Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___» _____ 202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
«ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНТЕРНЕТ-МАГАЗИНУ ВЕСІЛЬНОГО ОДЯГУ ТА АКСЕСУАРІВ»

зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня бакалавра

Виконавець роботи Пригода Олександра Володимирівна

_____ «___» _____ 202_ р.
(підпис)

Науковий керівник к.ф.-м.н., доц., Ольховська Олена Володимирівна

_____ «___» _____ 202_ р.
(підпис)

Рецензент _____

ПОЛТАВА 2026

ЗАТВЕРДЖУЮ
Завідувач кафедри _____ Олена ОЛЬХОВСЬКА
«____» вересня 202__ р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Проектування та розробка інтернет-магазину весільного одягу та аксесуарів»
зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавра
Прізвище, ім'я, по батькові Пригода Олександра Володимирівна
Затверджена наказом ректора № ___-Н від «__» _____ 202__ р.
Термін подання студентом роботи «__» _____ 202__ р.
Вихідні дані до кваліфікаційної роботи: статті та документації з теми розробки.
Зміст пояснювальної записки:
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ
ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1. Огляд існуючих рішень інтернет-магазинів весільних товарів
2.2. Підсумки аналізу та визначення вимог до системи
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Архітектура веб-додатку
3.2. Обґрунтування вибору технологій розробки
3.3. Проектування бази даних
3.4. Аутентифікація та безпека веб-додатку
4. ПРАКТИЧНА ЧАСТИНА
4.1. Налаштування середовища розробки
4.2. Реалізація структури та міграцій бази даних
4.3. Імплементація серверного API та бізнес-логіки
4.4. Побудова користувацького інтерфейсу
4.5. Тестування адаптивності та аналіз швидкодії
4.6. Опис функціональних можливостей додатку
ВИСНОВКИ
СПИСОК ЛІТЕРАТУРИ
ДОДАТОК А
Перелік графічного матеріалу: 41 рисунок.

Консультанти розділів кваліфікаційної роботи

Розділ	ПІБ, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі			
Інформаційний огляд			
Теоретична частина			
Практична реалізація			

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3. Постановка завдання		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6. Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доопрацювання (за необхідності), рецензування		

Дата видачі завдання «____»_____ 202__ р.

Здобувач вищої освіти Пригода Олександра Володимирівна

Науковий керівник к. ф.-м. н., Ольховська Олена Володимирівна

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____

(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____»_____ 202__ р.

Секретар ЕК _____ _____

(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к. ф.-м. н. Олена ОЛЬХОВСЬКА
«____» _____ 202_ р

Погоджено
Науковий керівник _____
_____
«____» _____ 202_ р

План

кваліфікаційної роботи на тему
«Проектування та розробка інтернет-магазину весільного одягу та аксесуарів»
зі спеціальності 122 Комп’ ютерні науки
освітня програма 122 «Комп’ютерні науки»
ступеня бакалавр
Прізвище, ім’я, по батькові Пригода Олександра Володимирівна

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ
ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1. Огляд існуючих рішень інтернет-магазинів весільних товарів
2.2. Підсумки аналізу та визначення вимог до системи
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Архітектура веб-додатку
3.2. Обгрунтування вибору технологій розробки
3.3. Проектування бази даних
3.4. Аутентифікація та безпека веб-додатку
4. ПРАКТИЧНА ЧАСТИНА
4.1. Налаштування середовища розробки
4.2. Реалізація структури та міграцій бази даних
4.3. Імплементація серверного API та бізнес-логіки
4.4. Побудова користувацького інтерфейсу
4.5. Тестування адаптивності та аналіз швидкодії
4.6. Опис функціональних можливостей додатку
ВИСНОВКИ
СПИСОК ЛІТЕРАТУРИ
ДОДАТОК А

Здобувач вищої освіти _____ Олександра ПРИГОДА
«____» _____ 202_ р.

РЕФЕРАТ

Записка: 47 стр., 41 Рисунок, 1 додаток, 23 джерела.

ВЕСІЛЬНИЙ ІНТЕРНЕТ-МАГАЗИН, ВЕБ-ДОДАТОК, REST API, АУТЕНТИФІКАЦІЯ, УПРАВЛІННЯ ЗАМОВЛЕННЯМИ

Об'єкт розробки – веб-додаток інтернет-магазину весільних суконь, костюмів та аксесуарів з повним циклом оформлення замовлень.

Мета роботи – розробка веб-додатку, що дозволяє користувачам переглядати каталог товарів, формувати кошик, оформлювати замовлення та керувати особистим кабінетом.

Методи дослідження та інформаційне забезпечення – середовище розробки Visual Studio Code, мова програмування TypeScript, бібліотека React, фреймворк NestJS, ORM Prisma, СУБД PostgreSQL, інструмент збірки Vite, бібліотека компонентів Material UI.

Результати дослідження. У межах роботи проаналізовано існуючі рішення у сфері весільної електронної комерції та визначено їхні переваги й недоліки. Розроблено клієнт-серверний додаток із модульною архітектурою на основі NestJS та компонентним підходом на React. Реалізовано систему аутентифікації з OTP-підтвердженням, каталог товарів із фільтрацією та пошуком, кошик, оформлення замовлень і повноцінний особистий кабінет. Забезпечено адаптивність інтерфейсу та оптимізацію продуктивності.

В результаті тестування підтверджено коректність роботи основних функціональних модулів: аутентифікації, каталогу, кошика та оформлення замовлень.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ **7**
ВСТУП **8**
1. ПОСТАНОВКА ЗАДАЧІ **10**
2. ІНФОРМАЦІЙНИЙ ОГЛЯД **12**
2.1. Огляд існуючих рішень інтернет-магазинів весільних товарів 12
2.2. Підсумки аналізу та визначення вимог до системи 14
3. ТЕОРЕТИЧНА ЧАСТИНА **17**
3.1. Архітектура веб-додатку 17
3.2. Обґрунтування вибору технологій розробки 18
3.3. Проектування бази даних 19
3.4. Аутентифікація та безпека веб-додатку 20
4. ПРАКТИЧНА ЧАСТИНА **22**
4.1. Налаштування середовища розробки 22
4.2. Реалізація структури та міграцій бази даних 25
4.3. Імплементація серверного API та бізнес-логіки 28
4.4. Побудова користувацького інтерфейсу 32
4.5. Тестування адаптивності та аналіз швидкодії 42
4.6. Опис функціональних можливостей додатку 44
ВИСНОВКИ **46**
СПИСОК ЛІТЕРАТУРИ **47**
ДОДАТОК А. ПРОГРАМНИЙ КОД **49**

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
API	Application Programming Interface
REST	Representational State Transfer
HTTP / HTTPS	HyperText Transfer Protocol / Secure
JSON	JavaScript Object Notation
JWT	JSON Web Token
OTP	One-Time Password
ORM	Object-Relational Mapping
СУБД	Система управління базами даних
DTO	Data Transfer Object
UUID	Universally Unique Identifier
CRUD	Create, Read, Update, Delete
UI / UX	User Interface / User Experience
CORS	Cross-Origin Resource Sharing
XSS	Cross-Site Scripting
HMR	Hot Module Replacement
CSV	Comma-Separated Values
URL	Uniform Resource Locator
SEO	Search Engine Optimization
FCP	First Contentful Paint
LCP	Largest Contentful Paint
CLS	Cumulative Layout Shift
TBT	Total Blocking Time
IP	Internet Protocol
CDN	Content Delivery Network
pnpm	Performant Node Package Manager

ВСТУП

Стрімкий розвиток інтернет-технологій суттєво змінив підходи до пошуку товарів та здійснення покупок в онлайн-середовищі. Веб-застосунки стали невід'ємною частиною повсякденного життя, забезпечуючи швидкий та зручний доступ до різноманітних торговельних платформ. Одним із перспективних напрямів є інтернет-магазини весільної продукції, що особливо актуально для покупців, які прагнуть зручно підібрати весільні сукні та аксесуари без необхідності відвідування фізичних салонів.

Актуальність цієї роботи зумовлена зростаючою потребою користувачів у зручних онлайн-інструментах для підбору весільної продукції. Із розширенням асортименту весільних товарів та зростанням популярності онлайн-шопінгу виникає необхідність у сервісах, які автоматизують процес пошуку та порівняння товарів. Традиційний підхід вимагає відвідування численних салонів і значних часових витрат, тоді як веб-додаток, що забезпечує гнучку фільтрацію за категоріями, розмірами та ціною, систему відгуків та повноцінне оформлення замовлення, значно спрощує завдання та покращує користувацький досвід при виборі весільних товарів.

Мета кваліфікаційної роботи — створення інтерактивного веб-додатку інтернет-магазину весільної продукції, який дозволяє користувачам зручно переглядати каталог товарів, здійснювати підбір за допомогою системи фільтрів, оформлювати замовлення та керувати особистим кабінетом. Такий інструмент має на меті спростити процес придбання весільних товарів, зробити його доступним для широкого кола покупців та забезпечити високий рівень безпеки та продуктивності платформи.

Об'єкт дослідження — процес розробки веб-застосунку, орієнтованого на онлайн-продаж весільної продукції.

Предмет дослідження — програмні механізми побудови клієнт-серверної архітектури, реалізації системи фільтрації товарів, JWT-аутентифікації та адаптивного інтерфейсу у межах веб-застосунку.

Новизна роботи — полягає у комплексному підході до вирішення задачі онлайн-продажу весільної продукції, з інтеграцією гнучкої системи фільтрації, захищеної аутентифікації на основі JWT-токенів, хмарного зберігання зображень та повнофункціонального особистого кабінету в єдиному застосунку.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновку, списку використаної літератури та додатків. Обсяг пояснювальної записки становить 91 сторінка, з них основна частина — 47 сторінок, додатки — 44 сторінки, джерела — 23 назви.

1. ПОСТАНОВКА ЗАДАЧІ

У межах кваліфікаційної роботи створено веб-додаток для онлайн-магазину весільної продукції. Головне призначення — надати користувачам можливість зручного пошуку, підбору та замовлення весільних суконь і аксесуарів із використанням гнучкої системи фільтрів за категоріями, розмірами та ціновим діапазоном. Платформа розрахована на покупців, оптимізуючи процес вибору та придбання товарів.

Перед початком реалізації було досліджено існуючі рішення та сформовано перелік функціональних і нефункціональних вимог. До основних вимог належать:

- адаптивність — забезпечення коректного відображення на мобільних пристроях, планшетах та настільних комп'ютерах;
- інтуїтивна навігація — чітка організація розділів: каталог, картка товару, кошик, оформлення замовлення, профіль користувача, аутентифікація;
- гнучка фільтрація — можливість звуження вибірки товарів за категорією, розміром та діапазоном цін;
- захищеність — реалізація аутентифікації на основі JWT-токенів, хешування паролів та механізму відновлення доступу через одноразовий код (OTP);
- продуктивність — мінімізація часу завантаження за рахунок відкладеного завантаження компонентів та оптимізації медіафайлів;
- кросбраузерність — стабільна робота у всіх сучасних браузерах;
- масштабованість — модульна побудова клієнтської та серверної частин, що забезпечує розширення функціоналу без перебудови існуючої архітектури;
- естетичність інтерфейсу — сучасне візуальне оформлення із застосуванням принципів UX/UI на базі бібліотеки Material UI.

Згідно зі спроектованою структурою, додаток охоплює наступні ключові розділи: головну сторінку з промо-банером та добіркою популярних позицій, каталог із механізмами фільтрації та сортування, сторінку окремого товару з фотогалереєю та блоком відгуків, кошик та процес оформлення покупки, модуль аутентифікації (реєстрація, вхід, відновлення паролю), а також персональний кабінет із розділами замовлень, адрес доставки, платіжних методів, списку бажань та відгуків.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Огляд існуючих рішень інтернет-магазинів весільних товарів

Перед початком розробки веб-додатку було проведено аналіз існуючих онлайн-платформ, що спеціалізуються на продажі весільної продукції. Метою дослідження було виявлення сильних та слабких сторін конкурентних рішень для формування оптимального набору функціональних можливостей власного додатку.

Wedboom.UA (wedboom.com.ua) — український інтернет-магазин весільних та вечірніх суконь. Пропонує каталог з категоріями (весільні, вечірні, «мама-дочка»), відображення цін і знижок, актуальні колекції від різних брендів. З недоліків — відсутні фільтри за розміром і ціною, система відгуків та особистий кабінет з історією замовлень (рис. 2.1).

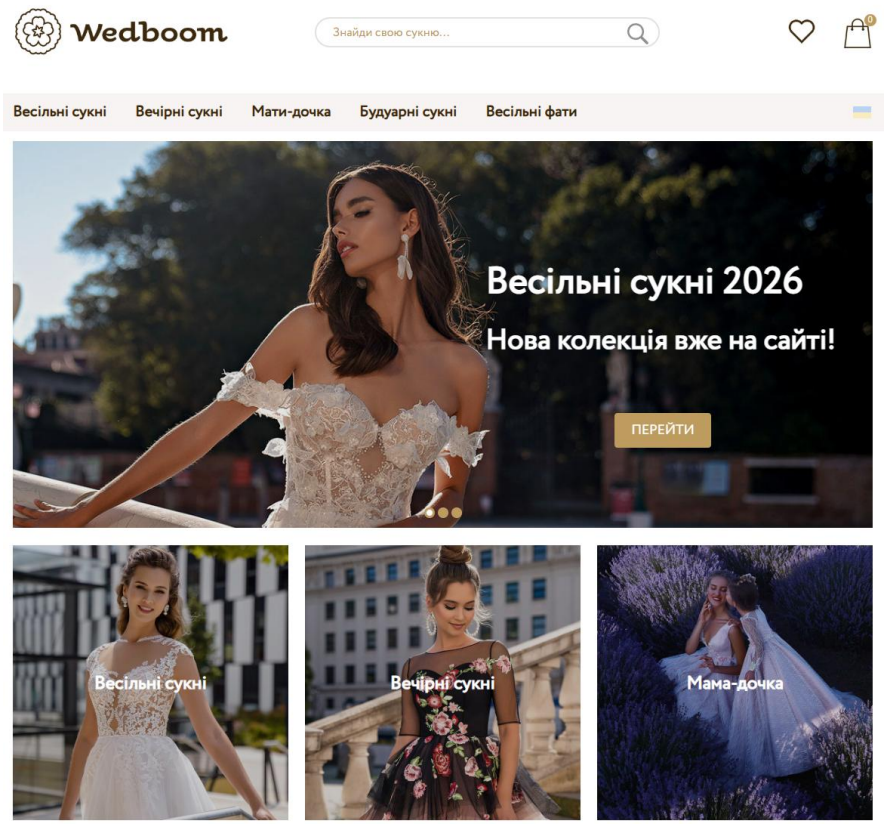

Рисунок 2.1 – Головна сторінка Wedboom.UA

Azazie (azazie.com) — міжнародний онлайн-магазин весільних та вечірніх суконь з широким асортиментом. Платформа вирізняється розвиненою системою фільтрації: за силуетом, довжиною, рукавами, вирізом, тканиною, стилем, ціною та кольором. Серед додаткових можливостей — пошиття за індивідуальними мірками, програма домашньої примірки, списки обраного та шоурум. Недоліками є відсутність локалізації для українського ринку та відсутність повноцінного модуля управління адресами доставки й способами оплати у профілі (рис. 2.2).

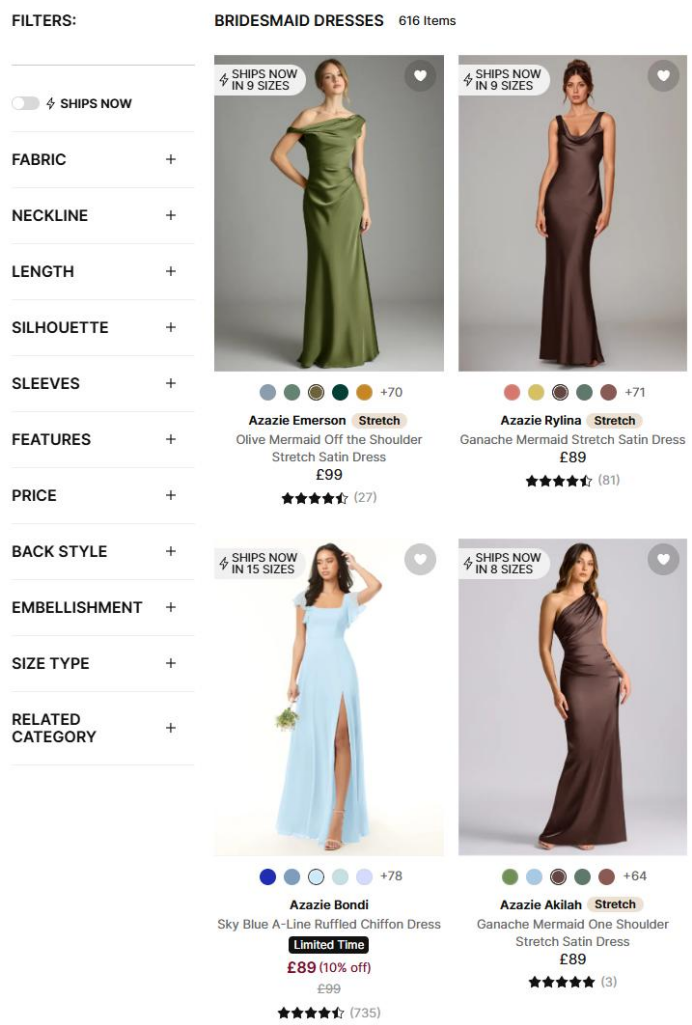

Рисунок 2.2 – Каталог Azazie з фільтрами

David's Bridal (davidsbridal.com) — один із найбільших американських ритейлерів весільної моди з онлайн-присутністю. Сайт пропонує бронювання безкоштовних примірок, великий каталог суконь із фільтрацією за розміром,

стилем та ціною, а також інтеграцію з офлайн-салонами. До переваг належить зручна навігація та розвинена інфраструктура обслуговування клієнтів. Серед недоліків — перевантажений інтерфейс та відсутність модулю списку бажань із прив'язкою до облікового запису (рис. 2.3).

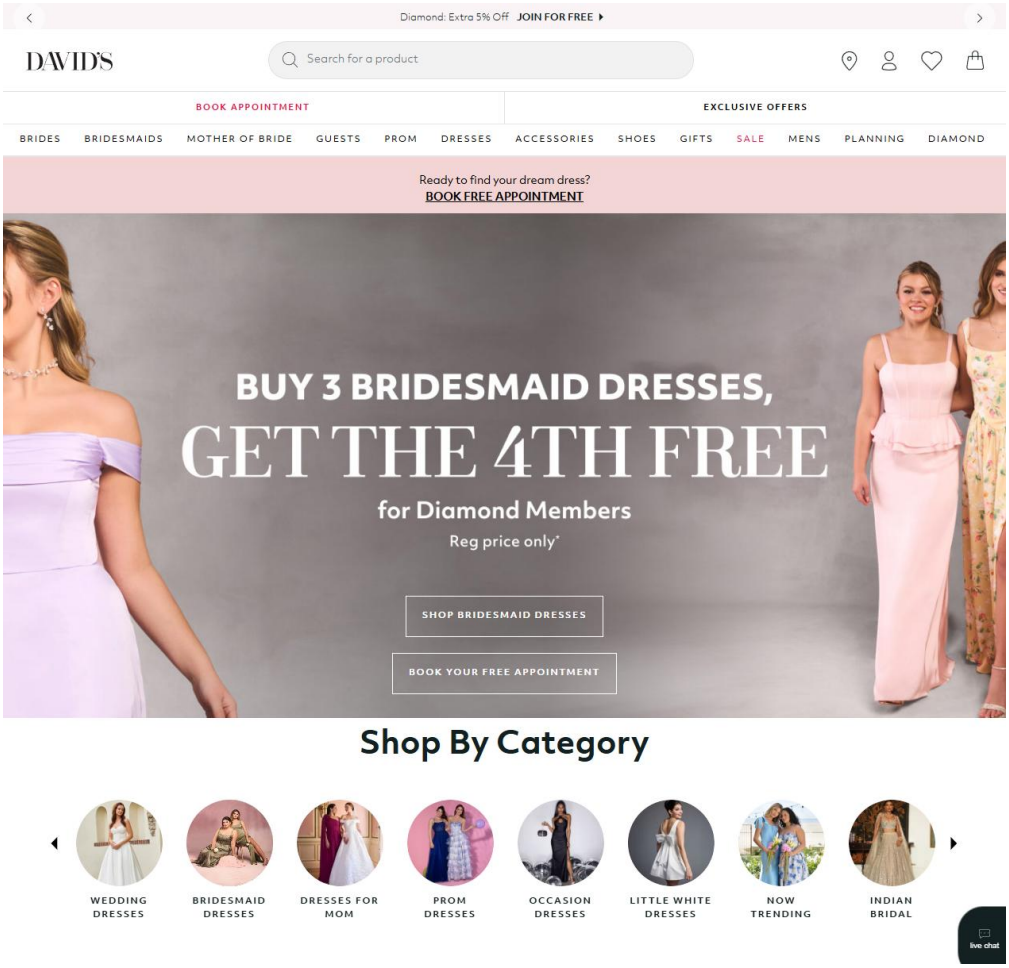

Рисунок 2.3 – Головна сторінка David's Bridal

2.2. Підсумки аналізу та визначення вимог до системи

За результатами аналізу встановлено, що існуючі рішення мають низку обмежень, які знижують зручність користувацького досвіду. Зокрема, більшість платформ не поєднують одночасно розвинену фільтрацію, повноцінний особистий кабінет, систему відгуків та гнучке управління замовленнями в одному додатку.

На підставі проведеного дослідження визначено перелік ключових вимог до розроблюваної системи (рис. 2.4):

- наявність каталогу товарів із гнучкою фільтрацією за категоріями, розмірами та ціновим діапазоном;
- можливість сортування товарів за різними критеріями;
- детальна картка товару з галереєю зображень, описом, вибором розміру та блоком відгуків;
- функціонал кошика з можливістю зміни кількості та розміру товарів;
- повноцінний процес оформлення замовлення з вибором адреси доставки та способу оплати;
- система аутентифікації з реєстрацією, авторизацією та відновленням паролю;
- особистий кабінет із розділами: замовлення, адреси доставки, способи оплати, список бажань, відгуки, налаштування профілю та управління сесіями;
- адаптивний інтерфейс, що коректно працює на пристроях різних типів;
- сучасний дизайн із дотриманням принципів UX/UI.

