Міністерство освіти і науки України

Відокремлений структурний підрозділ «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя»
(повне найменування вищого навчального закладу)

Відділення інформаційних технологій, менеджменту, туризму та підготовки іноземних громадян
(назва відділення)

Циклова комісія комп'ютерної інженерії
(повна назва циклової комісії)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

фахового молодшого бакалавра
(освітньо-професійного ступеня)

на тему: **Розробка вебсайту магазину косметики «BeautyUp»**

Виконав: студент(ка) IV курсу, групи КІ-406

Спеціальності 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

Софія ГОЛОВНЯК
(ім'я та прізвище)

Керівник Володимир ЛІСОВИЙ
(ім'я та прізвище)

Рецензент _____
(ім'я та прізвище)

Тернопіль – 2026

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
імені ІВАНА ПУЛЮЯ»

Відділення **інформаційних технологій, менеджменту, туризму та підготовки іноземних громадян**
Циклова комісія **комп'ютерної інженерії**
Освітньо-професійний ступінь **фаховий молодший бакалавр**
Освітньо-професійна програма: **Обслуговування комп'ютерних систем і мереж**
Спеціальність: **123 Комп'ютерна інженерія**
Галузь знань: **12 Інформаційні технології**

ЗАТВЕРДЖУЮ
Голова циклової комісії
комп'ютерної інженерії
_____ Андрій ЮЗЬКІВ
"30" березня 2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Головняк Софії Русланівні
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: **Розробка вебсайту магазину косметики «BeautyUp»**

Керівник роботи Лісовий Володимир Миколайович
(прізвище, ім'я, по батькові)

Затверджені наказом Відокремленого структурного підрозділу «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя» від 27.03.2026р № 4/9-167.

2. Строк подання студентом роботи: 15 червня 2026 року.

3. Вихідні дані до роботи: технічне завдання на розробку програмного забезпечення, мови програмування: JavaScript, HTML, CSS, бібліотека ReactJS, платформа Node JS, фреймовк ExpressJS, стандарти IEEE 29148-2018, IEEE 29119.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Загальний розділ. Розробка технічного та робочого проєкту. Спеціальний розділ. Економічний розділ. Охорона праці та безпека життєдіяльності.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

- структурна схема інтерфейсу вебсайту;
- лістинг контролера оплати;
- інфологічна модель бази даних;
- таблиця техніко-економічних показників.

6. Консультанти розділів роботи

Розділ	Ім'я, прізвище та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	Богдана МАРТИНЮК викладач		
Охорона праці та безпека життєдіяльності	Володимир ШТОКАЛО викладач		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	01.04	
2	Збір і узагальнення інформації	08.05	
3	Написання першого розділу	15.05	
4	Розробка технічного та робочого проекту	22.05	
5	Написання спеціального розділу	28.05	
6	Розрахунок економічної частини	1.06	
7	Написання розділу охорони праці	3.06	
8	Виконання графічної частини	8.06	
9	Оформлення проєкту	10.06	
10	Погодження нормоконтролю	11.06	
11	Попередній захист роботи	12.06	
12	Захист кваліфікаційної роботи		

7. Дата видачі завдання: 31 березня 2026 року

Студент _____ Софія ГОЛОВНЯК
(підпис) (ім'я та прізвище)

Керівник роботи _____ Володимир ЛІСОВИЙ
(підпис) ім'я та прізвище)

АНОТАЦІЯ

Розробка вебсайту магазину косметики «BeautyUp» // Кваліфікаційна робота // Головняк Софія // Відокремлений структурний підрозділ «Тернопільський фаховий коледж» Тернопільського національного технічного університету імені Івана Пулюя, група: КІ-406 // Тернопіль, 2026 // с. – 148, рис. – 36, табл. – 6 , кресл. – 4, додат. – 18.

Ключові слова: ВЕБСАЙТ, ІНТЕРНЕТ-МАГАЗИН, MERN-СТЕК, REACT, NODE.JS, MONGODB, TAILWIND CSS, STRIPE API, RENDER.

Мета роботи - розробка вебсайту для магазину косметики «BeautyUp».

Пояснювальна записка складається з п’яти розділів.

В першому розділі виконано аналітичний огляд існуючих рішень та обґрунтовано вибір технологій повного стеку розробки. Описано призначення даної розробки, вимоги до програмного, апаратного забезпечення та ведення документації та технічне завдання.

В другому розділі міститься опис архітектури сайту, методи побудови клієнтської та серверної частин з використанням вибраних мов програмування, компонентів, сервісів, бази даних, а також логіку розробки проєкту.

В третьому розділі описано процедури по встановленню програмного забезпечення, розгортанню хмарної платформи та його налаштуванню. Також описано конфігурування процесів автоматичного оновлення сайту, супровід та інструкцію з експлуатації веб-додатку.

В двох наступних розділах розраховано вартість розроблюваного сайту, а також описано техніку безпеки, ергономічні та гігієнічні вимоги під час виконання даного проекту.

ABSTRACT

Development of the "BeautyUp" cosmetics store website // Qualification thesis // Sofiia Holovniak // Separate Structural Unit "Ternopil Technical College of Ivan Puluj Ternopil National Technical University", group: KI-406 // Ternopil, 2026 // p. – 150, figs. – 36, tabs. – 6, drawings – 4, app. – 18.

Keywords: WEBSITE, E-COMMERCE, MERN STACK, REACT, NODE.JS, MONGODB, TAILWIND CSS, STRIPE API, RENDER.

The purpose of the work is to develop a website for the "BeautyUp" cosmetics store.

The explanatory note consists of five chapters.

The first chapter provides an analytical review of existing solutions and justifies the choice of full-stack development technologies. It describes the purpose of this development, requirements for software, hardware, documentation management, and the technical specification.

The second chapter contains a description of the website architecture, methods for building the client and server sides using the selected programming languages, components, services, and database, as well as the project development logic.

The third chapter describes the procedures for software installation, cloud platform deployment, and configuration. It also covers the configuration of automatic website update processes, maintenance, and the web application user manual.

The next two chapters calculate the cost of the developed website and describe safety techniques, ergonomic, and hygienic requirements during the execution of this project.

ЗМІСТ

ВСТУП

Стрімкий розвиток електронної торгівлі та цифровізації суспільства відкривають нову епоху у введені бізнесу, де онлайн-платформи стають основним засобом взаємодії з кінцевим споживачем. Перетворившись із допоміжного каналу продажів на рушійну силу сучасного ритейлу, електронна комерція ставить перед компаніями складне завдання – створювати не лише функціональні рішення, але й конкурентоспроможні, інтуїтивно зрозумілі та персоналізовані веб-продукти.

У такому середовищі успіх усе більше залежить від можливості надавати унікальну цінність через цифровий інтерфейс: пришвидшення процесів, професійний підхід, гнучкість до індивідуальних потреб користувача та створення емоційного досвіду взаємодії з продуктом. У висококонкурентних галузях, таких як косметична індустрія, стандартні корпоративні рішення часто не відповідають вимогам користувачів, які прагнуть отримати одночасно науково підкріплену інформацію про склад продукту, догляд та можливість креативно тестувати і з доступом до якісного візуального контенту, що формує довірливі стосунки між покупцем та продавцем і спонукає до покупки.

Ця необхідність у створенні спеціалізованого цифрового рішення, здатного заповнити розрив між універсальними можливостями платформ електронної комерції та вузькоспеціалізованими запитами ринку краси, створює основу проблематики цього дослідження. Її актуальність зумовлена потребою впровадження інтегрованого веб-рішення, яке поєднує електронну торгівлю з освітньо-консультаційними сервісами, забезпечуючи зручний доступ до широкої інформації про продукцію та цінності бренду. Ця робота має на меті розробити сучасний корпоративний веб-сайт для бренду, який сприятиме ефективному представленню компанії в онлайн-середовищі, структурованій демонстрації категорій продукції та впровадженню простої у використанні системи замовлень, яка дозволяє персоналізувати досвід клієнтів.

Функціональність і архітектура платформи орієнтовані на розробку естетично збалансованого дизайну, який відповідає встановленому позиціонуванню бренду, створення зручної навігації між основними розділами та забезпечення коректної роботи на різних пристроях. Як результат, сайт стане потужною цифровою візитівкою бренду, що сприятиме зростанню його впізнаваності, формуванню довіри серед клієнтів і об'єднанню ключової маркетингової інформації на єдиній платформі.

Практична значущість проєкту полягає у можливості запропонованого підходу підвищити конверсію та середній чек як для зазначеного бренду, так і в суміжних галузях. Це особливо важливо в сегментах, де вибір товару потребує виразної візуалізації, наприклад, у сфері парфумерної продукції, оптики чи преміального одягу.

Отримані в результаті дослідження висновки й рекомендації щодо використання інтерактивних елементів та побудови користувацького шляху можуть посилити стратегії цифрової трансформації компанії, для яких високоякісна взаємодія онлайн є ключовою конкурентною перевагою.

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

У наш час сектор електронної комерції стрімко розвивається та перетворюється на ключову складову для будь-якого бізнесу. Розробляючи веб-сайт, фахівці зазвичай обирають між використанням готових систем та розробкою власного програмного забезпечення з чистого аркуша. Спеціалізовані рішення, серед яких варто виокремити такі відомі платформи, як Wix (див. рис. 1.1), Shopify, Squarespace та Zyro, пропонують великий спектр функціональних підходів та максимально точно підходять для різних комерційних напрямків. Такі сервіси забезпечують підприємцям швидкий запуск онлайн-магазину без необхідності у володінні глибокими технічними знаннями або залученні дороговартісної команди спеціалістів.

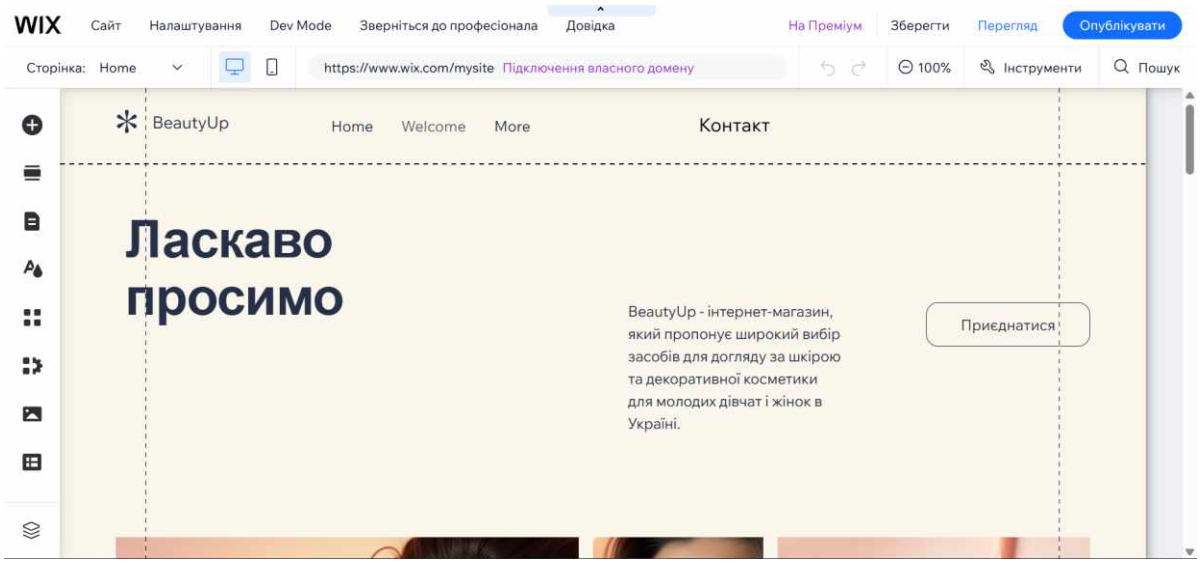


Рисунок 1.1 – Інтерфейс платформи Wix

Особлива перевага таких платформ у інтуїтивно зрозумілій панелі керування та зручному візуальному редакторі, який дозволяє власноруч змінювати дизайн сайту в режимі реального часу. Крім того, користувачі мають можливість створити свої вебсайти з нуля або за допомогою чат-бота зі штучним інтелектом

[18], який адаптує бажаний дизайн у необхідну специфіку бренду та створить оптимальну структуру сторінок.

Важливою перевагою перелічених платформ є наявність інструкцій та деталізованих підказок, які допомагають власникам бізнесів вносити необхідні корективи без сторонньої допомоги. Окремої уваги заслуговує можливість попереднього перегляду інтерфейсу веб-сторінки на різних типах пристроїв (див. рис. 1.2), що гарантує бездоганне відображення як на екранах смартфонів, так і на великих моніторах, а також дозволяє спробувати себе у ролі потенційного клієнта та відчути на собі досвід від користування майбутнім продуктом.

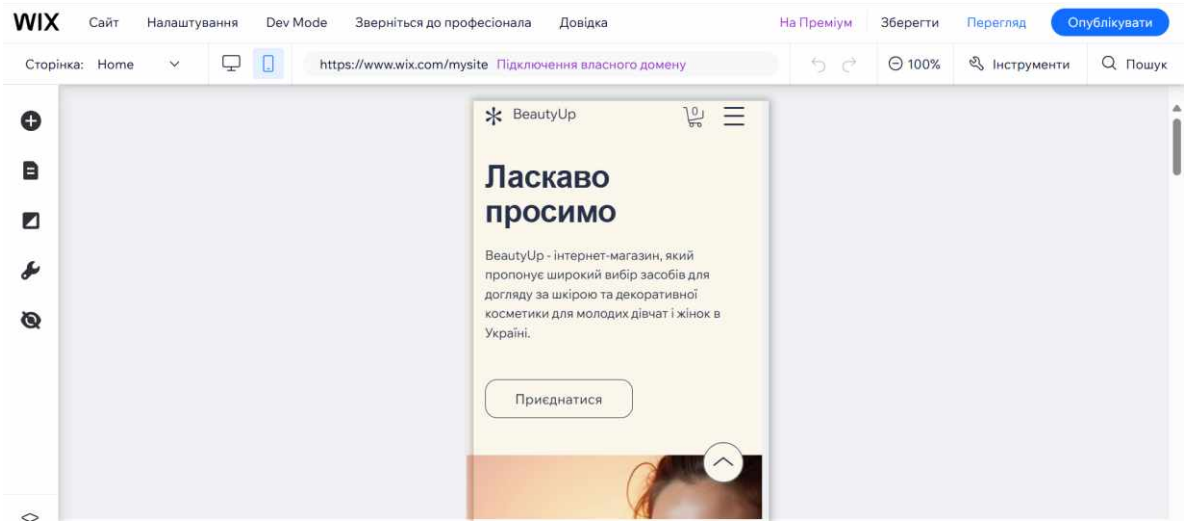


Рисунок 1.2 – Режим попереднього перегляду мобільної версії в середовищі Wix

При виборі альтернативного підходу у вигляді втілення власного проекту з нуля розробнику надається унікальна можливість абсолютної та нічим не обмеженої волі у формуванні візуальної концепції та програмної архітектури сайту. Ви самі керуєте контентом, дизайном і структурою без залежності від підрядників [25]. Таке рішення дозволяє створити унікальний веб-продукт, який ідеально транслюватиме цінності компанії через кожну деталь інтерфейсу та вигідно виділятиме його серед багатоконкурентного ринку.

Власний інтернет-магазин гарантує безпосередній та повний контроль над кожним функціональним модулем продукту, включаючи специфічні установки чи

унікальну внутрішню бізнес-логіку. Використання такого персоналізованого підходу звільняє власника бізнесу від потреби сплачувати регулярні комісійні внески за кожну транзакцію чи щомісячну плату стороннім провайдерам послуг.

Питання конфіденційності, обробки персональних даних користувачів та загального захисту транзакцій при такому виборі вирішується шляхом впровадження власних протоколів безпеки та персоналізованих алгоритмів обробки даних, що допоможе мінімізувати можливі ризики.

Хоча процес створення персоналізованого магазину вимагає значно більше часу та залучення команди фахівців з веб-розробки, що тягне за собою збільшення фінансових витрат, він стає надійною основною для довготривалого зростання та подальшого масштабування. Такі інвестиції у власну розробку забезпечують максимальну гнучкість системи та її повну адаптацію до конкретних вимог та бажаних функцій, що є критично важливим для амбітних проектів та великих бізнесів у сфері електронної комерції.

1.2 Технічне завдання

1.2.1 Найменування та область застосування

Тема дипломного проекту: Розробка вебсайту магазину косметики «BeautyUp».

Область застосування розробленого вебдодатку стосується електронної комерції, а саме – охоплення всього циклу продажу доглядової косметики під маркою «BeautyUp» через інтернет. Ця розробка має на меті надати кінцевим споживачам зручний набір функцій для підбору потрібних товарів через чітко організований каталог, безпечного проведення процедури оформлення замовлення та залучення до програми заохочень. Окрім інтерфейсу призначеного для користувачів, система також містить весь необхідний функціонал для керування комерційними операціями, що дає змогу контролювати етапи проходження замовлень та конфігурувати рекламні акції. Таким чином, таке програмне забезпечення являє собою гармонійне рішення, де вдало поєднанні

інтегровані передові підходи для демонстрації асортименту товарів покупцям та стабільні механізми для управління процесами у бізнесі.

1.2.2 Призначення розробки

Основна мета створення веб-застосунка «BeautyUp» полягає у формуванні актуальної та зручної платформи для електронної торгівлі. Платформа повинна забезпечити користувачам легкий доступ до придбання високоякісної доглядової косметичної продукції та різноманітних аксесуарів. Розробка такого проекту сфокусована на комбінуванні візуально привабливого оформлення із ефективними технологічними рішеннями для збільшення кількості онлайн-продажів та формування стійкої групи постійних покупців.

Основні цілі, поставленні при розробки такої системи:

– розробка інтуїтивно зрозумілого та мінімалістичного інтерфейсу, який спрощує як пошук потрібного товару і його огляд, так і покращує загальний користувацький досвід;

– інтеграція продуманої системи навігації та логічної побудови каталогу, яка дозволить користувачам оперативно знаходити бажаний продукт згідно з їхніми індивідуальними вимогами;

– формування деталізованих сторінок продуктів, які містять опис, склад продукції та високоякісне зображення для спрощення процесу прийняття рішення про купівлю;

– гарантування надійного захисту особистої інформації та фінансових транзакцій шляхом передових криптографічних рішень, методів автентифікації та контролю доступу;

– запровадження адаптивної програми заохочення клієнтів з промокодами, стимулюючи тим самим підвищення середнього чека;

– створення багатофункціонального інструменту для контролю та моніторингу сайту, який охоплює керування асортиментом, обробку надходжень замовлень та відображення аналітики.

Отже, розроблений веб-ресурс позиціонується як потужна цифрова система для розширення комерційної діяльності, забезпечуючи при цьому покупцям комфортний досвід взаємодії та створює умови для масштабування бренду у віртуальному просторі.

1.2.3 Вимоги до програмного забезпечення

Під час проектування та розробки онлайн-сервісу для торгівлі косметичними засобами «BeautyUp» було визначено набір ключових функціональних вимог, які забезпечують загальну працездатність системи та зручну взаємодію з кінцевим споживачем. Основні вимоги:

– наявність добре організованого переліку товарів, тобто платформа повинна мати логічно структурований розподіл товарів по категоріях, що дасть змогу відвідувачу миттєво знайти потрібну йому продукцію. Для покращення користувацького досвіду потрібно інтегрувати механізм фільтрації, який дозволить звузити пошук відповідно до заданих характеристик;

– деталізовані описи одиниць продукції, а саме кожна позиція зобов’язана мати докладний опис, представлений на окремій сторінці або у вигляді модального вікна. У цій секції відображатимуться високоякісні зображення, назва, категорія, актуальна ціна та характеристики, що дасть користувачеві змогу якомога краще ознайомитись з продуктом;

– повноцінний модуль обробки замовлень, включаючи функціонал для управління кошиком та списком бажаних позицій. Користувачі повинні мати змогу легко додавати позиції, коригувати їх кількість та ініціювати процес оплати;

– підтримка різнотипних фінансових операцій, тобто магазин має забезпечити інтеграцію з надійними платіжними шлюзами, що гарантуватиме стабільність проведення грошових переказів та ефективність платежів для різних типів клієнтів;

– персоналізація через особистий кабінет, а саме необхідно впровадити механізм реєстрації та аутентифікації з посиленими заходами безпеки даних.

Зареєстровані користувачі повинні мати доступ до свого профілю, де вони мають змогу переглянути історію виконаних транзакцій й моніторити статус поточних замовлень у реальному часі. Також повинен бути функціонал для відновлення доступу та швидкої реєстрації через сторонні застосунки;

– оптимізована пошукова система, а саме необхідно впровадити високошвидкісни1 пошук на основі ключових слів, який гарантує точність у підборі продукції та оперативну навігацію по всьому асортименту магазину;

– повноцінна зручність для мобільних пристроїв, яка має бути розроблена із застосуванням адаптивних принципів, щоб забезпечити коректне відображення всіх елементів системи та легкість у здійсненні покупок незалежно від типу пристрою, чи то стаціонарний комп'ютер, чи мобільний телефон;

– система обслуговування клієнтів через різні канали, а саме механізми комунікації, такі як електронна пошта чи чат підтримки, мають бути завжди доступними для оперативного вирішення будь-яких питань, що виникають у клієнтів;

– адміністративна панель управління, до якої необхідно інтегрувати спеціальний інтерфейс для контролю внутрішніх операцій, що надасть можливість керувати асортиментом продукції, існуючими користувачами, відстежувати статистику продажів та здійснювати повний контроль за життєвим циклом кожного оформленого замовлення.

1.2.4 Вимоги до програмної документації

Після завершення етапу розробки та локального тестування платформи «BeautyUp», необхідно підготувати комплекс супровідної документації для її успішного впровадження та подальшого функціонування. Цей комплект охоплює ключові аспекти експлуатації системи, що розподілені за наступними напрямками:

– інструкції з розробки та налаштування веб-застосунку в Інтернеті, де необхідно надати детальний опис процесу розробки фронтенду та бекенду на хмарних сервісах, включаючи визначення змінних середовища та налаштування

підключення до хмарної бази даних. Також слід надати конкретні інструкції щодо налаштування DNS-записів для прив'язки доменного імені;

– технічна підтримка та правила управління системою, у яких потрібно описати процеси моніторингу продуктивності сервера, оновлення залежностей проєкту через менеджер пакетів та підтримки цілісності коду. Цей розділ має містити рекомендації щодо створення регулярних резервних копій бази даних та файлової структури проєкту, щоб запобігти втраті даних клієнтів та історії замовлень у разі технічного збою;

– інструкції з управління контентом бренду, які повинні пояснювати процес оновлення каталогу товарів інтернет-магазину через адміністративний інтерфейс, тобто процедури додавання нових косметичних продуктів, редагування структур, розміщення оптимізованих зображень та оновлення цін. Окремо слід описати програму для створення купонів для стимулювання продажів.

– рекламна та SEO-стратегія платформи, яка повинна включати опис дій щодо підвищення впізнаваності бренду в онлайн-середовищі, що включає створення метатегів для пошукової оптимізації карток товарів, інтеграцію із соціальними мережами для залучення аудиторії та використання аналітичних інструментів для відстеження поведінки користувачів на сайті. Важливо розуміти, що головна мета SEO-просування полягає в тому, щоб допомогти пошуковим роботам краще зрозуміти і проіндексувати ваш сайт, щоб він міг ранжуватися вище в результатах пошуку за цільовими ключовими словами [27]. Такий комплексний підхід дозволяє не лише покращити видимість косметичних засобів у пошукових системах, а й забезпечити стабільний приплив потенційних покупців;

– взаємодія з клієнтами та методи обробки замовлень, а саме детальний опис робочого процесу для співробітників та відповідальних за комунікацію. У цьому розділі необхідно закріпити правила обробки вхідних замовлень, спосіб управління життєвим циклом замовлення в консолі адміністратора та систему вирішення технічних проблем користувачів через канали підтримки;

– функціональна верстка та дизайн API, де останній фрагмент документа має містити повний технічний опис можливостей розробленої системи, як-от логіку кошика, систему лояльності та особистий кабінет. Значною частиною документації має стати структурування всіх вузлів API, а також детальний опис методів, параметрів запиту та форматів відповідей сервера для забезпечення зрозумілості архітектури застосунку.

1.2.5 Техніко-економічні показники

Рішення щодо впровадження та подальшого функціонування інтернет-магазину базується на комплексному зважуванні необхідних вкладень та прогнозованої грошової віддачі. Ключові фінансові показники обчислюються, керуючись загальноприйнятими стандартами для розробки софту в секторі електронної комерції.

Успішність цієї ініціативи визначається низкою ключових технічних та матеріальних орієнтирів:

– витрати робочого часу на проектування архітектури, дизайн користувацьких інтерфейсів, розробку серверної логіки та фінальне тестування веб-сайту сумарно становлять 120 людино-годин;

– ліміт фінансових вкладень, необхідний для покриття усіх етапів розробки, реєстрації доменного ім’я та оренди хмарних середовищ, не повинен перевищувати 30 тисяч гривень;

– розрахунковий період для повного повернення інвестицій, витрачених на автоматизацію продажів та реалізації косметичної продукції, не має бути довшим за три роки.

Виконання заданих критеріїв забезпечить продукту значну перевагу серед конкурентів на ринку б’юті-сфери та закладе міцний фундамент для майбутнього технічного зростання розробленої системи.

1.2.6 Стадії та етапи розробки

Розробка вебплатформи для торгової марки косметики «BeautyUp» базується на циклічній моделі розробки, що забезпечує поступове розширення можливостей системи та швидке вирішення будь-яких технічних перешкод. Шляхи реалізації цього проєкту складаються з таких основних фаз:

– системний аналіз та концептуальне планування, що охоплює аналіз ринку засобів для догляду та формування вичерпного переліку технічних вимог до системи. Сюди входить поділ майбутнього застосунку на об’єкти, визначення логічної структури сховища даних MongoDB, а також архітектурне проєктування зв’язку між клієнтською та серверною частинами;

– інженерне проектування та UI/UX дизайн, що охоплює розробку візуальної концепції бренду та створення зручного для користувача інтерфейсу. Особлива увага приділяється логіці побудови карток товарів у вигляді інтерактивних компонентів, проєктуванню шляхів навігації та гарантуванню, що дизайн відповідає філософії компанії;

– програмна реалізація серверної архітектури, що передбачає налаштування серверного оточення на базі Node.js та Express для формування надійного API. Здійснюється створення моделей даних, впровадження бізнес-алгоритмів для обробки замовлень та інтеграція механізмів захисту особистої інформації користувачів;

– створення інтерактивного клієнтського інтерфейсу, який реалізується за допомогою бібліотеки React, акцентуючи увагу на формуванні модульних компонентів;

– комплексна перевірка та поліпшення швидкодії, що включає проведення тестів навантаження, перевірку безпеки авторизації та валідацію усіх полів для введення даних. Проводиться робота над прискоренням завантаження сторінок, мінімізацією часу відповіді сервера та усуненням знайдених помилок для досягнення максимальної стійкості готового продукту;

– фінальне розгортання та системна підтримка, тобто підготовка до релізу, перенесення його на хмарні інфраструктури та остаточну перевірку функціональності в реальних мережевих умовах. Подальша еволюція передбачає періодичне внесення нового контенту, впровадження додаткових можливостей на основі зворотного зв’язку від споживачів та технічну підтримку системи.

