

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

**Пояснювальна записка**

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

ІНТЕРНЕТ МАГАЗИН З ПРОДАЖУ МЕБЛІВ

SOFTWARE IMPLEMENTATION OF A WEB APPLICATION FOR

AN ONLINE FURNITURE STORE

Виконав(ла): студент(ка) 4 курсу, групи КНТ-139

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

КОРОСТИЛЬОВ Д.Ю.

(ПРИЗВИЩЕ та ініціали)

Керівник ГЛАДКОВА О.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент ЗЕЛЕНЬОВА І.Я.

(ПРИЗВИЩЕ та ініціали)

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**  
**Національний університет «Запорізька політехніка»**  
 (повне найменування закладу вищої освіти)

Факультет ФКНТ  
 Кафедра програмних засобів  
 Ступінь вищої освіти бакалавр  
 Спеціальність 121 Інженерія програмного забезпечення  
(код і найменування)  
 Освітня програма (спеціалізація) Інженерія програмного забезпечення  
(назва освітньої програми (спеціалізації))

**ЗАТВЕРДЖУЮ**

Завідувач кафедри ПЗ, д.т.н, проф.  
Сергій СУББОТІН  
 “      ”                      2023 року

**З А В Д А Н Н Я**

**НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)**

КОРОСТИЛЬОВА Дмитра Юрійовича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація вебзастосунку інтернет магазин з продажу меблів. Software Implementation of a Web Application for an Online Furniture Store

керівник проєкту (роботи) к.т.н., доцент, ГЛАДКОВА Ольга Миколаївна,  
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “04” квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 29 травня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області та огляд аналогів. 2. Вибір інструментарію розробки веб-застосунку. 3. Архітектура веб-застосунку. 4. Графічна частина веб-застосунку. 5. Підключення стороннього API, тестування системи.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

## 6. Консультанти розділів проєкту (роботи)

| Розділ              | ПРІЗВИЩЕ, ініціали та посада консультанта | Підпис, дата   |                           |
|---------------------|-------------------------------------------|----------------|---------------------------|
|                     |                                           | завдання видав | прийняв виконане завдання |
| 1-4 Основна частина | ГЛАДКОВА О.М., доцент                     |                |                           |
| Нормоконтроль       | ДЕЙНЕГА Л.Ю., ст.викладач                 |                |                           |

7. Дата видачі завдання « 04 » квітня 2023 року.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів дипломного проєкту (роботи)                             | Строк виконання етапів проєкту (роботи) | Примітка     |
|-------|----------------------------------------------------------------------|-----------------------------------------|--------------|
| 1     | Постановка завдання роботи.                                          | 1 тиждень                               | Завдання, ТЗ |
| 2     | Аналіз предметної області.                                           | 1 тиждень                               | Розділ 1     |
| 3     | Вибір мови програмування та інших технологій розробки.               | 2 тиждень                               | Розділ 2     |
| 4     | Розробка архітектури програми.                                       | 2 тиждень                               | Розділ 3     |
| 5     | Розробка програми.                                                   | 3-4 тижні                               | Розділ 3, 4  |
| 6     | Тестування та експериментальне дослідження програмного забезпечення. | 5 тиждень                               | Розділ 4     |
| 7     | Оформлення пояснювальної записки та документів до неї.               | 6 тиждень                               | Додатки      |
| 8     | Нормоконтроль та рецензування.                                       | 7 тиждень                               |              |
| 9     | Захист роботи.                                                       | 8 тиждень                               |              |

Студент

  
( підпис )

Дмитро КОРОСТИЛЬОВ  
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

\_\_\_\_\_  
( підпис )

Ольга ГЛАДКОВА  
(Ім'я ПРІЗВИЩЕ)

## РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:  
116 с., 45 рис., 2 табл., 2 дод., 17 джерел.

КОМП'ЮТЕРНИЙ ВЕБЗАСТОСУНОК, БАЗА ДАНИХ, МЕБЛІ  
РЕАЛІЗАЦІЯ ПРОДУКТУ, БАЗА ДАНИХ.

Об'єкт дослідження – програмні засоби для розробки вебзастосунку для продажу меблів

Предмет дослідження – сайт продажу меблів, який дозволяє користувачам вибирати та замовляти меблів з онлайн-каталогу

Мета роботи – розробити ефективну веб-програму для продажу меблів, яка полегшить і залучить споживачів і досягне успіху в ринковій конкуренції.

Для вирішення поставленої мети необхідно вирішити наступні завдання:

- провести аналіз потреб користувачів, щоб визначити функціонал, необхідний для програми;
- розробити зручний інтерфейс користувача, який дозволить переглядати каталог меблів, вибирати та замовляти товари;
- створити систему керування замовленнями та оплатою.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, веб-фреймворк Laravel, мови програмування PHP, JQuery та CSS, персональний комп'ютер під управлінням операційної системи Microsoft Windows 10.

Результати. Вебзастосунок, розроблений на Laravel, яка дозволяє користувачам створювати аккаунти, продивлятися онлайн-каталоги, додавати товари в кошик та здійснювати платежі.

Висновки. Розроблено вебзастосунок для онлайн продажу меблів.

Галузь використання – меблеві компанії, меблеві магазини, дизайнери інтер'єрів та інші учасники ринку, які хочуть продавати свою продукцію онлайн.

## ABSTRACT

Explanatory note to the diploma qualifying work of the master: 116 pages, 2 tables, 45 figures, 2 appendixes, 17 sources.

COMPUTER WEB APPLICATION, DATABASE, FURNITURE SALES, DATABASE.

Object of research – software tools for developing a web application for selling furniture.

Subject of research – a furniture sales website that allows users to select and order furniture from an online catalog.

Purpose – to develop an effective web application for furniture sales that will facilitate and attract consumers and succeed in market competition.

To achieve this goal, the following tasks need to be solved:

- analyze user needs to determine the functionality required for the program;
- develop a user-friendly interface that will allow users to browse the furniture catalog, select and order products;
- create a system for managing orders and payments.

Materials, methods and techniques: structural and object-oriented programming, Laravel web framework, PHP, JQuery and CSS programming languages, a personal computer running the Microsoft Windows 10 operating system.

Result. A web application developer in Laravel that allows users to create accounts, browse online catalogs, add products to the cart and make payments.

Conclusions: A web application for online furniture sales has been developed.

The field of use is furniture companies, furniture stores, interior designers and other market participants who want to sell their products online.

## ЗМІСТ

|                                                          |    |
|----------------------------------------------------------|----|
| Перелік скороень та умовних познач                       | 9  |
| Вступ                                                    | 10 |
| 1 Аналіз предметної області та огляд аналогів            | 11 |
| 1.1 Опис об'єкта дослідження                             | 11 |
| 1.2 Огляд популярних онлайн магазинів для продажу меблів | 11 |
| 1.3 Постановка завдань роботи                            | 11 |
| 1.3.1 Аналіз потреб ринку та конкретної середи           | 11 |
| 1.3.2 Розробка архітектури веб-застосунку                | 12 |
| 1.3.3 Розробка дизайну та інтерфейсу користувача         | 12 |
| 1.3.4 Розробка програмного забезпечення                  | 13 |
| 1.3.5 Тестування та впровадження                         | 13 |
| 1.4 Висновки за розділом                                 | 14 |
| 2 Вибір інструментарію розробки вебзастосунку            | 15 |
| 2.1 Вибір середовища розробки                            | 15 |
| 2.1.1 Wordpress                                          | 15 |
| 2.1.2 Wix                                                | 15 |
| 2.1.3 Joomla                                             | 16 |
| 2.1.4 HTML, CSS та JavaScript                            | 16 |
| 2.1.5 Laravel                                            | 17 |
| 2.2 Вибір бази даних                                     | 18 |
| 2.3 Вибір системи для платежів                           | 19 |
| 2.4 Вибір ПЗ для розробки вебзастосунку                  | 20 |
| 2.4.1 Github                                             | 20 |
| 2.4.2 PHPStorm                                           | 21 |
| 2.4.3 Docker                                             | 21 |
| 2.4.2 Ubuntu                                             | 21 |
| 2.5 Висновки за розділом 2                               | 21 |
| 3 Архітектура вебзастосунку                              | 23 |

|                                                           |    |
|-----------------------------------------------------------|----|
| 3.1 Основні вимоги до проєкту .....                       | 23 |
| 3.1.1 Основні вимоги до користувацької частини.....       | 23 |
| 3.1.2 Основні вимоги до адмінської частини.....           | 24 |
| 3.2 Архітектура системи .....                             | 25 |
| 3.2.1 Контролери.....                                     | 25 |
| 3.2.2 Middleware .....                                    | 27 |
| 3.2.3 Валідація даних.....                                | 27 |
| 3.2.4 MailGun.....                                        | 28 |
| 3.2.5 Моделі.....                                         | 28 |
| 3.2.6 Міграції.....                                       | 29 |
| 3.2.7 Збереження файлів .....                             | 29 |
| 3.2.8 Views .....                                         | 30 |
| 3.2.9 Роути .....                                         | 31 |
| 3.3 Архітектура бази даних.....                           | 32 |
| 3.3.1 Таблиця користувача і адміна .....                  | 33 |
| 3.3.2 Таблиця брендів та категорій.....                   | 34 |
| 3.3.3 Таблиця продуктів.....                              | 34 |
| 3.3.4 Таблиця кошику та обраних .....                     | 36 |
| 3.3.5 Таблиці транзакції .....                            | 37 |
| 3.3.6 Таблиці коментарів.....                             | 38 |
| 3.3.7 Загальна структура таблиць .....                    | 39 |
| 3.4 Послідовність дій користувача .....                   | 40 |
| 3.4.1 Користувач магазину.....                            | 40 |
| 3.4.1 Адмін-панель .....                                  | 41 |
| 4. Графічна частина вебзастосунку.....                    | 43 |
| 4.1 Основні вимоги і тренди .....                         | 43 |
| 4.2 Графічний дизайн для користувачів.....                | 44 |
| 4.2.1 Навігаційна панель, реєстрація та авторизація ..... | 44 |
| 4.2.2 Головна сторінка.....                               | 46 |
| 4.2.3 Каталог продуктів.....                              | 48 |

|                                                         |     |
|---------------------------------------------------------|-----|
| 4.2.4 Каталог продуктів.....                            | 50  |
| 4.2.5 Обрані .....                                      | 51  |
| 4.2.6 Кошик .....                                       | 52  |
| 4.2.7 Аккаунт користувача.....                          | 54  |
| 4.3 Графічний дизайн для адміністратора.....            | 56  |
| 4.3.1 Навігаційна панель .....                          | 56  |
| 4.3.2 Сторінка статистики.....                          | 57  |
| 4.3.3 Сторінка користувачів .....                       | 58  |
| 4.3.4 Сторінка товарів .....                            | 60  |
| 4.3.5 Сторінка категорії.....                           | 62  |
| 4.3.6 Сторінка брендів.....                             | 63  |
| 4.3.7 Сторінка транзакції .....                         | 65  |
| 4.4 Висновок.....                                       | 66  |
| 5. Підключення стороннього API, тестування системи..... | 68  |
| 5.1 Підключення API Stripe .....                        | 68  |
| 5.2 Підключення API Mailgun .....                       | 68  |
| 5.3 Тестування системи.....                             | 70  |
| 5.3.1 Функціональні тестування.....                     | 70  |
| 5.3.2 Тестування безпеки .....                          | 71  |
| 5.3.3 Тестування сумісності.....                        | 72  |
| 5.3.4 Відлагодження системи .....                       | 72  |
| 5.4 Висновок.....                                       | 73  |
| Висновки.....                                           | 74  |
| Перелік джерел посилання .....                          | 75  |
| Додаток А Текст програми .....                          | 77  |
| Додаток Б Слайди презентації.....                       | 108 |

## ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

|        |                                            |
|--------|--------------------------------------------|
| API    | – Application Programming Interface;       |
| ПЗ     | – програмне забезпечення;                  |
| БД     | – база даних                               |
| jQuery | – бібліотека                               |
| CSS    | – каскадні таблицю стилю                   |
| PHP    | – скриптова мова програмування             |
| HTML   | – мова розмітки гіпертексту                |
| Stripe | – API для онлайн-платежів                  |
| СКБД   | – система керування базами даних           |
| MySQL  | – реляційна система керування базами даних |
| CMS    | – система керування вмістом                |
| ОС     | – операційна система                       |
| IDE    | – інтегроване середовище розробки          |

## ВСТУП

Сьогодні покупці все частіше роблять покупки в Інтернеті, і це стосується не тільки товарів, а й меблів. У зв'язку з цим створення зручного та простого у використанні веб-додатку для продажу меблів стало ключовою роботою підприємств меблевої промисловості. Такий застосунок дасть змогу клієнтам швидко і легко вибрати і замовити необхідні меблі у зручний для них час і місце.

При розробці такого додатку особливу увагу необхідно приділити зручності та простоті інтерфейсу. Він має бути інтуїтивно зрозумілим і простим у використанні навіть для тих, хто не має досвіду купівлі меблів онлайн. Крім того, веб-додаток має надавати повну інформацію про товар, включаючи фотографії, опис і ціну, щоб покупець міг зробити усвідомлений вибір.

## **1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД АНАЛОГІВ**

### **1.1 Опис об'єкта дослідження**

Предметом дослідження є процес розроблення та впровадження веб-додатку для продажу меблів на Laravel, зокрема вивчення технологій та інструментів, необхідних для розроблення веб-додатку, вивчення потреб і вимог цільової аудиторії, планування функціоналу та дизайну, розроблення серверної та клієнтської частин, тестування та впровадження веб-додатку. Важливо враховувати також аспекти безпеки та захисту персональних даних користувачів, а також оптимізацію роботи веб-додатка для підвищення ефективності та зручності використання для користувачів.

### **1.2 Огляд популярних онлайн магазинів для продажу меблів**

Існує безліч онлайн-магазинів для продажу меблів, і кожен з них пропонує свої унікальні переваги та можливості. Нижче (табл.1.1) подано огляд деяких популярних онлайн-магазинів для продажу меблів із прикладами та посиланнями на магазини.

### **1.3 Постановка завдань роботи**

Для розробки веб-застосунків для продажу меблів на Laravel, необхідно вирішити наступні завдання:

#### **1.3.1 Аналіз потреб ринку та конкретної середовища**

Необхідно провести дослідження ринку для виявлення потенційних клієнтів, їх потреб і вимог до інтернет-магазину, що продає меблі. Також потрібно буде дослідити конкурентне середовище, щоб визначити їхні сильні та

слабкі сторони, щоб можна було взяти їх до уваги під час розробки власних вебзастосунків (тпбл.1.1).

Таблиця 1.1 – Порівняння існуючих аналогів

| Сайт             | Особливості                                                                                                       | Недоліки                                                                                                                                      |
|------------------|-------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| ІКЕА             | - Добре структурований та ефективний веб-сайт.<br>-Функціональність для налаштування меблів.                      | - Проблеми з навігацією та повільне завантаження сторінок                                                                                     |
| WayFair          | - Велика кількість інформації про кожний продукт<br>-Відгуки попередніх покупців<br>-Детальні описи та фотографії | - Складна навігація та велика кількість розділів<br>- Повільне завантаження сторінок<br>- Нерівномірний дизайн та нечітко визначена структура |
| Ashley Furniture | - Зручний та легкий пошук за категоріями, кольором, стилем та матеріалом                                          | - Відсутність інструментів для створення власних меблевих комплектів                                                                          |

### 1.3.2 Розробка архітектури веб-застосунку

Необхідно провести дослідження ринку для виявлення потенційних клієнтів, їх потреб і вимог до інтернет-магазину, що продає меблі. Також потрібно буде дослідити конкурентне середовище, щоб визначити їхні сильні та слабкі сторони, щоб врахувати ці фактори при розробці власного веб-застосунку.

### 1.3.3 Розробка дизайну та інтерфейсу користувача

Важливо зазначити, що дизайн веб-застосунку має велике значення для привабливості та зручності користування ним. Логотип та кольорова гама

повинні бути відповідними концепції та цілям веб-додатку, а також привабливими для потенційних клієнтів.

Розробка інтерфейсу користувача повинна бути зосереджена на забезпеченні максимальної зручності та простоти використання. Користувачі повинні бути здатні з легкістю здійснювати покупки та знаходити необхідну інформацію. Важливо ретельно продумати логіку взаємодії користувача з веб-додатком, розміщення елементів на сторінках та їхній дизайн.

Крім того, під час розробки дизайну та інтерфейсу користувача необхідно враховувати особливості різних пристроїв та їхніх розмірів екранів. Веб-додаток повинен бути пристосований для використання на різних пристроях, включаючи комп'ютери, планшети та смартфони.

#### **1.3.4 Розробка програмного забезпечення**

Необхідно провести аналіз вимог до програмного забезпечення та розробити детальний план реалізації функцій керування контентом, що включає створення, редагування та видалення продуктів та категорій. Для реєстрації користувачів потрібно розробити форму реєстрації та входу в систему, а також функцію відновлення пароллю. Для відстеження замовлень та оплати необхідно розробити функцію додавання товару в кошик, розрахунку вартості замовлення. Для забезпечення безпеки веб-застосунку необхідно розробити систему авторизації користувачів, валідацію даних введених користувачем. Крім того, необхідно розробити систему логування для відстеження помилок та проблем у веб-застосунку.

#### **1.3.5 Тестування та впровадження**

Після успішного розроблення програмного забезпечення необхідно провести тестування веб-застосунку з метою виявлення та виправлення помилок,

що можуть виникнути під час використання. Важливо забезпечити якість та надійність веб-додатку перед впровадженням в експлуатацію.

Після успішного тестування необхідно впровадити застосунок та підготувати його до використання клієнтами. Також важливо забезпечити підтримку та зв'язок з користувачами після впровадження веб-додатку для вирішення можливих технічних проблем та надання необхідної допомоги.

#### **1.4 Висновки за розділом**

Веб-додаток для продажу меблів на Laravel - це вагомий крок у розвитку сучасного бізнесу, який дає змогу розширити географію продажів і збільшити кількість покупців. Однак, розробка та впровадження такого застосунку потребує ретельного планування та виконання безлічі завдань.

Таким чином, щоб наш сайт був привабливим і зручним для користувачів, необхідно приділити увагу навігації та структурі сайту, надати інструменти для персоналізації, зменшити час завантаження сторінок і забезпечити детальну інформацію про продукти з відгуками покупців.

Розробка веб-додатку вимагає врахування потреб цільової аудиторії, вибору сервера, бази даних, бекенда і фронтенда, а також забезпечення безпеки. Крім того, важливо надавати якісний контент і керувати ним, а після тестування додаток має бути готовий до використання клієнтами.

Загалом, розробка веб-додатку для продажу меблів - це складний і багатоетапний процес, який вимагає безлічі знань і навичок. Однак, за правильного підходу, такий застосунок може стати незамінним інструментом для просування бізнесу і збільшення обсягів продажів.

## **2 ВИБІР ІНСТРУМЕНТАРІЮ РОЗРОБКИ ВЕБЗАСТОСУНКУ**

### **2.1 Вибір середовища розробки**

Створення веб-застосунку – це важливий процес, що вимагає вибору правильного інструментарію. Існує безліч інструментів для розробки сайтів, кожен з яких має свої переваги та недоліки, що може зробити вибір складним завданням. Розглянемо детальніше деякі з них.

#### **2.1.1 Wordpress**

Wordpress [1] — це CMS [2], яка дозволяє створювати веб-сайти будь-якого рівня складності, від блогів до інтернет-магазинів. Завдяки широкому вибору тем і плагінів користувачі можуть налаштувати вигляд і функціональність веб-сайту відповідно до своїх вимог і цілей. Крім того, у Wordpress є велика спільнота розробників, які постійно працюють над вдосконаленням і розвитком цієї CMS. Це забезпечує підтримку та оновлення, щоб уникнути порушень безпеки та збоїв у роботі сайту.

Однак використання WordPress також має свій мінуси. Наприклад використання великої кількості плагінів може призвести до низької продуктивності веб-застосунку. Крім того, Wordpress не завжди легко налаштувати для створення унікального дизайну веб-сайту. Він може бути обмежений попередньо встановленими темами та шаблонами, що ускладнює налаштування та створення унікального сайту. Це може бути особливо проблематично для сайтів, які вимагають високої унікальності та індивідуальності, як сайти для брендових компаній або професійних портфоліо.

#### **2.1.2 Wix**

Wix [3] — це онлайн-конструктор веб-сайтів [4], який надає простий у використанні інтерфейс для користувачів без досвіду веб-розробки. Wix також

пропонує різноманітні готові шаблони та інструменти для налаштування зовнішнього вигляду та функціональності вашого веб-сайту. Це дозволяє користувачам швидко створювати сайти та економити час.

Однак Wix також має свої недоліки. Обмежений рівень гнучкості є одним з головних недоліків. На відміну від ручного налаштування сайту за допомогою мови програмування, користувачі мають обмежений рівень контролю над своїм сайтом. Це може означати, що досвідченіші користувачі не зможуть налаштовувати свої сайти настільки, наскільки вони хочуть. Ще одним недоліком Wix є неможливість перенести сайт на іншу платформу. Якщо пізніше користувач вирішить змінити свої потреби та захоче перенести свій сайт на іншу платформу, він не зможе цього зробити.

### **2.1.3 Joomla**

Joomla [5] - це CMS, яка має як переваги, так і недоліки. Однією з головних переваг Joomla є широкий вибір тем і розширень, які дозволяють створювати веб-сайти з різними функціями та дизайном. Joomla має простий інтерфейс, що робить його доступним для користувачів без досвіду розробки.

Однак є свої недоліки. Наприклад, Joomla не має стандартних інструментів SEO, що може призвести до обмеження можливостей SEO. Крім того, Joomla має обмежені можливості інтеграції з іншими платформами та службами, що може стати проблемою для користувачів, які хочуть використовувати інструменти сторонніх розробників для свого веб-сайту.

Загалом Joomla є хорошим вибором для створення сайтів зі складною структурою та функціями, а також мають певний досвід у веб-розробці.

### **2.1.4 HTML, CSS та JavaScript**

HTML, CSS і JavaScript — це основні мови веб-розробки, які дозволяють створювати веб-сайти з нуля.

Однією з переваг його використання є повний контроль над його зовнішнім виглядом, функціональністю та поведінкою. Можливість створення унікальних та індивідуальних рішень для кожного проекту. Немає обмежень щодо вибору функцій, ви можете створити будь-яку необхідну функцію, не обмежуючись функціями, які надаються іншими інструментами. Проте є й мінуси. Створення веб-сайту за допомогою HTML, CSS і JavaScript вимагає більше часу та зусиль, ніж використання стандартних інструментів, таких як CMS або онлайн-платформи. Це пояснюється тим, що створення сайту з нуля вимагає ручного кодування, налаштування його зовнішнього вигляду та функціональності, а також налагодження та тестування. Цей процес може бути тривалим і вимогливим до технічного досвіду розробника.

### **2.1.5 Laravel**

Laravel — популярний фреймворк [6] PHP, який забезпечує гнучку та зручну структуру для розробки веб-додатків. Laravel пропонує багато переваг, зокрема:

**Висока продуктивність:** Laravel використовує багато оптимізацій для забезпечення високої продуктивності веб-додатків, що робить його найкращим вибором для розробки сайтів із високим навантаженням.

**Безпека:** Laravel надає багато інструментів для захисту веб-додатків, включаючи системи автентифікації та авторизації, захист від атак CSRF тощо.

**Гнучкість:** Laravel надає гнучку структуру, яка дозволяє розробникам створювати веб-додатки різної складності та функціональності. Він також надає багато розширень та інструментів для швидкої розробки та розширення функціональності вашого веб-сайту.

**Велика спільнота:** Laravel має велику спільноту розробників і користувачів, які надають обширну документацію, навчальні посібники, розширення та плагіни для нього.

Основним недоліком є високий поріг входу. Для використання Laravel необхідні знання PHP і досвід у веб-розробці, що може бути складнішим у використанні, ніж деякі інші інструменти [7].

Розглянувши всі варіанти для розробки вебзастосунка (табл.1.2), було обрано фреймворк Laravel, оскільки він цілком відповідає вимогам до проекту, таким як гнучкість структури, висока продуктивність, безпека і наявність безлічі розширень, що дає змогу виконати всі необхідні завдання для реалізації проекту [8].

Таблиця 2.1 – Порівняння середовище розробки

| Веб-інструмент | Переваги                                    | Недоліки                                     |
|----------------|---------------------------------------------|----------------------------------------------|
| Wordpress      | Широкий вибір тем і плагінів                | Низька продуктивність та обмежені можливості |
| Wix            | Простий та багато шаблонів                  | Обмежена гнучкість                           |
| Joomla         | Широкий вибір тем                           | Обмежені можливості інтеграції               |
| HTML, CSS, JS  | Унікальні рішення та повний контроль        | Багато часу на розробку                      |
| Laravel        | Висока продуктивність, безпека та гнучкість | Високий поріг та потребує знання PHP, JS     |

## 2.2 Вибір бази даних

Коли справа доходить до розробки веб-сайту, вибір правильної бази даних є вирішальним рішенням. Кожна база даних має свої переваги та недоліки, які необхідно враховувати перед тим, як зробити вибір. У той час як деякі бази даних пропонують чудову продуктивність і зручність для користувача, інші забезпечують розширені можливості обробки даних і розширені функції безпеки. Ретельна оцінка різних баз даних і ретельний аналіз їхніх сильних і слабких сторін може допомогти нам прийняти обґрунтоване рішення та вибрати базу даних, яка найкраще підходить для нашої системи.

MySQL — це база даних, яка широко використовується у веб-розробці, завдяки своїй високій продуктивності та масштабованості. Він пропонує широкий набір функціональних можливостей. Тим не менш, робота з величезними обсягами даних за допомогою MySQL може бути незручною, і вона має обмежені можливості для реалізації складних структур даних.

PostgreSQL — це база даних, орієнтована на продуктивність і масштабованість, яка працює на моделі з відкритим кодом. Оснащений набором можливостей обробки даних, він забезпечує оптимальну обробку даних. Недоліком є складніше налаштування та управління, ніж у MySQL, а також високі вимоги до ресурсів.

MongoDB - це база даних NoSQL, яка використовує документи BSON для зберігання інформації. Завдяки високій масштабованості та гнучкості його все частіше використовують для управління великими обсягами неструктурованих даних. Однак це може бути не так ефективно, коли йдеться про строго структуровані дані.

Після ретельного розгляду ми вибрали MySQL як нашу систему управління базами даних через її виняткову продуктивність, масштабованість і універсальну функціональність, яка ідеально відповідає вимогам нашого проекту. Крім того, популярність і широке поширення MySQL забезпечують достатню систему підтримки та навчальні ресурси, що робить її більш життєздатним вибором [9].

## **2.3 Вибір системи для платежів**

Вибір правильної платіжної системи має вирішальне значення при інтеграції її з веб-сайтом Laravel для обробки платежів. Як онлайн-продавець, важливо оцінити різні платіжні системи та визначити їхні відповідні переваги й обмеження. Це допоможе вибрати оптимальну платіжну систему, що відповідає конкретним вимогам проекту.

Stripe [10] - це широко використовувана платіжна система, яка пропонує безпечне та надійне приймання онлайн-платежів і просту інтеграцію з веб-сайтом через свій API. Суворе дотримання платформою стандартів PCI DSS забезпечує безпеку транзакцій та інформації. Проте важливо переконатися, що висока експлуатаційна мінливість системи може задовольнити конкретні потреби бізнесу.

PayPal [11] - ще одна популярна платіжна система, яка досягла значних успіхів у цій галузі. Він може похвалитися надійним API, першокласною безпекою та відповідністю стандартам PCI DSS. Однак одним недоліком є комісія, що стягується за транзакції, і потенційна складність її створення та інтеграції в систему.

Третім варіантом може бути Braintree [12], що належить PayPal. Він також надає легкий і простий для розуміння API, який можна легко інтегрувати у веб-сайти. Крім того, Braintree забезпечує високий рівень безпеки та відповідає стандарту PCI DSS.