Критерій	Wedboom.UA	Azazie	David's Bridal	Sandrela
Фільтрація за категоріями	+	+	+	+
Фільтрація за розміром	–	+	+	+
Фільтрація за ціною	–	+	+	+
Сортування товарів	–	+	+	+
Система відгуків	–	+	–	+
Список бажань	–	+	–	+
Особистий кабінет	–	+/–	+	+
Управління адресами доставки	–	–	–	+
Управління способами оплати	–	–	–	+
Адаптивний інтерфейс	+/–	+	+	+
Галерея зображень товару	+	+	+	+

Рисунок 2.4 – Порівняльна таблиця аналогів

Таким чином, розроблюваний додаток має об'єднати переваги досліджених аналогів та усунути їхні основні недоліки, забезпечивши повний цикл взаємодії покупця з інтернет-магазином весільних товарів.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Архітектура веб-додатку

Веб-додаток побудовано за клієнт-серверною архітектурою: клієнтська частина відповідає за відображення інтерфейсу, серверна — за бізнес-логіку, базу даних і REST API (рис. 3.1).

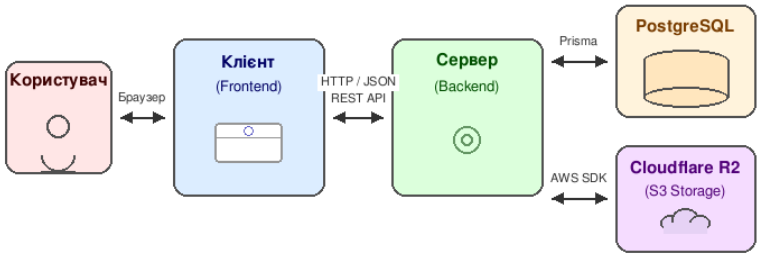

Рисунок 3.1 – Діаграма клієнт-серверної архітектури

Взаємодія відбувається через HTTP(S) у форматі JSON за адресами з префіксом /api/v1. Це забезпечує незалежну розробку обох частин і спрощує масштабування.

Серверна частина побудована на NestJS [1]: кожен функціональний блок (аутентифікація, товари, кошик, замовлення, відгуки) оформлено як окремий модуль зі своїми контролерами, сервісами та DTO. Це забезпечує високу згуртованість коду всередині модулів та слабку зв'язність між ними.

Клієнтська частина організована на React за компонентним підходом з розподілом на шари: сторінки, компоненти, API-шар, глобальний стан (Redux) [2] та утиліти.

Фізична структура проекту складається з двох основних директорій — client та server, кожна з яких має власні залежності, конфігурацію збірки та скрипти запуску.

3.2. Обґрунтування вибору технологій розробки

Клієнтська частина реалізована на React — бібліотеці з компонентним підходом і Virtual DOM для ефективного оновлення інтерфейсу. Мовою розробки обрано TypeScript [3], що забезпечує статичну типізацію, зменшує кількість помилок та спрощує підтримку проекту. Збірку здійснює Vite [4] — сучасний інструмент із нативною підтримкою ES-модулів, що забезпечує швидкий dev-сервер і оптимізовану продакшн-збірку (рис. 3.2).

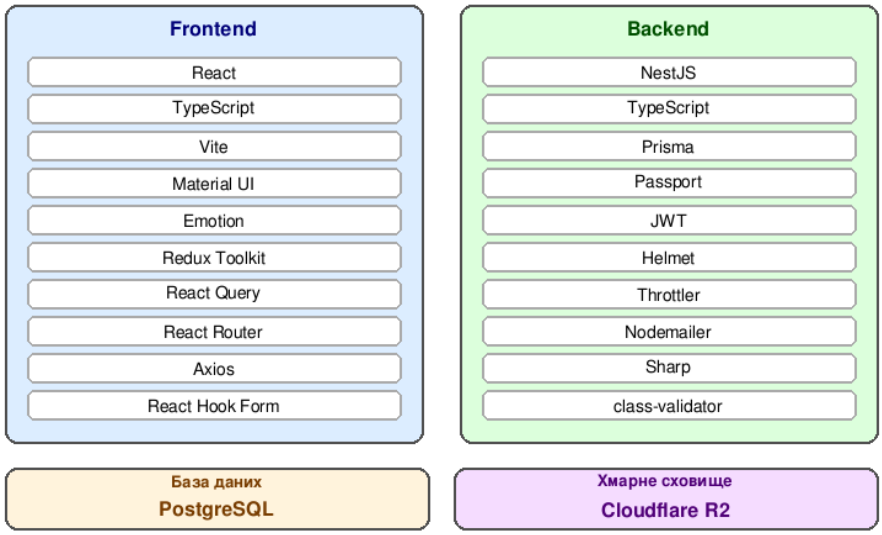

Рисунок 3.2 – Стек технологій проекту

Для стилізації використовується Material UI [5] — бібліотека адаптивних компонентів на основі принципів Material Design з підтримкою кастомної теми. Управління станом розділено між Redux Toolkit (глобальний стан) і TanStack React Query (серверний стан і кешування), що дозволяє ефективно працювати як з локальними даними, так і з відповідями сервера.

Серверна частина побудована на NestJS — Node.js-фреймворку з модульною архітектурою, декораторами та ін'єкцією залежностей, що забезпечує чітку структуру і масштабованість. Для доступу до бази даних використовується Prisma [6] — ORM із типобезпечним клієнтом, автогенерацією та системою міграцій.

3.3. Проектування бази даних

Для зберігання даних додатку використовується реляційна СУБД PostgreSQL [7]. Вибір обумовлений надійністю, підтримкою складних запитів, транзакцій та масштабованістю. Взаємодія з базою відбувається через ORM Prisma, яка надає декларативний опис моделей у файлі schema.prisma та автоматично генерує SQL-міграції.

Схема бази даних охоплює наступні основні сутності:

- User — зберігає дані користувача (ім'я, email, хеш паролю);
- Session — активні сесії користувача з інформацією про пристрій, браузер, IP та геолокацію;
- Category — категорії товарів (весільні сукні, аксесуари тощо);
- Product — товари з назвою, описом, ціною, знижкою, доступними розмірами та зображеннями;
- ProductImage — галерея зображень товару;
- Review — відгуки користувачів із рейтингом від 1 до 5;
- Cart / CartItem — кошик покупок з прив'язкою до товарів, розмірів та кількості;
- Order / OrderItem — оформлені замовлення з деталями доставки та оплати;
- Wishlist — список бажань користувача;
- ShippingAddress — збережені адреси доставки;
- Payment — збережені способи оплати (картка, PayPal, Amazon);
- PasswordReset — записи для відновлення паролю через OTP-код;
- Log — журнал подій системи для моніторингу та безпеки.

Між сутностями встановлено зв'язки: один-до-багатьох (User → Order, User → Review, Product → ProductImage) та багато-до-багатьох через проміжні таблиці (Cart ↔ Product через CartItem, Order ↔ Product через OrderItem). Для оптимізації запитів додано індекси на ключові поля (userId, productId, price, timestamp) (рис. 3.3).

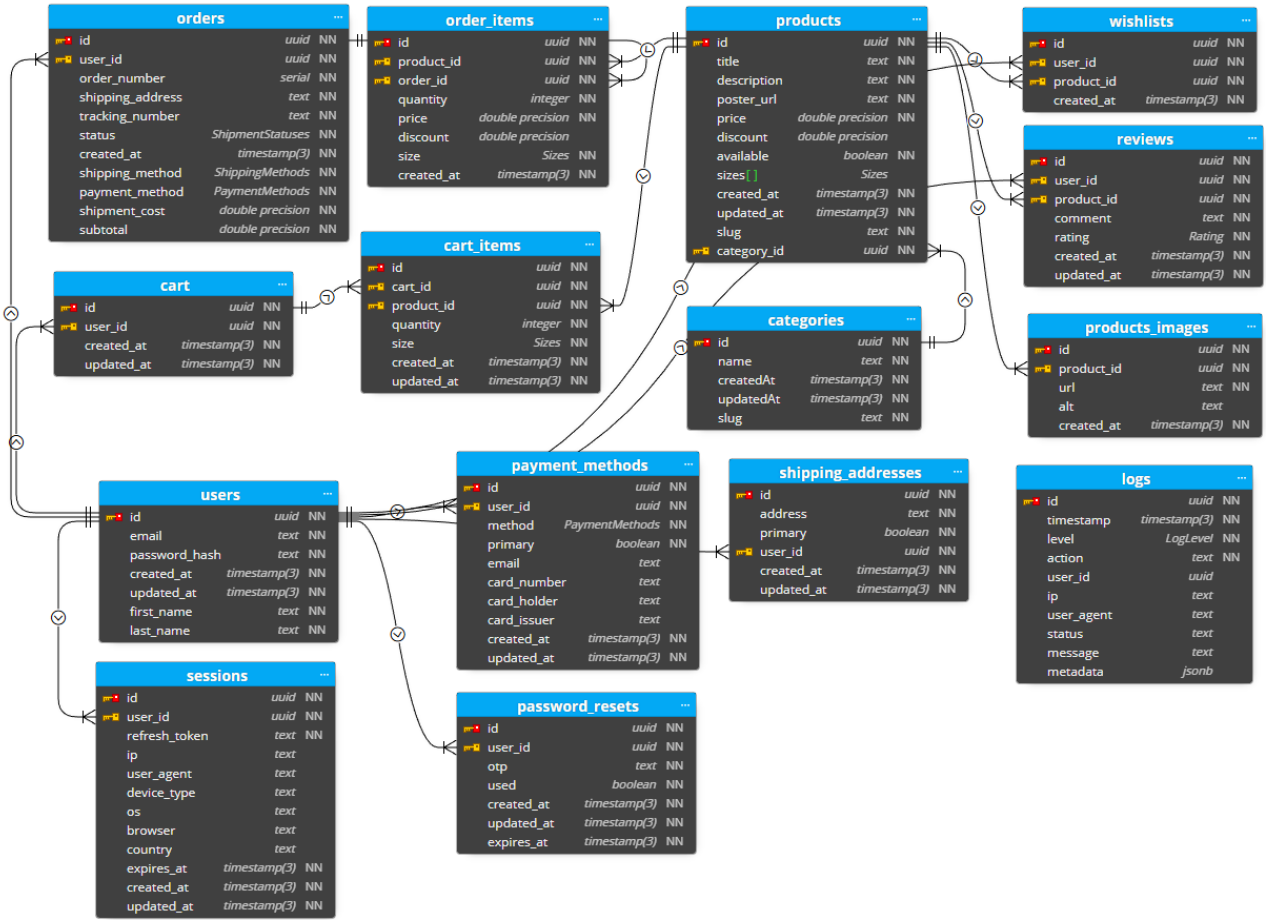

Рисунок 3.3 – ER-діаграма бази даних

3.4. Аутентифікація та безпека веб-додатку

Система аутентифікації побудована на основі JSON Web Tokens із використанням двох типів токенів: access-токен (короткоживучий, для авторизації запитів) та refresh-токен (довгоживучий, для оновлення access-токена без повторного входу). Refresh-токен зберігається у HTTP-only cookie, що захищає його від XSS-атак [8]. Access-токен передається у заголовку Authorization.

Процес аутентифікації реалізовано через модуль Passport.js з кастомними guard'ами (JwtGuard, RefreshGuard), що перевіряють валідність токенів на рівні кожного захищеного маршруту.

Для відновлення паролю реалізовано механізм OTP: користувач отримує одноразовий код на email, підтверджує особу та встановлює новий пароль. Коди мають обмежений термін дії та анулюються після першого використання.

Паролі користувачів хешуються за допомогою бібліотеки bcrypt перед збереженням у базі даних, що унеможливлює їх відновлення навіть у разі витоку бази (рис. 3.4).

Додатково застосовано комплекс заходів безпеки:

- Helmet — встановлення захисних HTTP-заголовків (Content-Security-Policy, X-Frame-Options);
- Throttler — обмеження кількості запитів для захисту від brute-force та DDoS-атак;
- CORS — контроль дозволених джерел запитів для запобігання міждоменним атакам;
- ValidationPipe — валідація та фільтрація вхідних даних за допомогою class-validator для запобігання ін'єкціям;
- Логування подій — запис критичних дій у таблицю Log для аудиту та моніторингу безпеки.

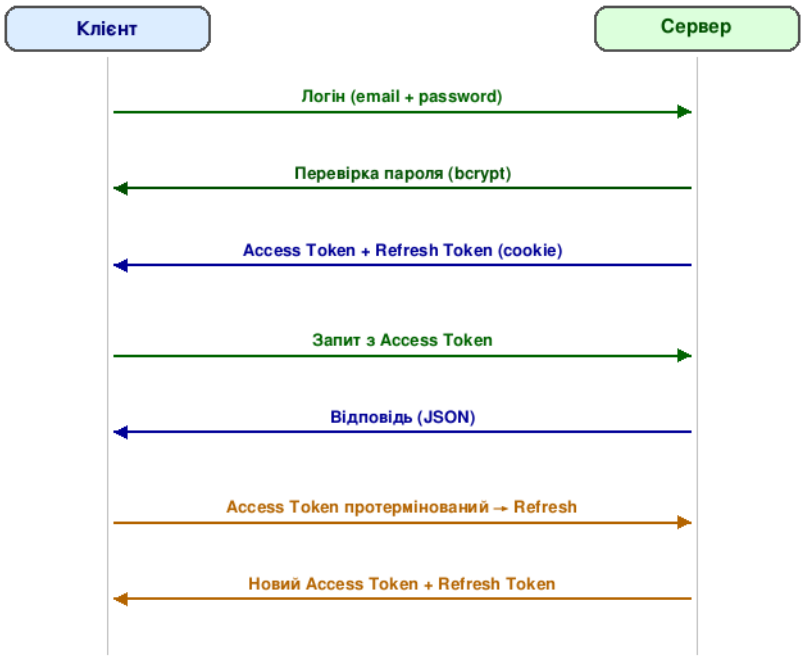

Рисунок 3.4 – Схема процесу аутентифікації (JWT flow)

4. ПРАКТИЧНА ЧАСТИНА

4.1. Налаштування середовища розробки

Для розробки використовувалось таке середовище:

- Node.js — середовище виконання JS для обох частин додатку [9];
- pnpm — менеджер пакетів із ефективним використанням дискового простору;
- Visual Studio Code — редактор із підтримкою TypeScript та інтеграцією Git [10];
- Git — система контролю версій;
- PostgreSQL — реляційна СУБД для локальної розробки та тестування;
- Prisma CLI — інструмент для генерації клієнта БД та управління міграціями.

Проект організовано як монорепозиторій із директоріями client та server, кожна з власним package.json (рис. 4.1 – 4.2).

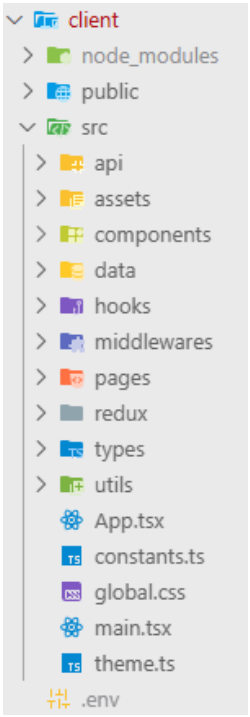

Рисунок 4.1 – Структура папки client

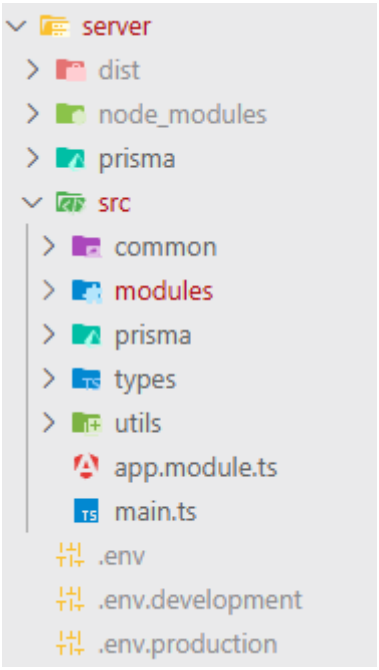

Рисунок 4.2 – Структура папки server

Клієнтський додаток запускається командою pnpm dev, яка піднімає Vite dev-server на порті 5173 з підтримкою HMR для миттєвого оновлення змін у браузері без перезавантаження сторінки. Серверний додаток запускається командою pnpm start:dev, яка запускає NestJS у watch-режимі на порті 3000 з автоматичною перекомпіляцією при зміні файлів (рис. 4.3).

```
ROLLDOWN-VITE v7.1.14  ready in 378 ms

→ Local:   http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
```

Рисунок 4.3 – Запуск dev-серверу Vite

Для конфігурації серверної частини використовуються змінні оточення, що зберігаються у файлах .env. Серед основних змінних: підключення до бази даних (DATABASE_URL), секрети для JWT-токенів (JWT_ACCESS_SECRET, JWT_REFRESH_SECRET), налаштування поштового сервісу (SMTP_HOST, SMTP_USER, SMTP_PASSWORD) та Cloudflare R2 R (R2_ACCESS_KEY_ID, R2_SECRET_ACCESS_KEY, R2_BUCKET) (рис. 4.4).

```
DATABASE_URL="postgresql://neondb_owner:npg_6UARcN40Epzt@ep-super-poettry-ag8oiifo-poole

JWT_ACCESS_SECRET=47039ca84e3f81e66c7980d19e0d340886610ea2
JWT_REFRESH_SECRET=5e8e0d2226169f8ab74ea18e64d04022ffe3e500
JWT_RESET_SECRET=0817f7gf45a9844bb31f523f1a7dec3062603c30

JWT_ACCESS_EXPIRY=900000       # 15 mins
JWT_REFRESH_EXPIRY=604800000   # 7 days

SMTP_USER = "info@sandrela.com"
SMTP_PASS = "Xo20kqP2%kq"

S3_API = https://616cf00cd36cde575517b7ec336ac4cd.r2.cloudflarestorage.com
R2_BUCKET = wedding-shop
R2_TOKEN = bZviwgaFMgzWL3a4UKXLVqIzYX8wydBu0y-geODg
R2_ACCESS_KEY_ID = b5cf140e7194c7d776a18f8d19caa15e
R2_SECRET_ACCESS_KEY = 6f0417b901f18290a4b6509381ed915a74fc78a0c667f5aae8917e13f5c143d8
```

Рисунок 4.4 – Файл змінних оточення .env

Для лінтингу клієнтського коду налаштовано ESLint із підтримкою TypeScript [12] та плагінами react-hooks і react-refresh (рис. 4.5). На серверній частині додатково використовується Prettier для автоматичного форматування коду.

```
import js from "@eslint/js";
import globals from "globals";
import reactHooks from "eslint-plugin-react-hooks";
import reactRefresh from "eslint-plugin-react-refresh";
import tseslint from "typescript-eslint";
import { defineConfig, globalIgnores } from "eslint/config";

export default defineConfig([
  globalIgnores(["dist"]),
  {
    files: ["**/*.{ts,tsx}"],
    extends: [
      js.configs.recommended,
      tseslint.configs.recommended,
      reactHooks.configs["recommended-latest"],
      reactRefresh.configs.vite,
    ],
    languageOptions: {
      ecmaVersion: 2020,
      globals: globals.browser,
    },
    rules: {
      "@typescript-eslint/no-explicit-any": "off",
    },
  },
]);
```

Рисунок 4.5 – Файл налаштування ESLint

4.2. Реалізація структури та міграцій бази даних

Структуру бази даних описано у файлі schema.prisma засобами Prisma Schema Language. Джерелом даних слугує PostgreSQL через змінну оточення DATABASE_URL; генератор prisma-client-js створює типобезпечний TypeScript-клієнт із відображенням усіх моделей, перелічень та зв'язків [13].

Схема містить 5 enum-типів: ShipmentStatuses, PaymentMethods, ShippingMethods, Sizes та Rating — вони забезпечують типізацію значень на рівні бази даних і коду.

Основні моделі та їхні ключові особливості:

– User — центральна модель, що зв'язана з усіма іншими через відношення один-до-багатьох (рис. 4.6). Пароль зберігається у вигляді хешу (passwordHash);

– Product — містить поля price, discount, available, масив sizes та зв'язки із категоріями, зображеннями, відгуками та позиціями кошика (рис. 4.7). Індексовано за полями available та price для оптимізації запитів каталогу;

– Cart / CartItem — реалізовано зв'язок через проміжну таблицю з унікальним обмеженням (cartId, productId, size), що запобігає дублюванню позицій. Видалення кошика каскадно видаляє усі CartItem;

– Order / OrderItem — замовлення з автоінкрементним номером (orderNumber), адресою доставки, способом оплати та статусом. Кожен OrderItem зберігає ціну та знижку на момент замовлення;

– Review — унікальне обмеження (userId, productId) гарантує один відгук від користувача на товар;

– Session — зберігає refresh-токен, інформацію про пристрій (ОС, браузер, тип пристрою), IP та країну для управління активними сесіями;

– Log — журнал подій з індексами на timestamp, level та userId для ефективного пошуку.

```
model User {
  id           String @id @default(uuid()) @db.Uuid
  email        String @unique
  firstName    String @map("first_name")
  lastName     String @map("last_name")
  passwordHash String @map("password_hash")

  session       Session[]
  review        Review[]
  cart          Cart[]
  passwordReset PasswordReset[]
  wishlist      Wishlist[]
  order         Order[]
  payment       Payment[]
  addresses     ShippingAddress[]

  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")

  @@map("users")
}
```

Рисунок 4.6 – Модель User

```
model Product {
  id          String  @id @default(uuid()) @db.Uuid
  title       String
  description String
  slug        String
  posterUrl   String  @map("poster_url")
  price       Float
  discount    Float?
  available   Boolean

  categoryId String   @map("category_id") @db.Uuid
  category   Category @relation(fields: [categoryId], references: [id])

  sizes     Sizes[]        @default([])
  review    Review[]
  cartItem  CartItem[]
  images    ProductImage[]
  wishlist  Wishlist[]
  orderItem OrderItem[]

  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")

  @@unique([slug])
  @@index([available])
  @@index([price])
  @@map("products")
}
```

Рисунок 4.7 – Модель Product

Протягом розробки було створено 19 міграцій, кожна з яких фіксує окрему зміну в структурі бази (рис. 4.8). Серед ключових:

- initialize — початкове створення основних таблиць;
- added_logs — додавання таблиці логування подій;
- shipping_address_model — створення таблиці адрес доставки;
- payment_methods_mode — створення таблиці способів оплати;
- add_product_categories — впровадження системи категорій товарів із slug-полем;
- added_subtotal_and_shipment_cost — додавання полів вартості доставки та проміжної суми.