1.2.7 Порядок тестування та прийому

Процедура верифікації та фінального ухвалення вебплатформи «BeautyUp» передбачає систематизовану перевірку програмного забезпечення на предмет відповідності заявленим технічним вимогам. Контроль якості розробки здійснюється за такими напрямками:

1. Комплексна перевірка працездатності функціоналу, тобто дослідження коректності реалізації всіх користувацьких сценаріїв, зокрема, як працюють механізми динамічного сортування косметики, стабільність роботи системи бажаних товарів, правильність обчислення загальної вартості замовлення з урахуванням застосованих знижок та успішність проходження етапів процесу купівлі.
2. Перевірка сумісності та адаптивності на різних пристроях, метою якої є підтвердження цілісності відображення інтерфейсу у різних браузерах та оцінка гнучкості макета на пристроях з різною діагоналлю екрана. Особливий акцент робиться на зручності користування елементами системи на мобільних телефонах та планшетах.
3. Моніторинг швидкодії та оптимізація відгуку, що передбачає фіксацію швидкості завантаження сторінок із застосування спеціалізованих інструментів, дослідження затримок, що виникають на стороні сервера, а також оцінку ефективності завантаження медіаконтенту з хмарного середовища.
4. Діагностика рівня захищеності системи, тобто тестування стійкості програмного рішення проти поширених загроз кібербезпеки, таких як несанкціоноване впровадження SQL-коду або реалізація міжсайтового

скриптингу. Також перевіряється надійність використання криптографії для захисту сесій клієнтів та безпека точок доступу до програмного інтерфейсу.

5. Експертна оцінка досвіду користувачів, тобто залучення реальних представників цільового сегменту споживачів для збору відгуків стосовно зручності розташування елементів, простоти орієнтування в інтерфейсі та візуальної привабливості інтерфейсу.

Офіційний етап прийняття результатів розробки здійснюється за участі комісії, котру формують представники розробника та залучених зацікавлених сторін. Для забезпечення успішного проходження цього етапу виконавець повинен надати наступне:

– фінальний, повністю функціональний варіант вебзастосунку, розгорнутий на відповідній хостинговій платформі й підготовлений до старту комерційної експлуатації;

– повний набір вихідних файлів програмного коду, доповнений технічною документацією, переданою на електронному носії.

Презентація завершеної праці проводиться у суворій послідовності, що включає коротке висвітлення архітектурних підходів, використаних у проєкті, демонстрацію основних можливостей розробленої платформи, виконання низки тестових сценаріїв системи для підтвердження її надійності, а також надання обґрунтованих відповідей на запитання щодо реалізації кінцевого продукту

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

2.1 Розробка структури сайту і веб-сторінок

Створення будь-якого сучасного веб-продукту починається з побудови його інформаційної моделі. Для розуміння логіки взаємодії користувача з платформою необхідно визначити базові принципи побудови її навігаційної схеми. Структура сайту — це схема розташування його сторінок, категорій, підкатегорій і товарів. Це своєрідний план, в якому прослідковується логічний зв'язок між сторінками. З технічної точки зору навігація ресурсу являє собою набір URL, що розташовані в певній послідовності. Вона нерозривно пов'язана з семантичним ядром. А саме воно визначає, які папки та документи мають бути на сайті [23].

Під час проєктування структури інтернет-магазину косметики «BeautyUp» було враховано ключові архітектурні вимоги, що висуваються до сучасних систем електронної комерції, зокрема [31]:

– на сторінках присутні навігаційні ланцюжки (хлібні крихти);

– рівень вкладеності сторінок не перевищує 4 (для переходу на іншу сторінку потрібно не більше трьох кліків від головної);

– усі сторінки повинні мати посилання на головну сторінку сайту;

– при збільшенні або зменшенні кількості категорій і підкатегорій структура сайту повинна залишатися незмінною.

На основі аналізу поведінки цільової аудиторії ринку індустрії краси для розробленого проєкту було обрано гібридну організаційну структуру. Гібридна організація є комбінацією з різних способів організації інформації на одному сайті. Вона застосовується для зручності споживачів та спрощення пошуку необхідної їм інформації. У цьому випадку структура охоплює все і відразу, тому людині простіше зорієнтуватися на сайті, вибравши той варіант пошуку, який найбільш зручний [22]. Таке поєднання дає користувачам більше свободи дій та спрощує пошук інформації [24].

При вході на онлайн-платформу відвідувач відразу потрапляє на головну сторінку, де у центральній частині розміщено кредо бренду та маркетинговий заклик до дії у вигляді кнопки переходу до каталогу продукції, що спонукає користувача розпочати покупки. Нижче представлено блок із найбільш популярними та трендовими косметичними засобами – бестселерами, які привертають увагу та стимулюють продовжити перегляд продукції. Додатково на головній сторінці розташовано секцію з наявними категоріями товарів, що забезпечує швидку навігацію сайтом та спрощує пошук необхідних товарів. Такий різновид елементів на початковій сторінці дозволяє задовольнити первинний інтерес користувача та заохочує його залишитись на вебсайті для подальшого перегляду асортименту.

Замість того, щоб перевантажувати інтерфейс класичними статичними підкатегоріями, архітектура магазину використовує фільтрацію за тегами. Коли користувач заходить у певну категорію, він має можливість швидко відсортувати продукцію за алфавітом або ціновим діапазоном за допомогою динамічних фільтрів. Це зменшує час пошуку і допомагає покупцеві не витрачати багато часу на перегляд усього набору товарів. Система навігації передбачає інтеграцію ряду товарів з першої позиції до умовної n-тої кількості, де кожна продукція отримує дані з бази даних. Коли обирається товар, замість переходу на нову сторінку відкривається інтерактивний елемент, що дозволяє утримати фокус користувача на поточному екрані та підвищує конверсію.

З точки зору кінцевого відвідувача інтернет-магазину, його зустрічає адаптивний інтерфейс, який містить унікальний дизайн бренду, структурований перелік категорій та модулі для перегляду карток товарів з коротким описом. При відкриванні модального вікна юзера зустрічає деталізована інформація про назву продукції, її категорію, повний опис, актуальну ціну та високоякісне зображення.

Для здійснення покупок у системі передбачений обов’язковий поділ прав доступу, що реалізується через форми авторизації існуючих клієнтів та реєстрації нових користувачів. Для успішного входу в систему існуючому користувачеві потрібно здійснити авторизацію ввівши свою електронну адресу та актуальний пароль, в той момент як новим користувачам потрібно заповнити форму з полями

для введення імені та прізвища, електронну адресу, номер телефону та пароль. При спробі входу функціонує сувора система валідації на відповідність введених даних міжнародним стандартам електронних адрес.

Для тих, кому вдалось успішно увійти у систему, відкривається розширений функціонал – кошик та список бажаного. Під час взаємодії з будь-яким наявним у каталозі товаром у відвідувача є можливість обрати необхідну кількість одиниць та добавити його у кошик для здійснення покупки або ж у список бажаного для подальших роздумів. Інтуїтивно зрозумілий інтерфейс корзини розроблений таким чином, щоб юзер зміг легко та швидко змінювати обсяг кожної позиції, застосувати промокод та видаляти не потрібні товари.

Окремим важливим компонентом для авторизованого покупця є сторінка профілю. У цьому інтерфейсі користувачеві в структурованому вигляді відображається наявна про нього інформація в системі, така як ім’я та прізвище, номер телефону та електронна адреса. Крім того, на такій сторінці реалізовано модуль перегляду історії замовлень з трекером статусу доставки продукції та інтегровано кнопку виходу з поточного облікового запису.

На етапі фінального оформлення та оплаті замовлення функціонує захищена форма для введення реквізитів платіжної картки, таких як номер картки, термін дії та секретний CVC-код. Номер картки піддається автоматичній валідації на відповідність платіжним системам, а підтвердження фінансової транзакції здійснюється за допомогою відповідної керуючої кнопки.

Увійшовши у систему в ролі адміністратора відкриється додатковий інструмент – адмін-панель, який забезпечує повний контроль над комерційними процесами та контентом веб-сторінки. Інтерфейс такого інструменту спроектований для виконання операцій керування товарами, користувачами, категоріями, купонами, замовленнями та маркетинговими засобами. Для прикладу, при створенні нового продукту передбачено введення назви, опису, вартості, вибору однієї із наявних категорій та завантаження медіафайлу у хмарне середовище Cloudinary [8]. Програма підтримує завантаження зображень, хмарне сховище, маніпуляції із зображеннями, оптимізацію зображень для Інтернету та доставку. Cloudinary дозволяє завантажувати та зберігати необмежену кількість

зображень конфіденційно та безпечно та включає автоматичне резервне копіювання та історичні зміни. Ви можете керувати зображеннями, використовуючи широкий спектр опцій, таких як застосування ефектів, зміна розміру, обрізка, виявлення обличчя, водяні знаки та багато іншого [9].

Для розмежування обов'язків та захисту конфіденційної інформації звичайних користувачів від функцій керування, архітектура системи чітко ізолює адмін-панель за допомогою захищених маршрутів. Кожен відвідувач та адміністратор має можливість безпечно завершити сесію та вийти з облікового запису.

В результаті комплексного аналізу функціональних вимог до веб-продукту була сформована чітка ієрархічна структура. Головна сторінка виступає центральним вузлом, від якого відгалужуються форми реєстрації та авторизації, інформативні сторінки такі як про нас і контакти, а також загальний каталог товарів, що поділений на тематичні категорії: «Hair», «Face», «Body», «Sun», «Accessories» та «Sets». Окремим захищеним вектором від загальнодоступних сторінок є перехід до адміністративної панелі, яка містить такі взаємопов'язані модулі:

– інструменти створення, редагування та видалення товарів, а також можливість додавання та видалення продукту зі списку відображаємих бестселерів на головній сторінці;

– модулі створення та видалення категорій каталогу;

– систему створення, перегляду та видалення маркетингових купонів;

– інтерфейс перегляду оплачених замовлень та зміна етапу їх доставки;

– менеджер користувачів із функціоналом перегляду існуючих профілів, динамічною зміною ролей та блокування облікових записів;

– екран деталізованої бізнес-аналітики для відстеження динаміки продажів.

На основі розроблених вимог до архітектури системи була створена графічна структурна схема інтерфейсу вебсайту магазину косметики «BeautyUp» (див. рис. 2.1.), яка подана на окремому плакаті 2026.КВР.123.406.06.00.00 СС в графічній частині кваліфікаційної роботи та детально відображає взаємозв'язки усіх компонентів у платформі.

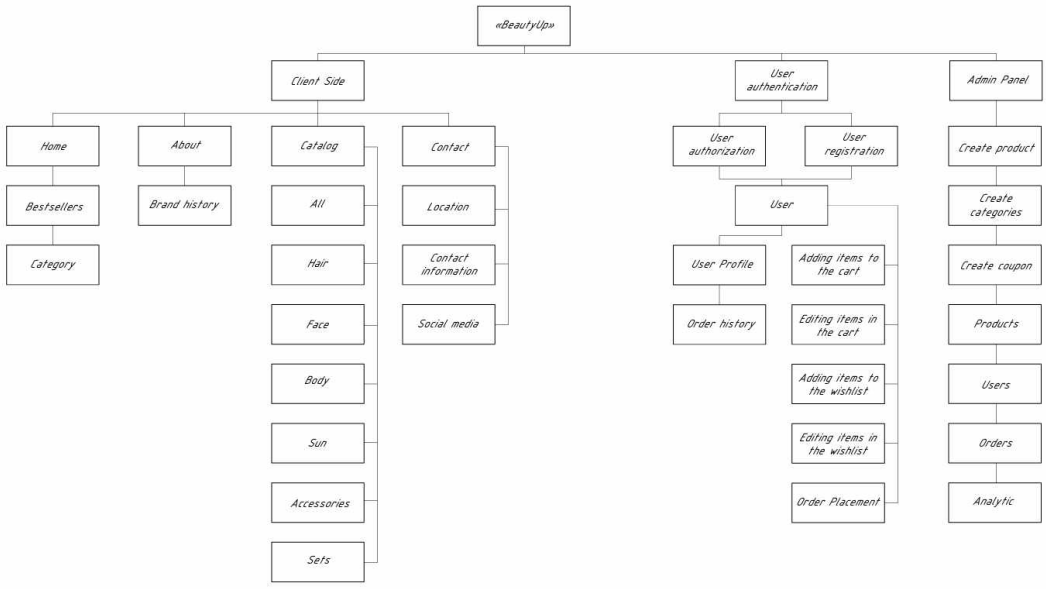


Рисунок 2.1 - Структурна схема інтерфейсу інтернет-магазину «BeautyUp»

2.2 Створення та верстка сторінок сайту

Верстка сайту – це перетворення макету дизайну на робочий сайт за допомогою програмного коду. Як правило, макет створюється за допомогою графічного редактора, а потім верстається [30]. Зазвичай такий процес відображає базову побудову неінтерактивного графічного каркасу, яке розглядається як перший і фундаментальний крок у створенні будь-якого веб-ресурсу. Саме тут формується архітектура майбутніх екранних форм проєкту, формується зовнішня оболонка системи та її візуальна привабливість для користувача, проте функціональна логіка обробки даних на цьому етапі ще не реалізується.

Класичний підхід, при якому суворо дотримується використання виключно окремих HTML і CSS файлів, втрачає свою актуальність. Таке рішення погано масштабується для великих проєктів, а подальша підтримка подібних веб-рішень стає дедалі складнішою. Прагнення оптимізувати процес розробки, підвищити гнучкість інтерфейсу та наздогнати усі актуальні тенденції в IT-індустрії зумовлює перехід до більш прогресивних практик.

З цієї причини було вирішено застосовувати компонентний підхід із використанням бібліотеки React [14] у поєднанні з утилітарним CSS-фреймворком

Tailwind CSS [16], щоб виконати необхідні роботи з реалізації фронтенд-частини для інтернет магазину косметики «BeautyUp». Це дозволило уникнути громіздкого глобального стилювання та написання великих традиційних таблиць стилів на основі інкапсульованого коду. Атомарні класи Tailwind CSS, вбудовані безпосередньо в React-компоненти, роблять стилізацію модальною, запобігаючи конфлікти стилів між різними елементами. Окрім того, вбудована адаптивність цього фреймворку у поєднанні з компонентним підходом React гарантує високу гнучкість, адже створена оболонка забезпечить коректне відображення на будь-якому розмірі екрана та стабільну роботу у різних орієнтаціях – як в портретній, так і в ландшафтній.

Побудова клієнтського середовища для веб-платформи вимагає послідовного дотримання впорядкованого списку інженерних завдань:

1. Налаштування робочого простору. Першим кроком є розгортання серверної платформи Node.js [12] та інтеграція пакетного менеджера npm (Node Package Manager) [13] для керування зовнішніми залежностями, утилітами та модулями проєкту.

2. Ініціалізація проєкту. Для створення базової структури застосунку доцільно обрати сучасний збірник Vite [17], який забезпечить надзвичайно швидке гаряче перезавантаження модулів під час розробки.

3. Декомпозиція інтерфейсу. Головна сторінка, адміністративна панель та різні додаткові модулі магазину розбиваються на деревоподібну ієрархію незалежних компонентів React. Головний каркас системи формується за допомогою адаптивної навігаційної панелі (Navbar) та нижнього колонтитула вебсайту (Footer), а сторінка каталогу продукції оперується модульними картками товарних позицій. Окремо варто виділити інтерактивні вузли досвіду користувача: кошик покупок, список бажаного, систему фільтрації категорій продукції та інтегровану панель адміністратора для керування наповненням магазину у режимі реального часу.

4. Валідація та збір даних. Оформлення замовлення , введення персональних даних та адрес доставки, створення облікових записів реалізується за допомогою

контрольованих форм, з обов’язковим впровадженням перевірки даних на коректність введення.

5. Асинхронна інтеграція з API. Враховуючи, що архітектура проєкту розподілена, тобто серверна частина функціонує незалежно, взаємодія між бекендом та фронтендом налаштовується через відправку асинхронних HTTP-запитів за допомогою спеціалізованої бібліотеки Axios [7]. Клієнтська частина через налаштовані екземпляри та перехоплювачі безпечно передає JWT-токени, динамічно отримує актуальну інформацію з бази даних, таку як каталог товарів або ціну, та ініціює процес створення замовлень.

У цільовій директорії, у терміналі середовища розробки виконується наступна команда ініціалізації, щоб розгорнути базовий каркас застосунку:

npm create vite@latest frontend -- --template react

Після введення такої команди створиться нова папка frontend з початковою структурою проекту React (див. рис. 2.2).

На етапі успішного завершення кешування усіх необхідних модулів у локальну директорію node_modules, уся подальша розробка, компіляція та запуск сервера можуть здійснюватися у автономному режимі без обов’язкового мережевого доступу.

Рисунок 2.2 – Базова структура клієнтської частини проєкту інтернет-магазину «BeautyUp»

Також варто зазначити, що процес початкової ініціалізації проєкту та подальшого розгортання дерева залежностей за допомогою пакетного менеджера npm потребує стабільного підключення до мережі Інтернет.

На відміну від застарілих підходів, у конфігурації Vite файл index.html виступає безпосередньою точкою входу та розташовується в корені проєкту. Такий файл містить єдиний базовий тег-контейнер із відповідним ідентифікатором, у який інтегрується весь динамічний інтерфейс. Специфіку розміщення кореневого елемента відображено на рисунку 2.3.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/favicon.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>frontend</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

Рисунок 2.3 – Кореневий HTML-елемент та підключення модуля ініціалізації інтерфейсу

Як показано на рисунку, динамічне виведення всього інтерфейсу користувача, логіки Redux-слайсів та маршрутизації сторінок у порожній блок відбувається за допомого підключення головного скрипта /src/main.jsx.

Запуск локального сервера для перевірки результатів розробки та тестування інтерфейсу в реальному часі здійснюється за допомогою наступної команди:

npm run dev

Така організація файлової структури та процесу збірки забезпечує швидку роботи клієнтської частини інтернет-магазину як під час роботи над кодом, так і після його фінального компресії для розміщення на хостингу.

2.3 Розробка структури бази даних сайту

База даних — це структурована колекція даних, яка зберігається в електронному форматі та доступна для операцій обробки, пошуку та аналізу [29]. При створенні сучасних динамічних інтернет-магазинів підсистема збереження даних дуже важлива. Електронна комерція потребує постійно зберігати, змінювати та швидко створювати різний контент – каталоги товарів, профілі клієнтів, замовлення покупців. Це реалізується засобами систем керування базами даних (СКБД).

На початку розробки вебплатформи «BeautyUp» розглядалось підключення звичайної реляційної бази даних, такої як MySQL. Проте детальний аналіз предметної області допоміг зрозуміти, що сувора реляційна модель вимагає створення розгалуженої структури з великою кількістю ізольованих таблиць. Це призвело б до надмірного використання операцій об’єднання, що суттєво знижує швидкодію системи при високому навантаженні.

Щоб забезпечити системі гнучкість, швидкість та подальше масштабування було прийняте рішення використовувати документо-орієнтовану NoSQL базу даних MongoDB. Ця система працює з документами у форматі BSON (Binary JSON) замість жорстких таблиць, що дозволяє зберігати масиви та вкладені структури всередині одного об’єкта.

Для кращого розуміння відмінностей між технологіями розробки у таблиці 2.1 наведено порівняльну характеристику реляційної бази даних MySQL та документо-орієнтованої бази даних MongoDB.

Використання MongoDB дозволило виконати ефективну денормалізацію даних та оптимізувати структуру збереження. Завдяки підтримці вбудованих об’єктів загальну кількість необхідних сутностей вдалось скоротити до п’яти базових колекцій.

Оскільки NoSQL підхід дозволяє зберігати данні про приналежність до категорії та медіа-контент безпосередньо у полях документа продукту, а перелік

замовлених позицій у вигляді масиву об’єктів всередині документа замовлення, це повністю виключило потребу у створенні проміжних колекцій зв’язку.

Таблиця 2.1 - Порівняльна характеристика реляційної СУБД MySQL та NoSQL бази даних MongoDB

Критерій порівняння	Microsoft SQL Server	MongoDB
Схема даних	Фіксована	Гнучка
Основна модель бази даних	Реляційна СКБД	Документоорієнтована СКБД
Ліцензування	Комерційне	Відкрите ПЗ
Мова програмування	C++	C++, Go, Python, JavaScript
Операційні системи сервера	Windows	Solaris, Linux, OSX, Windows

Для встановлення відповідності між термінологією класичних SQL-систем та NoSQL рішеннями, на рисунку 2.4 продемонстровано співвідношення структурних одиниць збереження інформації.

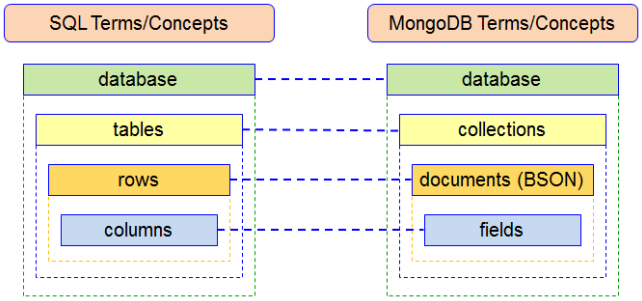


Рисунок 2.4 - Співвідношення термінологічних концептів у реляційних та NoSQL базах даних

Логічна архітектура розробленої бази даних інтернет-магазину косметики базується на п’яти ключових колекціях, які забезпечують повноцінне функціонування системи, швидкий доступ до інформації та ізоляцію сутностей.

До базових структурних одиниць сховища належать:

– Users – призначена для зберігання персональних даних користувачів;

– Products – призначена для зберігання товарних позицій;

– Orders – призначена для зберігання замовлень;

– Coupons – призначена для зберігання купонів;

– Categories – призначена для зберігання категорій.

Колекція Users міститиме документи з наступними полями:

– name (тип даних: String) – призначене для зберігання імені та прізвища користувача;

– email (тип даних: String) – призначене для зберігання унікальної адреси електронної пошти користувача;

– phone (тип даних: String) – призначене для зберігання унікального мобільного номера користувач;

– password (тип даних: String) – призначене для зберігання пароля користувача;

– wishlist (тип даних: Array) – призначене для зберігання товарних позицій, добавлених користувачем у список бажаного;

– resetPasswordCode (тип даних: String / null) – призначене для зберігання тимчасового секретного коду, що генерується системою для відновлення паролю;

– resetPasswordExpires (тип даних: Date / null) – призначене для зберігання часового штампу, який визначає термін дії секретного коду для скидання паролю;

– role (тип даних: String) – призначене для зберігання рівня доступу користувача;

– cartItem (тип даних: Array) - призначене для зберігання товарних позицій, добавлених користувачем у кошик;

– isBanned (тип даних: Boolean) – призначене для зберігання статусу блокування користувача.

Колекція Products міститиме документи з наступними полями:

– name (тип даних: String) – призначене для зберігання назви товарної позиції;

– description (тип даних: String) – призначене для зберігання опису товарної позиції;

– price (тип даних: Int) – призначене для зберігання вартості товарної позиції;

– image (тип даних: String) – призначене для зберігання уніфікованого покажчика ресурсу товарної позиції;

– category (тип даних: String) – призначене для зберігання текстового ідентифікатора категорії;

– isFeatured (тип даних: Boolean) – призначене для зберігання статусу обраного для відображення продукту на головній сторінці.

Колекція Orders міститиме документи з наступними полями:

– products (тип даних: Array) – призначене для зберігання товарних позицій придбаних користувачем;

– totalAmount (тип даних: Double) – призначене для зберігання кінцевої загальної вартості замовлення;

– stripeSessionId (тип даних: String) – призначене для зберігання унікального ідентифікатора платіжної сесії;

– status (тип даних: String) – призначене для зберігання поточного статусу замовлення;

– phone (тип даних: String) – призначене для зберігання унікального контактного номера отримувача;

– shippingAddress (тип даних: String) – призначене для зберігання адреси доставки.

Колекція Coupons міститиме документи з наступними полями:

– code (тип даних: String) – призначене для зберігання унікального текстово-цифрового ідентифікатора промокоду;

– discountPercentage (тип даних: Int) – призначене для зберігання числового значення відсоткової знижки;

– expirationDate (тип даних: Date) – призначене для зберігання календарної дати закінчення терміну дії купона;

– isActive (тип даних: Boolean) – призначене для зберігання поточного статусу промокоду: активований або деактивований;

– minimumPurchaseAmount (тип даних: Int) – призначене для зберігання порогової вартості замовлення для застосування купона;

– isFirstOrderOnly (тип даних: Boolean) – призначене для зберігання статусу, який визначає чи призначений даний купон лише для нових користувачів.

Колекція Categories міститиме документи з наступними полями:

– name (тип даних: String) – призначене для зберігання назви категорії товарів;

– href (тип даних: String) – призначене для зберігання текстового ідентифікатора, який необхідний для перенаправлення користувача на сторінку з відфільтрованим переліком товарів цієї категорії;

– imageUrl (тип даних: String) – призначене для зберігання уніфікованого покажчика ресурсу категорії товарів.

Алгоритмічна структура запитів до бази даних розглянуті в розділі 2.4.2.

З метою детального аналізу та кращого розуміння взаємозв’язків між сутностями бази даних, було розроблено графічну інфологічну модель бази даних (див. рис. 2.5), яка винесена на окремий плакат 2026.КВР.123.406.06.00.00 БД у графічній частині кваліфікаційної роботи та детально відображає логічну структуру збереження даних інтернет-магазину «BeautyUp».

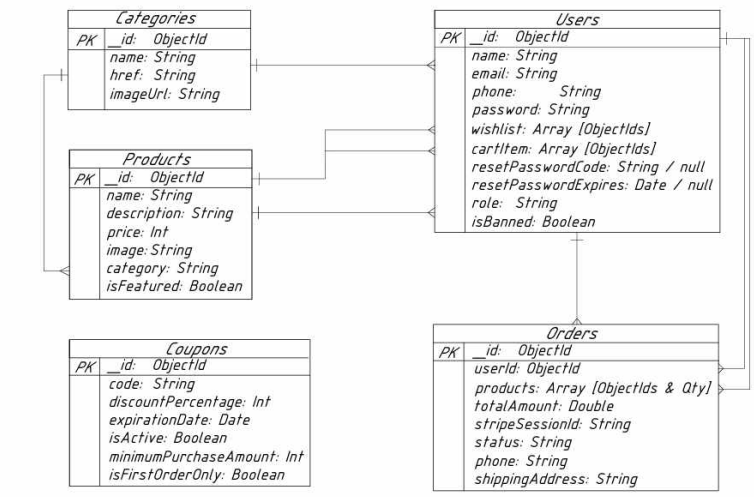


Рисунок 2.5 – Інфологічна модель бази даних

2.4 Програмування сайту

2.4.1 Написання клієнтської частини

Сучасне створення систем електронної комерції використовує концепцію розподілу зон відповідальності, де розробку вебсайту розбивають на два блоки: клієнтську частину та серверну частину. Фронтенд та бекенд працюють разом та є злагодженою системою, де взаємодіють багато компонентів. Бекенд — це бізнес-логіка застосунку чи сайту, а фронтенд — логіка всього, що відбувається на екрані [33]. Клієнтська частина створює інтерфейс, який працює безпосередньо у веббраузері користувача, забезпечуючи візуальну репрезентацію даних та ергономіку інтерфейсу.

У зв’язку з цим інженерні завдання розробника інтерфейсу виходять за межі простого відтворення графічних макетів. Фронтенд вимагає адаптації коду під різні типи дисплеїв, кросбраузерності та програмування логіки, що реагує на стороні клієнта.

Стандарти створення високонавантажених вебдодатків класу Single Page Applications (SPA) змушують розробників відмовитись від класичних маніпуляцій із DOM-деревом на користь нових прогресивних компонентів. Серед варіантів, таких як Vue.js або Angular, для інтернет-магазину косметики «BeautyUp» було обрано бібліотеку React. Його ключова ідея — компонентний підхід: розробник створює невеликі, незалежні частини інтерфейсу (кнопки, форми, меню), які потім збираються разом, як елементи конструктора, щоб утворити комплексний інтерфейс [15]. Такі компоненти дозволяють розділити інтерфейс користувача на незалежні частини, придатні до повторного використання, і сприймати їх як такі, що функціонують окремо один від одного [20].

