з stores;

– watch – Спостерігає за змінами в поточному замовленні та оновлює дані форми, якщо компонент знаходиться в режимі редагування;

– created() – викликає методи для завантаження початкових даних при створенні компоненту. Якщо компонент знаходиться в режимі редагування, завантажує дані поточного замовлення.

Повний код цього компоненту наведено у додатку Е.

Stores використовуються для управління станом та даними, які зазвичай використовуються в кількох компонентах. Використання stores дозволяє зберігати ці дані централізовано та забезпечує доступ до них з будь-якого компонента в додатку. У даному компоненті використовуються UserStore для отримання користувачів, ItemStore для отримання даних про товари, NovaPoshtaStore для отримання списку міст та відділень пошти та OrderStore для керування даними про замовлення. Структуру store продемонстровано на прикладі ItemStore (рис. 4.12, 4.13).



```
1 usages  Kateryna Shkidina
export const useItemStore : StoreDefinition<"item", {...}, {...}, {...}>  = defineStore( id: 'item',  options: {
    state: () :{...}  => ({
        items: [],
        producers: [],
        categories: [],

        options: [],
    }),

    getters: {
        getItems: (state : UnwrapRef<...> & UnwrapRef<...> ) => state.items,
        getProducers: (state : UnwrapRef<...> & UnwrapRef<...> ) => state.producers,
        getCategories: (state : UnwrapRef<...> & UnwrapRef<...> ) => state.categories,

        getOptions: (state : UnwrapRef<...> & UnwrapRef<...> ) => state.options,
    },
```

Рисунок 4.12 – Частина коду ItemStore

```
    actions: {
        async fetchItems(filters) : Promise<void>  {
            try {
                const res = await CatalogService.fetchItems(filters)
                this.items = res.data.data
            } catch (e) {

            }
        },

        async fetchProducers() : Promise<void>  {
            try {
                const res = await CatalogService.fetchProducers()
                this.producers = res.data.data
            } catch (e) {

            }
        },

        async fetchCategories() : Promise<void>  {
            try {
                const res = await CatalogService.fetchCategories()
                this.categories = res.data.data
            } catch (e) {

            }
        },

        async fetchOptions() : Promise<void>  {
            try {
                const res = await CatalogService.fetchOptions()
                this.options = res.data.data
            } catch (e) {
```

Рисунок 4.13 – Частина коду ItemStore

У цьому коді:

– state – зберігає стан, включаючи товари, виробників, категорії та опції;

– getters – визначає методи для доступу до стану;

– actions – визначає методи для взаємодії з API та оновлення стану.

Методи для взаємодії з API знаходяться у сервісах, у даному випадку CatalogService.js. На рисунку 4.14 наведено приклад запитів до API для отримання даних.

```
export default {
    async fetchItems(filters :{}  = {}) : Promise<any>  {
        return await ApiRequestProvider._get( endpoint: `${PREFIX}/items?${objectToQueryParams(filters).toString()}`)
    },

    async fetchProducers() : Promise<any>  {
        return await ApiRequestProvider._get( endpoint: `${PREFIX}/producers`)
    },
```

Рисунок 4.14 – Приклади запитів на отримання даних (список товарів та виробників)

На рисунку 4.15 наведено приклад POST-запиту на сервер з даними про нове замовлення з файлу OrderService.js.

```
async createOrder(order) : Promise<any>  {
    const formData : FormData  = new FormData

    formData.append( name: `user_id`,  value: order.user_id ?? '')
    formData.append( name: `user_name`,  value: order.user_name ?? '')
    formData.append( name: `user_phone`,  value: order.user_phone ?? '')
    formData.append( name: `user_email`,  value: order.user_email ?? '')
    formData.append( name: `payment_method_id`, order.payment_method_id)
    formData.append( name: `desc`,  value: order.desc ?? '')
    formData.append( name: `area`,  value: order.area ?? '')
    formData.append( name: `city`,  value: order.city ?? '')
    formData.append( name: `warehouse`,  value: order.warehouse ?? '')
    addItems(formData, order.items)

    return await ApiRequestProvider._post( endpoint: 'orders', formData)
},
```

Рисунок 4.15 – Приклад POST-запиту для створення замовлення

Stores забезпечують управління станом додатка, тоді як сервіси відповідають за взаємодію з зовнішніми ресурсами. Використання stores і сервісів дозволяє зберігати код чистим, організованим та легким для розуміння та підтримки.

4.6 Відправка сповіщень

Сповіщення надсилаються на електронну пошту при створені акаунта для його верифікації або при такому запиті від користувача, також при створенні замовлень та зміні їх статусу. На рисунку 4.16 наведено код класу OrderCreatedNotification, який розширює базовий клас Notification, для відправки сповіщення про створене замовлення.

```php
2 usages  Kateryna Shkidina *
class OrderCreatedNotification extends Notification
{
    use Queueable;

    public Order $order;

    1 usage  Kateryna Shkidina
    public function __construct(Order $order)
    {
        $this->order = $order;
    }

    Kateryna Shkidina
    public function via($notifiable)
    {
        return ['mail'];
    }

    no usages  Kateryna Shkidina
    public function toMail($notifiable)
    {
        return (new MailMessage)
            ->subject(__( key: 'order.status.created'))
            ->view( view: 'mails.orders.created', ['order' => $this->order]);
    }
```

Рисунок 4.16 – Клас OrderCreatedNotification

У цьому класі конструктор отримує замовлення, яке буде використовуватися в повідомленні. Метод via($notifiable) повертає масив з одним значенням «mail», що означає, що це сповіщення буде доставлене по електронній пошті. Метод toMail($notifiable) генерує електронне повідомлення, яке буде відправлене, він встановлює тему повідомлення і використовує шаблон mails.orders.created, у якому прописано розмітку email-повідомлення.

4.7 Керування рівнем доступу

Аби виключити зловживання рівнем доступу, тобто заборонити клієнтам доступ до адмін панелі, менеджеру – доступ до керування клієнтами та контентом на сторінках, було створено middleware для перевірки ролі користувача під час доступу до певних ресурсів (рис. 4.17).

```php
class Role
{
    new *
    public function handle(Request $request, Closure $next, ... $roles)
    {
        if($request->user()->isAdmin) return $next($request);
        if($request->user()->isBanned) abort( code: 403);

        foreach($roles as $role) {
            if($request->user()->role_id == $role) return $next($request);
        }

        abort( code: 404);
```

Рисунок 4.17 – Код middleware Role

У основному методі «handle» обробляються запити користувачів. Перевіряються ролі користувача. Якщо він адміністратор, йому дозволяються усі запити, якщо забанений – клієнт не має доступу ні до чого. Після виконується перевірка, чи користувач має необхідну роль, щоб отримати доступ до ресурсу, роль перевіряється порівнянням ідентифікатора ролі користувача з переліком ролей, переданим у параметрі $roles. На рисунку 4.18 наведено приклад використання цього middleware у роутері.

```
Route::middleware( middleware: 'auth')->group(function () {
    Route::prefix( prefix: 'admin')->middleware(['verified', 'role:' . User::ROLE_ADMIN . ',' . User::ROLE_MANAGER])->name( value: 'admin.')->group(function ()

        Route::get( uri: '/', [AdminPageController::class, 'index'])->name( name: 'index');
        Route::get( uri: '/search',  action: SearchController::class)->name( name: 'search');
```

Рисунок 4.18 – Використання middleware Role

Наведений фрагмент коду забезпечує доступ до маршрутів з префіксом «/admin» тільки для аутентифікованих користувачів, які мають підтверджену електронну пошту та ролі адміністратора або менеджера.

4.8 Огляд готового вебзастосунку

На рисунку 4.19 зображено головну сторінку сайту інтернет-магазину сонцезахисних окулярів. У верхній частині розташовано хедер з контактною інформацією, посиланнями та навігацією. Нижче знаходиться карусель з акційними пропозиціями, при натисканні на кнопку акції відбувається перенаправлення на сторінку цієї пропозиції. Нижче розміщено логотипи брендів, окуляри яких продаються у цьому магазині. При натисканні на логотип користувач перенаправляється до каталогу, у якому будуть відображатись товари обраного бренду. Далі знаходиться банер підбірки окулярів з заголовком, описом та кнопкою, при натисканні на яку активується перехід до каталогу з заданими параметрами підбірки. Далі можна побачити популярні товари. У нижній частині розташовано футер із загальною інформацією, посиланнями на соціальні мережі та додатковою навігацією по сайту.