З усіх платіжних систем я б вибрав Stripe як найбільш підходящу для нашого проекту. Stripe надає простий для розуміння API, який є дуже безпечним і сумісним з PCI DSS. Крім того, він широко використовується в Інтернеті та має досить низькі комісії за транзакції.

## **2.4 Вибір ПЗ для розробки вебзастосунку**

Оптимальний вибір параметрів середовища для розробки веб-додатків може значно підвищити продуктивність програміста і якість кінцевого продукту.

### **2.4.1 Github**

Для полегшення управління кодом і контролю версій я використовую платформу GitHub. Це дає мені змогу зберігати код проекту в хмарі та за потреби виконувати відкат до попередньої версії. GitHub надає інструменти для

створення нових гілок, їхнього злиття та управління конфліктами. Це дає мені змогу максимально ефективно працювати над кодом і контролювати його стан.

### **2.4.2 PHPStorm**

Серед усіх доступних варіантів IDE я вибрав PHPStorm, тому що він надає безліч інструментів і функцій, які можуть спростити та прискорити процес розробки, таких як завершення коду, налагодження, інтеграція з Git, підтримка плагінів тощо. Крім того, PHPStorm має зручний користувальницький інтерфейс, який спрощує налаштування та використання різних функцій.

### **2.4.3 Docker**

Я обираю Docker як інструмент контейнеризації та розгортання застосунків, оскільки він дає змогу створювати окремі контейнери для кожного компонента застосунку та спрощує їхнє розгортання й керування. Docker також забезпечує масштабованість і надійність додатків, а також скорочення часу розгортання і більш ефективне використання ресурсів сервера.

### **2.4.2 Ubuntu**

Я вибрав Ubuntu як ОС для розробки, тому що це одна з найпопулярніших і найбільш підтримуваних операційних систем у спільноті розробників. Ubuntu пропонує широкий спектр інструментів і програмного забезпечення для розробки, а також безпеку і стабільність додатків [13].

## **2.5 Висновки за розділом 2**

Цю главу присвячено вибору оптимального інструментарію для розробки веб-додатків із використанням. Було розглянуто різні аспекти, які слід

враховувати під час вибору середовища розроблення, а також їх порівняли для оптимального вибору.

Ми вибрали MySQL, Stripe і Laravel як інструменти для розробки веб-додатків, виходячи з їхніх індивідуальних переваг і здатності легко інтегруватися між собою. Наприклад, MySQL пропонує широкий набір функціональних можливостей і високу продуктивність, що забезпечує високу масштабованість програми. Stripe, зі свого боку, забезпечує високий рівень безпеки і легко інтегрується у веб-додаток на Laravel, що робить його хорошим вибором для забезпечення безпеки платежів. Laravel, зі свого боку, пропонує зручне середовище розробки, потужний набір функцій і простоту у використанні, що полегшує процес розроблення веб-додатка. У підсумку, вибір цих інструментів дає змогу створювати якісний і масштабований веб-додаток із високим рівнем безпеки та зручним середовищем розробки.

Також було розглянуто інструментарій, який підвищить продуктивність розробника і якість кінцевого продукту. Важливо враховувати не тільки функціональні можливості інструментарію, а й його зручність використання та відповідність вимогам веб-застосунку.

## **3 АРХІТЕКТУРА ВЕБЗАСТОСУНКУ**

### **3.1 Основні вимоги до проєкту**

Під час розробки веб-додатка було проведено аналітичне дослідження ринку та аналіз наявних магазинів для визначення оптимальних функцій і вимог до кінцевого продукту. Було вивчено потреби та вподобання користувачів, а також проведено аналіз конкурентів. На основі цього аналізу було визначено базові потреби у користувачів та адміністратора.

#### **3.1.1 Основні вимоги до користувацької частини**

Головна сторінка зі списком усіх товарів: Головна сторінка міститиме повний список доступних товарів. Кожен товар буде представлений з коротким описом, зображенням і ціною. Користувачі зможуть переглядати товари, отримувати загальну інформацію про них і переходити на сторінки окремих товарів для отримання більш детальної інформації.

Сортування товарів за категоріями, брендами та ціною: Користувачі зможуть фільтрувати товари за різними параметрами. Вони зможуть обирати певну категорію товарів (наприклад, одяг, взуття, аксесуари), певний бренд або вибрати сортування за ціною або алфавітом. Це спростить процес пошуку і допоможе користувачам швидко знаходити потрібні їм товари.

Кошик та обрані товари: Користувачі зможуть додавати товари в кошик і зберігати товари в розділі "Вибране". Користувачі зможуть переглядати обрані товари, керувати їхньою кількістю та здійснювати покупку. У розділі "Вибране" можна зберігати товари, які їм сподобалися, для подальшого перегляду і покупки.

Оплата за допомогою Stripe: Веб-сайт буде інтегрувати платіжну систему Stripe, щоб користувачі могли безпечно і зручно оплачувати свої покупки. Користувачі зможуть вибрати товари, додати їх у кошик і здійснити оплату через

Stripe, використовуючи кредитні або дебетові картки. Це забезпечить безпеку і зручність процесу оплати для користувачів.

Створення облікового запису та управління особистими даними: Користувачі зможуть створити обліковий запис на веб-сайті, заповнивши реєстраційну форму з необхідною інформацією

Авторизація та вхід в обліковий запис: Після створення облікового запису користувачі зможуть увійти у свій обліковий запис, використовуючи облікові дані, такі як електронна пошта та пароль. Це забезпечить персоналізований досвід для користувачів, дозволяючи їм отримати доступ до своїх замовлень, історії покупок та іншої персональної інформації.

Керування особистими даними в кабінеті користувача: Зареєстровані користувачі зможуть увійти до свого особистого кабінету, де вони зможуть керувати своїми особистими даними. Вони зможуть змінювати інформацію про свій профіль, змінювати пароль, адреси доставки та переглядати інформацію про свої попередні замовлення. Це дасть змогу користувачам легко керувати своїми даними та оновлювати інформацію за потреби.

Сповіщення на пошту при різних подіях, включно з реєстрацією нового користувача, здійсненням покупки товарів і оновленням вмісту кошика. Користувач отримуватиме сповіщення на свою зареєстровану пошту, що допоможе йому отримувати інформацію про замовлення та зміни.

### **3.1.2 Основні вимоги до адмінської частини**

Однією з важливих складових веб-додатка буде зручна панель бічного меню (sidebar), яка міститиме всі доступні функції та можливості для управління і навігації. У цій панелі будуть представлені основні розділи та посилання на різні модулі та функціональності системи.

Адміністративна панель. У веб-сайті на Laravel буде створена адміністративна частина, де можна керувати основними елементами системи.

Переглядати, змінювати, видаляти категорії товарів. Це дасть змогу організувати структуру товарів на веб-сайті, додавати нові категорії та вносити зміни.

Управління товарами: В адміністративній панелі можна переглядати список усіх товарів, доданих на веб-сайт. Змінювати інформацію про товари, таку як назва, опис, ціна і зображення. Також видаляти товари, якщо вони більше не актуальні або їх необхідно видалити з інших причин. Також видалені товари будуть відправлятися в сховище, звідки можуть бути відновлені.

Управління користувачами: В адміністративній панелі переглядати список зареєстрованих користувачів веб-сайту. Змінювати інформацію про користувачів, таку як ім'я, електронна пошта, адреса доставки та інші дані. Також видаляти користувачів, якщо це потрібно відповідно до політики веб-сайту.

Управління транзакціями: В адміністративній панелі можна переглядати список транзакцій, пов'язаних з платежами на веб-сайті. Можна переглянути інформацію про кожну транзакцію, включно з сумою, датою і пов'язаним з нею користувачем. Також можна змінювати і видаляти транзакції за необхідності.

Управління брендами: В адміністративній панелі можна переглядати список брендів, які пов'язані з товарами на веб-сайті. Можна змінювати інформацію про бренди, таку як назва та опис. Також видаляти бренди.

Загальна статистика: В адміністративній панелі ви зможете переглядати загальну статистику вашого веб-сайту, таку як кількість зареєстрованих користувачів, кількість проданих товарів, загальний дохід та інші показники. Це дасть вам змогу оцінити ефективність вашого веб-сайту.

## **3.2 Архітектура системи**

### **3.2.1 Контролери**

Структура коду проекту на Laravel організована за патерном Model-View-Controller (MVC). Контролери (Controllers) відповідають за обробку запитів і керування логікою програми. У проєкті контролери розподілені на дві категорії, для адміністратора та для користувача. Нижче наведено короткий опис кожного.

Контролери для адміністратора обробляють запити, пов'язані з управлінням даними, доступом і налаштуваннями адміністративних функцій. Вони відповідають за обробку запитів, пов'язаних зі створенням, читанням, оновленням і видаленням відповідних сутностей (брендів, категорій, коментарів, продуктів, транзакцій). Контролери можуть містити методи, які обробляють запити для відображення списків, форм створення/редагування та деталей сутностей. Також взаємодіяти з відповідними моделями (Models) і уявленнями (Views) для отримання і передачі даних. Це дає змогу відокремити функції адміністративні від призначених для користувача і забезпечити зручніше керування та взаємодію веб-застосунку для адміністратора та користувача.

Контролери для користувача відповідають за запити, пов'язані з функціональністю, доступною кінцевим користувачам (наприклад, зміна особистих даних, перегляд і управління кошиком, обраними і замовленнями). Контролер кошика обробляє запити, пов'язані з додаванням, видаленням і оновленням товарів у кошику користувача. Контролер каталогу відповідає за запити, пов'язані з відображенням категорій, фільтрацією за брендами, цінами та іншими параметрами, а також дає змогу користувачеві переглядати й вибирати товари для покупки. Контролер для платежів з використанням Stripe обробляє запити, пов'язані з оплатою через платіжну систему Stripe. Він взаємодіє з API Stripe для створення платежів.

Структура контролерів наведена на рисунку 3.1.

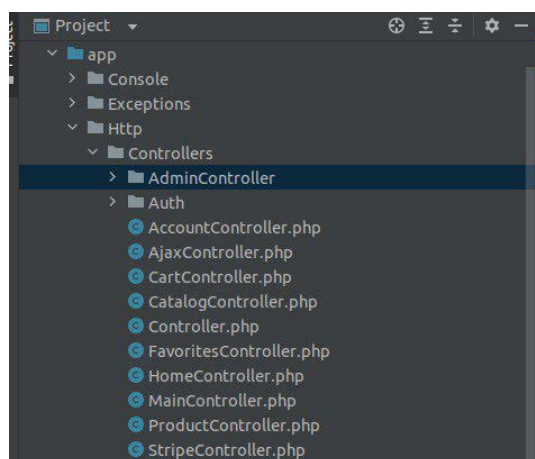


Рисунок 3.1 – Структура контролерів

### 3.2.2 Middleware

Middleware у Laravel є проміжними шарами, які дають змогу розмежовувати доступ і виконувати певні дії перед обробкою запитів. У нашому проєкті використовуються наведені нижче.

Middleware авторизації (Authentication Middleware): Цей middleware перевіряє, чи автентифікований користувач і чи має він доступ до певних ресурсів. Він здійснює перевірку автентичності користувача і встановлює сеанс автентифікації. Якщо користувач не пройшов аутентифікацію, його буде перенаправлено на сторінку входу.

Middleware перевірки бана (Ban Middleware): Цей middleware перевіряє, чи не заблокований користувач і чи не має він обмежень у доступі. Він перевіряє наявність користувача в списку заблокованих. Якщо користувач заблокований, йому буде показано сторінку з інформацією про блокування або відмовлено в доступі.

Middleware перевірки CSRF захищає від атак, генеруючи унікальний токен для кожної форми і перевіряючи його під час обробки запиту. Якщо токен не збігається або відсутній, запит буде відхилено.

Використання цих middleware у нашому проєкті дає змогу забезпечити безпеку та контроль доступу, а також захистити застосунок від атак і несанкціонованого використання. Це допомагає забезпечити безпеку користувацьких даних і зберегти цілісність програми.

### 3.2.3 Валідація даних

У Laravel для валідації даних, що надсилаються через запити, використовуються Request класи. Request класи надають зручний спосіб визначення правил валідації для кожного поля вхідного запиту.

У нашому проєкті ми також використовуємо валідацію даних для забезпечення цілісності та коректності інформації. Наприклад, під час зміни

даних користувача ми перевіряємо, щоб не було створено користувачів з однаковими поштовими адресами або без зазначення пароля. Це дає змогу забезпечити унікальність користувачів і зберегти безпеку їхніх акаунтів.

В адмінській частині проєкту, під час роботи з товарами, категоріями та брендами, ми також застосовуємо валідацію для запобігання введенню порожніх даних або некоректних зображень. Наприклад, при створенні або оновленні товару, ми перевіряємо, щоб було вказано назву, ціну, опис і коректне зображення товару. Це допомагає забезпечити правильність і повноту даних, що зберігаються в системі.

Використання валідації даних за допомогою Request класів у Laravel дає нам змогу легко визначити і застосувати правила валідації для кожного поля. Якщо вхідні дані не відповідають правилам валідації, Laravel автоматично повертає помилки, які можна відобразити користувачеві для вказівки на помилкові дані та запобігання збереженню некоректної інформації.

### **3.2.4 MailGun**

У нашому проєкті ми використовуємо сервіс Mailgun для надсилання сповіщень про реєстрацію, додавання товарів до кошика, додавання товарів до вибраного та про покупки. Сервіс Mailgun надає надійну та масштабовану інфраструктуру для надсилання електронної пошти.

### **3.2.5 Моделі**

У розробці веб-додатків на Laravel моделі являють собою класи, які забезпечують доступ до даних у базі даних і надають зручні методи для роботи з цими даними. Моделі зазвичай пов'язані з таблицями в базі даних і дають змогу виконувати різні операції, як-от створення, читання, оновлення та видалення даних. Можна задати, які поля будуть змінюватися (fillable), які будуть приховані (hidden), а також зв'язки з іншими класами

### 3.2.6 Міграції

Для роботи з даними в ларавел використовуються міграції, які дозволяють створювати таблиці в бд. Також у нас використовуються seeders для автоматичного заповнення деяких таблиць, наприклад таблиця адміна, де буде одразу створено акаунт адміна. За допомогою міграцій і сідерів у Laravel можна ефективно керувати структурою бази даних і заповнювати її початковими даними (рис.3.2). Це спрощує розробку та забезпечує консистентність даних у додатку. Більш детальну структуру нашої бази даних буде розглянуто в наступних пунктах.

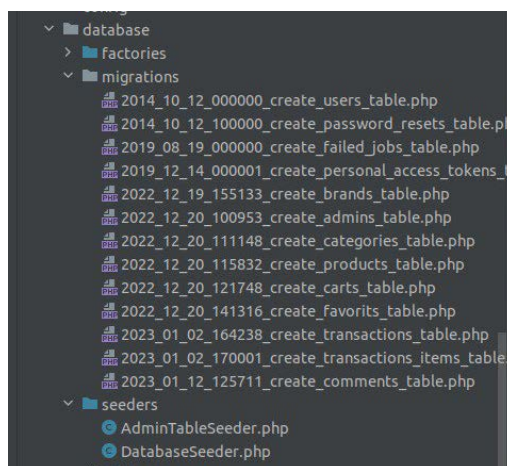


Рисунок 3.2 – Структура міграції

### 3.2.7 Збереження файлів

У нашому проєкті ми вирішили зберігати статичні файли, як-от зображення, скрипти та стилі, у папці public. Папка public є точкою входу для веб-сервера, тому всі файли, розміщені в цій папці, можуть бути безпосередньо доступні з браузера. Структура папки public має такий вигляд:

– public/images - папка для зберігання зображень, які будуть відображатися на веб-сторінках, у цій папці ми можемо створювати підпапки для категоризації зображень за певними критеріями, наприклад, public/images/products для зображень товарів;

- `public/js` - папка для зберігання JavaScript файлів, які будуть використовуватися на веб-сторінках, можна розміщувати тут усі скрипти, необхідні для функціональності нашого застосунку;

- `public/css` - папка для зберігання файлів стилів CSS, тут можна розміщувати всі файли CSS, які визначають зовнішній вигляд і макет наших веб-сторінок.

Зберігання статичних файлів у папці `public` дає нам змогу зручно керувати та звертатися до них у коді нашого застосунку. Можна посилатися на зображення, скрипти та стилі, використовуючи відносні шляхи, наприклад, `/images/logo.png` або `/js/app.js`. Це забезпечує простоту під час роботи з цими ресурсами.

### 3.2.8 Views

У проєкті організовано структуру зберігання подань (views) для забезпечення зручності розробки та підтримки. Є дві основні папки - `admin` і `user`, у яких зберігаються подання для адмінпанелі та користувацької частини відповідно (рис. 3.3).

У кожній із цих папок ми створили шаблони, які забезпечують загальний макет і включають загальні елементи, як-от заголовки, футери, підключення стилів і скриптів. Це дозволяє нам уникнути дублювання коду і забезпечити однаковий зовнішній вигляд нашого додатка.

Крім того, у нас є папка `partials`, у якій зберігаються окремі компоненти або частини сторінок, як-от модальні вікна, контент, що оновлюється за допомогою AJAX, та інші елементи, що повторно використовуються. Це дає нам змогу розділити код і повторно використовувати ці компоненти в різних поданнях.

В адмін-панелі в нас також є окремі папки для кожної сутності, такі як `users`, `products`, `categories`, `brands`, де містяться сторінки для загального списку,

редагування та видалення відповідних елементів. Це допомагає нам організувати код і забезпечити зручне управління даними в адмінській частині.

Структуроване зберігання подань дає нам змогу легко навігуватися по коду проєкту, знаходити потрібні файли та проводити зміни. Воно також сприяє повторному використанню коду і забезпечує легкість підтримки та розширення програми.

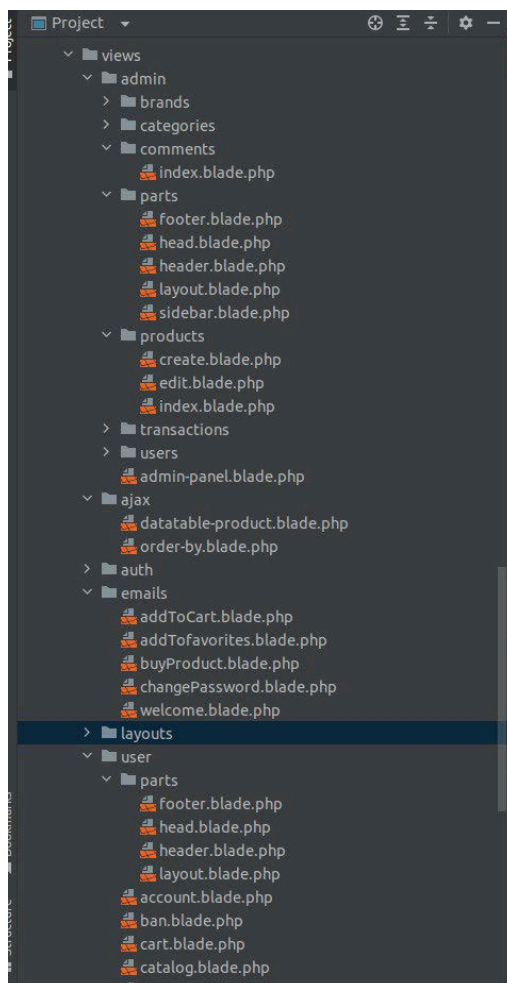


Рисунок 3.3 – Структура views

### 3.2.9 Роути

У нашому проєкті ми організували окремі роути для адмінської та користувацької частин застосунку.

Для адміністративної частини ми створили окремий роутер, який відповідає за обробку запитів, пов'язаних з адміністративними функціями. У цьому роутері визначено маршрути для різних дій, як-от перегляд, створення, редагування та видалення користувачів, товарів, категорій, брендів та інших сутностей. Наприклад, у нас може бути маршрут `/admin/users` для перегляду списку користувачів, `/admin/products/create` для створення нового товару тощо.

Для користувацької частини застосунку в нас також є окремий роутер, який відповідає за обробку запитів, пов'язаних із функціональністю, доступною користувачам. Тут ми визначаємо маршрути для сторінок, таких як головна сторінка, сторінка каталогу товарів, сторінка перегляду окремого товару, кошик, обрані та інші. Наприклад, у нас може бути маршрут `/products` для відображення списку товарів і `/cart` для роботи з кошиком.

Кожен роут визначає відповідний контролер і метод контролера, який оброблятиме запити та виконуватиме відповідні дії. Це дає нам змогу логічно структурувати функціональність застосунку та обробляти запити відповідно до їхнього призначення.

Організація окремих роутів для адмінської та призначеної для користувача частин застосунку допомагає нам забезпечити поділ функціональності, зручну навігацію та забезпечити безпеку, контролюючи доступ до різних частин застосунку залежно від ролей користувачів.

### **3.3 Архітектура бази даних**

У проєкті було використано 10 таблиць даних, для зручнішого розуміння, винесено кожен в окрему таблицю. Кожна таблиця має за замовчуванням поле `id`, яке є основним атрибутом у таблиці бази даних і являє собою унікальний ідентифікатор для кожного запису в таблиці. Кожне поле `id` є унікальним і має властивість автоінкрементції, тобто кожне поле автоматично збільшує його значення для кожного нового запису.

### 3.3.1 Таблиця користувача і адміна

Таблиця "користувачі" (Users) є однією з основних таблиць у базі даних, що використовується для зберігання інформації про користувача (рис. 3.4).

| Users                                                                                                                                                            |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>-id (int)</li><li>-name (string)</li><li>-email (string)</li><li>-password (string)</li><li>-is_banned (boolean)</li></ul> |

Рисунок 3.4 – Таблиця даних користувачів

Ім'я (Name): Зберігає ім'я користувача і використовується для відображення його імені в системі.

Електронна пошта (Email): Містить адресу електронної пошти користувача і використовується для зв'язку з ним і надсилання повідомлень.

Пароль (Password): Зберігає зашифрований або хешований пароль користувача і використовується для аутентифікації його при вході в систему.

IsBanned: Індикатор статусу блокування користувача і використовується для керування доступом користувача до функцій і ресурсів системи. Якщо поле має значення "заблоковано", користувачу обмежено доступ, якщо значення "не заблоковано", то доступ відкритий. Таблиця "адмін" (Admin) має такі ж поля, за винятком поля is\_banned (рис 3.5).

| Admins                                                                                                                              |
|-------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>-id (int)</li><li>-name (string)</li><li>-email (string)</li><li>-password (string)</li></ul> |

Рисунок 3.5 – Таблиця даних адміністратора

### 3.3.2 Таблиця брендів та категорій

Таблиця "бренди" (Brands) та "категорії" (Categories) створені для відокремлення товарів та сортуванням їх за цими параметрами та мають однакову структуру (рис.3.6).

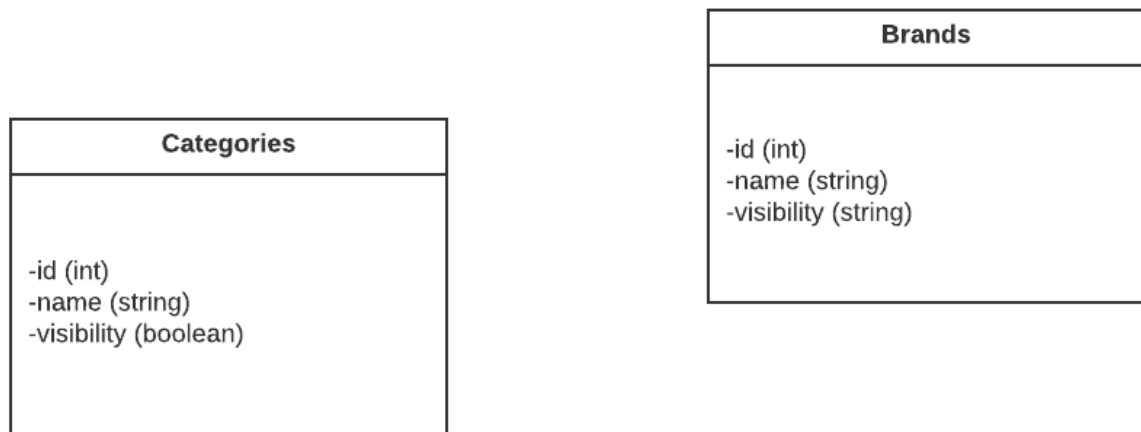


Рисунок 3.6 – Таблиця категорії та бренда

Таблиця "категорії" (Categories) містить поля:

- name (Назва): зберігає назву категорії, це текстове поле, яке містить назву категорії;
- visibility (Видимість), логічне значення, що вказує, чи є категорія видимою, це поле використовується для керування видимістю категорій у системі.

### 3.3.3 Таблиця продуктів

Таблиця "продукти" (Products) містить інформацію про різні продукти, доступні в системі (рис.3.7).

| Products                                                                                                                                                                                                                                                                                                                    |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -id (int)<br>-featured (boolean)<br>-name (string)<br>-description (text)<br>-price (float)<br>-category (string) -><br>CATEGORY:name (string)<br>-brand (string) -><br>BRAND:name (string)<br>-weight (string)<br>-dimensions (string)<br>-color (string)<br>-country (string)<br>-visibility (boolean)<br>-image (string) |

Рисунок 3.7 – Таблиця даних продукту

Рекомендовані (featured) – логічне значення, що вказує, чи є продукт рекомендованим.

Назва (name) – зберігає назву продукту.

Опис (description) – містить опис продукту..

Ціна (price) – зберігає ціну продукту. Це числове поле, яке вказує вартість продукту.

Категорія (category) – зовнішній ключ (foreign key), що посиляється на таблицю "категорії" і вказує категорію, до якої належить продукт.

Бренд (brand) – зовнішній ключ (foreign key), що посиляється на таблицю "бренди" і вказує бренд, до якого належить продукт.

Вага (weight) – зберігає вагу продукту.

Розміри (dimensions) – зберігає розміри продукту. Це текстове поле, яке містить інформацію про розміри продукту (наприклад, довжину, ширину, висоту).

Колір (color) – зберігає колір продукту.

Країна (country) – зберігає країну виробництва продукту.

### 3.3.4 Таблиця кошику та обраних

Таблиця обраних (Favorits), яка представлена на рис.3.8, та "кошику" (Carts), яка представлена на рис.3.9, та містить інформацію про товари, які знаходяться у користувача в кошику, або в обраних товарах.

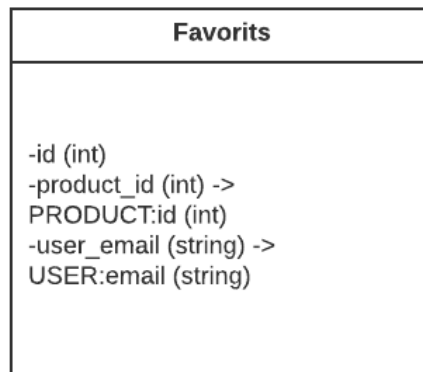


Рисунок 3.8 – Таблиця даних обраних

Таблиця даних обраних містить поля product\_id яке зберігає ідентифактор продукту, котрий був доданий у обрані товари, а також поле user\_email, яке зберігає адресу електронної пошти користувача, котрому належить кошик.

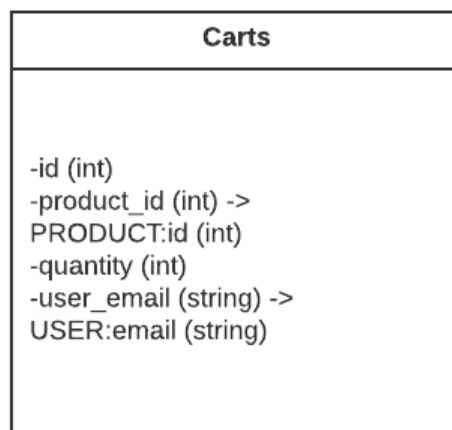


Рисунок 3.9 – Таблиця даних кошику

Таблиця кошика має ті самі поля, що й обрані, за винятком додавання нового поля кількість (quantity), яке відповідає за кількість певного продукту в кошику

### 3.3.5 Таблиці транзакції

У системі зберігаються дані про всі транзакції користувачів. Можна переглянути кожну оплату користувача (Transactions), та товари які були придбані (Transactions\_items). Ці таблиці пов'язані зовнішньою таблицею, яка знаходиться поза нашим проектом, а саме зберігається на серверах Stripe.