Цінність використання багаторазових інтерфейсних модулів у клієнтській платформі полягає в оптимізації швидкодії вебресурсу. Автономність кожної структурної одиниці, зумовлена наявністю інкапсульованої пам’яті, гарантує її

ізольованість від суміжних структур. Це суттєво полегшує виконання модульного тестування та мінімізує виникнення збоїв у загальній екосистемі вебресурсу.

Для опису зовнішнього вигляду компонентів використовується синтаксис JSX. Таке розширення дозволяє писати звичний HTML-подібний код безпосередньо всередині JavaScript-файлів. Це полегшує з'єднання логік роботи додатка з його візуальною розміткою.

Практична реалізація вищеописаних концепцій вимагає чітко організувати точку входу в застосунок. Після створення базової структури вебдодатка за допомогою інструменту Vite, головним керуючим файлом та точкою входу для запуску JavaScript-коду у папці src є модуль main.jsx (src/ main.jsx). У цьому файлі потрібно підключити і запустити основне дерево компонентів платформи.

Програмний алгоритм ініціалізації передбачає звернення до глобального методу createRoot з пакета react-dom/client, щоб прив'язати віртуальне дерево React до реального DOM-елементу у браузері. Такий глобальний метод createRoot працює через ідентифікатор root. Для забезпечення інтернет-магазину косметики масштабованості, безпеки та динамічності кореневий компонент App вбудовується у спеціалізовані системні обгортки (див. рис. 2.6).

```
createRoot(document.getElementById('root')).render(
  <StrictMode>
  <BrowserRouter>
    <App />
  </BrowserRouter>
  </StrictMode>,
)
```

Рисунок 2.6 – Структура файлу main.jsx

Повний лістинг коду конфігурації файлу src/main.jsx представлено у Додатку А пояснювальної записки до кваліфікаційної роботи.

Після ініціалізації базового вузла у модулі main.jsx, наступний крок у створенні клієнтської частини інтернет-магазину косметики «BeautyUp» це налаштування кореневого компонента App.jsx (файл src/App.jsx). Цей модуль виконує роль головного диспетчера додатка: саме в ньому зосереджена логіка

клієнтської маршрутизації, підключення наскрізних інтерфейсних елементів та розмежування прав доступу користувачів. Візуальне представлення структури файлу src/App.jsx наведено на рисунку 2.7.

```
return (
  <div className='min-h-screen bg-white text-gray-900 flex flex-col'>

    <div className='relative z-50 flex-grow'>
      <Navbar />

      <div className="pt-20">
        <Routes>
          <Route path='/' element={<HomePage />} />
          <Route path='/about' element={<AboutPage />} />
          <Route path='/contact' element={<ContactPage />} />
          <Route
            path='/profile'
            element={user ? <ProfilePage key={user._id} /> : <Navigate to='/login' />}
          />

          <Route path='/signup' element={!user ? <SignUpPage /> : <Navigate to='/' />} />
          <Route path='/login' element={!user ? <LoginPage /> : <Navigate to='/' />} />
          <Route path='/forgot-password' element={<ForgotPasswordPage />} />

          <Route
            path='/secret-dashboard'
            element={user?.role === "admin" ? <AdminPage /> : <Navigate to='/login' />}
          />

          <Route path='/catalog/:category?' element={<CatalogPage />} />
          <Route path='/wishlist' element={user ? <WishlistPage /> : <Navigate to='/login'  />} />
          <Route path='/cart' element={user ? <CartPage /> : <Navigate to='/login' />} />
```

Рисунок 2.7 - Структура файлу App.jsx

Кореневий компонент розбито за кількома основними функціональними напрямками, першим з яких є створення глобальної навігаційної оболонки. Компонент оголошує елементи інтерфейсу, які залишаються нерухомими та доступними на кожній сторінці сайту, такі як верхня панель навігації Navbar та підвал сайту Footer. Це інженерне рішення дозволяє зберегти єдиний візуальний стиль платформи, коли користувач переходить між різними розділами сайту.

Шляхи та маршрути в інтернет-магазині працюють через компоненти Routes і Route з бібліотеки react-router-dom. У структурі файлу чітко визначено та розмежовано URL-адреси для кожної окремої сторінки магазину, включаючи головну сторінку, каталог товарної продукції, кошик та список бажаного покупця, інтерфейс особистого профілю та блоки авторизації та реєстрації користувачів.

Важливою складовою логіки компонента є механізм захисту приватних маршрутів. У загальну структуру інтегровано алгоритм динамічної перевірки статусу автентифікації користувача та його поточної ролі у системі. Це дозволить

надійно заблокувати доступ для звичайних покупців чи гостей вебсайту до панелі керування AdminPage і автоматично переспрямовувати на сторінку логіна при спробі потрапити до закритих зон вебзастосунку.

Повний програмний лістинг вихідного коду компоненту <App /> наведено в Додатку Б кваліфікаційної роботи.

Для забезпечення високо рівня підтримуваності, масштабованості та читабельності вихідного коду інтернет-магазину, файлову структуру клієнтської частини було організовано на модульним принципом. Такий підхід розділяє зони відповідальності між елементами інтерфейсу, логікою керування станом та допоміжними інструментами.

Основне середовище розробки зосереджене всередині каталогу src. Окрім глобальних модулів ініціалізації main.jsx, App.jsx та таблиці стилів index.css, розташовано чотири ключові піддиректорії:

– папка components призначена для зберігання дрібних, ізольованих UI-елементів багаторазового використання, таких як картки товарної продукції, кнопки та модальні вікна;

– папка lib призначена для глобальних системних налаштувань та конфігурацій підключення зовнішніх бібліотек;

– папка pages призначена для зберігання повнорозмірних сторінок інтернет-магазину, які формують логічно завершенні інтерфейси для користувача;

– папка stores призначена для керування глобальним станом застосунку, що забезпечує централізований доступ до інформації з будь-якого компонента системи.

Така організація дозволяє мінімізувати зв’язність між виглядом і логікою, полегшуючи подальшу модернізацію та супровід програмного продукту. На рисунку 2.8 показано структуру каталогу src розробленого фронтенд-додатка.

Практична реалізація клієнтської взаємодії в інтернет-магазині стартує з модулів, які відповідають за безпечний доступ та платформи та ідентифікацію відвідувачів. Для цього у папці pages розроблено компоненти сторінок авторизації та реєстрації.

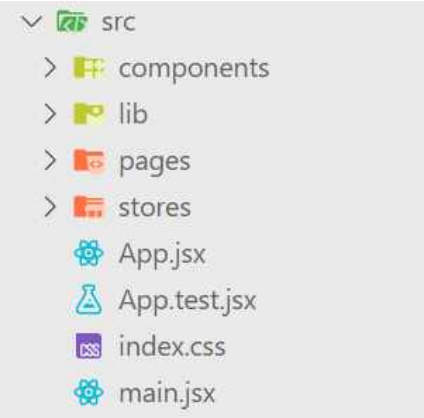


Рисунок 2.8 - Структура каталогу src

Програмний механізм стандартної автентифікації передбачає введення облікових даних користувача у форму, де під час натискання кнопки входу спрацьовує обробник подій, який передає параметри до сховища для відправки асинхронного HTTP-запиту на серверну частину. Безпека сесій та захист передачі даних реалізовані на основі технології JWT (JSON Web Tokens). Після успішної перевірки логіна та пароля сервер генерує унікальний токен, який фронтенд-додаток зберігає для подальшої автоматичної верифікації кожного запиту клієнта. Візуальний інтерфейс форми входу до системи на сторінці LoginPage (файл src/pages/Login.Page.jsx) представлений на рисунку 2.9.

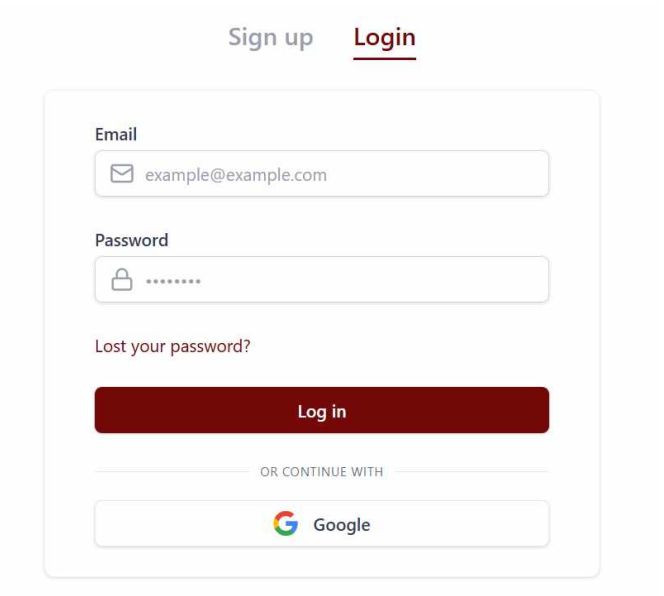


Рисунок 2.9 – Візуальний інтерфейс форми авторизації

Для забезпечення швидкого доступу до платформи, окрім класичної авторизації, до платформи було інтегровано альтернативний механізм входу за допомогою хмарної платформи Firebase. У користувача є можливість авторизуватись одним кліком без необхідності ручного введення паролів, через єдиний обліковий запис Google, після чого додаток отримує безпечний ідентифікатор користувача безпосередньо від сервісів Google та здійснює вхід у систему. Лістинг коду компоненту <LoginPage /> поданий в Додатку В кваліфікаційної роботи.

Аналогічним чином, із використанням JWT-токенів, побудовано процес реєстрації нових клієнтів на сторінці SignUpPage (файл src/pages/SignUpPage.jsx). Коли відвідувач заповнює розширену форму та натискає кнопку “Register” спрацьовує обробник handleSubmit, який зчитує введені дані, валідує їх на стороні клієнта та через метод signup глобального сховища useUserStore (файл src/stores/useUserStore.js) відправляє їх на сервер для внесення нового користувача до бази даних. Графічне відображення сторінки створення нового облікового запису наведено на рисунку 2.10.

Лістинг коду компоненту <SignUpPage /> поданий в Додатку Г кваліфікаційної роботи.

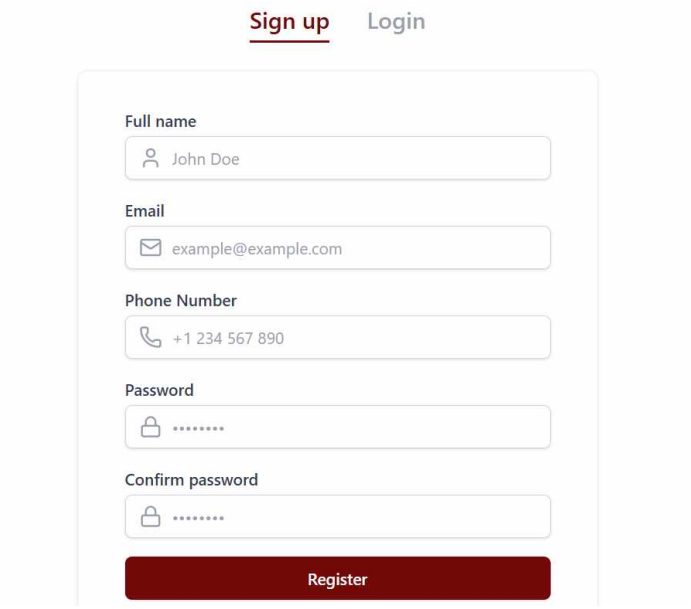


Рисунок 2.10 - Візуальний інтерфейс форми реєстрації

Важливою складовою підсистеми управління є забезпечення можливості самостійного відновлення паролю облікового запису користувача. Для цього у фронтенді була реалізована окрема сторінка ForgotPasswordPage (файл src/pages/ForgotPasswordPage.jsx) відновлення доступу, інтерфейс якої містить форму для зареєстрованної електронної пошти (див. рис. 2.11).

Кнопка “Send Reset Code” запускає обробник подій, який ініціює відправку спеціалізованого індивідуального коду на поштову скриньку клієнта, що дозволяє безпечно створити новий пароль та заново увійти в обліковий запис користувача інтернет-магазину «BeautyUp». Після введення електронної пошти система автоматично перекидає користувача на іншу форму.

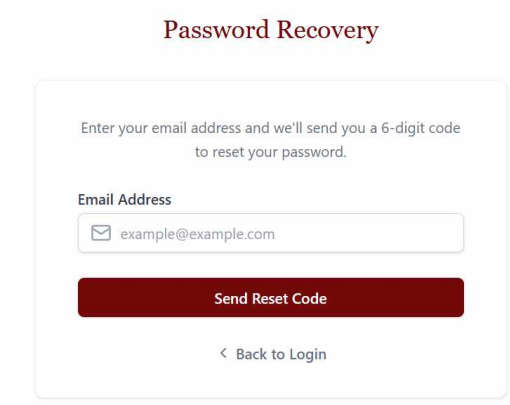


Рисунок 2.11 – Форма для введення зареєстрованої електронної пошти

На новій формі необхідно ввести шестизначний код, який надійшов у листі, а також вказати новий пароль для облікового запису (див. рис. 2.12).

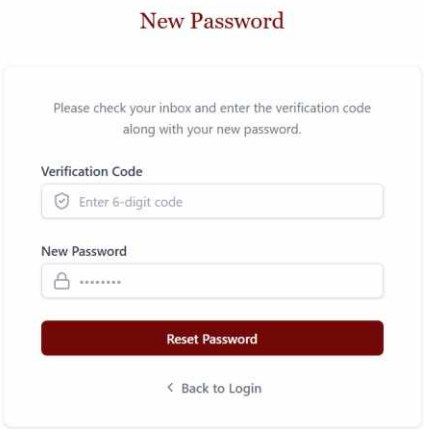


Рисунок 2.12 - Форма для створення нового пароля користувача

Важливий елемент, що тримає користувача в постійному контакті з сайтом це компонент навігаційного меню <Navbar /> (файл src/components/Navbar.jsx). Особливістю реалізації цього меню є його динамічна трансформація залежно від поточного стану глобального сховища авторизації. Меню адаптує свій вигляд, перелік доступних посилань та функціональних кнопок, під конкретну роль відвідувача платформи. Для неавторизованого гостя, який ще не здійснив вхід у систему вебдодатка, панель навігації показує базовий набір посилань на сторінки сайту, кнопку для пошуку товарної позиції та кнопка переходу до сторінок авторизації та реєстрації, що продемонстровано на рисунку 2.13. Лістинг коду компоненту <Navbar /> поданий в Додатку Д.

Рисунок 2.13 – Вигляд компоненту <Navbar /> для неавторизованого користувача

Коли користувач успішно проходить автентифікацію, компонент <Navbar /> одразу реагує на зміну стану в сховищі даних та змінює інтерфейс. Замість кнопок входу з'являється посилання на особистий профіль користувача, відображається іконка кошика та списку бажаного з динамічним лічильником кількості доданих товарів, як показано на рисунку 2.14.

Рисунок 2.14 – Вигляд компоненту <Navbar /> для авторизованого користувача

Найбільш розширений функціонал навігаційного меню є у випадку, коли в систему заходить користувач із правами адміністратора. Компонент зчитує роль користувача із захищеного токена та інтегрує спеціальну кнопку швидкого переходу до панелі керування інтернет-магазином, що дозволяє миттєво потрапити на адмін панель (див. рис. 2.15).

Рисунок 2.15 – Вигляд компоненту <Navbar /> для адміністратора

Якщо користувач не перейшов за жодним посиланням, система автоматично відобразить індексний маршрут, який співставляється з компонентом <HomePage /> (файл src/pages/HomePage.jsx). У цьому компоненті реалізовано виведення початкового інтерфейсу інтернет-магазину косметики «BeautyUp». На сторінці розміщено коротке привітання, а також інтерактивну кнопку для швидкого переходу до повного каталогу продукції (див. рис. 2.16). Лістинг коду компоненту <HomePage /> поданий в Додатку Е дипломного проекту.

Рисунок 2.16 – Візуальний інтерфейс компоненту < HomePage />

Нижче на сторінці розміщено блок із товарами-бестселерами магазину, який реалізовано за допомогою компонента <FeaturedProduct /> (файл src/components/ FeaturedProduct.jsx). Даний компонент спроектований як інтерактивний слайдер і приймає масив об’єктів featuredProducts як вхідний параметр(пропс). Наповнення цього масиву повністю контролюється через адміністративну панель вебсайту: адміністратор ставить або знімає позначку “featured” для будь-якої товарної позиції в базі даних, після чого зміни в реальному часі відображаються на головній сторінці (див. рис. 2.17).

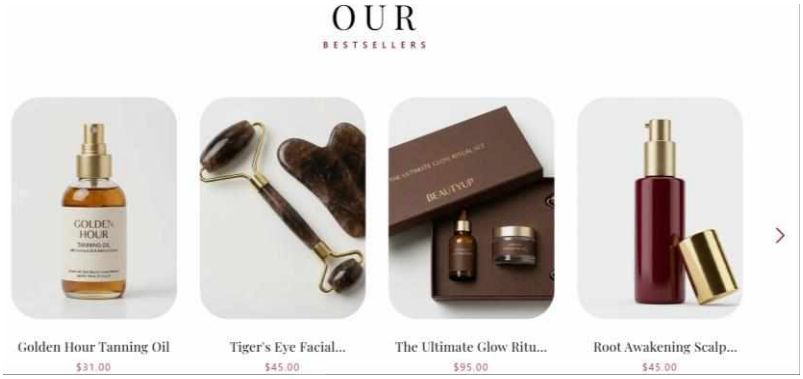


Рисунок 2.17 – Вигляд компоненту <FeaturedProduct />

Останній елемент інтерфейсу, який замикає загальну структуру кожної сторінки інтернет-магазину, є компонент нижньої панелі сайту <Footer /> (файл src/components/Footer.jsx). Даний компонент працює як інформаційний хаб, що постійно відображається в самому низу вебсторінки і надає користувачеві швидкий доступ до контактних даних магазину (див. рис. 2.18). Повний лістинг вихідного коду компоненту <Footer /> наведено в Додатку Є кваліфікаційної роботи.

Рисунок 2.18 – Інтерфейс компонента <Footer />

Коли користувач переходить до перегляду продукції система обробляє маршрут /catalog/:category?, який співставляється в головному маршрутизаторі з компонентом <CatalogPage /> (файл src/pages/CatalogPage.jsx). Оскільки параметр у шляху є необов’язковим, цей єдиний компонент відповідає як за виведення загального каталогу товарів, так і за відображення конкретної вибірки. Зовнішній вигляд сторінки CatalogPage відображено на рисунку 2.19.

Перелік категорій вебдодатку не прописаний у вихідному коді, оскільки адміністратор сайту може самостійно створювати чи видаляти їх через панель керування. Всі текстові дані динамічно завантажуються із хмарної бази даних MongoDB з колекції categories, а графічні зображення цих категорій, які

використовуються для візуального оформлення відповідної секції на сторінці HomePage, підтягуються з хмарного середовища Cloudinary з папки “categories”.

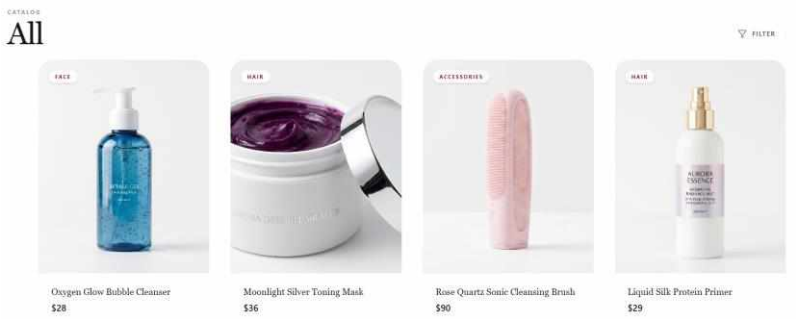


Рисунок 2.19 - Компонент <CatalogPage />

В компоненті <CatalogPage /> реалізовано поле пошуку для швидкої фільтрації косметичних засобів як за назвою, так і за категорією. Пошукове слово береться з URL-рядка, що дозволяє здійснювати глобальний пошук по всьому сайту (див. рис. 2.20). Усі застосовані користувачем фільтри об’єднуються в один запит і передаються на сервер через глобальне сховище.

Рисунок 2.20 - Фільтрація продуктів за пошуковим словом

На карточці товару компоненту <ProductCard /> (файл src/components/ProductCard.jsx) відображається основна інформація про товарну позицію та дві ключові кнопки: для додавання товару до кошика та для додавання товару у список бажаного (див. рис. 2.21). Лістинг коду компоненту <ProductCard /> поданий в Додатку Ж кваліфікаційної роботи.

Рисунок 2.21 – Зовнішній вигляд карточки товару

При натисканні кнопки “Add to Cart” авторизованим користувачем ініціюється асинхронний запит через сховище UseCartStore (файл src/stores/UseCartStore.js) на серверну сторону, де оновлюється вміст кошика в базі даних та змінюється значення лічильника товарів у шапці сайту. Процес успішного додавання продукту супроводжується миттєвим виведенням спливаючого повідомлення за допомогою вбудованих засобів бібліотеки react-hot-toast (див. рис. 2.22).

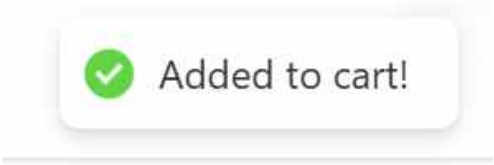


Рисунок 2.22 – Спливаюче повідомлення після додавання товару в кошик

Компонент <WishlistPage /> (файл src/pages/WishlistPage.jsx) призначений для відображення товарів, доданих авторизованим користувачем до списку бажань. На цій сторінці виводяться усі товари, які користувач раніше позначив відповідною позначкою. У відвідувача є можливість переглянути усі додані товари до списку бажаного, видалити їх або додати їх до кошику. Усі зміни стану синхронізуються з базою даних через сховище useWishlistStore (файл src/stores/useWishlistStore.js). На рисунку 2.23 зображений зовнішній вигляд компонента <WishlistPage />.

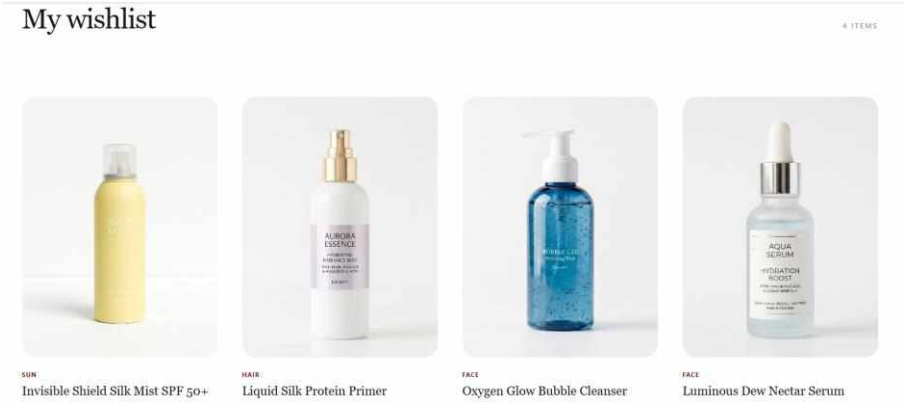


Рисунок 2.23 – Зовнішній вигляд компонента <WishlistPage />

На сторінці CartPage (файл scr/pages/CartPage.jsx) користувачеві відкривається можливість (див. рис. 2.24):

– переглянути сформоване замовлення;

– змінити кількість товарів;

– додати нові товари в кошик за допомогою компонента <PeopleAlsoBought /> (файл src/components/PeopleAlsoBought.jsx), в якому відображаються рекомендовані товари;

– видалити товари з кошика;

– застосувати промокод за допомогою компонента GiftCouponCard.jsx (файл src/components/GiftCouponCard.jsx), який аналізує доступні пропозиції та рекомендує користувачеві діючі купони;

– оплатити замовлення.

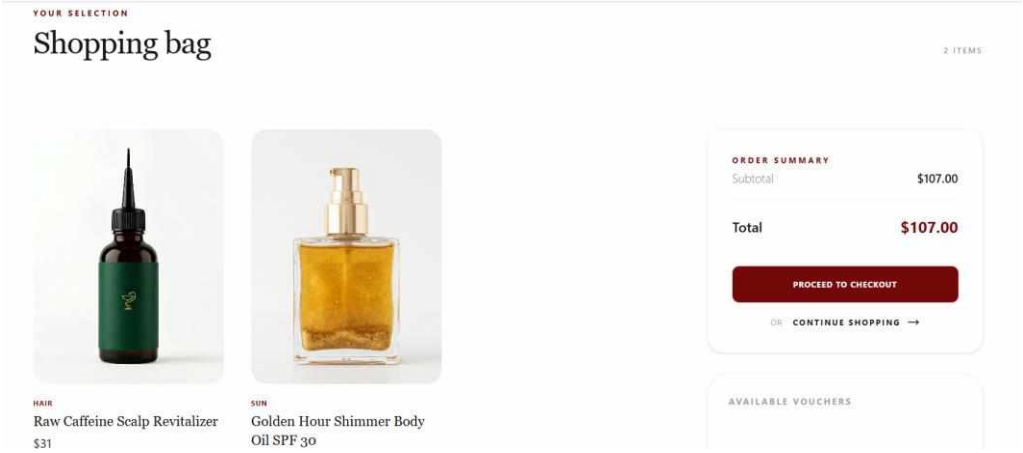


Рисунок 2.24 – Інтерфейс кошика

Лістинг коду компоненту <CartPage /> поданий в Додатку З кваліфікаційної роботи.

Фінальним етапом роботи з кошиком є перехід до безпечної оплати сформованого замовлення, яка реалізована за допомогою інтеграції з міжнародною платіжною платформою Stripe (див. рис. 2.25).

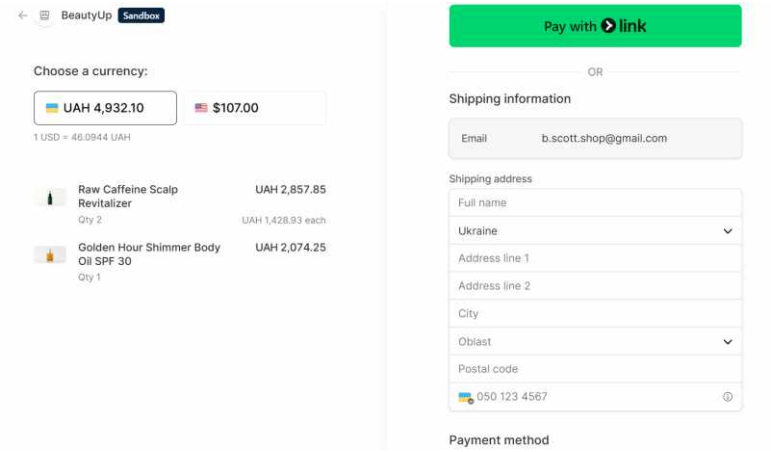


Рисунок 2.25 – Інтерфейс форми оплати замовлення

Після успішного проведення транзакції користувач автоматично перенаправляється на компонент <PurchaseSuccessPage /> (файл src/pages/PurchaseSuccessPage.jsx), який візуалізує підтвердження успішної оплати, виводить номер транзакції та очищає поточний стан кошика (див. рис. 2.26).

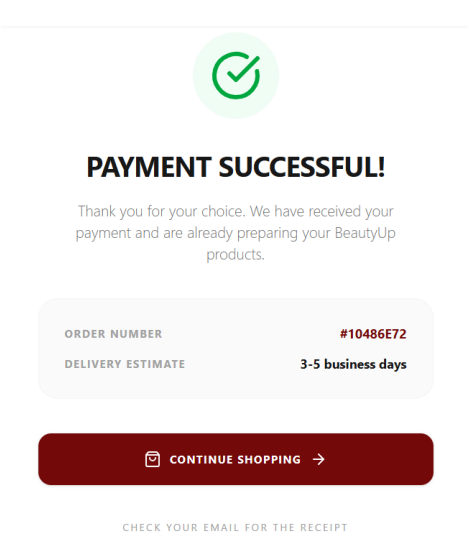


Рисунок 2.26 – Візуальне підтвердження оплати

Для перегляду персональних даних та відстеження історії замовлень авторизований користувач може перейти до свого профілю через навігаційне меню сайту (див. рис. 2.14). За рендеринг цієї сторінки відповідає компонент <ProfilePage /> (файл src/pages/ProfilePage.jsx). Унікальною візуальною особливістю кожної картки замовлення є трекер доставки, побудований на основі кроків масиву orderSteps, який включає такі етапи: “Paid”, “Packed”, “Shipped”, “Delivered”.