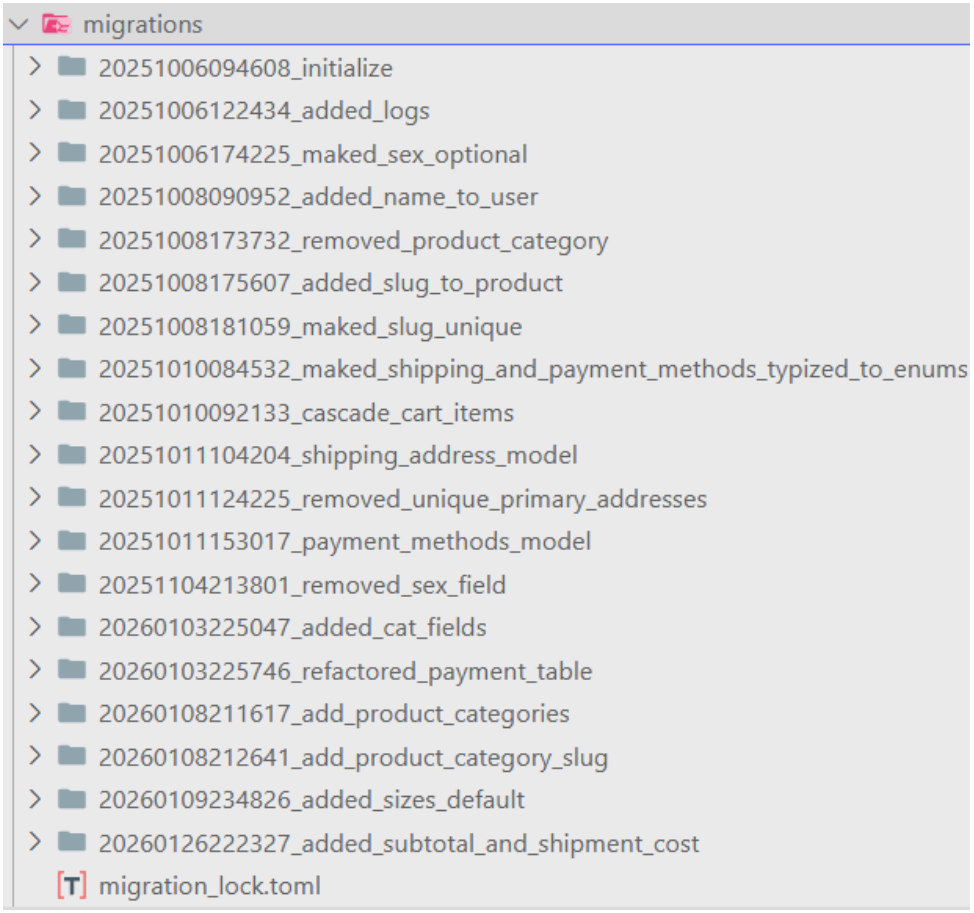

Рисунок 4.8 – Перелік міграцій

Для застосування міграцій використовується команда prisma migrate dev, після чого виконується prisma generate для перегенерації клієнта.

4.3. Імплементація серверного API та бізнес-логіки

Серверна частина побудована на NestJS із дотриманням модульної архітектури (рис. 4.9). Кожен функціональний блок оформлено як окремий модуль із трьома рівнями відповідальності [14]:

- Controller — приймає HTTP-запити, виконує маршрутизацію та валідацію вхідних даних через декоратори;
- Service — реалізує бізнес-логіку та взаємодіє з базою даних через Prisma Client;

- DTO (Data Transfer Object) — описує структуру вхідних даних із правилами валідації за допомогою декораторів class-validator (@IsString, @IsEmail, @IsEnum, @Min, @Max).

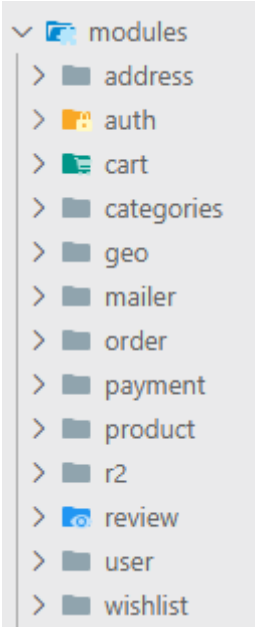

Рисунок 4.9 – Структура модулів серверної частини

Модуль аутентифікації обробляє 7 маршрутів: POST /register, /login, /refresh, /forgot-password, /verify-otp, /reset-password, /change-password (рис. 4.10).

При логіні створюється сесія з інформацією про пристрій, IP-адресою та країною. Токени передаються у httpOnly-cookies із налаштуваннями secure та sameSite відповідно до середовища [15]. Для захисту від перебору паролів застосовано rate limiting через декоратор @Throttle — до 3 спроб на хвилину.

```
@Post('login')
@Throttle({ default: { limit: 3, ttl: 60000 } })
async login(
  @Body() dto: LoginDto,
  @Res({ passthrough: true }) res: Response,
  @Req() req: Request,
  @IpAddress() ip: string,
) {
  const { accessToken, refreshToken, userId } =
    await this.authService.login(dto);

  const userAgent = req.headers['user-agent'] || 'unknown';

  const { os, deviceType, browser } = getDeviceInfo(userAgent);

  await this.authService.createSession({
    userId,
    refreshToken,
    ip: ip === '::1' ? '127.0.0.1' : String(ip),
    userAgent,
    os,
    deviceType,
    browser,
    country: (await this.geoService.getCountryByIp(ip)) ?? 'Unknown',
    expiresAt: new Date(
      Date.now() +
        parseInt(this.configService.get<string>('JWT_REFRESH_EXPIRY')!),
    ),
  });

  const isProd = this.configService.get<string>('NODE_ENV') === 'production';

  res.cookie('accessToken', accessToken, {
    httpOnly: true,
    secure: isProd,
    sameSite: isProd ? 'strict' : 'lax',
    maxAge: isProd
      ? parseInt(this.configService.get<string>('JWT_ACCESS_EXPIRY')!)
      : 15 * 24 * 60 * 60 * 1000,
  });

  res.cookie('refreshToken', refreshToken, {
    httpOnly: true,
    secure: isProd,
    sameSite: isProd ? 'strict' : 'lax',
    maxAge: parseInt(this.configService.get<string>('JWT_REFRESH_EXPIRY')!),
  });

  return { message: 'Login successful' };
}
```

Рисунок 4.10 – Фрагмент коду AuthController (метод login)

Модуль товарів надає маршрути для отримання списку товарів із фільтрацією та пагінацією (GET /products), пошуку за текстом (GET /products/search) і отримання товару за slug (GET /products/by-slug/:slug).

GetAllProductsDto (рис. 4.11) підтримує фільтрацію за ціновим діапазоном, розміром та категорією, а також сортування результатів. Коректність цінового діапазону перевіряє кастомний декоратор @MinLessThanMax.

```
export class GetAllProductsDto extends PaginationDto {
  @Min(0)
  @IsInt()
  @IsOptional()
  @Type(() => Number)
  minPrice?: number;

  @Max(250000)
  @IsInt()
  @IsOptional()
  @Type(() => Number)
  @MinLessThanMax('maxPrice')
  maxPrice?: number;

  @IsOptional()
  @IsEnum(ProductsSortBy)
  sortBy?: ProductsSortBy;

  @IsOptional()
  @IsEnum(Sizes)
  size?: Sizes;

  @IsOptional()
  @IsString()
  category?: string;
}
```

Рисунок 4.11 – Фрагмент коду GetAllProductsDto (валідація фільтрів)

Модуль кошика надає три захищені @Auth() маршрути: оновлення кошика (PATCH /cart), отримання вмісту (GET /cart) та видалення позиції (DELETE /cart). Після кожної мутації повертається оновлений стан кошика.

Модуль замовлень реалізує створення замовлення (POST /orders), отримання списку з пагінацією (GET /orders) та експорт замовлень у CSV (GET /orders/csv/:id, GET /orders/csv) за допомогою бібліотеки json2csv.

Модуль завантаження зображень відповідає за завантаження зображень у Cloudflare R2 через AWS SDK. Метод uploadFromUrl отримує зображення за URL, конвертує його у WebP через Sharp [16] (якість 80%), генерує UUID-ім'я файлу та зберігає у bucket через PutObjectCommand (рис. 4.12).

```
async uploadFromUrl(imageUrl: string): Promise<string> {
  try {
    const response = await axios.get(imageUrl, {
      responseType: 'arraybuffer',
    });
    const buffer = Buffer.from(response.data);

    const webpBuffer = await sharp(buffer)
      .resize({
        width: 800,
        height: 1000,
        fit: 'inside',
      })
      .webp({ quality: 95 })
      .toBuffer();
    const key = `uploads/${uuidv4()}.webp`;

    await this.s3.send(
      new PutObjectCommand({
        Bucket: this.configService.get<string>('R2_BUCKET'),
        Key: key,
        Body: webpBuffer,
        ContentType: 'image/webp',
      }),
    );

    const url = `${this.configService.get<string>('R2_PUBLIC')}/${key}`;
    return url;
  } catch (error) {
    console.error('❌ Upload failed full error:', error);
    throw new HttpException(`Failed to upload image`, HttpStatus.BAD_REQUEST);
  }
}
```

Рисунок 4.12 – Фрагмент коду R2Service (завантаження зображень)

Модуль поштових сповіщень використовує @nestjs-modules/mailer із шаблонізатором Handlebars; шаблон forgot-password.hbs формує лист із OTP-кодом для відновлення паролю.

У точці входу main.ts налаштовано такі middleware: Helmet, CORS, cookie-parser, request-ip та обмеження розміру тіла запиту до 500 КБ. Глобальний ValidationPipe автоматично валідує та трансформує вхідні дані, а AllExceptionsFilter перехоплює всі помилки й логує їх через LogService.

4.4. Побудова користувацького інтерфейсу

Клієнтську частину побудовано на бібліотеці React [17] із використанням TypeScript [18]. Точкою входу є файл main.tsx, який обгортає додаток провайдерами: ThemeProvider (Material UI), QueryClientProvider (React Query [19] з налаштуванням staleTime: 5 хвилин), Provider (Redux Store) та Toaster (toast-сповіщення через react-hot-toast).

Маршрутизація реалізована через React Router із вкладеними маршрутами та поділом на групи за допомогою layout-компонентів [20]:

- AuthLayout — обгортка для сторінок аутентифікації (без основного хедера та футера);
- HomeLayout — макет для головної сторінки з промо-банером;
- PrimaryLayout — основний макет із шапкою сайту, навігацією та підвалом;
- ProfileLayout — макет особистого кабінету з бічним меню.

Доступ до захищених маршрутів контролюється компонентом PrivateRoute, який перевіряє наявність авторизованого користувача в Redux-стані та перенаправляє на сторінку логіну у разі відсутності. PublicRoute навпаки перенаправляє авторизованих користувачів зі сторінок логіну/реєстрації на головну.

Для оптимізації завантаження усі 20+ сторінок підключено через React.lazy() з динамічним імпортом, що розділяє код на окремі чанки (code splitting) [21]. Додатково у vite.config.ts налаштовано виділення React та React DOM у окремий vendor-чанк.

Головна сторінка. містить промо-банер, секцію популярних товарів у вигляді каруселі, блок категорій із зображеннями та посиланнями на каталог, а також інформаційну секцію про бренд (рис. 4.13 – 4.14).

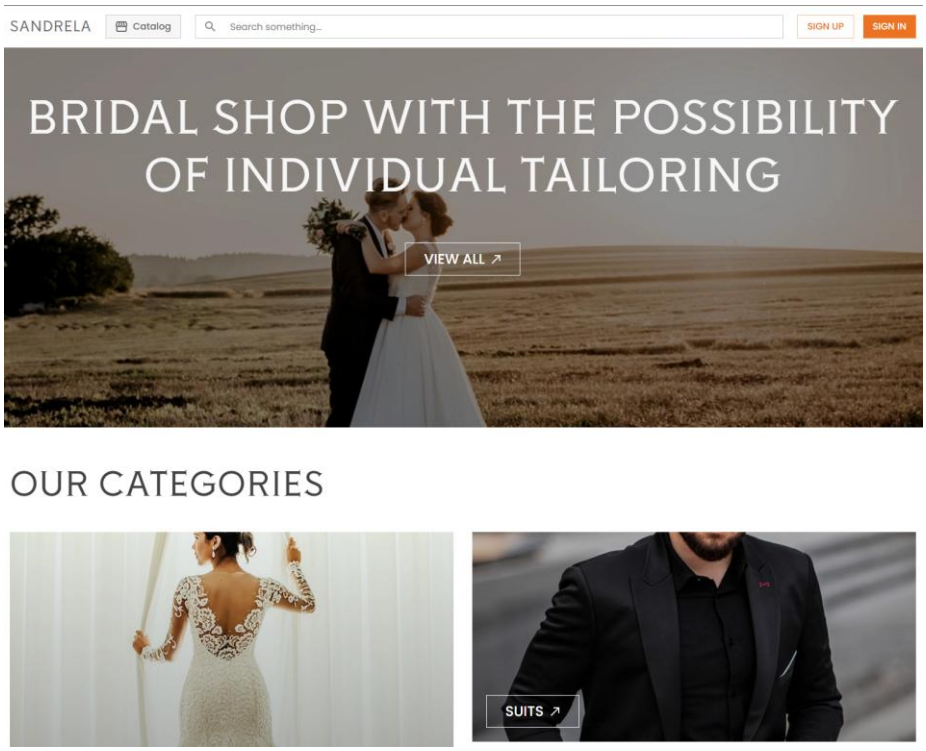

Рисунок 4.13 – Головна сторінка додатку

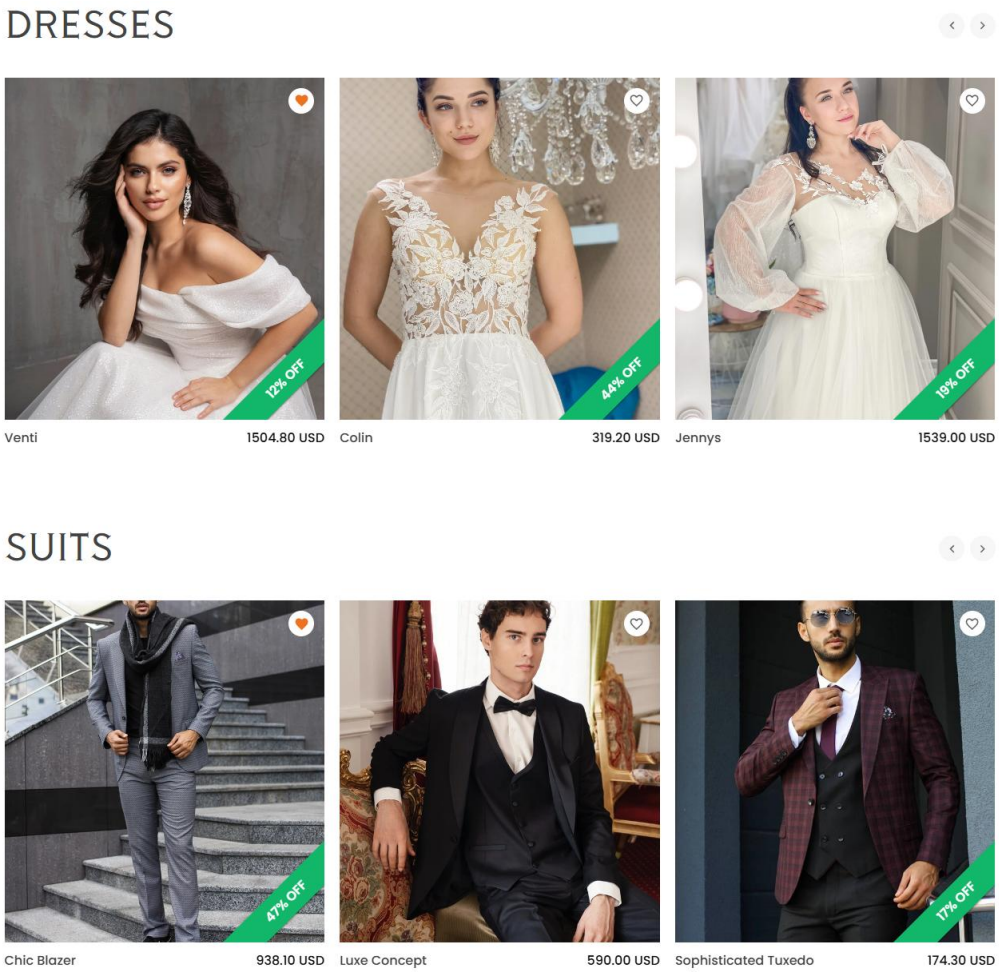

Рисунок 4.14 – Секція популярних товарів

Сторінка каталогу містить бічну панель фільтрів та сітку товарних карток. Доступні фільтри: вибір категорії, слайдер діапазону цін та вибір розміру від XXS до XXXL. Над сіткою розміщено компонент сортування за ціною, новизною та популярністю (рис. 4.15).

Кожен товар відображається у ProductCard із фотографією, назвою, ціною, позначкою знижки та кнопкою додавання до списку бажань.

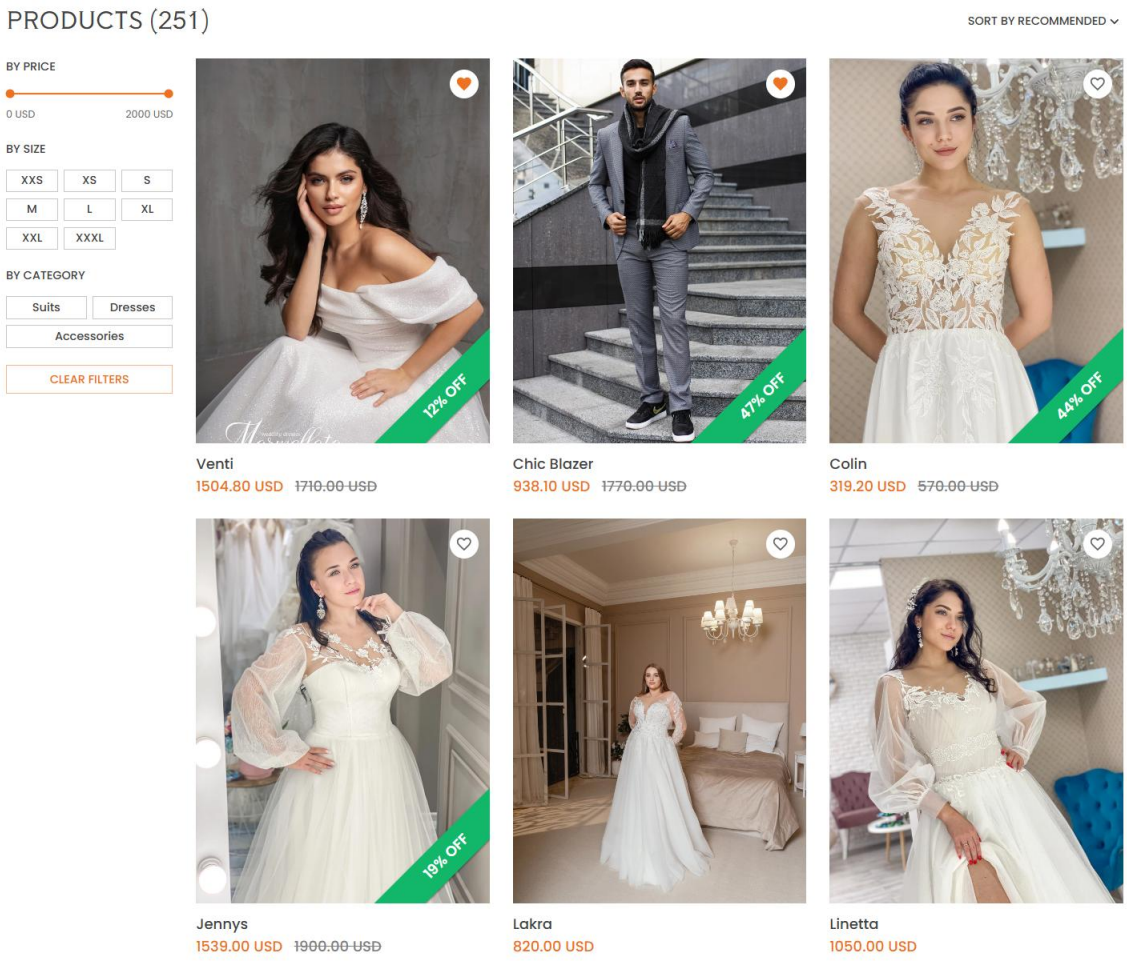

Рисунок 4.15 – Сторінка каталогу з фільтрами та сіткою товарів

Сторінка товару містить (рис. 4.16):

- галерею зображень із мініатюрами та відкриттям у lightbox;
- блок інформації — назва, ціна зі знижкою, опис та доступність;
- вибір розміру через SizesPicker;
- блок відгуків зі списком коментарів, рейтингом (1–5 зірок) та формою додавання відгуку (рис. 4.17).