Таблиця transactions слугує для зберігання інформації про платіжні транзакції у вашому вебзастосунку і дає змогу їх відстежувати (рис.3.10). Вона містить такі поля:

- Price – ціна транзакції;
- Stripe\_id – ідентифікатор транзакції в сервісі Stripe;
- User\_email – адреса електронної пошти користувача, який здійснив транзакцію;
- Paid – флаг, що вказує, чи оплачено транзакцію.

| Transactions                                                                                                                                              |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| -id (int)<br>-price (float)<br>-stripe_id (string) -><br>STRIPE:id (STRIPE DATABASE)<br>-user_email (string) -><br>USER:email (string)<br>-paid (boolean) |

Рисунок 3.10 – Таблиця транзакції

Таблиця transactions\_items (рис.3.11) служить для перегляду інформації про товари, які були оплачені:

- Stripe\_id – ідентифікатор транзакції в сервісі Stripe;
- Product\_id – ідентифікатор продукту, який відноситься к данії транзакції;
- Quantity – кількість продукту, придбаного в рамках цієї транзакції.

| Transactions_items                                                                                                                   |
|--------------------------------------------------------------------------------------------------------------------------------------|
| -id (int)<br>-stripe_id (string) -><br>STRIPE:id (STRIPE DATABASE)<br>-product_id (id) -><br>PRODUCT:id (int)<br>-quantity (integer) |

Рисунок 3.11 – Таблиця елементів транзакції

Таблиця transaction\_items пов'язує продукти з транзакціями. Кожен запис у таблиці являє собою елемент транзакції та зберігає інформацію про кількість кожного продукту, придбаного в межах транзакції.

### 3.3.6 Таблиці коментарів

У вашій базі даних буде таблиця comments (коментарі), яка міститиме такі поля (рис.3.12):

- product\_id – ідентифікатор продукту, до якого належить цей коментар;
- author – ім'я автора коментаря;
- text – текст коментаря.

| Comments                                                                                                                          |
|-----------------------------------------------------------------------------------------------------------------------------------|
| -id (int)<br>-name (string)<br>-product_id (int) -><br>PRODUCT:id<br>-author (string) -><br>USER:email (string)<br>-text (string) |

Рисунок 3.12 – Таблиця елементів транзакції

Таблиця comments дає змогу зберігати коментарі, пов'язані з певними продуктами. Кожен запис у таблиці містить інформацію про ідентифікатор продукту, до якого належить коментар, ім'я автора коментаря та текст коментаря. Це дозволяє користувачам залишати коментарі до певних продуктів.

### 3.3.7 Загальна структура таблиць

На рисунку 3.13 представлено стан бази даних з усіма таблицями та їхніми зв'язками

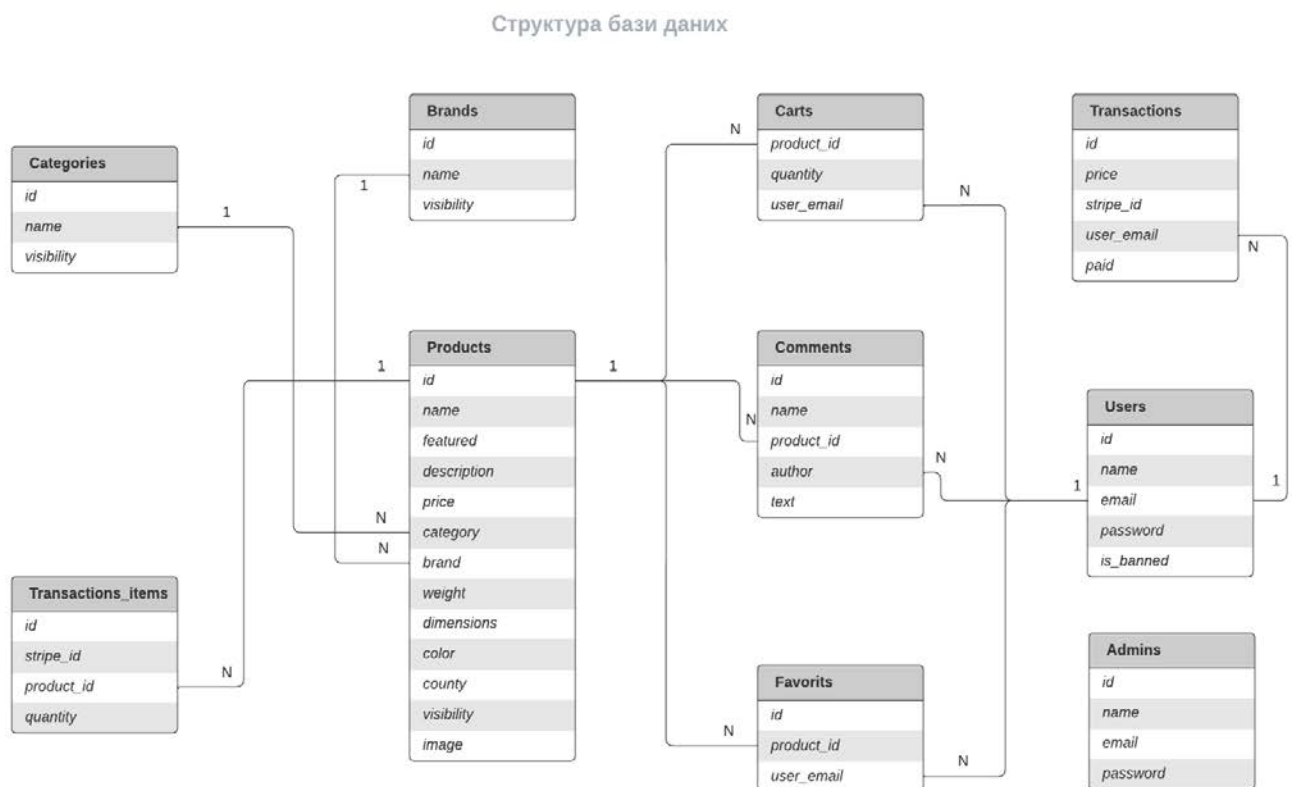


Рисунок 3.13 – Таблиця бази даних

Кожна таблиця має свої поля, які описують інформацію, що зберігається в них. Також таблиці мають між собою зв'язки один до багатьох, що означає, що один елемент із першої таблиці може мати кілька відповідних елементів у другій таблиці.

### 3.4 Послідовність дій користувача

#### 3.4.1 Користувач магазину

Під час розробки сайту, структура сторінок відіграє важливу роль у забезпеченні функціональності, логічності та ефективності взаємодії користувачів із сайтом (рис.3.14). У цьому розділі ми опишемо основні сторінки, які присутні в нашому проєкті, а також обговоримо логіку їхнього зв'язку.

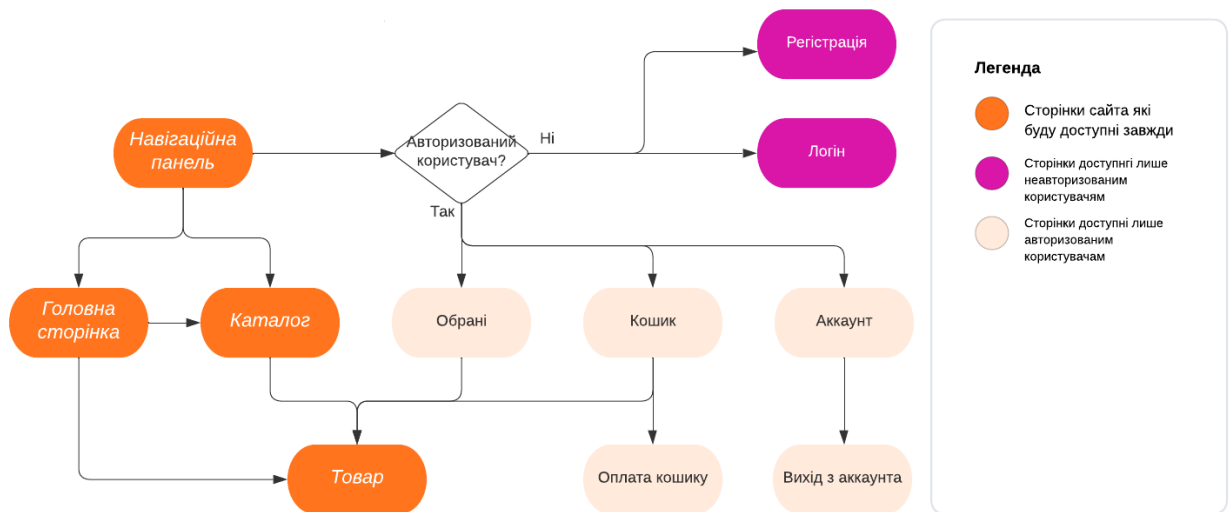


Рисунок 3.14 – Діаграма дії користувача

**Головна сторінка:** Це сторінка, на яку користувачі вперше потрапляють при відкритті сайту. Тут представлена загальна інформація про сайт, актуальні пропозиції, пропозиції які вибрав адмін та категорії.

**Сторінка каталогу:** Ця сторінка містить список усіх доступних товарів або послуг. Користувачі можуть переглядати товари за категоріями, фільтрувати за різними параметрами (наприклад, ціна, бренд), та здійснювати пошук конкретних товарів.

**Сторінка продукту:** Кожен товар має свою окрзалишити коментар му сторінку, на якій представлено детальну інформацію про товар, його характеристики, зображення та ціну. Тут користувачі можуть додати товар до кошика або обраного.

Сторінка кошика: Ця сторінка відображає список товарів, обраних користувачем для покупки. Тут користувачі можуть змінювати кількість товарів, видаляти товари з кошика і переходити до оформлення замовлення.

Сторінка оформлення замовлення: На цій сторінці користувачі вводять свої дані для доставки та обирають спосіб оплати. Після підтвердження замовлення відбувається перехід до процесу оплати.

Особистий кабінет користувача: Це сторінка, де користувачі можуть переглянути свої дані, історію замовлень, керувати налаштуваннями акаунта, змінювати особисті дані, а також відстежувати статус доставки.

Сторінка реєстрації: Це сторінка, на якій користувачі можуть створити свій акаунт на сайті. Тут вони заповнюють необхідні дані, як-от ім'я, електронна пошта та пароль, щоб зареєструватися й отримати доступ до додаткових функцій.

Сторінка входу (логіна): Ця сторінка надає користувачеві можливість увійти у свій акаунт, використовуючи раніше створені облікові дані. Після успішного входу користувач отримує доступ до свого особистого кабінету або інших обмежених функцій сайту.

### **3.4.1 Адмін-панель**

В адмін-панелі ми також маємо певну структуру сторінок, яка дозволяє адміністраторам керувати сайтом і виконувати необхідні дії (рис.3.15). Основні сторінки адмін-панелі включають:

Сторінка керування користувачами: Ця сторінка надає адміністраторам можливість переглядати список користувачів, список коментарів у користувача, блокувати або видаляти акаунти користувачів. Також можна логінітися під цим юзером

Сторінка керування товарами: На цій сторінці адміністратори можуть переглядати список товарів, редагувати їхні характеристики, зображення та інші

деталі, додавати нові товари або видаляти існуючі. Також можна приховати товари та подивитися коментарі.

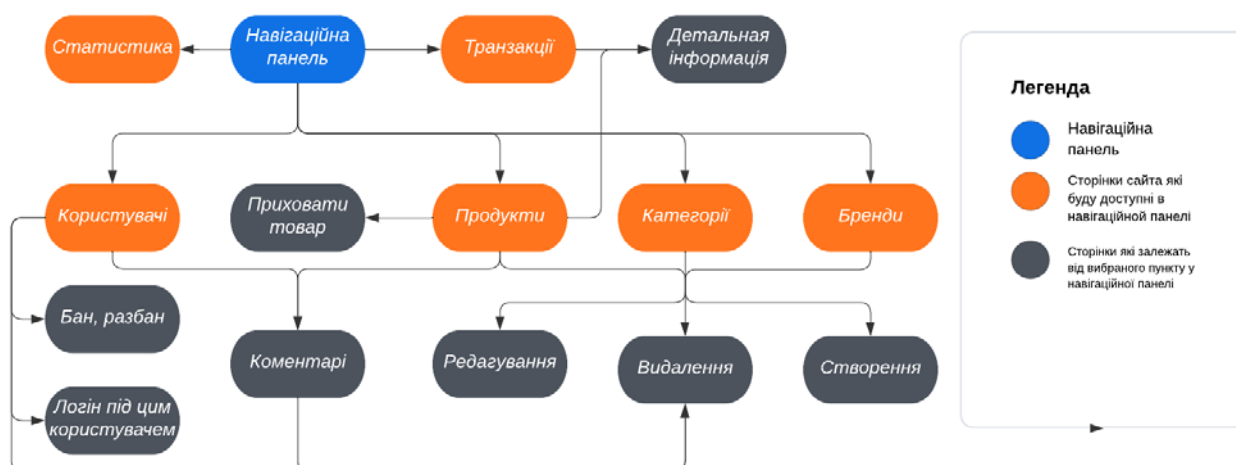


Рисунок 3.15 – Діаграма дії адміністратора

Сторінка керування категоріями: Тут адміністратори можуть переглядати список категорій товарів, додавати нові категорії, редагувати або видаляти існуючі.

Сторінка керування брендами: Адміністратори можуть переглядати список брендів, додавати нові бренди, редагувати або видаляти існуючі.

Сторінка керування коментаріями: На цій сторінці адміністратори можуть переглядати коментарі, які залишили користувачі на продуктах та видаляти їх.

Сторінка транзакції: Тут адміністратори можуть переглянути всі оплати користувачів, а також переглянути більш детальну інформацію про кожну оплату, які продукти були куплені і в яких кількостях.

Сторінка статистики: На цій сторінці адміністратори можуть продивитися статистику їх магазину. Кількість зареєстрованих користувачів, кількість проданого товару та вартість товарів.

Ця структура сторінок адмін-панелі дозволяє адміністраторам зручно керувати сайтом та контролювати всі аспекти функціонування веб-додатка.

## 4 ГРАФІЧНА ЧАСТИНА ВЕБЗАСТОСУНКУ

### 4.1 Основні вимоги і тренди

У сучасному дизайні веб-додатків існує декілька трендів, які варто враховувати під час розробки проекту. Ми плануємо втілити такі тренди в нашому проекті, враховуючи його тематику і вимоги. Ось основні тренди, які ми збираємося впровадити.

**Мінімалістичний дизайн:** Простота і мінімалізм залишаються популярними трендами. Чистий і простий дизайн з акцентом на простір, типографіку і кольори допомагає створити елегантний і сучасний вигляд. Мінімальна наявність кольорів, в основному білий чорний і зелений, а також простий шрифт для зручності читання. Велика частина сторінки ні чим не зайнята і залишається багато порожнього простору. Відсутність випадajoчих вікон, банерів та іншого.

**Адаптивний дизайн:** Зі збільшенням кількості пристроїв і різних розмірів екранів, збільшує привабливість нашого додатка, що робить його більш зручним на телефонах і охоплює більшу цільову аудиторію. Важливо створювати веб-додатки, які добре відображаються і працюють на різних пристроях, чи то настільний комп'ютер, чи то планшет, чи то смартфон.

**Інтерактивність:** Додавання інтерактивних елементів і анімацій робить користувацький досвід більш привабливим і привертає увагу користувачів. Це може включати анімовані переходи, навігацію та елементи, які реагують на дії користувача. Наприклад, під час наведення на товари або кнопки відбуватимуться анімації, щоб підкреслити їхню клікабельність і викликати візуальну реакцію у користувачів.

**Мікроінтеракції:** Мікроінтеракції - це маленькі анімації або реакції, які відбуваються у відповідь на дії користувача, як-от натискання кнопок або переміщення вказівника миші. Вони додають елементи задоволення і покращують взаємодію користувача з веб-додатком. на кожну дію користувача. Наприклад, під час додавання товару в кошик або вибране, з'явиться спливаюче вікно з повідомленням про успішне додавання.

З огляду на вимоги та поточні тренди, ми прагнемо створити сучасний, функціональний і привабливий дизайн для нашого веб-додатку, який задовольнятиме потреби та очікування користувачів [14].

## 4.2 Графічний дизайн для користувачів

### 4.2.1 Навігаційна панель, реєстрація та авторизація

Розглянемо для початку навігаційну панель (рис. 4.1). Вона призначена для навігації на більшість сторінок, не повинна містити надто багато інформації, але водночас повинна містити можливість перенаправлятися на більшість сторінок нашого сайту. Панель має два варіанти: для авторизованих і гостей (рис.4.2).



Рисунок 4.1 – Навігаційна панель магазину

Воно містить лого при натисканні на яке перенаправить на головну сторінку. Далі йде меню, що випадає, з категоріями, яке можна вибрати, не ходячи по сайту в пошуках категорій. Далі йдуть кнопки реєстрації та логіна

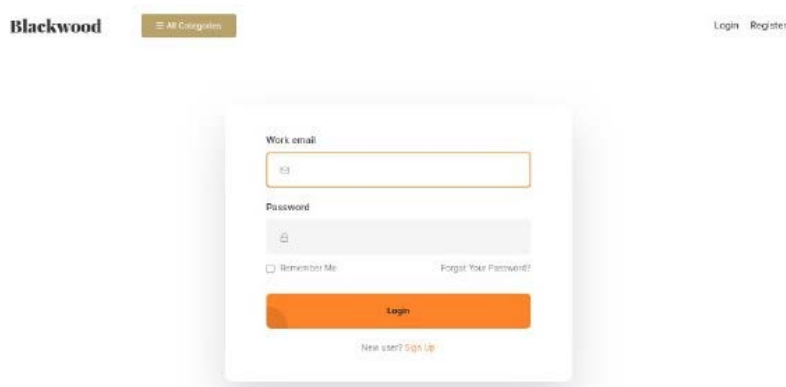


Рисунок 4.2 – Авторизація користувача

Меню реєстрації (рис.4.3) має максимально мінімалістичний вигляд, водночас залишаючись інтерактивним. Також присутня валідація полів, щоб користувач міг зрозуміти, які дані він ввів некоректні.

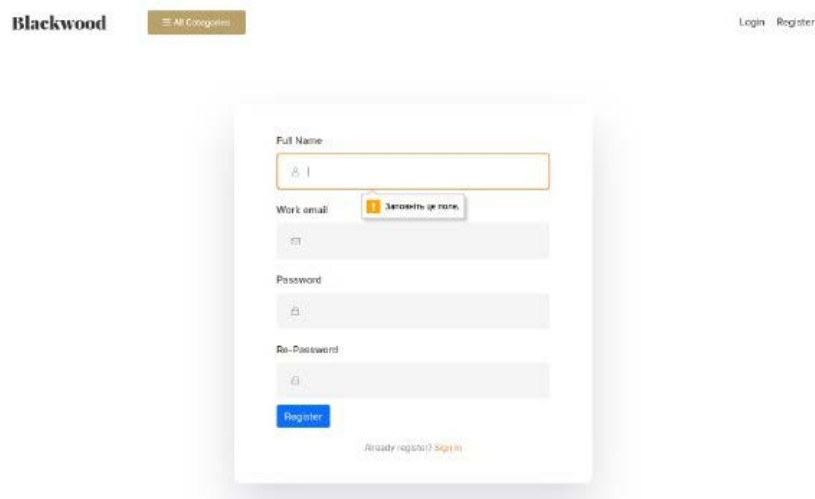
The image shows a registration form for a website named "Blackwood". At the top left is the "Blackwood" logo and a button labeled "All Categories". At the top right are links for "Login" and "Register". The form itself is centered and contains the following fields: "Full Name" with a text input and a validation icon; "Work email" with a text input, a validation icon, and a tooltip that says "Enter a valid email"; "Password" with a text input and a validation icon; and "Re-Password" with a text input and a validation icon. Below these fields is a blue "Register" button. At the bottom of the form, there is a link that says "Already register? Sign in".

Рисунок 4.3 – Реєстрація користувача

Сторінка логіна містить валідацію, а також можливість запам'ятати дані про вхід. Якщо користувач забуде пароль він зможе скористатися функцією скидання пароля. Між сторінками логіна та реєстрації можна легко перейти натиснувши на відповідне посилання під кнопкою. Після реєстрації користувачеві прийде на пошту лист із повідомленням про успішну реєстрацію.

На рисунку 4.4 зображена головна сторінка вебзастосунку.

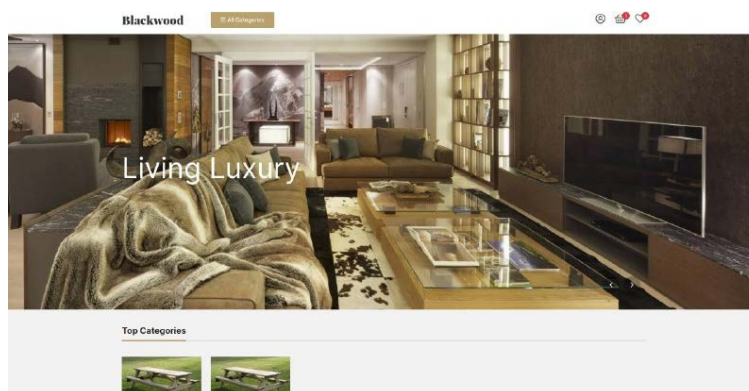


Рисунок 4.4 – Головна сторінка

Після того як юзер авторизується, його навігаційна панель зміниться. Він отримає доступ до акаунта свого, кошика та обраних товарів (рис.4.5).

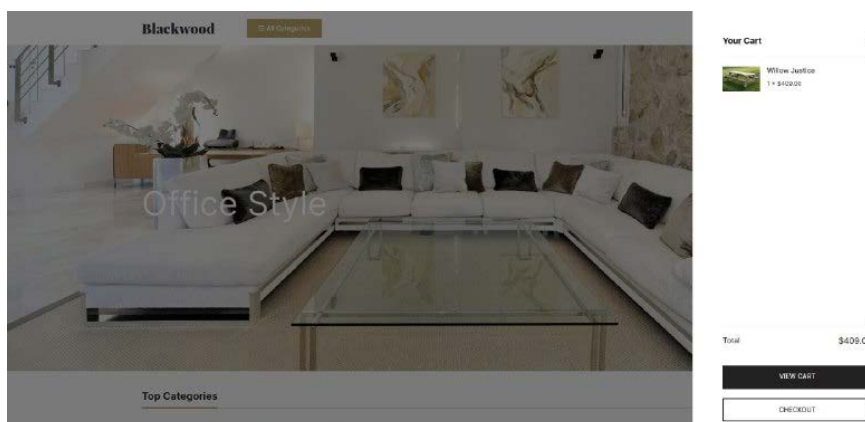


Рисунок 4.5 – Головна сторінка, вкладка кошик

Він може зробити попередній перегляд товарів, які перебувають у кошику, і одразу перейти до оплати або ж перейти до кошика, щоб докладніше подивитися про товар чи змінити кількість для купівлі.

#### 4.2.2 Головна сторінка

Головна сторінка містить інтерактивний слайдер (рис.4.6), який може містити інформацію про магазин і відокремлює візуально навігаційну панель від основного контенту, щоб користувач не губився на сторінці.

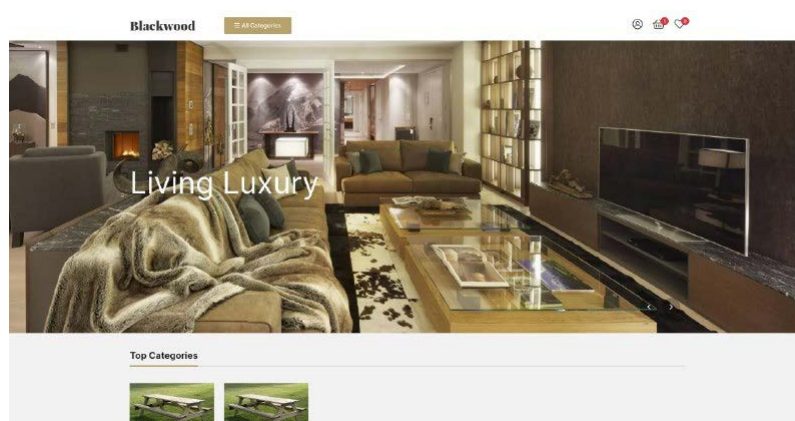


Рисунок 4.6 – Головна сторінка, слайдер

Далі йдуть усі категорії (рис.4.7), які додав адмін і зробив їх видимими. Далі знову йде меню з перевагами магазину, яке відокремлює наступний блок, де вже йтимуть товари. Можна перемикатися між відображенням товарів, які закріплені в адмін-панелі, і найостаннішими товарами.

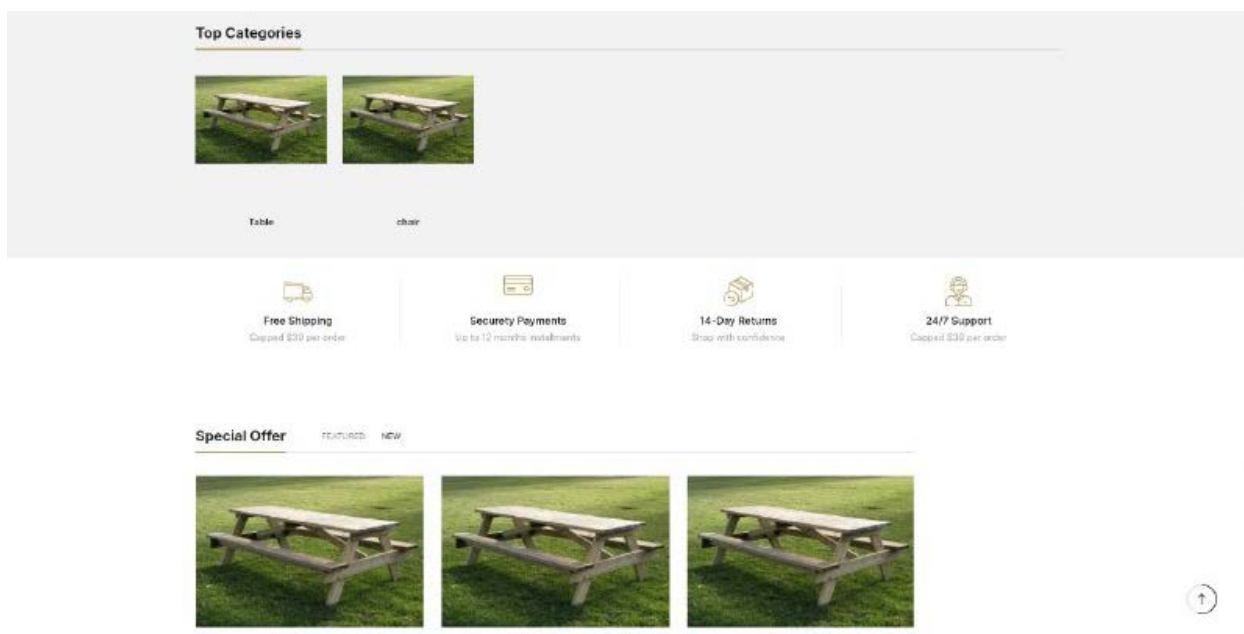


Рисунок 4.7 – Головна сторінка, категорії

Усі клікабельні елементи інтерактивні та мають додаткові кнопки. Наприклад, при наведенні на товар з'являються кнопки додавання в кошик і вибране.

Нижня панель (footer) є важною частиною вебзастосунку (рис.4.8).