Якщо авторизований користувач має роль адміністратора магазину, то для нього стають відкривається меню адміністративної панелі <AdminPage /> (файл src/pages/AdminPage/jsx) та стають доступними наступні компоненти:

– <CreateProductForm /> (файл src/components/CreateProductForm.jsx), у якому реалізовано функціонал додавання нової товарної продукції;

– <CreateCategory /> (файл src/components/CreateCategory.jsx), у якому реалізовано функціонал додавання нової категорії та видалення наявних категорій;

– <CouponManager /> (файл src/components/CouponManager.jsx), у якому реалізовано функціонал створення нового купона та видалення наявних купонів;

– <ProductsList /> (файл src/components/ProductsList.jsx), у якому реалізовано функціонал виведення продуктів магазину та керування ними (видалення, редагування товарів та ставити/знімати позначку “featured” з продукції;

– <UsersManager /> (файл src/components/UsersManager.jsx), у якому реалізовано функціонал виведення користувачів вебдодатку та керування ними(зміна ролі користувача та можливість блокування входу в систему);

– <OrdersList /> (файл src/components/OrdersList.jsx), у якому реалізовано функціонал виведення оплачених замовлень магазину;

– <AnalyticsTab /> (файл src/components/AnalyticsTab.jsx), у якому реалізовано функціонал виведення аналітики інтернет-магазину.

Повний лістинг компоненту <AdminPage /> наведено в додатку И кваліфікаційної роботи

Для керування інтернет-магазином адміністратору необхідно перейти до адмін-панелі, скориставшись вкладкою “Dashboard” у навігаційному меню (див. рис. 2.15). Функціонал додавання нових позицій здійснюється на вкладці “Create Product”, що реалізовано в компоненті <CreateProductForm />. Компонент має форму (див. рис. 2.27) для введення назви, опису й ціни товару, вибору його категорії та завантаження зображення, яке зберігається у хмарному сховищі Cloudinary у папці “products”.

Повний лістинг компоненту <CreateProductForm /> наведено в додатку І кваліфікаційної роботи.

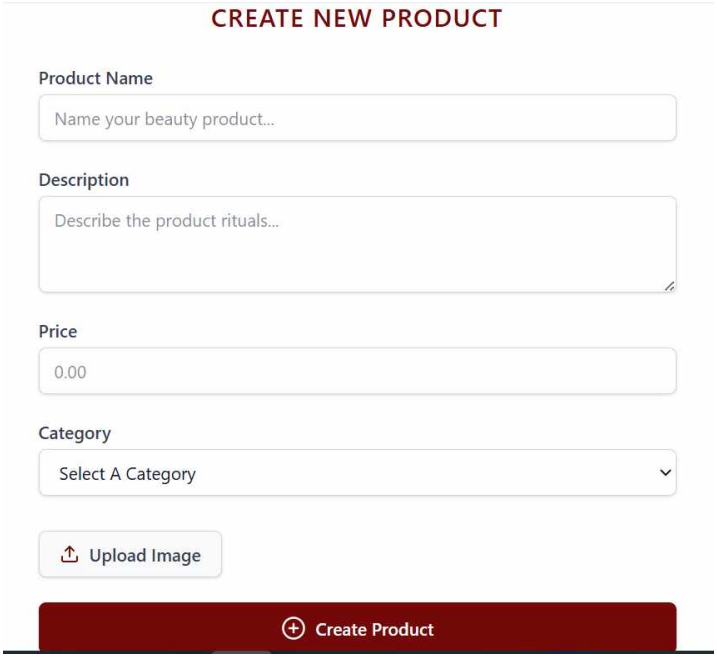


Рисунок 2.27 – Форма для створення нового продукту

Для моніторингу та адміністрування наявних у базі даних товарів призначена вкладка “Products” в адмін-панелі, за роботу якої відповідає компонент <ProductsList />. У цьому компоненті відображається таблиця усіх товарів із значенням (див. рис. 2.28).:

- їхнього зображення;
- назви;
- категорії;
- ціни.

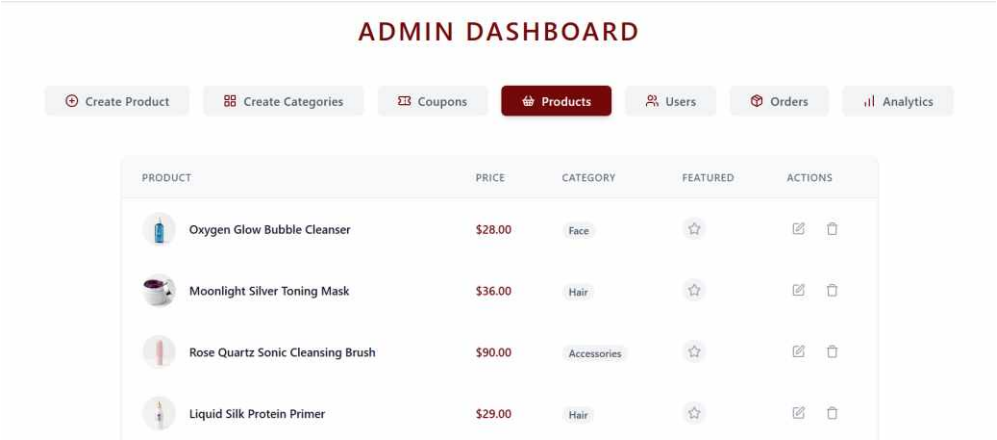


Рисунок 2.28 – Таблиця усіх товарів

Адміністратор може миттєво видалити продукт, змінити його статус на “featured” за допомогою інтерактивного перемикача, або перейти до його модифікації. Натискання на іконку редагування активує роботу компонента <EditProductForm /> (файл src/components/EditProductForm.jsx), який відкриває форму “Edit Product” (див. рис. 2.29). Форма дозволяє змінити будь-яку інформацію про вибрану косметику. Після внесених змін можна зберегти їх у базі даних або скасувати і повернутись до попереднього списку.

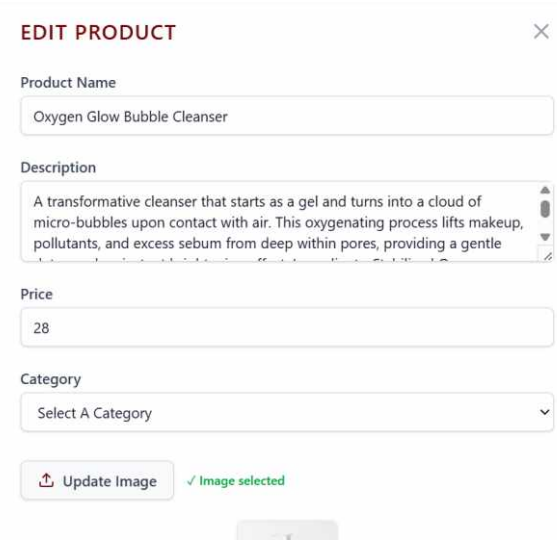


Рисунок 2.29 – Форма редагування товару

Повний лістинг компоненту <EditProductForm /> наведено в додатку Ї кваліфікаційної роботи.

2.4.2 Написання серверної частини

Back-end — це серверна частина сайту/додатку, прихована від очей користувача. Вона відповідає за логіку обробки запитів, збереження та отримання даних, управління обліковими записами, авторизацію, взаємодію з зовнішніми API тощо [26].

Серверна частина вебдодатка реалізована на платформі Node.js із використанням фреймворку Express.js [11], що забезпечує швидку обробку запитів завдяки своїй архітектурі. Для взаємодії з базою даних MongoDB застосовано об’єктно-реляційну бібліотеку (ORM) Mongoose, яка дозволяє чітко описати структуру даних через схеми.

Для розгортання базового каркаса бекенд-платформи в терміналі середовища розробки у корінній директорії проєкту виконується команда ініціалізації нового Node.js додатка, яка створює файл package.json:

npm init -y

Наступним кроком є інсталяція повного пулу необхідних залежностей за допомогою пакетного менеджера npm :

npm install express mongoose dotenv cors cookie-parser jsonwebtoken bcryptjs cloudinary lucide-react

Після цього необхідно самостійно створити головний файл server.js, який є центральним керуючим вузлом бекенду.

Процес початкової ініціалізації та розгортання залежностей за допомогою менеджера npm потребує стабільного підключення до мережі Інтернет. На етапі успішноо завершення кешування модулів у локальну директорію node_modules, уся подальша розробка, архітектурна побудова та локальний запуск сервера можуть здійснюватися в автономному режимі. Для запуску сервера у режимі розробки використовується системна команда npm run dev.

Файл server.js ініціалізує екземпляр додатка Express (const app = express()), зчитує змінні оточення з конфігураційного файлу .env через dotenv.config() та

запускає прослуховування вебпорту. На рівні глобального налаштування додатка підключаються такі проміжні обробники [2]:

– cors() – конфігурує політику доступу між різними доменами, дозволяючи обмін даними із клієнтською частиною з обов’язковою підтримкою сесій;

– express.json({ limit: "10mb" }) – головний парсер, який автоматично читає HTTP-запити в JSON-форматі та перетворює їх у зрозумілі JavaScript-об’єкти;

– express.urlencoded({ limit: "15mb", extended: true }) – парсер даних, який декодує інформацію, що надходить через звичайні форми, підтримуючи розширені вкладені структури;

– cookieParser() – парсер куків, що забезпечує автоматичне зчитування та розшифрування захищених куків користувача для перевірки токенів доступу.

Для розширення функціональних можливостей та забезпечення бізнес-логіки бекенду в проєкті використовуються такі сторонні пакети й утиліти [1]:

– mongoose – забезпечує зручний інтерфейс для виконання операцій створення, читання, оновлення та видалення у базі даних MongoDB на основі моделей;

– bcrypt – реалізовує безпечний алгоритм хешування паролів клієнтів для їхнього захищеного збереження;

– jsonwebtoken – генерує зашифровані токени доступу для авторизації та розмежування прав користувачів;

– cloudinary – надає програмний інтерфейс для завантаження, оптимізації та збереження зображень товарів та категорій у хмарному сховищі;

– dotenv – ізолює конфіденційні дані у зовнішньому конфігураційному файлі .env;

– nodemon – інструмент розробки, який автоматично перезапускає сервер при фіксації змін у вихідному коді, оптимізуючи процес налагодження.

Обробка HTTP-запитів користувачів працює через модульний підхід, використовуючи вбудований клас express.Router(). Замість того, щоб складати код в одному файлі, для кожного логічного сегмента інтернет-магазину створено окремі ізольовані модулі маршрутизації (див. рис. 2.30).

```
app.use("/api/categories", categoryRoutes);
app.use("/api/products", productRoutes);
app.use("/api/auth", authRoutes);
app.use("/api/cart", cartRoutes);
app.use("/api/coupons", couponRoutes);
app.use("/api/payments", paymentRoutes);
app.use("/api/analytics", analyticsRoutes);
app.use("/api/users", userRoutes);
app.use("/api/wishlist", wishlistRoutes);
app.use("/api/orders", orderRoutes);
```

Рисунок 2.30 – Ініціалізація модулів маршрутизації за допомогою методу app.use()

Лістинг файлу server.js поданий в Додатку Й кваліфікаційної роботи.

Усі вони підключаються до головного екземпляра додатка за допомогою методу app.use() із префіксом базового URL-шляху /api:

– /api/auth – реєстрація, авторизаці та сесії клієнтів;

– /api/products, /api/categories - виведення та модифікація каталогу товарів і категорій;

– /api/cart, /api/wishlist, /api/orders – обробка кошика, списку бажаного та фіксація замовлень;

– /api/coupons – логіка застосування промокодів;

– /api/payments – проведення оплати через Stripe;

– /api/analytics – збір аналітики;

– /api/users – адміністрування користувачами.

Кожен підключений модуль маршрутизації працює зі звичайними HTTP-методами, такими як .get(), .post(), .put() та .delete(), які передають виконання бізнес-логіки контролерам.

У файлі auth.route.js (в директорії routes) реалізовано маршрутизатор для обробки запитів, пов’язаних із керуванням сесіями користувачів, автентифікацією та безпекою облікових записів. Цей модуль оперує вбудованим конструктором express.Router() та забезпечує повний функціонал для обробки таких типів запитів:

– POST /signup – ініціює реєстрацію нового користувача в системі з первинним збереженням даних;

– POST /login – виконує авторизацію клієнта з перевіркою логіна та паролю;

– POST /logout – ініціює безпечний вихід із системи стираючи усі сесійні куки;

– POST /refresh-token – виконує автоматичне оновлення сесії за допомогою оновлення access токена, без необхідності повторного введення паролю;

– POST /forgot-password і POST /reset-password – відпоідають за відновлення доступу до системи у випадку втрати пароля;

– POST /google – інтегрує швидкий альтернативний логін через Google Auth;

– GET /profile – повертає дані профілю поточного користувача. Цей маршрут захищений protectRoute, який спочатку перевіряє, чи дійсна сесія, і лише потім відкриває доступ;

– PUT /users/block/:id – адміністративний ендпоінт. Він блокує або розблоковує користувача ча його унікальним ідентифікатором. Цей маршурт захищений двома маршрутами: protectRoute та adminRoute, перевіряючи права адміністратора.

Лістинг файлу auth.route.js поданий в Додатку К кваліфікаційної роботи.

Наступним критично важливим компонентом REST API є модуль обробки каталогу товарів та управління асортиментом інтернет-магазину, реалізований у файлі product.route.js у папці routes. Його архітектура чітко розділена на публічну зону (доступну для всіх відвідувачів сайту) та захищену зону адміністратора. До публічної зони належать GET-запити, які не потребують авторизації й забезпечують виведення інформації на клієнтській частині:

– GET / та GET /category/:category – завантаження повного переліку товарів або їх фільтрація за конкретною категорією косметики;

– GET /featured та GET /recommendations – отримання списку бестселерів для головної сторінки та списку рекомендованих для блока персональних рекомендацій;

– GET /categories – динамічне отримання унікальних категорій, які є в базі даних;

– GET /max-price – визначення максимальної вартості продукту для коректного налаштування фільтрів цінового діапазону на фронтенді.

Захищена зона адміністрування товарів вимагає обов'язкового проходження фільтрів безпеки protectRoute та adminRoute й містить такі ендпоінти:

– POST / – ініціює створення та додавання нової позиції до каталогу через метод createProduct (робота форми <CreateProductForm />);

– PUT /:id – відповідає за повне оновлення інформації про товар за допомогою контролера updateProduct (робота форми <EditProductForm />);

– PATCH /:id – викликає метод toggleFeaturedProduct для миттєвої зміни статусу продукту на рекомендований та навпаки;

– DELETE /:id – реалізує функціонал безпосереднього видалення продукту з каталогу та бази даних.

Лістинг файлу product.route.js наведено в додатку Л кваліфікаційної роботи.

Компонентом у структурі REST API, який відповідає за фіксацію фінансових транзакцій, моніторинг покупок та керування логістичними етапами, є модуль маршрутизації замовлень, реалізований у файлі order.route.js директорії routes. Його архітектурна логіка забезпечує взаємодію клієнта зі своєю історією покупок, а також надає адміністратору розширені інструменти для менеджменту продажів.

У структурі файлу чітко розмежовано клієнтський та адміністративний функціонал:

– GET /my-orders – користувацький маршрут, необхідний для завантаження з бази даних історії лише його особистих покупок для відображення в особистому кабінеті;

– GET /all – захищений адміністративний ендпоінт, який викликає метод getAllOrders для генерації повної звітності та виведення таблиці усіх замовлень в інтернет-магазині;

– PATCH /:id/status – адміністративний маршрут для швидкої модифікації статусу доставки конкретного замовлення.

Лістинг файлу order.route.js поданий в Додатку О кваліфікаційної роботи.

Взаємодія між розробленим Express-сервером та СКБД MongoDB реалізована через інструмент об’єктного моделювання Mongoose. Оскільки NoSQL-база даних за замовчуванням є безсхемною, використання цієї бібліотеки дозволило впровадити строгу структуризацію документів.

У межах побудови архітектури бекенду вебплатформи ключовими інструментами Mongoose виступають:

– асинхронне підключення до хмарного кластера за допомогою методу connect(), винесеного в окремий сервісний модуль src/lib/db.js;

– проєктування внутрішнього каркаса колекцій через конструктор new Schema(), де визначаються правила поведінки полів, їхня обов’язковість та зв’язки між сутностями;

– генерація активних моделей через метод model(), що дозволяє здійснювати прямі маніпуляції та CRUD-запит до бази даних.

Схеми та моделі даних визначені у таких файлах:

– models/user.model.js – призначений для запису у базу даних інформації про користувачів, а також збереження їхнього кошика та списку бажань;

– models/product.model.js – призначений для збереження детальної інформації про товари інтернет-магазину;

– models/category.model.js – призначений для запису у базу даних категорій товарів;

– models/order.model.js – призначений для запису у базу даних інформації про замовлення;

– models/coupon.model.js – призначений для збереження інформації про купони.

Повний програмний код зазначених моделей даних представлений у Додатку М кваліфікаційної роботи.

2.5 Тестування вебсайту

Будь-який програмний продукт, незалежно від складності і архітектури, перед релізом та представленням кінцевим користувачам повинен пройти комплексну перевірку на працездатність, відсутність критичних помилок та відповідність критеріям надійності. Процес верифікації інтернет-магазину косметики «BeautyUp» відбувався в кілька етапів: автоматизоване модульне й інтеграційне тестування коду, ручне функціональне тестування ключових модулів методом «чорного ящика», а також залучення сторонніх користувачів для отримання реального користувацького досвіду та оцінки візуальної стилістики сайту.

Сучасний підхід до розробки вебдодатків включає автоматизовані тести, які дозволяють запобігти регресії коду. Клієнтська частина інтернет-магазину написана на основі бібліотеки React. Щоб протестувати окремі частини інтерфейсу, компоненти, стани та логіки маршрутизації, було використано сучасне середовище розробки Vitest у поєднанні з утилітами бібліотеки React Testing Library.

Unit тести — це автоматичні тести, які перевіряють невеликі частини коду, такі як функції або методи, ізольовано від решти системи. Вони дають змогу розробникам переконатися, що кожна частина коду працює правильно і відповідає очікуваній поведінці [28]. Щоб перевірити базову працездатність клієнтського інтерфейсу було реалізовано тест головного компонента системи <App /> (файл frontend/src/App.test.jsx). Повний лістинг коду зазначеного модульного тесту клієнтської частини наведено в Додатку Н кваліфікаційної роботи.

У реалізованому сценарії за допомогою функції render віртуально монтується додаток, обгорнутий в ізольований маршрутизатор MemoryRouter. Оскільки компоненти взаємодіють з глобальними сховищами Zustand, у тесті застосовано механізм створення функцій-заглушок. Програма тестування здійснює пошук навігаційного елемента Catalog та верифікує його наявність у документі за допомогою toBeInTheDocument.

Для перевірки логіки серверної частини на Express.js, було організовано інтеграційне тестування API-ендоінтів (файл backend/server.test.js). На відміну від юніт-тестів, інтеграційні показують як працюють маршрути, проміжні програмні забезпечення middleware та конфігурації оброки запитів. Інтеграційне тестування сервера реалізовано за допомогою бібліотеки Supertest, яка дозволяє надсилати віртуальні HTTP-запити до створеного Express-додатка та аналізувати структуру відповідей.

Приклад реалізованого інтеграційного тесту для серверної частини поданий у Додатку О кваліфікаційної роботи. Тестовий сценарій виконує симуляцію GET-запиту на адресу /api/categories. Інструмент supertest перевіряє три критичні відповіді: статус-код (повинно бути 200 OK), відповідність заголовка типу контенту JSON-формату, а також перевіряє, чи повернуте тіло відповіді є структурованим масивом даних.

Запуск всього пакету автоматизованих тестів здійснюється з кореня проєкту за допомогою команди:

npm run test

Виконання команди примусово ініціює запуск Vitest у середовищі jsdom. Успішний результат проходження обох сценаріїв відображається у вікні термінала (див. рис. 2.31).

Рисунок 2.31 - Результат успішного виконання тестів

Як показано на рисунку 2.31, обидва файли тестів пройшли валідацію – Test Files: 2 passed. Це підтверджує стабільність архітектурного каркаса як серверної, так і клієнтської частин магазину косметики.

Після підтвердження стабільності коду на системному рівні було проведено комплексне ручне тестування готового програмного продукту. При ручному

аналізі використовувався метод «чорного ящика» - це метод тестування програмного забезпечення, який перевіряє функціональність системи без доступу до її внутрішньої структури коду. У цьому підході тестувальник працює як кінцевий користувач, оцінюючи, чи правильно система реагує на вхідні дані [32].

Першочергово перевірці піддалися модулі автентифікації, розмежування прав доступу та взаємодії клієнтської частини з базою даних. Тест розпочався з форми реєстрації нового користувача. Для цього було здійснено перехід на сторінку реєстрації сайту та заповнено всі необхідні інтерактивні поля, як зображено на рисунку 2.32

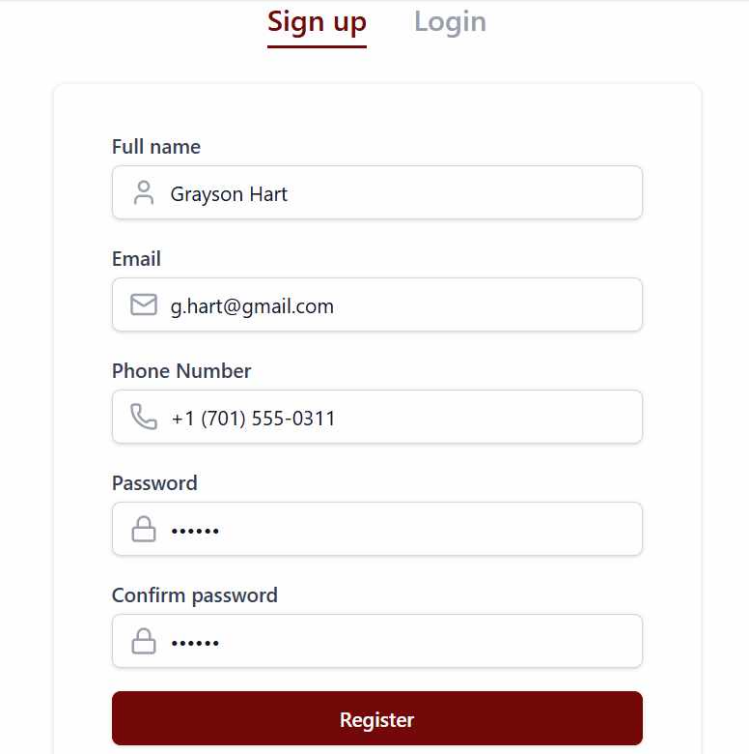


Рисунок 2.32 - Заповнення форми реєстрації нового клієнта

Після натискання кнопки “Register” аккаунт успішно створено, дані відправляються на Express-сервер, пароль шифрується за допомогою алгоритму bcrypt, а користувач заноситься до бази даних. Інтеграція з бекендом працює, новий об’єкт записується в хмарне середовище MongoDB (див. рис. 2.33).

Рисунок 2.33 - Підтвердження збереження даних користувача в БД MongoDB

Оскільки тестування реєстрації пройшло успішно і дані коректно зафіксувалися в системі, аналогічним ручним способом було перевірено весь інший функціонал інтернет-магазину.

Зокрема, проведено тестування форми авторизації, де після введення облікових даних система записує захищений JWT-токен у cookies браузера та перенаправляє покупця на головний екран. Валідація механізму захищених маршрутів підтвердила, що авторизований користувач автоматично блокується від повторного переходу на сторінки /login чи /signup через URL-рядок.

Останнім кроком перевірки сайту стало залучення незалежних користувачів для проведення бета-тестування. Тестувальники виконували стандартний сценарій покупця інтернет-магазину: здійснювали пошук товарів в каталозі, користувались фільтрами, додавали товари в кошик, змінювали кількість позицій та проходили етап імітації оплати через систему Stripe.

Таким чином, на основі результатів усіх проведених етапів тестування було зроблено остаточний висновок про абсолютно коректну, стабільну та надійну роботу всіх модулів і функцій розробленого вебсайту.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Інструкція з розміщення сайту в Інтернеті

На даний момент у нас вже є протестований веб-застосунок для інтернет-магазину косметики «BeautyUp». Проєкт розроблено як дворівневий додаток, де фронтенд-частина реалізована за допомогою React, а бекенд-платформа функціонує на Express. Оскільки ці дві складові є окремими структурними одиницями проєкту, їхній безперебійний зв'язок координується проміжним програмним забезпеченням cors.

Публічне розміщення вебсайту в мережі реалізовано через хмарний сервіс Render. Обраний майданчик дозволяє легко розмістити статичні файли і динамічні служби, підтримуючи безперервну інтеграцію для швидкого оновлення та захищає з’єднання за допомогою протоколу шифрування HTTPS.

Перед розміщенням на хостингу весь початковий код проєкту необхідно завантажити у віддаллений репозиторій на платформі GitHub, що дозволить платформі Render автоматично зчитувати код та оновлювати сайт при внесенні будь-яких майбутніх змін. Для успішного запуску сайту адміністратору необхідно здійснити наступні кроки:

1. Перейти на офіційний сайт платформи за адресою render.com та пройти реєстрацію. Найкраще обрати авторизацію через існуючий профіль GitHub.
2. У головній панелі керування “Dashboard” натиснути кнопку “New +” у верхньому правому кутку, де у випадаючому меню натиснути на “Web Service”.
3. Підключити свій репозиторій із кодом вебдодатку, обравши його із списку існуючих репозиторіїв зареєстрованого облікового запису GitHub.
4. У формі налаштувань необхідно вказати такі параметри:

Name: project-beautyupstore

Language: Node

Branch: main

Region: Ohio (US East)

Build Command: npm run build

Start Command: npm run start

5. Для захисту конфіденційних даних та налаштування зв'язку всередині платформи, у вкладці “Environment Variables” для створеної вебслужби необхідно додати змінні, які раніше зберігалися у файлі .env.