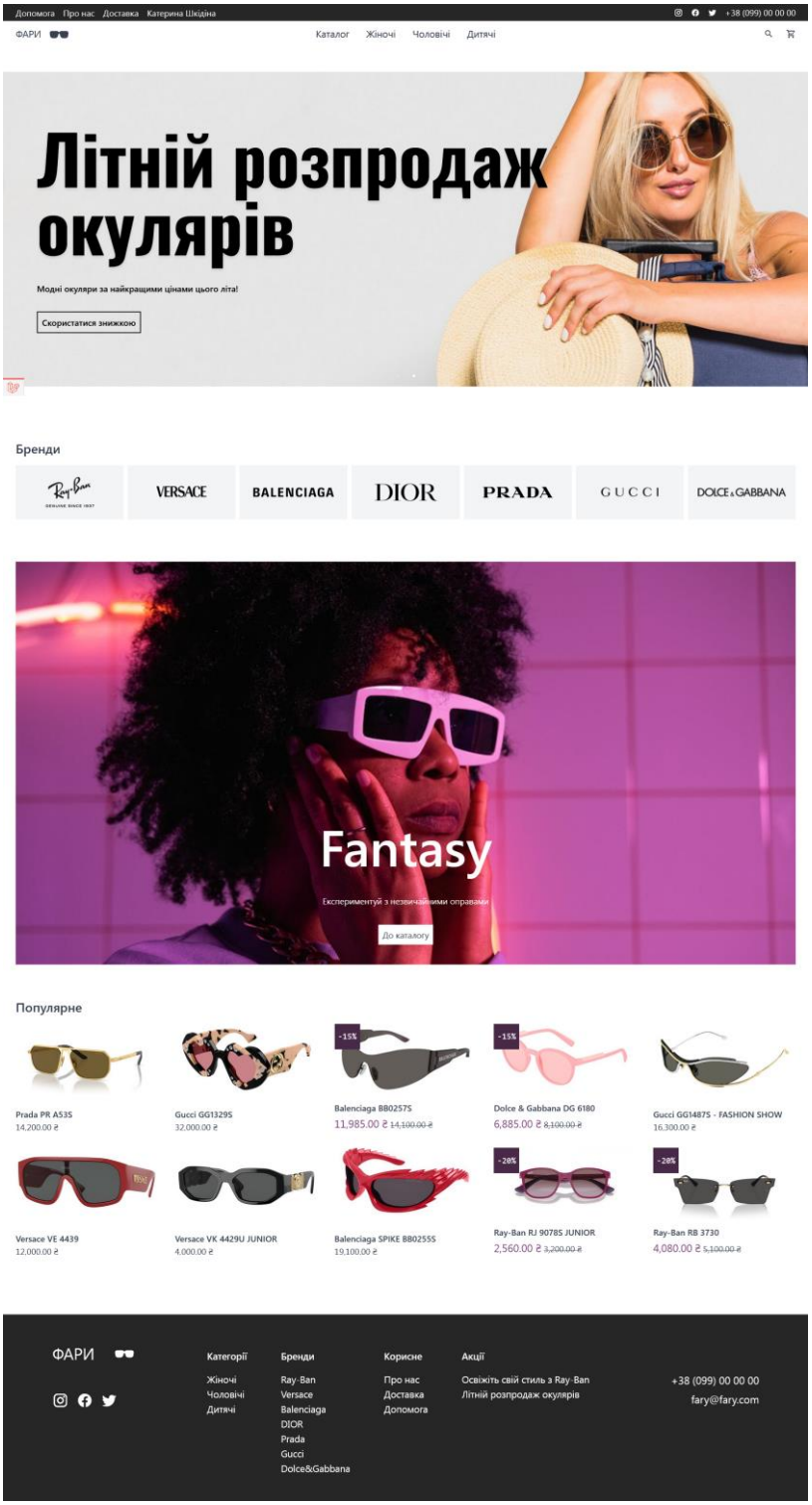


Рисунок 4.19 – Головна сторінка магазину

На рисунку 4.20 зображено частину сторінки каталогу. Зліва знаходятся посилання на акції та підбірки, посередині знаходяться товари, справа – кнопка, яка відкриває модальне окно фільтрів.

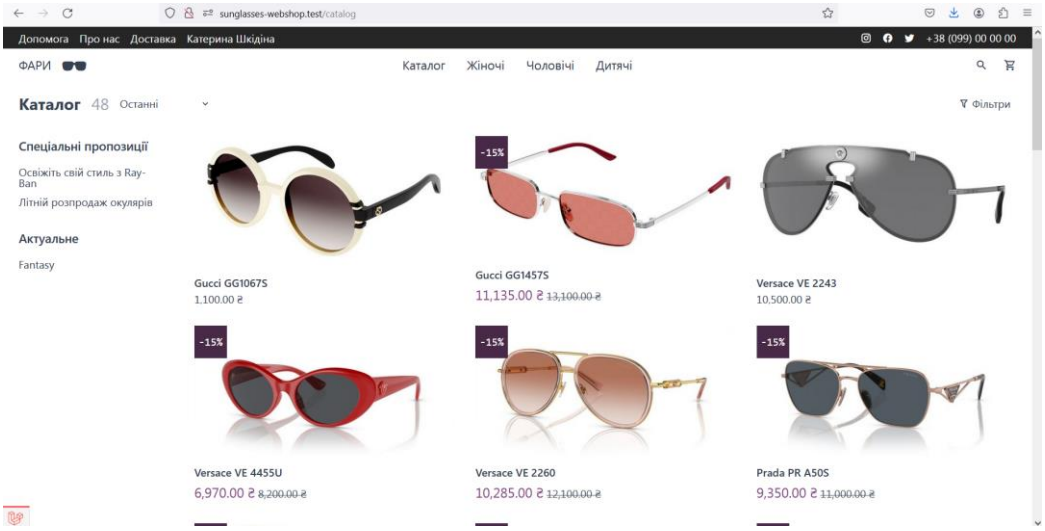


Рисунок 4.20 – Сторінка каталогу

На рисунку 4.21 продемонстровано адаптовані під мобільний пристрій сторінку каталогу (зліва) з товарами з заданими фільтрами, справа показано модальне вікно фільтрів.

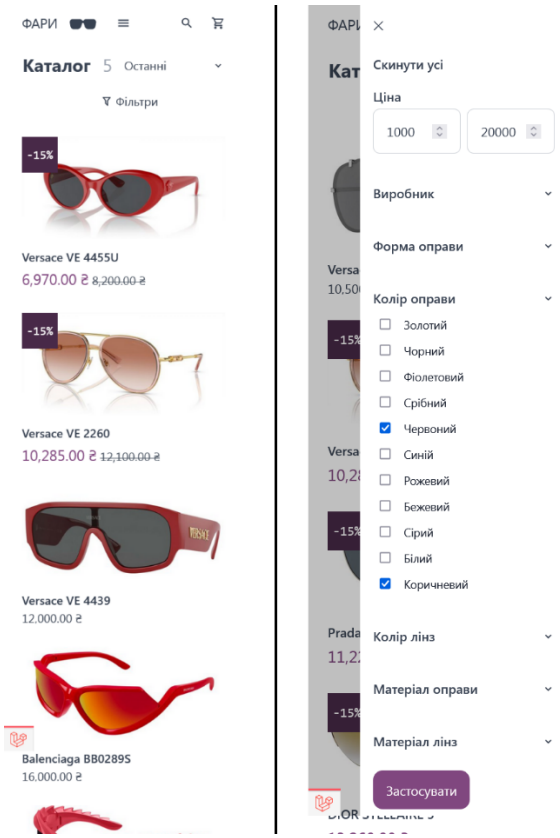


Рисунок 4.21 – Сторінка каталогу на мобільному пристрої (зліва), фільтри (справа)

На рисунку 4.22 показано сторінку товару. На ній розміщено карусель з зображеннями товару, справа знаходиться деяка інформація про товар, кнопка «Про цю модель», при натисканні на яку відкривається модальне вікно з повною інформацією про товар, нижче за неї – кнопка додавання до кошика.

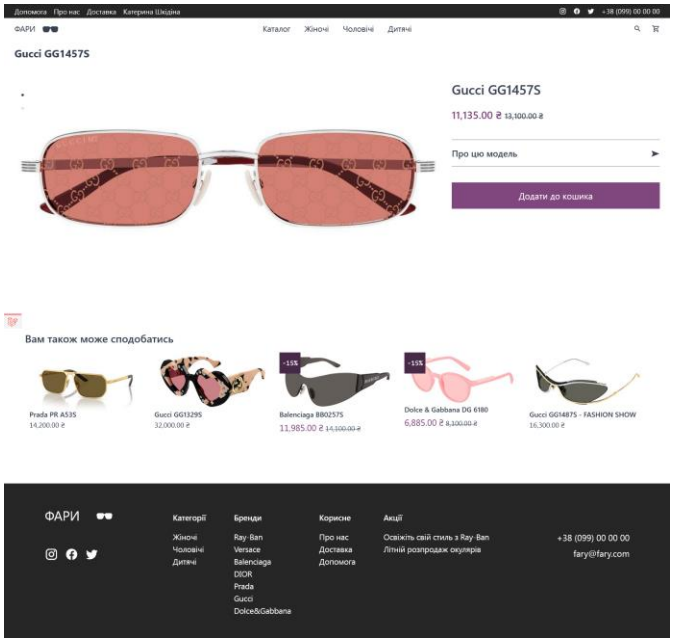


Рисунок 4.22 – Сторінка товару

На рисунку 4.23 зображено модальне вікно кошика. У ньому можна змінити кількість товарів, видалити товар звідти та перейти до оформлення замовлення.