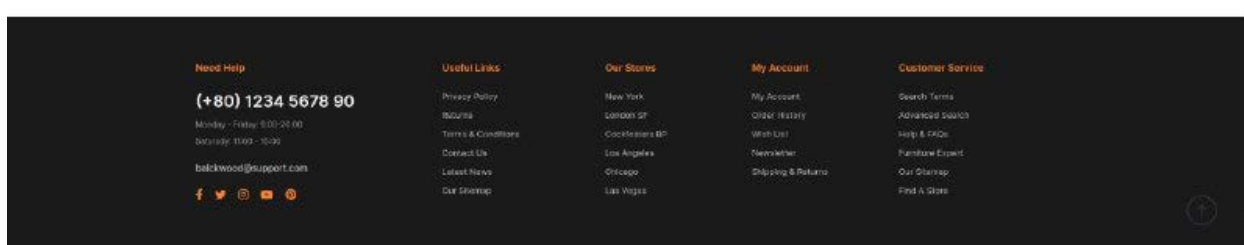


Рисунок 4.8 – Головна сторінка, нижня панель

Вона містить різноманітну інформацію про магазин і надає користувачам додаткові корисні посилання та контактну інформацію. Вона розташовується

внизу сторінки і часто містить контактні дані, посилання на важливі сторінки, соціальні медіа та інші гіперсилки.

### 4.2.3 Каталог продуктів

Панель каталогу (рис.4.9) - це функціональний інструмент, який допомагає користувачам знайти потрібні товари у вашому магазині, обираючи певну категорію або бренд. Вона надає зручний спосіб навігації каталогом товарів і полегшує процес пошуку.

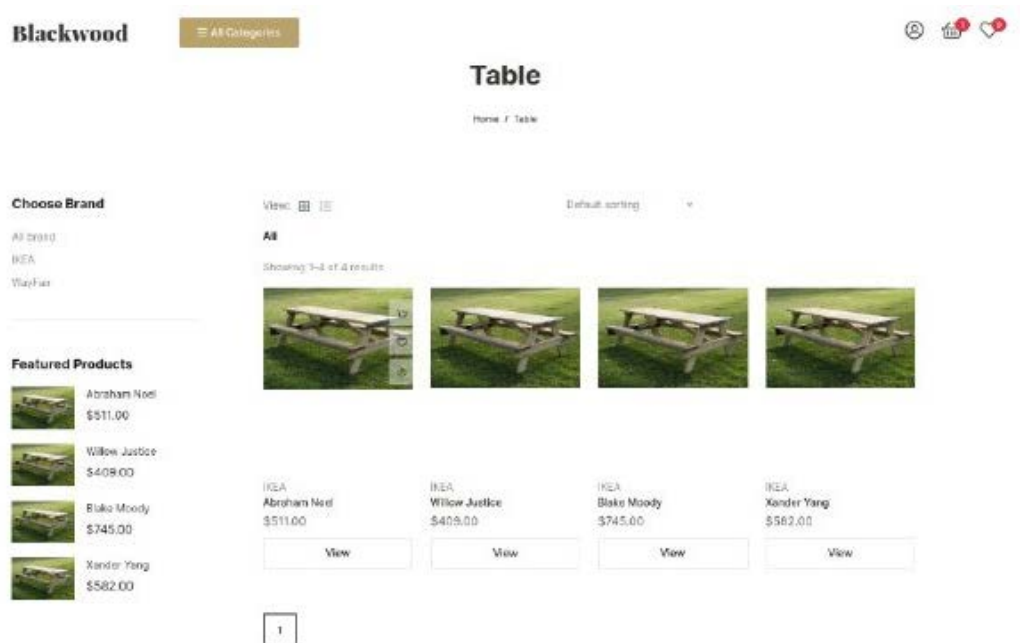


Рисунок 4.9 – Каталог товарів

У лівій частині панелі каталогу розташовується меню вибору брендів, яке дає змогу користувачам фільтрувати товари за конкретними виробниками або брендами. Це зручно для тих користувачів, які вважають за краще купувати товари певних марок або хочуть швидко ознайомитися з продукцією певного бренду.

Під меню брендів розташовуються товари, які були закріплені адміністратором. Це може бути добірка популярних або акційних товарів,

новинок або будь-яких інших товарів, які адміністратор вирішує виділити і представити на головній сторінці каталогу. Розміщення таких товарів під меню брендів дає змогу користувачам швидко ознайомитися з актуальними пропозиціями та звернути на них увагу.

Також можна змінювати зовнішній вигляд карток продуктів (рис.4.10) для більш зручного перегляду товарів.

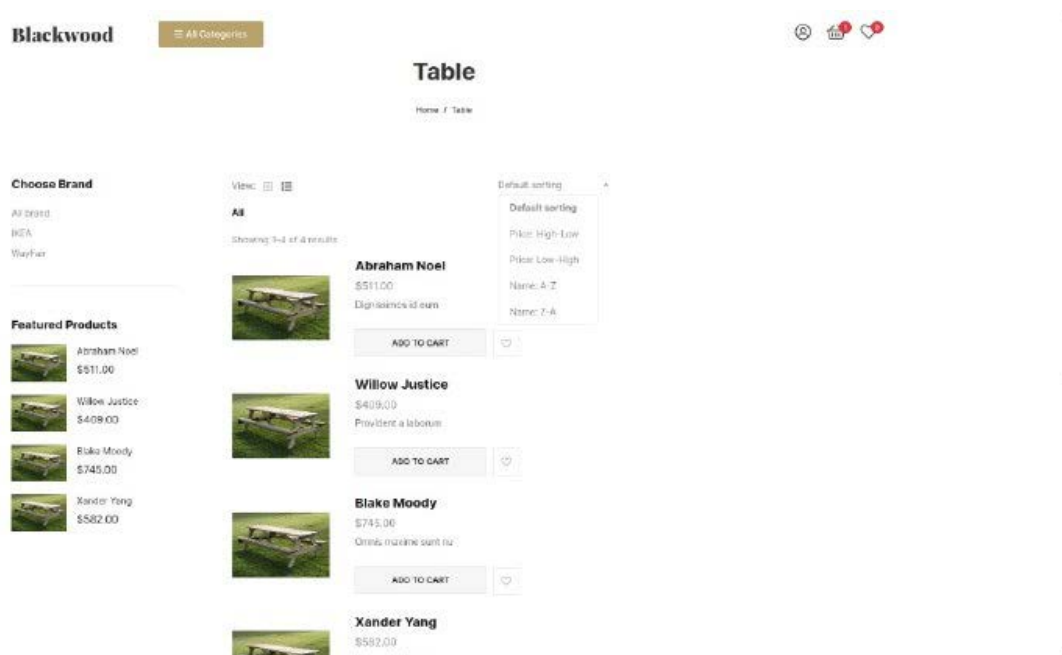


Рисунок 4.10 – Каталог товарів

В панелі каталогу також передбачена можливість сортування товарів за ціною та алфавітом. Це дає змогу користувачам легко знаходити потрібні товари відповідно до їхніх уподобань або потреб.

Сортування за ціною дає змогу відсортувати товари за зростанням або зменшенням ціни. Це корисно для користувачів, які хочуть знайти товари в певному ціновому діапазоні або порівняти ціни різних товарів. Наприклад, користувач може вибрати сортування "від дешевих до дорогих" або "від дорогих до дешевих" для зручного перегляду доступних товарів.

Сортування за алфавітом дає змогу впорядкувати товари за алфавітним порядком. Це особливо корисно, коли користувач шукає товари за назвою або

брендом. Сортування за алфавітом може бути здійснено в алфавітному порядку від А до Z або у зворотному порядку від Z до А.

#### 4.2.4 Каталог продуктів

На сторінці товару (рис.4.11) в нашому вебзастосунку можна знайти більш детальну інформацію про обраний товар. Тут будуть представлені всі характеристики, опис, зображення і ціна товару, щоб користувач міг отримати повне уявлення про нього.

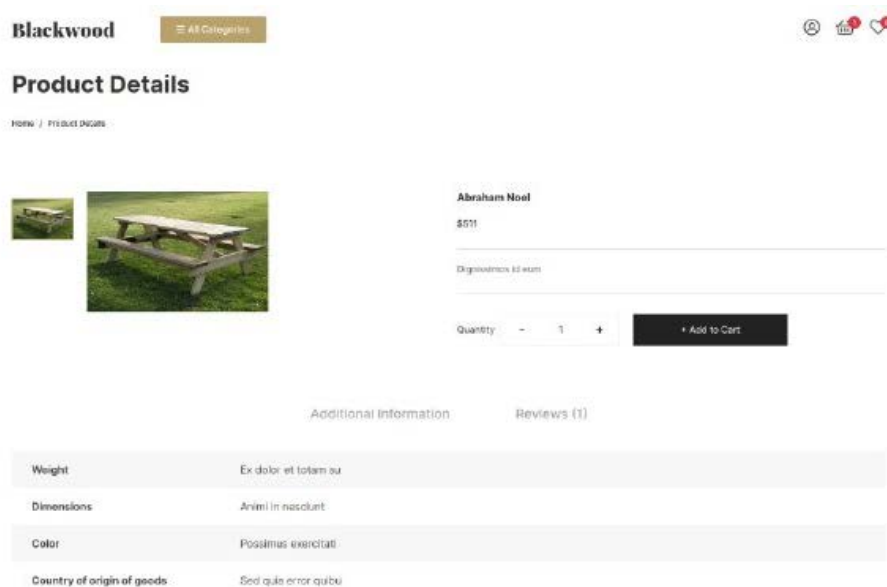


Рисунок 4.11 – Сторінка товару, детальна інформація

Також на сторінці товару буде розділ, в якому відображаються коментарі інших користувачів (рис. 4.12). Це надає можливість ознайомитися з відгуками, думками та рекомендаціями інших людей. Коментарі можуть містити корисні відомості про якість, функціональність або досвід використання товару.

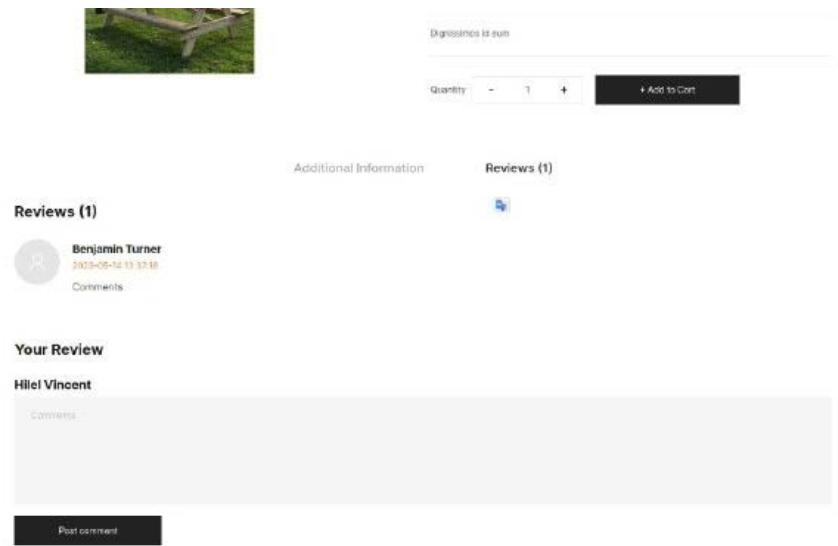


Рисунок 4.12 – Сторінка товару, коментарії

Більш того, на сторінці товару користувачі зможуть залишити свій власний коментар, щоб поділитися своєю думкою і відгуком про товар

4.2.5 Обрані

У розділі "Обрані" (рис. 4.13) в нашому веб-додатку користувачі зможуть побачити список товарів, які вони додали в обране. Кожен товар буде представлений у вигляді картки з короткою інформацією про нього.

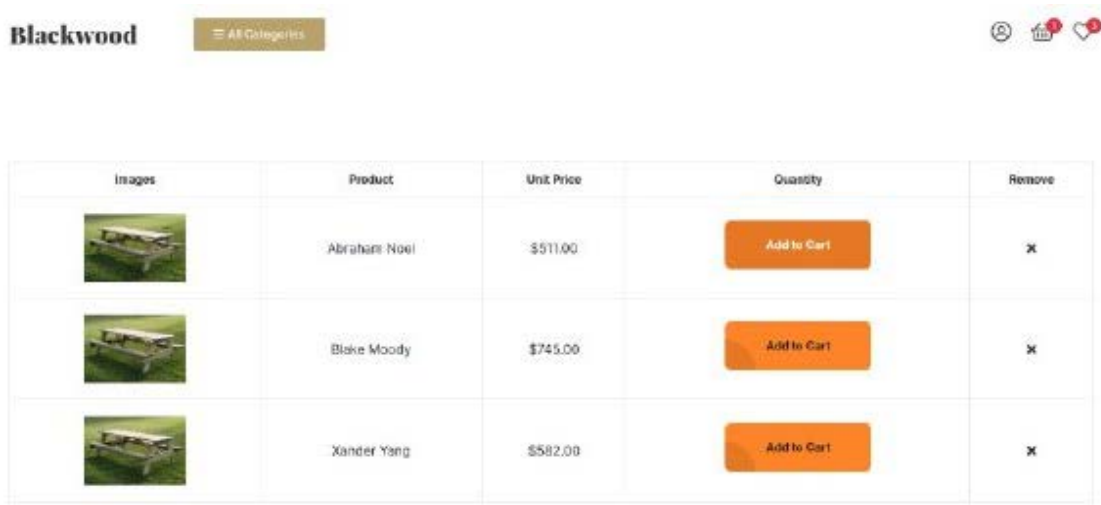


Рисунок 4.13 – Сторінка обраного

При кліці на товар зі списку обраних користувачі будуть перенаправлені на сторінку з детальною інформацією про обраний товар. Тут вони зможуть знайти всі характеристики, опис, зображення і ціну товару.

На сторінці товару в обраному буде доступна можливість додавання товару в кошик. Користувачі зможуть вибрати потрібну кількість товару і натиснути на кнопку "Додати в кошик". Це дасть змогу перенести товар з обраних у кошик для подальшого оформлення покупки.

Крім того, на сторінці товару в обраному буде присутня кнопка "Видалити з обраних". При натисканні на неї товар буде видалено зі списку обраних, і користувач більше не бачитиме його в цьому розділі.

Таким чином, розділ "Вибрані" надає зручну функціональність для збереження та управління товарами, які можна відстежувати або переглядати.

#### 4.2.6 Кошик

У розділі "Кошик" (рис. 4.14) нашого веб-додатку користувачі зможуть побачити список товарів, які вони додали для покупки. Кожен товар буде представлений у вигляді рядка, що містить інформацію про нього, таку як назва, зображення, кількість і ціна за одиницю товару.

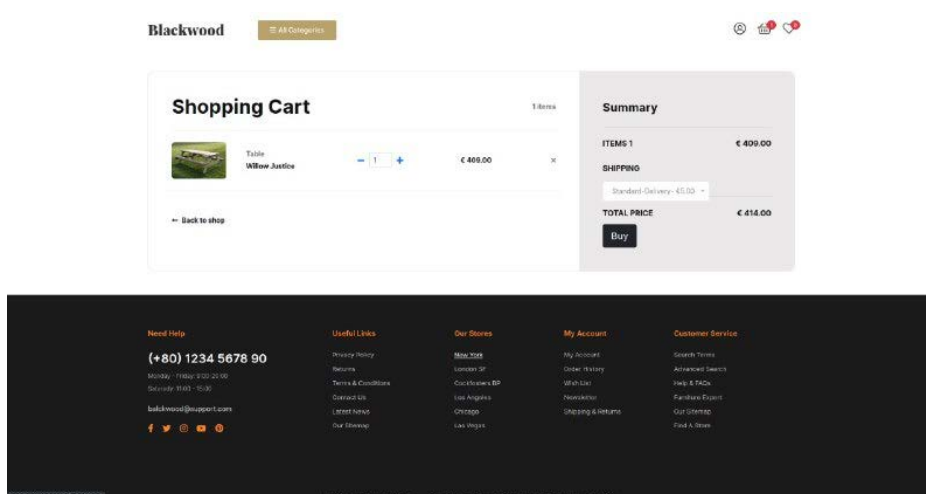


Рисунок 4.14 – Сторінка кошика

Для кожного товару буде доступна можливість зміни його кількості. Користувачі зможуть збільшувати або зменшувати кількість товару, щоб відобразити свої уподобання.

Праворуч від списку товарів у кошику відображатиметься вартість доставки. Це може бути фіксована сума або розрахована на основі обраного способу доставки та місця доставки.

Також на сторінці кошика буде вказана загальна вартість усіх товарів у кошику, включно з вартістю доставки. Це дасть змогу користувачам бачити і контролювати загальну суму замовлення перед переходом до оформлення покупки.

Використання такої структури в кошику дасть змогу користувачам зручно керувати своїми покупками, бачити вартість кожного товару, загальну суму замовлення і вартість доставки, що допоможе їм прийняти інформоване рішення перед оформленням покупки.

Після того, як користувач додав необхідні товари в кошик і переконався в правильності вибору, наш вебзастосунок надає можливість оплати з використанням сервісу Stripe (рис. 4.15).

The image shows a two-panel view of a Stripe payment interface. The left panel, titled 'Rezonsha TEST MODE', displays the order total as 'Order 414,00 \$'. The right panel, titled 'Pay' and 'Оплата с linked card', contains a form for card payment. The form includes fields for 'Email', 'Card number' (with a Visa logo), 'MM/YY' (with a card icon), 'Name and surname, as on the card', and 'Country or region' (set to 'Україна'). A checkbox option is present for saving card details. At the bottom of the right panel is a blue 'Pay' button.

Рисунок 4.15 – Сторінка оплати

При переході до процесу оплати, користувачеві буде запропоновано ввести свої дані платіжної картки, такі як номер картки, термін дії та CVV-код. Ці дані передаються на сервер Stripe для обробки платежу.

Використання сервісу Stripe для обробки платежів забезпечує зручність, безпеку та надійність у процесі оплати товарів у нашому веб-додатку. Це дає змогу користувачам впевнено та зручно здійснювати покупки, а нам - забезпечити безпеку та ефективність платіжного процесу.

#### 4.2.7 Аккаунт користувача

Після завершення реєстрації та авторизації користувач може перейти до свого акаунта, де доступні різні функціональності та налаштування.

У кабінеті користувача доступна інформація про всі його покупки. Користувач може переглянути список своїх замовлень, включно з деталями кожного замовлення, як-от дата покупки, статус доставки та сума платежу (рис. 4.16).

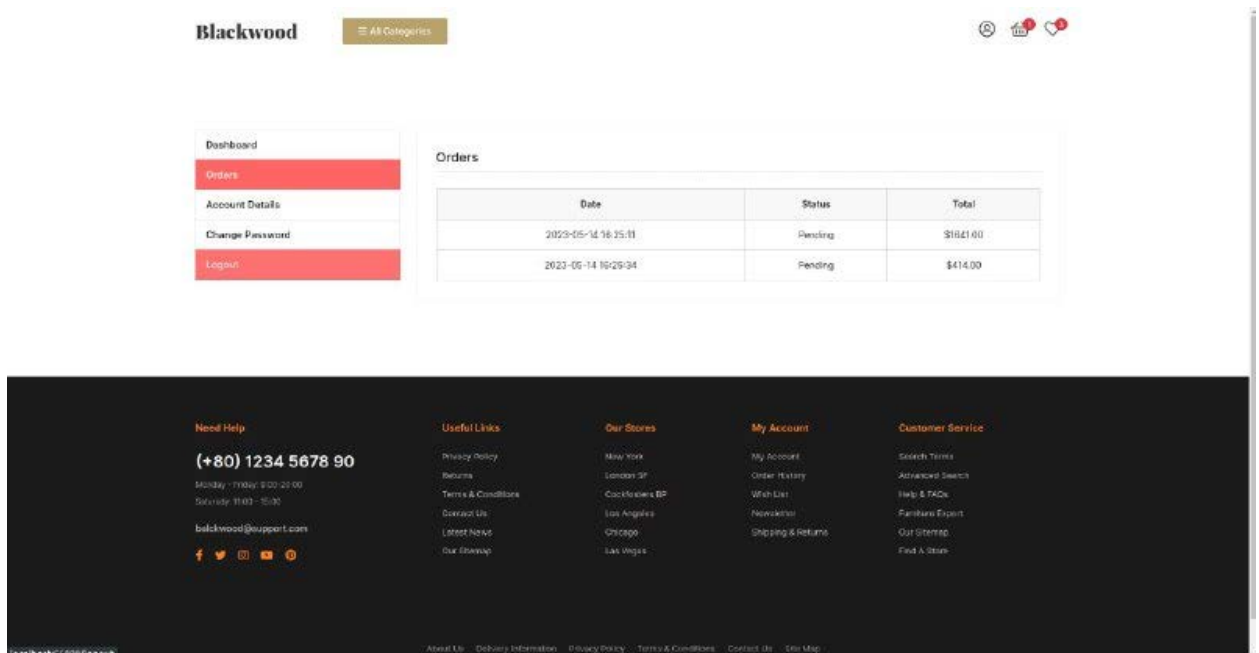
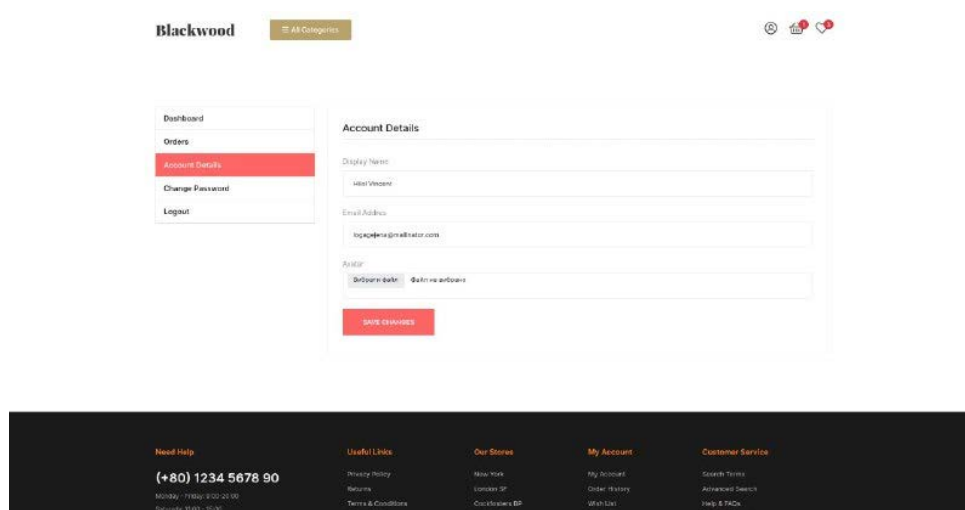


Рисунок 4.16 – Сторінка заказів

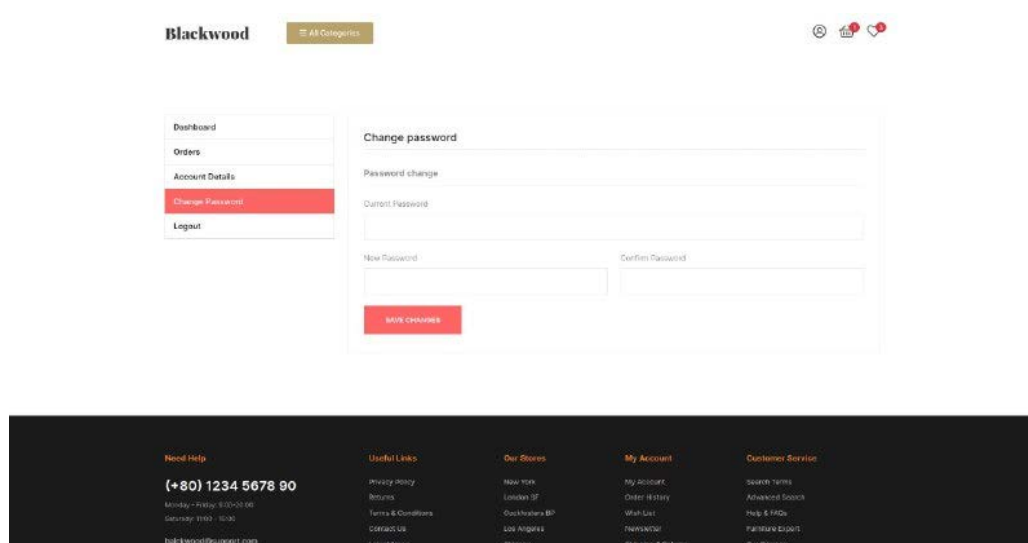
У кабінеті користувача також доступні функції зміни даних користувача (рис.4.17). Користувач може оновити свої персональні дані, такі як ім'я, адреса електронної пошти або контактні дані. Для цього надається відповідна форма, в якій користувач може внести необхідні зміни та зберегти їх.



The screenshot shows the Blackwood user interface. At the top, there's a navigation bar with the Blackwood logo and a 'All Categories' button. Below this is a sidebar menu with options: Dashboard, Orders, Account Details (highlighted in red), Change Password, and Logout. The main content area is titled 'Account Details' and contains a form with the following fields: 'Display Name' (with a placeholder 'John Wilson'), 'Email Address' (with a placeholder 'john.wilson@gmail.com'), and 'Avatar' (with a placeholder 'Profile picture'). There are two radio buttons under 'Avatar': 'Upload new photo' and 'Use default photo'. A red 'SAVE CHANGES' button is at the bottom of the form. The footer contains a 'Need Help' section with a phone number (+80) 1234 5678 90, 'Useful Links' (Privacy Policy, Returns, Terms & Conditions, Contact Us), 'Our Stores' (New York, London SF, Stockholm BP, Los Angeles, Chicago), 'My Account' (My Account, Order History, Wish List, Newsletter, Shipping & Returns), and 'Customer Service' (Search Terms, Advanced Search, Help & FAQs, Partnerships, Our Story).

Рисунок 4.17 – Сторінка зміни даних

Крім того, у кабінеті користувача є можливість змінити пароль (рис.4.18). Користувач може ввести новий пароль у відповідному полі та зберегти зміни. Це забезпечує безпеку облікового запису користувача та дозволяє йому підтримувати актуальні дані для входу.



The screenshot shows the Blackwood user interface with the 'Change password' form. The sidebar menu is the same as in the previous screenshot, but 'Change Password' is now highlighted in red. The main content area is titled 'Change password' and contains a form with the following fields: 'Password change' (placeholder), 'Current Password', 'New Password', and 'Confirm Password'. A red 'SAVE CHANGES' button is at the bottom of the form. The footer is identical to the previous screenshot.

Рисунок 4.18 – Сторінка зміни паролю

Нарешті, у кабінеті користувача є опція виходу з облікового запису. Користувач може натиснути на кнопку "Вихід" або аналогічний елемент, щоб завершити сеанс і вийти зі свого облікового запису.

### **4.3 Графічний дизайн для адміністратора**

Адмін-панель розроблено з урахуванням максимальної зручності та інформативності для забезпечення ефективної роботи адміністратора. Інтерфейс адмін-панелі розроблено таким чином, щоб бути інтуїтивно зрозумілим і зручним у використанні.

В адмін-панелі дані організовано у вигляді таблиць, що полегшує їхній перегляд і керування. Кожна таблиця представляє певну сутність, таку як користувачі, продукти, категорії, бренди тощо. Кожен рядок таблиці містить інформацію про конкретний об'єкт, а стовпці представляють різні властивості цього об'єкта.

Для зручності пошуку і фільтрації даних в адмін-панелі передбачено пошуковик. Адміністратор може ввести ключові слова або фільтри для швидкого пошуку потрібних даних. Це дає змогу економити час і швидко знаходити потрібні об'єкти або інформацію.

Також, для забезпечення зручності роботи з великим обсягом даних, в адмін-панелі використовується пагінація. Дані розбиваються на сторінки, і адміністратор може переходити між сторінками для перегляду та управління даними.

#### **4.3.1 Навігаційна панель**

Для зручності навігації по адмін-панелі використовується навігаційна панель, яка розташована зліва екрана (рис. 4.19). Вона містить посилання на різні розділи адмін-панелі, як-от керування користувачами, продуктами, категоріями,

брендами тощо. Навігаційна панель може бути згорнута або розгорнута, залежно від уподобань адміністратора, що допомагає економити місце на екрані та спрощує навігацію.