6. Після цього потрібно натиснути кнопку “Deploy Web Service” і зачекати (див. рис. 3.1), поки Render згенерує для сервера унікальне посилання: https://project-beautyupstore.onrender.com

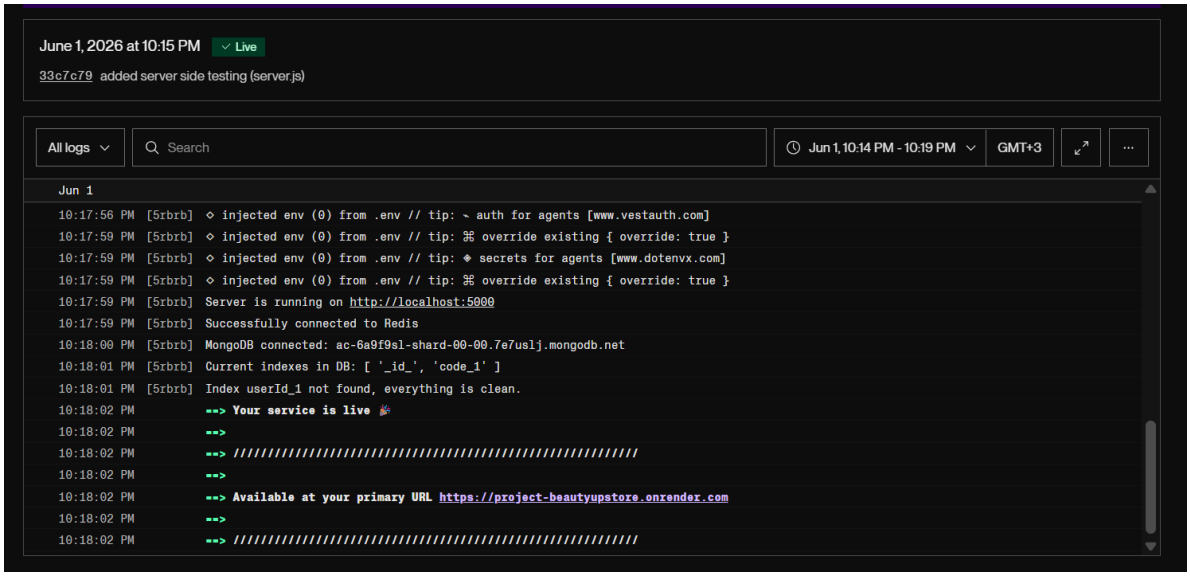


Рисунок 3.1 – Успішне розгортання додатку в середовищі Render

Завдяки монолітному підходу для production-середовища, де Express-сервер безпосередньо обслуговує клієнтську частину React, весь додаток успішно розгорнуто в межах однієї вебслужби на Render. Пряма інтеграція хостингу з репозиторієм автоматично забезпечує роботу циклу безперервної інтеграції та розгортання. Це означає, що завантаження будь-яких стабільних змін у код ініціює самостійний запуск команд npm run build та npm run start, відразу оновлюючи вебсайт для клієнтів та без зупинки роботи інтернет-магазину. Така архітектурна оптимізація значно знижує витрати на хмарну інфраструктуру, усуваючи необхідність платити за окремі хостингові платформи для фронтенду та

бекенду. Крім того, автоматизація процесів CI/CD мінімізує вплив людського фактора під час релізів та спрощує подальше обслуговування платформи.

3.2 Інструкція з обслуговування та наповнення сайту

Процес подальшого обслуговування та наповнення інтернет-магазину косметики розроблений таким чином, щоб поточна робота з асортиментом товарів, категоріями, замовленнями та аналітикою не вимагала від власника чи менеджера суттєвих технічних знань. Подальше візуальне та контентне наповнення сайту не буде радикально відрізнятись від поточного вигляду, адже будь-яка інша глобальна зміна структури вимагала б безпосереднього втручання в початковий програмний код вебдодатка.

Для уникнення додаткового залучення IT-спеціалістів та автоматизації адміністрування, у системі реалізовано інтерактивну панель керування Admin Dashboard. Завдяки цьому інструменту користувач з відповідним рівнем доступу може самостійно виконувати усі операції з управління контентом через зручний та простий у користуванні графічний інтерфейс.

Для того щоб мати можливість здійснювати вміст сайту, обліковий запис повинен володіти правами адміністратора. Зміна ролі звичайного клієнта(user) на адміністратора (admin) реалізується вже існуючим модератором системи за таким алгоритмом:

1. Необхідно увійти у систему та перейти у меню «Dashboard» (на мобільних пристроях відображається як «Admin Dashboard»).

2. У навігаційному меню панелі необхідному вибрати вкладку “Users”.

3. У таблиці користувачів знайти потрібну особу та у стовпці “Actions” натиснути кнопку-перемикач ролі.

Після цього користувач миттєво набуває адміністративних прав і отримує повний доступ до керування контентом магазину.

Для підтримання актуальності каталогу косметичних засобів адміністратору доступні модулі створення, видалення, моніторингу та модифікації позицій:

– Керування продуктами, а саме створення нових товарів відбувається через інтерактивну форму додавання (див. рис. 2.27). Моніторинг, редагування та видалення вже наявних товарів здійснюється у загальній базі на вкладці “Products” (див. рис. 2.28);

– Управління категоріями, тобто додавання нових або видалення вже неактуальних категорій відбувається на вкладці “Create Categories”. Нові категорії реєструються в базі даних та динамічно відображаються на головній сторінці сайту.

Окрім роботи з каталогом товарів, адміністративна панель має інструменти для ведення бізнес-логіки та маркетингу:

– Керування купонами, в конкретності створення та видалення унікальних промокодів відбувається на вкладці “Coupons”;

– Обробка замовлень, де надається можливість моніторингу усіх транзакцій покупців, контактних даних, адрес доставки та ручне керування статусами замовлень відбувається на вкладці “Orders”;

– Аналітика, тобто візуалізація даних про роботу інтернет-магазину в реальному часі для оцінки ефективності майданчика відбувається на вкладці “Analytics”.

3.3 Інструкція з популяризації та підтримки сайту

Після успішного розгортання інтернет-магазину косметики у глобальній мережі, перед адміністратором постають завдання з забезпечення стабільної відвідуваності ресурсу та підтримки його технічної працездатності й безпеки.

Для залучення нових клієнтів та підвищення позицій сайту в пошукових системах адміністратору необхідно координувати такі процеси:

– пошукова оптимізація(SEO), а саме, при додаванні нових товарів та категорій через адмін-панель важливо вносити унікальні, інформативні описи та назви, що містять ключові запити, які користувачі можуть використовувати при

пошуку. Фотографії товарів повинні бути стиснутими, щоб не сповільнювати завантаження сторінок;

– маркетингові інструменти сайту, щоб утримати людей та стимулювати їх купувати знову, адміністратор повинен активно використовувати вбудований модуль купонів, генеруючи святкові промокоди на знижку, а також виносити найпопулярніші позиції на головну сторінку, ставлячи їм статус isFeatured: true;

– інтеграція аналітики, аналіз цільової аудиторії та розширення каталогу, де на основі даних із вкладки “Analytics” адміністратор може відстежувати динаміку продажів та поведінку користувачів. Це дозволить швидко реагувати на актуальні тренди б’юті-ринку, розширювати асортимент популярних товарів та створювати нові категорії косметичних засобів, адаптуючи контент платформи під змінний попит покупців.

Для забезпечення безперервної роботи вебдодатка власник інтернет-магазину повинен виконувати регулярний технічний нагляд на трьома ключовими вузлами інфраструктури:

– контроль стану сервера, тобто адміністратору необхідно періодично здійснювати перевірку панелі керування Render (вкладка Logs та Events). Завдяки вбудованому CI/CD, технічна підтримка самого коду автоматизована, тобто будь-які оновлення інтерфейсу, завантажені в репозиторій GitHub, застосовуються самостійно без зупинки роботи магазину;

– моніторинг бази даних, хоча сховище даних працює в хмарі автоматично, власнику вебсайту рекомендується періодично перевіряти рівень заповнення дискового простору та вмикати автоматичне резервне копіювання в інструментарії Atlas для запобігання втрати інформації про користувачів та замовлення;

– безпека транзакцій, оскільки обробка платежів та введення даних банківських даних повністю винесені на сторону сертифікованого шлюзу Stripe, в адміністратора немає потреби зберігати конфіденційні платежі на своєму сервері. Головним завданням є стежити за актуальністю секретних ключів у змінних оточення сервісу Render та не передавати їх третім особам.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

Метою економічного розділу кваліфікаційної роботи є комплексне фінансово-економічне обґрунтування доцільності створення інтернет-магазину «BeautyUp», визначення загального обсягу капіталовкладень у проєкт та доведення його комерційної життєздатності на ринку. Проведені розрахунки виступають підґрунтям для прийняття остаточного рішення щодо доцільності подальшого фінансування, запуску та практичного впровадження даного програмного продукту, або ж підтвердження фінансових ризиків його розробки.

Для розрахунку вартості НДР необхідно виконати наступні етапи:

– описати технологічний процес розробки із зазначенням трудомісткості кожної операції;

– визначити суму витрат на оплату праці основного і допоміжного персоналу, включаючи відрахування на соціальні заходи;

– визначити суму матеріальних затрат;

– обчислити витрати на електроенергію для науково-виробничих цілей;

– розрахувати транспортні витрати;

– нарахувати суму амортизаційних відрахувань;

– визначити суму накладних витрат;

– скласти кошторис та визначити собівартість НДР;

– розрахувати ціну НДР;

– визначити економічну ефективність та термін окупності продукту.

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР

З метою точного визначення тривалості науково-дослідного процесу, витрати часу на виконання окремих технологічних операцій було узагальнено й систематизовано у таблиці 4.1.

Таблиця 4.1 - Середній час виконання НДР та стадії (операції) технологічного процесу

№ п/п	Назва операції (стадії)	Виконавець	Середній час виконання операції, год.
1	Підготовка	Керівник проєкту	8
2	Розробка алгоритму вебдодатку	Веб-розробник	10
3	Написання текстів вебдодатку	Веб-розробник	72
4	Тестування вебдодатку	Лаборант	14
5	Розміщення вебдодатку на хостингу	Веб-розробник	6
6	Створення інструкції по експлуатації веб-додатку	Лаборант	6
	Р а з о м	–	116

Згідно з проведеними розрахунками, сумарний час, необхідний для виконання всіх етапів розробки інтернет-магазину, становить 116 годин.

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

Відповідно до Закону України “Про оплату праці” заробітна плата – “це винагорода, обчислена, як правило, у грошовому виразі, яку за трудовим договором роботодавець виплачує працівникові за виконану ним роботу”.

Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно-ділових якостей працівника, результатів його праці та господарської діяльності підприємства.

Головна складова оплати праці базується на встановлених окладах чи тарифних нормативах за відпрацьований час і не пов'язана з поточною прибутковістю або загальними господарськими показниками підприємства.

Основна заробітна плата розраховується за формулою:

$$З_{осн.} = Т_с \cdot К_г, \quad (4.1)$$

де $Т_с$ – годинна тарифна ставка працівника, грн/год. (приймаємо для керівника проєкту – 175 грн/год, веб-розробника – 125 грн/год, лаборанта – 75 грн/год). Вихідні дані щодо вартості годинних ставок спеціалістів відповідної кваліфікації були визначені на основі аналізу актуальних вакансій та статистичних звітів про заробітні плати в IT-секторі, взятих із профільного офіційного вебпорталу DOU [10].;

$К_г$ – кількість відпрацьованих годин.

$$З_{осн.} = 175*8+125*88+75*20=13900 \text{ грн.}$$

Додаткова заробітна плата це винагорода за працю понад установлені норми, за трудові успіхи та винахідливість і за особливі умови праці. Вона включає доплати, надбавки, гарантійні і компенсаційні виплати, передбачені чинним законодавством; премії, пов'язані з виконанням виробничих завдань і функцій.

Величина додаткової оплати праці розраховується у межах 10–15 % від сформованого фонду основної заробітної плати працівників.

$$З_{дод.} = З_{осн.} \cdot К_{опл.}, \quad (4.2)$$

де $К_{опл.}$ – коефіцієнт додаткових виплат працівникам.

$$З_{дод.} = 13900*0,15=2085 \text{ грн.}$$

Таким чином, загальний обсяг витрат на оплату праці ($В_{о.п.}$) обчислюється за формулою:

$$В_{о.п.} = З_{осн.} + З_{дод.}, \quad (4.3)$$

$$В_{о.п.} = 13900+2085=15985 \text{ грн.}$$

Окрім прямих витрат на оплату праці, фінансовий план проєкту враховує нарахування єдиного соціального внеску в розмірі 22%. У зв'язку з цим, підсумковий обсяг відрахувань на соціальні заходи визначається так:

$$В_{с.з.} = ФОП \cdot 0,22, \quad (4.4)$$

де *ФОП* – фонд оплати праці, грн.

$$B_{с.з.} = 15985*0,22=3516,70 \text{ грн.}$$

Отримані результати розрахунків витрат на оплату праці та соціальні нарахування систематизовано й представлено у таблиці 4.2.

Таблиця 4.2 - Зведені розрахунки витрат на оплату праці

№ п/п	Категорія працівників	Основна заробітна плата, грн.			Додатко-ва заробітна плата, грн.	Нарах. на ФОП, грн.	Всього витрати на оплату праці, грн.
		Тарифна ставка, грн.	К-сть від-працьов. год.	Фактично нарах. з/пл., грн.			
А	Б	1	2	3	4	5	6
1	Керівник проєкту	175	8	1400,00	210,00	-	-
2	Розробник	125	88	11000,00	1650,00	-	-
3	Лаборант	75	20	1500,00	225,00	-	-
	Разом	-	-	13900,00	2085,00	3516,70	19501,70

4.3 Розрахунок матеріальних витрат

Загальна величина матеріальних витрат визначається як добуток кількості спожитих матеріалів та ціни їх придбання:

$$M_{Bi} = q_i \cdot p_i \,, \qquad (4.5)$$

де q_i – кількість витраченого матеріалу і-го виду;

p_i – ціна матеріалу і-го виду.

Звідси, можна визначити загальну суму матеріальних витрат:

$$З_{м.в.} = \Sigma M_{Bi} \qquad (4.6)$$

$$З_{м.в.} = 400+280+280=960 \text{ грн.}$$

Проведені розрахунки занесемо у таблицю 4.3.

Таблиця 4.3 - Зведені розрахунки матеріальних витрат

№ п/п	Найменування матеріальних ресурсів	Од. виміру	Факт. витр. матеріалів	Ціна 1-ці, грн.	Заг. сума витрат, грн.
1	Флеш пам'ять USB Kingston DataTraveler Exodia 64GB USB 3.2 Gen1	шт	1	400	400,00
2	Друк документації	шт	140	2	280,00
3	Друк плакатів	шт	4	70	280,00
	Р а з о м				960,00

Отже, загальна сума матеріальних витрат на матеріальні ресурси становить 960 грн.

4.4 Розрахунок витрат на електроенергію

Затрати на електроенергію 1-ці обладнання визначаються за формулою:

$$З_е = W \cdot T \cdot S, \quad (4.7)$$

де W – необхідна потужність, кВт;

T – кількість годин роботи обладнання;

S – вартість кіловат-години електроенергії.

Повний цикл розробки та впровадження інтернет-магазину здійснювався з використанням одного ноутбука. Розрахунок вартості спожитої ним електроенергії базується на загальній тривалості робочого процесу (згідно табл. 4.1) та номінальній потужності зарядного пристрою комп'ютера, яка становить 0,15 кВт.

$$З_е = 0{,}15 * 116 * 15{,}94 = 277{,}36 \text{ грн.}$$

4.5 Визначення транспортних затрат

Транспортні витрати слід прогнозувати у розмірі 8–10 % від загальної суми матеріальних затрат.

$$T_в = З_{м.в.} \cdot 0{,}08 \ldots 0{,}1, \quad (4.8)$$

де $T_В$ – транспортні витрати.

$$T_в = 960 * 0{,}08 = 76{,}80 \text{ грн}$$

4.6 Розрахунок суми амортизаційних відрахувань

Характерною особливістю застосування основних фондів у процесі виробництва є їх відновлення. Для відновлення засобів праці у натуральному виразі необхідне їх відшкодування у вартісній формі, яке здійснюється шляхом амортизації.

Амортизація – це процес перенесення вартості основних фондів на вартість новоствореної продукції з метою їх повного відновлення.

Для визначення амортизаційних відрахувань застосовуємо формулу:

$$A = \frac{Б_В \cdot Н_А}{100\%}, \quad (4.9)$$

де A – амортизаційні відрахування за звітний період, грн.;

$Б_В$ – балансова вартість групи основних фондів на початок звітного періоду, грн.;

$Н_А$ – норма амортизації, %.

Технічне забезпечення процесів розробки та розгортання інтернет-магазину здійснювалося за допомогою одного ноутбука (вартість якого становить 14 700 грн). Загальний час експлуатації даного комп'ютерного обладнання в межах проєкту склав 116 годин.

$$A = 14700 * 0{,}04 * 116 / 150 = 454{,}72 \text{ грн.}$$

4.7 Обчислення накладних витрат

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління підприємства та створення необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого суб'єкта, накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників.

$$H_в = B_{о.п.} \cdot 0{,}2 \ldots 0{,}6, \quad (4.10)$$

де H_B – накладні витрати.

$$H_B = 15985 * 0{,}2 = 3197 \text{ грн.}$$

4.8 Складання кошторису витрат та визначення собівартості НДР

Результати проведених вище розрахунків зведено у таблицю 4.4.

Таблиця 4.4 - Кошторис витрат на НДР

Зміст витрат	Сума, грн.	В % до заг. суми
Витрати на оплату праці (основну і додаткову заробітну плату)	15985,00	65,33
Відрахування на соціальні заходи	3516,70	14,37
Матеріальні витрати	960,00	3,92
Витрати на електроенергію	277,36	1,13
Транспортні витрати	76,80	0,31
Амортизаційні відрахування	454,72	1,86
Накладні витрати	3197,00	13,08
Собівартість	24467,58	100,00

Собівартість ($С_В$) НДР розрахуємо за формулою:

$$С_в = B_{о.п.} + B_{с.з.} + З_{м.в.} + З_е + Т_в + А + Н_в \quad (4.11)$$

$$С_в = 15985 + 3516{,}70 + 960 + 277{,}36 + 76{,}80 + 454{,}72 + 3197 = 24467{,}58 \text{ грн.}$$

4.9 Розрахунок ціни НДР

Ціну НДР можна визначити за формулою:

$$Ц = С_в \cdot (1 + Р_{рен.}) \cdot (1 + ПДВ), \quad (4.12)$$

де $Р_{рен.}$ – рівень рентабельності, 25 %;

ПДВ – ставка податку на додану вартість, (20 %).

$$Ц=24467,58*(1+0,25)*(1+0,2)= 36701,36 \text{ грн.}$$

4.10 Визначення економічної ефективності і терміну окупності капітальних вкладень

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Комплексна оцінка фінансової ефективності та доцільності впровадження програмного продукту здійснюється шляхом розрахунку чистої теперішньої вартості за формулою 4.13, а також визначення терміну окупності інвестованого капіталу за формулою 4.14.

$$ЧТВ= -К_В+\sum_{i=1}^{t}\frac{Г_П}{(1+i)^t}, \quad (4.13)$$

де $К_В$ – затрати на проект;

$Г_п$ – грошовий потік за t – ий рік;

t – відповідний рік проекту;

i - величина дисконтної ставки (10…15%).

Якщо ЧТВ ≥ 0, то проект може бути рекомендований до впровадження.

$$ЧТВ= -24467,58+12233,79/(1+0,15)+12233,79/(1+0,15)^2+12233,79/(1+0,15)^3=$$

$$-24435,18+10638,08+9250,50+8043,91=3464,92 \text{ грн.}$$

Термін окупності визначається за формулою:

$$Т_{ок}= Т_{ПВ}+\frac{Н_В}{Г_{ПР}} \quad (4.14)$$

де $Т_{ПВ}$ – період до повного відшкодування витрат, років;

$Н_В$ – невідшкодовані витрати на початок року, грн.;

$Г_{ПР}$ – грошовий потік на початок року, грн.

$$Т_{ОК}=2+4578,98/12233,79=2,4$$

Всі дані розрахунків внесемо в зведену таблицю 4.5 техніко-економічних показників.

Таблиця 4.5 -Техніко-економічні показники НДР

№ п/п	Показник	Значення
1	Собівартість, грн.	24467,58
2	Плановий прибуток, грн.	12233,79
3	Ціна, грн.	36701,36
4	Чиста теперішня вартість, грн	3464,92
5	Термін окупності, рік	2,4

Враховуючи основні економічні показники, зведені у таблицю, можна зробити висновок, що при терміні окупності – 2,4 року проводити роботи по впровадженню даного вебпроєкту є доцільним та економічно вигідним. Як можна побачити із розрахунків, основними є витрати на оплату праці розробників. Тому, з метою зниження собівартості, у майбутньому варто автоматизувати процеси розробки через CI/CD інструменти.

5 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ

5.1 Управління режимами праці та відпочинку для зниження психоемоційного напруження ІТ-фахівців

Специфіка професійної діяльності фахівців у сфері інформаційних технологій характеризується високим рівнем інтелектуального та психоемоційного навантаження. Розробка сучасних програмних продуктів супроводжується тривалою роботою за персональним комп’ютером, необхідністю обробки великих обсягів інформації, високою концентрацією уваги, а також жорсткими часовими обмеженнями. Усі ці фактори створюють передумови для виникнення хронічної втоми, стресу, зниження працездатності та розвитку професійного вигорання.

Згідно з нормативно-правовими актами України з охорони праці, раціональна організація режимів праці та відпочинку є базовим превентивним заходом для збереження здоров'я та зниження психоемоційного напруження персоналу.

Протягом робочої зміни, залежно від категорії важкості та напруженості праці, необхідно встановлювати регламентовані перерви. Для розробників програмного забезпечення, чия діяльність належить до категорії з високим рівнем зорового та нервово-емоційного напруження, сумарна тривалість перерв має становити не менше 50–60 хвилин за 8-годинну зміну.

Впроваджуючи перші кроки до балансу між ефективною працею та відновленням ресурсів організму, фахівцю необхідно опиратися на базові індивідуальні організаційні заходи:

– створити чіткий графік початку та завершення робочого дня;

– встановити час для перерв, обіду та руху (наприклад, 5 хвилин розминки кожні 45 хвилин);

– регулярно спілкуватися з колегами або одногрупниками, навіть якщо немає "робочих" тем – це знижує рівень ізоляції;

– вимикати сповіщення на ніч, не перевіряти пошту після 19:00;

– вести щоденник стану: це допомагає помічати накопичення напруги до того, як вона стане проблемою [19].

Ефективність регламентованих перерв значно підвищується, якщо вони використовуються не для пасивного відпочинку, наприклад, перегляду соціальних мереж на тому ж екрані, а для активної зміни мікросередовища. Для зниження психоемоційного напруження та профілактики соматичних розладів під час перерв рекомендується виконувати комплекси вправ:

– офтальмотренаж, тобто гімнастика для очей для зняття спазму акомодації;

– фізична розминка, яка необхідна для зняття статики з м'язів шиї та спини;

– психологічне розвантаження, тобто різноманітні дихальні техніки.

Окрім особистої дисципліни, на працездатність розробника критично впливають параметри виробничого середовища. Оптимальні мікрокліматичні умови – це таке поєднання кількісних показників мікроклімату, які при тривалій і систематичній дії на людину забезпечують збереження нормального теплового стану організму без напруження механізмів терморегуляції. Вони забезпечують почуття теплового комфорту і створюють передумови для високого рівня працездатності. Людина працездатна і гарно себе почуває, якщо температура навколишнього повітря знаходиться у межах 18-20 °С, відносна вологість – 40-60%, а швидкість руху повітря – 0,1-0,2 м/с [21].

У сучасних реаліях, коли значна частина ІТ-фахівців працює у дистанційному або гібридному режимі, виникає додатковий ризик розмивання кордонів між робочим та особистим часом. Для запобігання постійним перепрацюванням необхідно впроваджувати правила «цифрової гігієни» та поважати право працівника на відпочинок. Це передбачає вирішення всіх робочих питань виключно в межах робочого дня та чітке планування обсягу завдань, що дозволяє повністю уникати систематичної роботи в понадурочний час.

Таким чином, впровадження науково обґрунтованого режиму праці та відпочинку, підтримання оптимальних параметрів мікроклімату та захист особистого часу від позаробочих цифрових подразників дозволяють зберегти високу продуктивність праці ІТ-фахівців, мінімізувати кількість помилок у

програмному коді та забезпечити безпечні умови життєдіяльності під час виконання професійних обов'язків.

5.2 Профілактика зорової втоми та нормалізація освітлення при тривалій роботі з відеодисплеями.

Процес розробки та тестування інтерфейсу інтернет-магазину пов'язаний із безперервним зоровим контролем інформації на екрані відеодисплея. Тривала робота з екранними пристроями висуває підвищені вимоги до зорової системи розробника. Головними чинниками, що викликають зорову втому та розвиток комп'ютерного зорового синдрому, є:

- постійне фокусування погляду на близькій відстані;
- мерехтіння екрана;
- наявність відблисків;
- невідповідність параметрів освітлення робочої зони нормативним вимогам.

Для запобігання зоровому перевтомленню першочергову увагу слід приділити нормалізації природного та штучного освітлення в приміщенні, де виконуються роботи. Відповідно до державних санітарних норм, освітлення має бути рівномірним, не створювати різких тіней та прямих чи відбитих відблисків на екрані монітора.

Природне освітлення має здійснюватися через віконні прорізи, бажано з боковим лівим розташуванням щодо робочого місця розробника. Це дозволяє уникнути падіння прямого сонячного світла на екран комп'ютера та в очі працівника. Для регулювання світлового потоку в сонячні дні вікна необхідно обладнувати сонцезахисними пристроями, такими як жалюзі або щільні штори.

Штучне освітлення в приміщенні з відеодисплеями має бути комбінованим і складатися із загального та місцевого освітлення. Організація лише місцевого освітлення робочого столу суворо заборонена, оскільки різкий контраст між

яскраво освітленим документом чи клавіатурою та темною кімнатою викликає швидку втому очей. Нормовані значення освітленості мають становити:

– на поверхні робочого столу в зоні розміщення документів – не менше 300-500 лк;

– на поверхні екрана монітора – не більше 300 лк, щоб уникнути засвічування та втрати контрастності.

Як джерела світла для загального освітлення слід використовувати люмінесцентні лампи або сучасні світлодіодні світильники з тепло-білим або природно-білим спектром випромінювання. Світильники загального освітлення мають розташовуватися у вигляді суцільних або переривчастих ліній паралельно лінії зору розробника, що мінімізує блікування екрана.

Окрім параметрів освітлення, важливу роль у профілактиці зорової втоми відіграє правильне розташування монітора ноутбука та організація робочого простору:

1. Відстань до екрана, а саме оптимальною відстанню від очей користувача до дисплея повинна становити 60–70 см, але не менше 50 см.

2. Кут зору, тобто екран монітора слід розташовувати так, щоб його верхня кромка знаходилася на рівні очей або трохи нижче. Напрямок погляду на центр екрана має бути спрямований зверху вниз під кутом 15–20 градусів.

3. Розташування відносно джерел світла, тобто монітор не повинен розвертатися екраном до вікна або задньою панеллю до вікна. Оптимально — кут розвороту столу 90 градусів до вікна.

Для індивідуальної профілактики зорової втоми під час тривалого кодування або дизайну інтерфейсу вебсайту рекомендується застосовувати правило «20-20-20»: кожні 20 хвилин роботи необхідно переводити погляд на предмет, що знаходиться на відстані 20 футів, близько 6 метрів, від очей, щонайменше на 20 секунд. Це дозволяє розслабити циліарний м'яз ока, який відповідає за акомодацію.

Таким чином, підтримання нормативних параметрів штучного й природного освітлення, усунення блимання та відблисків на екрані, а також дотримання ергономічних вимог до розміщення відеодисплея є надійним інструментом

збереження зорового здоров'я ІТ-фахівця та забезпечення високої точності й ефективності його праці.

5.3 Ергономічне проєктування робочого простору розробника інтерфейсу вебсайту “BeautyUp”

Ергономічна організація робочого простору є ключовим фактором, що визначає продуктивність праці, точність виконання завдань та збереження здоров’я розробника. Створення інтерфейсу інтернет-магазину косметики «BeautyUp» вимагає від спеціаліста тривалого перебування в сидячому положенні, виконання високоточних рухів мишею під час проєктування макетів, а також тривалої концентрації уваги. Неправильно організоване робоче місце призводить до статичного перевантаження опорно-рухового апарату, порушення постави та розвитку тунельних синдромів кистей рук.

Оскільки весь цикл розробки та впровадження вебпроєкту виконувався на базі одного портативного комп’ютера, ергономічне проєктування простору має свої специфічні особливості. Фіксоване з’єднання екрана та клавіатури в ноутбуці зазвичай змушує користувача нахиляти голову вперед і сутулитися. Для усунення цього деструктивного чинника робоче місце розробника обов'язково має бути укомплектоване додатковим обладнанням: спеціальною похилою підставкою під ноутбук або зовнішнім монітором, а також окремою периферійною клавіатурою та ергономічною мишею.