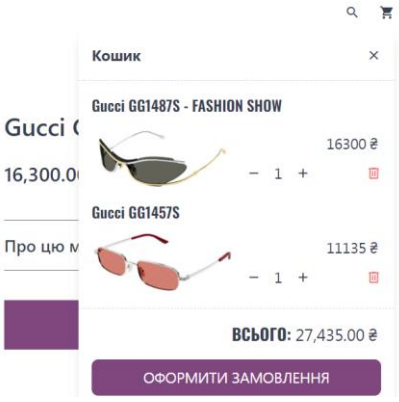


Рисунок 4.23 – Модальне вікно кошика

На рисунку 4.24 продемонстровано сторінку оформлення замовлення для незареєстрованого користувача. Для замовлення необхідно ввести контактну інформацію, спосіб оплати, обрати місто та відділення Нової Пошти, якщо

потрібно – комментар. Також можна змінювати кількість товарів та видаляти їх. За потреби цей користувач може зареєструватись саме під час оформлення замовлення. Для цього він має додатково ввести пароль.

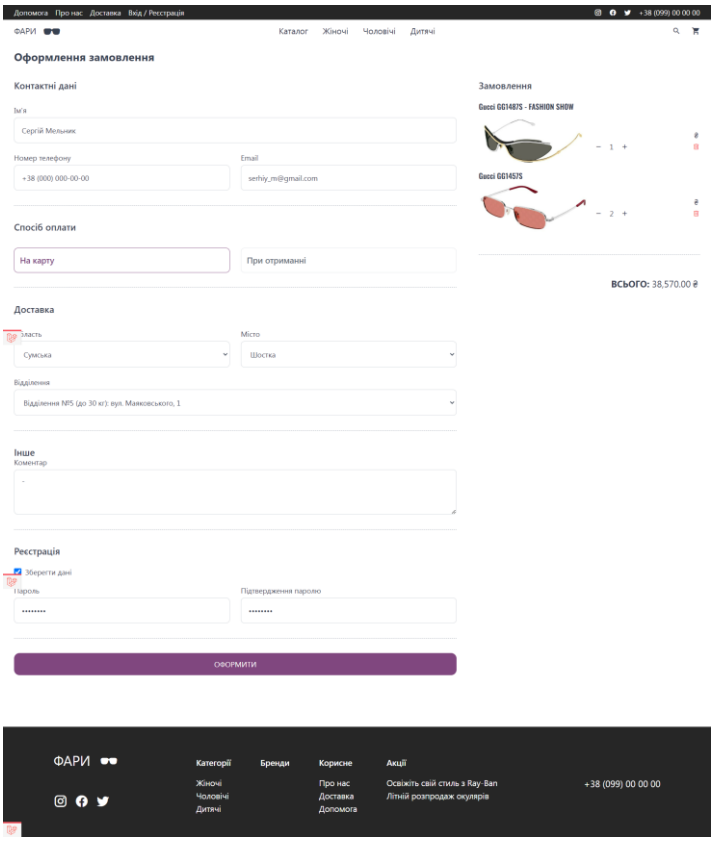


Рисунок 4.24 – Сторінка оформлення замовлення

Після успішного створення замовлення клієнту приходить повідомлення про це на електронну адресу. На рисунку 4.25 показано приклад такого повідомлення.

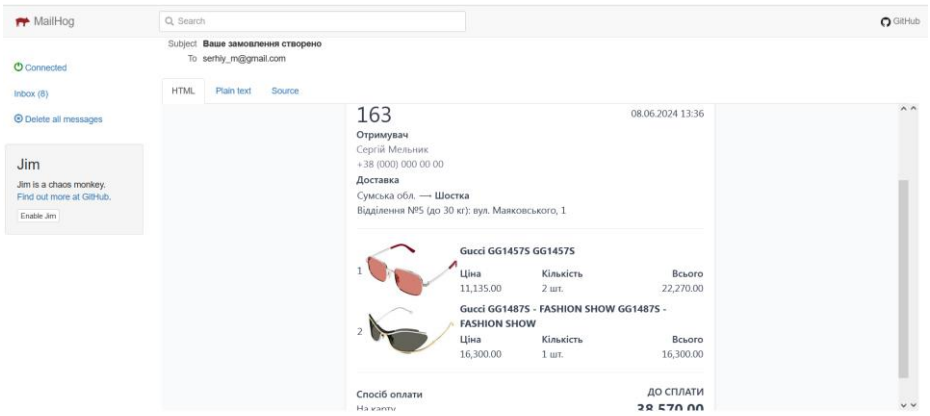


Рисунок 4.25 – Повідомлення про створене замовлення

У особистому кабінеті користувач може переглядати та редагувати особисту інформацію, вийти з акаунту а також переглядати історію замовлень. На рисунку 4.26 продемонстровано сторінку замовлень користувача.

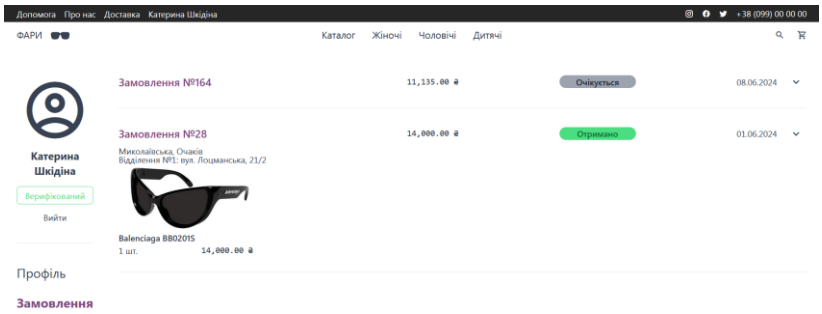


Рисунок 4.26 – Сторінка замовлень з особистого кабінету користувача

Адміністративна панель

Оскільки панель адміністратора містить багато сторінок, буде продемонстровано лише ключові.

На головній сторінці панелі адміністратора можна переглянути необхідну статистику продаж, кількість нових клієнтів, популярність джерел замовлень за різні проміжки часу (рис. 4.27).

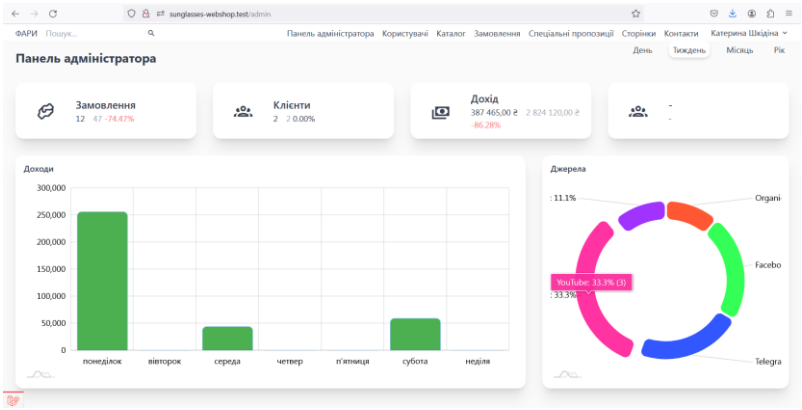


Рисунок 4.27 – Статистика за тиждень

Як вже зазначалось, адміністратор може управляти акаунтами користувачів, товарами, атрибутами товарів, замовленнями, акціями, підбірками, деякими сторінками, контактами. На рисунку 4.28 продемонстровано модальне вікно створення користувача адміністратором.

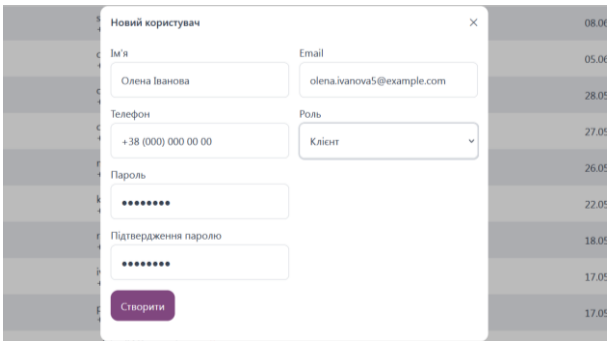


Рисунок 4.28 – Модальне вікно створення користувача

На рисунку 4.29 зображено частину сторінки списку замовлень. Їх можна фільтрувати за параметрами, зазначеними під хедером.