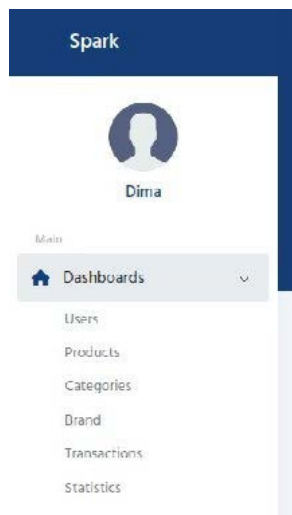


Рисунок 4.19 – Навігаційна панель адміністратора

#### 4.3.2 Сторінка статистики

На сторінці дашборда адмін-панелі міститься статистика (рис. 4.20), яка надає інформацію про реєстрацію користувачів та їхні покупки. Ця статистика допомагає адміністратору отримати загальне уявлення про активність користувачів і успішність продажів. Статистика про реєстрацію користувачів включає такі дані: загальна кількість зареєстрованих користувачів за певний період часу, або ж за весь час.

Статистика про покупки користувачів включає такі дані:

- загальна кількість здійснених замовлень за певний період часу;
- загальна кількість куплених товарів за певний період часу;
- загальну суму виручки від продажів за певний період часу.

Усю цю статистику надають у зручному та наочному форматі, щоб адміністратор міг швидко та легко переглянути статистику.

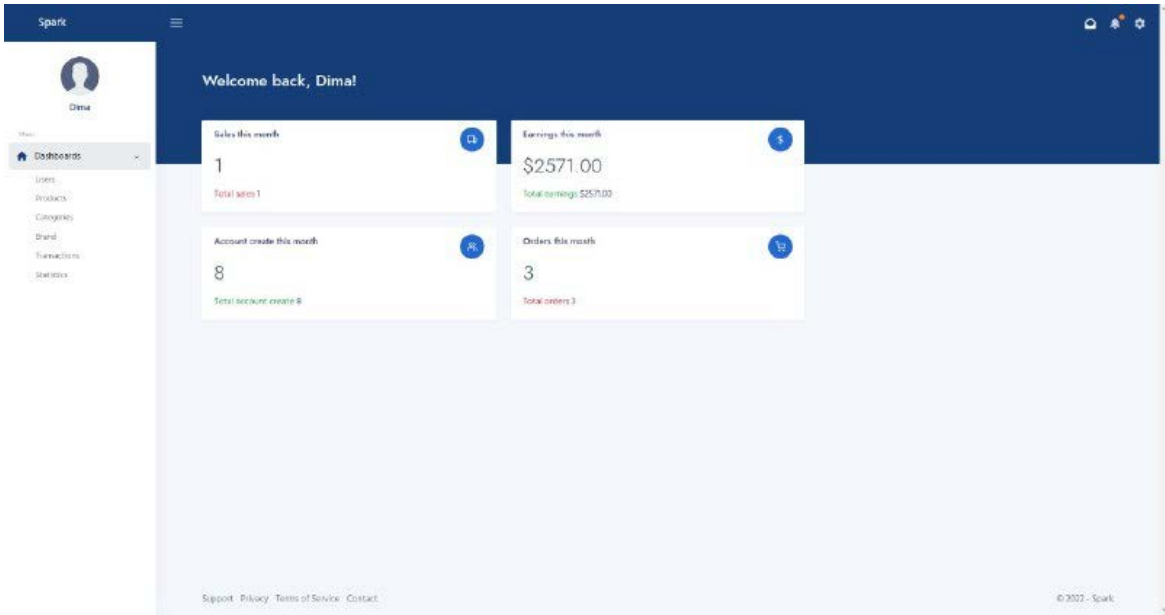


Рисунок 4.20 – Статистика магазину

4.3.3 Сторінка користувачів

На сторінці користувачів в адмін-панелі міститься вся інформація про користувачів (рис. 4.21), яка дає змогу адміністратору взаємодіяти з ними. Кожен користувач представлений у вигляді запису зі своїми персональними даними, такими як ім'я, електронна пошта, дата реєстрації та інші додаткові відомості.

The screenshot shows the 'Users' page in the Spark admin panel. The page has a sidebar with navigation links (Dashboards, Users, Products, Categories, Brand, Transactions, Statistics) and a main area displaying a table of users. The table has columns for Name, Email, Password, Created, Is\_Banned, Edit, Ban, Unban, Login, and Delete. The table lists several users, including Dima, Wayne Meyer, Salvador Giles, Griffin Wiggins, Benjamin Tanner, Abdul Cote, Nina Senior, and Heli Vincent. Each user has a corresponding row with their details and action buttons.

| Name            | Email                      | Password                                                    | Created             | Is_Banned | Edit | Ban | Unban | Login | Delete |
|-----------------|----------------------------|-------------------------------------------------------------|---------------------|-----------|------|-----|-------|-------|--------|
| Dima            | dima.kor22@gmail.com       | \$2y\$10\$2Bqur1B1KqKQ38eW100VZ1LE99nd9BOMC0R6TU1Y5u16AFU   | 2023-05-10 18:00:10 | 1         |      |     |       |       |        |
| Wayne Meyer     | hijvuf@gmailinator.com     | \$2y\$10\$3q9BL5w1wosserdpTndyGd7YKLTU0XpYEuS1keofwub1TTS   | 2023-05-12 16:13:19 | 0         |      |     |       |       |        |
| Salvador Giles  | hghape@gmailinator.com     | \$2y\$10\$ALQ2B6Rc1y15y9RNC1e0X0dvuvvQhZBVLup0BR03pne0AG    | 2023-05-12 16:13:32 | 0         |      |     |       |       |        |
| Griffin Wiggins | kyncog@mailinator.com      | \$2y\$10\$g1ed66hdn1JL17gpcuCH9u10rku1e0KCYwNB2yV1Jmpv1R    | 2022-05-12 16:13:55 | 0         |      |     |       |       |        |
| Benjamin Tanner | piwtfpr@gmailinator.com    | \$2y\$10\$pdWPhnAu51pNH4v01vZgpmV1b1qJZwwaP0FpYh8Aua8MA1wC  | 2023-05-14 15:13:28 | 0         |      |     |       |       |        |
| Abdul Cote      | skkxou@gmailinator.com     | \$2y\$10\$g17W84EDK4RChENB1r18eTMemQ4u1Rb3C3yT1mh04w8A9JTC  | 2023-05-14 15:13:28 | 0         |      |     |       |       |        |
| Nina Senior     | budogmailinator.com        | \$2y\$10\$K27GULee6u1t6Gabi0odvH0gkuf0xVCL020hmpg1p191u2    | 2023-05-14 15:13:29 | 0         |      |     |       |       |        |
| Heli Vincent    | lopapejewa@gmailinator.com | \$2y\$10\$gR031Y5u0EcnLcXb1u1uM1Mc20U0m3e5QW1k72712Q1W7U1p1 | 2023-05-14 15:13:30 | 0         |      |     |       |       |        |

Рисунок 4.21 – Сторінка з користувачами

Можливості взаємодії з користувачами включають.

**Забанити користувача:** Адміністратор може застосувати бан до користувача, що обмежить його доступ до певних функцій або ресурсів веб-додатка. Це може бути корисно у випадках порушення правил або неприпустимої поведінки з боку користувача.

**Розбанити користувача:** Якщо користувача було забанено раніше, адміністратор може зняти з нього обмеження і повернути йому повний доступ до функцій веб-додатку.

**Змінити дані користувача (рис. 4.22).** Адміністратор може вносити зміни в персональні дані користувача, наприклад, виправити помилки в імені або оновити контактну інформацію. Це може знадобитися в разі, якщо користувач звернувся з проханням про внесення змін.

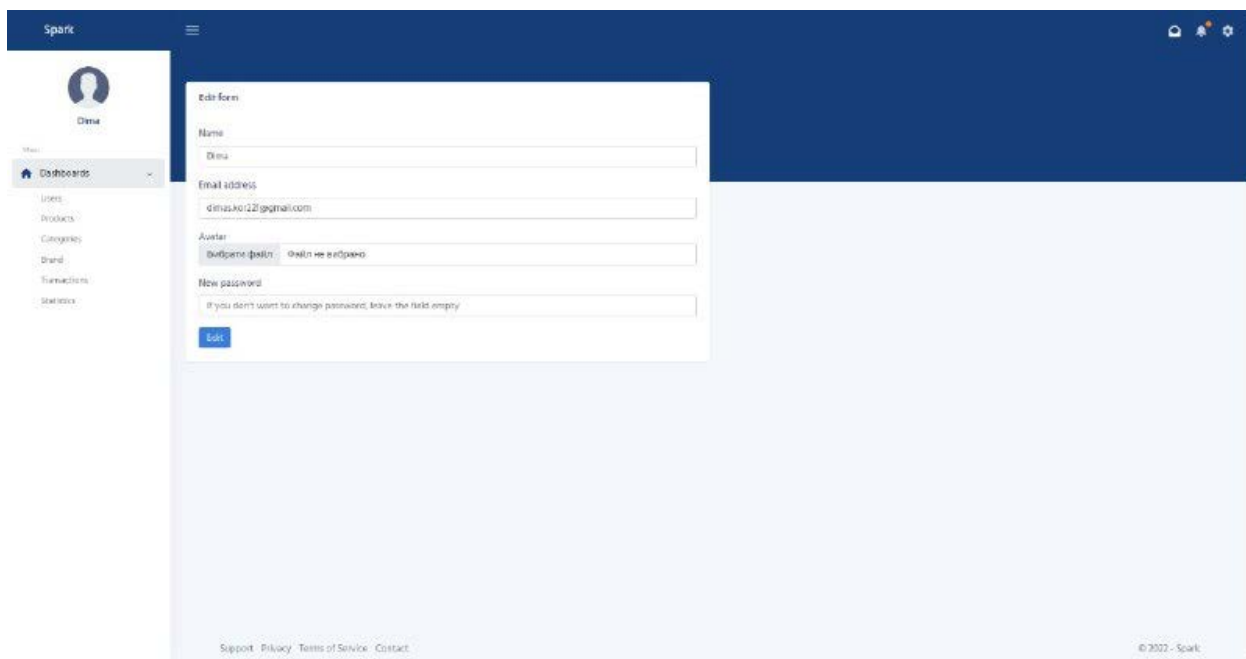


Рисунок 4.22 – Сторінка зміни даних користувача

**Залогінитися під обліковим записом користувача:** У деяких випадках адміністратору може знадобитися увійти в систему від імені користувача для виконання певних дій або вирішення проблем. Ця функція дає змогу

адміністратору тимчасово увійти в акаунт користувача без необхідності знати його пароль.

**Видалити користувача:** Якщо необхідно видалити акаунт користувача, адміністратор може вибрати цю опцію. Зазвичай це робиться тільки за особливих обставин або відповідно до правил і політик веб-додатка.

Взаємодія з користувачами в адмін-панелі дає змогу адміністратору ефективно керувати користувачами та забезпечувати безпеку і відповідність правилам веб-додатка.

### 4.3.4 Сторінка товарів

В адмін-панелі адміністратор має повний контроль над управлінням продуктами (рис. 4.23).

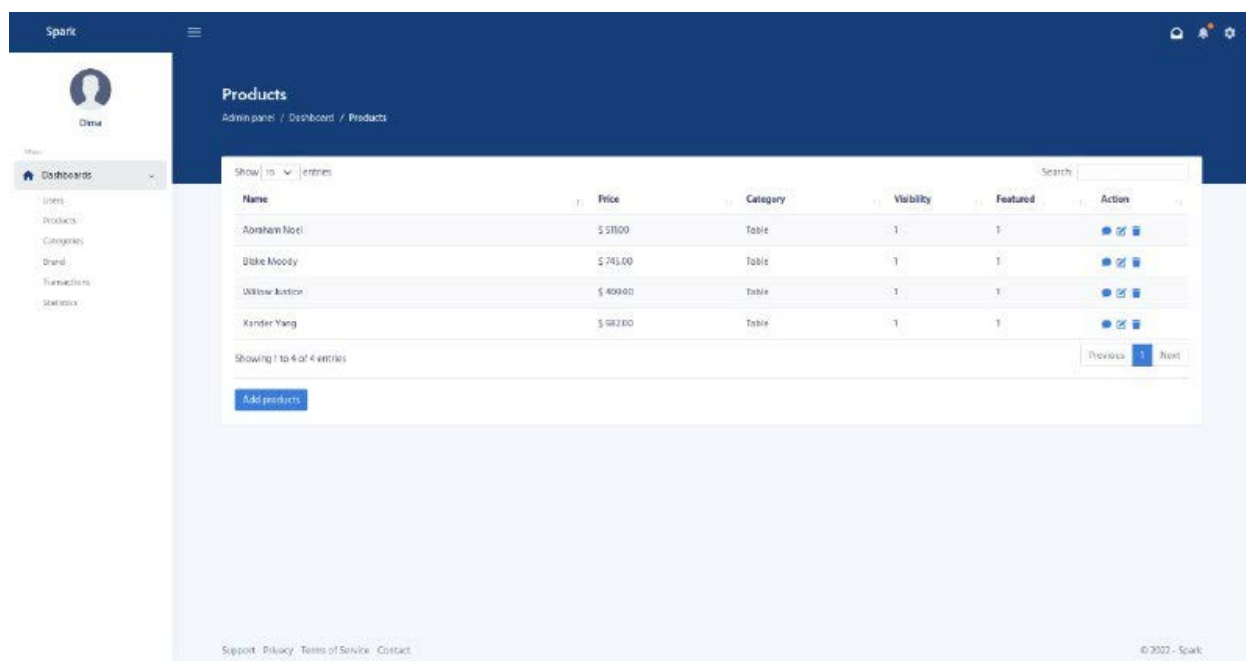


Рисунок 4.23 – Сторінка товарів

Він може виконувати перераховані нижче дії.

**Створення нових продуктів:** Адміністратор може створювати нові продукти, вказуючи всі необхідні деталі, такі як назва, опис, ціна, зображення,

категорія, бренд тощо. Це дає змогу адміністратору розширювати асортимент продуктів і додавати нові позиції в каталог.

Зміна інформації про продукти (рис. 4.24). Адміністратор має можливість змінювати інформацію про продукти. Він може оновлювати назву, опис, ціну, зображення, категорію, бренд та інші характеристики, щоб відобразити зміни або виправити помилки в даних.

The screenshot displays the 'Products edit' interface. On the left, a sidebar lists navigation options: Home, Dashboards, Users, Products, Categories, Brand, Transactions, and Statistics. The main content area is titled 'Products edit' and contains a form with the following fields: Name (with a placeholder 'Apurshen Nicer'), Description (with a placeholder 'Dignissimod id eum'), Price (with a placeholder '1/10'), Category (set to 'Table'), Brand (set to 'NEA'), Weight (with a placeholder 'Ex debet ut totam sa'), Dimensions (with a placeholder 'Anisi la nascunt'), Color (with a placeholder 'Pecunias exercitati'), Manufacturer country (with a placeholder 'Sed quia error quibus'), and a product image (showing a wooden picnic table). Below the image, there are checkboxes for 'Featured' and 'Visible', and an 'Edit' button. The footer includes links for Support, Privacy, Terms of Service, and Contact, along with a copyright notice for Spark.

Рисунок 4.24 – Сторінка зміни товару

Видалення продуктів: Адміністратор може видаляти продукти з каталогу, якщо вони більше не актуальні або не потрібні.

Перегляд і управління коментарями (рис. 4.25). Адміністратор може переглядати коментарі, залишені користувачами до конкретного продукту. Він може видаляти небажані коментарі.

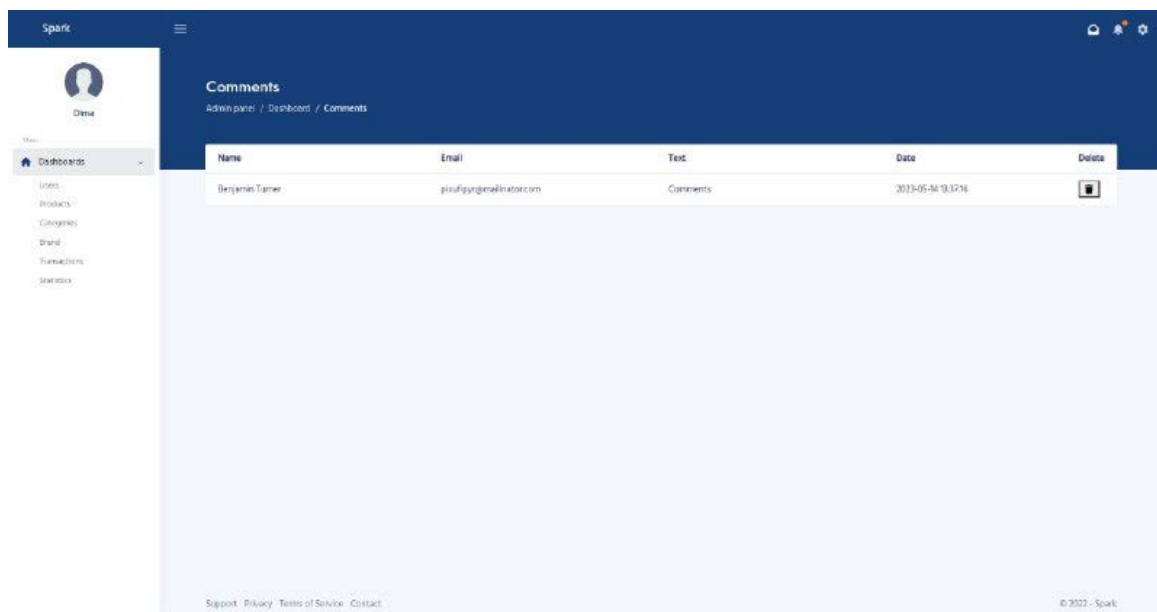


Рисунок 4.25 – Сторінка перегляду коментарів

Перегляд, створення, зміна та видалення продуктів, а також керування коментарями дають змогу адміністратору ефективно керувати каталогом продуктів і підтримувати високу якість та актуальність пропонованих товарів.

#### 4.3.5 Сторінка категорії

В адмін-панелі адміністратор має можливість керувати категоріями продуктів (рис. 4.26). У меню категорій доступні дії які перераховані нижче.

**Додавання категорії:** Адміністратор може створювати нові категорії продуктів, вказуючи назву, опис та інші необхідні деталі. Це дає змогу адміністратору організовувати продукти в логічні групи для зручності навігації користувачів.

**Зміна категорії:** Адміністратор може змінювати інформацію про наявні категорії. Він може оновлювати назву, опис або інші характеристики категорії, щоб відобразити зміни або виправити помилки.

**Видалення категорії:** Адміністратор може видалити категорію продуктів. Замість фактичного видалення, категорія стає неактивною, що означає, що вона

не буде відображатися на сайті і нові продукти не можуть бути додані в цю категорію. Усі пов'язані з цією категорією продукти також стають неактивними.

Відновлення категорії: Якщо раніше видалена категорія стала необхідна знову, адміністратор може відновити її, зробивши її знову активною. Це дає змогу відновити категорію разом з її пов'язаними продуктами.

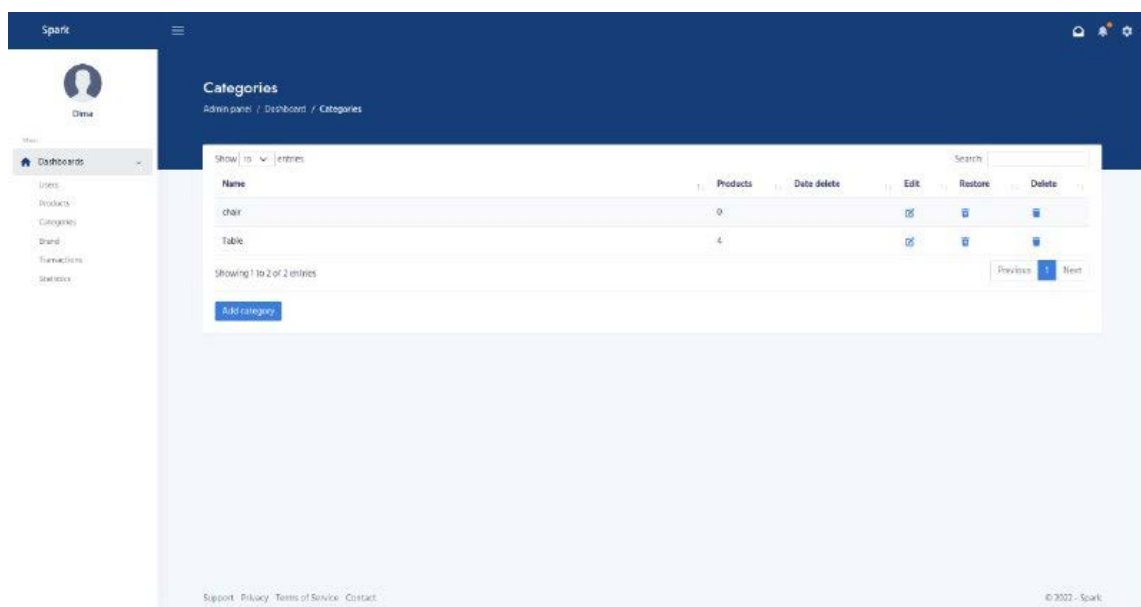


Рисунок 4.26 – Сторінка перегляду категорії

Керування категоріями продуктів дає змогу адміністратору ефективно організовувати та керувати структурою продуктів веб-додатка, а також надавати користувачам зручні способи навігації та фільтрації продуктів.

#### 4.3.6 Сторінка брендів

В адмін-панелі доступна вкладка для роботи з брендами (рис. 4.27). У цій вкладці адміністратор може виконувати дії наведені нижче.

Створення бренду: Адміністратор може створювати нові бренди, вказуючи назву, опис та інші деталі. Створення брендів дає змогу адміністратору класифікувати продукти за виробниками та полегшує навігацію і пошук для користувачів.

**Зміна бренду:** Адміністратор може змінювати інформацію про наявні бренди. Він може оновлювати назву, опис або інші характеристики бренду, щоб відобразити зміни або внести коригування.

**Видалення бренду:** Адміністратор може видалити бренд, якщо він більше не потрібен. При видаленні бренду, пов'язані з ним продукти все ще залишаються доступними на сайті, але інформація про бренд не відображатиметься. Це може бути корисно, якщо бренд припинив свою діяльність або якщо потрібно приховати деякі бренди від користувачів.

**Приховування бренду:** Адміністратор може зробити бренд невідображуваним для користувачів. Це означає, що інформацію про бренд не буде видно на сайті, але пов'язані з ним продукти залишаться доступними. Це може бути корисно, якщо бренд тимчасово неактивний або потрібно приховати його через будь-які причини.

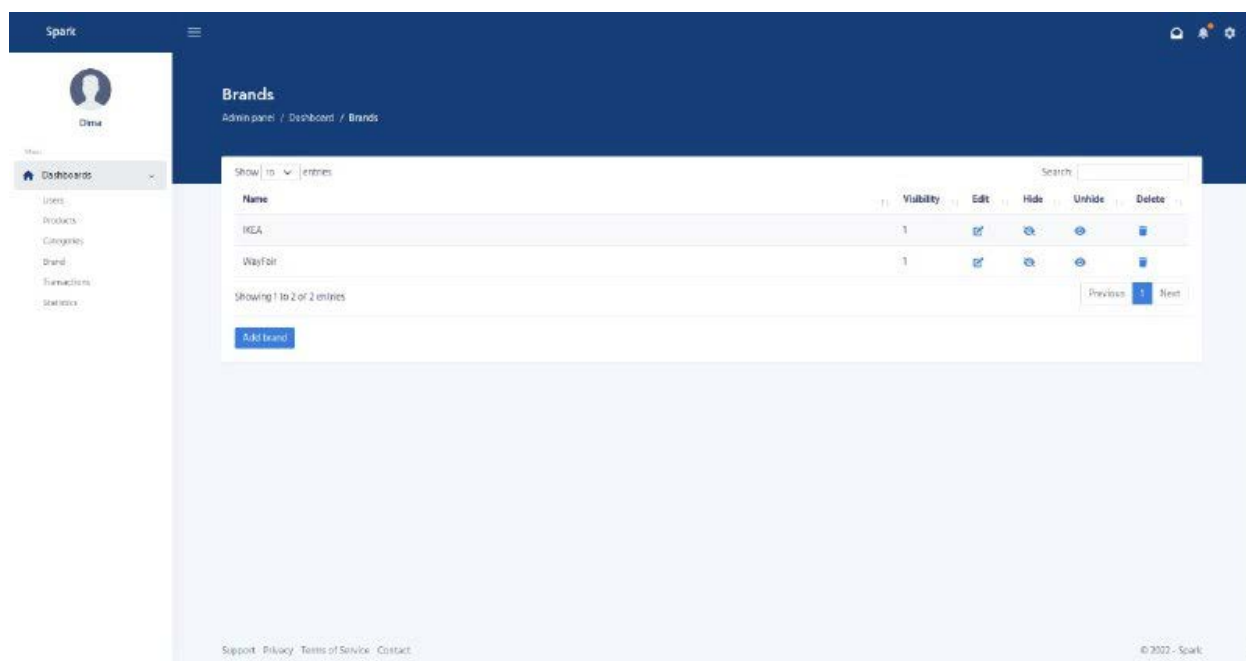


Рисунок 4.27 – Сторінка перегляду брендів

Керування брендами в адмін-панелі забезпечує гнучкість і контроль над класифікацією продуктів за виробниками, а також дає змогу адміністратору

легко додавати, змінювати та видаляти бренди відповідно до потреб проєкту та вимог користувачів.

### 4.3.7 Сторінка транзакції

В адмін-панелі доступна вкладка "Транзакції", яка містить інформацію про покупки користувачів і платежі (рис. 4.28). У цій вкладці адміністратор може отримати таку інформацію та виконати відповідні дії.

Загальна інформація про транзакції: Адміністратор може переглянути список усіх транзакцій, включно з датою і часом купівлі, сумою платежу і статусом платежу (успішний або неуспішний).

