НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
В.о. завідувача кафедри
_____ Артем ВОЛОКИТА
(підпис)

“__” _____2026 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

на тему: Вебзастосунок електронної комерції для продажу рослин

Виконав : студент __4__ курсу, групи _ІМ-21_
(шифр групи)

_____Горай Ксенія Олексіївна_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____доцент, к.т.н., Роковий О. П._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль), асистент кафедри ОТ, доктор філософії, Міщенко Л. Д.
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ІСТ, к.т.н., Деведжіогуллари А. В._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2026 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Артем ВОЛОКИТА
(підпис)

“__” _____2026 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Горай Ксенії Олексіївни

1. Тема проєкту *Вебзастосунок електронної комерції для продажу рослин*
керівник проєкту Роковий Олександр Петрович, к.т.н., доцент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 3 червня 2026 року №2055-с

2. Термін здачі студентом закінченого проєкту *4 червня 2026 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Аналіз існуючих рішень.

Розділ 2. Огляд технологій та підходів у розробці вебзастосунку електронної комерції для продажу рослин.

Розділ 3. Розробка основного функціоналу вебзастосунку.

Розділ 4. Огляд та тестування розробленого проєкту.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема, алгоритм дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Міщенко Л. Д.		

7. Дата видачі завдання «12» січня 2026 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	20.01.2026-30.01.2026	
2.	*Вивчення та аналіз завдання*	02.02.2026-23.02.2026	
3.	*Розробка архітектури та загальної структури застосунку*	23.02.2026-20.03.2026	
4.	*Розробка структур окремих частин застосунку*	20.03.2026-16.04.2026	
5.	*Програмна реалізація застосунку*	16.04.2026-11.05.2026	
6.	*Оформлення пояснювальної записки*	11.05.2026-25.05.2026	
7.	*Захист програмного продукту*	26.05.2026	
8.	*Передзахист*	01.06.2026	
9.	*Захист*	16.06.2026	

Студент-дипломник _____ Ксенія ГОРАЙ
(підпис)

Керівник проєкту _____ Олександр РОКОВИЙ
(підпис)

АНОТАЦІЯ

У даній роботі проведено аналіз існуючих вебзастосунків електронної комерції та визначено їхні переваги й недоліки з метою розробки інтернет-магазину рослин. На основі аналізу сформовано вимоги до системи та спроєктовано її архітектуру. У результаті роботи розроблено вебзастосунок для продажу рослин із реалізацією автентифікації користувачів, управління товарами та категоріями, кошика покупок, оформлення замовлень, онлайн-оплати, системи купонів, адміністративної панелі та аналітики продажів. Також реалізовано кешування даних і механізми захисту інформації. Застосунок створено з використанням React, Node.js, Express, MongoDB, Redis та Stripe. Розроблений програмний продукт може бути використаний для автоматизації продажу рослин в онлайн-середовищі.

Ключові слова: вебзастосунок, інтернет-магазин, електронна комерція, React, Node.js, MongoDB, Redis, Stripe.

ANNOTATION

In this Bachelor’s Degree project, existing e-commerce web applications were analyzed, and their advantages and disadvantages were identified with the goal of developing a modern online plant store. Based on the analysis, system requirements and architecture were defined. As a result, a web application for plant sales was developed, including user authentication, product and category management, shopping cart, checkout, online payments, coupon system, admin dashboard, sales analytics, caching, and data protection mechanisms. The application was built using React, Node.js, Express, MongoDB, Redis, and Stripe. The developed software product can be used to automate online plant sales.

Keywords: web application, online store, e-commerce, React, Node.js, Express, MongoDB, Redis, Stripe.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Вебзастосунок електронної	4		
			комерції для продажу			
			рослин			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Вебзастосунок електронної	84		
			комерції для продажу			
			рослин			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Вебзастосунок електронної	1		
			комерції для продажу			
			рослин			
			Структурна схема			
			системи			
	А4	***ІАЛЦ.467200.005 Д2***	Вебзастосунок електронної	1		
			комерції для продажу			
			рослин			
			Функціональна схема			
	А4	***ІАЛЦ.467200.006 Д3***	Вебзастосунок електронної	1		
			комерції для продажу			
			рослин			
			Алгоритм дій програмного			
			забезпечення			
	А4	***ІАЛЦ.467200.007 Д4***	Вебзастосунок електронної	47		
			комерції для продажу			
			рослин			
			Текст програмного коду			

					ІАЛЦ.467200.001 ОА		
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*			
Розроб		Горай К.О.			*Вебзастосунок електронної комерції для продажу рослин* Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перев		Роковий О.П.					
						КПІ ім. Ігоря Сікорського ФІОТ ІМ-21	

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Вебзастосунок електронної комерції для продажу рослин*»

Київ – 2026

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання охоплює розробку вебзастосунку електронної комерції для продажу рослин, а також подальшу підтримку, оновлення та вдосконалення створеної системи.

Сфера застосування системи включає електронну комерцію, автоматизацію продажу товарів, управління каталогом продукції, обробку онлайн-замовлень та адміністрування інтернет-магазину. Система може використовуватися для продажу рослин, управління асортиментом, аналізу продажів та забезпечення зручної взаємодії між продавцем і покупцями.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Розробка даної системи здійснюється відповідно до завдання на виконання кваліфікаційної роботи для здобуття освітнього ступеня «бакалавр» за спеціальністю «Інженерія програмного забезпечення», затвердженого факультетом інформатики та обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка сучасного вебзастосунку для продажу рослин, який забезпечить зручний процес перегляду товарів, оформлення замовлень, онлайн-оплати, адміністрування товарів та аналізу продажів.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проєкту є офіційна документація, наукові публікації, тематичні статті в мережі Інтернет, а також науково-технічна література, присвячена розробці вебзастосунків та систем електронної комерції.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблений вебзастосунок електронної комерції для продажу рослин має відповідати таким вимогам:

- Забезпечувати зручний, зрозумілий та сучасний користувацький інтерфейс.
- Надавати користувачам можливість переглядати асортимент товарів
- Підтримувати функції реєстрації та входу в систему.
- Реалізовувати механізм формування кошика, оформлення замовлень і проведення онлайн-оплати.
- Забезпечувати можливість адміністрування каталогу товарів.
- Містити засоби моніторингу й аналізу продажів через адміністративну панель.
- Гарантувати належний рівень безпеки, захист персональних даних і надійність роботи системи.

5.2. Вимоги до програмного забезпечення

- ОС: Windows, macOS або Linux із сучасним веббраузером (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari).
- Підтримка JavaScript та стабільне підключення до мережі Інтернет.
- Серверна частина повинна підтримувати середовище виконання Node.js, систему керування базами даних MongoDB та Redis для кешування.

5.3. Вимоги до апаратної частини

Для клієнтського пристрою:

- ЦП з тактовою частотою не менше 1.8 ГГц.

- RAM не менше 4 ГБ.
- Вільний дисковий простір не менше 500 МБ.
- Підключення до мережі Інтернет.

Для серверної частини:

- Багатоядерний процесор.
- RAM не менше 8 ГБ.
- Достатній обсяг дискового простору для зберігання даних.
- Постійне мережеве з’єднання.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	20.01.2026-30.01.2026
Вивчення та аналіз завдання	02.02.2026-23.02.2026
Проєктування архітектури та загальної структури вебзастосунку.	23.02.2026-20.03.2026
Розробка окремих модулів системи (клієнтської та серверної частин).	20.03.2026-16.04.2026
Програмна реалізація системи	16.04.2026-11.05.2026
Виправлення помилок	11.05.2026-25.05.2026
Оформлення пояснювальної записки	11.05.2026-25.05.2026

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Вебзастосунок електронної комерції для продажу рослин*»

Київ – 2026

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API	(Application Programming Interface) Інтерфейс програмування застосунків
CDN	(Content Delivery Network) Мережа доставки контенту
CRUD	(Create, Read, Update, Delete) Базові операції з даними
CSS	(Cascading Style Sheets) Каскадні таблиці стилів
DOM	(Document Object Model) Об'єктна модель документа
HTML	(HyperText Markup Language) Мова гіпертекстової розмітки
JSON	(JavaScript Object Notation) Формат обміну даними на основі JavaScript
JWT	(JSON Web Token) Токен аутентифікації у форматі JSON
MERN	MongoDB, Express.js, React.js, Node.js — технологічний стек
NoSQL	(Not Only SQL) Нереляційна база даних
npm	(Node Package Manager) Менеджер пакетів для Node.js
PCI DSS	(Payment Card Industry Data Security Standard) Стандарт безпеки платіжних карток
REST	(Representational State Transfer) Архітектурний стиль побудови API
SPA	(Single Page Application) Односторінковий застосунок
SQL	(Structured Query Language) Мова структурованих запитів
URL	(Uniform Resource Locator) Уніфікований локатор ресурсу
UI	(User Interface) Користувацький інтерфейс
UX	(User Experience) Користувацький досвід
XSS	(Cross-Site Scripting) Міжсайтовий скриптинг — тип атаки
BSON	(Binary JSON) Бінарний формат зберігання даних у MongoDB

ВСТУП

У сучасних умовах розвитку інформаційних технологій використання вебзастосунків стало важливою складовою ведення бізнесу та організації продажів. Особливо це стосується електронної комерції, яка дає змогу забезпечити швидку взаємодію між продавцем і покупцем, спростити процес замовлення товарів та автоматизувати значну частину торговельних процесів. З кожним роком онлайн-магазини стають дедалі популярнішими, адже дозволяють користувачам отримувати доступ до товарів у зручний час та незалежно від місця перебування.

Серед напрямів електронної комерції окремої уваги заслуговують нішеві магазини, зокрема магазини рослин. Попит на рослини постійно зростає, а разом із цим виникає потреба у сучасних цифрових рішеннях, які дозволяють зручно представляти каталог товарів, спрощувати процес покупки та забезпечувати якісне управління асортиментом. Традиційні підходи до продажу поступово поступаються онлайн-формату, що надає більше можливостей як для продавців, так і для покупців.

Існуючі платформи електронної комерції пропонують широкий набір можливостей, проте часто є або надто універсальними, або недостатньо адаптованими до потреб окремого бізнесу. Саме тому актуальною є розробка спеціалізованого вебзастосунку, який поєднає зручний інтерфейс, сучасний функціонал вебзастосунку електронної комерції, безпечну систему оплат та засоби адміністрування.

Метою даного дипломного проєкту є розробка вебзастосунку електронної комерції для продажу рослин, який надасть користувачам зручний інструмент для перегляду та придбання товарів, а адміністратору — засоби для управління каталогом, замовленнями та аналітикою продажів. Використання такого вебзастосунку дозволить автоматизувати процес онлайн-продажу рослин та забезпечити ефективну взаємодію між магазином і клієнтами.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Основні поняття

Стрімкий розвиток інформаційних технологій та цифровізація бізнес-процесів сприяли активному поширенню електронної комерції [2]. Якщо раніше продаж товарів переважно здійснювався через фізичні магазини, то з розвитком мережі Інтернет і вебтехнологій дедалі більше компаній почали переносити свою діяльність в онлайн-середовище. Це дозволило зробити процес купівлі товарів швидшим, зручнішим та доступнішим для користувачів незалежно від їхнього місцезнаходження.