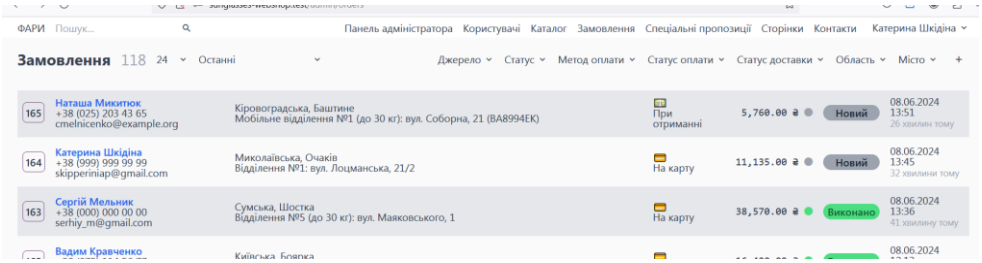


Рисунок 4.29 – Частина сторінки списку замовлень

На рисунку 4.30 продемонстровано сторінку замовлення, з якої можна перейти до редагування замовлення та змінювати його статуси.

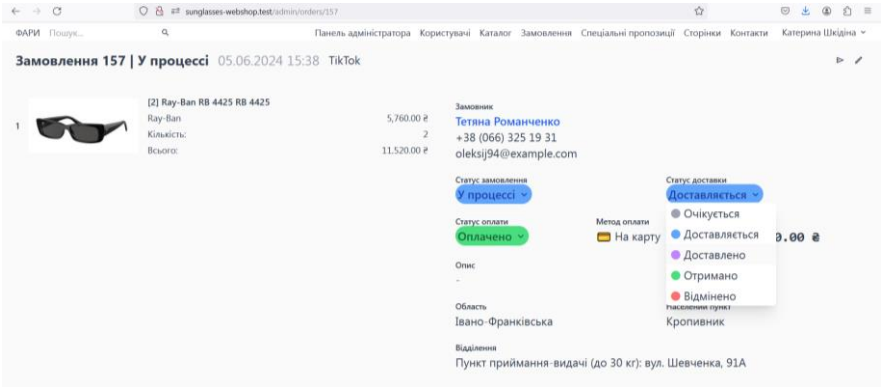


Рисунок 4.30 – Сторінка замовлення

На рисунку 4.31 зображено сторінку редагування товару. Зліва можна додати, видалити фото товару, а також обрати головне фото (для превью). Справа розташовано поля для редагування інформації про товар.

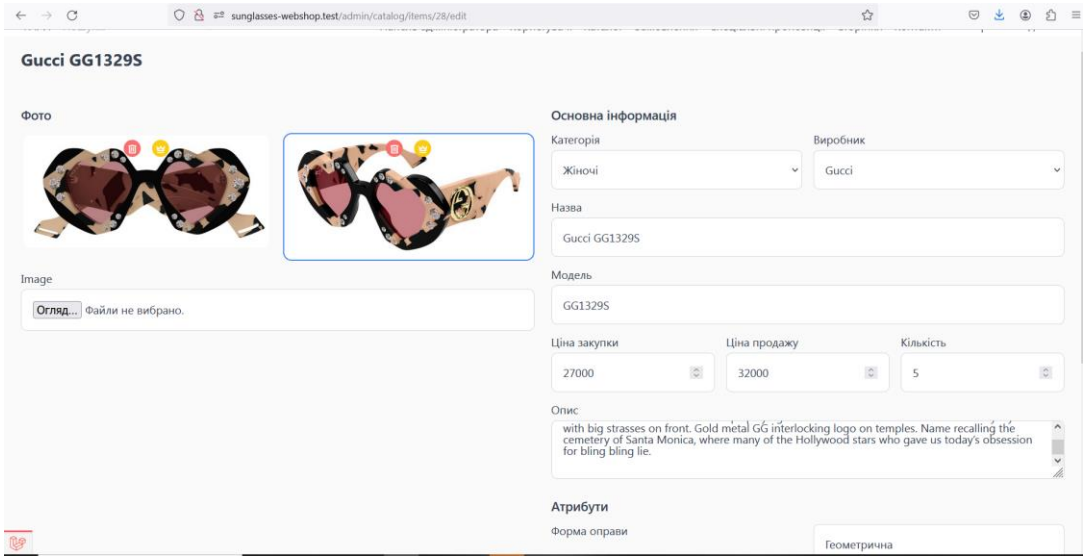


Рисунок 4.30 – Сторінка редагування товару

На рисунку 4.32 зображено частину модального вікна редагування спеціальної пропозиції (акції). У ньому менеджер/адміністратор може обрати необхідні товари, що беруть участь у акції, процент знижки, додати банер та іншу необхідну інформацію.

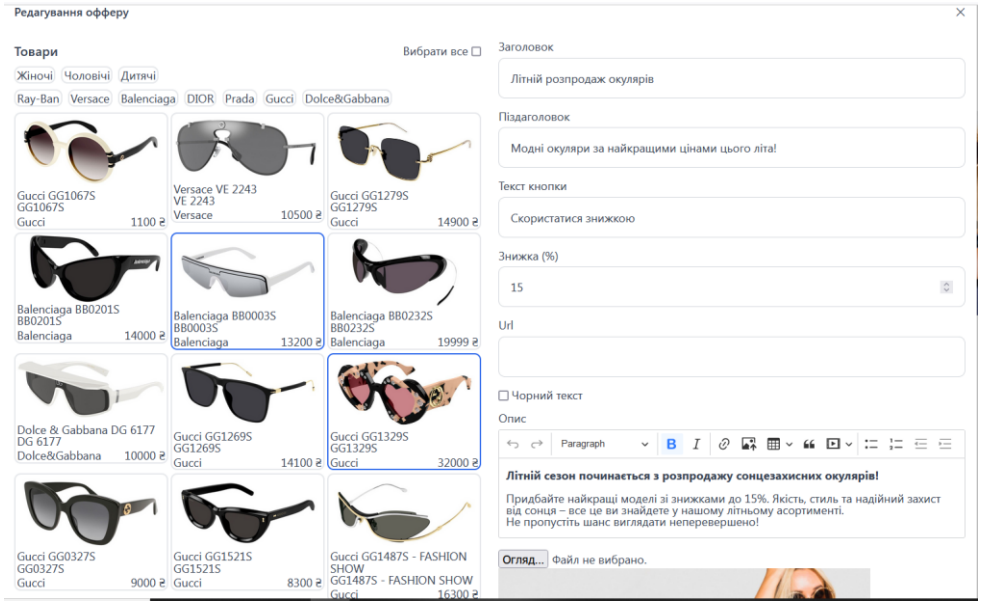


Рисунок 4.31 – Модальне вікно редагування акції

Адміністратор також може створювати та редагувати статичні сторінки (такі як «Про нас», «Допомога», «Доставка» і тому подібні). На рисунку 4.32 продемонстровано модальне вікно редагування такої сторінки.

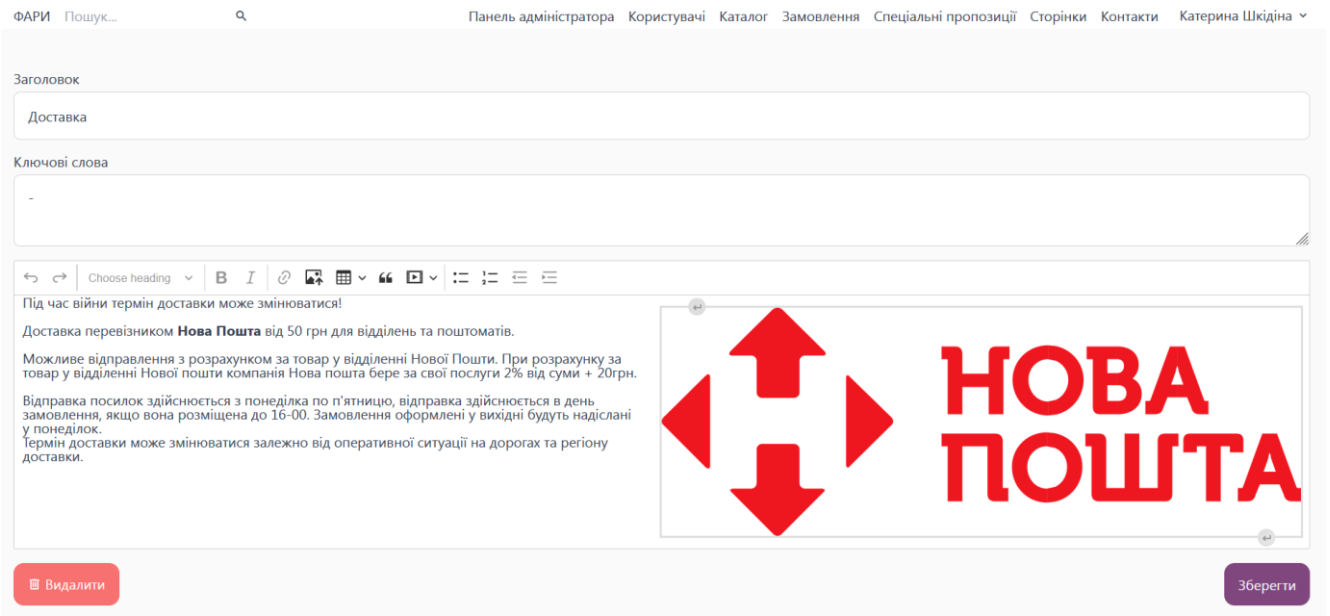


Рисунок 4.32 – Сторінка редагування статичної сторінки

Було інтегровано редактор Ckeditor WYSIWYG (What You See Is What You Get), за допомогою якого можна редагувати вміст сторінки так, як він відображається на екрані без необхідності знання HTML.