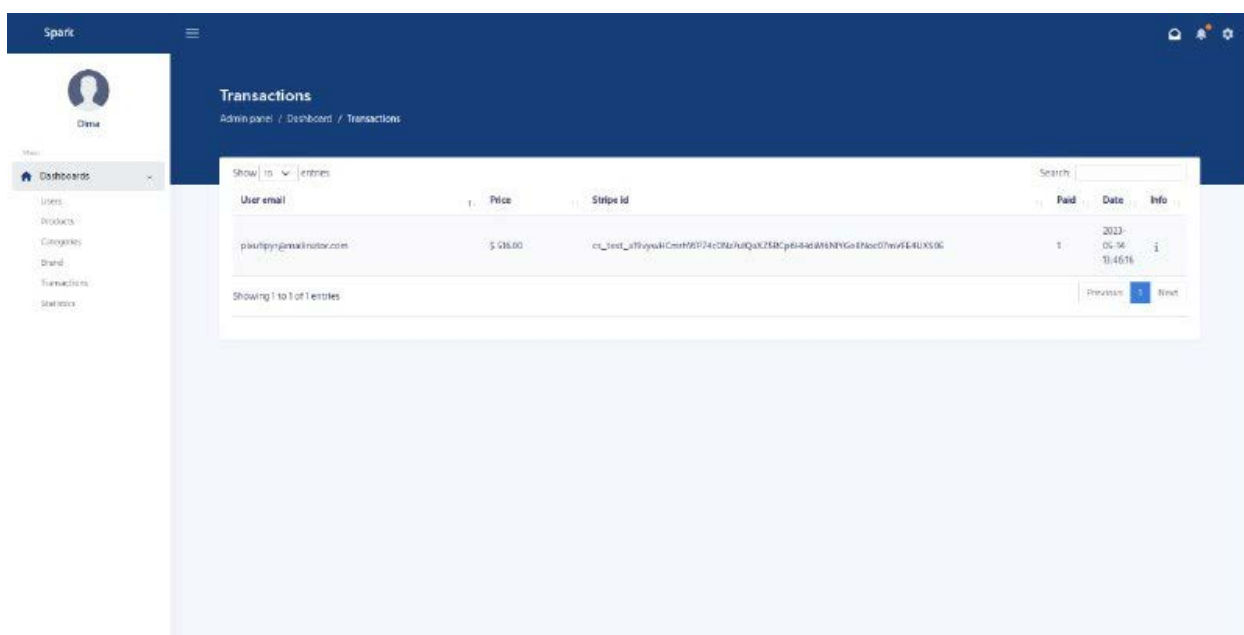
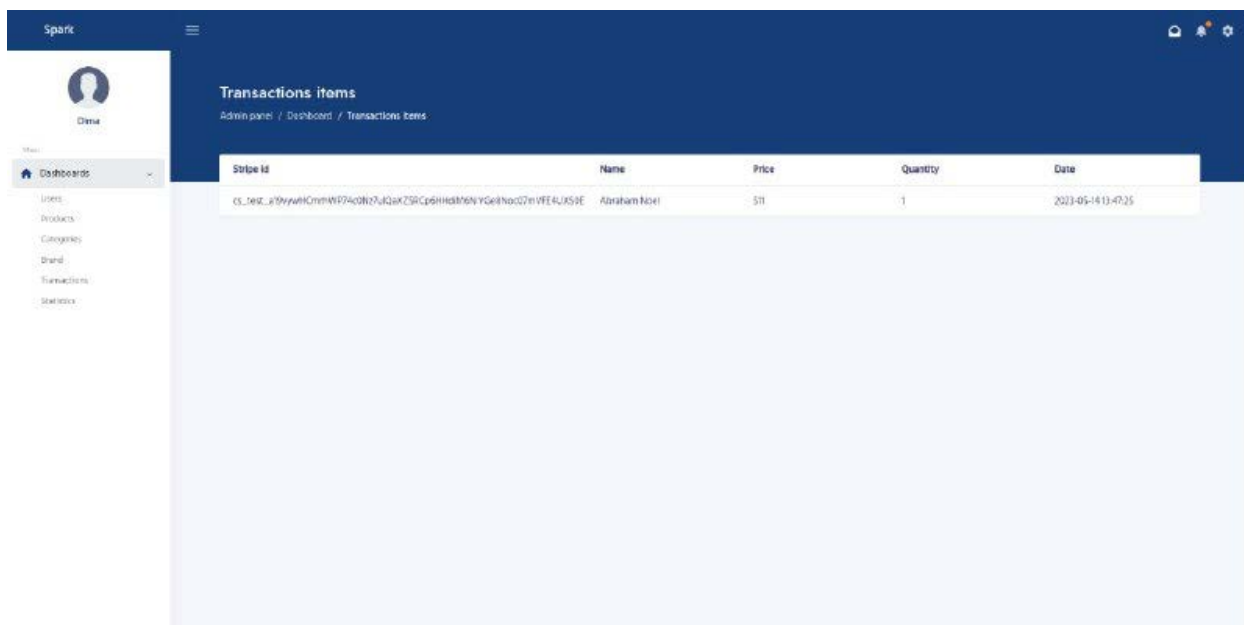


Рисунок 4.28 – Сторінка транзакції

Детальна інформація про платіж (рис. 4.29). При виборі певної транзакції адміністратор може переглянути детальнішу інформацію про платіж, включно зі списком куплених товарів, їхньою кількістю та ціною. Це дає змогу адміністратору отримати повне уявлення про покупку користувача.



| Stripe Id                                                        | Name         | Price | Quantity | Date                |
|------------------------------------------------------------------|--------------|-------|----------|---------------------|
| cs_test_a39yywhCrmnWtF74c0tZ7uQxZ5RcPsnHdthnyYGeIhacQ2nVFE4UXS9E | Abraham Noel | \$11  | 1        | 2021-05-14 13:49:25 |

Рисунок 4.29 – Сторінка детальної інформації транзакцій

Інформація про транзакції дає змогу адміністратору відстежувати й керувати платежами, а також отримувати докладні відомості про кожну покупку, включно з інформацією про товари, їхню кількість і ціну. Це допомагає забезпечити прозорість і контроль над процесом оплати веб-додатка.

#### 4.4 Висновок

Графічна складова сайту відображає сучасні тренди веб-дизайну. Дизайн сайту є мінімалістичним і привабливим, з акцентом на простоті та функціональності. Він добре адаптується до різних пристроїв і розмірів екранів завдяки адаптивному підходу. Інтерактивні елементи та мікроінтеракції додають динамічності та привабливості користувацькому досвіду.

Навігація сайтом інтуїтивно зрозуміла та легко доступна. Головне меню і бічна панель адміністратора забезпечують зручний доступ до всіх розділів і функцій сайту. Користувачі можуть легко знайти потрібні категорії, бренди та товари за допомогою зручної системи фільтрів.

Колірна палітра сайту є збалансованою та естетично приємною. Вона допомагає виділити важливі елементи та створює гармонійний загальний вигляд сторінок.

Загальний дизайн сайту, включно зі шрифтами, іконками та компонентами, спроектовано з урахуванням зручності використання та ефективності взаємодії з користувачем. Усі сторінки мають логічну структуру і зрозумілі користувацькі інтерфейси, що дає змогу користувачам легко орієнтуватися і взаємодіяти з сайтом.

Загалом, графічна складова сайту створює приємне візуальне враження і сприяє зручності використання, що є важливим аспектом для підвищення привабливості та ефективності веб-додатку.

## 5 ПІДКЛЮЧЕННЯ СТОРОННЬОГО API, ТЕСТУВАННЯ СИСТЕМИ

### 5.1 Підключення API Stripe

Stripe є одним із найзручніших і найлегших у використанні платіжних API. Він надає розробникам простий та інтуїтивно зрозумілий інтерфейс, який дає змогу швидко та легко інтегрувати платіжні функції у веб-додатки. Установку можна виконати за кілька кроків.

Необхідно зареєструватися на офіційному сайті Stripe і створити акаунт, де необхідно буде заповнити даними про себе

Після успішної реєстрації буде отримано доступ до ключів API, які необхідні для взаємодії з платіжною системою Stripe. Ці ключі забезпечують безпечний зв'язок між вашим веб-додатком і Stripe.

Встановити відповідний SDK Stripe . У нашому випадку ми можемо встановити офіційний Laravel Cashier, який надає зручні методи для роботи зі Stripe.

Конфігурація ключів API: У вашому проєкті необхідно налаштувати ключі API Stripe, які були отримані на попередньому етапі. Це робиться у файлі конфігурації нашого застосунку, де вказати публічний і приватний ключ API.

Створення платіжних форм для обробки платежів. Ми можемо створити відповідні форми або інтерфейси для введення даних платежів, таких як інформація про картку, сума платежу та інша необхідна інформація.

Обробка відповідей і управління транзакціями: Після обробки платежів ми отримаємо відповіді від Stripe, які містять інформацію про транзакції. Вони будуть використані для оновлення статусу платежів у нашій базі даних або для прийняття додаткових дій, пов'язаних із транзакціями [15].

### 5.2 Підключення API Mailgun

Mailgun використовується для надсилання повідомлень про різні дії та події в інтернет-магазині, такі як реєстрація користувачів, купівля товарів,

оплата замовлень та інші важливі операції. Надсилання електронних сповіщень є важливим аспектом взаємодії з користувачами та забезпечує своєчасну і достовірну інформацію про вчинені дії. Для налаштування Mailgun на нашому проєкті ми виконали такі кроки.

Створили обліковий запис Mailgun: Ми зареєстрували обліковий запис на офіційному сайті Mailgun і вибрали відповідний план, що відповідає нашим вимогам.

Отримали домен Mailgun: Ми зареєстрували домен у Mailgun, який буде використовуватися для надсилання та отримання електронної пошти. Mailgun надав нам доменне ім'я, яке ми додали в конфігурацію нашого веб-додатка.

Підтвердили домен: Щоб забезпечити безпечну доставку електронної пошти, Mailgun зажадав підтвердження нашого домену. Ми дотримувалися інструкцій Mailgun і додали спеціальні DNS-записи в налаштування нашого домену.

Налаштували SMTP-сервер: Ми налаштували SMTP-сервер Mailgun у нашому веб-додатку, вказавши відповідні облікові дані, такі як SMTP-хост, порт, ім'я користувача та пароль. Це дозволило нашому додатку надсилати електронні листи через Mailgun.

Інтегрували API Mailgun: Ми також інтегрували API Mailgun у наш веб-додаток, щоб розширити його функціональність у сфері електронної пошти. Ми використовували API для надсилання автоматичних сповіщень, відстеження доставки листів та інших завдань, які надає Mailgun API.

Провели тестування і налагодження: Після налаштування Mailgun ми провели тестування, щоб переконатися, що надсилання та отримання електронної пошти працюють коректно. Ми також використовували інструменти налагодження, що надаються Mailgun, для перевірки статусу доставки листів і виявлення можливих проблем.

У результаті цих дій Mailgun був успішно налаштований для роботи на нашому проєкті, і ми можемо використовувати його для надсилання повідомлень при реєстрації користувача або успішній покупці (рис.5.1) [16].

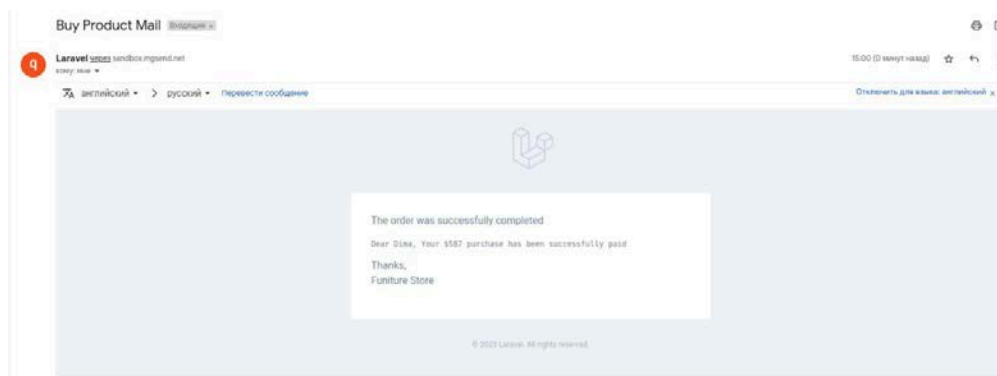


Рисунок 5.1 – Повідомлення на пошту при покупці

### 5.3 Тестування системи

Одним із фінальних етапів розробки веб-додатку є тестування та налагодження системи. Вони дають змогу виявити та виправити помилки, поліпшити продуктивність і забезпечити стабільну роботу додатка. Ці процеси допомагають переконатися, що всі функціональні можливості працюють належним чином, взаємодія з базою даних та іншими компонентами системи відбувається коректно, а користувацький досвід є позитивним. Тестування допомагає також перевірити систему на безпеку та захист від зовнішніх загроз. Важливо приділяти належну увагу тестуванню та налагодженню, щоб забезпечити якість і надійність веб-додатка перед його впровадженням і запуском [17].

#### 5.3.1 Функціональні тестування

Функціональне тестування є одним з ключових етапів випробування веб-додатків. Його основна мета - перевірити, чи працюють основні функції додатку так, як очікується.

Під час функціонального тестування проводяться наступні дії.

Перевірка коректності виконання основних функцій: Тестування спрямоване на перевірку того, чи працюють основні функції додатку згідно з

очікуваннями. Наприклад, перевірка реєстрації нового користувача, авторизації, додавання товарів в кошик, оформлення замовлення тощо.

Валідація введених даних: Перевірка коректності введених даних у формах. Наприклад, перевірка валідності електронної пошти, правильного формату номера телефону або відповідності паролю вимогам безпеки.

Робота з базою даних: Перевірка правильності збереження даних у базі даних. Включає перевірку додавання, зміни та видалення записів, а також перевірку коректного відображення даних на відповідних сторінках.

Взаємодія з іншими компонентами: Перевірка взаємодії з іншими системами або компонентами, такими як stripe.

Обробка помилок: Перевірка обробки помилок і виведення відповідних повідомлень користувачеві в разі некоректного виконання дій.

Функціональне тестування проводиться з використанням тестових сценаріїв, які охоплюють різні варіанти використання додатку. Результати тестування допомагають виявити помилки та недоліки.

### **5.3.2 Тестування безпеки**

У процесі розробки веб-додатка було проведено тестування безпеки з метою виявлення та усунення можливих вразливостей. Це включало тестування на наявність SQL-ін'єкцій, XSS, CSRF та інших поширених атак. У результаті перевірки було переконано, що застосунок надійно захищений від таких атак і не схильний до можливих зломів.

Також було здійснено перевірку безпеки передавання даних, де було переконливо підтверджено, що дані, які передаються між клієнтом і сервером, захищені з використанням надійних протоколів шифрування, таких як SSL/TLS. Критична інформація, така як паролі та платіжні дані, була надійно зашифрована, щоб запобігти можливому перехопленню зловмисниками.

Також було протестовано механізми автентифікації та авторизації в застосунку, щоб переконатися в їхній коректній роботі та надійності. Лише

правильно аутентифікованим користувачам надано доступ до відповідних функцій і ресурсів, а несанкціонованому доступу даних або дій запобігли.

### **5.3.3 Тестування сумісності**

Було проведено тестування сумісності для забезпечення коректної роботи веб-додатка на різних платформах, браузерях і пристроях. Застосунок було протестовано на популярних браузерах, як Chrome, Firefox, Safari та Edge, щоб переконатися, що відображається і функціонує належним чином на кожному.

Також було проведено тестування на різних пристроях, включно з настільними комп'ютерами, планшетами та смартфонами. Метою цього тестування було переконатися, що застосунок адаптивний і чуйний, а його інтерфейс правильно відображається і забезпечує зручну навігацію на різних пристроях з різними розмірами екранів.

У процесі тестування сумісності було виявлено та виправлено можливі проблеми, пов'язані з відображенням і функціональністю застосунку на різних платформах і пристроях. Таким чином, додаток гарантує однаково гарний користувацький досвід незалежно від обраного браузера або пристрою.

Тестування сумісності відіграє важливу роль у забезпеченні доступності та зручності використання веб-додатку для широкого кола користувачів, що дає змогу досягти максимального охоплення цільової аудиторії та забезпечити його успішне функціонування на різних платформах і пристроях.

### **5.3.4 Відлагодження системи**

Для перевірки нашого веб-сайту ми використовували різні інструменти та методи налагодження. Одним з основних інструментів були журнали помилок, які надавали інформацію про виниклі помилки, повідомлення про винятки та інші корисні дані. Аналізуючи журнали помилок, ми могли зрозуміти причину проблеми і вжити відповідних заходів для її виправлення.

Консоль розробника браузера також була важливим інструментом налагодження. Вона надавала набір інструментів і функцій, що давали нам змогу відстежувати помилки JavaScript, аналізувати мережевий трафік, перевіряти стилі та багато іншого. За допомогою консолі розробника ми могли ідентифікувати й аналізувати проблемні місця в коді програми.

Дебагери ще один інструмент, який використовували для налагодження. Вони давали змогу виконати код покроково й аналізувати його стан у процесі виконання. За допомогою дебагерів можна досліджувати внутрішню логіку і потік виконання програми, що допомогло виявляти і виправляти помилки.

Налагодження відіграє важливу роль у розробці веб-додатків, оскільки допомагає забезпечити якість і стабільність роботи програми. Завдяки використанню різних інструментів і методів налагодження, ми могли ефективно ідентифікувати й усувати проблеми, що в кінцевому підсумку сприяло підвищенню надійності та поліпшенню користувацького досвіду нашого веб-сайту.

## **5.4 Висновок**

Тестування системи та інтеграція сторонніх API є невід'ємною частиною розробки веб-додатків. Вони допомагають забезпечити надійність, безпеку і функціональність застосунку, а також гарантувати правильну взаємодію із зовнішніми сервісами. Функціональне тестування перевіряє працездатність основних функцій застосунку, тестування безпеки виявляє та виправляє вразливості, тестування сумісності гарантує коректну роботу на різних платформах, а тестування продуктивності оцінює ефективність застосунку за різних навантажень. Інтеграція сторонніх API, як-от Stripe і Mailgun, вимагає налаштування і тестування для правильної роботи функціональності, пов'язаної з оплатою і надсиленням повідомлень. Усі ці заходи допомагають створити стабільний і функціональний веб-додаток, що відповідає вимогам користувачів.

## ВИСНОВКИ

Під час розроблення веб-додатка для магазину меблів було розглянуто та вивчено інші аналогічні веб-додатки, що дало змогу отримати уявлення про наявні рішення в цій галузі. Це допомогло визначити вимоги до проектування нашого веб-додатка.

Було розглянуто та обрано оптимальні варіанти архітектури, які забезпечать ефективну та надійну роботу нашого веб-додатка. Це дало змогу створити надійну та ефективну систему для продавця, яка дає змогу керувати своїми товарами, а для покупця - легко та зручно робити покупк

Було обрано сучасний дизайн і розроблено зручний інтерфейс, що враховують тренди веб-дизайну і забезпечують зручність використання для користувачів. Це включало створення функціональних сторінок, зручної навігації, а також можливість пошуку, перегляду та аналізу товарів.

Для розширення функціональності та підвищення зручності використання були інтегровані сторонні сервіси, як-от Stripe для обробки платежів і Mailgun для надсилання повідомлень. Це забезпечує безпеку, надійність і комфортну роботу користувачів веб-додатку.

Після завершення розробки було проведено тестування всієї системи, включно з функціональним тестуванням, тестуванням безпеки та тестуванням сумісності. Це дало змогу виявити й виправити можливі помилки та проблеми, а також переконатися в правильній роботі застосунку в різних сценаріях використання.

У результаті розробки було створено веб-додаток для магазину меблів, що дає змогу продавцю ефективно керувати своїми товарами, а покупцеві - зручно та безпечно здійснювати покупки.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Фреймворки у веб-розробці — що це, які існують і для чого потрібні [Електрон. ресурс]. — Режим доступу: <https://highload.today/uk/frejmworki-u-veb-rozrobtsi-shho-tse-yaki-isnuyut-i-dlya-chogo-potribni/>.
2. Що таке CMS сайту [Електрон. ресурс]. — Режим доступу: <https://hostiq.ua/wiki/ukr/cms/>.
3. WordPress: Publish your passion [Electronic source]. — Access mode: <https://wordpress.org/>.
4. Wix — Create a website without limits [Electronic source]. — Access mode: <https://www.wix.com/>.
5. Конструктор сайту [Електрон. ресурс]. — Режим доступу: <https://hostiq.ua/wiki/ukr/about-sitebuilder/>.
6. Joomla — The flexible platfrom empowering website creators [Electronic source]. — Access mode: <https://www.joomla.org/>.
7. Laravel — Documentation [Electronic source]. — Access mode: <https://laravel.com/docs/10.x>.
8. Переваги Laravel [Електрон. ресурс]. — Режим доступу: <https://wezom.com.ua/blog/17-preimuschestv-ispolzovaniya-laravel-v-it-industrii>.
9. Основні переваги субд MySQL [Електрон. ресурс]. — Режим доступу: <https://studfile.net/preview/5607354/page:3/>.
10. Stripe — Financial structure for the internet [Electronic source]. — Access mode: <https://stripe.com/>.
11. Paypal [Електрон. ресурс]. — Режим доступу: <https://www.paypal.com/ua/home>.
12. Braintree [Electronic source]. — Access mode: <https://www.braintreepayments.com/>.
13. Переваги та недоліки дистрибутивів на ядрі Linux [Електрон. ресурс]. — Режим доступу: <https://uk.soringpcrepair.com/pros-and-cons-of-linux/>.
14. Прогноз трендів у графічному дизайні на 2023 рік [Електрон. ресурс].

— Режим доступу: <https://www.the-village.com.ua/village/culture/culture-news/336665-animekor-i-estetika-militari-depositphotos-dali-prognoz-trendiv-u-grafichnomu-dizayni-na-2023-rik>.

15. How to Integrate Stripe Connect with Laravel [Electronic source]. – Access mode: <https://www.ultimateakash.com/blog-details/IixTKGAKYAo=/How-to-Integrate-Stripe-Connect-with-Laravel-2022>.

16. Build Laravel 9.x email authentication with Mailgun [Electronic source]. – Access mode: <https://www.mailgun.com/blog/email/build-laravel-email-authentication-mailgun-digital-ocean/>.

17. Основні види та цілі тестування [Електрон. ресурс]. — Режим доступу: <https://voll.com.ua/uk/blog/kak-testirovat-veb-sajt-osnovnye-vidy-i-celi-testirovaniya>.

**ДОДАТОК А**  
**Текст програми**

## A.1 BrandController.php

```
<?php
```

```
namespace App\Http\Controllers\AdminController;
```

```
use App\Http\Controllers\Controller;
```

```
use App\Http\Requests>CreateBrandRequest;
```

```
use App\Models\Brand;
```

```
class BrandController extends Controller
```

```
{
```

```
    public function index()
```

```
    {
```

```
        $brands = Brand::all();
```

```
        return view('admin.brands.index', ['brands' => $brands]);
```

```
    }
```

```
    public function create()
```

```
    {
```

```
        return view('admin.brands.create');
```

```
    }
```

```
    public function store(CreateBrandRequest $request)
```

```
    {
```

```
        Brand::firstOrCreate(
```

```
            ['name' => $request->name],
```

```
        );
```

```
        return redirect()->route('admin.brands.index')->with('success', 'Brand created successfully.');
```

```
    }
```

```
    public function edit($id)
```

```
    {
```

```
        $brand = Brand::find($id);
```

```
        return view('admin.brands.edit', ['brand' => $brand]);
```

```
    }
```

```
    public function update(CreateBrandRequest $request, $id)
```

```
    {
```

```
        Brand::where('id', $id)
```

```
            ->update(['name' => $request->name]);
```

```
        return redirect()->route('admin.brands.index')->with('success', 'Brand updated successfully.');
```

```
    }
```

```
    public function destroy($id)
```

```
    {
```

```
        Brand::where('id', $id)->delete();
```

```
        return redirect()->route('admin.brands.index')->with('success', 'Brand deleted successfully.');
```

```
    }
```

```
    public function hide($id)
```

```

{
    Brand::where('id', $id)
        ->update(['visibility' => false]);
    return redirect()->route('admin.brands.index')->with('success', 'Brand hided successfully.');
```

```

}

public function unhide($id)
{
    Brand::where('id', $id)
        ->update(['visibility' => true]);
    return redirect()->route('admin.brands.index')->with('success', 'Brand unhided successfully.');
```

```

}
}

```

## A.2 CategoryController.php

```
<?php
```

```
namespace App\Http\Controllers\AdminController;
```

```
use App\Http\Controllers\Controller;
use App\Http\Requests\CreateCategoryRequest;
use App\Http\Requests\EditCategoryRequest;
use App\Models\Category;
use Illuminate\Support\Facades\Storage;
```

```
class CategoryController extends Controller
{
```

```
    public function index()
    {
        $categories = Category::withTrashed()->get();
        return view('admin.categories.index', ['categories' => $categories]);
    }
```

```
    public function create()
    {
        return view('admin.categories.create');
    }
```

```
    public function store(CreateCategoryRequest $request)
    {
        $category = Category::firstOrCreate(
            ['name' => $request->name],
        );
        Storage::disk('local')->makeDirectory('public/categories/'.$category->name);
        $request->file('image')->storeAs('public/categories/'.$category->name, 'image.jpg');
        return redirect()->route('admin.categories.index')->with('success', 'Category created successfully.');
```

```
    }
```

```
    public function edit($id)
```

```

{
    $category = Category::find($id);
    return view('admin.categories.edit', ['category' => $category]);
}

public function update(EditCategoryRequest $request, $id)
{
    $category = Category::find($id);
    Category::where('id', $id)
        ->update(['name' => $request->name]);
    if ($request->image != "") {
        Storage::delete('public/categories/'.$category->name.'/image.jpg');
        $request->file('image')->storeAs('public/categories/'.$request->name, 'image.jpg');
    }
    return redirect()->route('admin.categories.index')->with('success', 'Category updated successfully.');
```

```

    }

    public function destroy($id)
    {
        $category = Category::find($id);
        Category::where('name', $category->name)->delete();
        Storage::deleteDirectory('public/categories/'.$id);
        return redirect()->route('admin.categories.index')->with('success', 'Category deleted successfully.');
```

```

    }

    public function restore($id)
    {
        Category::withTrashed()->find($id)->restore();
        return redirect()->route('admin.categories.index')->with('success', 'Category restored successfully.');
```

```

    }
}

```

### A.3 CommentController.php

```

<?php

namespace App\Http\Controllers\AdminController;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use App\Models\Comment;
use App\Models\Product;
use Illuminate\Support\Facades\Auth;
class CommentController extends Controller
{
    public function index($product_id)
    {
        $comments = Comment::where('product_id', $product_id)->get();
        return view('admin.comments.index', ['comments' => $comments]);
    }
}

```

```

public function destroy($id)
{
    Comment::where('id', $id)->delete();
}

public function store(Request $request)
{
    Comment::Create(
        ['product_id' => $request->product_id, 'author' => Auth::user()->email, 'name' =>
Auth::user()->name, 'text' => $request->comments]
    );

    $product = Product::where('id', $request->product_id)->firstOrFail();
    return redirect()->route('products.show', ['product' => $product->name])->with('success',
'Comment added successfully.');
```

#### A.4 ProductController.php

```

<?php
namespace App\Http\Controllers\AdminController;

use App\Http\Controllers\Controller;
use App\Http\Requests>CreateProductRequest;
use App\Http\Requests>EditProductRequest;
use App\Models\Brand;
use App\Models\Category;
use App\Models\Product;
use Illuminate\Support\Facades\Storage;
use Illuminate\Http\Request;

class ProductController extends Controller
{
    public function index()
    {
        $products = Product::all();
        return view('admin.products.index', ['products' => $products]);
    }
    public function edit($id)
    {
        $brands = Brand::all();
        $categories = Category::all();
        $product = Product::find($id);
        return view('admin.products.edit', ['product' => $product, 'categories' => $categories, 'brands' =>
$brands]);
    }
    public function update(EditProductRequest $request, $id)
    {
        Product::where('id', $id)
            ->update(['name' => $request->name, 'featured' => (bool)$request->featured, 'description' =>
$request->description, 'price' => $request->price,
```

```

        'weight' => $request->weight, 'dimensions' => $request->dimensions, 'color' => $request-
>color, 'country' => $request->country,
        'category' => $request->category, 'visibility' => (bool)$request->visibility]);
    if($request->links != null) {
        Storage::deleteDirectory('public/products/'.$id);
        Storage::disk('local')->makeDirectory('public/products/'.$id);
        for ($i = 1; $i <= count($request->links); $i++) {
            $request->file('links')['link'.($i-1)]->storeAs('public/products/'.$id, $i.'.jpg');
        }
    }
    return redirect()->route('admin.products.index')->with('success', 'Product updated
successfully.');
```

```

    }
    public function destroy(Request $request, $id)
    {
        Product::where('id', $id)->delete();
        Storage::deleteDirectory('public/products/'.$id);
        return redirect()->route('admin.products.index')->with('success', 'Product deleted
successfully.');
```

```

    }
    public function create()
    {
        $categories = Category::all();
        $brands = Brand::all();
        return view('admin.products.create', ['categories' => $categories, 'brands' => $brands]);
    }

    public function store(CreateProductRequest $request)
    {
        $product = Product::firstOrCreate(
            ['name' => $request->name],
            ['featured' => (bool)$request->featured, 'description' => $request->description, 'price' =>
$request->price, 'weight' => $request->weight,
            'dimensions' => $request->dimensions, 'color' => $request->color, 'country' => $request-
>country,
            'category' => $request->category, 'visibility' => (bool)$request->visibility, 'image' => 'null',
'brand' => $request->brand]
        );
        for ($i = 1; $i <= count($request->links); $i++) {
            Storage::disk('local')->makeDirectory('public/products/'.$product->id);
            $request->file('links')['link'.($i-1)]->storeAs('public/products/'.$product->id, $i.'.jpg');
        }
        return redirect()->route('admin.products.index')->with('success', 'Product created
successfully.');
```

```

    }
}