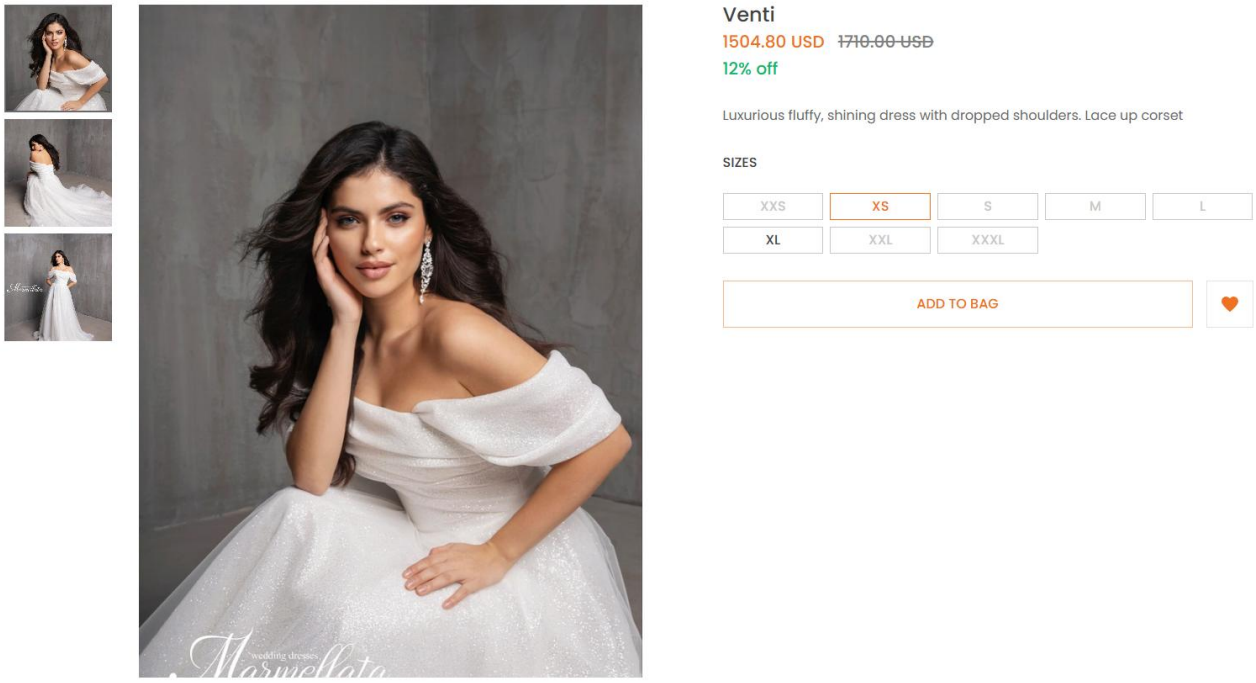

Рисунок 4.16 – Сторінка товару

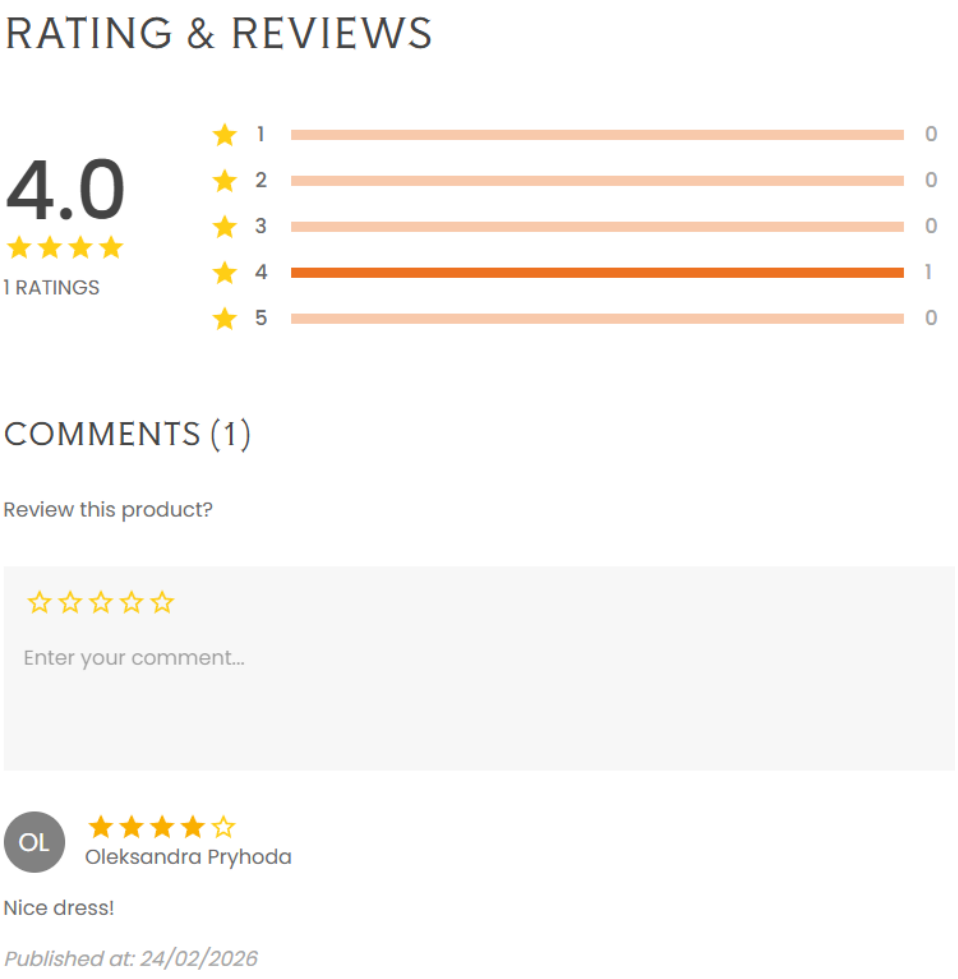

Рисунок 4.17 – Блок відгуків

Компонент Cart відображає список доданих товарів із можливістю зміни кількості, розміру та видалення (рис. 4.18). Загальну суму підраховує компонент Total. За порожнього кошика відображається Empty із посиланням на каталог (рис. 4.19).

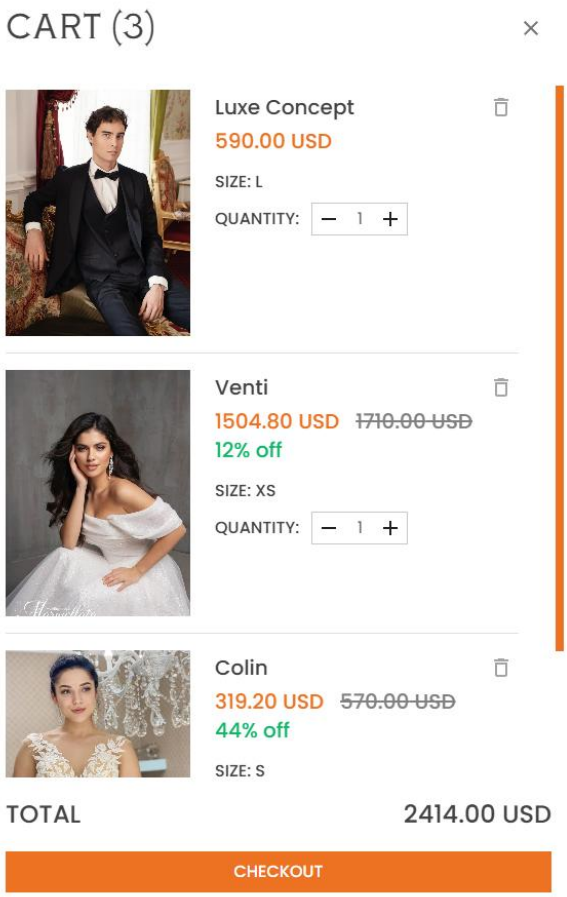

Рисунок 4.18 – Кошик з товарами

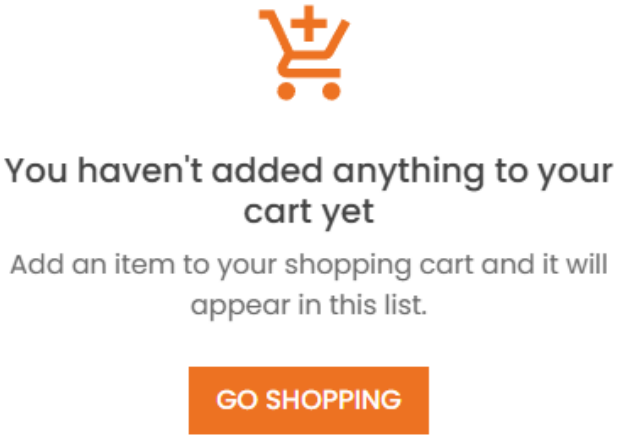

Рисунок 4.19 – Порожній кошик

Сторінка оформлення замовлення складається з трьох секцій: вибір збереженої адреси доставки або введення нової, вибір способу оплати та перегляд замовлених товарів із підсумком (рис. 4.20). Після успішного оформлення користувач перенаправляється на сторінку CheckoutSuccess (рис. 4.21).

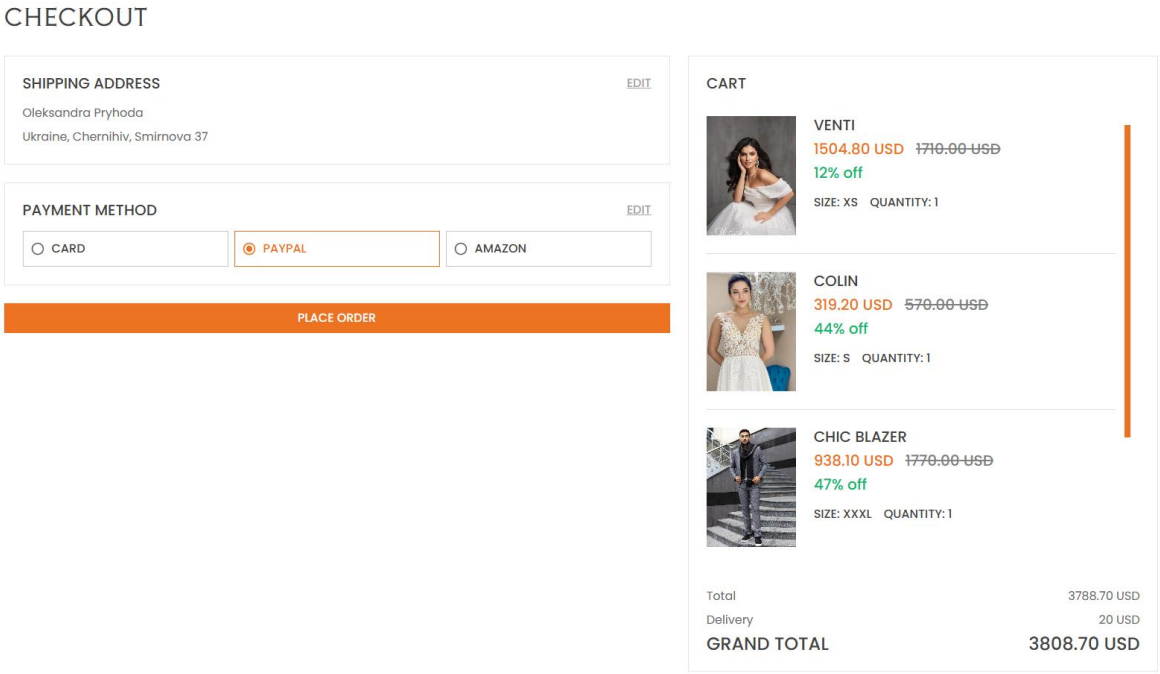

Рисунок 4.20 – Сторінка оформлення замовлення

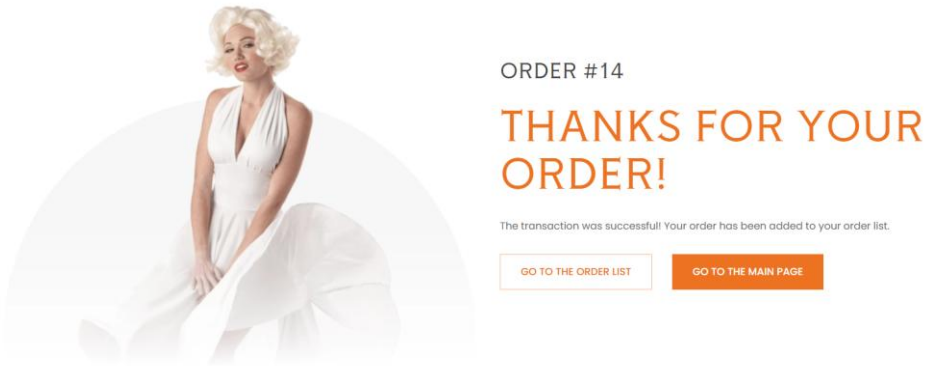

Рисунок 4.21 – Сторінка успішного оформлення замовлення

Форми реєстрації та логіну побудовано через React Hook Form із валідацією полів (email, пароль, ім'я) (рис. 4.22 – 4.23). Відновлення паролю складається з чотирьох кроків:

- введення email;
- введення OTP-коду з email (рис. 4.24);
- встановлення нового паролю;
- сторінка успіху.

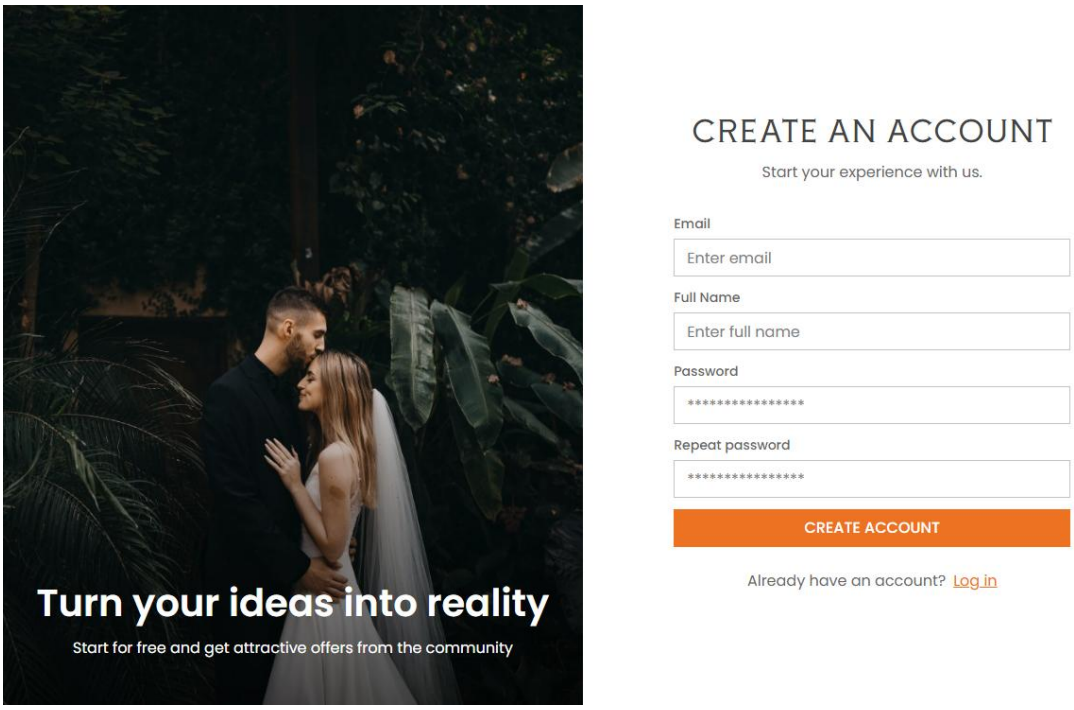

Рисунок 4.22 – Сторінка реєстрації

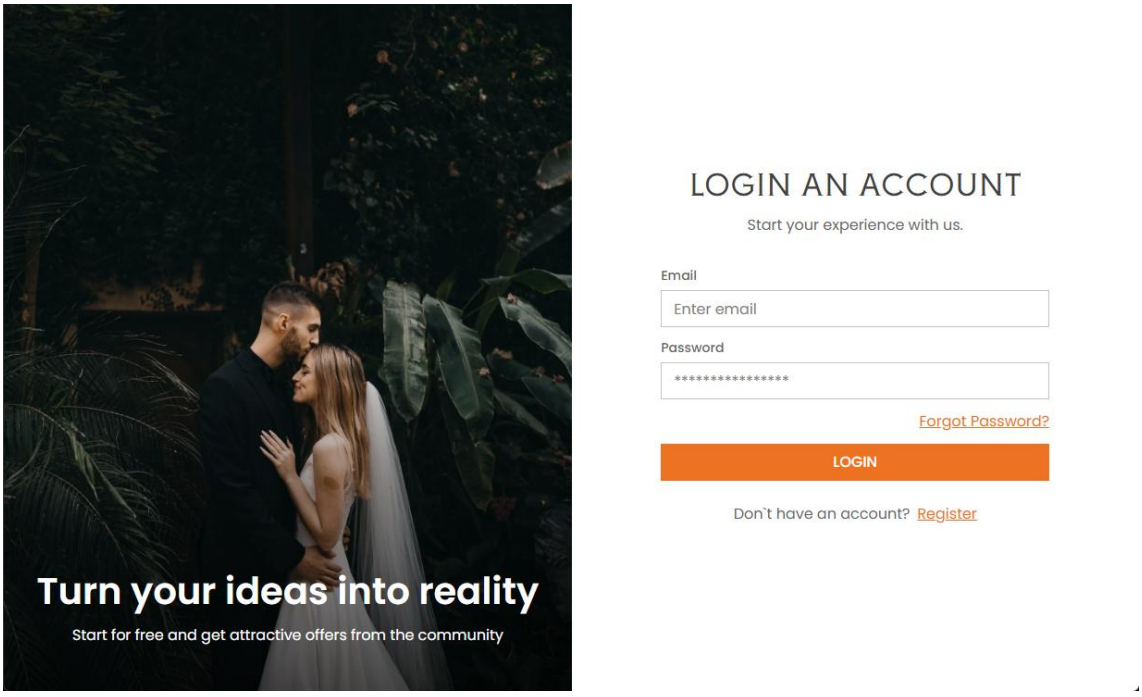

Рисунок 4.23 – Сторінка входу

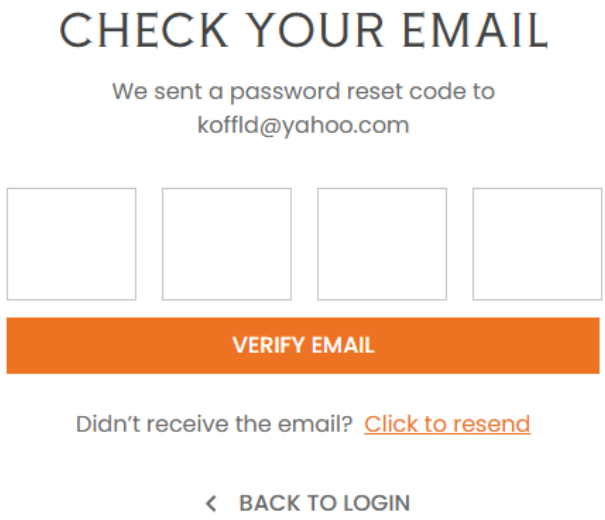

Рисунок 4.24 – Введення OTP-коду для відновлення паролю

Особистий кабінет побудовано на основі ProfileLayout із бічною навігацією та містить 7 розділів. Account відображає ім'я, email та дату реєстрації. Orders надає список замовлень із деталями (рис. 4.25), друком та завантаженням CSV. Shipping і Payment відповідають за управління адресами доставки (рис. 4.26) та способами оплати (рис. 4.27). Wishlist містить список бажань (рис. 4.28), Reviews — відгуки користувача, Settings — зміну паролю та управління активними сесіями. Для кожного розділу реалізовано скелетон-завантажувачі та компоненти порожніх станів.

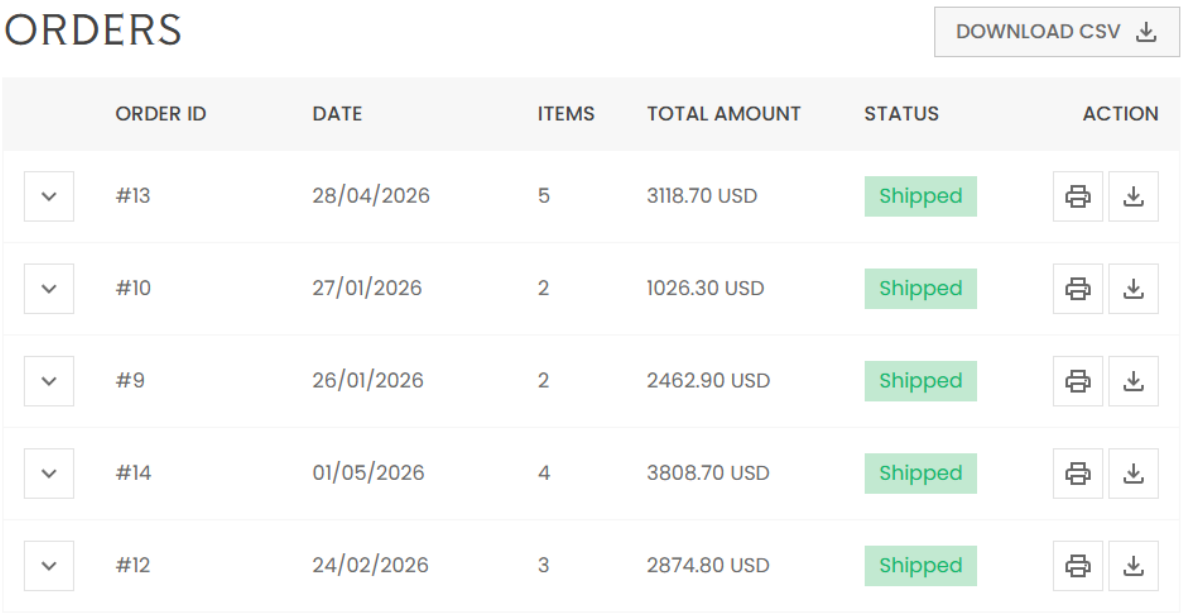

Рисунок 4.25 – Особистий кабінет (розділ замовлень)

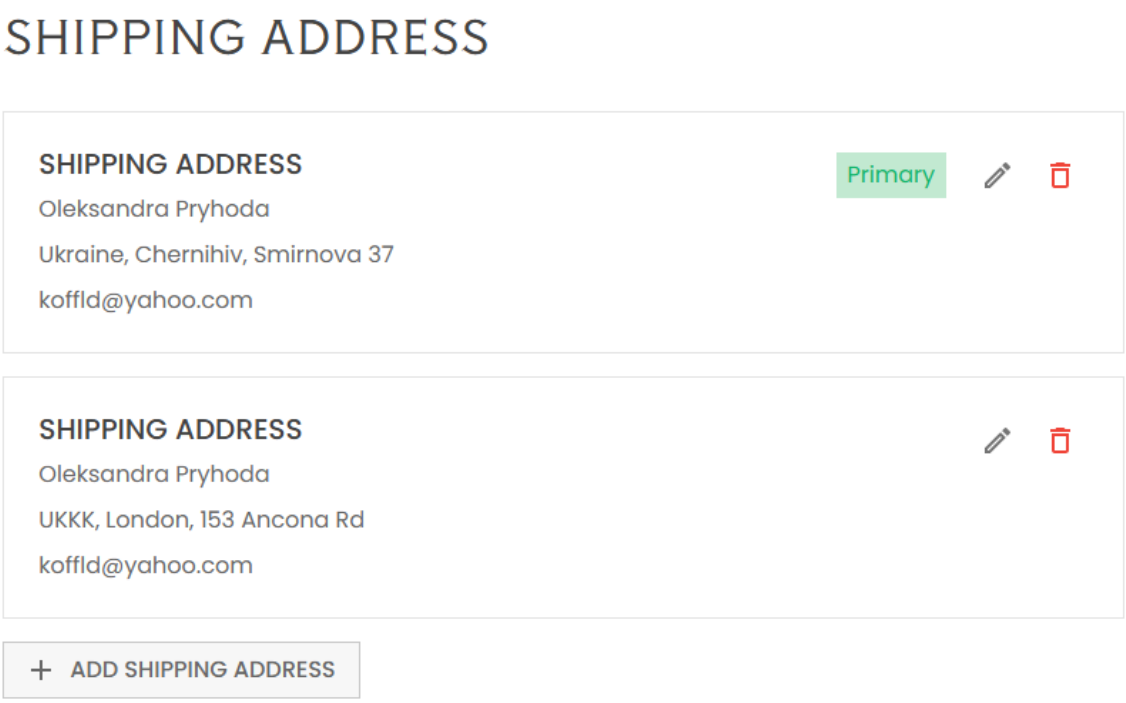

Рисунок 4.26 – Особистий кабінет (адреси доставки)