З розвитком вебтехнологій змінилися і самі підходи до створення онлайн-сервісів. Якщо раніше вебресурси мали переважно інформаційний характер, то сьогодні вони перетворилися на повноцінні програмні системи, здатні виконувати складну бізнес-логіку, працювати з великими обсягами даних та забезпечувати інтерактивну взаємодію з користувачем [3]. Такі системи дістали назву вебзастосунків. Вебзастосунок — це програмний продукт, доступ до якого здійснюється через браузер, а взаємодія з його функціоналом реалізується через користувацький інтерфейс [4].

Одним із найбільш поширених типів вебзастосунків є інтернет-магазини. Інтернет-магазин являє собою програмну систему, призначену для представлення товарів, організації процесу їх вибору, оформлення замовлення та проведення онлайн-оплати [5]. Типовий інтернет-магазин включає каталог товарів, систему пошуку та фільтрації, кошик покупок, оформлення замовлень, особистий кабінет користувача та інтеграцію з платіжними сервісами.

У межах даної роботи розглядається вебзастосунок інтернет-магазину рослин. Особливістю такого типу системи є поєднання стандартного

функціоналу електронної комерції з можливістю зручного представлення товарів, систем знижок та інструментів адміністрування асортименту. Для таких систем важливим є не лише забезпечення продажу товарів, а й створення комфортного користувацького досвіду.

Важливою складовою будь-якого вебзастосунку є інтерфейс користувача. Саме через нього забезпечується взаємодія користувача із системою, навігація сторінками та оформлення покупки. Зручний та інтуїтивно зрозумілий інтерфейс значною мірою впливає на ефективність використання системи та загальне сприйняття сервісу [6].

Не менш важливим поняттям у контексті електронної комерції є безпека інформації. Оскільки вебзастосунки працюють із персональними даними користувачів, обробляють замовлення та інтегруються з платіжними системами, необхідним є забезпечення захисту облікових записів, безпечної автентифікації та запобігання несанкціонованому доступу до даних [7].

Ще однією важливою характеристикою сучасних e-commerce рішень є продуктивність системи. Швидкість завантаження сторінок, оперативна обробка запитів та стабільна робота застосунку є важливими факторами для забезпечення зручності користувачів. Для цього застосовуються сучасні підходи до оптимізації, зокрема кешування даних, оптимізація запитів до баз даних та використання сучасних серверних технологій [8].

Таким чином, вебзастосунки інтернет-магазинів є актуальним і ефективним рішенням для організації онлайн-продажів. Поєднання функціональності електронної комерції, зручного інтерфейсу, безпеки та продуктивності робить їх важливим інструментом сучасного бізнесу та основою для побудови спеціалізованих рішень, зокрема інтернет-магазину рослин.

1.2 Огляд існуючих вебзастосунків для інтернет-магазинів з продажу рослин

Далі будуть розглянуті існуючі вебзастосунки для продажу рослин, зокрема, такі рішення, як Флоріум.ua, Agro-Market та The Sill, їхній функціонал, особливості реалізації, а також переваги й недоліки з точки зору організації електронної комерції та користувацького досвіду. Це дозволить проаналізувати наявні підходи до побудови інтернет-магазинів у даній предметній області та визначити функціональні рішення, які доцільно врахувати при розробці власного вебзастосунку.

1.2.1 Флоріум.ua

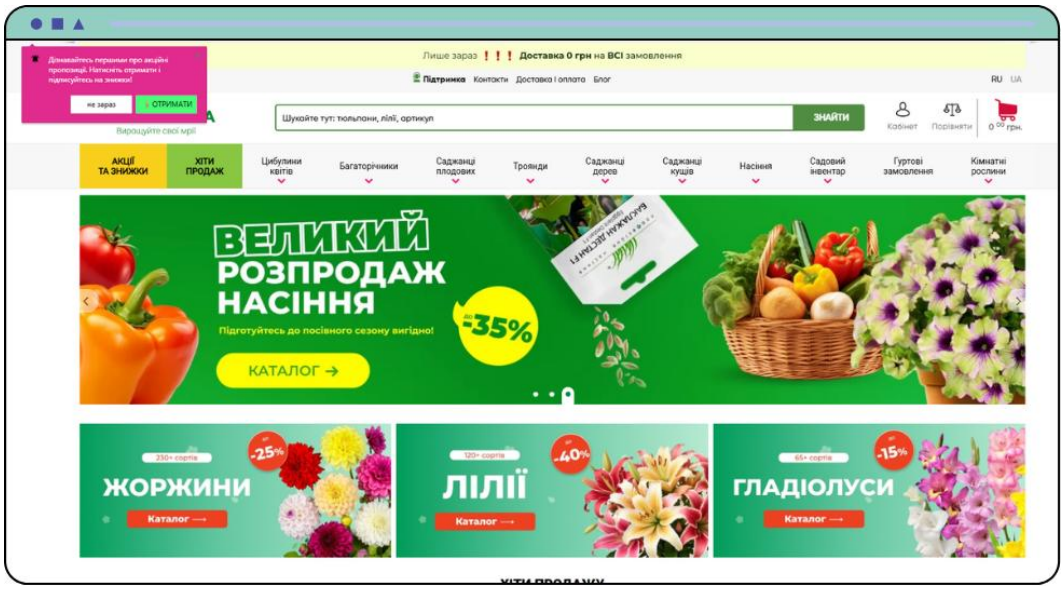


Рисунок 1.1 – Інтерфейс користувача вебзастосунку Флоріум.ua

Флоріум.ua — це вебзастосунок електронної комерції, орієнтований на продаж садових, декоративних і кімнатних рослин, а також супутніх товарів для садівництва. Даний сервіс поєднує функціонал інтернет-магазину з елементами інформаційної платформи, надаючи користувачам можливість не лише

переглядати й замовляти продукцію, а й отримувати корисну інформацію щодо вирощування та догляду за рослинами [9].

Переваги Флоріум.ua:

- зручна та логічно структурована навігація каталогу;
- широкий асортимент рослин і супутніх товарів;
- наявність фільтрів для швидкого пошуку продукції;
- адаптивний інтерфейс для мобільних пристроїв;

Недоліки Флоріум.ua:

- надмірно насичений візуальний дизайн;
- pop-up вікна можуть відволікати під час взаємодії із сайтом;
- окремі сторінки перевантажені інформацією;
- відсутня англомовна версія вебзастосунку.

1.2.2 Agro-Market

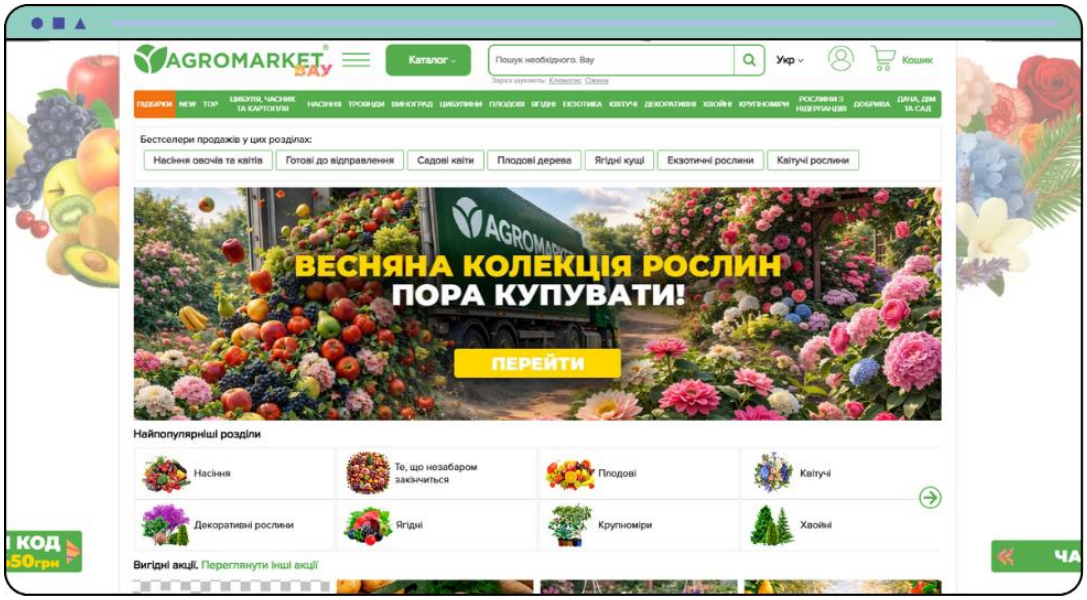


Рисунок 1.2 – Інтерфейс користувача вебзастосунку Agro-Market

Agro-Market — це онлайн-платформа для продажу рослин, саджанців, насіння та товарів для садівництва, яка пропонує користувачам широкий вибір

продукції та додаткові інструменти для оформлення й супроводу замовлень. Система містить розгалужений каталог товарів, функції пошуку та фільтрації, а також додаткові сервіси для взаємодії з користувачами [10].

Переваги Agro-Market:

- великий каталог товарів;
- розвинена система фільтрів для пошуку продукції;
- наявність інтеграції чат-боту;
- наявність мобільного додатка.

Недоліки Agro-Market:

- інтерфейс сайту місцями перевантажений елементами;
- складна навігація через велику кількість категорій і фільтрів;
- дизайн виглядає дещо застарілим;
- велика кількість рекламних блоків може відволікати користувача;
- відсутня англомовна версія вебзастосунку.

1.2.3 The Sill

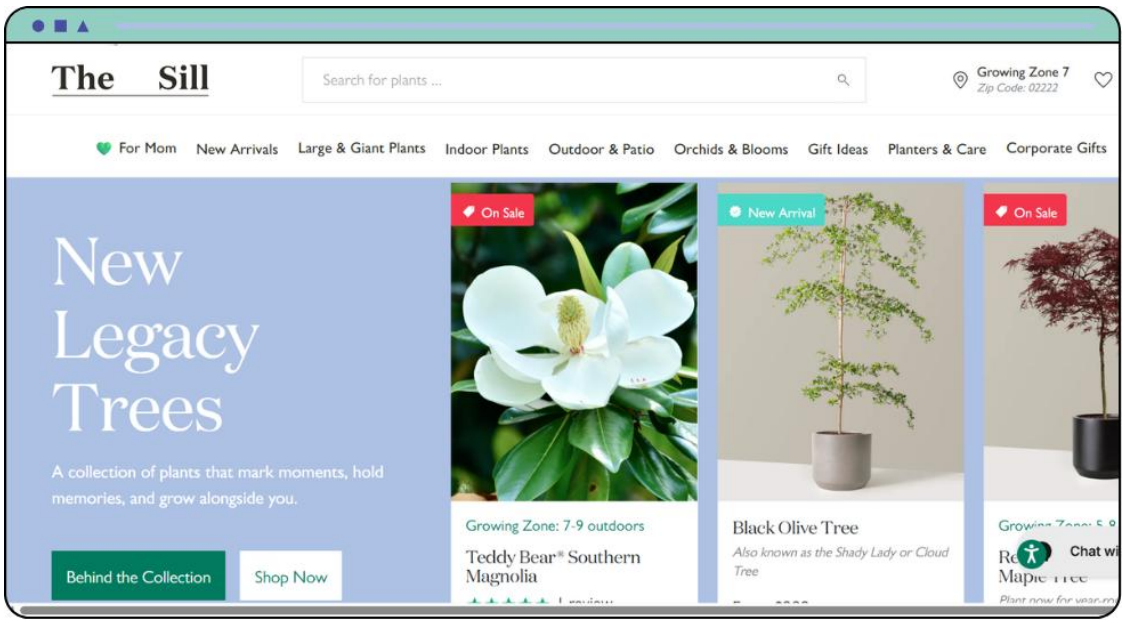


Рисунок 1.3 – Інтерфейс користувача вебзастосунку The Sill

The Sill — американський вебзастосунок, спеціалізований на продажі кімнатних рослин та супутніх товарів для догляду за ними. Платформа поєднує можливості інтернет-магазину з інформаційними матеріалами для користувачів, надаючи користувачам можливість не лише придбати рослини, а й отримати рекомендації щодо догляду та вибору товарів [11].