```

## A.5 TransactionsController.php

```

<?php
namespace App\Http\Controllers\AdminController;

```

```

use App\Http\Controllers\Controller;
use App\Models\Product;
use App\Models\Transaction;
use App\Models\Transactions_item;

class TransactionsController extends Controller
{
    public function index()
    {
        Transaction::where('paid', false)->delete();
        $transactions = Transaction::all()->all();
        return view('admin.transactions.index', ['transactions' => $transactions]);
    }
    public function show($stripe_id)
    {
        $transactions_item = Transactions_item::where('stripe_id', $stripe_id)->get();
        $products_all = Product::all()->where('visibility', true);
        return view('admin.transactions.show', ['transactions_item' => $transactions_item, 'products_all'
=> $products_all]);
    }
}

```

## A.6 UserController.php

```
<?php
```

```

namespace App\Http\Controllers\AdminController;

use App\Http\Controllers\Controller;
use App\Http\Requests>EditUserRequest;
use App\Models\User;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Storage;
use Illuminate\Support\Facades\Hash;
class UserController extends Controller
{
    public function index()
    {
        $users = User::all();
        return view('admin.users.index', ['users' => $users]);
    }

    public function edit($id)
    {
        $user = User::find($id);
        return view('admin.users.edit', ['user' => $user]);
    }

    public function update(EditUserRequest $request, $id)
    {
        if ($request->password == "") {
            User::where('id', $id)

```

```

        ->update(['name' => $request->name, 'email' => $request->email]);
    }
    else {
        User::where('id', $id)
            ->update(['name' => $request->name, 'email' => $request->email, 'password' =>
Hash::make($request->password)]);
    }
    if($request->image != "") {
        Storage::disk('local')->makeDirectory('public/users/'.$id);
        $request->file('image')->storeAs('public/users/'.$id, 'avatar.jpg');
    }
    return redirect()->route('admin.users.index')->with('success', 'User updated successfully.');
```

```

    }
    public function destroy($id)
    {
        User::where('id', $id)->delete();
        return redirect()->route('admin.users.index')->with('success', 'User deleted successfully.');
```

```

    }
    public function login($id)
    {
        $user = User::find($id);
        Auth::guard('web')->login($user);
        return redirect()->route('admin.users.index')->with('success', 'Active user changed
successfully.');
```

```

    }
    public function ban($id)
    {
        User::where('id', $id)
            ->update(['is_banned' => true]);
        return redirect()->route('admin.users.index')->with('success', 'User banned successfully.');
```

```

    }
    public function unban($id)
    {
        User::where('id', $id)
            ->update(['is_banned' => false]);
        return redirect()->route('admin.users.index')->with('success', 'User unbanned successfully.');
```

```

    }
}

```

## A.7 AccountController.php

```
<?php
```

```

namespace App\Http\Controllers;

use App\Http\Requests\ChangePasswordRequest;
use App\Http\Requests>EditUserRequest;
use App\Mail\changePassword;
use App\Models\Category;
use App\Models\Transaction;
use App\Models\User;
use Illuminate\Support\Facades\Auth;

```

```

use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Mail;
use Illuminate\Support\Facades\Storage;

class AccountController extends Controller
{
    public function show()
    {
        $orders = Transaction::where('user_email', Auth::user()->email)->get();
        $categories = Category::all()->where('visibility', true);
        $user = User::where('name', Auth::user()->name )->firstOrFail();
        return view("user.account", ['categories' => $categories, 'user' => $user, 'orders' => $orders]);
    }

    public function update(EditUserRequest $request, $id)
    {
        if($request->email != Auth::user()->email){
            $request->validate([
                'email' => 'required|unique:users|min:3|max:255',
            ]);
        }
        User::where('id', $id)
            ->update(['name' => $request->name, 'email' => $request->email]);
        if($request->image != "") {
            Storage::delete('public/users/' . $id . '/avatar.jpg');
            $request->file('image')->storeAs('public/users/' . $id, 'avatar.jpg');
        }
        return redirect()->route('account')->with('success', 'Account data changed successfully.');
```

```

    }

    public function passwordUpdate(ChangePasswordRequest $request, $id)
    {
        $user = User::where('name', Auth::user()->name)->firstOrFail();
        if (Hash::check($request->oldpassword, $user->password)) {
            if($request->password1 == $request->password2) {
                User::where('id', $id)
                    ->update(['password' => Hash::make($request->password1)]);
            }
        }
        else
        {
            return redirect()->route('account')->with('error', 'Incorrect current password');
        }
        Mail::to($user->email)->send(new changePassword($user));
        return redirect()->route('account')->with('success', 'Password changed successfully.');
```

```

    }
}

```

## A.8 CartController.php

```

<?php
namespace App\Http\Controllers;

```

```

class CartController extends Controller
{
    public function index() {
        $cart_item = 0;
        return view('user.cart', ['cart_item' => $cart_item]);
    }
}

```

## A.9 CatalogController.php

```
<?php
```

```
namespace App\Http\Controllers;
```

```

use App\Models\Brand;
use App\Models\Category;
use App\Models\Product;
use Illuminate\Http\Request;

```

```

class CatalogController extends Controller
{
    public function show(Request $request, $id)
    {
        $products_featured = Product::where('category', $id)->where('featured', true)->where('visibility', true)->take(6)->get();
        $brands = Brand::where('visibility', true)->get();
        $category = Category::where('name', $id)->where('visibility', true)->firstOrFail();

        if($request->brand == "") {
            $brand_current = "All";
            $page_total = ceil(count(Product::where('category', $id)->where('visibility', true)->get())/12);
            $products = Product::where('category', $id)->take(12)->get();
            $page_current = 1;
            $orderBy = "default";
            $products_quantity_from = 1;
            $products_quantity_to = count($products);
            $products_quantity_total = count(Product::where('category', $id)->where('visibility', true)->get());
        }

        if(isset($request->orderBy)){
            $orderBy = $request->orderBy;
            $page_current = $request->page;
            $brand_current = $request->brand;
            $products_quantity_from = (12 * ($page_current - 1)) + 1;
            if($request->brand == "All") {
                if ($request->orderBy == 'default') {

```

```

        $products = Product::where('category', $id)->where('visibility', true)-
>skip((( $page_current) - 1) * 12)->take(12)->get();
    }
    if ($request->orderBy == 'price-low-high') {
        $products = Product::where('category', $id)->where('visibility', true)->orderBy('price')-
>skip((( $page_current) - 1) * 12)->take(12)->get();
    }
    if ($request->orderBy == 'price-high-low') {
        $products = Product::where('category', $id)->where('visibility', true)->orderBy('price',
'desc')->skip((( $page_current) - 1) * 12)->take(12)->get();
    }
    if ($request->orderBy == 'name-a-z') {
        $products = Product::where('category', $id)->where('visibility', true)->orderBy('name')-
>skip((( $page_current) - 1) * 12)->take(12)->get();
    }
    if ($request->orderBy == 'name-z-a') {
        $products = Product::where('category', $id)->where('visibility', true)->orderBy('name',
'desc')->skip((( $page_current) - 1) * 12)->take(12)->get();
    }
    $products_quantity_to = $products_quantity_from + count($products) - 1;
    $products_quantity_total = count(Product::where('category', $id)->where('visibility', true)-
>get());
    $page_total = ceil(count(Product::where('category', $id)->where('visibility', true)-
>get())/10);
}
else {
    if ($request->orderBy == 'default') {
        $products = Product::where('category', $id)->where('visibility', true)->where('brand' ,
$brand_current)->skip((( $page_current) - 1) * 12)->take(12)->get();
    }
    if ($request->orderBy == 'price-low-high') {
        $products = Product::where('category', $id)->where('visibility', true)->where('brand' ,
$brand_current)->orderBy('price')->skip((( $page_current) - 1) * 12)->take(12)->get();
    }
    if ($request->orderBy == 'price-high-low') {
        $products = Product::where('category', $id)->where('visibility', true)->where('brand' ,
$brand_current)->orderBy('price', 'desc')->skip((( $page_current) - 1) * 12)->take(12)->get();
    }
    if ($request->orderBy == 'name-a-z') {
        $products = Product::where('category', $id)->where('visibility', true)->where('brand' ,
$brand_current)->orderBy('name')->skip((( $page_current) - 1) * 12)->take(12)->get();
    }
    if ($request->orderBy == 'name-z-a') {
        $products = Product::where('category', $id)->where('visibility', true)->where('brand' ,
$brand_current)->orderBy('name', 'desc')->skip((( $page_current) - 1) * 12)->take(12)->get();
    }
    $products_quantity_to = $products_quantity_from + count($products) - 1;
    $products_quantity_total = count(Product::where('category', $id)->where('visibility', true)-
>where('brand' , $brand_current)->get());
    $page_total = ceil(count($products)/12);
}
}
}

```

```

    if($request->ajax()){
        return view('ajax.order-by',[
            'page_total' => $page_total,
            'products' => $products,
            'brand_current' => $brand_current,
            'page_current' => $page_current,
            'products_quantity_to' => $products_quantity_to,
            'products_quantity_from' => $products_quantity_from,
            'products_quantity_total' => $products_quantity_total,
            'orderBy' => $orderBy,
        ]->render());
    }

    return view("user.catalog", ['products' => $products, 'category' => $category, 'brands' =>
$brands, 'orderBy' => $orderBy, 'brand_current' => $brand_current,
        'products_quantity_total' => $products_quantity_total, 'page_total' => $page_total,
        'products_featured' => $products_featured, 'page_current' => $page_current,
        'products_quantity_to' => $products_quantity_to, 'products_quantity_from' =>
$products_quantity_from]);
}

public function showBrand(Request $request, $id, $brand)
{
    $brands = Brand::all();
    $categories = Category::all()->where('visibility', true);
    $category = Category::where('name', $id)->where('visibility', true)->firstOrFail();
    $products = Product::where('category', $id)->where('visibility', true)->where('brand', $brand)-
>get();

    if(isset($request->orderBy)){
        if($request->orderBy == 'price-low-high'){
            $products = Product::where('category', $id)->where('visibility', true)->where('brand',
$brand)->orderBy('price')->get();
        }
        if($request->orderBy == 'price-high-low'){
            $products = Product::where('category', $id)->where('visibility', true)->where('brand',
$brand)->orderBy('price', 'desc')->get();
        }
        if($request->orderBy == 'name-a-z'){
            $products = Product::where('category', $id)->where('visibility', true)->where('brand',
$brand)->orderBy('name')->get();
        }
        if($request->orderBy == 'name-z-a'){
            $products = Product::where('category', $id)->where('visibility', true)->where('brand',
$brand)->orderBy('name', 'desc')->get();
        }
    }

    if($request->ajax()){
        return view('ajax.order-by',[
            'products' => $products

```

```

        ]->render();
    }

    return view("user.catalog", ['products' => $products, 'categories' => $categories, 'category' =>
$category, 'brands' => $brands ]);
    }
}

```

### A.10 FavoritsController.php

```

<?php

namespace App\Http\Controllers;

class FavoritesController extends Controller
{
    public function index() {
        return view('user.favorites');
    }
}

```

### A.11 HomeController.php

```

<?php

namespace App\Http\Controllers;

use App\Mail\AddtoCartMail;
use App\Mail\AddtofavoritesMail;
use App\Models\Cart;
use App\Models\Favorit;
use App\Models\Product;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Mail;

class HomeController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }

    public function index()
    {
        $value = session('cart');
        if ($value != "") {
            for ($i = 0; $i < count($value); $i++) {
                Cart::firstOrCreate(
                    ['product_id' => $value[$i], 'quantity' => 1, 'user_email' => Auth::user()->email],
                );
            }
        }
    }
}

```

```

        $product = Product::find($value[$i]);
        Mail::to(Auth::user()->email)->send(new AddtoCartMail($product));
    }
}
$favorite = session('favorite');
if ($favorite != "") {
    for ($i = 0; $i < count($favorite); $i++) {
        Favorit::firstOrCreate(
            ['product_id' => $favorite[$i]],
            ['user_email' => Auth::user()->email]
        );
        $product = Product::find($favorite[$i]);
        Mail::to(Auth::user()->email)->send(new AddtofavoritesMail($product));
    }
}
$feature_product = Product::where('featured', true)->where('visibility', true)->get();
$new_product = Product::all()->where('visibility', true)->sortBy('created_at');
return view('user.main', ['feature_product' => $feature_product, 'new_product' =>
$new_product]);
}
}

```

## A.12 MainController.php

```
<?php
```

```
namespace App\Http\Controllers;
```

```
use App\Models\Product;
use App\Models\Transaction;
use App\Models\Transactions_item;
use App\Models\User;
use Carbon\Carbon;
```

```
class MainController extends Controller
{
```

```
    public function main() {
        $feature_product = Product::where('featured', true)->where('visibility', true)->get();
        $new_product = Product::all()->where('visibility', true)->sortByDesc('created_at');
        return view('user.main', ['feature_product' => $feature_product, 'new_product' =>
$new_product]);
    }
```

```
    public function admin() {

        $sales_total = count(Transactions_item::all());
        $sales_month = Transactions_item::whereDate('created_at', '>', Carbon::now()->subDays(30))-
>count();

        $account_total = count(User::all());
        $account_month = User::whereDate('created_at', '>', Carbon::now()->subDays(30))->count();
    }
}
```

```

$transactions_all = Transaction::all();
$earnings_total = 0;
foreach ($transactions_all as $transaction){
    $earnings_total = $earnings_total + $transaction->price;
}

$transactions_month = Transaction::whereDate('created_at', '>', Carbon::now()->subDays(30))-
>get();
$earnings_month = 0;
foreach ($transactions_month as $transaction){
    $earnings_month = $earnings_month + $transaction->price;
}

$orders_total = count(Transaction::all());
$orders_month = Transaction::whereDate('created_at', '>', Carbon::now()->subDays(30))-
>count();

return view('admin.admin-panel', [
    'sales_total' => $sales_total,
    'sales_month' => $sales_month,
    'account_total' => $account_total,
    'account_month' => $account_month,
    'earnings_total' => $earnings_total,
    'earnings_month' => $earnings_month,
    'orders_total' => $orders_total,
    'orders_month' => $orders_month,
]);
}

public function ban() {
    return view('user.ban');
}
}

```

### A.13 ProductController.php

```

<?php

namespace App\Http\Controllers;

use App\Mail\AddtoCartMail;
use App\Mail\AddtofavoritesMail;
use App\Models\Favorit;
use App\Models\Product;
use App\Models\Comment;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use App\Models\Cart;
use Illuminate\Support\Facades\Mail;
use Illuminate\Support\Facades\Session;

```

```

class ProductController extends Controller
{

    public function show($name)
    {
        $product = Product::where('name', $name)->where('visibility', true)->firstOrFail();
        $comments = Comment::where('product_id', $product->id)->get();
        return view("user.product", ['product' => $product, 'comments' => $comments]);
    }

    public function addToCart(Request $request)
    {
        $product = Product::where('name', $request->name)->firstOrFail();
        $comments = Comment::where('product_id', $product->id)->get();
        if (Auth::check()) {
            $cart = Cart::firstOrCreate(
                ['product_id' => $request->id],
                ['quantity' => $request->quantity, 'user_email' => Auth::user()->email],
            );
            Mail::to($cart->user_email)->send(new AddtoCartMail($product));
        }
        else {
            session()->push('cart', $request->id);
        }

        return redirect()->view("user.product", ['product' => $product, 'comments' => $comments])-
        >with('success', 'Product added to cart successfully.');
```

```

    }

    public function cart()
    {
        return view("user.cart");
    }

    public function addToCartId($id)
    {
        if (Auth::check()) {
            Cart::firstOrCreate(
                ['product_id' => $id],
                ['quantity' => 1, 'user_email' => Auth::user()->email],
            );
            $product = Product::find($id);
            Mail::to(Auth::user()->email)->send(new AddtoCartMail($product));
        }
        else {
            session()->push('cart', $id);
        }
    }

    public function deleteFromCart($id)
    {

```

```

    Cart::where('product_id', $id)->delete();
    /*return redirect()->view('user.main')->with('success', 'Product added to cart successfully.');*/
}

public function addToFavorite($id) {
    if (Auth::check()) {
        Favorit::firstOrCreate(
            ['product_id' => $id],
            ['user_email' => Auth::user()->email]
        );
        $product = Product::find($id);
        Mail::to(Auth::user()->email)->send(new AddtofavoritesMail($product));
    }
    else {
        session()->push('favorite', $id);
    }
    /*return redirect()->route('catalog', ['id' => $product->category])->with('success', 'Product
added to favorite successfully.');*/
}

public function deleteFromFavorite($id)
{
    Favorit::where('product_id', $id)->delete();
    /*return redirect()->view('user.main')->with('success', 'Product added to cart successfully.');*/
}
}

```

## A.14 StripeController.php

```

<?php
namespace App\Http\Controllers;

use App\Mail\BuyProductMail;
use App\Models\Transactions_item;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use App\Models\Transaction;
use App\Models\Cart;
use Illuminate\Support\Facades\Mail;

class StripeController extends Controller
{

    public function fail()
    {
        Transaction::where('paid', false)->delete();
        return redirect()->route('cart.index')->with('error', 'Order has been canceled');
    }

    public function checkout(Request $request)
    {
        \Stripe\Stripe::setApiKey(config('stripe.sk'));
    }
}

```

```

$session = \Stripe\Checkout\Session::create([
    'line_items' => [
        [
            'price_data' => [
                'currency' => 'usd',
                'product_data' => [
                    'name' => 'Order',
                ],
                'unit_amount' => $request->price * 100,
            ],
            'quantity' => 1,
        ],
    ],
    'mode' => 'payment',
    'success_url' => route('success').'?session_id={CHECKOUT_SESSION_ID}',
    'cancel_url' => route('fail'),
]);
Transaction::firstOrCreate(
    ['stripe_id' => $session->id],
    ['price' => $request->price, 'user_email' => Auth::user()->email, 'paid' => false]
);
return redirect()->away($session->url);
}

public function success()
{
    Transaction::where('stripe_id', request()->session_id)
        ->update(['paid' => true]);
    $order = Transaction::where('stripe_id', request()->session_id)->firstOrFail();
    $carts = Cart::where('user_email', Auth::user()->email)
        ->get();
    foreach ($carts as $cart){
        Transactions_item::create(
            ['stripe_id' => request()->session_id, 'product_id' => $cart->product_id, 'quantity' => $cart-
>quantity]
        );
    }
    Mail::to($order->user_email)->send(new BuyProductMail($order));
    return redirect()->route('main')->with('success', 'Payment was successful');
}
}

```

### A.15 ChangepasswordRequest.php

```

public function rules()
{
    return [
        'password1' => 'required|min:4|max:20',
    ];
}

```

### **A.16 CreateBrandRequest.php**

```
public function rules()
{
    return [
        'name' => 'required|unique:brands|min:3|max:255',
    ];
}
```

### **A.17 CreateCategoryRequest.php**

```
public function rules()
{
    return [
        'name' => 'required|unique:categories|min:3|max:255',
        'image' => 'required'
    ];
}
```

### **A.18 CreateProductRequest.php**

```
public function rules()
{
    return [
        'name' => 'required|unique:products|min:3|max:20',
        'description' => 'required|min:5|max:1000',
        'price' => 'required|doesnt_start_with:0',
        'brand' => 'required',
        'color' => 'required',
        'weight' => 'required',
        'dimensions' => 'required',
        'country' => 'required',
        'links' => 'required'
    ];
}
```

### **A.19 EditCategoryRequest.php**

```
public function rules()
{
    return [
        'name' => 'required|unique:categories|min:3|max:255',
    ];
}
```

### **A.20 EditProductRequest.php**

```
public function rules()
{
    return [
```

```

        'name' => 'required|min:3|max:255',
        'description' => 'required|min:5|max:1000',
        'price' => 'required|doesnt_start_with:0',
    ];
}

```

### **A.21 EditUserRequest.php**

```

public function rules()
{
    return [
        'name' => 'required|min:3|max:255',
        'email' => 'required|min:4|max:255',
    ];
}

```

### **A.22 Admin.php**

```

class Admin extends Authenticatable
{
    use Notifiable;

    protected $fillable = [
        'name', 'email', 'password',
    ];

    protected $hidden = [
        'password', 'remember_token',
    ];
}

```

### **A.23 Brand.php**

```

class Brand extends Model
{
    use HasFactory, SoftDeletes;

    protected $fillable = ['name'];
}

```

### **A.24 Cart.php**

```

class Cart extends Model
{
    use HasFactory;

    protected $fillable = ['product_id', 'quantity', 'user_email'];
}

```

### **A.25 Category.php**

```
class Category extends Model
{
    use HasFactory, SoftDeletes;

    protected $fillable = ['name'];
}
```

### **A.26 Comment.php**

```
class Comment extends Model
{
    use HasFactory;
    protected $fillable = ['product_id', 'author', 'name', 'text'];
}
```

### **A.27 Favorit.php**

```
class Favorit extends Model
{
    use HasFactory;
    protected $fillable = ['product_id', 'user_email'];
}
```

### **A.28 Product.php**

```
class Product extends Model
{
    use HasFactory;
    protected $fillable = [
        'name',
        'featured',
        'description',
        'price',
        'category',
        'visibility',
        'image',
        'brand',
        'weight',
        'dimensions',
        'color',
        'country'
    ];
}
```

### **A.29 Transaction.php**

```
class Transaction extends Model
```

```
{
    use HasFactory;

    protected $fillable = [
        'stripe_id',
        'price',
        'user_email',
        'paid',
    ];
}
```

### **A.30 Transactions\_item.php**

```
class Transactions_item extends Model
{
    use HasFactory;

    protected $fillable = [
        'stripe_id',
        'product_id',
        'quantity',
    ];
}
```

### **A.31 User.php**

```
class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;

    protected $fillable = [
        'name',
        'email',
        'password',
    ];

    protected $hidden = [
        'password',
        'remember_token',
    ];

    protected $casts = [
        'email_verified_at' => 'datetime',
    ];
}
```

### **A.32 Create\_users\_table.php**

```
return new class extends Migration
{
```

```

public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password');
        $table->boolean('is_banned')->default(false);
        $table->rememberToken();
        $table->timestamps();
    });
}

public function down()
{
    Schema::dropIfExists('users');
}
};

```

### A.33 Create\_password\_reset\_table.php

```

return new class extends Migration
{

    public function up()
    {
        Schema::create('password_resets', function (Blueprint $table) {
            $table->string('email')->index();
            $table->string('token');
            $table->timestamp('created_at')->nullable();
        });
    }

    public function down()
    {
        Schema::dropIfExists('password_resets');
    }
};

```

### A.34 Create\_brands\_table.php

```

return new class extends Migration
{

    public function up()
    {
        Schema::create('brands', function (Blueprint $table) {
            $table->id();
            $table->string('name')->unique();
            $table->boolean('visibility')->default(true);

```

```

        $table->timestamps();
        $table->softDeletes();

    });
}

public function down()
{
    Schema::dropIfExists('brands');
}
};

```

### A.35 Create\_admins\_table\_php

```

return new class extends Migration
{

    public function up()
    {
        Schema::create('admins', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('email');
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('admins');
    }
};

```

### A.36 Create\_categories\_table\_php

```

return new class extends Migration
{

    public function up()
    {
        Schema::create('categories', function (Blueprint $table) {
            $table->id();
            $table->string('name')->unique();
            $table->boolean('visibility')->default(true);
            $table->timestamps();
            $table->softDeletes();
        });
    }
}

```

```

public function down()
{
    Schema::dropIfExists('categories');
}
};

```

### A.37 Create\_products\_table.php

```

return new class extends Migration
{

    public function up()
    {
        Schema::create('products', function (Blueprint $table) {
            $table->id();
            $table->boolean('featured')->default(0);
            $table->string('name');
            $table->text('description');
            $table->float('price');
            $table->string('category');
            $table->foreign('category')->references('name')->on('categories')->onUpdate('cascade');
            $table->string('brand');
            $table->string('weight');
            $table->string('dimensions');
            $table->string('color');
            $table->string('country');
            $table->foreign('brand')->references('name')->on('brands')->onUpdate('cascade');
            $table->boolean('visibility')->default(true);
            $table->string('image');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('products');
    }
};

```

### A.38 Create\_carts\_table.php

```

return new class extends Migration
{

    public function up()
    {
        Schema::create('carts', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('product_id');
            $table->foreign('product_id')->references('id')->on('products');
            $table->integer('quantity');

```

```

        $table->string('user_email');
        $table->foreign('user_email')->references('email')->on('users')->onDelete('cascade');
        $table->timestamps();
    });
}

public function down()
{
    Schema::dropIfExists('carts');
}
};

```

### A.39 Create\_favorites\_table.php

```

return new class extends Migration
{

    public function up()
    {
        Schema::create('favorites', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('product_id');
            $table->foreign('product_id')->references('id')->on('products')->onDelete('cascade');
            $table->string('user_email');
            $table->foreign('user_email')->references('email')->on('users')->onDelete('cascade');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('favorites');
    }
};

```

### A.40 Create\_transactions\_table.php

```

return new class extends Migration
{

    public function up()
    {
        Schema::create('transactions', function (Blueprint $table) {
            $table->id();
            $table->float('price');
            $table->string('user_email');
            $table->string('stripe_id');
            $table->boolean('paid');
            $table->timestamps();
        });
    }
}

```

```

public function down()
{
    Schema::dropIfExists('transactions');
}
};

```

#### **A.41 Create\_transactions\_items\_table.php**

```

return new class extends Migration
{

    public function up()
    {
        Schema::create('transactions', function (Blueprint $table) {
            $table->id();
            $table->float('price');
            $table->string('user_email');
            $table->string('stripe_id');
            $table->boolean('paid');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('transactions');
    }
};

```

#### **A.42 Create\_comments\_table.php**

```

return new class extends Migration
{

    public function up()
    {
        Schema::create('comments', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('product_id');
            $table->string('author');
            $table->string('name');
            $table->foreign('product_id')->references('id')->on('products')->onDelete('cascade');
            $table->foreign('author')->references('email')->on('users')->onDelete('cascade');
            $table->text('text')->nullable();
            $table->timestamps();
        });
    }

    public function down()
    {

```

```

        Schema::dropIfExists('comments');
    }
};

```

### A.43 AdminTableSeeder.php

```

class AdminTableSeeder extends Seeder
{
    public function run()
    {
        DB::table('admins')->insert([
            'name' => Str::random(10),
            'email' => Str::random(10).'@gmail.com',
            'password' => Hash::make('password'),
        ]);
    }
}

```

### A.44 Web.php

```

<?php
use Illuminate\Support\Facades\Route;
use App\Mail\WelcomeMail;
use Illuminate\Support\Facades\Mail;

Route::get('/ban', [App\Http\Controllers\MainController::class, 'ban'])->name('ban');

Route::group(['middleware' => 'ban'], function () {

    Route::get('/', [App\Http\Controllers\MainController::class, 'main'])->name('main');

    Route::get('/catalog/{name}', [App\Http\Controllers\CatalogController::class, 'show'])->name('catalog.show');

    Route::get('/catalog/{name}/{brand}', [App\Http\Controllers\CatalogController::class, 'showBrand'])->name('catalog.show.brand');

    Route::get('/products/{product}', [App\Http\Controllers\ProductController::class, 'show'])->name('products.show');

    Route::get('/products/addToCart/{id}', [App\Http\Controllers\ProductController::class, 'addToCartId'])->name('products.addToCartId');
    Route::post('/products', [App\Http\Controllers\ProductController::class, 'addToCart'])->name('products.addToCart');
    Route::get('/products/addToFavorite/{id}', [App\Http\Controllers\ProductController::class, 'addToFavorite'])->name('products.addToFavorite');

    Route::middleware("auth")->group(function () {