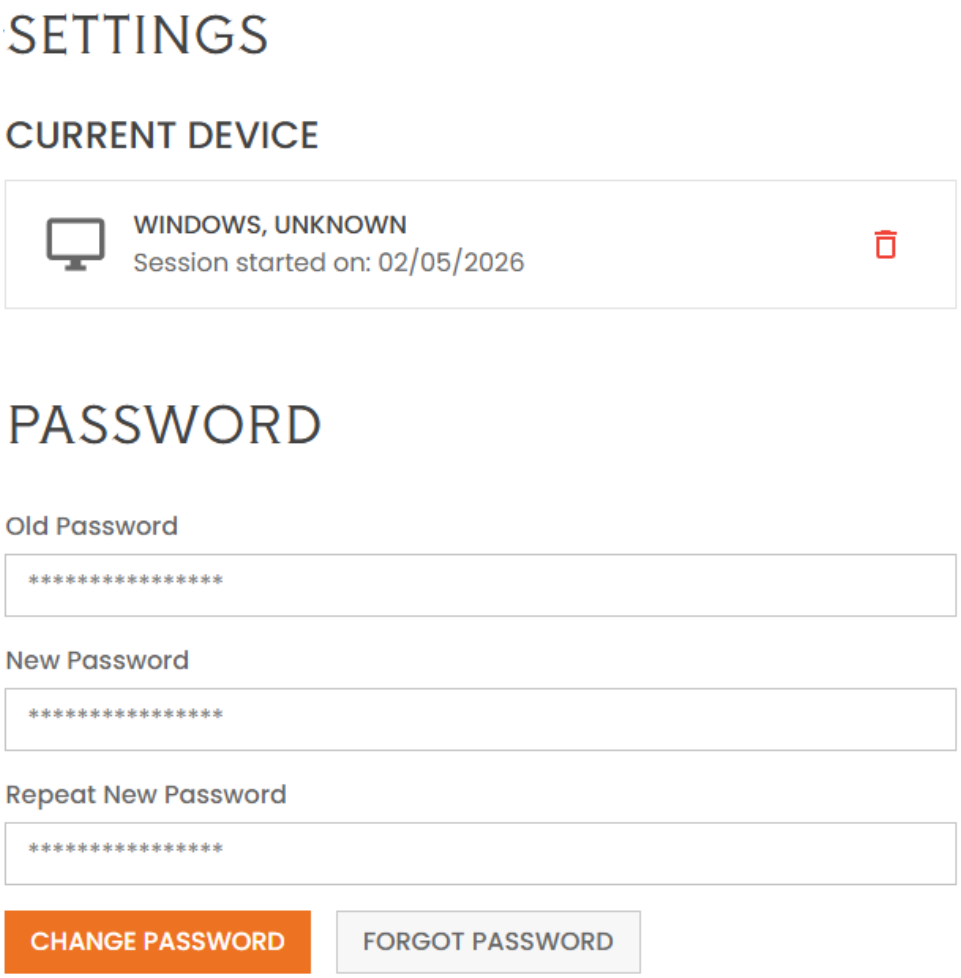

Рисунок 4.27 – Особистий кабінет (управління сесіями)

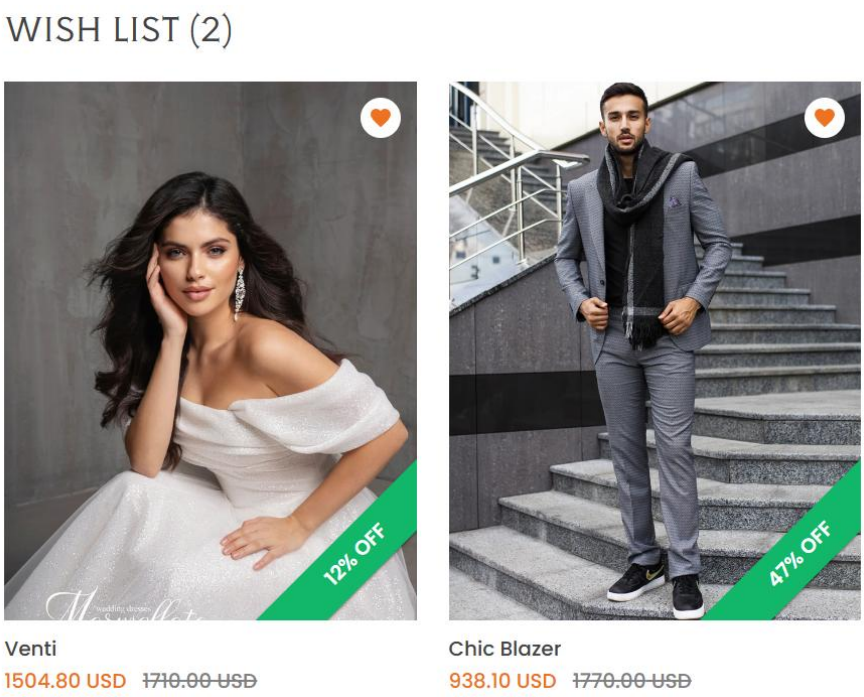

Рисунок 4.28 – Особистий кабінет (список бажань)

Для комунікації із сервером створено 10 API-модулів (auth, products, cart, orders, wishlist, reviews, payment, shipping, categories, user), кожен з яких використовує Axios. Кастомні хуки інкапсулюють виклики React Query та забезпечують автоматичне кешування, інвалідацію та повторні спроби.

4.5. Тестування адаптивності та аналіз швидкодії

Адаптивність забезпечується системою breakpoints Material UI: xs (0–600px), sm (600–900px), md (900–1200px), lg (1200–1536px), xl (1536px+). Адаптивні макети будуються через responsive-пропси компонентів Grid, Stack та Box — наприклад, сітка каталогу змінює кількість колонок від 1 на мобільному до 3–4 на десктопі. Тестування проведено в DevTools Chrome для трьох розмірів: iPhone SE (375×667px) (рис. 4.29), iPad (768×1024px) (рис. 4.30) та Desktop (1440×900px).

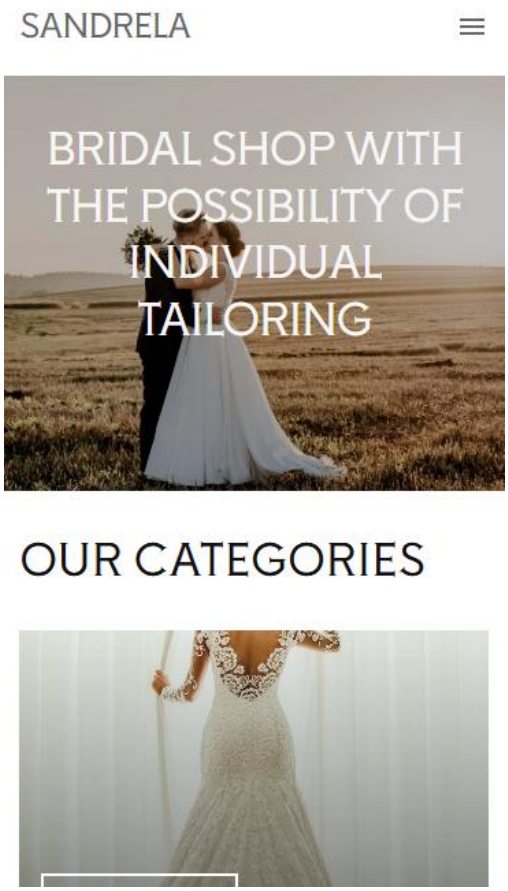

Рисунок 4.29 – Головна сторінка на мобільному пристрої

Рисунок 4.30 – Каталог на планшеті

Швидкодію перевірено через Google Lighthouse [22] для головної сторінки та каталогу. Оцінювались метрики Performance, FCP, LCP, CLS та TBT (рис. 4.31).

Оптимізація досягається завдяки:

- lazy loading сторінок та зображень;
- розділенню бандлу на чанки;
- кешуванню серверних даних через React Query;
- конвертації зображень у WebP через Sharp на сервері;
- debounce-обробці запитів при фільтрації.

Рисунок 4.31 – Результати Google Lighthouse для головної сторінки

4.6. Опис функціональних можливостей додатку

Веб-додаток надає такі функціональні можливості:

- перегляд із фільтрацією за категоріями, розмірами та ціною, сортування та текстовий пошук. Сторінка товару містить галерею з lightbox, опис, вибір розміру, ціну зі знижкою та блок відгуків (1–5 зірок);
- додавання товарів із вибором розміру і кількості, автоматичний підрахунок суми. Оформлення замовлення включає вибір адреси доставки, способу доставки (кур'єр, DHL, самовивіз) та методу оплати (картка, PayPal, Amazon);
- реєстрація, вхід, автооновлення токенів та 4-крокове відновлення паролю через OTP;

- особистий кабінет: інформація про акаунт, історія замовлень (друк, CSV), управління адресами та способами оплати, список бажань, відгуки та налаштування (зміна паролю, управління активними сесіями);
- список бажань через кнопку-серце на картці; відгуки з можливістю редагування та видалення; toast-сповіщення для всіх дій користувача (рис. 4.32); статичні інформаційні сторінки (рис. 4.33).

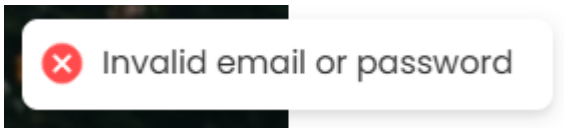

Рисунок 4.32 – Toast-сповіщення помилки при вході

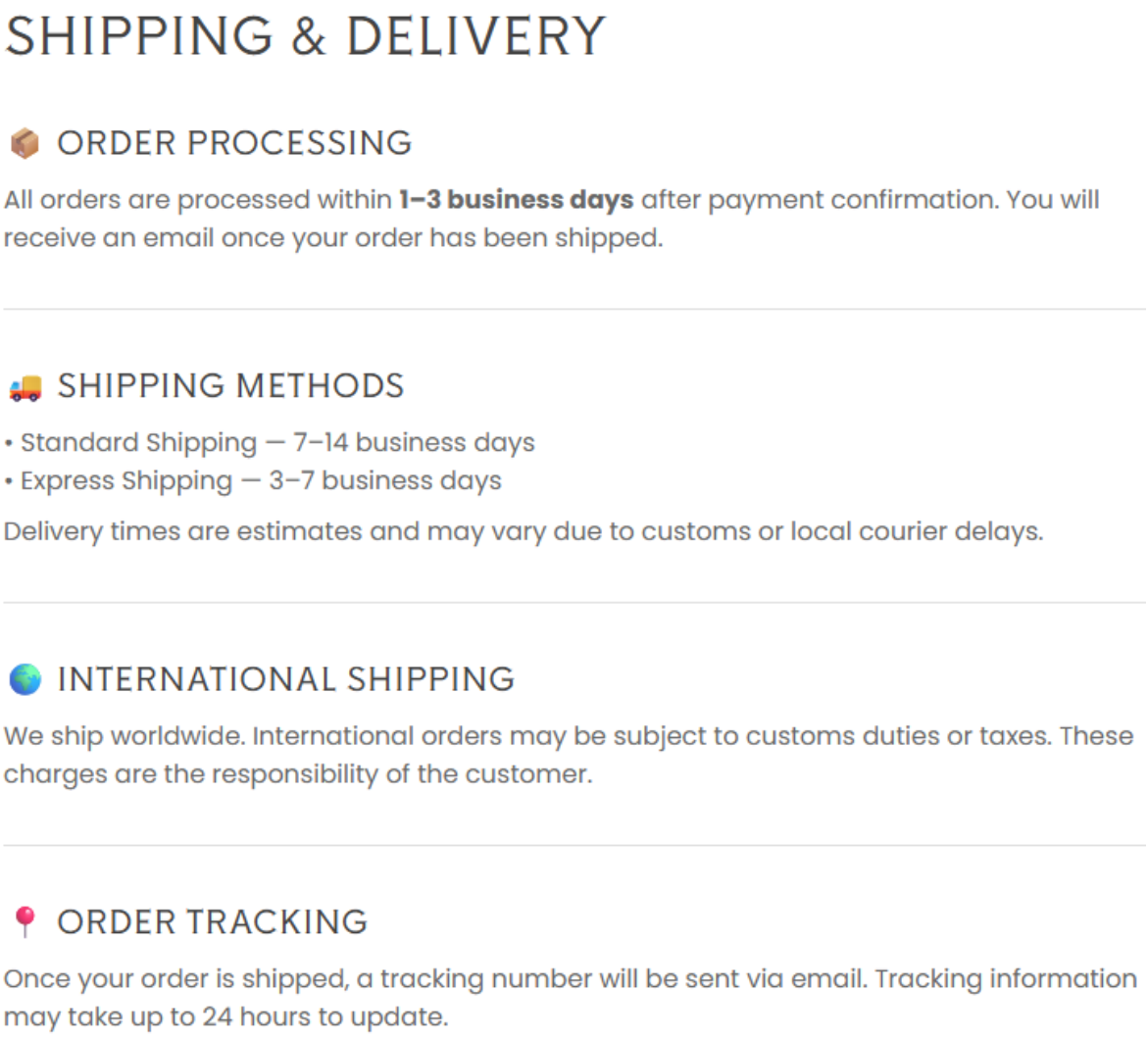

Рисунок 4.33 – Інформаційна сторінка (Доставка та логістика)

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи розроблено веб-додаток інтернет-магазину весільного одягу та аксесуарів, що забезпечує повний цикл взаємодії покупця з платформою — від перегляду каталогу до оформлення замовлення.

Клієнтську частину реалізовано на React із TypeScript, з використанням Material UI для побудови адаптивного інтерфейсу, Redux Toolkit для управління глобальним станом та TanStack React Query для кешування серверних даних. Серверну частину побудовано на NestJS із модульною архітектурою, ORM Prisma та СУБД PostgreSQL. Для зберігання зображень застосовано хмарний сервіс Cloudflare R2 з автоматичною конвертацією у формат WebP.

Реалізовано систему аутентифікації на основі JWT-токенів з OTP-підтвердженням для відновлення паролю, управління активними сесіями та комплекс заходів безпеки (Helmet, CORS, rate limiting, валідація вхідних даних). Забезпечено адаптивність інтерфейсу для мобільних, планшетних і десктопних пристроїв та оптимізацію продуктивності за метриками Google Lighthouse.

За результатами тестування підтверджено коректність роботи всіх функціональних модулів системи та відповідність реалізованого додатку визначеним вимогам.

Окрім технічної реалізації, результати кваліфікаційної роботи пройшли апробацію у вигляді доповіді та тез на конференції «XLVIX Міжнародна наукова студентська конференція».

СПИСОК ЛІТЕРАТУРИ

1. NestJS – Режим доступу: https://nestjs.com (дата звернення: 02.05.2026).

2. Redux Toolkit – Режим доступу: https://redux-toolkit.js.org (дата звернення: 02.05.2026).

3. TypeScript – Режим доступу: https://www.typescriptlang.org (дата звернення: 02.05.2026).

4. Vite – Режим доступу: https://vite.dev (дата звернення: 02.05.2026).

5. Material UI – Режим доступу: https://mui.com (дата звернення: 02.05.2026).

6. Prisma ORM – Режим доступу: https://www.prisma.io/docs (дата звернення: 02.05.2026).

7. PostgreSQL – Режим доступу: https://www.postgresql.org (дата звернення: 02.05.2026).

8. Хоффман Е. Безпека веб-застосунків: техніки атак та захисту. 2-е вид. Себастополь: O'Reilly Media, 2023. 426 с.

9. Node.js – Режим доступу: https://nodejs.org (дата звернення: 02.05.2026).

10. Visual Studio Code – Режим доступу: https://code.visualstudio.com (дата звернення: 02.05.2026).

11. Cloudflare R2 – Режим доступу: https://developers.cloudflare.com/r2 (дата звернення: 02.05.2026).

12. Черний Б. Програмування на TypeScript: масштабування JavaScript-застосунків. Себастополь: O'Reilly Media, 2019. 352 с.

13. Обе Р., Су Л. PostgreSQL: швидкий старт. 3-є вид. Себастополь: O'Reilly Media, 2017. 268 с.

14. Герон Д. Веб-розробка на Node.js. 5-е вид. Бірмінгем: Packt Publishing, 2020. 760 с.

15. Фланаган Д. JavaScript: вичерпний посібник. 7-е вид. Себастополь: O'Reilly Media, 2020. 706 с.

16. Sharp – Режим доступу: https://sharp.pixelplumbing.com (дата звернення: 02.05.2026).

17. React – Режим доступу: https://react.dev (дата звернення: 02.05.2026).

18. Вандеркам Д. Ефективний TypeScript: 83 способи покращити код. 2-е вид. Себастополь: O'Reilly Media, 2024. 368 с.

19. TanStack React Query – Режим доступу: https://tanstack.com/query (дата звернення: 02.05.2026).

20. React Router – Режим доступу: https://reactrouter.com (дата звернення: 02.05.2026).

21. Бенкс А., Порселло І. Вивчаємо React: сучасні патерни розробки React-застосунків. 2-е вид. Себастополь: O'Reilly Media, 2020. 310 с.

22. Google Lighthouse – Режим доступу: https://developer.chrome.com/docs/lighthouse (дата звернення: 02.05.2026).

23. Ольховська О. Методичні рекомендації до виконання кваліфікаційної роботи. Полтава: РВВ ПУЕТ, 2025. 58 с.

ДОДАТОК А. ПРОГРАМНИЙ КОД

ГОЛОВНА СТОРІНКА

```
import { Box, Grid, Stack } from "@mui/material";
import { useRef } from "react";
import HomepageSection from "@/components/sections/HomepageSection";
import HomepageBanner from "@/components/ui/layout/HomepageBanner";
import ImageWithButton from "@/components/ui/cards/ImageWithButton";
import ProductsSwiper from "@/components/features/product/Swiper";
import HomepageCompanies from "@/components/ui/layout/HomepageCompanies";
import { useProducts } from "@/hooks/useProducts";
import type { Swiper } from "swiper/types";
const Home: React.FC = () => {
  const dressesRef = useRef<Swiper | null>(null);
  const suitsRef = useRef<Swiper | null>(null);
  const accessoriesRef = useRef<Swiper | null>(null);
  const { products: dresses, isLoading: dressesLoading } = useProducts({ category: "dresses" });
  const { products: suits, isLoading: suitsLoading } = useProducts({ category: "suits" });
   const { products: accessories, isLoading: accessoriesLoading } = useProducts({ category:
"accessories" });
  const swiperSections = [
    {
      key: "dresses",
      title: "Dresses",
      data: dresses,
      isLoading: dressesLoading,
      ref: dressesRef,
    },
    {
      key: "suits",
      title: "Suits",
      data: suits,
      isLoading: suitsLoading,
      ref: suitsRef,
    },
    {
      key: "accesories",
      title: "Accessories",
      data: accessories,
      isLoading: accessoriesLoading,
      ref: accessoriesRef,
    },
  ] as const;
  return (
    <Stack>
      <HomepageBanner />
      <HomepageSection title="Our categories">
        <Box sx={{ height: 750 }}>
          <Grid container spacing={4} sx={{ height: "100%" }}>
```

```
          <Grid size={{ xs: 6 }} sx={{ height: "100%" }}>
            <ImageWithButton
              imgSrc="/categories-dresses.webp"
              linkText="Dresses"
              linkHref="/catalog?category=dresses"
            />
          </Grid>
          <Grid size={{ xs: 6 }} sx={{ height: "100%" }}>
            <Box
              sx={{
                display: "grid",
                gridTemplateRows: "1fr 1fr",
                gap: 4,
                height: "100%",
              }}
            >
                              <ImageWithButton imgSrc="/categories-suits.webp" linkText="Suits"
linkHref="/catalog?category=suits" />
              <ImageWithButton
                imgSrc="/categories-accessories.webp"
                linkText="Accessories"
                linkHref="/catalog?category=accessories"
              />
            </Box>
          </Grid>
        </Grid>
      </Box>
    </HomepageSection>
    <HomepageSection title="Clothing">
      <Box sx={{ height: 450 }}>
        <Grid container spacing={4} sx={{ height: "100%" }}>
          <Grid size={{ xs: 6 }} sx={{ height: "100%" }}>
            <ImageWithButton
              imgSrc="/clothing-for-her.webp"
              linkText="For her"
              linkHref="/catalog?category=dresses"
            />
          </Grid>
          <Grid size={{ xs: 6 }} sx={{ height: "100%" }}>
                          <ImageWithButton imgSrc="/clothing-for-him.webp" linkText="For him"
linkHref="/catalog?category=suits" />
          </Grid>
        </Grid>
      </Box>
    </HomepageSection>
    {swiperSections.map(({ key, title, data, isLoading, ref }) => (
      <HomepageSection key={key} title={title} isSwiper swiperRef={ref}>
        <ProductsSwiper data={data} swiperRef={ref} isLoading={isLoading} />
      </HomepageSection>
    ))}
    <HomepageCompanies />
  </Stack>
```

```
  );
};
export default Home;
```

СТОРІНКА РЕЄСТРАЦІЇ

```
import RegisterForm from "@/components/forms/auth/Register";
const Register: React.FC = () => {
  return <RegisterForm />;
};
export default Register;
```