Висновки до розділу 4

У четвертому розділі кваліфікаційної роботи було описано кроки розробки вебзастосунку. Показано які залежності та бібліотеки було встановлено, створення бази даних, реалізацію CRUD-операцій, підключення Vuejs та використання API, описано основні сервіси та спосіб виключення зловживання рівнем доступу. За рахунок чого було закріплено навички роботи з мовами програмування PHP, JavaScript, їх фреймворками та бібліотеками – Laravel, Vuejs, jQuery та іншими.

Також було показано результати роботи над застосунком, описано інтерфейс застосунку та можливості користувачів.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було розроблено вебзастосунок для продажу сонцезахисних окулярів.

Для досягнення поставленої мети виконано наступні завдання:

- проаналізовано специфіку діяльності інтернет-магазинів;
- проаналізовано вимоги користувачів до інтернет-магазинів;
- розроблено дизайн інтернет-магазину;
- спроєктовано базу даних;
- розроблено інтерфейс та функціонал вебзастосунку.

Завдяки дослідженю предметної області та аналізу існуючих рішень отримано уявлення про діяльність інтернет-магазинів, потреби користувачів, визначено вимоги до програмного забезпечення. Також сформовано специфікацію вимог.

Для моделювання та проєктування ПЗ було розроблено 5 видів UML-діаграм для візуалізації взаємодії між користувачами та системою, розуміння архітектури системи і взаємозв’язків між її складовими та логіки роботи програмного забезпечення. Написано сценарії використання системи. Створено ER-діаграму для визначення структури бази даних та взаємозв’язків між її складовими. Відповідно до технічного завдання були обрані технології, які найбільш підходять для розробки цього програмного забезпечення.

Реалізовано усі функції, необхідні для діяльності інтернет-магазину, розроблено прибавливий та інтуїтивно зрозумілий інтерфейс з використанням сучасних технологій веброзробки.

Результатом виконання роботи є готовий вебзастосунок для продажу сонцезахисних окулярів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1) What is an online store and why do you need one? WIXBlog: вебсайт. URL: https://www.wix.com/blog/what-is-an-online-store. (Last accessed: 24.04.2024).

2) The Sunglasses Shop: вебсайт. URL: https://www.thesunglassesshop.co.uk. (Last accessed: 01.04.2024).

3) Polaroid: вебсайт. URL: https://polaroid.in.ua. (Дата звернення: 01.04.2024).

4) Multioptic: вебсайт. URL: https://multioptic.com.ua. (Дата звернення: 01.04.2024).

5) Use Case Diagrams | Unified Modeling Language (UML). GeeksforGeeks: вебсайт. URL: https://www.geeksforgeeks.org/use-case-diagram/. (Last accessed: 25.04.2024).

6) UML - Statechart Diagrams. Tutorialspoint: вебсайт. URL: https://www.tutorialspoint.com/uml/uml_statechart_diagram.htm (Last accessed: 17.05.2024).

7) PHP. URL: https://www.php.net (Last accessed: 17.05.2024).

8) JavaScript. Mdn web docs: вебсайт. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (Last accessed: 17.05.2024).

9) Laravel docs. Laravel: вебсайт. URL: https://laravel.com/docs/8.x (Last accessed: 20.05.2024).

10) Blade Templates. Laravel: вебсайт. URL: https://laravel.com/docs/8.x/blade (Last accessed: 20.05.2024).

11) Introduction. Pinia: вебсайт. URL: https://pinia.vuejs.org/introduction.html (Last accessed: 20.05.2024).

12) Introduction. Vue.js: вебсайт. URL: https://vuejs.org/guide/introduction.html (Last accessed: 23.05.2024).

13) Bartholomew D. MariaDB vs MySQL. Dostopano T. 7. №. 10. 2012. P. 1-6.

14) What is an Entity Relationship Diagram? Secoda: вебсайт. URL: https://www.secoda.co/glossary/entity-relationship-diagram (Last accessed: 26.04.2024).

15) TailwindCSS: вебсайт. URL: https://tailwindcss.com (Last accessed: 23.05.2024).

16) Singh S., Iyer J. Comparative study of MVC (model view controller) architecture with respect to struts framework and PHP. International Journal of Computer Science Engineering (IJCSE) №. 3. 2016. P. 142-150.

17) MVC. Mdn web docs: вебсайт. URL: https://developer.mozilla.org/en-US/docs/Glossary/MVC (Last accessed: 29.04.2024).

18) Class Diagram | Unified Modeling Language (UML). GeeksforGeeks: вебсайт. URL: https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/ (Last accessed: 22.05.2024).

19) Component Diagram Tutorial. Lucidchart: вебсайт. URL: https://www.lucidchart.com/pages/uml-component-diagram (Last accessed: 30.04.2024).

20) Laravel Mail CSS Inliner. GitHub. URL: https://github.com/fedeisas/laravel-mail-css-inliner (Last accessed: 25.05.2024).

21) Database: Migrations. Laravel: вебсайт. URL: https://laravel.com/docs/8.x/migrations (Last accessed: 25.05.2024).

Додаток А

Item.php

```
class Item extends Model
{
    use SoftDeletes, HasFactory, HasImages, HasAttributes, HasOffers;

    public const IMAGE_PATH = 'catalog/items';

    protected $fillable = [
        'name',
        'model',
        'slug',
        'category_id',
        'producer_id',
        'purchase_price',
        'price',
        'amount',
        'desc',
        'meta_keywords',
        'meta_description',
    ];

    public function category(): BelongsTo
    {
        return $this->belongsTo(Category::class);
    }

    public function producer(): BelongsTo
    {
        return $this->belongsTo(Producer::class);
    }

    public function images(): HasMany
    {
        return $this->hasMany(ItemImage::class);
    }

    public function mainImage(): HasOne
    {
        return $this->hasOne(ItemImage::class)->main();
    }

    public static function filterQuery(array $filters = [], Builder $builder =
null): Builder
    {
        $builder = $builder ?: Item::with('category', 'producer', 'images',
'mainImage', 'attributes', 'offer_item.offer');

        $search = (string)($filters['search'] ?? null);

        $attributes = (array)($filters['attribute'] ?? []);
        $categories = (array)($filters['category'] ?? []);
        $producers = (array)($filters['producer'] ?? []);

        $minPrice = (float)($filters['min_price'] ?? 0);
        $maxPrice = (float)($filters['max_price'] ?? 0);

        $sort = (string)($filters['sort'] ?? null);
```

```
        if ($search) {
            $builder->where('name', 'LIKE', "%$search%")
                ->orWhere('model', 'LIKE', "%$search%")
                ->orWhere('desc', 'LIKE', "%$search%");
        }

        if ($attributes) {
            foreach ($attributes as $attr) {
                $builder->whereHas('attributes', function ($query) use ($attr) {
                    $query->whereIn('value', $attr);
                });
            }
        }

        if ($categories) {
            $builder->whereIn('category_id', $categories);
        }

        if ($producers) {
            $builder->whereIn('producer_id', $producers);
        }

        if ($minPrice) {
            $builder->where('price', '>=', $minPrice);
        }

        if ($maxPrice) {
            $builder->where('price', '<=', $maxPrice);
        }

        if ($sort && $sort !== 'latest') {
            $sort = explode('-', $sort);

            $builder->orderBy($sort[0], $sort[1] ?? 'asc');
        } else {
            $builder->latest();
        }


        return $builder;
    }


    public function scopeInStock(Builder $builder)
    {
        $builder->where('amount', '>', 0);
    }

    public function scopeWithOffer(Builder $builder)
    {
        $builder->has('offer');
    }

    public function scopeWithoutOffer(Builder $builder)
    {
        $builder->doesntHave('offer');
    }

}
```