Переваги The Sill:

- сучасний та мінімалістичний інтерфейс користувача;
- зручна категоризація товарів і система фільтрації;
- наявність рекомендацій щодо вибору та догляду за рослинами;
- підтримка онлайн-оплати та зручного процесу оформлення замовлення.

Недоліки The Sill:

- обмежені можливості персоналізації замовлень;
- відсутність локалізації для різних мов;
- обмежені можливості персоналізації для користувачів;
- функціонал орієнтований переважно на клієнтську частину, а не на гнучке адміністрування.

1.3 Порівняння

Після аналізу вебзастосунків Флоріум.ua, Agro-Market та The Sill доцільно узагальнити їхні ключові характеристики у порівняльній таблиці. Такий підхід дозволяє наочно оцінити функціональні можливості існуючих рішень, визначити їх переваги та недоліки, а також сформувати вимоги до вебзастосунку, що розробляється.

Таблиця 1.1 – Порівняння існуючих рішень

Критерій	Флоріум.ua	Agro-Market	The Sill
Зручність користування	-	-	+
Сучасний дизайн	-	-	+
Підтримка англомовної версії	-	-	+
Адміністративна панель	-	-	-
Мобільний додаток	-	+	-
Сучасна система онлайн-оплати	+	+	+

Проаналізувавши таблицю, можна побачити, що The Sill виграє за зручністю користування та сучасністю інтерфейсу, тоді як Флоріум.ua і Agro-Market мають базовий функціонал, але поступаються закордонному аналогу за користувацьким досвідом. Основними недоліками українських рішень є

незручна навігація, застарілий дизайн, відсутність англомовної версії та обмежений функціонал адміністрування.

Це свідчить про доцільність розробки сучасного вебзастосунку з підтримкою англійської мови, зручним інтерфейсом та ширшим функціоналом.

ВИСНОВОК ДО РОЗДІЛУ 1

На основі проведеного аналізу існуючих вебзастосунків можна зробити висновок, що наявні рішення для продажу рослин мають як переваги, так і суттєві недоліки. До сильних сторін розглянутих аналогів можна віднести наявність базового функціоналу інтернет-магазину, зокрема каталогів товарів, фільтрації, онлайн-оплати та адаптивності для різних пристроїв. Окремі рішення також пропонують додаткові можливості, такі як мобільний додаток, рекомендації щодо догляду за рослинами або допоміжні сервіси для користувачів.

Разом із тим, аналіз показав, що існуючі вебзастосунки не повною мірою задовольняють сучасні вимоги до зручності користування, дизайну та функціональності. Для українських рішень характерними недоліками є перевантажений або застарілий інтерфейс, недостатньо продумана навігація, відсутність англомовної версії, а також обмежені можливості адміністрування. Навіть більш сучасні закордонні аналоги не завжди поєднують в одному продукті зручний користувацький інтерфейс, гнучкий функціонал та можливості масштабування.

Проведений аналіз дозволяє зробити висновок, що існує потреба у розробці сучасного вебзастосунку для продажу рослин, який поєднуватиме сильні сторони існуючих аналогів та усуватиме їхні недоліки. Такий застосунок має забезпечувати зручний і сучасний інтерфейс, підтримку української та англійської мов, безпечну онлайн-оплату, а також функціонал для ефективного управління товарами та замовленнями.

Отже, результатом аналізу існуючих рішень є обґрунтування доцільності розробки власного вебзастосунку, орієнтованого на сучасні вимоги електронної

комерції та покращення користувацького досвіду в сфері онлайн-продажу рослин.

РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЙ ТА ПІДХОДІВ У РОЗРОБЦІ ВЕБЗАСТОСУНКУ ЕЛЕКТРОННОЇ КОМЕРЦІЇ ДЛЯ ПРОДАЖУ РОСЛИН

2.1 Вибір архітектури вебзастосунку

Для розробки вебзастосунку інтернет-магазину рослин обрано клієнт-серверну архітектуру, яка є одним із найбільш поширених підходів до побудови сучасних вебсистем. Основна ідея такого підходу полягає у розподілі системи на дві незалежні частини: клієнтську, що відповідає за взаємодію з користувачем, і серверну, яка забезпечує обробку даних, бізнес-логіку та зберігання інформації. Це дає змогу зробити систему більш структурованою, масштабованою та зручною в підтримці.

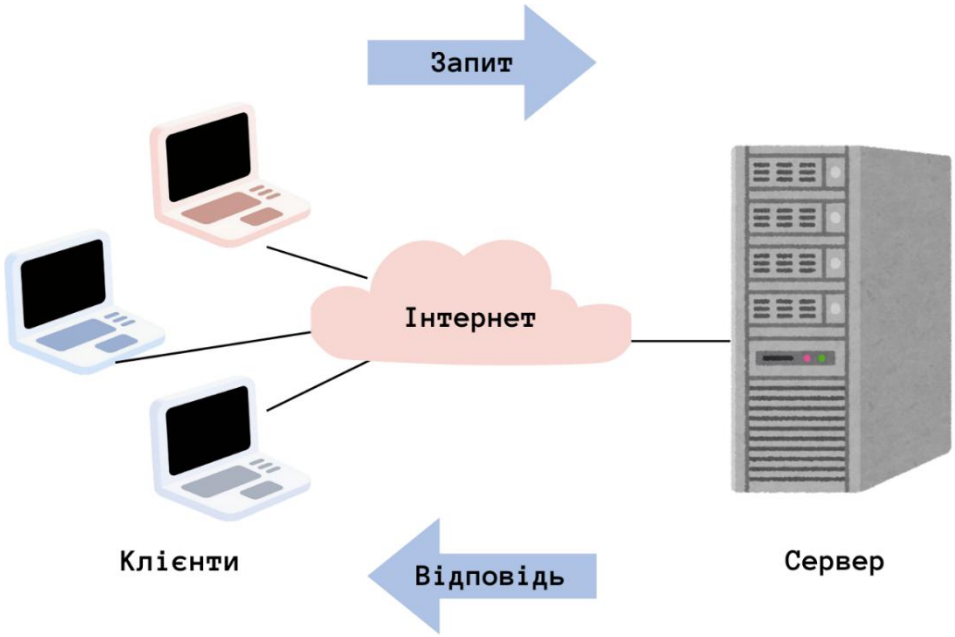


Рисунок 2.1 – Схема клієнт-серверної архітектури вебзастосунку

Клієнт-серверна архітектура складається з двох основних компонентів — клієнта та сервера, між якими відбувається обмін даними через запити та

відповіді. Такий підхід дозволяє централізовано керувати даними, реалізувати безпечну обробку інформації та ефективно організувати роботу вебзастосунку [12].

Взаємодія між клієнтом і сервером здійснюється за допомогою HTTP-запитів та відповідей у форматі JSON, що є одним із найпоширеніших стандартів обміну даними у веброзробці. Для реалізації взаємодії між компонентами системи використовуються REST-принципи, які передбачають використання стандартних HTTP-методів, таких як GET, POST, PUT та DELETE, для роботи з ресурсами вебзастосунку.

Клієнт — це частина системи, з якою безпосередньо взаємодіє користувач через веббраузер. Вона забезпечує відображення інтерфейсу інтернет-магазину, перегляд товарів, пошук рослин, роботу з кошиком та оформлення замовлення. Клієнтська частина також надсилає запити на сервер для отримання або оновлення даних.

Сервер — це частина системи, яка приймає та обробляє запити від клієнта, реалізує бізнес-логіку застосунку та взаємодіє з базою даних. Сервер забезпечує реалізацію основної бізнес-логіки вебзастосунку, зокрема авторизацію користувачів, управління товарами та категоріями, обробку замовлень, інтеграцію платіжної системи, а також функціонал адміністративної панелі й аналітики продажів. Окрім цього, серверна частина відповідає за взаємодію з базою даних та контроль доступу до ресурсів системи.

У розроблюваному вебзастосунку клієнтська частина відповідає за зручну взаємодію користувача з магазином, а серверна — за надійну обробку запитів, зберігання даних у MongoDB, кешування за допомогою Redis та реалізацію безпечної автентифікації на основі JWT. Так можна поєднати сучасний користувацький інтерфейс із продуктивною серверною логікою.

Отже, використання клієнт-серверної архітектури є доцільним для даного вебзастосунку, оскільки вона забезпечує розподіл функцій між компонентами

системи, підвищує безпеку, спрощує масштабування та дозволяє ефективно реалізувати функціонал інтернет-магазину.

2.2 Вибір технологічного стеку

Важливим етапом проєктування вебзастосунку є вибір технологічного стеку, на основі якого буде реалізовано програмний продукт. Під технологічним стеком розуміють сукупність технологій, інструментів і програмних засобів, що використовуються для розробки застосунку. Вдалий вибір стеку впливає на продуктивність системи, її масштабованість, вартість підтримки та зручність подальшого розвитку [13].

Типовий технологічний стек для вебзастосунку складається з трьох основних рівнів: клієнтської частини (front-end), серверної частини (back-end) та бази даних. Такий підхід дозволяє розділити систему на окремі логічні компоненти, кожен з яких відповідає за власний набір задач.

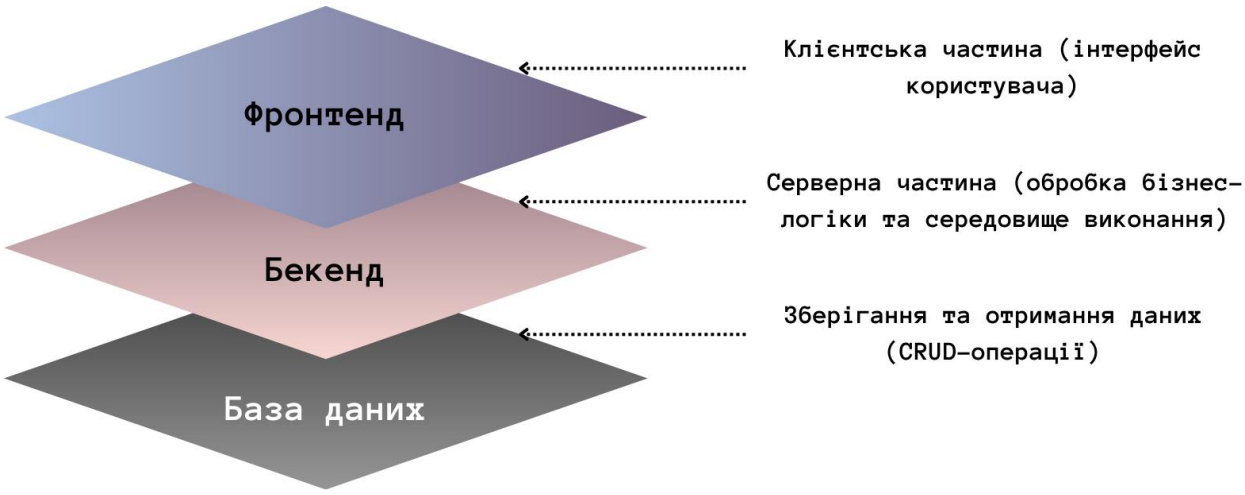


Рисунок 2.2 – Основні складові технологічного стеку

Клієнтська частина відповідає за інтерфейс користувача та взаємодію з вебзастосунком. Саме на цьому рівні реалізуються сторінки каталогу товарів,

кошик, оформлення замовлення, особистий кабінет користувача та інші елементи інтерфейсу. Основою фронтенд-розробки зазвичай є HTML, CSS та JavaScript, а для побудови сучасних інтерфейсів часто використовуються спеціалізовані бібліотеки та фреймворки.