ФОРМА РЕЄСТРАЦІЇ

```
import { Button, Stack, TextField } from "@mui/material";
import { useForm } from "react-hook-form";
import { useNavigate } from "react-router";
import toast from "react-hot-toast";
import AuthFormLayout from "@/components/forms/auth/FormLayout";
import FormField from "@/components/ui/layout/FormField";
import AuthService from "@/api/auth/auth.service";
type RegisterFormFields = {
  email: string;
  fullName: string;
  password: string;
  repeatPassword: string;
};
const RegisterForm: React.FC = () => {
  const navigate = useNavigate();
  const {
    handleSubmit,
    register,
    watch,
    formState: { errors, isSubmitting },
  } = useForm<RegisterFormFields>();
  const password = watch("password");
  const onSubmit = async (formData: RegisterFormFields) => {
    const { repeatPassword, fullName, ...data } = formData;
    void repeatPassword;
    const [firstName, lastName = ""] = fullName.trim().split(" ");
    await AuthService.register({ ...data, firstName, lastName });
    toast.success("Registration successful");
    navigate("/login");
  };

  return (
    <AuthFormLayout
      title="Create an account"
      description="Start your experience with us."
      footerText="Already have an account?"
      footerLinkText="Log in"
      footerLinkHref="/login"
      isBackToLogin={false}
    >
```

```
    <Stack component="form" noValidate onSubmit={handleSubmit(onSubmit)} sx={{ gap: 1.5 }}>
      <FormField label="Email">
        <TextField
          placeholder="Enter email"
          {...register("email", {
            required: "Email is required",
            pattern: {
              value: /^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i,
              message: "Invalid email address",
            },
          })}
          error={!!errors.email}
          helperText={errors.email?.message}
        />
      </FormField>
      <FormField label="Full Name">
        <TextField
          placeholder="Enter full name"
          {...register("fullName", {
            required: "Full name is required",
            validate: (value) => {
              const parts = value.trim().split(/\s+/);
              if (parts.length < 2) return "Please enter both first and last name";
              if (parts.some((p) => p.length < 2)) return "Each name must be at least 2 characters";
                if (parts.some((p) => p.length > 32)) return "Each name must be no more than 32 characters";
              return true;
            },
            pattern: {
              value: /^[A-Za-zA-Яа-яЁёIiЇїЄє\s'-]+$/,
              message: "Full name can only contain letters, spaces, hyphens, and apostrophes",
            },
          })}
          error={!!errors.fullName}
          helperText={errors.fullName?.message}
        />
      </FormField>
      <FormField label="Password">
        <TextField
          placeholder="**********"
          type="password"
          {...register("password", {
            required: "Password is required",
            minLength: { value: 8, message: "Password is too short (minimum 8 characters)" },
          })}
          error={!!errors.password}
          helperText={errors.password?.message}
        />
      </FormField>
      <FormField label="Repeat password">
        <TextField
```

```
          placeholder="****************"
          type="password"
          {...register("repeatPassword", {
            required: "Please confirm your password",
            validate: (value) => value === password || "Passwords do not match",
          })}
          error={!!errors.repeatPassword}
          helperText={errors.repeatPassword?.message}
        />
      </FormField>
      <Button type="submit" variant="contained" size="small" disabled={isSubmitting}>
        Create Account
      </Button>
    </Stack>
  </AuthFormLayout>
 );
};
export default RegisterForm;
```

СТОРІНКА ВХОДУ

```
import LoginForm from "@/components/forms/auth/Login";
const Login: React.FC = () => {
 return <LoginForm />;
};
export default Login;
```

ФОРМА ВХОДУ

```
import { Button, Link, Stack, TextField } from "@mui/material";
import { useAppDispatch } from "@/redux/store";
import { useForm } from "react-hook-form";
import { useNavigate } from "react-router";
import AuthFormLayout from "@/components/forms/auth/FormLayout";
import FormField from "@/components/ui/layout/FormField";
import { login } from "@/redux/auth/auth.actions";
type LoginFormFields = {
 email: string;
 password: string;
};
const LoginForm: React.FC = () => {
 const dispatch = useAppDispatch();
 const navigate = useNavigate();
 const {
   handleSubmit,
   register,
   formState: { errors, isSubmitting },
 } = useForm<LoginFormFields>();
 const onSubmit = async (data: LoginFormFields) => {
   const result = await dispatch(login(data));
   if (login.rejected.match(result)) return;
   navigate("/");
 };
 return (
```

```
   <AuthFormLayout
    title="Login an account"
    description="Start your experience with us."
    footerText="Don`t have an account?"
    footerLinkText="Register"
    footerLinkHref="/register"
    isBackToLogin={false}
   >
    <Stack component="form" noValidate onSubmit={handleSubmit(onSubmit)} sx={{ gap: 1.5
}}>
     <FormField label="Email">
      <TextField
       placeholder="Enter email"
       {...register("email", {
        required: "Email is required",
        pattern: {
         value: /^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i,
         message: "Invalid email address",
        },
       })}
       error={!!errors.email}
       helperText={errors.email?.message}
      />
     </FormField>
     <FormField label="Password">
      <TextField
       placeholder="****************"
       type="password"
       {...register("password", {
        required: "Password is required",
        minLength: { value: 8, message: "Password is too short (minimum 8 characters)" },
       })}
       error={!!errors.password}
       helperText={errors.password?.message}
      />
     </FormField>
     <Link href="/forgot-password" variant="underlined" sx={{ alignSelf: "flex-end" }}>
      Forgot Password?
     </Link>
     <Button type="submit" variant="contained" size="small" disabled={isSubmitting}>
      Login
     </Button>
    </Stack>
   </AuthFormLayout>
  );
};
export default LoginForm;
```

СТОРІНКА ЗАБУЛИ ПАРОЛЬ

```
import ForgotPasswordForm from "@/components/forms/auth/ForgotPassword";
const ForgotPassword = () => {
  return <ForgotPasswordForm />;
```

```
};
export default ForgotPassword;
```

ФОРМА ЗАБУЛИ ПАРОЛЬ

```
import { Button, Stack, TextField } from "@mui/material";
import { useForm } from "react-hook-form";
import { useNavigate } from "react-router";
import toast from "react-hot-toast";
import AuthFormLayout from "@/components/forms/auth/FormLayout";
import FormField from "@/components/ui/layout/FormField";
import AuthService from "@/api/auth/auth.service";
type ForgotPasswordFormFields = {
  email: string;
};
const ForgotPasswordForm: React.FC = () => {
  const navigate = useNavigate();
  const {
    handleSubmit,
    register,
    formState: { errors, isSubmitting },
  } = useForm<ForgotPasswordFormFields>();
  const onSubmit = async ({ email }: ForgotPasswordFormFields) => {
    const result = await AuthService.sendForgotPasswordOtp(email);
    toast.success(result.message);
    navigate("/verify-otp", { state: { email } });
  };
  return (
     <AuthFormLayout title="Forgot Password?" description="No worries, we'll send you reset instructions">
       <Stack component="form" noValidate onSubmit={handleSubmit(onSubmit)} sx={{ gap: 1.5 }}>
         <FormField label="Email">
           <TextField
             placeholder="Enter email"
             {...register("email", {
               required: "Email is required",
               pattern: {
                 value: /^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i,
                 message: "Invalid email address",
               },
             })}
             error={!!errors.email}
             helperText={errors.email?.message}
           />
         </FormField>
         <Button type="submit" variant="contained" size="small" disabled={isSubmitting}>
           Send OTP
         </Button>
       </Stack>
     </AuthFormLayout>
  );
};
```

```
export default ForgotPasswordForm;
```

СТОРІНКА ВІДНОВЛЕННЯ ПАРОЛЮ

```
import ResetPasswordForm from "@/components/forms/auth/ResetPassword";
const ResetPassword: React.FC = () => {
  return <ResetPasswordForm />;
};
export default ResetPassword;
```

ФОРМА ВІДНОВЛЕННЯ ПАРОЛЮ

```
import { Button, Stack, TextField } from "@mui/material";
import { useForm } from "react-hook-form";
import { useNavigate } from "react-router";
import AuthFormLayout from "@/components/forms/auth/FormLayout";
import FormField from "@/components/ui/layout/FormField";
import AuthService from "@/api/auth/auth.service";
type ResetPasswordFormFields = {
  password: string;
  repeatPassword: string;
};
const ResetPasswordForm: React.FC = () => {
  const navigate = useNavigate();
  const params = new URLSearchParams(location.search);
  const resetToken = params.get("resetToken");
  const {
    handleSubmit,
    register,
    watch,
    formState: { errors, isSubmitting },
  } = useForm<ResetPasswordFormFields>();
  const password = watch("password");
  const onSubmit = async (formData: ResetPasswordFormFields) => {
    if (!resetToken) return navigate("/login");
    await AuthService.resetPassword({ resetToken, password: formData.password });
    navigate("/reset-password-success");
  };
  return (
    <AuthFormLayout
      title="Set new password"
      description="Your new password must be different to previously used passwords."
    >
      <Stack component="form" noValidate onSubmit={handleSubmit(onSubmit)} sx={{ gap: 1.5
}}>
        <FormField label="Password">
          <TextField
            placeholder="**********"
            type="password"
            {...register("password", {
              required: "Password is required",
              minLength: { value: 8, message: "Password is too short (minimum 8 characters)" },
            })}
            error={!!errors.password}
```

```
          helperText={errors.password?.message}
         />
        </FormField>
        <FormField label="Repeat password">
         <TextField
          placeholder="****************"
          type="password"
          {...register("repeatPassword", {
           required: "Please confirm your password",
           validate: (value) => value === password || "Passwords do not match",
          })}
          error={!!errors.repeatPassword}
          helperText={errors.repeatPassword?.message}
         />
        </FormField>
        <Button type="submit" variant="contained" size="small" disabled={isSubmitting}>
         Reset password
        </Button>
      </Stack>
    </AuthFormLayout>
  );
};
export default ResetPasswordForm;
```

СТОРІНКА ПІДТВЕРДЖЕННЯ ОДНОРАЗОВОГО КОДУ

```
import VerifyOtpForm from "@/components/forms/auth/VerifyOtp";
const VerifyOtp: React.FC = () => {
  return <VerifyOtpForm />;
};
export default VerifyOtp;
```

ФОРМА ПІДТВЕРДЖЕННЯ ОДНОРАЗОВОГО КОДУ

```
import { Button, Stack, Typography } from "@mui/material";
import { useLocation, useNavigate } from "react-router";
import { useEffect } from "react";
import { useForm, Controller } from "react-hook-form";
import { MuiOtpInput } from "mui-one-time-password-input";
import AuthFormLayout from "@/components/forms/auth/FormLayout";
import AuthService from "@/api/auth/auth.service";
import { VerifyOtpBody } from "@/api/auth/auth.types";
const VerifyOtpForm: React.FC = () => {
  const navigate = useNavigate();
  const location = useLocation();
  const email = location.state?.email;
  const {
    control,
    handleSubmit,
    formState: { errors, isSubmitting },
  } = useForm<VerifyOtpBody>({
    defaultValues: {
      otp: "",
    },
```

```
  });
  useEffect(() => {
    if (!email) {
      navigate("/login");
    }
  }, [email, navigate]);
  const onSubmit = async ({ otp }: { otp: string }) => {
    const { resetToken } = await AuthService.verifyOtp({ email, otp });
    navigate("/verify-otp-success", { state: { resetToken } });
  };
  return (
    <AuthFormLayout
      title="Check your email"
      description={`We sent a password reset code to ${email}`}
      footerText="Didn’t receive the email?"
      footerLinkText="Click to resend"
      footerLinkHref="/forgot-password"
    >
      <Stack component="form" noValidate onSubmit={handleSubmit(onSubmit)} sx={{ gap: 1.5
}}>
        <Controller
          name="otp"
          control={control}
          rules={{
            required: "OTP is required",
            minLength: {
              value: 4,
              message: "OTP must be 4 digits",
            },
            maxLength: {
              value: 4,
              message: "OTP must be 4 digits",
            },
          }}
          render={({ field }) => (
            <MuiOtpInput
              {...field}
              length={4}
              TextFieldsProps={{
                InputProps: {
                  sx: (theme) => ({
                    width: 92,
                    height: 80,
                    fontSize: 50,
                    fontWeight: 700,
                    textAlign: "center",
                    "& input": {
                      color: theme.palette.primary.main,
                    },
                  }),
                },
              }}
```

```
          />
        )}
      />
      {errors.otp && (
        <Typography color="error" variant="body2" sx={{ textAlign: "center" }}>
          {errors.otp.message}
        </Typography>
      )}
      <Button type="submit" variant="contained" size="small" disabled={isSubmitting}>
        {isSubmitting ? "Verifying..." : "Verify Email"}
      </Button>
    </Stack>
  </AuthFormLayout>
 );
};
export default VerifyOtpForm;
```

СТОРІНКА ПРОФІЛЮ (ОСНОВНА ІНФОРМАЦІЯ)

```
import { Stack, Typography } from '@mui/material';
import UpdateUserForm from '@/components/forms/profile/UpdateUser';
const AccountPage: React.FC = () => {
 return (
  <Stack gap={2}>
   <Typography variant="h3">Profile Information</Typography>
   <UpdateUserForm />
  </Stack>
 );
};
export default AccountPage;
```

ФОРМА ДЛЯ ЗМІНИ ОСНОВНОЇ ІНФОРМАЦІЇ

```
import { Button, Stack, TextField } from '@mui/material';
import { useQueryClient } from '@tanstack/react-query';
import { useForm } from 'react-hook-form';
import toast from 'react-hot-toast';
import FormField from '@/components/ui/layout/FormField';
import UserService from '@/api/user/user.service';
import { useUser } from '@/hooks/useUser';
import { User } from '@/api/user/user.types';
type UpdateUserFormFields = {
 firstName: string;
 lastName: string;
};
const UpdateUserForm: React.FC = () => {
 const queryClient = useQueryClient();
 const { data: user } = useUser();
 const {
  handleSubmit,
  register,
  formState: { errors, isSubmitting },
 } = useForm<UpdateUserFormFields>({
  defaultValues: {
```

```
    firstName: user?.firstName || '',
    lastName: user?.lastName || '',
   },
  });
  const onSubmit = async (data: UpdateUserFormFields) => {
   try {
    const updatedUser = await UserService.updateUser(data);
    queryClient.setQueryData<User>(['user'], updatedUser);
    toast.success('Profile updated successfully');
   } catch {
    toast.error('Failed to update profile');
   }
  };
  return (
    <Stack component="form" noValidate onSubmit={handleSubmit(onSubmit)} sx={{ gap: 2,
maxWidth: 600, width: '100%' }}>
    <FormField label="Email" labelFontSize={16}>
          <TextField placeholder="Enter email" slotProps={{ input: { readOnly: true } }}
value={user?.email} />
    </FormField>
    <FormField label="First Name" labelFontSize={16}>
     <TextField
      placeholder="Enter your first name"
      {...register('firstName', {
       required: 'First name is required',
       minLength: { value: 2, message: 'First name is too short (minimum 2 characters)' },
       maxLength: { value: 50, message: 'First name is too long (maximum 50 characters)' },
       pattern: {
        value: /^[A-Za-zA-Яа-я\s'-]+$/,
        message: 'First name can only contain letters, spaces, hyphens, and apostrophes',
       },
      })}
      error={!!errors.firstName}
      helperText={errors.firstName?.message}
     />
    </FormField>
    <FormField label="Last Name" labelFontSize={16}>
     <TextField
      placeholder="Enter your last name"
      {...register('lastName', {
       required: 'Last name is required',
       minLength: { value: 2, message: 'Last name is too short (minimum 2 characters)' },
       maxLength: { value: 50, message: 'Last name is too long (maximum 50 characters)' },
       pattern: {
        value: /^[A-Za-zA-Яа-я\s'-]+$/,
        message: 'Last name can only contain letters, spaces, hyphens, and apostrophes',
       },
      })}
      error={!!errors.lastName}
      helperText={errors.lastName?.message}
     />
    </FormField>
```

```
      <Button type="submit" variant="contained" size="small" disabled={isSubmitting} sx={{
maxWidth: 'max-content' }}>
       Save Profile
     </Button>
   </Stack>
  );
};
export default UpdateUserForm;
```

СТОРІНКА ЗАМОВЛЕНЬ

```
import { Button, Stack, Typography } from '@mui/material';
import Orders from '@/components/features/profile/Orders';
import OrdersService from '@/api/orders/orders.service';
import { useFileDownload } from '@/hooks/useFileDownload';
import { useOrders } from '@/hooks/useOrders';
import { FileDownloadOutlined } from '@mui/icons-material';
const OrdersPage: React.FC = () => {
  const download = useFileDownload();
  const { data: orders } = useOrders();
  const exportAllToCsv = async () => {
    const data = await OrdersService.getAllCsv();
    download(data, `orders.csv`, 'text/csv;charset=utf-8;');
  };
  return (
    <Stack gap={2}>
      <Stack flexDirection="row" alignItems="center" justifyContent="space-between" gap={4}>
        <Typography variant="h3">Orders</Typography>
        <Button
          variant="outlined"
          color="grey"
          size="small"
          endIcon={<FileDownloadOutlined />}
          onClick={exportAllToCsv}
          disabled={orders?.total === 0}
        >
          Download CSV
        </Button>
      </Stack>
      <Orders />
    </Stack>
  );
};
export default OrdersPage;
```

КОМПОНЕНТ СПИСКУ ЗАМОВЛЕНЬ

```
import { useState } from 'react';
import { Pagination, Stack, Table, TableBody, TableCell, TableCellProps, TableHead, TableRow
} from '@mui/material';
import Order from './Item';
import EmptyState from '@/components/ui/EmptyState';
import OrdersSkeleton from '@/components/ui/loaders/skeletons/Orders';
```

```
import { useOrders } from '@/hooks/useOrders';
import { PAGE_LIMIT } from '@/constants';
import { RemoveShoppingCartOutlined } from '@mui/icons-material';
const Orders: React.FC = () => {
  const [page, setPage] = useState<number>(1);
  const { data: orders, isLoading } = useOrders(page);
  const pages = orders?.total ? Math.ceil(orders.total / PAGE_LIMIT) : 0;
  const columns = [
    { sx: { width: 40 }, title: '' },
    { sx: { minWidth: 140 }, title: 'Order ID' },
    { sx: { minWidth: 160 }, title: 'Date' },
    { sx: { minWidth: 50 }, title: 'Items' },
    { sx: { minWidth: 60 }, title: 'Total Amount' },
    { sx: { minWidth: 60 }, title: 'Status' },
    { sx: { minWidth: 90 }, title: 'Action', align: 'right' },
  ];
  return (
    <Stack gap={3}>
      {isLoading ? (
        <OrdersSkeleton />
      ) : orders && orders?.total > 0 ? (
        <>
          <Table>
            <TableHead>
              <TableRow>
                {columns.map(({ sx, title, align }) => (
                  <TableCell sx={sx} align={(align as TableCellProps['align']) ?? 'left'}>
                    {title}
                  </TableCell>
                ))}
              </TableRow>
            </TableHead>
            <TableBody>
              {orders?.orders.map((order) => (
                <Order {...order} />
              ))}
            </TableBody>
          </Table>
              {pages > 1 && <Pagination page={page} count={pages} onChange={(_, val) =>
setPage(val)} />}
        </>
      ) : (
        <EmptyState
          title="Your orders is empty"
          description="Your orders will appear here once you make a purchase."
          icon={RemoveShoppingCartOutlined}
        />
      )}
    </Stack>
  );
};
export default Orders;
```

СТОРІНКА БАЖАНІ ТОВАРИ

```
import { useState } from 'react';
import { useSelector } from 'react-redux';
import { Divider, Pagination, Stack, Typography } from '@mui/material';
import { useQuery } from '@tanstack/react-query';
import ReviewsItem from '@/components/features/product/Reviews/Item';
import ReviewSkeleton from '@/components/ui/loaders/skeletons/Review';
import EmptyState from '@/components/ui/EmptyState';
import ReviewsService from '@/api/reviews/reviews.service';
import { authSelector } from '@/redux/auth/auth.selectors';
import { StarHalf } from '@mui/icons-material';
import { REVIEWS_LIMIT } from '@/constants';
const ReviewsPage: React.FC = () => {
  const { isAuth } = useSelector(authSelector);
  const [page, setPage] = useState<number>(1);
  const { data: reviews = { reviews: [], total: 0 }, isLoading } = useQuery({
    queryKey: ['reviews', { page, limit: REVIEWS_LIMIT }],
    queryFn: () => ReviewsService.getMyReviews({ page, limit: REVIEWS_LIMIT }),
    enabled: isAuth,
  });
  const pages = Math.ceil(reviews.total / REVIEWS_LIMIT);
  return (
    <Stack gap={4} sx={{ width: '100%' }}>
      <Typography variant="h3">Reviews ({reviews.total})</Typography>
      {isLoading ? (
        [...Array(3)].map((_, idx) => <ReviewSkeleton key={idx} />)
      ) : reviews.total > 0 ? (
        reviews.reviews.map((review, idx) => (
          <>
            <ReviewsItem key={review.id} variant="profile" {...review} />
            {idx + 1 !== reviews.reviews.length && <Divider />}
          </>
        ))
      ) : (
        <EmptyState
          icon={StarHalf}
          title="You haven't left any reviews yet"
            description="Reviews you write will appear here. Share your experience with other
shoppers once you've tried our products."
        />
      )}
      {pages > 1 && (
        <>
          <Divider />
          <Pagination page={page} count={pages} onChange={(_, p) => setPage(p)} />
        </>
      )}
    </Stack>
  );
};
export default ReviewsPage;
```

СТОРІНКА ПЛАТІЖНОЇ ІНФОРМАЦІЇ

```
import { Stack, Typography } from "@mui/material";
import PaymentMethods from "@/components/features/profile/PaymentMethods";
const PaymentPage: React.FC = () => {
 return (
  <Stack gap={4} sx={{ width: "100%" }}>
   <Typography variant="h3">Payment methods</Typography>
   <PaymentMethods />
  </Stack>
 );
};
export default PaymentPage;
```

КОМПОНЕНТ СПИСКУ ПЛАТІЖНИХ МЕТОДІВ

```
import { Button, Stack } from '@mui/material';
import { useState } from 'react';
import PaymentMethod from './Item';
import EmptyState from '../EmptyState';
import CustomModal from '@/components/ui/layout/CustomModal';
import PaymentMethodForm from '@/components/forms/profile/PaymentMethod';
import PaymentMethodSkeleton from '@/components/ui/loaders/skeletons/PaymentMethod';
import { useUserPaymentMethods } from '@/hooks/useUserPaymentMethods';
import { Add } from '@mui/icons-material';
const PaymentMethods: React.FC = () => {
 const [isAddModalOpened, setIsAddModalOpened] = useState<boolean>(false);
 const { data: payment = [], isLoading } = useUserPaymentMethods();
 const handleClose = () => {
  setIsAddModalOpened(false);
 };
 const handleOpen = () => {
  setIsAddModalOpened(true);
 };
 return (
  <Stack gap={2}>
   {isLoading ? (
    [...Array(3)].map((_, idx) => <PaymentMethodSkeleton key={idx} />)
   ) : payment.length > 0 ? (
    payment.map((method) => <PaymentMethod {...method} />)
   ) : (
    <EmptyState
     title="No payment method saved"
     description="Checkout faster by saving a payment method"
     buttonText="Add Payment"
     onClick={handleOpen}
    />
   )}
   {payment.length < 3 && payment.length > 0 && (
    <Button
     startIcon={<Add />}
     variant="outlined"
     color="grey"
     size="small"
```

```
        sx={{ width: 'max-content' }}
        onClick={handleOpen}
       >
        Add Payment Method
       </Button>
     )}
                    <CustomModal maxWidth={600} title="Add Payment Method"
open={isAddModalOpened} onClose={handleClose}>
      <PaymentMethodForm mode="create" afterSubmit={handleClose} />
     </CustomModal>
   </Stack>
  );
};
export default PaymentMethods;
```