```

```

Route::get('/products/deleteFromCart/{id}', [App\Http\Controllers\ProductController::class,
'deleteFromCart'])->name('products.deleteFromCart');
Route::get('/products/deleteFromFavorite/{id}',
[App\Http\Controllers\ProductController::class, 'deleteFromFavorite'])-
>name('products.deleteFromFavorite');

Route::post('/comments', [App\Http\Controllers\AdminController\CommentController::class,
'store'])->name('comments.store');

Route::get('/favorites', [App\Http\Controllers\FavoritesController::class, 'index'])-
>name('favorites.index');

Route::get('/cart', [App\Http\Controllers\CartController::class, 'index'])->name('cart.index');

Route::get('/account', [App\Http\Controllers\AccountController::class, 'show'])-
>name('account');
Route::put('/account/{id}', [App\Http\Controllers\AccountController::class, 'update'])-
>name('account.update');
Route::put('/account/{id}/password', [App\Http\Controllers\AccountController::class,
'passwordUpdate'])->name('account.password.update');

Route::get('/order/fail', [App\Http\Controllers\StripeController::class, 'fail'])->name('fail');
Route::post('/order/checkout', [App\Http\Controllers\StripeController::class, 'checkout'])-
>name('checkout');
Route::get('/order/success', [App\Http\Controllers\StripeController::class, 'success'])-
>name('success');
});
});
Auth::routes();
Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])->name('home');

```

## A.45 Admin.php

```
<?php
```

```
use Illuminate\Support\Facades\Route;
```

```

Route::prefix('admin')->group(function () {
    Route::get('login', [App\Http\Controllers\Auth\AdminLoginController::class, 'login'])-
>name('admin.auth.login');
    Route::post('login', [App\Http\Controllers\Auth\AdminLoginController::class, 'loginAdmin'])-
>name('admin.auth.loginAdmin');
    Route::post('logout', [App\Http\Controllers\Auth\AdminLoginController::class, 'logout'])-
>name('admin.auth.logout');
});

Route::group(['middleware' => ['auth:admin']], function () {
    Route::prefix('admin')->group(function () {
        Route::get('/dashboard', [App\Http\Controllers\MainController::class, 'admin'])-
>name('admin');
    });
});

```

```

Route::get('/users', [App\Http\Controllers\AdminController\UserController::class, 'index'])->name('admin.users.index');
Route::get('/users/{id}/edit', [App\Http\Controllers\AdminController\UserController::class, 'edit'])->name('admin.users.edit');
Route::put('/users/{id}', [App\Http\Controllers\AdminController\UserController::class, 'update'])->name('admin.users.update');
Route::delete('/users/{id}', [App\Http\Controllers\AdminController\UserController::class, 'destroy'])->name('admin.users.destroy');
Route::post('/users/{id}/login', [App\Http\Controllers\AdminController\UserController::class, 'login'])->name('admin.users.login');
Route::post('/users/{id}/ban', [App\Http\Controllers\AdminController\UserController::class, 'ban'])->name('admin.users.ban');
Route::post('/users/{id}/unban', [App\Http\Controllers\AdminController\UserController::class, 'unban'])->name('admin.users.unban');

```

```

Route::get('/products', [App\Http\Controllers\AdminController\ProductController::class, 'index'])->name('admin.products.index');
Route::get('/products/{id}/edit', [App\Http\Controllers\AdminController\ProductController::class, 'edit'])->name('admin.products.edit');
Route::put('/products/{id}', [App\Http\Controllers\AdminController\ProductController::class, 'update'])->name('admin.products.update');
Route::get('/products/delete/{id}', [App\Http\Controllers\AdminController\ProductController::class, 'destroy'])->name('admin.products.destroy');
Route::get('/products/create', [App\Http\Controllers\AdminController\ProductController::class, 'create'])->name('admin.products.create');
Route::post('/products', [App\Http\Controllers\AdminController\ProductController::class, 'store'])->name('admin.products.store');

```

```

Route::get('/comments/index/{product_id}', [App\Http\Controllers\AdminController\CommentController::class, 'index'])->name('admin.comments.index');
Route::delete('/comments/{id}', [App\Http\Controllers\AdminController\CommentController::class, 'destroy'])->name('admin.comments.destroy');

```

```

Route::get('/categories', [App\Http\Controllers\AdminController\CategoryController::class, 'index'])->name('admin.categories.index');
Route::get('/categories/{id}/edit', [App\Http\Controllers\AdminController\CategoryController::class, 'edit'])->name('admin.categories.edit');
Route::put('/categories/{id}', [App\Http\Controllers\AdminController\CategoryController::class, 'update'])->name('admin.categories.update');
Route::get('/categories/{id}/destroy', [App\Http\Controllers\AdminController\CategoryController::class, 'destroy'])->name('admin.categories.destroy');
Route::get('/categories/{id}/restore', [App\Http\Controllers\AdminController\CategoryController::class, 'restore'])->name('admin.categories.restore');

```

```

    Route::get('/categories/create',
[App\Http\Controllers\AdminController\CategoryController::class,          'create'])-
>name('admin.categories.create');
    Route::post('/categories', [App\Http\Controllers\AdminController\CategoryController::class,
'store'])->name('admin.categories.store');

    Route::get('/brands', [App\Http\Controllers\AdminController\BrandController::class, 'index'])-
>name('admin.brands.index');
    Route::get('/brands/{id}/edit', [App\Http\Controllers\AdminController\BrandController::class,
'edit'])->name('admin.brands.edit');
    Route::put('/brands/{id}', [App\Http\Controllers\AdminController\BrandController::class,
'update'])->name('admin.brands.update');
    Route::get('/brands/{id}/destroy',
[App\Http\Controllers\AdminController\BrandController::class,          'destroy'])-
>name('admin.brands.destroy');
    Route::get('/brands/{id}/hide', [App\Http\Controllers\AdminController\BrandController::class,
'hide'])->name('admin.brands.hide');
    Route::get('/brands/{id}/unhide',
[App\Http\Controllers\AdminController\BrandController::class,          'unhide'])-
>name('admin.brands.unhide');
    Route::get('/brands/create', [App\Http\Controllers\AdminController\BrandController::class,
'create'])->name('admin.brands.create');
    Route::post('/brands', [App\Http\Controllers\AdminController\BrandController::class, 'store'])-
>name('admin.brands.store');

    Route::get('/transactions',
[App\Http\Controllers\AdminController\TransactionsController::class,      'index'])-
>name('admin.transactions.index');
    Route::get('/transactions/{stripe_id}',
[App\Http\Controllers\AdminController\TransactionsController::class,      'show'])-
>name('admin.transactions.show');
});
});

```

**ДОДАТОК Б**  
**Слайди презентації**

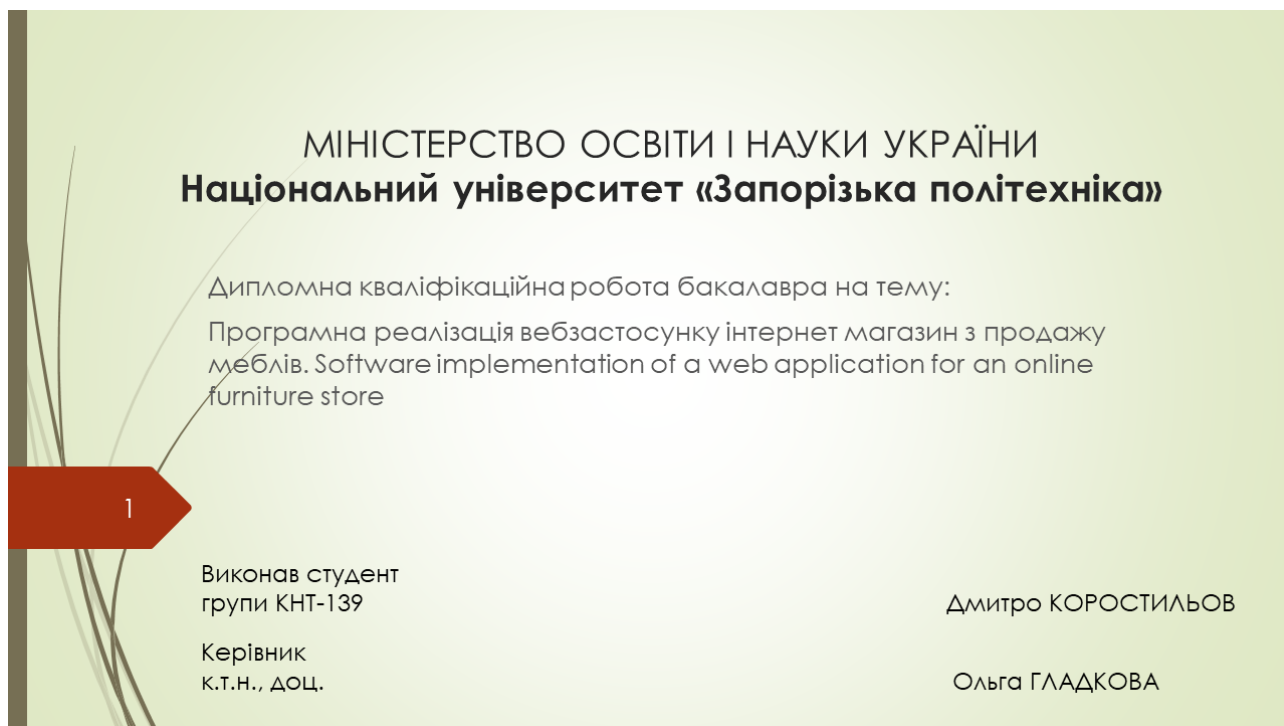


Рисунок Б.1 – Слайд № 1

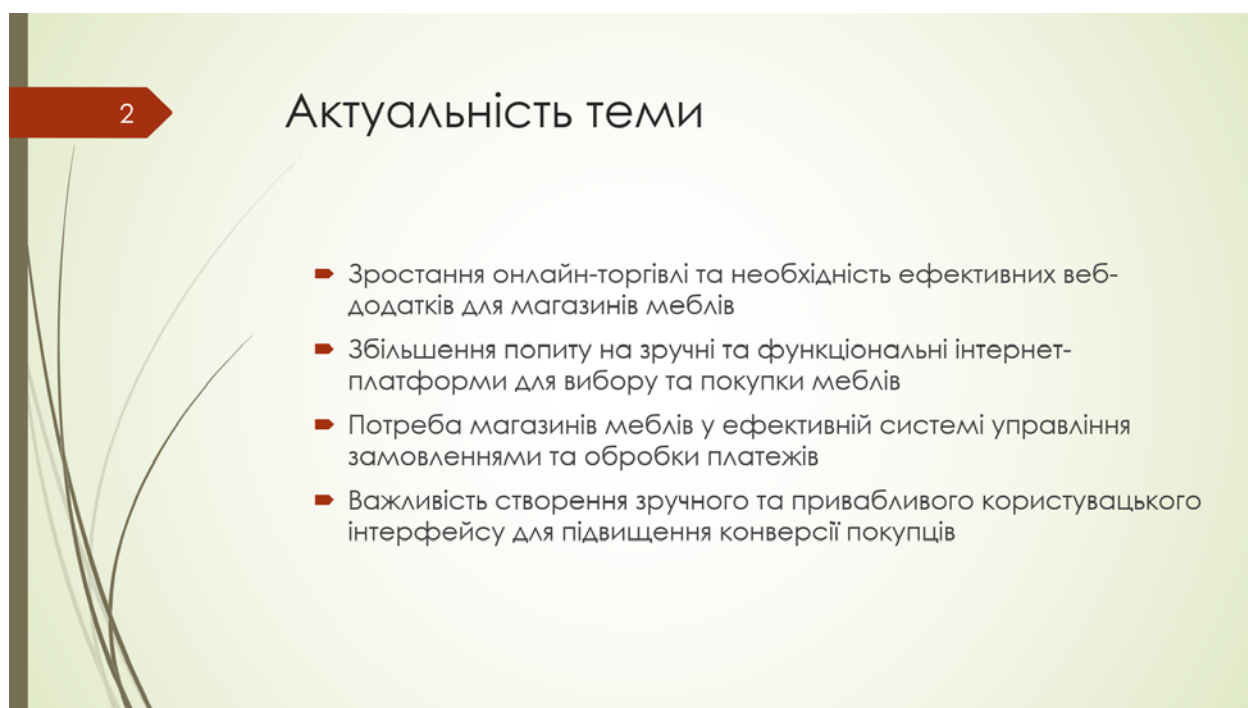


Рисунок Б.2 – Слайд № 2

3

## Мета та завдання

Мета роботи – створити вебзастосунок для магазину меблів з розділеною адміністративною та користувацькою частиною

Для вирішення поставленої мети необхідно вирішити наступні завдання

- Порівняти існуючі аналоги
- Вибрати інструментарій розробки для вебзастосунку
- Розробити архітектуру
- Розробити графічну частину сайту
- Підключити стороннє API
- Протестувати систему

Рисунок Б.3 – Слайд № 3

4

## Порівняння існуючих аналогів

| Сайт             | Переваги                                                                                      | Недоліки                                                             |
|------------------|-----------------------------------------------------------------------------------------------|----------------------------------------------------------------------|
| IKEA             | - Добре структурований та ефективний веб-сайт.<br>- Функціональність для налаштування меблів. | - Проблеми з навігацією та повільне завантаження сторінок            |
| WayFair          | - Велика кількість інформації про кожний продукт<br>- Відгуки попередніх покупців             | - Нерівномірний дизайн та нечітко визначена структура                |
| Ashley Furniture | - Зручний та легкий пошук за категоріями, кольором, стилем та матеріалом                      | - Відсутність інструментів для створення власних меблевих комплектів |

Рисунок Б.4 – Слайд № 4

5

## Вибір середовища розробки

| Веб-інструмент | Переваги                                    | Недоліки                                     |
|----------------|---------------------------------------------|----------------------------------------------|
| Wordpress      | Широкий вибір тем і плагінів                | Низька продуктивність та обмежені можливості |
| Wix            | Простий та багато шаблонів                  | Обмежена гнучкість                           |
| Joomla         | Широкий вибір тем                           | Обмежені можливості інтеграції               |
| HTML, CSS, JS  | Унікальні рішення та повний контроль        | Багато часу на розробку                      |
| Laravel        | Висока продуктивність, безпека та гнучкість | Високий поріг та потребує знання PHP, JS     |

Рисунок Б.5 – Слайд № 5

6

## Вибір інструментарію розробки

- База даних: MySQL
- Платіжна система: Stripe
- Контроль версій: GitHub
- IDE: PHPStorm
- Розгортання проекту: Docker
- Операційна система: Ubuntu

Рисунок Б.6 – Слайд № 6

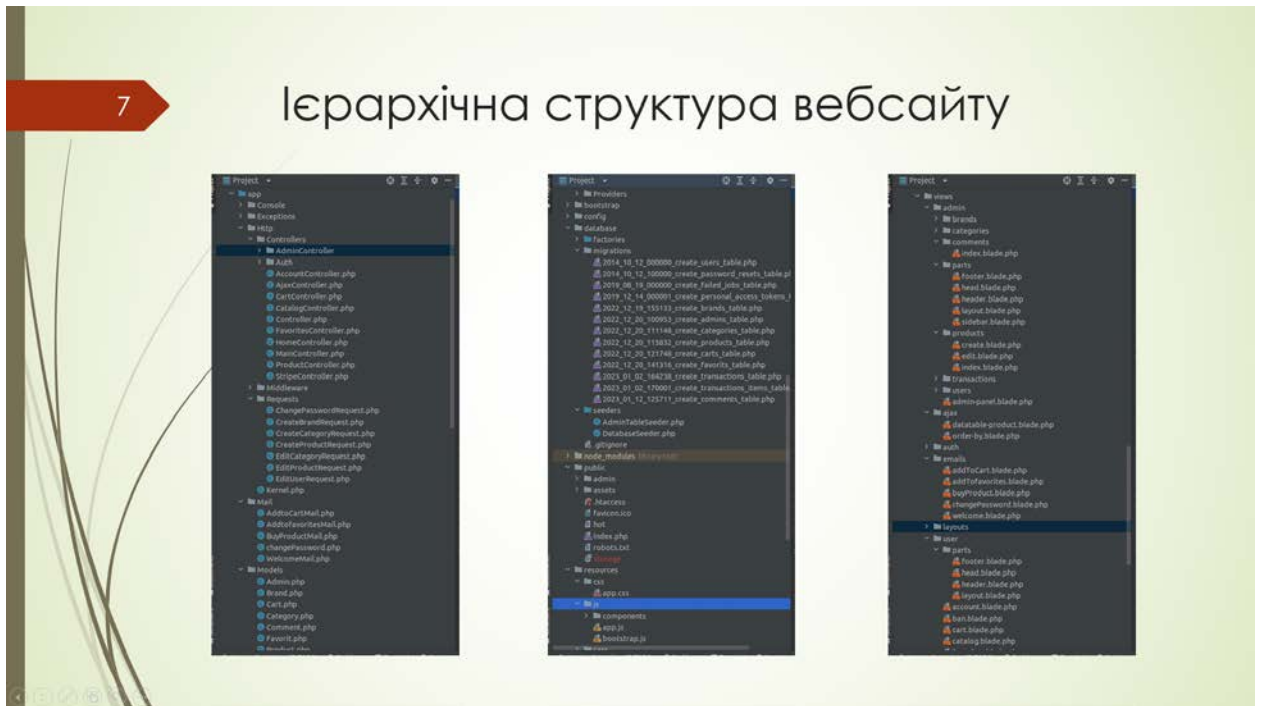


Рисунок Б.7 – Слайд № 7



Рисунок Б.8 – Слайд № 8

The image is a screenshot of a web application interface for a company named "Blackwood". The page is titled "Table" and is part of a catalog. On the left, there is a sidebar with a "Choose Brand" section containing links for "All brands", "WCA", and "Wayfarer". Below this is a "Featured Products" section displaying six items with small images and names: Abramson Noel (\$111.00), Willow Justice (\$409.00), Blake Moody (\$745.00), Kander Yang (\$582.00), Jordan Atkins (\$115.00), and another Abramson Noel (\$111.00). The main content area shows a grid of table products. At the top of this area, it says "Showing 1-5 of 5 results". There are five table products displayed in a grid, each with a small image, a brand name (WCA), a product name, and a price. The products are: Blake Moody (\$745.00), Kander Yang (\$582.00), Jordan Atkins (\$115.00), Abramson Noel (\$111.00), and Willow Justice (\$409.00). Each product has a "View" button below it. The page has a clean, modern design with a white background and a dark header. The "Blackwood" logo is in the top left. The "Table" title is in the top right. The breadcrumb "Home > Table" is below the title. The "Default setting" dropdown is also visible.

Рисунок Б.10 – Слайд № 10

11

## Послідовність дій адміністратора

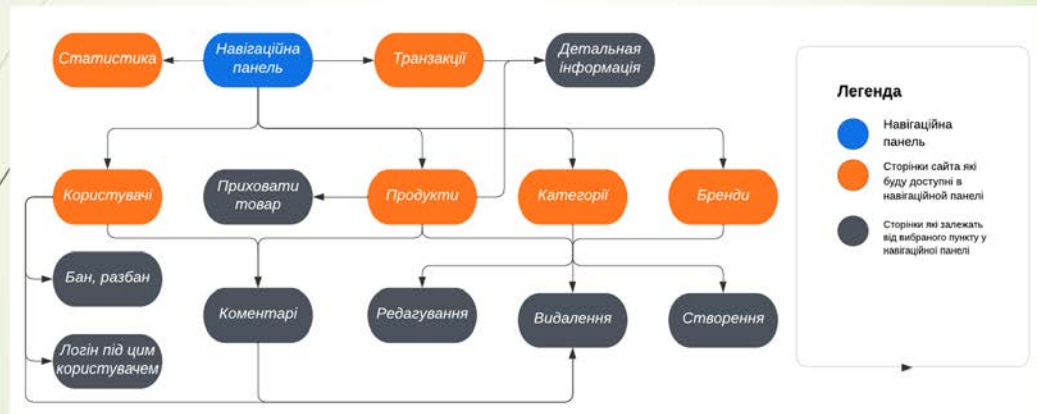


Рисунок Б.11 – Слайд № 11

12

## Графічне відображення сторінки продуктів

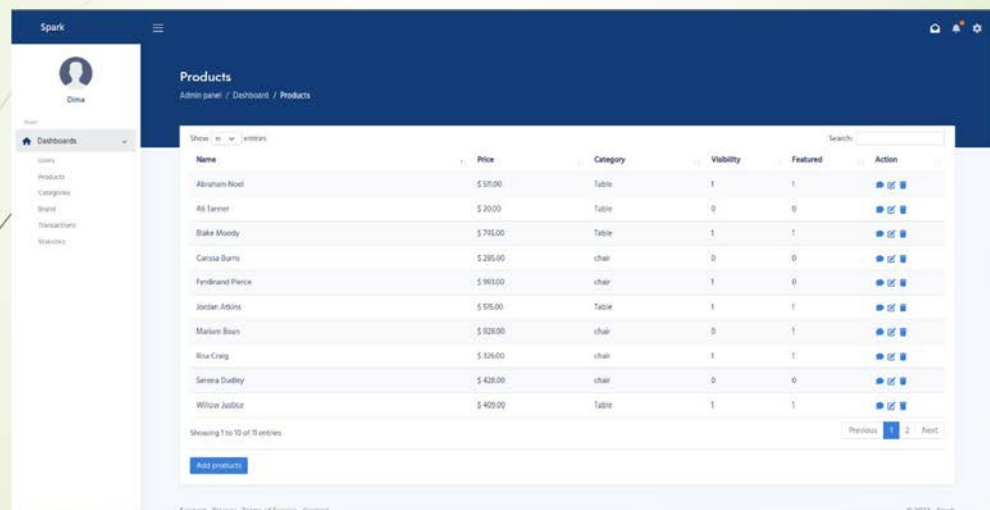


Рисунок Б.12 – Слайд № 12

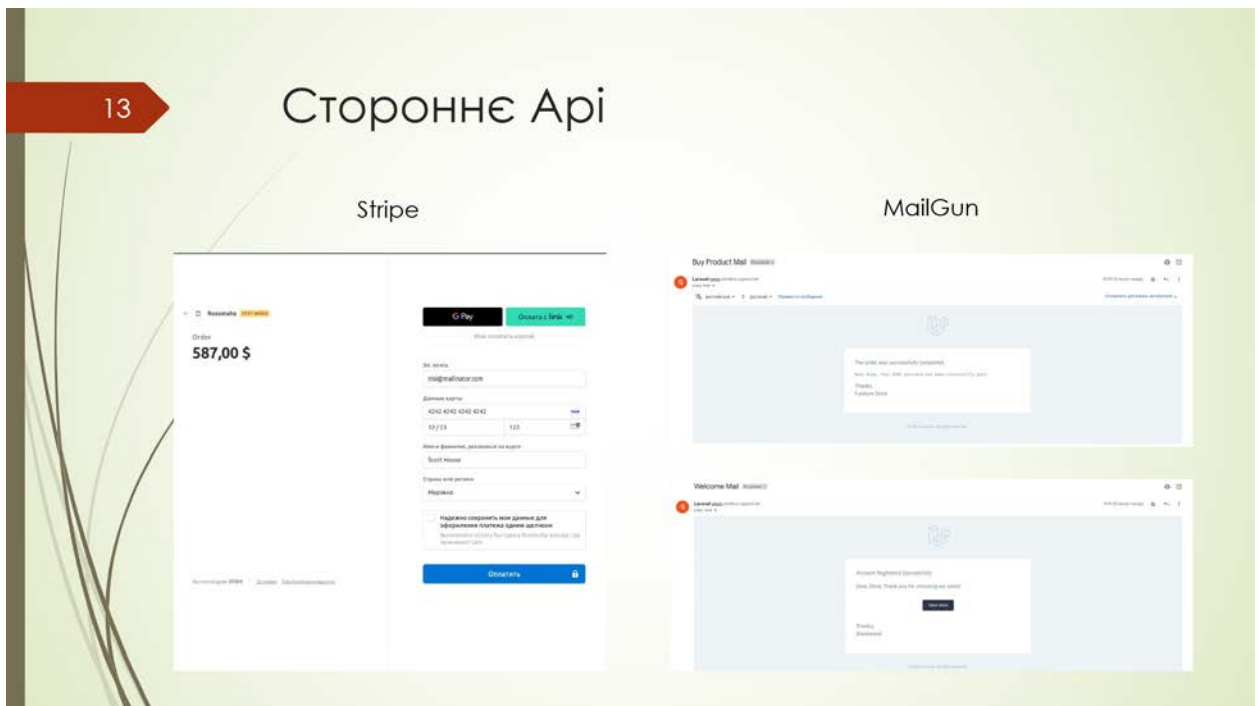


Рисунок Б.13 – Слайд № 13

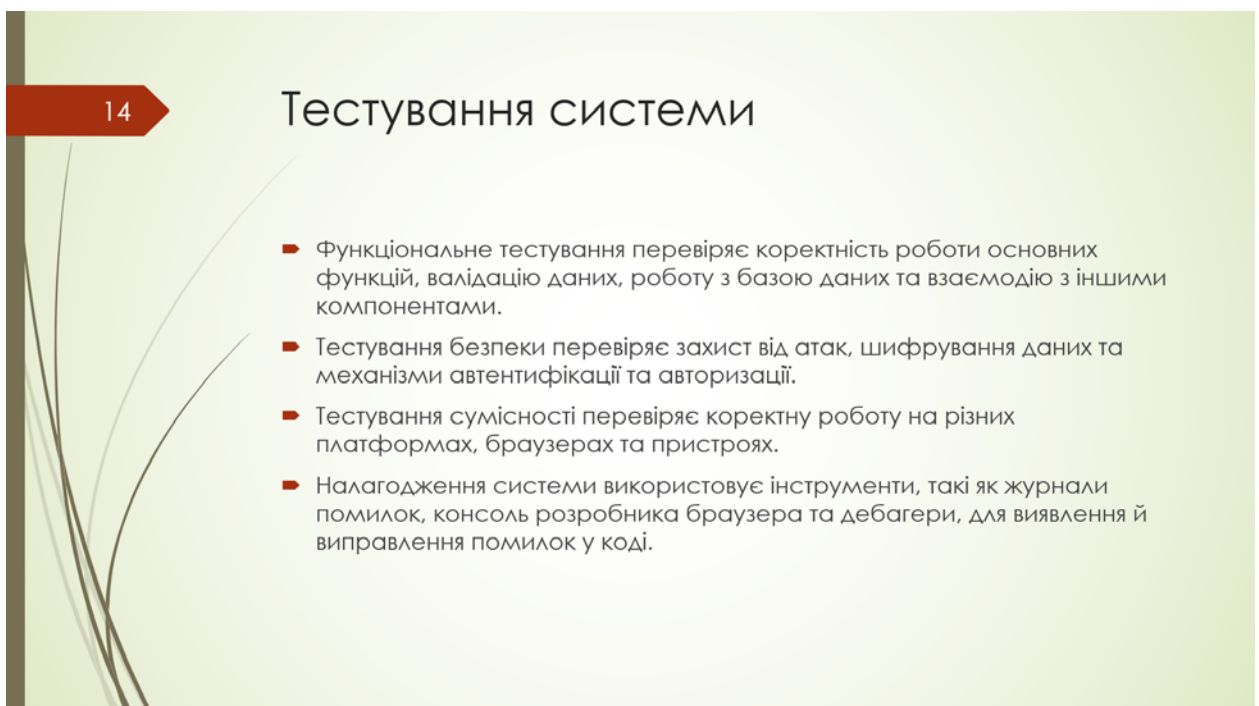


Рисунок Б.14 – Слайд № 14

## Висновки

Під час розробки нашого веб-додатка для магазину меблів ми ретельно вивчили та аналізували існуючі рішення у цій галузі. Це дало нам важливі ідеї та вимоги для проектування нашого додатка.

Ми обрали оптимальні варіанти архітектури, що забезпечують надійну та ефективну роботу системи.

Дизайн та інтерфейс нашого додатка були створені з урахуванням сучасних тенденцій у веб-дизайні, щоб забезпечити зручність використання для користувачів.

Ми інтегрували сторонні сервіси, такі як Stripe та Mailgun, для розширення функціональності та забезпечення безпеки.

Проведене тестування допомогло виявити та виправити помилки та проблеми, забезпечивши правильну роботу додатка в різних сценаріях використання.

В результаті ми створили веб-додаток для магазину меблів, який ефективно задовольняє потреби продавця та забезпечує зручність та безпеку для покупців.

Рисунок Б.15 – Слайд № 15