Проєктування робочої зони базується на просторовому компонуванні меблів, а саме столу та крісла, з урахуванням антропометричних характеристик тіла людини:

1. Висота робочої поверхні столу повинна встановлюватися в межах 725–750 мм, а його має бути матовою, щоб унеможливити виникнення світлових відблисків. Розміри стільниці повинні забезпечувати раціональне розміщення всього обладнання, а простір для ніг під столом повинен мати висоту не менше 600 мм і ширину не менше 500 мм.

2. Крісло розробника має бути підйомно-поворотним, обладнаним регуляторами висоти сидіння в межах 400-550 мм та кута нахилу спинки. Спинка крісла повинна мати анатомічну форму з валиком у зоні поперекового прогину, що дозволяє знизити статичне навантаження на хребет та обов'язковою є наявність підлокітників із регулюванням висоти, які знімають напруження з плечового поясу під час тривалої роботи з клавіатурою та мишею.

При правильній посадці кут між стегном і тулубом, а також у колінних та ліктьових суглобах має становити приблизно 90–100 градусів. Стопи ніг повинні повністю стояти на підлозі або на спеціальній похилій підставці.

Організація робочих зон на столі здійснюється за принципом частоти використання предметів. Усі інструменти поділяють на три зони доступності:

1. Зона оптимальної досяжності, тобто ближня зона – в ній розташовується предмети на відстані 30-40 см від тулуба. Тут розміщують окрему клавіатуру та мишу.

2. Зона максимальної досяжності, тобто середня зона – предмети знаходяться на відстані 40-50 см. Тут встановлюється ноутбук на підставці. Екран піднімається на таку висоту, щоб верхня лінія тексту була на рівні очей, а напрямок погляду був спрямований трохи зверху вниз.

3. Допоміжна зона, тобто дальня зона – понад 50 см. Тут розміщують предмети, які використовуються рідко, наприклад довідкова література чи канцелярія.

Важливим елементом ергономіки є оптимізація самого цифрового робочого простору. Для зниження зорового і нервового напруження під час нічного кодування вебдодатку «BeautyUp» розробнику доцільно використовувати темні теми в середовищах розробки та графічних редакторах, налаштовувати комфортний масштаб шрифтів та акцентувати увагу на фірмових кольорах проєкту за оптимальних налаштувань яскравості та контрастності екрана.

Таким чином, комплексне ергономічне проєктування робочого простору мінімізує фізичне навантаження на організм розробника, запобігає появі втоми та створює оптимальні умови для успішного завершення кваліфікаційної роботи.

ВИСНОВКИ

У даній кваліфікаційній роботі було розроблено повнофункціональний інтернет-магазин косметики "BeautyUp" на основі сучасного повного пакету технологій MERN. Клієнтська частина була реалізована з використанням бібліотеки React, а серверна частина - на платформі Node.js з використанням фреймворку Express.js та бази даних MongoDB.

В рамках аналітичної частини роботи було проведено дослідження існуючих на ринку аналогів, задокументовано переваги створення власного програмного продукту з гнучкою архітектурою та сформульовано технічний проект.

В рамках розробки створено адаптивний дизайн, інтерактивний каталог із фільтрацією та кошик покупок. На сервері реалізовано REST API для взаємодії з базою даних, автентифікацію користувачів та інтеграцію з платіжною системою Stripe.

Під час розробки та автоматизації, завдяки монолітному підходу, веб-сайт було успішно розгорнуто як єдиний веб-сервіс на хмарній платформі Render. Пряма інтеграція хостингу з репозиторієм дозволила налаштувати автоматичний цикл безперервної інтеграції та розробки для оновлення веб-сайту без переривання його роботи.

У фінансовій частині було розраховано повну вартість реалізації проекту, де завдяки позитивній чистій приведеній вартості, інвестовані кошти повністю окупляться за 2,4 роки, що підтверджує економічну вигідність та доцільність новоствореного бізнесу.

У частині охорони праці обґрунтовано раціональний режим відпочинку, нормалізовано параметри освітлення та спроєктовано ергономічний робочий простір для тривалої роботи з ноутбуком.

Розроблений інтернет-магазин повністю відповідає вимогам технічного завдання та дозволив закріпити навички веброзробки. Проєкт успішно розміщений у мережі Інтернет на платформі Render за адресою: https://project-beautyupstore.onrender.com.

ПЕРЕЛІК ПОСИЛАНЬ

1. Eddy Wilson MERN Quick Start Guide: Build web applications with MongoDB, Express, React, and Node. – Birmingham: Packt Publishing, 2018. – 281c.
2. Evan Hahn Express.js in Action: Writing and testing Data-driven Node applications. – Shelter Island: Manning Publications, 2016. – 236c.
3. В.І. Кошель, Г.П. Сав'юк, Б.С. Дзундза. Основи охорони праці. навчально-методичний посібник для студентів вищих навчальних закладів педагогічного напрямку. Івано-Франківськ: НАІР, 2020. 182 с
4. Грибан В. Г., Фоменко А. Є., Казначеєв Д. Г. Безпека життєдіяльності та охорона праці: підруч. Дніпро: Дніпроп. держ. ун-т внутр. справ, 2022. 388 с.
5. Інструментальні засоби вебтехнологій: навчальний посібник / за ред. проф. – Львів: Магнолія 2006, 2026. – 197 с.
6. Карпюк Г. І. Основи підприємництва : навч. посіб. для здобувачів професійної (професійно-технічної) освіти. Київ : Міністерство освіти і науки України, 2021. 105 с.
7. Axios : офіц. сайт. URL: https://axios.rest/ (дата звернення: 23.05.2026).
8. Cloudinary : офіц. сайт. URL: https://cloudinary.com/ (дата звернення: 23.05.2026).
9. Cloudinary. Про продукт, інтеграцію. Hellip : веб-сайт. URL: https://hellip.com/ua/product/cloudinary.html (дата звернення: 28.05.2026).
10. DOU.ua – Головний сайт сервісу пошуку роботи та аналітики IT-ринку України : веб-сайт. URL: https://jobs.dou.ua/ (дата звернення: 05.06.2026).
11. Express.js : офіц. сайт. URL: https://expressjs.com/en/ (дата звернення: 28.05.2026).
12. Node js : офіц. сайт. URL: https://nodejs.org/en (дата звернення: 23.05.2026).
13. Node Package Manager : офіц. сайт. URL: https://www.npmjs.com/ (дата звернення: 23.05.2026).
14. React : офіц. сайт. URL: https://react.dev/ (дата звернення: 23.05.2026).

15. React.js: популярний фреймворк фронтенд-розробки. Wezom : веб-сайт. URL: https://wezom.com.ua/ua/blog/reactjs-populyarniy-freymvork-frontend-rozrobki (дата звернення: 27.05.2026).

16. Tailwind css : офіц. сайт. URL: https://tailwindcss.com/ (дата звернення: 23.05.2026).

17. Vite : офіц. сайт. URL: https://vite.dev/ (дата звернення: 23.05.2026).

18. Wix. Вікіпедія : вільна енциклопедія : веб-сайт. URL: https://uk.wikipedia.org/wiki/Wix.com (дата звернення: 12.05.2026).

19. Емоційне вигорання на роботі: що робити, якщо не хочеться працювати. Комп'ютерна академія IT STEP : блог. URL: https://kiev.itstep.org/blog/emotional-burnout-at-work-what-to-do-if-you-dont-feel-like-working (дата звернення: 06.06.2026).

20. Компоненти і пропси. React Documentation : веб-сайт. URL: https://uk.legacy.reactjs.org/docs/components-and-props.html (дата звернення: 27.05.2026).

21. Мікроклімат виробничих приміщень. Південно-Західне міжрегіональне управління Державної служби України з питань праці : офіц. сайт. URL: https://smu.dsp.gov.ua/mikroklimat-vyrobnychykh-prymishchen/ (дата звернення: 06.06.2026).

22. Правильна структура веб сайту під SEO: приклади, види та 15+ рекомендації щодо розробки структури. Impulse Design : веб-сайт. URL: https://impulse-design.com.ua/ua/pravilnaya-struktura-veb-sajta-pod-seo.html (дата звернення: 20.05.2026).

23. Структура сайту: основні види та правила їх розробки. Webtune : веб-сайт. URL: https://webtune.com.ua/statti/web-rozrobka/struktura-sajtu/ (дата звернення: 20.05.2026).

24. Структура сайту: основні види, вимоги та приклади. Wedex : веб-сайт. URL: https://wedex.com.ua/blog/struktura-sajtu-osnovni-vydy-vymogy-ta-pryklady/ (дата звернення: 20.05.2026).

25. Чи варто робити сайт самому. Webatom : веб-сайт. URL: https://webatom.com.ua/chi-varto-robiti-sajt-samomu/ (дата звернення: 23.05.2026).

26. Що таке Back-end розробка. Wezom : веб-сайт. URL: https://wezom.com.ua/ua/blog/chto-takoe-back-end-razrabotka (дата звернення: 28.05.2026).

27. Що таке SEO: основи SEO. Idea Digital Agency : веб-сайт. URL: https://ideadigital.agency/blog/shcho-take-seo-i-navishcho-potribna-poshukova-optimizaciya/ (дата звернення: 12.05.2026).

28. Що таке Unit тести і як їх писати. Foxminded : веб-сайт. URL: https://foxminded.ua/yunit-testy/ (дата звернення: 01.06.2026).

29. Що таке бази даних? Sigma Software University : веб-сайт. URL: https://university.sigma.software/what-is-database/ (дата звернення: 23.05.2026).

30. Що таке верстка сайту. Impulse Design : веб-сайт. URL: https://impulse-design.com.ua/ua/chto-takoe-verstka-sajta.html (дата звернення: 23.05.2026).

31. Що таке структура сайту, як її створити та проаналізувати. Netpeak Software : веб-сайт. URL: https://netpeaksoftware.com/uk/blog/site-structure-how-to-build-and-analyze-it (дата звернення: 20.05.2026).

32. Що таке тестування за методом “чорної скриньки”? Sigma Software University : веб-сайт. URL: https://university.sigma.software/black-box-testing/ (дата звернення: 01.06.2026).

33. Що таке фронтенд? Від ідеї до досконалості. Asabix : веб-сайт. URL: https://asabix.com.ua/what-is-frontend/ (дата звернення: 27.05.2026).

ДОДАТКИ

Додаток А – Лістинг файлу src/main.jsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'

import { BrowserRouter } from "react-router-dom";

createRoot(document.getElementById('root')).render(
  <StrictMode>
  <BrowserRouter>
                        <App />
                </BrowserRouter>
  </StrictMode>,
)
```

Додаток Б – Лістинг компонента <App />

```
import { Route, Routes, Navigate, useLocation } from "react-router-dom";
import { useEffect } from "react";
import HomePage from "./pages/HomePage";
import SignUpPage from "./pages/SignUpPage";
import LoginPage from "./pages/LoginPage";
import AdminPage from "./pages/AdminPage";
import AboutPage from "./pages/AboutPage";
import CatalogPage from "./pages/CatalogPage";
import ContactPage from "./pages/ContactPage";
import CartPage from "./pages/CartPage";
import WishlistPage from "./pages/WishlistPage";
import ForgotPasswordPage from "./pages/ForgotPasswordPage";
import PurchaseSuccessPage from "./pages/PurchaseSuccessPage";
import PurchaseCancelPage from "./pages/PurchaseCancelPage";
import ProfilePage from "./pages/ProfilePage";
import Navbar from "./components/Navbar";
import Footer from "./components/Footer";
import { Toaster } from "react-hot-toast";
import { useUserStore } from "./stores/useUserStore";
import { useCartStore } from "./stores/useCartStore";
import LoadingSpinner from "./components/LoadingSpinner";
function App() {
const { user, checkAuth, checkingAuth } = useUserStore();
const { getCartItems } = useCartStore();
const location = useLocation();
useEffect(() => {
checkAuth();
}, [checkAuth]);
```

```
useEffect(() => {
if (user) {
getCartItems();}
}, [user, getCartItems]);
if (checkingAuth) return <LoadingSpinner />;
const noFooterRoutes = ["/login", "/signup", "/secret-dashboard"];
return (
<div className='min-h-screen bg-white text-gray-900 flex flex-col'>
<div className='relative z-50 flex-grow'>
<Navbar />
<div className="pt-20">
<Routes>
<Route path='/' element={<HomePage />} />
<Route path='/about' element={<AboutPage />} />
<Route path='/contact' element={<ContactPage />} />
<Route path='/profile'
element={user ? <ProfilePage key={user._id} /> : <Navigate to='/login' />}/>
<Route path='/signup' element={!user ? <SignUpPage /> : <Navigate to='/' />} />
<Route path='/login' element={!user ? <LoginPage /> : <Navigate to='/' />} />
<Route path='/forgot-password' element={<ForgotPasswordPage />} />
<Route path='/secret-dashboard'
element={user?.role === "admin" ? <AdminPage /> : <Navigate to='/login' />}/>
<Route path='/catalog/:category?' element={<CatalogPage />} />
<Route path='/wishlist' element={user ? <WishlistPage /> : <Navigate to='/login'
/>} />
<Route path='/cart' element={user ? <CartPage /> : <Navigate to='/login' />} />
<Route path='*' element={<Navigate to='/' />} />
<Route path='/purchase-success'
element={user ? <PurchaseSuccessPage /> : <Navigate to='/login' />}/>
<Route path='/purchase-cancel' element={user ? <PurchaseCancelPage /> :
<Navigate to='/login' />} />
```

```
</Routes>
</div>
</div>
{!noFooterRoutes.includes(location.pathname) && <Footer />}
<Toaster />
</div> );
}
export default App;
```

Додаток В – Лістинг компоненту <LoginPage />

```
import { useState } from "react";
import { Link } from "react-router-dom";
import { Mail, Lock, Loader } from "lucide-react";
import { motion } from "framer-motion";
import { useUserStore } from "../stores/useUserStore";
const LoginPage = () => {
const [formData, setFormData] = useState({
email: "",
password: "", });
const { loginWithGoogle, login, loading } = useUserStore();
const handleSubmit = async (e) => {
e.preventDefault();
await login(formData.email, formData.password);};
return (
<div className='flex flex-col justify-center py-6 sm:px-6 lg:px-8 min-h-[85vh]'>
<motion.div
className='sm:mx-auto sm:w-full sm:max-w-md'
initial={{ opacity: 0, y: -20 }}
animate={{ opacity: 1, y: 0 }}
transition={{ duration: 0.8 }}>
<div className="flex justify-center gap-8 mb-6 text-xl font-semibold">
<Link to="/signup" className="text-gray-400 hover:text-[#74090A] transition">
Sign up
</Link>
<Link to="/login" className="text-[#74090A] border-b-2 border-[#74090A] pb-
1">
Login
</Link>
```

```
</div>
</motion.div>
<motion.div
className='sm:mx-auto sm:w-full sm:max-w-md'
initial={{ opacity: 0, y: 20 }}
animate={{ opacity: 1, y: 0 }}
transition={{ duration: 0.8 }}>
<div className='bg-white py-6 px-4 shadow sm:rounded-lg sm:px-10 border border-gray-100'>
<form onSubmit={handleSubmit} className='space-y-6'>
<div>
<label className='block text-sm font-medium text-gray-700'>Email</label>
<div className='mt-1 relative rounded-md shadow-sm'>
<div className='absolute inset-y-0 left-0 pl-3 flex items-center pointer-events-none'>
<Mail className='h-5 w-5 text-gray-400' />
</div>
<input
type='email'
id='email'
required
placeholder='example@example.com'
value={formData.email}
onChange={(e) => setFormData({ ...formData, email: e.target.value })}
className='block w-full px-3 py-2 pl-10 border border-gray-300 rounded-md
placeholder-gray-400 focus:ring-[#74090A] focus:border-[#74090A] sm:text-sm'/>
</div>
</div>
<div>
<label className='block text-sm font-medium text-gray-700'>Password</label>
<div className='mt-1 relative rounded-md shadow-sm'>
```

```
<div className='absolute inset-y-0 left-0 pl-3 flex items-center pointer-events-none'>
<Lock className='h-5 w-5 text-gray-400' />
</div>
<input
type='password'
id='password'
required
placeholder='••••••••'
value={formData.password}
onChange={(e) => setFormData({ ...formData, password: e.target.value })}
className='block w-full px-3 py-2 pl-10 border border-gray-300
rounded-md placeholder-gray-400 focus:ring-[#74090A]
focus:border-[#74090A] sm:text-sm' />
</div>
</div>
<div className="flex justify-start text-sm">
<Link to="/forgot-password" className="text-[#74090A] hover:underline">
Lost your password?
</Link>
</div>
<button
type='submit'
className='w-full flex justify-center py-2 px-4 border
border-transparent rounded-md shadow-sm text-sm font-medium
text-white bg-[#74090A] hover:bg-[#5a0708] transition duration-150'
disabled={loading} >
{loading ? <Loader className='animate-spin' /> : "Log in"}
</button>
</form>
<div className='mt-6'>
```

```
<div className='relative'>
<div className='absolute inset-0 flex items-center'>
<div className='w-full border-t border-gray-200'></div>
</div>
<div className='relative flex justify-center text-sm'>
<span className='px-2 bg-white text-gray-500 uppercase tracking-wider text-[10px]'>
Or continue with
</span>
</div>
</div>
<button
onClick={loginWithGoogle}
type='button'
disabled={loading}
className='mt-4 w-full flex justify-center items-center gap-3 py-2 px-4 border border-gray-200 rounded-md shadow-sm bg-white text-sm font-medium text-gray-700 hover:bg-gray-50 transition duration-150 disabled:opacity-50'>
<img src="https://www.gstatic.com/firebasejs/ui/2.0.0/images/auth/google.svg"
alt="Google"
className="h-5 w-5"/>
<span>Google</span>
</button>
</div>
</div>
</motion.div>
</div>
);
};
export default LoginPage;
```

Додаток Г – Лістинг компоненту <SignUpPage />

```
import { useState } from "react";
import { Link } from "react-router-dom";
import { UserPlus, Mail, User, Lock, Loader, Phone } from "lucide-react";
import { motion } from "framer-motion";
import { useUserStore } from "../stores/useUserStore";
const SignUpPage = () => {
const [formData, setFormData] = useState({
name: "",
email: "",
phone: "",
password: "",
confirmPassword: "",
});
const { signup, loading } = useUserStore();
const handleSubmit = (e) => {
e.preventDefault();
signup(formData);
};
return (
<div className='flex flex-col justify-center py-4 sm:px-6 lg:px-8'>
<motion.div
className='sm:mx-auto sm:w-full sm:max-w-md'
initial={{ opacity: 0, y: -20 }}
animate={{ opacity: 1, y: 0 }}
transition={{ duration: 0.8 }}>
<div className="flex justify-center gap-8 mb-6 text-xl font-semibold">
<Link to="/signup" className="text-[#74090A] border-b-2 border-[#74090A] pb-
1">
```

```
Sign up
</Link>
<Link to="/login" className="text-gray-400 hover:text-[#74090A] transition">
Login
</Link>
</div>
</motion.div>
<motion.div
className='sm:mx-auto sm:w-full sm:max-w-md'
initial={{ opacity: 0, y: 20 }}
animate={{ opacity: 1, y: 0 }}
transition={{ duration: 0.8 }}>
<div className='bg-white py-8 px-4 shadow sm:rounded-lg sm:px-10 border border-gray-100'>
<form onSubmit={handleSubmit} className='space-y-4'>
{[
{ id: 'name', label: 'Full name', icon: User, placeholder: 'John Doe', type: 'text', autoComplete: 'name' },
{ id: 'email', label: 'Email', icon: Mail, placeholder: 'example@example.com', type: 'email', autoComplete: 'email' },
{ id: 'phone', label: 'Phone Number', icon: Phone, placeholder: '+1 234 567 890', type: 'tel', autoComplete: 'tel' }, // Нове поле
{ id: 'password', label: 'Password', icon: Lock, placeholder: '••••••••', type: 'password', min: 6, autoComplete: 'new-password' },
{ id: 'confirmPassword', label: 'Confirm password', icon: Lock, placeholder: '••••••••', type: 'password', min: 6, autoComplete: 'new-password' },
].map((field) => (
<div key={field.id}>
<label htmlFor={field.id} className='block text-sm font-medium text-gray-700'>
{field.label}
</label>
```

```
<div className='mt-1 relative rounded-md shadow-sm'>
<div className='absolute inset-y-0 left-0 pl-3 flex items-center pointer-events-none'>
<field.icon className='h-5 w-5 text-gray-400' />
</div>
<input
id={field.id}
name={field.id}
type={field.type}
autoComplete={field.autoComplete}
required
minLength={field.min}
value={formData[field.id]}
onChange={(e) => setFormData({ ...formData, [field.id]: e.target.value })}
className='block w-full px-3 py-2 pl-10 border border-gray-300 rounded-md placeholder-gray-400 focus:outline-none focus:ring-[#74090A] focus:border-[#74090A] sm:text-sm'
placeholder={field.placeholder}/>
</div>
</div>
))}
<button
type='submit'
className='w-full flex justify-center py-2 px-4 border border-transparent rounded-md shadow-sm text-sm font-medium text-white bg-[#74090A] hover:bg-[#5a0708] transition duration-150'
disabled={loading}>
{loading ? <Loader className='animate-spin' size={20} /> : "Register"}
</button>
</form>
</div>
```

```
</motion.div>
</div>
);
};
export default SignUpPage;
```

Додаток Д – Лістинг компоненту <Navbar />

```
import { useState, useRef } from "react";
import { ShoppingCart, User, CircleUser, Lock, Search, Heart, Menu, X } from "lucide-react";
import { useLocation, Link, useNavigate } from "react-router-dom";
import { useUserStore } from "../stores/useUserStore";
import { useCartStore } from "../stores/useCartStore";
import { useWishlistStore } from "../stores/useWishlistStore";
import { motion, AnimatePresence } from "framer-motion";
const Navbar = () => {
const location = useLocation();
const navigate = useNavigate();
const { user } = useUserStore();
const { cart } = useCartStore();
const { wishlist } = useWishlistStore();
const [isSearchOpen, setIsSearchOpen] = useState(false);
const [searchTerm, setSearchTerm] = useState("");
const [isMenuOpen, setIsMenuOpen] = useState(false);
const searchInputRef = useRef(null);
const isAdmin = user?.role === "admin";
const cartItemsCount = cart.length;
const wishlistItemsCount = wishlist.length;
const isActive = (path) => location.pathname === path;
const getNavLinkClass = (path) =>
`font-medium transition duration-300 flex items-center hover:text-[#A3090A] ${
isActive(path) ? "text-[#A3090A]" : "text-gray-900" }`;
const toggleMenu = () => setIsMenuOpen(!isMenuOpen);
const handleSearchSubmit = (e) => {
e.preventDefault();
```

```
if (searchTerm.trim()) {
navigate(`/catalog?search=${encodeURIComponent(searchTerm.trim())}`);
setIsSearchOpen(false);
setSearchTerm("");
if (isMenuOpen) setIsMenuOpen(false); }};
const badgeVariants = {
initial: { scale: 0.8, opacity: 0 },
animate: { scale: 1, opacity: 1 },
transition: { type: "spring", stiffness: 500, damping: 25 }};
const badgeClassName = "absolute -top-2 -right-2 bg-[#74090A] text-white text-[10px] w-5 h-5 flex items-center justify-center rounded-full font-bold shadow-sm border border-white/10";
return (
<header className='fixed top-0 left-0 w-full bg-white/80 backdrop-blur-md shadow-sm z-40 border-b border-gray-100'>
<div className='container mx-auto px-4 sm:px-6 py-4'>
<div className='flex justify-between items-center gap-4'>
<div className="flex-1 flex items-center gap-6">
<Link to='/' className='text-2xl font-sans font-semibold flex items-center shrink-0'>
<span className='text-black'>Beauty</span>
<span className='text-[#A3090A]'>Up</span>
</Link>
</div>
{/* DESKTOP NAVIGATION */}
<nav className='hidden lg:flex items-center justify-center gap-10 font-sans'>
<Link to={"/"} className={getNavLinkClass("/")}>Home</Link>
<Link to={"/about"} className={getNavLinkClass("/about")}>About</Link>
<Link to={"/catalog"} className={getNavLinkClass("/catalog")}>Catalog</Link>
```

```
    <Link                                                    to={"/contact"}
className={getNavLinkClass("/contact")}>Contact</Link>
    </nav>
    <div className='flex-1 flex items-center justify-end gap-2 sm:gap-6'>
    <div className="flex items-center">
    <AnimatePresence>
    {isSearchOpen && (
    <motion.form
    initial={{ width: 0, opacity: 0 }}
    animate={{ width: window.innerWidth < 640 ? 100 : 180, opacity: 1 }}
    exit={{ width: 0, opacity: 0 }}
    onSubmit={handleSearchSubmit}
    className="overflow-hidden" >
    <input
    ref={searchInputRef}
    autoFocus
    type="text"
    value={searchTerm}
    onChange={(e) => setSearchTerm(e.target.value)}
    placeholder="Search..."
    className="bg-transparent border-b border-gray-300
    focus:border-[#A3090A] outline-none px-2 py-1 text-sm w-full" />
    </motion.form> )}
    </AnimatePresence>
    <button onClick={() => setIsSearchOpen(!isSearchOpen)} className="text-gray-
900
    hover:text-[#A3090A] transition p-1">
    {isSearchOpen ? <X size={20} /> : <Search size={22} />}
    </button>
    </div>
```

```
{user && (
<div className="hidden md:flex items-center gap-6">
<Link to={"/wishlist"}
className="text-gray-900 hover:text-[#A3090A] relative transition-colors">
<Heart size={22} />
<motion.span key={`wishlist-${wishlistItemsCount}`} {...badgeVariants}
className={badgeClassName}>
{wishlistItemsCount}
</motion.span>
</Link>
<Link to={"/cart"}
className="text-gray-900 hover:text-[#A3090A] relative transition-colors">
<ShoppingCart size={22} />
<motion.span key={`cart-${cartItemsCount}`} {...badgeVariants}
className={badgeClassName}>
{cartItemsCount}
</motion.span>
</Link>
</div> )}
{user ? (
<Link to="/profile"
className="text-gray-900 hover:text-[#A3090A] transition-colors"
title="My Profile">
<CircleUser size={24} />
</Link>
) : (
<Link to="/login"
className="text-gray-900 hover:text-[#A3090A] transition"
title="Login">
<User size={22} />
</Link>
```

```
)}
{isAdmin && (
<Link className='hidden xl:flex bg-[#74090A]
text-white px-4 py-2 rounded-md hover:bg-[#5a0708] transition duration-300
items-center gap-1 font-medium shadow-sm whitespace-nowrap'
to={"/secret-dashboard"}>
<Lock size={18} /> Dashboard
</Link>
)}
<button className='lg:hidden text-gray-900 p-1 hover:text-[#A3090A] transition'
onClick={toggleMenu}>
{isMenuOpen ? <X size={26} /> : <Menu size={26} />}
</button>
</div>
</div>
</div>
{/* BURGER MENU */}
<AnimatePresence>
{isMenuOpen && (
<motion.div
initial={{ opacity: 0, height: 0 }}
animate={{ opacity: 1, height: "auto" }}
exit={{ opacity: 0, height: 0 }}
className='lg:hidden absolute top-full left-0 w-full bg-white border-b
border-gray-100 shadow-xl overflow-hidden font-sans' >
<div className='flex flex-col p-6 gap-6'>
{user && (
<div className="flex items-center justify-center gap-12 py-2">
<Link to={"/wishlist"} onClick={toggleMenu} className="text-gray-900
relative transition-colors flex flex-col items-center gap-1">
<div className="relative">
```