СТОРІНКА АДРЕСИ ДЛЯ ДОСТАВКИ

```
import { Stack, Typography } from "@mui/material";
import Addresses from "@/components/features/profile/Addresses";
const ShippingPage: React.FC = () => {
  return (
    <Stack gap={4} sx={{ width: "100%" }}>
      <Typography variant="h3">Shipping Address</Typography>
      <Addresses />
    </Stack>
  );
};
export default ShippingPage;
```

КОМПОНЕНТ СПИСКУ АДРЕС ДЛЯ ДОСТАВКИ

```
import { Button, Stack } from '@mui/material';
import { useState } from 'react';
import Address from './Item';
import EmptyState from '../EmptyState';
import CustomModal from '@/components/ui/layout/CustomModal';
import ShippingAddressSkeleton from '@/components/ui/loaders/skeletons/ShippingAddress';
import ShippingAddressForm from '@/components/forms/profile/ShippingAddress';
import { useUserShippingAddresses } from '@/hooks/useUserShippingAddresses';
import { Add } from '@mui/icons-material';
const Addresses: React.FC = () => {
  const [isCreateModalOpened, setIsCreateModalOpened] = useState<boolean>(false);
  const { data: addresses = [], isLoading } = useUserShippingAddresses();
  const handleClose = () => {
    setIsCreateModalOpened(false);
  };
  const handleOpen = () => {
    setIsCreateModalOpened(true);
  };
  return (
    <Stack gap={2}>
      {isLoading ? (
        [...Array(3)].map((_, idx) => <ShippingAddressSkeleton key={idx} />)
      ) : addresses.length > 0 ? (
```

```
      addresses.map((address) => <Address key={address.id} {...address} />)
    ) : (
      <EmptyState
        title="No shipping address saved"
        description="Checkout faster by saving a shipping address"
        buttonText="Add Shipping Address"
        onClick={handleOpen}
      />
    )}
    {addresses.length < 3 && addresses.length > 0 && (
      <Button
        startIcon={<Add />}
        variant="outlined"
        color="grey"
        size="small"
        sx={{ width: 'max-content' }}
        onClick={handleOpen}
      >
        Add Shipping Address
      </Button>
    )}
                <CustomModal maxWidth={580} title="Add Shipping Address"
open={isCreateModalOpened} onClose={handleClose}>
      <ShippingAddressForm mode="create" afterSubmit={handleClose} />
    </CustomModal>
  </Stack>
 );
};
export default Addresses;
```

КОМПОНЕНТ HEADER

```
import { Badge, Box, Button, IconButton, Stack } from '@mui/material';
import { useSelector } from 'react-redux';
import { useEffect, useState } from 'react';
import { useQuery, useQueryClient } from '@tanstack/react-query';
import Logo from '../Logo';
import Searchbar from './Searchbar';
import CustomContainer from '@/components/ui/layout/CustomContainer';
import Cart from '@/components/features/cart/Cart';
import { useCart } from '@/hooks/useCart';
import WishlistService from '@/api/wishlist/wishlist.service';
import { authSelector } from '@/redux/auth/auth.selectors';
import { GetAllWishlist } from '@/api/wishlist/wishlist.types';
import { FavoriteBorderOutlined, LocalMallOutlined, PersonOutline, StorefrontOutlined } from
'@mui/icons-material';
import { PAGE_LIMIT } from '@/constants';
const Header: React.FC = () => {
  const queryClient = useQueryClient();
  const { isAuth } = useSelector(authSelector);
  const [isCartOpened, setIsCartOpened] = useState<boolean>(false);
  const [scrolled, setScrolled] = useState(false);
```

```
  const { data: cart = [] } = useCart();
  const { data: wishlistTotal = 0 } = useQuery<GetAllWishlist, Error, number>({
    queryKey: ['wishlist', { page: 1, limit: PAGE_LIMIT }],
    queryFn: () => WishlistService.getAll({ page: 1, limit: PAGE_LIMIT }),
    select: (res) => res.total ?? 0,
    placeholderData: () =>
      queryClient.getQueryData<GetAllWishlist>(['wishlist', { page: 1, limit: PAGE_LIMIT }]) ??
{
        wishlist: [],
        total: 0,
      },
    staleTime: 60000,
    enabled: isAuth,
  });

  useEffect(() => {
    const handleScroll = () => {
      setScrolled(window.scrollY > 0);
    };

    window.addEventListener('scroll', handleScroll);
    return () => window.removeEventListener('scroll', handleScroll);
  }, []);

  return (
    <Box
      component="header"
      position="sticky"
      py={2}
      sx={{
        top: 0,
        left: 0,
        backgroundColor: 'common.white',
        zIndex: 100,
        transition: 'box-shadow 0.3s',
        boxShadow: scrolled ? 2 : 0,
      }}
    >
      <CustomContainer>
                    <Stack      flexDirection="row"      gap={3}      justifyContent="space-between"
alignItems="center">
          <Logo />
          <Button
            startIcon={<StorefrontOutlined />}
            size="small"
            color="grey"
            variant="outlined"
            href="/catalog"
            sx={{ textTransform: 'none' }}
          >
            Catalog
          </Button>
```

```
          <Searchbar />
          {isAuth ? (
            <Stack flexDirection="row" alignItems="center" gap={0.5}>
              <IconButton href="/profile/wishlist">
                <Badge color="primary" badgeContent={wishlistTotal}>
                  <FavoriteBorderOutlined />
                </Badge>
              </IconButton>
              <IconButton onClick={() => setIsCartOpened(true)}>
                <Badge color="primary" badgeContent={cart.reduce((acc, i) => acc + i.quantity, 0)}>
                  <LocalMallOutlined />
                </Badge>
              </IconButton>
              <Button href="/profile" variant="iconary" color="grey">
                <PersonOutline />
              </Button>
            </Stack>
          ) : (
            <Stack flexDirection="row" gap={2} alignItems="center">
              <Button href="/register" color="primary" variant="outlined" size="small">
                Sign Up
              </Button>
              <Button href="/login" color="primary" variant="contained" size="small">
                Sign In
              </Button>
            </Stack>
          )}
        </Stack>
      </CustomContainer>
       {isAuth && <Cart isOpened={isCartOpened} handleClose={() => setIsCartOpened(false)}
/>}
    </Box>
  );
};
export default Header;
```

КОМПОНЕНТ FOOTER

```
import { Box, Grid, Link, Stack, Typography } from "@mui/material";
import CustomContainer from "@/components/ui/layout/CustomContainer";
import Socials from "./Socials";
import Logo from "../Logo";
import { footerContacts, footerHelpMenu } from "@/data/menus";
const Footer: React.FC = () => {
  return (
    <Box component="footer" sx={{ backgroundColor: "common.black" }} py={4}>
      <CustomContainer>
        <Grid container spacing={4}>
          <Grid size={{ xs: 3 }}>
            <Stack gap={3}>
              <Logo color="light" />
              <Socials />
            </Stack>
```

```
        </Grid>
        <Grid size={{ xs: 6 }}>
          <Stack gap={2}>
                        <Typography color="common.white" fontSize={20} fontWeight={500}
textTransform="uppercase">
              Get Help
            </Typography>
            <Stack component="nav" gap={2}>
              {footerHelpMenu.map(({ title, href }) => (
                <Link href={href} color="grey">
                  {title}
                </Link>
              ))}
            </Stack>
          </Stack>
        </Grid>
        <Grid size={{ xs: 3 }}>
          <Stack gap={2}>
                        <Typography color="common.white" fontSize={20} fontWeight={500}
textTransform="uppercase">
              Contacts
            </Typography>
            <Stack component="nav" gap={2}>
              {footerContacts.map(({ title, href }) => (
                <Link href={href} color="grey">
                  {title}
                </Link>
              ))}
            </Stack>
          </Stack>
        </Grid>
      </Grid>
    </CustomContainer>
  </Box>
 );
};
export default Footer;
```

КОМПОНЕНТ PROFILE MENU

```
import { Link, Stack } from "@mui/material";
import { useLocation } from "react-router";
import { profileMenu } from "@/data/menus";
const ProfileMenu: React.FC = () => {
 const location = useLocation();
 return (
  <Stack gap={0.5}>
   {profileMenu.map(({ title, href, icon: Icon }, idx) => {
    const isActive = location.pathname === `/profile/${href}`;
    return (
     <Link
      key={idx}
      href={`/profile/${href}`}
```

```
            variant="plain"
            textTransform="uppercase"
            color={isActive ? "primary.main" : "grey.500"}
            fontWeight={500}
            sx={{
              display: "flex",
              gap: 1,
              alignItems: "center",
              p: 1,
              transition: "all .15s ease-in-out",
              "&:hover": { color: !isActive ? "common.black" : null },
            }}
          >
            <Icon />
            {title}
          </Link>
        );
      })}
    </Stack>
  );
};
export default ProfileMenu;
```

КОМПОНЕНТ PRIVATE ROUTE

```
import { useSelector } from 'react-redux';
import { Navigate, Outlet } from 'react-router';
import Loader from '@/components/ui/loaders/Loader';
import { authSelector } from '@/redux/auth/auth.selectors';
import { Statuses } from '@/types/enums.types';
const PrivateRoute: React.FC = () => {
  const { isAuth, status } = useSelector(authSelector);
  if (status === Statuses.LOADING) return <Loader />;
  if (status === Statuses.ERROR || !isAuth) return <Navigate replace to="/login" />;
  return <Outlet />;
};
export default PrivateRoute;
```

КОМПОНЕНТ EMPTY STATE

```
import { Stack, SvgIconTypeMap, Typography } from "@mui/material";
import { OverridableComponent } from "@mui/material/OverridableComponent";
type EmptyStateProps = {
  icon: OverridableComponent<SvgIconTypeMap<object, "svg">>;
  title: string;
  description: string;
  additional?: React.ReactNode;
};
const EmptyState: React.FC<EmptyStateProps> = ({ icon: Icon, title, description, additional })
=> {
  return (
    <Stack alignItems="center" gap={3} mt={8}>
      <Icon sx={{ width: 64, height: 64, color: "primary.main" }} />
      <Stack alignItems="center" gap={1}>
```

```
        <Typography variant="medium">{title}</Typography>
        <Typography>{description}</Typography>
      </Stack>
      {additional}
    </Stack>
  );
};
export default EmptyState;
```

ФАЙЛ НАЛАШТУВАННЬ AXIOS

```
import axios from "axios";
import AuthService from "@/api/auth/auth.service";
import { normalizeAxiosError } from "@/utils/normalizeAxiosError";
export const instance = axios.create({
  baseURL: `${import.meta.env.VITE_API_BASE_URL}/api/v1`,
  withCredentials: true,
});
instance.interceptors.response.use(
  (response) => response,
  async (error) => {
    const originalRequest = error.config as any;
    const isAuthEndpoint =
                                                originalRequest?.url?.includes("/auth/login")                ||
originalRequest?.url?.includes("/auth/register");
    if (
      error?.response?.status === 401 &&
      !originalRequest?._isRetry &&
      !originalRequest?.url?.includes("/auth/refresh") &&
      !isAuthEndpoint
    ) {
      originalRequest._isRetry = true;
      try {
        await AuthService.refresh();
        return instance.request(originalRequest);
      } catch (err) {
        AuthService.logout();
        window.location.href = "/login";
        return Promise.reject(normalizeAxiosError(err));
      }
    }
    if (!originalRequest?.url?.includes("/auth/refresh")) {
      return Promise.reject(normalizeAxiosError(error));
    }
  },
);
export async function httpGet<T>(url: string, config?: any): Promise<T> {
  const { data } = await instance.get<T>(url, config);
  return data;
}
export async function httpPost<T>(url: string, body?: any, config?: any): Promise<T> {
  const { data } = await instance.post<T>(url, body, config);
  return data;
```

```
}
export async function httpPatch<T>(url: string, body?: any, config?: any): Promise<T> {
  const { data } = await instance.patch<T>(url, body, config);
  return data;
}
export async function httpDelete<T>(url: string, config?: any): Promise<T> {
  const { data } = await instance.delete<T>(url, config);
  return data;
}
```

ФАЙЛ APP.TSX

```
import { BrowserRouter, Route, Routes } from 'react-router-dom';
import { lazy, useEffect } from 'react';
import { useAppDispatch } from './redux/store';
import PrimaryLayout from './components/layouts/PrimaryLayout';
import HomeLayout from './components/layouts/HomeLayout';
import AuthLayout from './components/layouts/AuthLayout';
import AppLayout from './components/layouts/AppLayout';
import ProfileLayout from './components/layouts/ProfileLayout';
import PrivateRoute from './components/providers/PrivateRoute';
import PublicRoute from './components/providers/PublicRoute';
import { refresh } from './redux/auth/auth.actions';
const RegisterPage = lazy(() => import('@/pages/auth/Register'));
const LoginPage = lazy(() => import('@/pages/auth/Login'));
const ForgotPasswordPage = lazy(() => import('@/pages/auth/ForgotPassword'));
const VerifyOtpPage = lazy(() => import('@/pages/auth/VerifyOtp'));
const VerifyOtpSuccessPage = lazy(() => import('@/pages/auth/VerifyOtpSuccess'));
const ResetPasswordPage = lazy(() => import('@/pages/auth/ResetPassword'));
const ResetPasswordSuccessPage = lazy(() => import('@/pages/auth/ResetPasswordSuccess'));
const HomePage = lazy(() => import('@/pages/Home'));
const CatalogPage = lazy(() => import('@/pages/Catalog'));
const CatalogProductPage = lazy(() => import('@/pages/CatalogProduct'));
const CheckoutPage = lazy(() => import('@/pages/checkout/Checkout'));
const CheckoutSuccessPage = lazy(() => import('@/pages/checkout/CheckoutSuccess'));
const NotFoundPage = lazy(() => import('@/pages/NotFound'));
const AccountPage = lazy(() => import('@/pages/profile/Account'));
const WishlistPage = lazy(() => import('@/pages/profile/Wishlist'));
const SettingsPage = lazy(() => import('@/pages/profile/Settings'));
const ReviewsPage = lazy(() => import('@/pages/profile/Reviews'));
const OrdersPage = lazy(() => import('@/pages/profile/Orders'));
const ShippingPage = lazy(() => import('@/pages/profile/Shipping'));
const PaymentPage = lazy(() => import('@/pages/profile/Payment'))
const ShippingAndDeliveryPage = lazy(() => import('@/pages/help/ShippingAndDelivery'));
const ReturnsPage = lazy(() => import('@/pages/help/Returns'));
const PaymentOptionsPage = lazy(() => import('@/pages/help/PaymentOptions'));
const ContactUsPage = lazy(() => import('@/pages/help/ContactUs'));
const TermsOfUsePage = lazy(() => import('@/pages/help/TermsOfUse'));
const PrivacyAndPolicyPage = lazy(() => import('@/pages/help/PrivacyAndPolicy'));
function App() {
  const dispatch = useAppDispatch();
  useEffect(() => {
    dispatch(refresh());
```

```
  }, [dispatch]);
  return (
    <BrowserRouter>
      <Routes>
        <Route element={<AuthLayout />}>
          <Route element={<PublicRoute />}>
            <Route path="/register" element={<RegisterPage />} />
            <Route path="/login" element={<LoginPage />} />
          </Route>
          <Route path="/forgot-password" element={<ForgotPasswordPage />} />
          <Route path="/verify-otp" element={<VerifyOtpPage />} />
          <Route path="/verify-otp-success" element={<VerifyOtpSuccessPage />} />
          <Route path="/reset-password" element={<ResetPasswordPage />} />
          <Route path="/reset-password-success" element={<ResetPasswordSuccessPage />} />
        </Route>
        <Route element={<AppLayout />}>
          <Route element={<HomeLayout />}>
            <Route path="/" element={<HomePage />} />
          </Route>
          <Route element={<PrimaryLayout />}>
            <Route path="/catalog" element={<CatalogPage />} />
            <Route path="/catalog/:slug" element={<CatalogProductPage />} />
            <Route path="*" element={<NotFoundPage />} />
            <Route path="/shipping-and-delivery" element={<ShippingAndDeliveryPage />} />
            <Route path="/returns" element={<ReturnsPage />} />
            <Route path="/payment-options" element={<PaymentOptionsPage />} />
            <Route path="/contact-us" element={<ContactUsPage />} />
            <Route path="/terms-of-use" element={<TermsOfUsePage />} />
            <Route path="/privacy-and-policy" element={<PrivacyAndPolicyPage />} />
            <Route element={<PrivateRoute />}>
              <Route path="/checkout" element={<CheckoutPage />} />
              <Route path="/checkout/success" element={<CheckoutSuccessPage />} />
              <Route element={<ProfileLayout />} path="/profile">
                <Route path="" element={<AccountPage />} />
                <Route path="wishlist" element={<WishlistPage />} />
                <Route path="settings" element={<SettingsPage />} />
                <Route path="reviews" element={<ReviewsPage />} />
                <Route path="orders" element={<OrdersPage />} />
                <Route path="shipping-address" element={<ShippingPage />} />
                <Route path="payment" element={<PaymentPage />} />
              </Route>
            </Route>
          </Route>
        </Route>
      </Routes>
    </BrowserRouter>
  );
}
export default App;
```