Додаток Б

admin.catalog.item.create.blade.php

```
<div id="AdminCatalogItemCreateModal" class="modal fade hidden" role="dialog"
aria-labelledby="AdminCatalogItemCreateModal" aria-modal="true">
    <div id="AdminCatalogItemCreateFrame" class="modal-frame-lg">
        <div class="modal-header px-4 flex justify-between items-center">
            <h3 class="font-semibold">Новий товар</h3>
            <span class="modal-close cursor-pointer">&#10005;</span>
        </div>

        <div class="modal-body p-4">
            <form id="AdminCatalogItemsCreateForm" action="{{
route('admin.catalog.items.store') }}" method="POST" enctype="multipart/form-
data">
                @csrf
                <div class="grid grid-cols-2 gap-6 py-4">

                    <div>
                        <x-form.input-image label="Image"
id="AdminCatalogItemsCreateFormImage" name="images[]" :multiple="true"/>
                    </div>

                    <ul class="grid grid-cols-6 gap-4">

                        <li class="col-span-3">
                            <x-form.select label="Категорія"
id="AdminCatalogItemsCreateFormCategory" name="category_id"

:options="$filters['config']['categories']"
                                          selected="{{ old('category_id') }}"
                            />
                        </li>

                        <li class="col-span-3">
                            <x-form.select label="Виробник"
id="AdminCatalogItemsCreateFormProducer" name="producer_id"

:options="$filters['config']['producers']"
                                          selected="{{ old('producer_id') }}"
                            />
                        </li>


                        <li class="col-span-full">
                            <x-form.input-text label="Назва"
id="AdminCatalogItemsCreateFormName" name="name" value="{{ old('name') }}"/>
                        </li>

                        <li class="col-span-full">
                            <x-form.input-text label="Модель"
id="AdminCatalogItemsCreateFormModel" name="model" value="{{ old('model') }}"/>
                        </li>


                        <li class="col-span-2">
                            <x-form.input-number label="Ціна закупки"
id="AdminCatalogItemsCreateFormPurchasePrice" name="purchase_price" value="{{
old('purchase_price') }}" step="0.5"/>
                        </li>
```

```
                        <li class="col-span-2">
                            <x-form.input-number label="Ціна продажу"
id="AdminCatalogItemsCreateFormPrice" name="price" value="{{ old('price') }}"
step="0.5"/>
                        </li>

                        <li class="col-span-2">
                            <x-form.input-number label="Кількість"
id="AdminCatalogItemsCreateFormAmount" name="amount" value="{{ old('amount') }}"/>
                        </li>

                        <li class="col-span-full">
                            <x-form.input-textarea label="Опис"
id="AdminCatalogItemsCreateFormDesc" name="desc" value="{{ old('desc') }}"/>
                        </li>
                    </ul>

                </div>

                <button type="submit" class="btn-lg btn-primary">Створити</button>
            </form>
        </div>
    </div>
</div>
```

Додаток В

admin.catalog.item.show.blade.php

```
@extends('layouts.admin')

@section('title', $data->name)

@section('buttons')
    <x-button route="admin.catalog.items.edit" :data="$data->id" dataKey="item"
icon="pencil-outline"/>
@endsection

@section('content')
    <div class="px-6 md:container md:mx-auto">
        <div class="grid grid-cols-2 gap-6 py-4">
            <ul class="grid grid-cols-2 gap-4">
                @foreach($data->images as $image)
                    <li class="relative rounded-xl">
                        <img src="{{ $image->path }}"
                             class="w-full h-auto rounded-xl border-4 {{ $image-
>main ? 'border-blue-500' : 'border-transparent' }}">
                    </li>
                @endforeach
            </ul>
            <div class="grow">
                <ul class="grid grid-cols-6 gap-4 text-xl">

                    <li class="col-span-3">
                        <span class="font-semibold text-sm">Категорія</span>
                        <p>
                            <a class="text-blue-500" href="{{
route('admin.catalog.items.index', ['category'=>[$data->category_id]]) }}">
                                {{ $data->category->name }}
                            </a>
                        </p>
                    </li>

                    <li class="col-span-3">
                        <span class="font-semibold text-sm">Виробник</span>
                        <p>
                            <a class="text-blue-500" href="{{
route('admin.catalog.items.index', ['producer'=>[$data->producer_id]]) }}">
                                {{ $data->producer->name }}
                            </a>
                        </p>
                    </li>


                    <li class="col-span-full">
                        <span class="font-semibold text-sm">Назва</span>
                        <p>{{ $data->name }}</p>
                    </li>

                    <li class="col-span-full">
                        <span class="font-semibold text-sm">Модель</span>
                        <p>{{ $data->model }}</p>
                    </li>


                    <li class="col-span-2">
                        <span class="font-semibold text-sm">Ціна закупки</span>
```

```
                        <p>{{ $data->purchase_price }}</p>
                    </li>

                    <li class="col-span-2">
                        <span class="font-semibold text-sm">Ціна продажу</span>
                        <p>{{ $data->price }}</p>
                    </li>

                    <li class="col-span-2">
                        <span class="font-semibold text-sm">Кількість</span>
                        <p>{{ $data->amount }}</p>
                    </li>

                    <li class="col-span-full">
                        <span class="font-semibold text-sm">Опис</span>
                        <p>{{ $data->desc }}</p>
                    </li>
                </ul>
                <div class="pt-8">
                    <h3 class="mb-3 text-lg font-semibold">Атрибути</h3>
                    <ul class="grid grid-cols-2 gap-4">
                        @foreach($data->attributes as $attribute)
                            <li>{{ $attribute->attribute->name }}</li>
                            <li>{{ $attribute->value }}</li>
                        @endforeach



                    </ul>
                </div>
            </div>
        </div>
    </div>
@endsection
```

Додаток Г

admin.catalog.item.edit.blade.php

```
@extends('layouts.admin')

@section('title', $data->name)

@section('content')
    <div class="px-6 md:container md:mx-auto">
        <form id="AdminCatalogItemsEditForm" action="{{
route('admin.catalog.items.update', $data->id) }}" method="POST"
              enctype="multipart/form-data">
            @csrf
            @method('PUT')
            <div class="grid grid-cols-2 gap-6 py-4">

                <div>
                    <h3 class="mb-3 text-lg font-semibold">Фото</h3>
                    <ul class="grid grid-cols-1 gap-4">
                        <li>
                            <ul class="grid grid-cols-2 gap-4">
                                @foreach($data->images as $image)
                                    <li class="relative rounded-xl border-2 {{
$image->main ? 'border-blue-500' : 'border-transparent' }}">
                                        <div class="w-full flex justify-center
gap-4 absolute top-2">
                                            <a href="{{
route('admin.catalog.items.deleteImage', ['item' => $data->id, 'image' => $image-
>id ]) }}"
                                               class="block w-6 h-6 p-1 rounded-
full bg-red-400 hover:bg-red-600 text-white text-center">
                                                <i class="mdi mdi-trash-can-
outline"></i>
                                            </a>

                                            <a href="{{
route('admin.catalog.items.setMainImage', ['item' => $data->id, 'image' => $image-
>id ]) }}"
                                               class="block w-6 h-6 p-1 rounded-
full bg-yellow-400 hover:bg-yellow-600 text-white text-center">
                                                <i class="mdi mdi-crown-
outline"></i>
                                            </a>

                                        </div>
                                        <img src="{{ $image->path }}"
                                             class="w-full h-auto rounded-xl
border-2 {{ $image->main ? 'border-white' : 'border-transparent' }}">
                                    </li>
                                @endforeach
                            </ul>
                        </li>

                        <li>
                            <x-form.input-image label="Image"
id="AdminCatalogItemsCreateFormImage" name="images[]"
                                                :multiple="true"
                                                :required="false"
                            />
                        </li>
                    </ul>
```

```
                </div>

                <div>
                    <h3 class="mb-3 text-lg font-semibold">Основна інформація</h3>
                    <ul class="grid grid-cols-6 gap-4">

                        <li class="col-span-3">
                            <x-form.select label="Категорія"
id="AdminCatalogItemsCreateFormCategory" name="category_id"

:options="$filters['config']['categories']"
                                           selected="{{ $data->category_id }}"
                            />
                        </li>

                        <li class="col-span-3">
                            <x-form.select label="Виробник"
id="AdminCatalogItemsCreateFormProducer" name="producer_id"

:options="$filters['config']['producers']"
                                           selected="{{ $data->producer_id }}"
                            />
                        </li>


                        <li class="col-span-full">
                            <x-form.input-text label="Назва"
id="AdminCatalogItemsCreateFormName" name="name"
                                               value="{{ $data->name }}"
                            />
                        </li>

                        <li class="col-span-full">
                            <x-form.input-text label="Модель"
id="AdminCatalogItemsCreateFormModel" name="model"
                                               value="{{ $data->model }}"/>
                        </li>


                        <li class="col-span-2">
                            <x-form.input-number label="Ціна закупки"
id="AdminCatalogItemsCreateFormPurchasePrice"
                                                 name="purchase_price"
                                                 value="{{ $data->purchase_price
}}"
                                                 step="0.01"/>
                        </li>

                        <li class="col-span-2">
                            <x-form.input-number label="Ціна продажу"
id="AdminCatalogItemsCreateFormPrice" name="price"
                                                 value="{{ $data->price }}"
                                                 step="0.01"/>
                        </li>

                        <li class="col-span-2">
                            <x-form.input-number label="Кількість"
id="AdminCatalogItemsCreateFormAmount" name="amount"
                                                 value="{{ $data->amount }}"/>
                        </li>

                        <li class="col-span-full">
```

```
                            <x-form.input-textarea label="Опис"
id="AdminCatalogItemsCreateFormDesc" name="desc"
                                                   value="{{ $data->desc }}"/>
                        </li>
                    </ul>
                </div>

                <div></div>

                <div>
                    <h3 class="mb-3 text-lg font-semibold">Атрибути</h3>
                    <ul class="grid grid-cols-2 gap-4">
                        @foreach($filters['config']['attributes'] as $attribute)
                            <li>{{ $attribute->name }}</li>
                            <li>
                                <input type="hidden" name="attributes[{{
$attribute->id }}][attribute_id]"
                                       value="{{ $attribute->id }}"
                                />
                                <x-form.input-text name="attributes[{{ $attribute-
>id }}][value]"
                                                   value="{{
$filters['config']['current_attributes'][$attribute->id] ?? '' }}"
                                                   :required="false"
                                />
                            </li>
                        @endforeach
                    </ul>
                </div>

            </div>
            <div class="mt-4 flex justify-between">
                <button type="button" class="btn-lg btn-red"
                        onclick="event.preventDefault(); if(confirm('Видалити
товар?')) { getElementById('AdminCatalogItemDeleteForm').submit() }"
                >
                    <i class="mdi mdi-trash-can-outline"></i> Видалити
                </button>
                <button type="submit" class="btn-lg btn-primary">Зберегти</button>
            </div>
        </form>

        <form id="AdminCatalogItemDeleteForm" action="{{
route('admin.catalog.items.destroy', $data->id) }}"
              method="POST">
            @csrf
            @method('delete')
        </form>
    </div>
@endsection
```