Серверна частина забезпечує реалізацію бізнес-логіки застосунку, обробку запитів від клієнта, взаємодію з базою даних, автентифікацію користувачів, обробку замовлень та інші серверні процеси. Для її реалізації використовуються серверні мови програмування, фреймворки та середовища виконання.

Третім компонентом технологічного стеку є база даних, яка забезпечує зберігання та отримання інформації. У базі даних зберігаються відомості про користувачів, товари, категорії, замовлення та інші дані, необхідні для роботи системи. Залежно від особливостей проєкту можуть використовуватися реляційні або NoSQL бази даних.

Для даного вебзастосунку обрано full-stack підхід, що охоплює всі три рівні технологічного стеку. Це дозволяє створити цілісну систему, де клієнтська частина, серверна логіка та робота з даними узгоджено взаємодіють між собою. На основі цього підходу в наступному підрозділі розглядається вибір MERN-стеку, який використано для створення розроблюваного інтернет-магазину.

2.2.1 MERN-стек для розробки вебзастосунку

Для реалізації розроблюваного вебзастосунку обрано технологічний стек MERN, який є одним із сучасних підходів до створення full-stack вебсистем. MERN — це поєднання чотирьох технологій: MongoDB, Express.js, React.js та Node.js, які разом формують єдине середовище для розробки клієнтської частини, серверної логіки та роботи з даними [14].

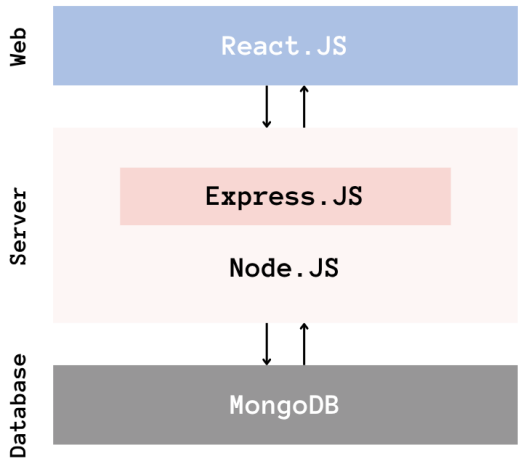


Рисунок 2.3 – Структура MERN-стеку вебзастосунку

Вибір саме MERN-стеку обумовлений тим, що всі його компоненти базуються на JavaScript, що дозволяє використовувати одну мову програмування на всіх рівнях системи. Такий підхід спрощує розробку, зменшує складність інтеграції між компонентами та полегшує підтримку програмного продукту. Крім того, MERN добре підходить для побудови динамічних вебзастосунків з великою кількістю взаємодій із даними, до яких належить і інтернет-магазин.

React.js використовується для реалізації клієнтської частини застосунку [15]. Його компонентний підхід дозволяє будувати гнучкий інтерфейс користувача, повторно використовувати окремі елементи системи та забезпечувати швидке оновлення сторінок без повного перезавантаження. Це особливо важливо для реалізації каталогу товарів, кошика, оформлення замовлення та адміністративної панелі.

Для серверної частини обрано Node.js та Express.js. Node.js забезпечує виконання серверного JavaScript-коду [16], а Express.js використовується для створення API, маршрутизації та обробки HTTP-запитів [17]. Завдяки цьому реалізується взаємодія між клієнтом і сервером, робота з товарами, замовленнями, авторизацією користувачів та платіжною системою.

Для зберігання даних використовується MongoDB, яка є документоорієнтованою NoSQL базою даних [18]. Її вибір зумовлений гнучкістю структури даних та інтеграцією з JavaScript-екосистемою. MongoDB доцільно використовувати для зберігання товарів, категорій, користувачів, замовлень, купонів та інших сутностей інтернет-магазину.

Важливою перевагою MERN-стеку є також те, що обмін даними між його компонентами відбувається у JSON-подібному форматі, що спрощує передачу даних між фронтендом, сервером і базою даних. Це зменшує складність реалізації CRUD-операцій та підвищує ефективність роботи застосунку [14].

Вибір MERN також обумовлений його масштабованістю та широкими можливостями розширення. Стек легко інтегрується з додатковими технологіями, які використовуються в даному проєкті, зокрема Redis для кешування, Stripe для онлайн-платежів, Cloudinary для роботи із зображеннями та JWT для автентифікації користувачів.

Отже, MERN-стек було обрано як основу розробки через його цілісність, продуктивність, зручність та відповідність вимогам e-commerce застосунку. Його використання дозволяє реалізувати сучасний, масштабований та функціональний вебзастосунок.

2.2.2 MongoDB як база даних

Одним із компонентів обраного MERN-стеку є MongoDB — NoSQL база даних, яка використовується для зберігання та обробки даних вебзастосунку.

На відміну від реляційних баз даних, де інформація зберігається у вигляді таблиць, MongoDB працює з колекціями та документами у JSON-подібному форматі BSON [19]. Такий підхід дозволяє гнучко описувати структуру даних та легко змінювати її за потреби без складних модифікацій схеми бази даних.

Це є важливою перевагою для e-commerce застосунків, де структура даних може змінюватися в процесі розвитку системи.

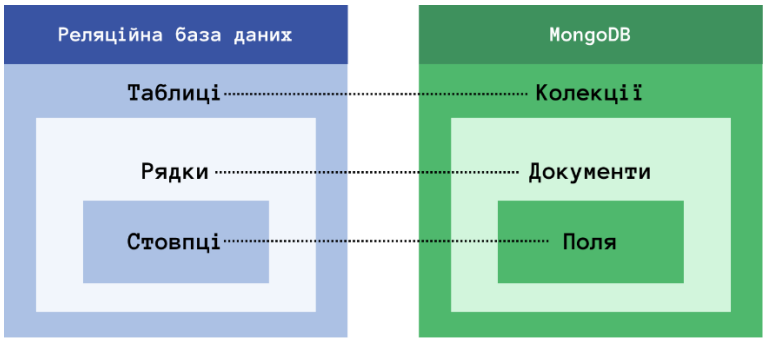


Рисунок 2.4 – Порівняння структури реляційної бази даних та MongoDB

У розроблюваному вебзастосунку MongoDB використовується для зберігання даних про користувачів, товари, категорії рослин, замовлення, купони та інші сутності системи.

Ще однією перевагою MongoDB є зручна інтеграція з Node.js та Express.js, що спрощує реалізацію CRUD-операцій: створення, отримання, оновлення та видалення даних. Це дозволяє ефективно реалізувати керування каталогом товарів, обробку замовлень і адміністративний функціонал вебзастосунку.

Важливими факторами вибору MongoDB також є її масштабованість та продуктивність. База даних добре працює з великими обсягами інформації та підтримує горизонтальне масштабування, що є актуальним для інтернет-магазинів із потенційним зростанням кількості користувачів і товарів.

Для взаємодії з MongoDB у проєкті використовується бібліотека Mongoose, яка забезпечує моделювання даних, валідацію та спрощує роботу з колекціями бази даних через JavaScript-моделі [20]. Це дозволяє зробити роботу з даними більш структурованою та зручною в реалізації.

Додатковим аргументом на користь вибору MongoDB є її поширеність і популярність у професійному середовищі. Згідно з результатами Stack Overflow

Developer Survey 2025, MongoDB входить до числа найпоширеніших баз даних, які активно використовуються розробниками у професійній діяльності [21].

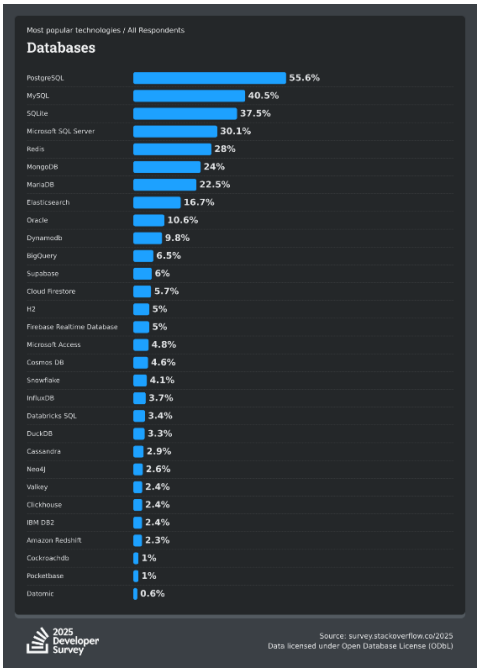


Рисунок 2.5 – Популярність баз даних за результатами Stack Overflow Developer Survey 2025

Крім того, за рейтингом TOPDB Index MongoDB входить до десятки найпопулярніших баз даних світу, що підтверджує її актуальність та широке застосування у реальних програмних проєктах [22].

Worldwide, Apr 2026 :

Rank	Change	Database	Share	1-year trend
1		Oracle	35.15 %	+1.3 %
2		MySQL	11.73 %	-3.0 %
3	↑	PostgreSQL	9.53 %	+2.3 %
4	↓	SQL Server	8.88 %	-2.0 %
5	↑↑↑↑	Supabase	6.47 %	+4.1 %
6		MongoDB	5.6 %	-0.2 %
7		Redis	3.97 %	+0.2 %
8	↓↓↓	Microsoft Access	3.77 %	-2.2 %
9	↑	SQLite	2.66 %	+0.4 %
10	↓↓	Splunk	2.31 %	-0.3 %

Рисунок 2.6 – Порівняння популярності баз даних за TOPDB Index

2.2.3 Express.js для серверного API

Наступним компонентом MERN-стеку є Express.js — фреймворк для Node.js, який використовується для створення серверної частини вебзастосунку та організації API. У розроблюваному інтернет-магазині рослин Express.js відповідає за обробку запитів від клієнтської частини, маршрутизацію, роботу з даними та передачу відповідей назад до інтерфейсу користувача [17].

Express.js є доцільним вибором для даного проєкту, оскільки він дає змогу швидко створювати серверні маршрути для різних частин системи. Наприклад, окремі маршрути можуть відповідати за реєстрацію та вхід користувачів, отримання списку товарів, додавання рослин до кошика, оформлення замовлення, застосування купонів та роботу адміністративної панелі.

У межах вебзастосунку серверне API забезпечує зв’язок між клієнтською частиною, базою даних MongoDB та зовнішніми сервісами. Коли користувач виконує дію в інтерфейсі, наприклад, переглядає каталог або оформлює покупку, клієнтська частина надсилає запит на сервер. Express.js приймає цей запит, виконує необхідну логіку, звертається до бази даних або платіжної системи та повертає результат.

Важливою перевагою Express.js є підтримка REST API, що дозволяє зручно реалізовувати CRUD-операції: створення, читання, оновлення та видалення даних [23]. Це особливо важливо для інтернет-магазину, де необхідно керувати товарами, категоріями, користувачами, замовленнями та купонами.