ФАЙЛ СХЕМИ БАЗИ ДАННИХ

```
generator client {
```

```
  provider = "prisma-client-js"
}
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}
enum LogLevel {
  INFO     @map("Info")
  WARN     @map("Warn")
  ERROR    @map("Error")
  SECURITY @map("Security")
}
enum ShipmentStatuses {
  DELIVERED @map("Delivered")
}
enum PaymentMethods {
  PAYPAL @map("Paypal")
  CARD   @map("Card")
  AMAZON @map("Amazon")
}
enum ShippingMethods {
  COURIER @map("courier")
  DHL     @map("dhl")
  PICKUP  @map("pickup")
}
enum Sizes {
  XXS
  XS
  S
  M
  L
  XL
  XXL
  XXXL
}
enum Rating {
  ONE   @map("1")
  TWO   @map("2")
  THREE @map("3")
  FOUR  @map("4")
  FIVE  @map("5")
}
model User {
  id           String @id @default(uuid()) @db.Uuid
  email        String @unique
  firstName    String @map("first_name")
  lastName     String @map("last_name")
  passwordHash String @map("password_hash")
  session      Session[]
  review       Review[]
  cart         Cart[]
  passwordReset PasswordReset[]
```

```
  wishlist      Wishlist[]
  order         Order[]
  payment       Payment[]
  addresses     ShippingAddress[]
  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")
  @@map("users")
}
model Session {
  id           String   @id @default(uuid()) @db.Uuid
  userId       String   @map("user_id") @db.Uuid
  refreshToken String   @unique @map("refresh_token")
  ip           String?
  userAgent    String?  @map("user_agent")
  deviceType   String?  @map("device_type")
  os           String?
  browser      String?
  country      String?
  expiresAt    DateTime @map("expires_at")
  user User @relation(fields: [userId], references: [id])
  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")
  @@index([userId])
  @@map("sessions")
}
model Category {
  id   String @id @default(uuid()) @db.Uuid
  name String
  slug String @unique
  products Product[]
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  @@map("categories")
}
model Product {
  id          String  @id @default(uuid()) @db.Uuid
  title       String
  description String
  slug        String
  posterUrl   String  @map("poster_url")
  price       Float
  discount    Float?
  available   Boolean
  categoryId String   @map("category_id") @db.Uuid
  category   Category @relation(fields: [categoryId], references: [id])
  sizes     Sizes[]       @default([])
  review    Review[]
  cartItem  CartItem[]
  images    ProductImage[]
  wishlist  Wishlist[]
  orderItem OrderItem[]
  createdAt DateTime @default(now()) @map("created_at")
```

```
  updatedAt DateTime @updatedAt @map("updated_at")
  @@unique([slug])
  @@index([available])
  @@index([price])
  @@map("products")
}
model ProductImage {
  id        String  @id @default(uuid()) @db.Uuid
  productId String  @map("product_id") @db.Uuid
  url       String
  alt       String?
  product Product @relation(fields: [productId], references: [id])
  createdAt DateTime @default(now()) @map("created_at")
  @@map("products_images")
}
model Review {
  id        String @id @default(uuid()) @db.Uuid
  userId    String @map("user_id") @db.Uuid
  productId String @map("product_id") @db.Uuid
  comment   String
  rating    Rating
  user    User    @relation(fields: [userId], references: [id])
  product Product @relation(fields: [productId], references: [id])
  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")
  @@unique([userId, productId])
  @@index([productId])
  @@map("reviews")
}
model Cart {
  id     String     @id @default(uuid()) @db.Uuid
  userId String     @map("user_id") @db.Uuid
  items  CartItem[]
  user User @relation(fields: [userId], references: [id])
  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")
  @@unique([userId])
  @@map("cart")
}
model CartItem {
  id        String @id @default(uuid()) @db.Uuid
  cartId    String @map("cart_id") @db.Uuid
  productId String @map("product_id") @db.Uuid
  quantity  Int
  size      Sizes
  product Product @relation(fields: [productId], references: [id])
  cart    Cart    @relation(fields: [cartId], references: [id], onDelete: Cascade)
  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")
  @@unique([cartId, productId, size])
  @@map("cart_items")
}
```

```
model PasswordReset {
  id     String  @id @default(uuid()) @db.Uuid
  userId String  @map("user_id") @db.Uuid
  otp    String
  used   Boolean @default(false)
  user User @relation(fields: [userId], references: [id])
  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")
  expiresAt DateTime @map("expires_at")
  @@index([userId])
  @@map("password_resets")
}
model Wishlist {
  id        String @id @default(uuid()) @db.Uuid
  userId    String @map("user_id") @db.Uuid
  productId String @map("product_id") @db.Uuid
  user    User    @relation(fields: [userId], references: [id])
  product Product @relation(fields: [productId], references: [id])
  createdAt DateTime @default(now()) @map("created_at")
  @@unique([userId, productId])
  @@index([productId])
  @@map("wishlists")
}
model Order {
  id              String          @id @default(uuid()) @db.Uuid
  userId          String          @map("user_id") @db.Uuid
  orderNumber     Int             @default(autoincrement()) @map("order_number")
  shippingAddress String          @map("shipping_address")
  trackingNumber  String          @map("tracking_number")
  subtotal        Float
  shipmentCost    Float           @default(20) @map("shipment_cost")
  shippingMethod  ShippingMethods @map("shipping_method")
  paymentMethod   PaymentMethods  @map("payment_method")
  status ShipmentStatuses @default(DELIVERED)
  user  User        @relation(fields: [userId], references: [id])
  items OrderItem[]
  createdAt DateTime @default(now()) @map("created_at")
  @@index([userId])
  @@map("orders")
}
model OrderItem {
  id        String @id @default(uuid()) @db.Uuid
  productId String @map("product_id") @db.Uuid
  orderId   String @map("order_id") @db.Uuid
  quantity  Int
  price     Float
  discount  Float?
  size      Sizes
  product Product @relation(fields: [productId], references: [id])
  order   Order   @relation(fields: [orderId], references: [id])
  createdAt DateTime @default(now()) @map("created_at")
  @@unique([orderId, productId, size])
```

```
  @@index([productId])
  @@map("order_items")
}
model Log {
  id        String   @id @default(uuid()) @db.Uuid
  timestamp DateTime @default(now())
  level     LogLevel
  action    String
  userId    String?  @map("user_id") @db.Uuid
  ip        String?
  userAgent String?  @map("user_agent")
  status    String?
  message   String?
  metadata  Json?
  @@index([timestamp])
  @@index([level])
  @@index([userId])
  @@map("logs")
}
model ShippingAddress {
  id      String  @id @default(uuid()) @db.Uuid
  userId  String  @map("user_id") @db.Uuid
  address String
  primary Boolean
  user User @relation(fields: [userId], references: [id])
  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")
  @@index([userId])
  @@map("shipping_addresses")
}
model Payment {
  id         String         @id @default(uuid()) @db.Uuid
  userId     String         @map("user_id") @db.Uuid
  method     PaymentMethods
  primary    Boolean
  email      String?
  cardIssuer String?        @map("card_issuer")
  last4      String?        @map("card_number")
  cardHolder String?        @map("card_holder")
  user User @relation(fields: [userId], references: [id])
  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")
  @@index([userId])
  @@map("payment_methods")
}

CEPBIC AUTH
import {
  ConflictException,
  Injectable,
  Logger,
  NotFoundException,
```

```
  UnauthorizedException,
} from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { JwtService } from '@nestjs/jwt';
import { MailerService } from '@nestjs-modules/mailer';
import { randomInt } from 'crypto';
import * as bcrypt from 'bcrypt';
import { PrismaService } from 'src/prisma/prisma.service';
import { LogService } from 'src/common/logging/log.service';
import { UserService } from '../user/user.service';
import { CreateSessionDto } from './dto/create-session.dto';
import { RegisterDto } from './dto/register.dto';
import { LoginDto } from './dto/login.dto';
import { VerifyOtpDto } from './dto/verify-otp.dto';
import { ResetPasswordDto } from './dto/reset-password.dto';
import { ChangePasswordDto } from './dto/change-password.dto';
@Injectable()
export class AuthService {
  private readonly logger = new Logger(AuthService.name);
  constructor(
    private readonly prisma: PrismaService,
    private readonly jwtService: JwtService,
    private readonly configService: ConfigService,
    private readonly logService: LogService,
    private readonly mailerService: MailerService,
    private readonly userService: UserService,
  ) {}
  async validateUser(email: string, password: string) {
    const user = await this.prisma.user.findUnique({
      where: { email },
    });
    if (!user) return null;
    const isValid = await bcrypt.compare(password, user.passwordHash);
    if (!isValid) return null;
    return user;
  }
  private createResetToken(userId: string) {
    return this.jwtService.sign(
      { id: userId, type: 'reset' },
      {
        secret: this.configService.get<string>('JWT_RESET_SECRET'),
        expiresIn: '15m',
      },
    );
  }
  private verifyResetToken(token: string) {
    try {
      const payload = this.jwtService.verify(token, {
        secret: this.configService.get<string>('JWT_RESET_SECRET'),
      });
      if (payload.type !== 'reset') throw new Error('Invalid token type');
      return payload.id;
```

```
  } catch (err) {
    console.error('JWT verification error:', err);
    throw new Error('Invalid or expired token');
  }
}
private async generateTokens(userId: string) {
  const payload = { id: userId };
  const accessToken = await this.jwtService.signAsync(payload, {
    secret: this.configService.get<string>('JWT_ACCESS_SECRET'),
    expiresIn: '15m',
  });
  const refreshToken = await this.jwtService.signAsync(payload, {
    secret: this.configService.get<string>('JWT_REFRESH_SECRET'),
    expiresIn: '7d',
  });
  return { accessToken, refreshToken };
}
async createSession(data: CreateSessionDto) {
  await this.prisma.session.create({ data });
}
async register(dto: RegisterDto) {
  const { email, password, firstName, lastName } = dto;
  this.logger.log(`Register attempt with email: ${email}`);
  const existingUser = await this.prisma.user.findUnique({
    where: { email },
  });
  if (existingUser) {
    this.logger.error('User with this email already exists');
    await this.logService.write({
      level: 'ERROR',
      action: 'auth.register',
      status: 'fail',
      message: 'User already exists',
      metadata: { dto },
    });
    throw new ConflictException('User with this email already exists');
  }
  const passwordHash = await bcrypt.hash(password, 10);
  const user = await this.prisma.user.create({
    data: {
      email,
      firstName,
      lastName,
      passwordHash,
    },
  });
  this.logger.log(`Registration successful with email: ${user.email}`);
  await this.logService.write({
    level: 'INFO',
    action: 'auth.register',
    userId: user.id,
    status: 'success',
```

```
  });
 }
 async login({ email, password }: LoginDto) {
  this.logger.log(`Login attempt: ${email}`);
  const user = await this.validateUser(email, password);
  if (!user) {
   this.logger.warn(`Login failed: ${email}`);
   await this.logService.write({
    level: 'SECURITY',
    action: 'auth.login',
    status: 'fail',
    message: 'Invalid credentials',
    metadata: { email },
   });
   throw new UnauthorizedException('Invalid email or password');
  }
  const tokens = await this.generateTokens(user.id);
  this.logger.log(`Login success: ${user.id}`);
  await this.logService.write({
   level: 'INFO',
   action: 'auth.login',
   userId: user.id,
   status: 'success',
  });
  return { ...tokens, userId: user.id };
 }
 async refreshTokens(userId: string, oldRefreshToken: string) {
  const session = await this.prisma.session.findUnique({
   where: { refreshToken: oldRefreshToken },
  });
  if (!session || session.userId !== userId) {
   throw new UnauthorizedException('Invalid refresh token');
  }
  const { accessToken, refreshToken } = await this.generateTokens(userId);
  await this.prisma.session.update({
   where: { id: session.id },
   data: {
    refreshToken,
    expiresAt: new Date(
     Date.now() +
      parseInt(this.configService.get<string>('JWT_REFRESH_EXPIRY')!),
    ),
   },
  });
  return { accessToken, refreshToken };
 }
 async sendForgotPasswordOtp(email: string) {
  this.logger.log(`Password reset request: ${email}`);
  const user = await this.prisma.user.findUnique({ where: { email } });
  if (!user) {
   this.logger.warn(
    `Password reset request for non-existent user: ${email}`,
```

```
    );
    return;
  }
  const otp = randomInt(1000, 9999);
  await this.prisma.passwordReset.deleteMany({
    where: { userId: user.id, used: false },
  });
  await this.prisma.passwordReset.create({
    data: {
      userId: user.id,
      otp: otp.toString(),
      expiresAt: new Date(Date.now() + 1000 * 60 * 30),
    },
  });
  await this.mailerService.sendMail({
    to: user.email,
    subject: 'Reset password',
    template: 'forgot-password',
    context: {
      email,
      otp,
    },
  });
  this.logger.log(`Password reset email sent: ${user.id}`);
  await this.logService.write({
    level: 'INFO',
    action: 'auth.forgotPassword',
    userId: user.id,
    status: 'success',
  });
}
async verifyOtp(dto: VerifyOtpDto) {
  const { otp, email } = dto;
  this.logger.log(`Verify otp attempt with otp: ${otp}...`);
  const user = await this.prisma.user.findUnique({ where: { email } });
  if (!user) throw new NotFoundException('User not found');
  const resetData = await this.prisma.passwordReset.findFirst({
    where: {
      userId: user.id,
      otp,
      expiresAt: { gte: new Date() },
      used: false,
    },
  });
  if (!resetData) {
    this.logger.warn(`Invalid or expired reset token: ${otp}...`);
    await this.logService.write({
      level: 'SECURITY',
      action: 'auth.',
      status: 'fail',
      message: 'Invalid or expired reset token',
    });
```

```
    throw new UnauthorizedException('Invalid or expired reset token');
  }
  await this.prisma.passwordReset.update({
    where: { id: resetData.id },
    data: { used: true },
  });
  return this.createResetToken(user.id);
}
async changePassword(userId: string, dto: ChangePasswordDto) {
  this.logger.log(`Change Password attempt: ${userId}`);
  await this.userService.get(userId);
  const userData = await this.prisma.user.findUnique({
    where: { id: userId },
    select: { passwordHash: true },
  });
  const isPasswordValid = await bcrypt.compare(
    dto.oldPassword,
    userData!.passwordHash,
  );
  if (!isPasswordValid) {
    await this.logService.write({
      level: 'SECURITY',
      action: 'auth.changePassword',
      status: 'fail',
      message: 'Invalid credentials',
      userId,
    });
    throw new UnauthorizedException("Current password doesn't match");
  }
  const newPasswordHash = await bcrypt.hash(dto.newPassword, 10);
  await this.prisma.user.update({
    where: { id: userId },
    data: { passwordHash: newPasswordHash },
  });
  this.logger.log(`Change password success: ${userId}`);
  await this.logService.write({
    level: 'INFO',
    action: 'auth.changePassword',
    userId,
    status: 'success',
  });
}
async resetPassword(dto: ResetPasswordDto) {
  const { resetToken, password } = dto;
  this.logger.log(`Password reset attempt with token: ${resetToken}...`);
  const userId = this.verifyResetToken(resetToken);
  if (!userId) {
    this.logger.warn(`Invalid or expired reset token: ${resetToken}...`);
    await this.logService.write({
      level: 'SECURITY',
      action: 'auth.resetPassword',
      status: 'fail',
```

```
      message: 'Invalid or expired reset token',
    });
    throw new UnauthorizedException('Invalid or expired reset token');
  }
  const newPasswordHash = await bcrypt.hash(password, 10);
  await this.prisma.user.update({
    where: { id: userId },
    data: { passwordHash: newPasswordHash },
  });
  this.logger.log(`Password reset successful: ${userId}`);
  await this.logService.write({
    level: 'INFO',
    action: 'auth.resetPassword',
    userId,
    status: 'success',
  });
}
async logout(refreshToken: string) {
  if (!refreshToken) return;
  const session = await this.prisma.session.findUnique({
    where: { refreshToken },
  });
  if (session) {
    await this.prisma.session.delete({ where: { refreshToken } });
    this.logger.log(`Logout is succesful for user: ${session.userId}`);
    await this.logService.write({
      level: 'INFO',
      action: 'auth.logout',
      userId: session.userId,
      status: 'success',
    });
  }
}
async logoutFromAnotherSession(userId: string, sessionId: string) {
  this.logger.log(`Logout attempt session with id: ${sessionId}`);
  await this.prisma.session.delete({
    where: { id: sessionId, userId },
  });
  this.logger.log(`Logout session with id successful: ${sessionId}`);
  await this.logService.write({
    level: 'INFO',
    action: 'auth.logoutId',
    userId,
    message: 'success',
  });
}
async logoutAll(userId: string, refreshToken: string) {
  this.logger.log(`Logout all sessions attempt user: ${userId}`);
  await this.prisma.session.deleteMany({
    where: { userId, NOT: { refreshToken } },
  });
  this.logger.log(`Logout all sessions is successful user: ${userId}`);
```

```
    await this.logService.write({
      level: 'INFO',
      action: 'auth.logoutAll',
      userId,
      message: 'success',
    });
  }
  async getSessions(userId: string, currentRefreshToken: string) {
    const sessions = await this.prisma.session.findMany({
      where: { userId, expiresAt: { gt: new Date() } },
      select: {
        id: true,
        deviceType: true,
        os: true,
        country: true,
        refreshToken: true,
        createdAt: true,
      },
      take: 5,
    });
    return sessions.map(({ refreshToken, ...s }) => ({
      ...s,
      isCurrent: refreshToken === currentRefreshToken,
    }));
  }
}
```

CEPBIC CART

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { PrismaService } from 'src/prisma/prisma.service';
import { ProductService } from '../product/product.service';
import { UpdateCartDto } from './dto/update-cart.dto';
import { DeleteFromCartDto } from './dto/delete-from-cart.dto';
@Injectable()
export class CartService {
  constructor(
    private readonly prisma: PrismaService,
    private readonly productService: ProductService,
  ) {}
  async getCart(userId: string) {
    const cart = await this.prisma.cart.findUnique({
      where: { userId },
      select: {
        items: {
          orderBy: { createdAt: 'desc' },
          select: {
            id: true,
            quantity: true,
            size: true,
            product: {
              select: {
                id: true,
```

```
            title: true,
            posterUrl: true,
            price: true,
            discount: true,
          },
        },
      },
    },
  },
 });
 return cart?.items || [];
}
async updateCart(userId: string, dto: UpdateCartDto) {
 const { productId, size, change } = dto;
 await this.productService.get(productId);
 const isSizeAvailable = await this.prisma.product.findUnique({
  where: { id: productId, sizes: { has: size } },
 });
 if (!isSizeAvailable) {
  throw new NotFoundException(`Size ${size} is not found for this product`);
 }
 const cart = await this.prisma.cart.upsert({
  where: { userId },
  update: {},
  create: { userId },
 });
 const existingItem = await this.prisma.cartItem.findUnique({
  where: {
   cartId_productId_size: {
    cartId: cart.id,
    productId,
    size,
   },
  },
 });
 if (existingItem) {
  let newQuantity = existingItem.quantity + change;
  if (newQuantity < 1) newQuantity = 1;
  if (newQuantity > 5) newQuantity = 5;
  await this.prisma.cartItem.update({
   where: {
    cartId_productId_size: {
     cartId: cart.id,
     productId,
     size,
    },
   },
   data: {
    quantity: newQuantity,
   },
  });
 } else if (change > 0) {
```

```
      await this.prisma.cartItem.create({
        data: {
          cartId: cart.id,
          productId,
          size,
          quantity: 1,
        },
      });
    }
  }
  async deleteFromCart(userId: string, dto: DeleteFromCartDto) {
    const cart = await this.prisma.cart.findUnique({
      where: { userId },
    });
    if (!cart) throw new NotFoundException('Cart not found');
    try {
      await this.prisma.cartItem.delete({
        where: {
          id: dto.id,
          cartId: cart.id,
        },
      });
    } catch {
      throw new NotFoundException('Item not found in cart');
    }
  }
}
```

СЕРВІС КАТЕГОРІЙ

```
import {
  BadRequestException,
  ConflictException,
  Injectable,
  InternalServerErrorException,
} from '@nestjs/common';
import { Prisma } from '@prisma/client';
import { PrismaService } from 'src/prisma/prisma.service';
import { CreateCategoryDto } from './dto/create-category.dto';
@Injectable()
export class CategoriesService {
  constructor(private readonly prisma: PrismaService) {}
  async all() {
    return this.prisma.category.findMany({
      orderBy: { name: 'desc' },
      select: {
        name: true,
        slug: true,
      },
    });
  }
  async create(dto: CreateCategoryDto) {
    try {
```

```
      await this.prisma.category.create({
        data: dto,
      });
    } catch (err) {
      if (
        err instanceof Prisma.PrismaClientKnownRequestError &&
        err.code === 'P2002'
      ) {
        throw new ConflictException('Category with this slug already exists');
      }
      throw new InternalServerErrorException('Failed to create category');
    }
  }
  async remove(categoryId: string) {
    try {
      await this.prisma.category.delete({ where: { id: categoryId } });
    } catch (err) {
      if (err instanceof Prisma.PrismaClientKnownRequestError) {
        throw new BadRequestException("Category with this id doesn't exist");
      }
      throw new InternalServerErrorException('Failed to delete category');
    }
  }
}
```

МОДУЛЬ ПОШТОВОГО КЛІЄНТА

```
import { ConfigModule, ConfigService } from '@nestjs/config';
import { Module } from '@nestjs/common';
import { MailerModule } from '@nestjs-modules/mailer';
import { HandlebarsAdapter } from '@nestjs-modules/mailer/dist/adapters/handlebars.adapter';
import { join } from 'path';
@Module({
  imports: [
    MailerModule.forRootAsync({
      imports: [ConfigModule],
      inject: [ConfigService],
      useFactory: (configService: ConfigService) => ({
        transport: {
          service: 'gmail',
          host: 'smtp.gmail.com',
          port: 465,
          secure: true,
          auth: {
            user: configService.get<string>('SMTP_USER'),
            pass: configService.get<string>('SMTP_PASS'),
          },
        },
        defaults: {
          from: '"Sandrela" <noreply@sandrela.xyz>',
        },
        template: {
          dir: join(__dirname, 'templates'),
```

```
        adapter: new HandlebarsAdapter(),
        options: { strict: true },
      },
    }),
  }),
 ],
 exports: [MailerModule],
})
export class MailModule {}
```

СЕРВІС USER

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { PrismaService } from 'src/prisma/prisma.service';
import { UpdateUserDto } from './dto/update-user.dto';
@Injectable()
export class UserService {
 constructor(private readonly prisma: PrismaService) {}
 async get(id: string) {
  const user = await this.prisma.user.findUnique({
   where: { id },
   select: {
    id: true,
    email: true,
    firstName: true,
    lastName: true,
   },
  });
  if (!user) {
   throw new NotFoundException('User is not found');
  }
  return user;
 }
 async update(id: string, dto: UpdateUserDto) {
  await this.get(id);
  await this.prisma.user.update({
   where: { id },
   data: dto,
  });
 }
}
```

СЕРВІС ЗАВАНТАЖЕННЯ КАРТИНОК НА ХМАРУ

```
import { Injectable, HttpException, HttpStatus } from '@nestjs/common';
import { S3Client, PutObjectCommand } from '@aws-sdk/client-s3';
import axios from 'axios';
import * as sharp from 'sharp';
import { v4 as uuidv4 } from 'uuid';
import { ConfigService } from '@nestjs/config';
@Injectable()
export class R2Service {
 private s3: S3Client;
 private bucket: string;
```

```
  private apiEndpoint: string;
  constructor(private readonly configService: ConfigService) {
    this.bucket = this.configService.getOrThrow<string>('R2_BUCKET');
    this.apiEndpoint = this.configService.getOrThrow<string>('S3_API');
    this.s3 = new S3Client({
      region: 'auto',
      endpoint: this.apiEndpoint,
      credentials: {
        accessKeyId: this.configService.getOrThrow<string>('R2_ACCESS_KEY_ID'),
        secretAccessKey: this.configService.getOrThrow<string>(
          'R2_SECRET_ACCESS_KEY',
        ),
      },
    });
  }
  async uploadFromUrl(imageUrl: string): Promise<string> {
    try {
      const response = await axios.get(imageUrl, {
        responseType: 'arraybuffer',
      });
      const buffer = Buffer.from(response.data);
      const webpBuffer = await sharp(buffer)
        .resize({
          width: 800,
          height: 1000,
          fit: 'inside',
        })
        .webp({ quality: 95 })
        .toBuffer();
      const key = `uploads/${uuidv4()}.webp`;
      await this.s3.send(
        new PutObjectCommand({
          Bucket: this.configService.get<string>('R2_BUCKET'),
          Key: key,
          Body: webpBuffer,
          ContentType: 'image/webp',
        }),
      );
      const url = `${this.configService.get<string>('R2_PUBLIC')}/${key}`;
      return url;
    } catch (error) {
      console.error('Upload failed full error:', error);
      throw new HttpException(`Failed to upload image`, HttpStatus.BAD_REQUEST);
    }
  }
}
```

ФАЙЛ APP.MODULE.TS

```
import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';
import { APP_FILTER, APP_GUARD } from '@nestjs/core';
import { ThrottlerGuard, ThrottlerModule } from '@nestjs/throttler';
```

```
import { PrismaModule } from './prisma/prisma.module';
import { AuthModule } from './modules/auth/auth.module';
import { LogService } from './common/logging/log.service';
import { AllExceptionsFilter } from './common/filters/all-exception.filter';
import { UserModule } from './modules/user/user.module';
import { GeoModule } from './modules/geo/geo.module';
import { ProductModule } from './modules/product/product.module';
import { CartModule } from './modules/cart/cart.module';
import { ReviewModule } from './modules/review/review.module';
import { WishlistModule } from './modules/wishlist/wishlist.module';
import { OrderModule } from './modules/order/order.module';
import { AddressModule } from './modules/address/address.module';
import { PaymentModule } from './modules/payment/payment.module';
import { R2Module } from './modules/r2/r2.module';
import { CategoriesModule } from './modules/categories/categories.module';
import { DevOnlyGuard } from './common/guards/dev-only.guard';
@Module({
  imports: [
    ConfigModule.forRoot({
      isGlobal: true,
      envFilePath: [
        `.env.${process.env.NODE_ENV || 'development'}`,
        '.env.development',
        '.env',
      ],
    }),
    ThrottlerModule.forRoot({ throttlers: [{ ttl: 60000, limit: 10000 }] }),
    PrismaModule,
    AuthModule,
    UserModule,
    GeoModule,
    ProductModule,
    CartModule,
    ReviewModule,
    WishlistModule,
    OrderModule,
    AddressModule,
    PaymentModule,
    R2Module,
    CategoriesModule,
  ],
  controllers: [],
  providers: [
    LogService,
    {
      provide: APP_FILTER,
      useClass: AllExceptionsFilter,
    },
    {
      provide: APP_GUARD,
      useClass: ThrottlerGuard,
    },
```

```
    {
      provide: APP_GUARD,
      useClass: DevOnlyGuard,
    },
  ],
})
export class AppModule {}
```

ФАЙЛ MAIN.TS

```
import helmet from 'helmet';
import * as cookieParser from 'cookie-parser';
import * as requestIp from 'request-ip';
import * as express from 'express';
import { NestFactory } from '@nestjs/core';
import { ValidationPipe } from '@nestjs/common';
import { AppModule } from './app.module';
import { ConfigService } from '@nestjs/config';
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  const configService = app.get(ConfigService);
  app.use(helmet());
  const corsEnv = configService.get<string>('FRONTEND_URL');
  const origins = corsEnv
    ? corsEnv.split(',').map((s) => s.trim())
    : ['http://localhost:5173'];
  app.enableCors({
    origin: origins,
    methods: ['GET', 'POST', 'PUT', 'PATCH', 'DELETE', 'OPTIONS'],
    credentials: true,
  });
  app.useGlobalPipes(
    new ValidationPipe({
      transform: true,
      whitelist: true,
    }),
  );
  app.setGlobalPrefix('api/v1');
  app.use(cookieParser());
  app.use(requestIp.mw());
  app.use(express.json({ limit: '500kb' }));
  app.use(express.urlencoded({ limit: '500kb', extended: true }));
  await app.listen(process.env.PORT ?? 3000);
}
bootstrap();
```