Додаток Д

Сервіси для роботи з файлами та обробки зображень

FileService.php

```
class FileService {


    public static function getInstance(): Filesystem
    {
        return Storage::disk('public');
    }

    public static function save(UploadedFile $file, $path): string
    {
        $filename = Uuid::uuid4() . '.' . $file->getClientOriginalExtension();

        $file->storeAs($path, $filename, 'public');

        return $filename;
    }

    public static function copy($from, $to): string
    {
        $fs = self::getInstance();
        return $fs->copy($from, $to);
    }

    public static function delete($path): bool
    {
        $fs = self::getInstance();

        if(!$fs->exists($path)){
            return false;
        }

        return $fs->delete($path);
    }

}
```

ImageService.php

```
class ImageService
{

    const DEFAULT_PATH = 'app/public/';

    public static function getImageInstance($path)
    {
        return (new ImageResize(self::getPath($path)));
    }

    public static function getPath($path): string
    {
        return storage_path(self::DEFAULT_PATH . $path);
    }

    public static function resizeX($src, $size)
```

```
    {
        self::getImageInstance($src)->resizeToWidth($size, true)-
>save(self::getPath($src), IMAGETYPE_JPEG);
    }

    public static function resizeY($src, $size)
    {
        self::getImageInstance($src)->resizeToHeight($size, true)-
>save(self::getPath($src), IMAGETYPE_JPEG);
    }

    public static function resize($src, $x, $y)
    {
        self::getImageInstance($src)->resize($x, $y, true)-
>save(self::getPath($src), IMAGETYPE_JPEG);
    }

    public static function crop($src, $w, $h, $x, $y)
    {
        self::getImageInstance($src)->freecrop($w, $h, $x, $y)-
>save(self::getPath($src), IMAGETYPE_JPEG);
    }

    public static function cropCenter($src, $w, $h)
    {
        self::getImageInstance($src)->crop($w, $h)->save(self::getPath($src),
IMAGETYPE_JPEG);
    }

    public static function cropCenter_old($src, $w, $h)
    {
        $image = self::getImageInstance($src);
        if ($image->getSourceWidth() >= $w && $image->getSourceHeight() >= $h) {
            $image->freecrop(
                $w,
                $h,
                intval(($image->getSourceWidth() - $w) / 2),
                intval(($image->getSourceHeight() - $h) / 2),
            )->save(self::getPath($src), IMAGETYPE_JPEG);
        }
    }

}
```