Також Express.js добре інтегрується з middleware — проміжними функціями, які можуть виконувати перевірку токенів, обробку помилок, валідацію даних або захист окремих маршрутів [24]. Завдяки цьому можна реалізувати безпечну автентифікацію користувачів і розмежування доступу між звичайними користувачами та адміністраторами.

Express.js було обрано для реалізації серверного API через його простоту, гнучкість, зручну маршрутизацію та хорошу інтеграцію з іншими компонентами MERN-стеку. Його використання дозволяє ефективно організувати серверну логіку інтернет-магазину рослин і забезпечити стабільну взаємодію між клієнтською частиною, базою даних та зовнішніми сервісами.

2.2.4 React для клієнтської частини

Для реалізації клієнтської частини вебзастосунку обрано React.js — популярну JavaScript-бібліотеку для створення користувацьких інтерфейсів [15]. Вона використовується для побудови динамічних і зручних вебінтерфейсів, що є особливо важливим для інтернет-магазину.

React базується на компонентному підході, що дозволяє розділяти інтерфейс на незалежні частини — компоненти [25]. Кожен компонент відповідає за окремий елемент інтерфейсу, наприклад, картку товару, список рослин, кошик або форму оформлення замовлення. Такий підхід спрощує розробку, повторне використання коду та подальшу підтримку застосунку.

У розроблюваному вебзастосунку React використовується для створення основних сторінок вебзастосунку, зокрема каталогу товарів, сторінки окремого продукту, кошика, оформлення замовлення та адміністративної панелі. Завдяки React інтерфейс оновлюється динамічно без повного перезавантаження сторінки, що забезпечує швидку та зручну взаємодію користувача із системою.

Однією з ключових переваг React є використання віртуального DOM, що дозволяє ефективно оновлювати лише ті частини сторінки, які змінилися [26]. Це позитивно впливає на продуктивність застосунку та покращує користувацький досвід, особливо при роботі з великою кількістю товарів.

Також React добре інтегрується з серверною частиною через API, реалізоване на Express.js. Дані передаються у форматі JSON, що забезпечує швидкий обмін інформацією між клієнтом і сервером.

Додатковою перевагою React є наявність великої кількості бібліотек та інструментів, які розширюють його можливості. У даному проєкті використовується Tailwind CSS для створення сучасного та адаптивного дизайну, що дозволяє забезпечити зручність використання вебзастосунку на різних пристроях.

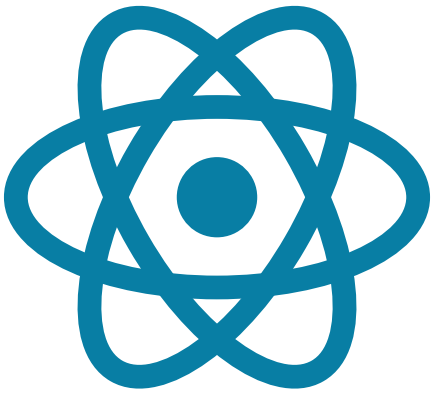


Рисунок 2.7 – Логотип бібліотеки React

Використання React.js для клієнтської частини обумовлене його гнучкістю, високою продуктивністю, компонентною структурою та можливістю створення сучасного інтерактивного інтерфейсу. Це дозволяє забезпечити зручний користувацький досвід та ефективну взаємодію з вебзастосунком інтернет-магазину рослин.

2.2.5 Node.js як серверне середовище

Для реалізації серверної частини вебзастосунку використовується Node.js — середовище виконання JavaScript, яке дозволяє запускати серверний код поза

межами браузера. Node.js широко застосовується для створення вебзастосунків завдяки своїй продуктивності, масштабованості та можливості працювати з великою кількістю одночасних запитів [16].

Однією з ключових особливостей Node.js є асинхронна модель обробки запитів, яка базується на неблокуючому вводу/виводу [27]. Це означає, що сервер може одночасно обробляти багато запитів без затримок, що є важливим для інтернет-магазину, де користувачі одночасно переглядають товари, додають їх у кошик та оформлюють замовлення.

У межах розроблюваного вебзастосунку Node.js використовується як основа серверної логіки. Він забезпечує виконання API, створеного за допомогою Express.js, обробку запитів від клієнтської частини, взаємодію з базою даних MongoDB та інтеграцію з додатковими сервісами, такими як платіжні системи або сервіси зберігання зображень.

Важливою перевагою Node.js є використання JavaScript як на клієнтській, так і на серверній стороні. Це дозволяє уніфікувати процес розробки, спростити передачу даних у форматі JSON та зменшити складність взаємодії між різними частинами системи.

Node.js також має велику екосистему бібліотек і пакетів, доступних через менеджер пакетів npm. Це значно пришвидшує розробку, оскільки дозволяє використовувати готові рішення для реалізації автентифікації, роботи з базою даних, обробки платежів та інших функцій.

У поєднанні з Express.js Node.js дозволяє створити ефективний серверний рівень застосунку, який відповідає за обробку бізнес-логіки, управління даними та забезпечення безпечної взаємодії з клієнтською частиною.

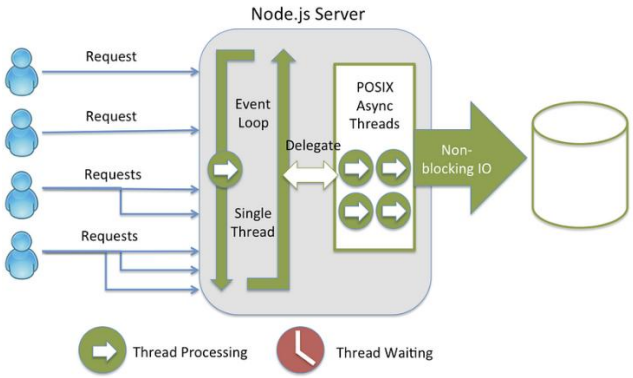


Рисунок 2.8 – Архітектура обробки запитів у Node.js

Отже, вибір Node.js як серверного середовища обумовлений його продуктивністю, підтримкою асинхронної обробки запитів, широкими можливостями розширення та природною інтеграцією з іншими компонентами MERN-стеку, що робить його оптимальним рішенням для реалізації вебзастосунку інтернет-магазину рослин.

2.3 Безпека та автентифікація - JWT access та refresh tokens

Забезпечення безпеки є важливою складовою розробки вебзастосунків, особливо у сфері електронної комерції, де обробляються персональні дані користувачів та фінансова інформація. У розроблюваному інтернет-магазині рослин реалізовано сучасні підходи до автентифікації та захисту даних, що дозволяє забезпечити надійний контроль доступу до системи.

Основною задачею автентифікації є перевірка особи користувача та надання доступу до функціоналу системи відповідно до його прав. У даному вебзастосунку використовується токен-орієнтований підхід до автентифікації, який базується на використанні JSON Web Tokens (JWT) [28].

У розроблюваному вебзастосунку для реалізації автентифікації використовується технологія JSON Web Token (JWT). JWT — це відкритий

стандарт (RFC 7519), який визначає компактний і самодостатній спосіб безпечної передачі інформації між клієнтом і сервером у вигляді JSON-об'єкта.

Ідея JWT полягає в тому, що сервер після успішної автентифікації користувача створює токен, який містить певні дані і підписується за допомогою секретного ключа або криптографічної пари ключів. Завдяки цьому сервер може перевірити цілісність токена та впевнитися, що він не був змінений [29].

JWT складається з трьох частин, розділених крапками: заголовка (Header), корисного навантаження (Payload) та підпису (Signature). Заголовок містить інформацію про тип токена (JWT) і алгоритм підпису, наприклад HS256 або RSA [30]. Корисне навантаження включає дані про користувача, такі як його ідентифікатор, роль або час дії токена. Підпис формується на основі заголовка, payload та секретного ключа і використовується для перевірки цілісності токена та захисту від підробки.

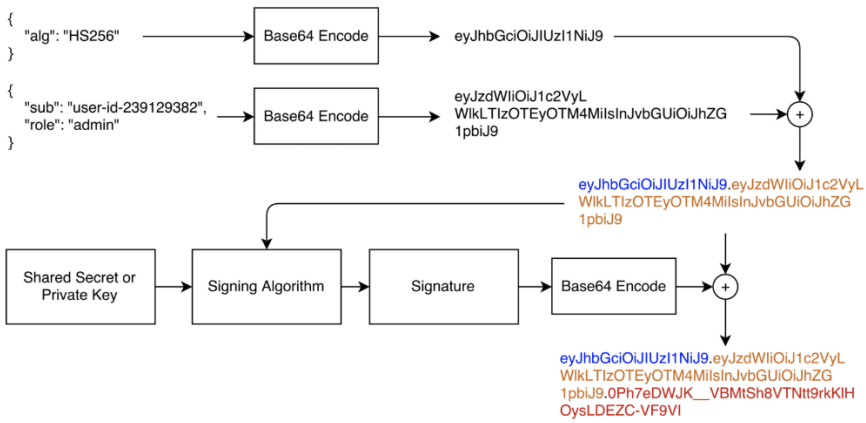


Рисунок 2.9 – Схема формування JWT-токена

Такий формат дозволяє зручно передавати токен у HTTP-запитах, зазвичай через заголовок Authorization у вигляді: Bearer <token>. У системі використовуються два типи токенів: access token та refresh token.

Access token є короткочасним і використовується для доступу до захищених ресурсів [31]. Він передається разом із кожним запитом до

серверного API та перевіряється на валідність. Обмежений час дії підвищує рівень безпеки, оскільки навіть у разі компрометації токен швидко стає недійсним.

Refresh token має довший термін дії та використовується для отримання нового access token без повторної авторизації користувача [32]. Це дозволяє забезпечити баланс між безпекою та зручністю використання системи.

Процес роботи виглядає наступним чином: користувач вводить свої облікові дані, після чого сервер перевіряє їх і генерує пару токенів — access та refresh. Надалі клієнт використовує access token для доступу до API. Після завершення строку його дії клієнт звертається до сервера з refresh token для отримання нового access token.

JWT дозволяє реалізувати так звану stateless-автентифікацію, тобто сервер не зберігає інформацію про сесії користувачів [33]. Усі необхідні дані містяться безпосередньо в токені, що спрощує масштабування системи та зменшує навантаження на сервер.

Важливою особливістю є те, що дані в payload не шифруються, а лише кодуються, тому вони можуть бути прочитані. З цієї причини не рекомендується зберігати у JWT конфіденційну інформацію, таку як паролі або платіжні дані.

У процесі роботи з JWT виділяють два основні етапи: валідацію — перевірку структури токена, терміну його дії та коректності даних, а також верифікацію — перевірку підпису, яка гарантує, що токен не був змінений.