```
<Heart size={24} />
<motion.span key={`mob-wish-${wishlistItemsCount}`} {...badgeVariants}
className={badgeClassName}>
{wishlistItemsCount}
</motion.span>
</div>
<span className="text-[10px] uppercase font-bold text-gray-400">
Wishlist
</span>
</Link>
<Link to={"/cart"} onClick={toggleMenu} className="text-gray-900
relative transition-colors flex flex-col items-center gap-1">
<div className="relative">
<ShoppingCart size={24} />
<motion.span key={`mob-cart-${cartItemsCount}`} {...badgeVariants}
className={badgeClassName}>
{cartItemsCount}
</motion.span>
</div>
<span className="text-[10px] uppercase font-bold text-gray-400">Cart</span>
</Link>
</div>
)}
<hr className="border-gray-100" />
<nav className="flex flex-col gap-5 text-lg">
<Link to="/" onClick={toggleMenu}
className={getNavLinkClass("/")}>Home</Link>
<Link to="/about" onClick={toggleMenu}
className={getNavLinkClass("/about")}>About</Link>
<Link to="/catalog" onClick={toggleMenu}
className={getNavLinkClass("/catalog")}>Catalog</Link>
```

```
<Link to="/contact" onClick={toggleMenu}
className={getNavLinkClass("/contact")}>Contact</Link>
</nav>
{isAdmin && (
<>
<hr className="border-gray-100" />
<Link to="/secret-dashboard" onClick={toggleMenu}
className="flex items-center gap-2 text-[#74090A] font-bold tracking-wide">
<Lock size={18} /> Admin Dashboard
</Link>
</>
)}
</div>
</motion.div>
)}
</AnimatePresence>
</header>
);
};
export default Navbar;
```

Додаток Е – Лістинг компоненту <HomePage />

```
import { useEffect } from "react";
import { Link } from "react-router-dom";
import { useProductStore } from "../stores/useProductStore";
import { useCategoryStore } from "../stores/useCategoryStore";
import FeaturedProducts from "../components/FeaturedProducts";
import CategoryItem from "../components/CategoryItem";
const HomePage = () => {
const { fetchFeaturedProducts, featuredProducts, loading: productsLoading } = useProductStore();
const { fetchCategories, categories, loading: categoriesLoading } = useCategoryStore();
useEffect(() => {
fetchFeaturedProducts();
fetchCategories();
}, [fetchFeaturedProducts, fetchCategories]);
const isLoading = productsLoading || categoriesLoading;
return (
<div className='relative min-h-screen bg-white'>
<div className="relative min-h-[60vh] flex items-center justify-center pt-20">
<div className="container mx-auto px-6 flex flex-col items-center text-center">
<div className="relative flex flex-col items-center mb-10">
<h1 className='font-["Playfair_Display"] text-4xl sm:text-5xl lg:text-6xl
uppercase text-black tracking-[0.2em] leading-none'>
Light up your
</h1>
<span className='font-["Monsieur_La_Doulaise"] text-5xl sm:text-7xl
lg:text-8xl text-[#74090A] lowercase
mt-[-10px] sm:mt-[-20px] ml-20 sm:ml-40 z-10 select-none'>
```

```
natural beauty
</span>
</div>
<p className="font-sans text-black/90 text-[10px] sm:text-xs tracking-[0.3em]
leading-relaxed max-w-sm mb-12">
Explore our premium range of beauty products, meticulously
crafted to enhance your natural radiance. Each formula is designed to
empower your self-expression, perfect your daily ritual,
and inspire a new level of confidence every single day
</p>
<Link
to="/catalog"
className="font-sans bg-[#74090A] text-white text-[10px] font-medium
tracking-[0.4em] px-12 py-4 rounded-sm hover:bg-black transition-all duration-500"
>
Shop now
</Link>
</div>
</div>
<div className="py-10 max-w-6xl mx-auto px-4 sm:px-6">
{!productsLoading && featuredProducts && featuredProducts.length > 0 && (
<FeaturedProducts featuredProducts={featuredProducts} />
)}
</div>
<section className='py-16 md:py-24 bg-white border-t border-neutral-50'>
<div className='container mx-auto px-6'>
<div className='flex flex-col items-center mb-16'>
<div className='flex flex-col items-center gap-1'>
<h2 className='font-["Playfair_Display"] text-3xl sm:text-4xl
md:text-5xl uppercase text-black tracking-[0.15em] text-center leading-tight'>
Choose your care for
```

```
</h2>
<span className='font-["Monsieur_La_Doulaise"] text-5xl sm:text-6xl md:text-7xl
text-[#74090A] lowercase mt-[-15px] md:mt-[-25px] ml-16 md:ml-32 select-none'>
today
</span>
</div>
</div>
<div className='max-w-4xl mx-auto'>
<div className='grid grid-cols-2 md:grid-cols-3 gap-6 sm:gap-10'>
{!isLoading && categories.length > 0 ? (
categories.map((category) => (
<CategoryItem category={category} key={category._id} />
))
) : (
!isLoading && (
<p className="col-span-full text-center text-gray-400 text-xs
tracking-widest uppercase">
Collection coming soon
</p>
)
)}
</div>
</div>
</div>
</section>
</div>
);
};
export default HomePage;
```

Додаток Є – Лістинг компоненту <Footer />

```
import React from "react";
import { Mail, Phone } from "lucide-react";
const Footer = () => {
return (
<footer className='bg-[#5E0304] py-6 border-t border-black/10 mt-auto'>
<div className='container mx-auto px-6 max-w-5xl flex flex-col
md:flex-row justify-between items-center gap-y-4'>
<div className='flex items-baseline gap-x-2'>
<p className='font-["Playfair_Display"] text-[10px] sm:text-xs
text-white/60 uppercase tracking-[0.2em] whitespace-nowrap'>
Light up your
</p>
<p className='font-["Monsieur_La_Doulaise"] text-2xl sm:text-2xl
text-white/50 lowercase whitespace-nowrap'>
natural beauty
</p>
</div>
<div className='flex flex-col items-center md:items-end gap-y-1'>
<a
href="mailto:webstorebeautyup@gmail.com"
className="flex items-center gap-x-2 text-white/70
hover:text-white transition-colors duration-300 text-[10px] sm:text-xs tracking-wide"
>
<span className="font-sans">webstorebeautyup@gmail.com</span>
<Mail size={12} className="text-white/30" />
</a>
<a
href="tel:+12125550198"
```

```
    className="flex items-center gap-x-2 text-white/70
    hover:text-white transition-colors duration-300 text-[10px] sm:text-xs tracking-wide"
>
    <span className="font-sans">+1 (212) 555-0198</span>
    <Phone size={12} className="text-white/30" />
    </a>
    </div>
    </div>
    </footer>
    );
    };
    export default Footer;
```

Додаток Ж – Лістинг компоненту <ProductCard />

```
import { ShoppingCart, Heart } from "lucide-react";
import { useCartStore } from "../stores/useCartStore";
import { useWishlistStore } from "../stores/useWishlistStore";
import { useUserStore } from "../stores/useUserStore";
import { motion } from "framer-motion";
import toast from "react-hot-toast";
const ProductCard = ({ product, isCompact }) => {
const { addToCart } = useCartStore();
const { toggleWishlist, isInWishlist } = useWishlistStore();
const { user } = useUserStore();
const isFavorite = isInWishlist(product._id);
const handleWishlistClick = (e) => {
e.preventDefault();
e.stopPropagation();
if (!user) {
toast.error("Please login to add products to your wishlist");
return;
}
toggleWishlist(product);
};
const handleCartClick = (e) => {
e.preventDefault();
e.stopPropagation();
if (!user) {
toast.error("Please login to add products to your cart");
return;
}
addToCart(product);
```

```
};
return (
<motion.div
layout
initial={{ opacity: 0, y: 20 }}
animate={{ opacity: 1, y: 0 }}
className="group relative bg-white rounded-[24px] overflow-hidden border
border-neutral-50 hover:shadow-2xl hover:shadow-neutral-100 transition-all
duration-500 w-full" >
<div className="relative aspect-[4/5] overflow-hidden bg-neutral-50">
<img
src={product.image}
alt={product.name}
className="w-full h-full object-cover transition-transform duration-700 group-
hover:scale-110" />
<div className="absolute inset-0 bg-black/5 opacity-0 group-hover:opacity-100
transition-opacity duration-300" />
<div className={`absolute bottom-3 right-3 md:bottom-4 md:right-4 flex flex-col
gap-2
translate-y-4 opacity-0 group-hover:translate-y-0 group-hover:opacity-100
transition-all duration-500 ${isCompact ? "scale-75 md:scale-100" : ""}`}>
<button
onClick={handleWishlistClick}
className={`p-2.5 md:p-3 rounded-full backdrop-blur-md shadow-sm transition-all
duration-300 ${
isFavorite
? "bg-[#74090A] text-white"
: "bg-white/90 text-neutral-600 hover:text-[#74090A]"
}`}
>
<Heart size={18} className={isFavorite ? "fill-white" : "fill-none"} />
```

```
</button>
<button
onClick={handleCartClick}
className="p-2.5 md:p-3 rounded-full bg-[#74090A] text-white
hover:bg-[#4F0608] backdrop-blur-md shadow-sm transition-all duration-300" >
<ShoppingCart size={18} />
</button>
</div>
{product.category && (
<span className="absolute top-3 left-3 md:top-4 md:left-4 text-[7px] md:text-[8px]
uppercase
tracking-[0.2em] bg-white/90 backdrop-blur-md text-[#74090A] font-bold px-2
md:px-2.5 py-1
rounded-full shadow-sm z-10">
{product.category}
</span>
)}
</div>
<div className="p-3 md:p-5">
<h3 className={`font-serif text-neutral-800 mb-1 leading-snug
group-hover:text-[#74090A] transition-colors line-clamp-1 ${
isCompact ? "text-[10px] md:text-sm" : "text-sm"
}`}>
{product.name}
</h3>
<div className="flex justify-between items-center">
<p className={`font-medium text-neutral-900 ${
isCompact ? "text-[11px] md:text-sm" : "text-sm"
}`}>
${product.price}
</p>
```

```
</div>
</div>
</motion.div>
);
};
export default ProductCard;
```

Додаток З – Лістинг компоненту <CartPage />

```
import { useState } from "react";
import { Link } from "react-router-dom";
import { ShoppingBag } from "lucide-react";
import { motion, AnimatePresence } from "framer-motion";
import { useCartStore } from "../stores/useCartStore";
import CartItem from "../components/CartItem";
import PeopleAlsoBought from "../components/PeopleAlsoBought";
import OrderSummary from "../components/OrderSummary";
import GiftCouponCard from "../components/GiftCouponCard";
import ProductModal from "../components/ProductModal";
const CartPage = () => {
const { cart, addToCart } = useCartStore();
const [selectedProduct, setSelectedProduct] = useState(null);
const [isModalOpen, setIsModalOpen] = useState(false);
return (
<div className="py-12 md:py-16 bg-white min-h-screen mt-10">
<div className="mx-auto max-w-6xl px-6">
<header className="mb-12 border-b border-neutral-100 pb-8">
<span className="text-[9px] uppercase tracking-[0.3em] text-[#74090A] font-bold
mb-2 block">
Your selection
</span>
<div className="flex items-baseline justify-between">
<h1 className="text-3xl md:text-4xl font-light
tracking-tight text-neutral-900 font-serif lowercase first-letter:uppercase">
Shopping Bag
</h1>
```

```
<span className="text-neutral-400 text-[9px] uppercase tracking-[0.2em] font-medium">
{cart.length} {cart.length === 1 ? "Item" : "Items"}
</span>
</div>
</header>
<div className="flex flex-col lg:flex-row gap-12 lg:gap-14">
<div className="flex-1 min-w-0">
{cart.length === 0 ? (
<EmptyCartUI />
) : (
<>
<div
className="grid grid-cols-2 md:grid-cols-3 lg:grid-cols-2 xl:grid-cols-3 gap-x-4 gap-y-10
md:gap-x-8 md:gap-y-12" >
<AnimatePresence>
{cart.map((item) => (
<motion.div
key={item._id}
layout
initial={{ opacity: 0, scale: 0.9 }}
animate={{ opacity: 1, scale: 1 }}
exit={{ opacity: 0, scale: 0.9 }} >
<CartItem
item={item}
onOpenModal={(product) => {
setSelectedProduct(product);
setIsModalOpen(true);
}}
/>
```

```
</motion.div>
))}
</AnimatePresence>
</div>
<div className="mt-16 pt-12 border-t border-neutral-100">
<PeopleAlsoBought />
</div>
</>
)}
</div>
{cart.length > 0 && (
<aside className="w-full lg:w-[300px] xl:w-[320px] shrink-0">
<div className="lg:sticky lg:top-28 space-y-6">
<OrderSummary />
<GiftCouponCard />
</div>
</aside>
)}
</div>
</div>
<ProductModal
product={selectedProduct}
isOpen={isModalOpen}
onClose={() => setIsModalOpen(false)}
onAddToCart={addToCart} />
</div>
);
};
export default CartPage;
const EmptyCartUI = () => (
<div
```

```
className="flex flex-col items-center justify-center py-24 text-center bg-neutral-50/30 rounded-[32px]
border border-dashed border-neutral-100" >
<div className="relative mb-6">
<ShoppingBag className="h-10 w-10 text-neutral-400" strokeWidth={1} />
<div className="absolute top-0 right-0 w-1.5 h-1.5 bg-[#74090A] rounded-full" />
</div>
<h3 className="text-lg font-light text-neutral-700 font-serif">Your bag is empty</h3>
<p
className="text-neutral-500 text-[10px] mt-2 mb-8 tracking-wide font-light max-w-[200px] mx-auto
leading-relaxed" >
Your self-care journey starts here. Add your essentials to begin.
</p>
<Link
className="inline-block bg-[#74090A] text-white px-10 py-3 rounded-full text-[9px] font-bold uppercase
tracking-[0.2em] hover:bg-black transition-all duration-500"
to="/catalog" >
Discover products
</Link>
</div>
);
```

Додаток И – Лістинг компоненту <AdminPage />

```
import { BarChart, PlusCircle, ShoppingBasket, Users, ChevronLeft, ChevronRight, Ticket, Package, LayoutGrid } from "lucide-react";
import { useEffect, useState } from "react";
import { motion } from "framer-motion";
import AnalyticsTab from "../components/AnalyticsTab";
import CreateProductForm from "../components/CreateProductForm";
import EditProductForm from "../components/EditProductForm";
import CreateCategory from "../components/CreateCategory";
import ProductsList from "../components/ProductsList";
import UsersManager from "../components/UsersManager";
import CouponManager from "../components/CouponManager";
import OrdersList from "../components/OrdersList";
import { useProductStore } from "../stores/useProductStore";
import { useUserStore } from "../stores/useUserStore";
import { useOrderStore } from "../stores/useOrderStore";
const tabs = [
{ id: "create", label: "Create Product", icon: PlusCircle },
{ id: "categories", label: "Create Categories", icon: LayoutGrid },
{ id: "coupons", label: "Coupons", icon: Ticket },
{ id: "products", label: "Products", icon: ShoppingBasket },
{ id: "users", label: "Users", icon: Users },
{ id: "orders", label: "Orders", icon: Package },
{ id: "analytics", label: "Analytics", icon: BarChart },
];
const AdminPage = () => {
const [activeTab, setActiveTab] = useState("create");
const [orderFilter] = useState("All");
const [editingProduct, setEditingProduct] = useState(null);
```

```
const { fetchAllProducts, totalPages: productPages, currentPage: productPage } = useProductStore();
const { fetchAllUsers, totalPages: userPages, currentPage: userPage } = useUserStore();
const { fetchAllOrders, totalPages: orderPages, currentPage: orderPage } = useOrderStore();
useEffect(() => {
if (activeTab === "products") fetchAllProducts(1);
if (activeTab === "users") fetchAllUsers(1);
if (activeTab === "orders") fetchAllOrders(1);
}, [fetchAllProducts, fetchAllUsers, fetchAllOrders, activeTab]);
const handlePageChange = (page) => {
if (activeTab === "products") fetchAllProducts(page);
if (activeTab === "users") fetchAllUsers(page);
if (activeTab === "orders") fetchAllOrders(page, orderFilter);
window.scrollTo({ top: 0, behavior: "smooth" });
};
const currentPaginationPage =
activeTab === "products" ? productPage :
activeTab === "users" ? userPage :
orderPage;
const currentTotalPages =
activeTab === "products" ? productPages :
activeTab === "users" ? userPages :
orderPages;
return (
<div className='min-h-screen relative overflow-hidden bg-white'>
<div className='relative z-10 container mx-auto px-4 py-16'>
<motion.h1
className='text-3xl font-semibold mb-12 text-[#74090A] text-center tracking-widest uppercase'
```

```
initial={{ opacity: 0, y: -20 }}
animate={{ opacity: 1, y: 0 }}
transition={{ duration: 0.8 }} >
Admin Dashboard
</motion.h1>
<div className='flex flex-wrap justify-center mb-12 gap-y-4'>
{tabs.map((tab) => (
<button
key={tab.id}
onClick={() => {
setActiveTab(tab.id);
setEditingProduct(null);
}}
className={`flex items-center px-6 py-2 mx-2 rounded-md transition-all
duration-200 font-medium text-sm tracking-wide ${
activeTab === tab.id
? "bg-[#74090A] text-white shadow-md"
: "bg-gray-100 text-gray-600 hover:bg-gray-200"
}`} >
<tab.icon className={`mr-2 h-4 w-4 ${activeTab === tab.id ? "text-white" :
"text-[#74090A]"}`} />
<span className="whitespace-nowrap">{tab.label}</span>
</button>
))}
</div>
<div className="bg-white rounded-lg">
{activeTab === "create" && <CreateProductForm />}
{activeTab === "categories" && <CreateCategory />}
{activeTab === "coupons" && <CouponManager />}
{activeTab === "products" && (
<>
```

```
{editingProduct ? (
<EditProductForm
product={editingProduct}
onCancel={() => setEditingProduct(null)}
/>
) : (
<ProductsList onEdit={setEditingProduct} />
)}
</>
)}
{activeTab === "users" && <UsersManager />}
{activeTab === "orders" && <OrdersList />}
{activeTab === "analytics" && <AnalyticsTab />}
{(activeTab === "products" || activeTab === "users" || activeTab === "orders") &&
!editingProduct && currentTotalPages > 1 && (
<div className='flex justify-center items-center mt-12 gap-4 pb-10'>
<button
disabled={currentPaginationPage === 1}
onClick={() => handlePageChange(currentPaginationPage - 1)}
className='p-2 rounded-full border border-gray-200 bg-white disabled:opacity-30
hover:bg-gray-50 transition-colors shadow-sm' >
<ChevronLeft className='text-[#74090A]' size={20} />
</button
<div className='flex gap-2'>
{[...Array(currentTotalPages)].map((_, i) => (
<button
key={i + 1}
onClick={() => handlePageChange(i + 1)}
className={`w-9 h-9 rounded-md text-[11px] font-bold transition-all shadow-sm ${
currentPaginationPage === i + 1
? "bg-[#74090A] text-white shadow-md"
```

```
: "bg-white border border-gray-100 text-neutral-400 hover:bg-neutral-50"
}`}
>
{i + 1}
</button>
))}
</div>
<button
disabled={currentPaginationPage === currentTotalPages}
onClick={() => handlePageChange(currentPaginationPage + 1)}
className='p-2 rounded-full border border-gray-200
bg-white disabled:opacity-30 hover:bg-gray-50 transition-colors shadow-sm'
>
<ChevronRight className='text-[#74090A]' size={20} />
</button>
</div>
)}
</div>
</div>
</div>
);
};
export default AdminPage;
```

Додаток І – Лістинг компоненту <CreateProductForm />

```
import { useState, useEffect } from "react";
import { motion } from "framer-motion";
import { PlusCircle, Upload, Loader } from "lucide-react";
import { useProductStore } from "../stores/useProductStore";
const CreateProductForm = () => {
const [newProduct, setNewProduct] = useState({
name: "",
description: "",
price: "",
category: "",
image: "",
});
const {
createProduct,
loading,
categories,
fetchAllCategories
} = useProductStore();
useEffect(() => {
if (fetchAllCategories) fetchAllCategories();
}, [fetchAllCategories]);
const handleSubmit = async (e) => {
e.preventDefault();
if (!newProduct.image) {
alert("Please upload an image first!");
return;
}
try {
```

```
const productData = {
...newProduct,
price: parseFloat(newProduct.price)
};
await createProduct(productData);
setNewProduct({ name: "", description: "", price: "", category: "", image: "" });
} catch (error) {
console.error("Full error object:", error);
}};
const handleImageChange = (e) => {
const file = e.target.files[0];
if (file) {
const reader = new FileReader();
reader.onloadend = () => {
setNewProduct({ ...newProduct, image: reader.result });
};
reader.readAsDataURL(file);
}};
return (
<motion.div
className='bg-white shadow-sm rounded-lg p-8 mb-8 max-w-xl mx-auto border border-gray-100'
initial={{ opacity: 0, y: 20 }}
animate={{ opacity: 1, y: 0 }}
transition={{ duration: 0.8 }} >
<h2 className='text-xl font-semibold mb-6 text-[#74090A] tracking-wider uppercase text-center'>
Create New Product
</h2>
<form onSubmit={handleSubmit} className='space-y-5'>
<div>
```

```
<label htmlFor='name' className='block text-sm font-medium text-gray-700 mb-1'>
Product Name
</label>
<input
type='text'
id='name'
value={newProduct.name}
onChange={(e) => setNewProduct({ ...newProduct, name: e.target.value })}
className='block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm
focus:outline-none focus:ring-1 focus:ring-[#74090A] focus:border-[#74090A]
sm:text-sm'
placeholder='Name your beauty product...'
required />
</div>
<div>
<label htmlFor='description' className='block text-sm font-medium text-gray-700
mb-1'>
Description
</label>
<textarea
id='description'
value={newProduct.description}
onChange={(e) => setNewProduct({ ...newProduct, description: e.target.value })}
rows='3'
className='block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm
focus:outline-none focus:ring-1 focus:ring-[#74090A] focus:border-[#74090A]
sm:text-sm'
placeholder='Describe the product rituals...'
required />
</div>
<div>
```

```
<label htmlFor='price' className='block text-sm font-medium text-gray-700 mb-1'>
Price
</label>
<input
type='number'
id='price'
value={newProduct.price}
onChange={(e) => setNewProduct({ ...newProduct, price: e.target.value })}
step='0.5'
className='block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm
focus:outline-none focus:ring-1 focus:ring-[#74090A] focus:border-[#74090A]
sm:text-sm'
placeholder='0.00'
required />
</div>
<div>
<label htmlFor='category' className='block text-sm font-medium text-gray-700 mb-
1'>
Category
</label>
<select
id='category'
value={newProduct.category}
onChange={(e) => setNewProduct({ ...newProduct, category: e.target.value })}
className='block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm
focus:ring-[#74090A] focus:border-[#74090A] sm:text-sm capitalize outline-none'
required >
<option value="">Select a category</option>
{categories?.filter(c => c !== "All").map((cat) => (
<option key={cat._id || cat} value={cat.name || cat}>
{cat.name || cat}
```

```
</option>
))}
</select>
</div>
<div className='flex items-center pt-2'>
<input type='file' id='image' className='sr-only' accept='image/*' onChange={handleImageChange} />
<label
htmlFor='image'
className='cursor-pointer bg-gray-50 py-2 px-4 border border-gray-300 rounded-md
shadow-sm text-sm font-medium text-gray-700
hover:bg-gray-100 flex items-center transition-colors' >
<Upload className='h-4 w-4 mr-2 text-[#74090A]' />
Upload Image
</label>
{newProduct.image && (
<span className='ml-3 text-xs text-green-600 font-medium'>✓ Image ready</span>
)}
</div>
<button
type='submit'
className='w-full flex justify-center py-2.5 px-4 border border-transparent
rounded-md shadow-sm text-sm font-medium text-white bg-[#74090A]
hover:bg-[#5a0708] disabled:opacity-50 mt-4 transition-colors'
disabled={loading} >
{loading ? (
<>
<Loader className='mr-2 h-5 w-5 animate-spin' />
Creating...
```

```
</>
) : (
<>
<PlusCircle className='mr-2 h-5 w-5' />
Create Product
</>
)}
</button>
</form>
</motion.div>
);
};
export default CreateProductForm;
```

Додаток Ї – Лістинг компоненту <EditProductForm />

```
import { useState, useEffect } from "react";
import { motion } from "framer-motion";
import { Upload, Loader, Save, X } from "lucide-react";
import { useProductStore } from "../stores/useProductStore";
const categories = ["face", "body", "hair", "sun", "accessories", "sets"];
const EditProductForm = ({ product, onCancel }) => {
const [updatedProduct, setUpdatedProduct] = useState({
name: "",
description: "",
price: "",
category: "",
image: "",
});
const { updateProduct, loading } = useProductStore();
useEffect(() => {
if (product) {
setUpdatedProduct({
name: product.name,
description: product.description,
price: product.price,
category: product.category,
image: product.image,
});
}
}, [product]);
const handleSubmit = async (e) => {
e.preventDefault();
try {
```

```
await updateProduct(product._id, updatedProduct);
onCancel();
} catch (error) {
console.log("Error updating product:", error);
}};
const handleImageChange = (e) => {
const file = e.target.files[0];
if (file) {
const reader = new FileReader();
reader.onloadend = () => {
setUpdatedProduct({ ...updatedProduct, image: reader.result });
};
reader.readAsDataURL(file);
}};
return (
<motion.div
className="bg-white shadow-sm rounded-lg p-8 mb-8 max-w-xl mx-auto border border-gray-100"
initial={{ opacity: 0, y: 20 }}
animate={{ opacity: 1, y: 0 }}
transition={{ duration: 0.8 }}>
<div className="flex justify-between items-center mb-6">
<h2 className="text-xl font-semibold text-[#74090A] tracking-wider uppercase">
Edit Product
</h2>
<button
onClick={onCancel}
className="text-gray-400 hover:text-[#74090A] transition-colors">
<X size={24} />
</button>
</div>
```

```
<form onSubmit={handleSubmit} className="space-y-5 text-left">
<div>
<label
htmlFor="edit-name"
className="block text-sm font-medium text-gray-700 mb-1">
Product Name
</label>
<input
type="text"
id="edit-name"
value={updatedProduct.name}
onChange={(e) =>
setUpdatedProduct({ ...updatedProduct, name: e.target.value })}
className="block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm
focus:outline-none focus:ring-1 focus:ring-[#74090A] focus:border-[#74090A]
sm:text-sm
transition-all"
required/>
</div>
<div>
<label
htmlFor="edit-description"
className="block text-sm font-medium text-gray-700 mb-1" >
Description
</label>
<textarea
id="edit-description"
value={updatedProduct.description}
onChange={(e) =>
setUpdatedProduct({
...updatedProduct,
```

```
description: e.target.value,
})}
rows="3"
className="block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm
focus:outline-none focus:ring-1 focus:ring-[#74090A] focus:border-[#74090A]
sm:text-sm transition-all"
required />
</div>
<div>
<label
htmlFor="edit-price"
className="block text-sm font-medium text-gray-700 mb-1" >
Price
</label>
<input
type="number"
id="edit-price"
value={updatedProduct.price}
onChange={(e) =>
setUpdatedProduct({ ...updatedProduct, price: e.target.value })}
step="0.5"
className="block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm
focus:outline-none focus:ring-1 focus:ring-[#74090A] focus:border-[#74090A]
sm:text-sm"
required />
</div>
<div>
<label
htmlFor="edit-category"
className="block text-sm font-medium text-gray-700 mb-1" >
Category
```