Додаток Е

OrderCreate.vue

```
<template>
    <div id="AdminOrdersCreateForm">
        <div class="grid grid-cols-2 gap-6 py-4">
            <div>
                <h3 class="text-lg font-semibold">Товари</h3>
                <ul class="flex flex-col gap-2">
                    <li v-for="(item, i) in data.items" :key="i" class="grid grid-
cols-8 items-center gap-2">
                        <div class="p-1">{{ i + 1 }}</div>

                        <select @change="selectItem(i, item.item_id)" v-
model.number="item.item_id"
                                class="col-span-2 input-normal input-sm">
                            <option value="" selected disabled>- вибрати -
</option>
                            <option v-for="_item in items" :key="_item.id"
:value="_item.id">{{ _item.name }}
                                {{ _item.model }}
                            </option>
                        </select>

                        <div class="col-span-2 flex flex-row justify-between
items-center gap-2">
                            <div v-if="!isMinAmountReached(i)"
@click="decrementItem(i)" class="p-1"><i
                                class="mdi mdi-minus"></i></div>
                            <div v-else class="p-1 text-gray-400"><i class="mdi
mdi-minus"></i></div>

                            <input v-model.number="item.amount"
@input="[handleItemAmount(i), recountItem(i)]"
                                   type="number"
                                   class="input-normal input-sm appearance-none"/>

                            <div v-if="!isMaxAmountReached(i)"
@click="incrementItem(i)" class="p-1"><i
                                class="mdi mdi-plus"></i></div>
                            <div v-else class="p-1 text-gray-400"><i class="mdi
mdi-plus"></i></div>
                        </div>

                        <input v-model.number="item.item_price" type="text"
readonly/>

                        <input v-model.number="item.total_price" type="text"
readonly/>

                        <div @click="deleteItem(i)" class="p-1 text-center text-
red-400">
                            <i class="mdi mdi-trash-can-outline"></i>
                        </div>
                    </li>
                </ul>
                <div @click="addItem()"
                     class="py-2 mt-4 border-dashed border-2 border-gray-300
rounded-md text-center cursor-pointer">
                    +
                </div>
```

```
            </div>

            <ul class="grid grid-cols-2 gap-4">
                <li class="col-span-full">
                    <div class="flex flex-col gap-2">
                        <label for="AdminOrdersCreateUser">Користувач</label>
                        <select v-model="data.user_id" @change="handleUserChange"
class="input-normal input-md">
                            <option value="" selected>- вибрати -</option>
                            <option v-for="user in users" :key="user.id"
:value="user.id">{{ user.name }}</option>
                        </select>
                    </div>
                </li>

                <li v-if="!data.user_id">
                    <div class="flex flex-col gap-2">
                        <label for="AdminOrdersCreateUserName">Ім'я</label>
                        <input v-model="data.user_name"
id="AdminOrdersCreateUserName" type="text"
                               class="input-normal input-md"/>
                    </div>
                </li>

                <li v-if="!data.user_id">
                    <div class="flex flex-col gap-2">
                        <label for="AdminOrdersCreateUserPhone">Телефон</label>
                        <input v-maska data-maska="+38 (###) ### ## ##"
                               v-model="data.user_phone"
id="AdminOrdersCreateUserEmail" type="text"
                               class="input-normal input-md"
                        />
                    </div>
                </li>

                <li v-if="!data.user_id" class="col-span-full">
                    <div class="flex flex-col gap-2">
                        <label for="AdminOrdersCreateUserEmail">Email</label>
                        <input v-model="data.user_email"
id="AdminOrdersCreateUserEmail" type="email"
                               class="input-normal input-md"/>
                    </div>
                </li>
                <li class="col-span-full">
                    <h3 class="mb-2">Спосіб оплати</h3>
                    <ul class="w-full grid grid-cols-2 gap-2">
                        <li>
                            <input v-model.number="data.payment_method_id"
type="radio"
                                   id="AdminOrderCreatePaymentMethodCard"
name="payment_method_id" value="0"
                                   class="hidden peer" required/>
                            <label for="AdminOrderCreatePaymentMethodCard"
                                   class="inline-flex items-center justify-between
w-full p-3 text-gray-500 bg-white border border-gray-200 rounded-lg cursor-pointer
peer-checked:border-primary peer-checked:text-primary hover:text-primary-light">
                                <div class="block">
                                    <div class="w-full text-lg font-semibold">На
карту</div>
                                </div>
                            </label>
                        </li>
```

```
                        <li>
                            <input v-model.number="data.payment_method_id"
type="radio"
                                   id="AdminOrderCreatePaymentMethodOnReceived"
name="payment_method_id" value="1"
                                   class="hidden peer">
                            <label for="AdminOrderCreatePaymentMethodOnReceived"
                                   class="inline-flex items-center justify-between
w-full p-3 text-gray-500 bg-white border border-gray-200 rounded-lg cursor-pointer
peer-checked:border-primary peer-checked:text-primary hover:text-primary-light">
                                <div class="block">
                                    <div class="w-full text-lg font-semibold">При
отриманні</div>
                                </div>
                            </label>
                        </li>
                    </ul>

                </li>
                <li>
                    <div class="flex flex-col gap-2">
                        <label for="AdminOrdersCreateDeliveryArea">Область</label>
                        <loading-spinner :loading="npLoading.areas"/>
                        <select v-show="!npLoading.areas" v-model="data.area"
@change="setArea($event.target)"
                                id="AdminOrdersCreateDeliveryArea"
                                class="input-normal input-md">
                            <option value="" selected disabled>- вибрати -
</option>
                            <option v-for="area in areas" :key="area['Ref']"
:value="area['Description']"
                                    :data-ref="area['Ref']">{{ area['Description']
}}
                            </option>
                        </select>
                    </div>
                </li>

                <li>
                    <div v-show="selected.areas" class="flex flex-col gap-2">
                        <label for="AdminOrdersCreateDeliveryCity">Місто</label>
                        <loading-spinner :loading="npLoading.cities"/>
                        <select v-show="!npLoading.cities" v-model="data.city"
@change="setCity($event.target)"
                                id="AdminOrdersCreateDeliveryCity"
                                name="user_id"
                                class="input-normal input-md">
                            <option value="" selected disabled>- вибрати -
</option>
                            <option v-for="city in cities" :key="city['Ref']"
:value="city['Description']"
                                    :data-ref="city['Ref']">{{ city['Description']
}}
                            </option>
                        </select>
                    </div>
                </li>

                <li class="col-span-full">
                    <div v-show="selected.cities" class="flex flex-col gap-2">
                        <label
for="AdminOrdersCreateDeliveryWarehouse">Відділення</label>
```

```
                        <loading-spinner :loading="npLoading.warehouses"/>
                        <select v-show="!npLoading.warehouses" v-
model="data.warehouse"
                                id="AdminOrdersCreateDeliveryWarehouse"
name="user_id"
                                class="input-normal input-md">
                            <option value="" selected disabled>- вибрати -
</option>
                            <option v-for="wh in warehouses" :key="wh['Ref']"
:value="wh['Description']">
                                {{ wh['Description'] }}
                            </option>
                        </select>
                    </div>
                </li>

                <li class="col-span-full">
                    <label for="AdminOrdersCreateDesc">Опис</label>
                    <textarea v-model="data.desc" id="AdminOrdersCreateDesc"
                              class="mt-2 input-normal input-md"></textarea>
                </li>

                <li>
                    <div class="flex flex-col gap-2">
                        <label for="AdminOrdersCreateSource">Джерело
замовлення</label>
                        <loading-spinner :loading="sourcesLoading"/>
                        <select v-show="!sourcesLoading" v-model="data.source_id"
                                id="AdminOrdersCreateSource"
                                class="input-normal input-md">
                            <option value="" selected disabled>- вибрати -
</option>
                            <option v-for="(source, key) in sources" :key="key"
:value="source.name">
                                {{ source.name }}
                            </option>
                        </select>
                    </div>
                </li>
            </ul>
        </div>
        <div class="flex justify-between items-center gap-2">
            <span class="font-semibold text-2xl">Сума: {{ total_price.toFixed(2)
}} ₴</span>

            <button v-show="!edit && !orderLoading.order"
@click.stop="createOrder(data)" type="submit"
                    class="btn-lg btn-primary">Створити
            </button>

            <button v-show="edit && !orderLoading.order"
@click.stop="updateOrder(data)" type="submit"
                    class="btn-lg btn-primary">Зберегти
            </button>
        </div>
    </div>
</template>

<script>
import {mapActions, mapState} from "pinia";
import {useUserStore} from "../../stores/UserStore";
import {useItemStore} from "../../stores/ItemStore";
```

```
import {useNovaPoshtaStore} from "../../stores/NovaPoshtaStore";
import LoadingSpinner from "../LoadingSpinner.vue";
import {useOrderStore} from "../../stores/OrderStore";

export default {
    name: 'OrderCreate',
    components: {LoadingSpinner},

    data() {
        return {
            data: {
                user_id: '',
                user_name: '',
                user_email: '',
                user_phone: '',
                payment_method_id: '',
                area: '',
                city: '',
                warehouse: '',
                desc: '',
                source_id: 0,
                items: [],
            },
            selectedItems: [],
            availableItems: [],
            amounts: [],
            total_price: 0,
        }
    },

    computed: {
        ...mapState(useUserStore, {
            users: 'getUsers',
        }),
        ...mapState(useItemStore, {
            items: 'getItems',
        }),
        ...mapState(useNovaPoshtaStore, {
            areas: 'getAreas',
            cities: 'getCities',
            warehouses: 'getWarehouses',
            selected: 'getSelected',
            npLoading: 'getLoading',
        }),
        ...mapState(useOrderStore, {
            order: 'getOrder',
            sources: 'getSources',
            orderLoading: 'orderLoading',
            sourcesLoading: 'sourcesLoading',
        }),

        edit() {
            return parseInt(window.location.href.split('/').pop()) > 0
        },
    },

    methods: {
        ...mapActions(useUserStore, [
            'fetchUsers',
        ]),
        ...mapActions(useItemStore, [
            'fetchItems',
```

```
        ]),
        ...mapActions(useNovaPoshtaStore, [
            'fetchAreas',
            'fetchCities',
            'fetchWarehouses',
        ]),
        ...mapActions(useOrderStore, [
            'fetchOrder',
            'fetchSources',
            'createOrder',
            'updateOrder',
        ]),

        setArea(ref) {
            this.fetchCities(ref.selectedOptions[0].dataset.ref)
        },

        setCity(ref) {
            this.fetchWarehouses(ref.selectedOptions[0].dataset.ref)
        },

        addItem() {
            if (this.data.items.length === 0 ||
this.data.items[this.data.items.length - 1].item_id != null) {
                this.data.items.push({
                    item_id: null,
                    amount: 1,
                    item_price: 0,
                    total_price: 0,
                })

                this.amounts.push(0)
            }
        },

        selectItem(i, item_id) {
            let item = this.items.find(item => item.id === item_id)

            this.amounts[i] = item.amount
            this.data.items[i].item_price = item.price ?? 0

            this.recountItem(i)
        },

        decrementItem(i) {
            this.data.items[i].amount -= 1
            this.recountItem(i)
        },

        incrementItem(i) {
            this.data.items[i].amount += 1
            this.recountItem(i)
        },

        recountItem(i) {
            this.total_price = 0
            this.data.items[i].total_price = (this.data.items[i].amount *
this.data.items[i].item_price).toFixed(2)
            this.data.items.forEach(item => this.total_price +=
parseFloat(item.total_price))
        },
```

```
        deleteItem(i) {
            this.amounts = this.amounts.filter((item, index) => index !== i)
            this.data.items = this.data.items.filter((item, index) => index !== i)

            this.recountItem(i)
        },

        isMinAmountReached(i) {
            return this.data.items[i].amount <= 1
        },

        isMaxAmountReached(i) {
            return this.amounts[i] <= this.data.items[i].amount
        },

        handleItemAmount(i) {
            if (this.isMinAmountReached(i)) this.data.items[i].amount = 1
            if (this.isMaxAmountReached(i)) this.data.items[i].amount =
this.amounts[i]
        },

        handleUserChange() {
            if (this.data.user_id) {
                this.data.user_name = '';
                this.data.user_email = '';
                this.data.user_phone = '';
            }
        },

    },

    watch: {
        order(newOrder) {
            if (this.edit) {
                this.data = newOrder
            }
        }
    },

    async created() {
        this.fetchUsers()
        this.fetchItems()
        this.fetchAreas()
        this.fetchSources()
        if (this.edit) await
this.fetchOrder(parseInt(window.location.href.split('/').pop()))
    }

}
</script>
```