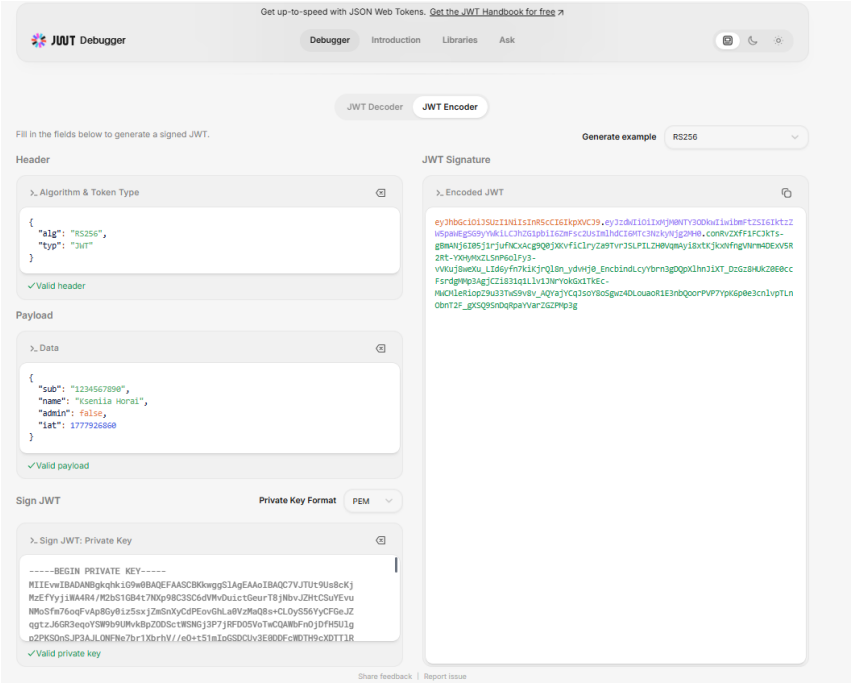


Рисунок 2.10 – Приклад структури JWT у JWT Debugger

Отже, використання JWT у поєднанні з access та refresh токенами забезпечує надійний механізм автентифікації, який добре підходить для сучасних вебзастосунків і дозволяє підвищити безпеку та ефективність роботи інтернет-магазину рослин.

2.4 Додаткові технології та сервіси

2.4.1 Redis для кешування

У сучасних вебзастосунках важливу роль відіграє оптимізація продуктивності, особливо при роботі з великою кількістю запитів. Для підвищення швидкодії у даному проєкті використовується Redis — високопродуктивне in-memory сховище даних, яке застосовується для кешування [34].

Redis зберігає дані у оперативній пам’яті, що дозволяє значно швидше отримувати інформацію порівняно з традиційними базами даних. У розроблюваному вебзастосунку Redis використовується для тимчасового зберігання часто запитуваних даних, що дозволяє зменшити кількість звернень до MongoDB та скоротити час відповіді сервера.

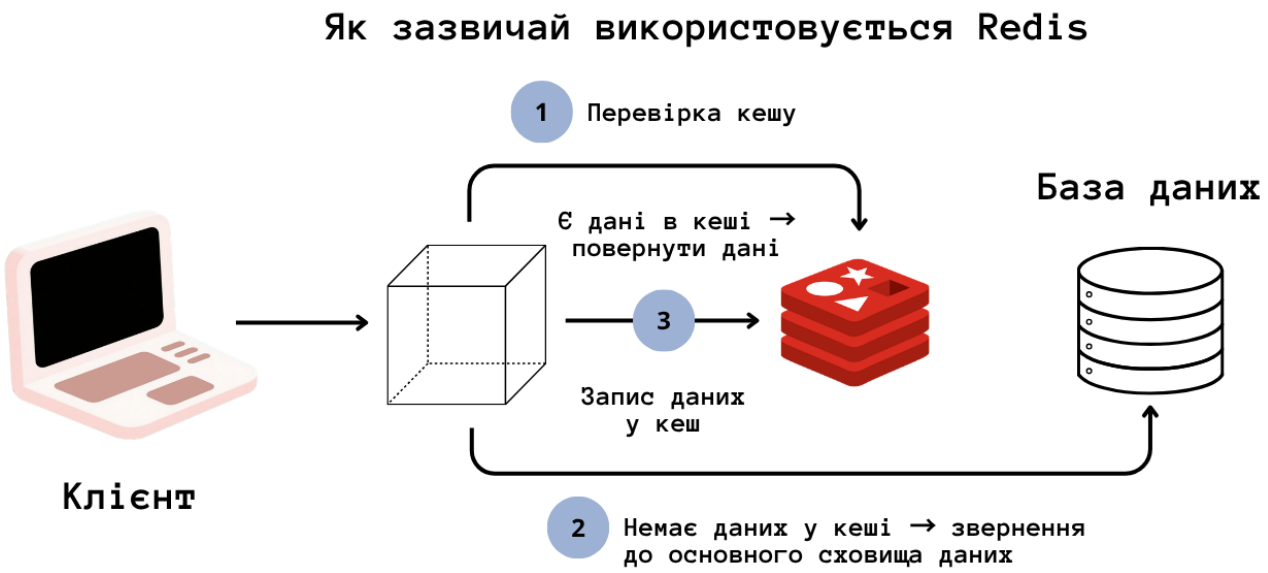


Рисунок 2.11 – Принцип роботи кешування з використанням Redis

Зокрема, кешування може застосовуватися для збереження списків популярних товарів, результатів пошуку або інших даних, які часто використовуються користувачами. При повторному запиті таких даних система звертається до Redis замість бази даних, що значно підвищує ефективність роботи застосунку.

Використання Redis також дозволяє краще масштабувати систему, оскільки зменшується навантаження на основну базу даних. Це особливо важливо для e-commerce застосунків, де кількість користувачів і запитів може зростати.

2.4.2 Cloudinary для зберігання зображень

У вебзастосунках електронної комерції важливу роль відіграє ефективне зберігання та обробка зображень товарів. Для вирішення цієї задачі у даному проєкті використовується сервіс Cloudinary — хмарна платформа для роботи з медіафайлами [35].

Cloudinary дозволяє зберігати зображення у віддаленому сховищі та отримувати до них доступ через URL-посилання. Це дає змогу зменшити навантаження на сервер застосунку, оскільки медіафайли не зберігаються локально, а обробляються зовнішнім сервісом.

Однією з ключових переваг Cloudinary є автоматична оптимізація зображень. Сервіс може змінювати розмір, формат і якість зображень залежно від пристрою користувача та умов мережі. Це дозволяє зменшити час завантаження сторінок і покращити загальний користувацький досвід.

Рисунок 2.12 – Логотип сервісу Cloudinary

У межах розроблюваного вебзастосунку Cloudinary використовується для зберігання фотографій рослин, що відображаються у каталозі товарів та на сторінках окремих продуктів. Під час додавання нового товару адміністратор може завантажити зображення, яке автоматично обробляється та зберігається у хмарному сховищі.

Cloudinary також забезпечує можливість швидкого отримання зображень через CDN (Content Delivery Network), що підвищує швидкість доступу до медіафайлів незалежно від географічного розташування користувача [36].

Використання Cloudinary дозволяє ефективно організувати зберігання та обробку зображень у вебзастосунку, зменшити навантаження на сервер, покращити продуктивність та забезпечити високу якість відображення товарів інтернет-магазину рослин.

2.4.3 Tailwind CSS для стилізації інтерфейсу

Для реалізації сучасного та зручного користувацького інтерфейсу у вебзастосунку використовується утилітарний CSS-фреймворк Tailwind CSS [37]. Він дозволяє швидко створювати стилі без написання великої кількості власного CSS-коду, використовуючи готові класи безпосередньо в HTML-розмітці.

Основною особливістю Tailwind CSS є підхід utility-first, при якому кожен клас відповідає за одну конкретну властивість стилю, наприклад відступи, кольори, розміри або позиціонування. Це дозволяє гнучко керувати зовнішнім виглядом елементів і швидко змінювати дизайн без редагування окремих стилів у файлах CSS.

У розроблюваному вебзастосунку Tailwind CSS використовується для оформлення сторінок каталогу товарів, карток рослин, кошика, форм авторизації та адміністративної панелі. Завдяки цьому забезпечується єдиний стиль оформлення, зручна навігація та привабливий зовнішній вигляд інтерфейсу.

Ще однією важливою перевагою Tailwind CSS є підтримка адаптивного дизайну [38]. За допомогою вбудованих інструментів можна легко налаштувати відображення сторінок для різних типів пристроїв — від мобільних телефонів

до персональних комп’ютерів. Це є важливим для вебзастосунків, оскільки користувачі можуть здійснювати покупки з різних пристроїв.

Також Tailwind CSS дозволяє уникнути надлишковості стилів, оскільки використовуються лише ті класи, які реально застосовуються у проєкті. Це сприяє зменшенню обсягу фінального CSS-файлу та покращує продуктивність застосунку.

Tailwind CSS є доцільним рішенням для даного вебзастосунку, оскільки він забезпечує швидку розробку інтерфейсу, гнучкість налаштування дизайну, адаптивність та високу продуктивність.

ВИСНОВОК ДО РОЗДІЛУ 2

У даному розділі було розглянуто основні технології та підходи, що використовуються для розробки вебзастосунку електронної комерції для продажу рослин. Обґрунтовано вибір клієнт-серверної архітектури, яка дозволяє ефективно розділити функції між клієнтською та серверною частинами, забезпечити централізоване зберігання даних і підвищити безпеку системи.

Було визначено технологічний стек застосунку та обґрунтовано доцільність використання MERN-стеку, який поєднує MongoDB, Express.js, React.js та Node.js. Такий підхід дозволяє використовувати єдину мову програмування на всіх рівнях системи, спрощує розробку, інтеграцію компонентів та подальшу підтримку програмного продукту.

Детально розглянуто роль кожного елемента стеку: MongoDB забезпечує гнучке зберігання даних, Express.js — реалізацію серверного API, React — створення динамічного користувацького інтерфейсу, а Node.js — ефективне виконання серверної логіки.

Окрему увагу приділено питанням безпеки, зокрема реалізації автентифікації за допомогою JWT-токенів. Використання access та refresh токенів дозволяє забезпечити надійний контроль доступу до системи та зручну взаємодію користувача із застосунком.

Також розглянуто додаткові технології та сервіси, які підвищують функціональність і продуктивність системи: Redis для кешування даних, Cloudinary для зберігання зображень та Tailwind CSS для створення сучасного адаптивного інтерфейсу.

Таким чином, обрані архітектурні рішення та технології є доцільними для реалізації вебзастосунку електронної комерції для продажу рослин, оскільки

вони забезпечують високу продуктивність, масштабованість, безпеку та зручність використання системи.

РОЗДІЛ 3. РОЗРОБКА ОСНОВНОГО ФУНКЦІОНАЛУ ВЕБЗАСТОСУНКУ

3.1 Аналіз вимог до системи

Перед початком розробки вебзастосунку було проведено аналіз вимог, який дозволив визначити необхідний функціонал системи, а також якісні характеристики, яким вона має відповідати. Вимоги поділяються на функціональні, що описують конкретні можливості застосунку, та нефункціональні, які визначають критерії якості, безпеки та продуктивності.

3.1.1 Загальні вимоги до функціоналу

Вебзастосунок електронної комерції для продажу рослин повинен реалізовувати такі основні можливості:

- реєстрація та автентифікація користувача з розмежуванням ролей покупця та адміністратора;
- перегляд головної сторінки з категоріями товарів та підбіркою рекомендованих рослин;
- перегляд каталогу товарів у межах обраної категорії;
- перегляд рекомендацій на сторінці кошика;
- додавання товарів до кошика та керування його вмістом;
- оформлення замовлення з онлайн-оплатою через платіжний сервіс Stripe;
- застосування купонів на знижку при оформленні замовлення;
- автоматичне нарахування купону користувачу при здійсненні покупки на певну суму;

- керування каталогом товарів адміністратором, зокрема додавання, видалення та позначення товарів як рекомендованих;
- перегляд аналітики продажів адміністратором із графіком динаміки.

3.1.2 Функціональні вимоги

Функціональні вимоги описують конкретні функції та можливості, які має реалізовувати вебзастосунок [39].