```
</label>
<select
id="edit-category"
value={updatedProduct.category}
onChange={(e) =>
setUpdatedProduct({ ...updatedProduct, category: e.target.value })}
className="block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm
focus:outline-none focus:ring-1 focus:ring-[#74090A] focus:border-[#74090A]
sm:text-sm capitalize"
required >
<option value="" className="capitalize">
Select a category
</option>
{categories.map((category) => (
<option key={category} value={category} className="capitalize">
{category}
</option>
))}
</select>
</div>
<div className="flex items-center pt-2">
<input
type="file"
id="edit-image"
className="sr-only"
accept="image/*"
onChange={handleImageChange} />
<label
htmlFor="edit-image"
className="cursor-pointer bg-gray-50 py-2 px-4 border border-gray-300 rounded-
md shadow-sm
```

```
text-sm font-medium text-gray-700 hover:bg-gray-100 transition-colors flex items-
center" >
<Upload className="h-4 w-4 mr-2 text-[#74090A]" />
Update Image
</label>
{updatedProduct.image && (
<span className="ml-3 text-xs text-green-600 font-medium">
Image selected
</span>
)}
</div>
{updatedProduct.image && (
<div className="mt-2 flex justify-center">
<img
src={updatedProduct.image}
alt="Preview"
className="h-24 w-24 object-cover
rounded-md border border-gray-200 shadow-sm" />
</div>
)}
<div className="flex gap-4 pt-4">
<button
type="button"
onClick={onCancel}
className="flex-1 justify-center py-2.5 px-4 border border-gray-300 rounded-md
shadow-sm text-sm font-medium text-gray-700 bg-white hover:bg-gray-50
focus:outline-none transition-all" >
Cancel
</button>
<button
type="submit"
```

```
    className="flex-1 flex justify-center py-2.5 px-4 border border-transparent rounded-md
    shadow-sm text-sm font-medium text-white bg-[#74090A] hover:bg-[#5a0708]
    focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-[#74090A]
    transition-all disabled:opacity-50"
    disabled={loading} >
    {loading ? (
    <>
    <Loader className="mr-2 h-5 w-5 animate-spin" />
    Saving...
    </>
    ) : (
    <>
    <Save className="mr-2 h-5 w-5" />
    Save Changes
    </> )}
    </button>
    </div>
    </form>
    </motion.div>
    );};
    export default EditProductForm;
```

Додаток Й – Лістинг файлу server.js

```
import express from "express";
import dotenv from "dotenv";
import cookieParser from "cookie-parser";
import cors from "cors";
import path from "path";
import authRoutes from "./routes/auth.route.js";
import productRoutes from "./routes/product.route.js";
import cartRoutes from "./routes/cart.route.js";
import couponRoutes from "./routes/coupon.route.js";
import paymentRoutes from "./routes/payment.route.js";
import analyticsRoutes from "./routes/analytics.route.js";
import userRoutes from "./routes/user.route.js";
import wishlistRoutes from "./routes/wishlist.route.js";
import categoryRoutes from "./routes/category.route.js";
import orderRoutes from "./routes/order.route.js";
import { connectDB } from "./lib/db.js";
dotenv.config();
const app = express();
const PORT = process.env.PORT || 5000;
const __dirname = path.resolve();
app.use(
cors({
origin: process.env.CLIENT_URL || "http://localhost:5173",
credentials: true,
}),
);
app.use(express.json({ limit: "10mb" }));
app.use(express.urlencoded({ limit: "15mb", extended: true }));
```

```
app.use(cookieParser());
app.use("/api/categories", categoryRoutes);
app.use("/api/products", productRoutes);
app.use("/api/auth", authRoutes);
app.use("/api/cart", cartRoutes);
app.use("/api/coupons", couponRoutes);
app.use("/api/payments", paymentRoutes);
app.use("/api/analytics", analyticsRoutes);
app.use("/api/users", userRoutes);
app.use("/api/wishlist", wishlistRoutes);
app.use("/api/orders", orderRoutes);
app.use((err, req, res, next) => {
if (err.type === "entity.too.large") {
return res.status(413).json({
message:
"The photo is too big. Please choose a smaller file size (up to 5 MB).",
});
}
res.status(500).json({ message: "Something went wrong on the server" });
});
if (process.env.NODE_ENV === "production") {
app.use(express.static(path.join(__dirname, "/frontend/dist")));
app.get(/.*/, (req, res) => {
res.sendFile(path.resolve(__dirname, "frontend", "dist", "index.html"));
});
}
app.listen(PORT, async () => {
console.log("Server is running on http://localhost:" + PORT);
await connectDB();
try {
const mongoose = (await import("mongoose")).default;
```

```
const collection = mongoose.connection.db.collection("coupons");
const indexes = await collection.indexes();
console.log(
"Current indexes in DB:",
indexes.map((i) => i.name), );
if (indexes.some((i) => i.name === "userId_1")) {
await collection.dropIndex("userId_1");
console.log("SUCCESS: Index userId_1 was found and deleted");
} else {
console.log("Index userId_1 not found, everything is clean."); }
} catch (err) {
console.log("Note: Index cleaning skipped or index doesn't exist.");
}});
```

Додаток К – Лістинг файлу auth.route.js

```
import express from "express";
import {
    login, logout, signup, refreshToken, getProfile,
    forgotPassword, resetPassword, googleLogin, toggleBlockUser
} from "../controllers/auth.controller.js";
import { protectRoute, adminRoute } from "../middleware/auth.middleware.js";
const router = express.Router();
router.post("/signup", signup);
router.post("/login", login);
router.post("/logout", logout);
router.post("/refresh-token", refreshToken);
router.post("/forgot-password", forgotPassword);
router.post("/reset-password", resetPassword);
router.get("/profile", protectRoute, getProfile);
router.post("/google", googleLogin);
router.put("/users/block/:id", protectRoute, adminRoute, toggleBlockUser);
export default router;
```

Додаток Л – Лістинг файлу product.route.js

```
import express from "express";
import {
createProduct,
deleteProduct,
updateProduct,
getAllProducts,
getFeaturedProducts,
getProductsByCategory,
getRecommendedProducts,
toggleFeaturedProduct,
getUniqueCategories,
getMaxProductPrice,
} from "../controllers/product.controller.js";
import { adminRoute, protectRoute } from "../middleware/auth.middleware.js";
const router = express.Router();
router.get("/", getAllProducts);
router.get("/featured", getFeaturedProducts);
router.get("/recommendations", getRecommendedProducts);
router.get("/category/:category", getProductsByCategory);
router.get("/categories", getUniqueCategories);
router.get("/max-price", getMaxProductPrice);
router.post("/categories", protectRoute, adminRoute, getUniqueCategories);
router.post("/", protectRoute, adminRoute, createProduct);
router.patch("/:id", protectRoute, adminRoute, toggleFeaturedProduct);
router.delete("/:id", protectRoute, adminRoute, deleteProduct);
router.put("/:id", protectRoute, adminRoute, updateProduct);
export default router;
```

Додаток М – Лістинг файлів схем і моделей бази даних

```
//Файл models/category.model

import mongoose from "mongoose";
const categorySchema = new mongoose.Schema({
    name: { type: String, required: true, unique: true },
    href: { type: String, required: true },
    imageUrl: { type: String, required: true },
}, { timestamps: true });
const Category = mongoose.model("Category", categorySchema);
export default Category;

//Файл models/coupon.model

import mongoose from "mongoose";
const couponSchema = new mongoose.Schema({
code: {
type: String,
required: true,
unique: true,
trim: true, },
discountPercentage: {
type: Number,
required: true,
min: 0,
max: 100, },
expirationDate: {
type: Date,
required: true, },
```

```
isActive: {
type: Boolean,
default: true, },
minimumPurchaseAmount: {
type: Number,
default: 0, },
isFirstOrderOnly: {
type: Boolean,
default: false, },
userId: {
type: mongoose.Schema.Types.ObjectId,
ref: "User",
required: false, },
},{
timestamps: true, });
const Coupon = mongoose.models.Coupon || mongoose.model("Coupon", couponSchema);
export default Coupon;

//Файл models/order.model

import mongoose from "mongoose";
const orderSchema = new mongoose.Schema({
user: {
type: mongoose.Schema.Types.ObjectId,
ref: "User",
required: true, },
products: [ {
product: {
type: mongoose.Schema.Types.ObjectId,
ref: "Product",
```

```
required: true, },
quantity: {
type: Number,
required: true,
min: 1, },
price: {
type: Number,
required: true,
min: 0, },
}, ],
totalAmount: {
type: Number,
required: true,
min: 0, },
stripeSessionId: {
type: String,
required: false,
unique: true, },
status: {
type: String,
required: true,
enum: ["Pending", "Paid", "Packed", "Shipped", "Delivered"],
default: "Pending",},
phone: {
type: String,
required: false, },
shippingAddress: {
type: String,
default: "", },
},
{ timestamps: true });
```

```
orderSchema.index({ user: 1, createdAt: -1 });
const Order = mongoose.model("Order", orderSchema);
export default Order;

//Файл models/product.model

import mongoose from "mongoose";
const productSchema = new mongoose.Schema( {
name: {
type: String,
required: true, },
description: {
type: String,
required: true, },
price: {
type: Number,
min: 0,
required: true, },
image: {
type: String,
required: [true, "Image is required"], },
category: {
type: String,
required: true, },
isFeatured: {
type: Boolean,
default: false, },
},
{ timestamps: true } );
const Product = mongoose.model("Product", productSchema);
export default Product;
```

```
//Файл models/user.model

import mongoose from "mongoose";
import bcrypt from "bcryptjs";
const userSchema = new mongoose.Schema( {
name: {
type: String,
required: [true, "Name is required"], },
email: {
type: String,
required: [true, "Email is required"],
unique: true,
lowercase: true,
trim: true, },
phone: {
type: String,
required: [true, "Phone is required"],
trim: true },
password: {
type: String,
required: [true, "Password is required"],
minlength: [6, "Password must be at least 6 characters long"], },
cartItems: [ {
quantity: {
type: Number,
default: 1, },
product: {
type: mongoose.Schema.Types.ObjectId,
ref: "Product", },
}, ],
```

```
wishlist: [ {
type: mongoose.Schema.Types.ObjectId,
ref: "Product", },
],
resetPasswordCode: {
type: String,
default: null, },
resetPasswordExpires: {
type: Date,
default: null, },
role: {
type: String,
enum: ["customer", "admin"],
default: "customer", },
isBanned: {
type: Boolean,
default: false
}, },
{
timestamps: true,
});
userSchema.pre("save", async function () {
if (!this.isModified("password")) return;
try {
const salt = await bcrypt.genSalt(10);
this.password = await bcrypt.hash(this.password, salt);
} catch (error) {
throw error;}
});
userSchema.methods.comparePassword = async function (password) {
return bcrypt.compare(password, this.password);
```

```
};
const User = mongoose.model("User", userSchema);
export default User;
```

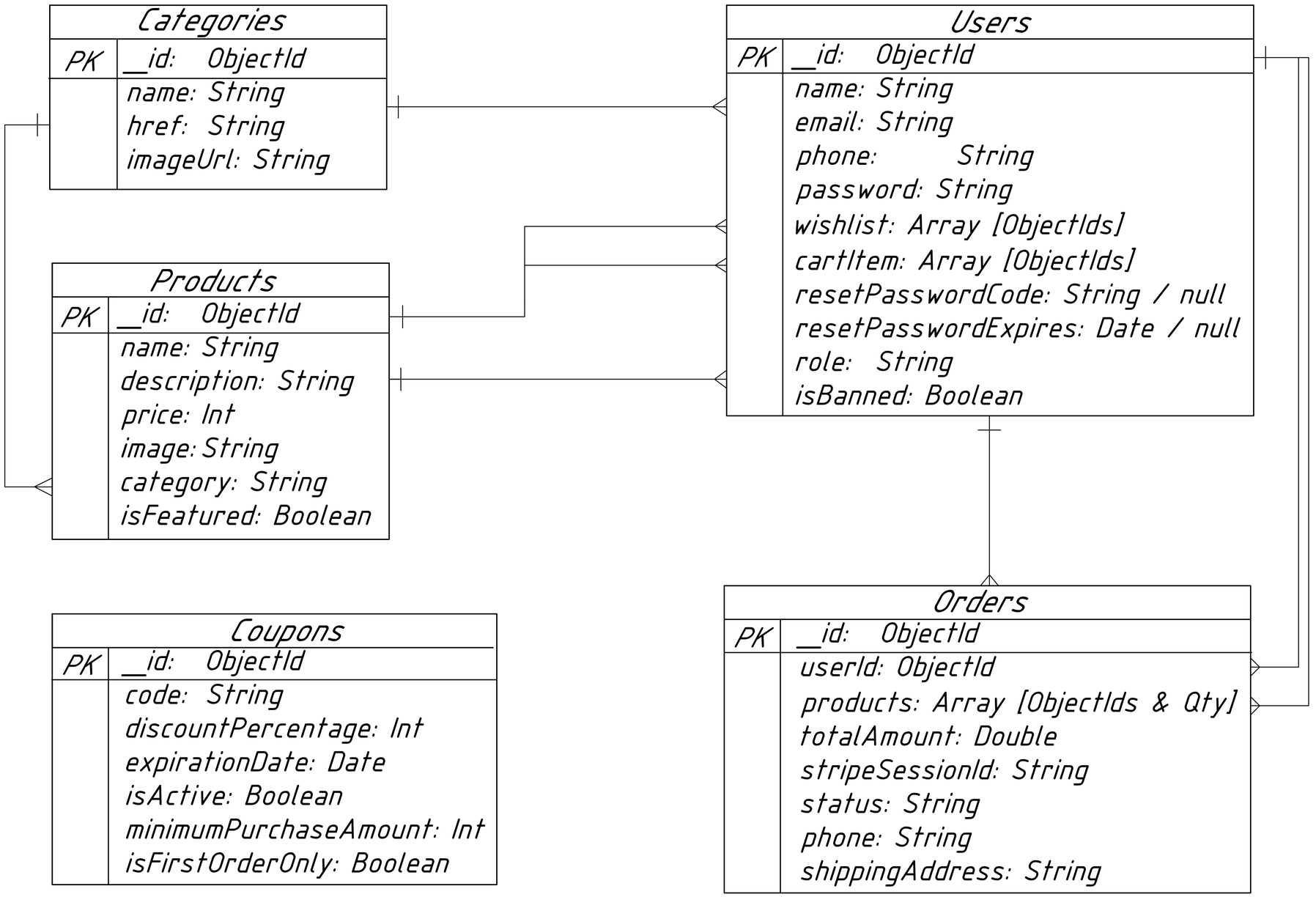
Додаток Н – Лістинг файлу App.test.jsx

```
import { vi, describe, test, expect } from 'vitest';
import { render, screen } from '@testing-library/react';
import { MemoryRouter } from 'react-router-dom';
import App from './App';
import * as matchers from '@testing-library/jest-dom/matchers';
expect.extend(matchers);
vi.mock('./stores/useUserStore', () => ({
useUserStore: () => ({ checkAuth: vi.fn(), checkingAuth: false, user: null, wishlist: [] })}));
vi.mock('./stores/useCartStore', () => ({
useCartStore: () => ({ getCartItems: vi.fn(), cart: [] })
}));
vi.mock('./stores/useProductStore', () => ({
useProductStore: () => ({ fetchFeaturedProducts: vi.fn(), products: [], loading: false })}));
vi.mock('./stores/useCategoryStore', () => ({
useCategoryStore: () => ({ fetchCategories: vi.fn(), categories: [], loading: false })
}));
describe('Modular testing of the main system component', () => {
test('Successful launch and rendering of the application', () => {
render(
<MemoryRouter initialEntries={['/']}>
<App />
</MemoryRouter> );
const textElement = screen.getByText(/Catalog/i);
expect(textElement).toBeInTheDocument();
});
})
```

Додаток О – Лістинг файлу server.test.js

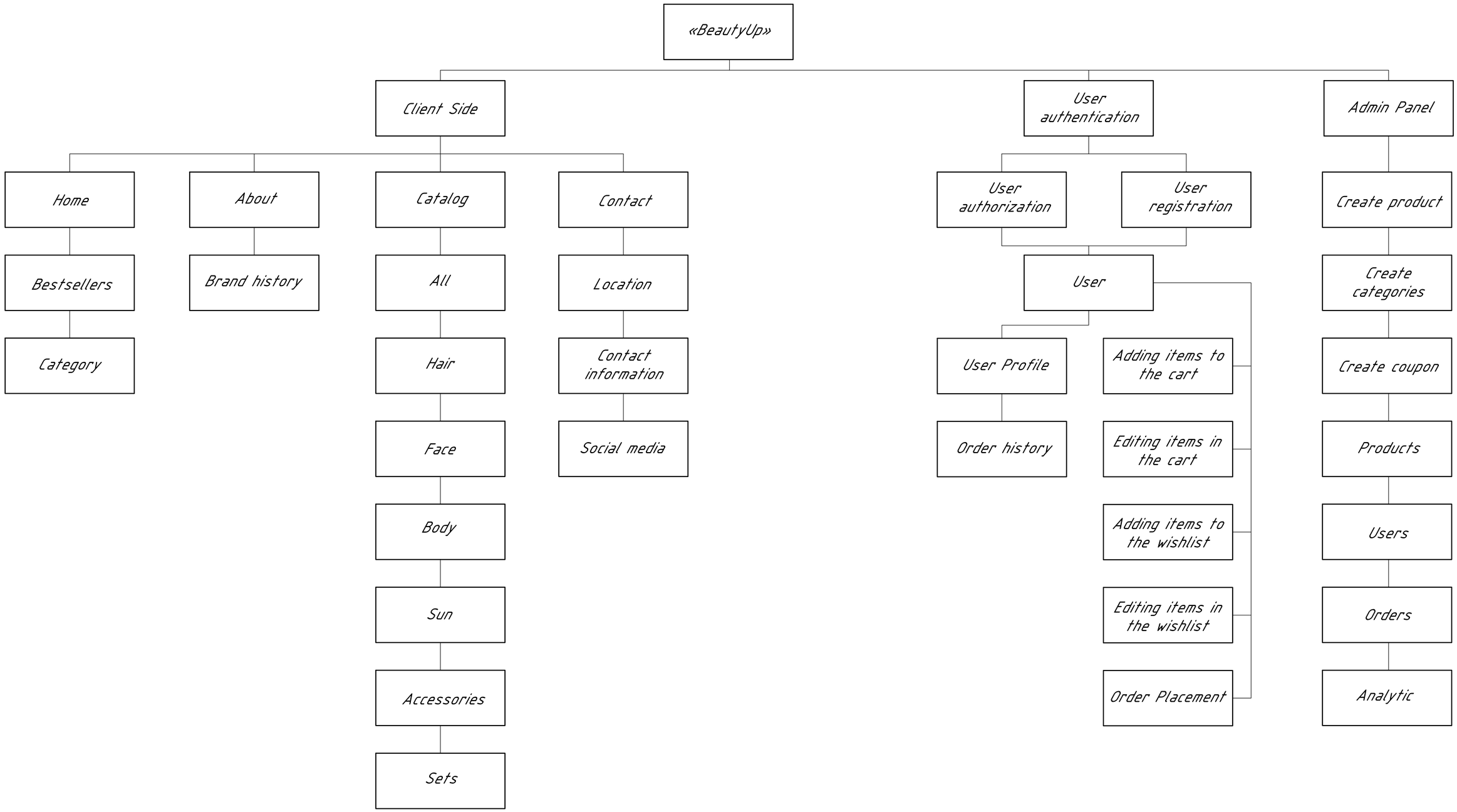
```
import { vi, describe, test, expect } from 'vitest';
import request from 'supertest';
import express from 'express';
import categoryRoutes from './routes/category.route.js';
const testApp = express();
testApp.use(express.json());
testApp.use('/api/categories', categoryRoutes);
vi.mock('./routes/category.route.js', () => ({
default: express.Router().get('/', (req, res) => {
res.status(200).json([
{ _id: "1", name: "Face Care", slug: "face-care" },
{ _id: "2", name: "Hair Care", slug: "hair-care" }
]);
})}));
describe('Integration testing of the server part', () => {
test('GET /api/categories — Successful receipt of the list of product categories', async
() => {
const response = await request(testApp)
.get('/api/categories')
.set('Accept', 'application/json');
expect(response.statusCode).toBe(200);
expect(response.headers['content-type']).toMatch(/json/);
expect(Array.isArray(response.body)).toBe(true);
});
});
```

Інфологічна модель бази даних



Зм.	Арк.	№ документу	Підпис	Дата	2026.КВР.123.406.06.00.00. БД
Розроб.		Головняк С.Р.			Розробка вебсайту магазину косметики «BeautyUp» Бази даних обслуговування
Перевір.		Лісовий В.М.			Літ. / Маса / Масшт.
Т. контр.					Аркуш / Аркушів 1
Н. контр.		Приймак В.А.			ВСП ТФК ТНТУ КІ-406 м.Тернопіль
Затв.					

Структурна схема інтерфейсу інтернет-магазину «BeautyUp»



					2026.КВР.123.406.06.00.00. СС		
Зм.	Арк.	№ документу	Підпис	Дата	Розробка вебсайту магазину косметики «BeautyUp» Схема структурна	Літ.	Маса	Масшт.
Розроб.		Головняк С.Р.						
Перевір.		Лісовий В.М.				Аркуш	Аркушів 1	
Т. контр.						ВСП ТФК ТНТУ КІ-406 м.Тернопіль		
Н. контр.		Приймак В.А.						
Затв.								

Таблиця техніко-економічних показників

№ п.п	Назва показника	Значення	№ п.п	Назва показника	Одиниці виміру	Значення
1	Технології розмітки та стилізації	JSX, Tailwind CSS	7	Трудомісткість розробки	год	116
2	Мова програмування та фреймворки	JavaScript, React.js, Node.js	8	Собівартість	грн	24467,58
3	Менеджер пакетів	npm	9	Плановий прибуток	грн	12233,79
4	База даних	MongoDB	10	Ціна	грн	36701,36
5	Логіка та керування станом	Zustand	11	Чиста теперішня вартість	грн	3464,92
6	Платформа розгортання	Render	12	Термін окупності	рік	2,4

						2026.КВР.123.406.06.00.00. ТБ	
Зм.	Арк.	№ документу	Підпис	Дата	Розробка вебсайту магазину косметики «BeautyUp» Таблиця техніко-економічних показників	Літ.	Маса	Масшт.
Розроб.		Головняк С.Р.						
Перевір.		Лісовий В.М.				Аркуш	Аркушів 1	
Т. контр.								
Н. контр.		Приймак В.А.				ВСП ТФК ТНТУ КІ-406 м.Тернопіль	
Затв.							

Лістинг файлу payment.controller.js

```
import Coupon from "../models/coupon.model.js";
import Order from "../models/order.model.js";
import { stripe } from "../lib/stripe.js";

export const createCheckoutSession = async (req, res) => {
    try {
        const { products, couponCode } = req.body;
        if (!Array.isArray(products) || products.length === 0) {
            return res.status(400).json({ error: "Invalid or empty products array" });
        }
        const baseTotalAmount = products.reduce((sum, product) => {
            const price = Number(product.price) || 0;
            const quantity = Number(product.quantity) || 1;
            return sum + (price * quantity);
        }, 0);
        let discountPercent = 0;
        let appliedCouponCode = "";
        if (couponCode) {
            const coupon = await Coupon.findOne({ code: couponCode, isActive: true });
            if (coupon) {
                const isExpired = new Date() > new Date(coupon.expirationDate);
                const minAmountRequired = Number(coupon.minimumPurchaseAmount) || 0;
                if (!isExpired && baseTotalAmount >= minAmountRequired) {
                    discountPercent = Number(coupon.discountPercentage) || 0;
                    appliedCouponCode = couponCode;
                } else if (isExpired) {
                    console.log(`Coupon ${couponCode} has expired`);
                } else {
                   console.log(`The sum of ${baseTotalAmount.toFixed(2)} is less than the
minimum of (${minAmountRequired})`);
                }
            }
        }
        let totalOrderAmount = 0;
        const lineItems = products.map((product) => {
        const originalPriceInCents = Math.round(Number(product.price) * 100);
        const finalPriceInCents = discountPercent > 0
?       Math.round(originalPriceInCents * (1 - discountPercent / 100))                    {
                : originalPriceInCents;                                                   {
            totalOrderAmount += (finalPriceInCents / 100) * (Number(product.quantity) ||
1);
            return {
                price_data: {
                  currency: "usd",
                  product_data: {
name: product.name,
images:       product.image ? [product.image] : [],},
                    unit_amount: finalPriceInCents,},
                quantity: Number(product.quantity) || 1,};
                });
        const newOrder = new Order({
user:  req.user._id,
            products: products.map((p) => ({
              product: p._id || p.id,
               quantity: Number(p.quantity) || 1,
price: Number(p.price),
            })),
            totalAmount: Number(totalOrderAmount.toFixed(2)),
            stripeSessionId: "temp_" + Date.now(),
phone:       "Pending",
status:      "Pending",
        });
        await newOrder.save();
        const session = await stripe.checkout.sessions.create({
          payment_method_types: ["card"],
          line_items: lineItems,
mode: "payment",
            customer_email: req.user.email,
            phone_number_collection: { enabled: true },
            shipping_address_collection: {
             allowed_countries: ["US", "UA", "GB", "PL", "CA"],
              },
            success_url:
`${process.env.CLIENT_URL}/purchase-success?session_id={CHECKOUT_SESSION_ID}`,
            cancel_url:  `${process.env.CLIENT_URL}/purchase-cancel`,
            metadata: {
                userId: req.user._id.toString(),
                orderId: newOrder._id.toString(),
                couponCode: appliedCouponCode,
              },
        });
        newOrder.stripeSessionId = session.id;
        await newOrder.save();
        res.status(200).json({ id: session.id, url: session.url });
    } catch (error) {
        console.error("Error creating checkout session:", error);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};
```

```
export const checkoutSuccess = async (req, res) => {
    try {
        const { sessionId } = req.body;
        if (!sessionId) return res.status(400).json({ message: "Session ID is
required" });
        const session = await stripe.checkout.sessions.retrieve(sessionId);
        if (session.payment_status === "paid") {
            const orderId = session.metadata.orderId;
            const order = await Order.findById(orderId);
            if (!order) {
                return res.status(404).json({ message: "Order not found" });
            }
            if (order.status === "Paid") {
                return res.status(200).json({ success: true, orderId: order._id });
            }
            const shipping = session.shipping_details;
            const customer = session.customer_details;
            const addr = shipping?.address || customer?.address;
            let addressStr = "Address not specified";
            if (addr) {
                addressStr = [
                    addr.line1,
                    addr.line2,
                    addr.city,
                    addr.state,
                    addr.postal_code,
                    addr.country
                ].filter(Boolean).join(", ");
            }
            const stripeName = shipping?.name || customer?.name || "Client";
            order.shippingAddress = `${stripeName} | ${addressStr}`;
            order.phone = customer?.phone || shipping?.phone || "No phone";
            order.status = "Paid";
            order.stripeSessionId = sessionId;
            await order.save();
            if (session.metadata.couponCode) {
                await Coupon.findOneAndUpdate(
    code: session.metadata.couponCode },
    $inc: { uses: 1 } }
                );
            }
            res.status(200).json({ success: true, orderId: order._id });
          } else {
            res.status(400).json({ message: "Payment not confirmed" });
          }
        } catch (error) {
            console.error("CRITICAL ERROR IN CHECKOUT SUCCESS:", error);
            res.status(500).json({ message: "Server error", error: error.message });
        }
};
```

2026.КВР.123.406.06.00.00. ТП

					2026.КВР.123.406.06.00.00. ТП			
Зм.	Арк.	№ документу	Підпис	Дата	Розробка вебсайту магазину косметики «BeautyUp» Текст програми	Літ.	Маса	Масшт.
Розроб.		Головняк С.Р.						
Перевір.		Лісовий В.М.				Аркуш	Аркушів 1	
Т. контр.								
Н. контр.		Приймак В.А.				ВСП ТФК ТНТУ КІ-406 м.Тернопіль		
Затв.								