1. Система автентифікації та авторизації. Застосунок повинен забезпечувати реєстрацію нових користувачів із перевіркою унікальності електронної пошти, вхід до системи за допомогою електронної пошти та пароля, захищений вихід із системи з видаленням токенів, розмежування прав доступу між звичайними користувачами та адміністраторами, а також автоматичне оновлення access токена за допомогою refresh токена.
2. Каталог товарів. Застосунок повинен надавати можливість перегляду товарів за категоріями, відображення рекомендованих товарів на головній сторінці, перегляду списку випадкових рекомендацій на сторінці кошика, а також отримання повного списку товарів адміністратором.
3. Кошик покупок. Застосунок повинен реалізовувати додавання товарів до кошика, зміну кількості одиниць кожного товару, видалення окремих товарів або повне очищення кошика, а також автоматичний перерахунок загальної суми замовлення.
4. Платіжна система. Застосунок повинен забезпечувати створення платіжної сесії через Stripe Checkout.
5. Система купонів. Застосунок повинен підтримувати перевірку валідності купонів при введенні коду користувачем, застосування знижки до загальної суми замовлення, автоматичне деактивування використаного

купону, а також автоматичне нарахування нового купону користувачам, чия сума покупки перевищує встановлений поріг.

6. Адміністративна панель. Застосунок повинен надавати адміністратору можливість додавати нові товари з назвою, описом, ціною, категорією та зображенням, видаляти існуючі товари, позначати товари як рекомендовані або знімати це позначення, а також переглядати аналітику продажів із графіком динаміки.

3.1.3 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи, які не стосуються безпосередньо функцій, але впливають на ефективність і надійність її роботи [40].

1. Безпека. Паролі користувачів повинні зберігатися у захищеному вигляді з використанням алгоритму bcrypt. Автентифікація реалізується на основі JWT з окремими access та refresh токенами. Захищені маршрути мають бути недоступні без валідного токена. Адміністративні функції повинні бути доступні виключно для користувачів із роллю адміністратора.
2. Продуктивність. Часто запитувані дані, зокрема список рекомендованих товарів, повинні кешуватися за допомогою Redis для зменшення навантаження на базу даних. Зображення товарів мають зберігатися у хмарному сховищі Cloudinary та доставлятися через CDN для прискорення завантаження.
3. Зручність використання. Інтерфейс застосунку повинен бути адаптивним та коректно відображатися на пристроях різних розмірів. Навігація між сторінками має здійснюватися без повного перезавантаження сторінки завдяки використанню React Router. Усі ключові дії користувача повинні супроводжуватися відповідними повідомленнями про успіх або помилку.

4. Масштабованість. Архітектура застосунку повинна забезпечувати можливість розширення функціоналу без значних змін у наявній кодовій базі. Серверна частина має бути побудована з чітким розділенням маршрутів, контролерів і моделей даних.
5. Надійність. Система повинна коректно обробляти помилки на рівні сервера та повертати відповідні HTTP-статуси і повідомлення. У разі невдалої спроби оновлення токена система має автоматично виконувати вихід користувача з системи.

3.2 Проєктування бази даних

Основою зберігання даних у розробленому застосунку є MongoDB — документоорієнтована база даних, з якою взаємодія здійснюється через бібліотеку Mongoose. Було спроєктовано чотири моделі: User для зберігання інформації про користувачів, Product для опису товарів магазину, Order для фіксації оформлених замовлень та Coupon для управління купонами на знижку. Ці моделі сформували основу для організації даних про користувачів, товари, замовлення та знижки у вебзастосунку.

Модель User зберігає інформацію про зареєстрованих користувачів системи та містить поля імені, електронної пошти, захищеного пароля, ролі у системі та вмісту кошика. Роль може набувати значень "user" або "admin", що дозволяє розмежовувати доступ між покупцями та адміністраторами. Поле cartContents є масивом об'єктів, кожен із яких містить посилання на товар та кількість одиниць. Перед збереженням пароль автоматично хешується через bcryptjs у Mongoose pre-save хуку, а метод comparePassword використовується під час автентифікації.

Рисунок 3.1 – Програмна реалізація моделі User

Модель Product описує товари магазину рослин. Документ продукту містить назву, опис, ціну, посилання на зображення у Cloudinary, категорію та поле isFeatured, яке визначає чи є товар рекомендованим.

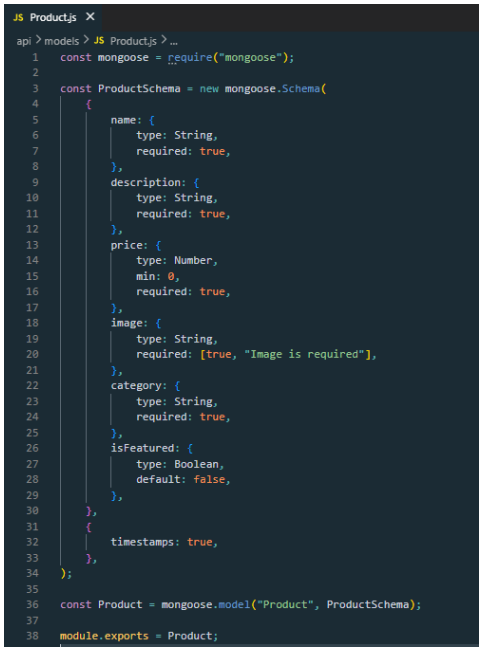


Рисунок 3.2 – Програмна реалізація моделі Product

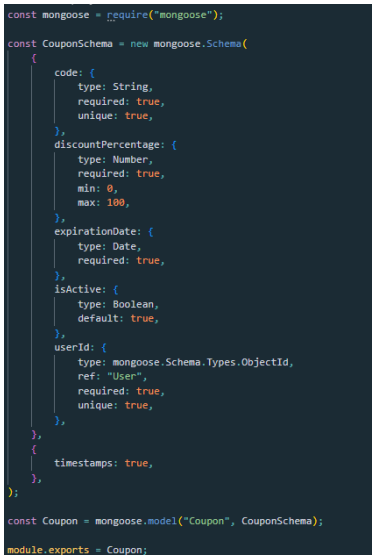
Модель Order фіксує успішно оформлені замовлення після підтвердження оплати. Кожен документ містить посилання на користувача, масив придбаних

товарів із зазначенням кількості та ціни на момент покупки, загальну суму та унікальний ідентифікатор платіжної сесії Stripe.

```
const mongoose = require("mongoose");

const OrderSchema = new mongoose.Schema(
    {
        user: {
            type: mongoose.Schema.Types.ObjectId,
            ref: "User",
            required: true,
        },
        products: [
            {
                product: {
                    type: mongoose.Schema.Types.ObjectId,
                    ref: "Product",
                    required: true,
                },
                quantity: {
                    type: Number,
                    required: true,
                    min: 1,
                },
                price: {
                    type: Number,
                    required: true,
                    min: 0,
                },
            },
        ],
        totalAmount: {
            type: Number,
            required: true,
            min: 0,
        },
        stripeSessionId: {
            type: String,
            unique: true,
        },
    },
    { timestamps: true },
);

const Order = mongoose.model("Order", OrderSchema);

module.exports = Order;
```

Рисунок 3.3 – Програмна реалізація моделі Order

Модель Coupon зберігає купони на знижку. Кожен купон має унікальний код, відсоток знижки, дату закінчення дії, статус активності та посилання на конкретного користувача через поле userId. Поля code та userId визначені як унікальні, що гарантує наявність не більше одного активного купону на одного користувача.



```
const mongoose = require("mongoose");

const CouponSchema = new mongoose.Schema(
    {
        code: {
            type: String,
            required: true,
            unique: true,
        },
        discountPercentage: {
            type: Number,
            required: true,
            min: 0,
            max: 100,
        },
        expirationDate: {
            type: Date,
            required: true,
        },
        isActive: {
            type: Boolean,
            default: true,
        },
        userId: {
            type: mongoose.Schema.Types.ObjectId,
            ref: "User",
            required: true,
            unique: true,
        },
    },
    {
        timestamps: true,
    },
);

const Coupon = mongoose.model("Coupon", CouponSchema);

module.exports = Coupon;
```

Рисунок 3.4 – Програмна реалізація моделі Coupon

Зв'язки між колекціями реалізовано через посилання типу ObjectId. User через поле cartContents посилається на документи Product. Order містить одночасно посилання на User та масив документів Product. Coupon пов'язаний із конкретним користувачем через поле userId. Загальну схему колекцій та зв'язків між ними наведено на рисунку 3.5.

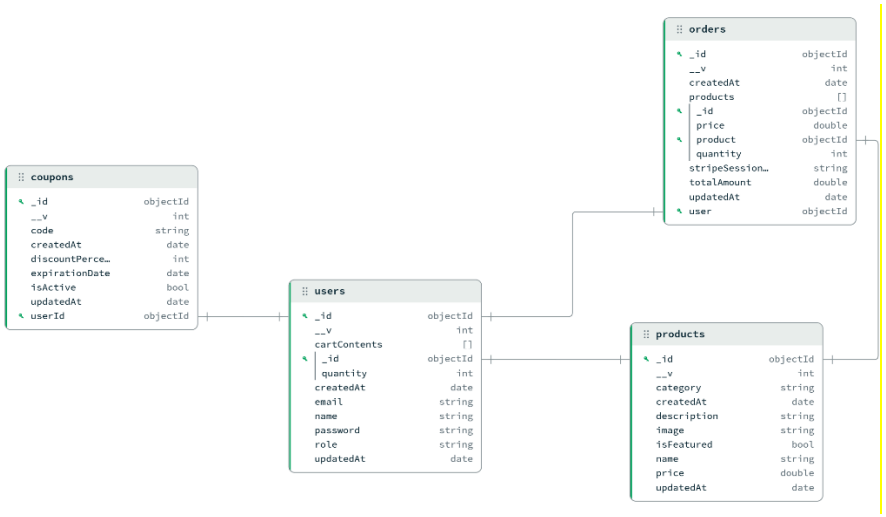


Рисунок 3.5 – Схема колекцій бази даних вебзастосунку

3.3 Розробка серверної частини

3.3.1 Структура серверного додатку

В основі серверної частини лежить Node.js у поєднанні з Express.js — саме цей фреймворк забезпечує маршрутизацію, обробку запитів та підключення проміжного програмного забезпечення. Головний файл app.js є точкою входу в застосунок: тут реєструються всі маршрути, підключаються необхідні middleware та після старту сервера ініціюється з'єднання з базою даних через функцію setupDatabase.

З-поміж підключених middleware варто виділити два ключових: express.json із встановленим обмеженням на розмір тіла запиту у 10 МБ — це потрібно для коректної передачі зображень товарів у форматі base64 — та

cookieParser, який забезпечує зчитування cookie-файлів, де зберігаються токени автентифікації.

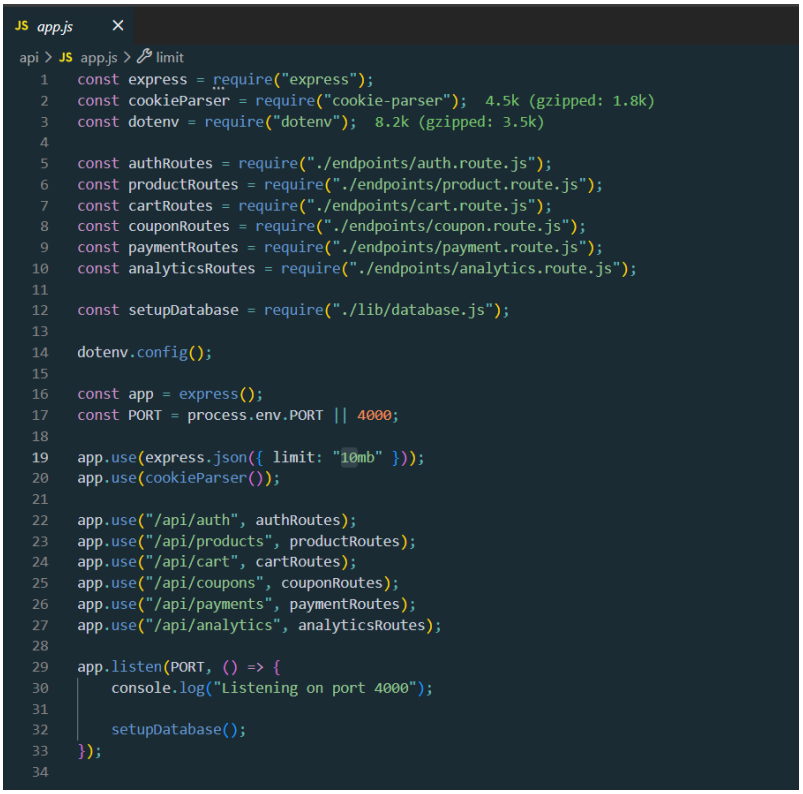


Рисунок 3.6 – Програмна реалізація файлу app.js

Код організовано за принципом розділення відповідальності: кожен функціональний модуль системи має власний роутер і контролер. Загалом у застосунку визначено шість модулів — автентифікація, товари, кошик, купони, платежі та аналітика, кожен із яких підключається до окремого префіксу API. Роутери відповідають виключно за визначення HTTP-методів, шляхів і прив'язку middleware, тоді як уся логіка обробки запитів зосереджена в контролерах. Такий поділ суттєво полегшує орієнтування в коді та його подальший розвиток.

Окремої уваги заслуговує організація захисту маршрутів. Частина ендпоінтів доступна лише авторизованим користувачам. Для цього перед викликом контролера спрацьовує middleware перевірки токена. Маршрути

адміністративної панелі мають додатковий рівень захисту у вигляді перевірки ролі, що унеможливлює доступ до них для звичайних користувачів.

Усі модулі для роботи із зовнішніми сервісами, такими як MongoDB, Redis, Cloudinary та Stripe, винесено до окремої директорії lib. Це дозволяє тримати логіку підключення ізольованою від бізнес-логіки контролерів і за потреби змінювати налаштування інтеграцій, не торкаючись решти коду.

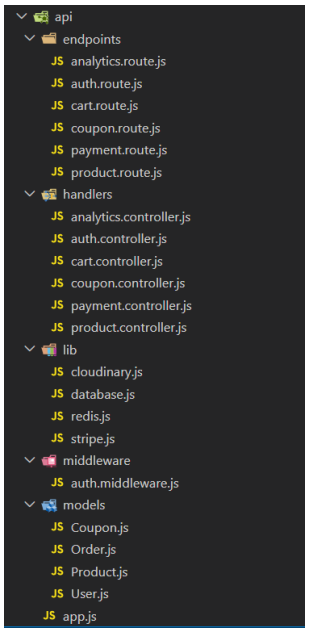


Рисунок 3.7 – Структура директорій серверної частини

3.3.2 Реалізація автентифікації та авторизації

У застосунку передбачено два рівні доступу — звичайний користувач та адміністратор. Звичайний користувач має доступ до перегляду каталогу, управління кошиком та оформлення замовлень. Адміністратор додатково отримує доступ до керування товарами та аналітики продажів. Перевірка прав відбувається на рівні серверних маршрутів ще до того, як запит потрапляє до контролера.

Механізм автентифікації побудований на основі JWT із двома типами токенів. Короткочасний access-токен діє 15 хвилин та використовується для

доступу до захищених ресурсів. Довготривалий refresh-токен зберігається 10 днів і слугує для отримання нового access-токен без повторного входу.

Обидва токени передаються клієнту не у тілі відповіді, а через httpOnly cookie.

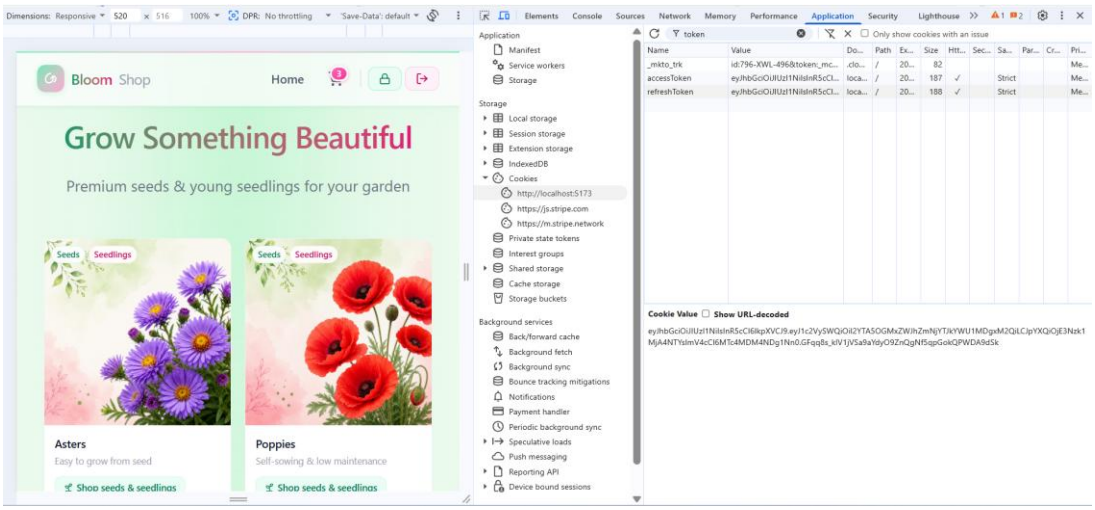


Рисунок 3.8 – Збереження access та refresh токенів у cookie браузера

Це принципово з точки зору безпеки: такі cookie недоступні зі скриптів на сторінці, що унеможливлює їх викрадення через XSS-атаки. Атрибут sameSite зі значенням strict додатково захищає від міжсайтових CSRF-атак.

```
const getMyTokens = function (userId) {
    const accessToken = jwt.sign({ userId }, process.env.SECRET_ACCESS_TOKEN, {
        expiresIn: "15m",
    });
    const refreshToken = jwt.sign(
        { userId },
        process.env.SECRET_REFRESH_TOKEN,
        {
            expiresIn: "10d",
        },
    );
    return { accessToken, refreshToken };
};

const saveRefreshToken = async (userId, refreshToken) => {
    await redis.set(
        `refresh_token:${userId}`,
        refreshToken,
        "EX",
        10 * 24 * 60 * 60,
    );
};

const setCookies = (res, accessToken, refreshToken) => {
    res.cookie("accessToken", accessToken, {
        httpOnly: true, //against XSS hacking
        secure: process.env.NODE_ENV === "production",
        sameSite: "strict", //against CSRF attacks
        maxAge: 10 * 60 * 1000,
    });

    res.cookie("refreshToken", refreshToken, {
        httpOnly: true, //against XSS hacking
        secure: process.env.NODE_ENV === "production",
        sameSite: "strict", //against CSRF attacks
        maxAge: 10 * 24 * 60 * 60 * 1000,
    });
};
```

Рисунок 3.9 – Програмна реалізація функцій генерації токенів та встановлення cookie

Під час реєстрації сервер першочергово перевіряє, чи не існує вже облікового запису з вказаною електронною поштою. Якщо такий знайдено — повертається помилка. Інакше створюється новий запис, генеруються токени і відразу встановлюються cookie. У відповідь клієнту повертаються лише безпечні дані — ідентифікатор, ім'я, імейл та роль. Пароль, навіть у хешованому вигляді, у відповідь не включається.

```
const signup = async (req, res) => {
    const { email, password, name } = req.body;
    try {
        const userIsInDB = await User.findOne({ email });

        if (userIsInDB) {
            return res.status(400).json({ message: "User already exists" });
        }
        const newUser = await User.create({ name, email, password });

        //AUTHENTICATION
        const { accessToken, refreshToken } = getMyTokens(newUser._id);
        await saveRefreshToken(newUser._id, refreshToken);

        setCookies(res, accessToken, refreshToken);

        res.status(201).json({
            newUser: {
                _id: newUser._id,
                name: newUser.name,
                email: newUser.email,
                role: newUser.role,
            },
            message: "User created successfully",
        });
    } catch (error) {
        console.log("Signup error occured", error.message);
        res.status(500).json({ message: error.message });
    }
};
```

Рисунок 3.10 – Програмна реалізація функції signup

Оновлення access-токену відбувається через окремий ендпоінт. Сервер зчитує refresh-токен із cookie та звіряє його зі значенням, яке зберігається у Redis для конкретного користувача. Якщо значення збігаються — видається новий access-токен. Якщо ж після виходу з системи refresh-токен було видалено з Redis, повторне його використання стає неможливим, що унеможливлює зловживання вкраденим токеном.

```
const refreshToken = async (req, res) => {
    try {
        const refreshToken = req.cookies.refreshToken;

        if (!refreshToken) {
            return res
                .status(401)
                .json({ message: "Refresh token is not found" });
        }

        const decoded = jwt.verify(
            refreshToken,
            process.env.SECRET_REFRESH_TOKEN,
        );
        const storedToken = await redis.get(`refresh_token:${decoded.userId}`);

        if (storedToken !== refreshToken) {
            return res.status(401).json({ message: "Invalid refresh token" });
        }

        const accessToken = jwt.sign(
            { userId: decoded.userId },
            process.env.SECRET_ACCESS_TOKEN,
            { expiresIn: "15m" },
        );

        res.cookie("accessToken", accessToken, {
            httpOnly: true, //against XSS hacking
            secure: process.env.NODE_ENV === "production",
            sameSite: "strict", //against CSRF attacks
            maxAge: 10 * 60 * 1000,
        });

        res.json({ message: "Token refreshed successfully" });
    } catch (error) {
        console.log("Error in refreshToken controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};
```

Рисунок 3.11 – Програмна реалізація функції refreshToken

Захист маршрутів забезпечується двома middleware. Перший, verifyRoute, перехоплює кожен захищений запит, витягує access-токен із cookie та декодує його. Якщо токен відсутній або прострочений, запит негайно відхиляється з відповідним статусом. У разі успіху до об'єкта запиту додаються дані поточного користувача, які стають доступними в контролері. Другий middleware, adminOnlyRoute, застосовується поверх першого на маршрутах адміністративної панелі та пропускає запит далі виключно за умови, що роль користувача відповідає значенню "admin".

```
const verifyRoute = async (req, res, next) => {
    try {
        const accessToken = req.cookies.accessToken;

        if (!accessToken) {
            return res
                .status(401)
                .json({ message: "Unauthorized - No access token provided" });
        }

        try {
            const decoded = jwt.verify(
                accessToken,
                process.env.SECRET_ACCESS_TOKEN,
            );
            const user = await User.findById(decoded.userId).select(
                "-password",
            );

            if (!user) {
                return res.status(401).json({ message: "User not found" });
            }

            req.user = user;

            next();
        } catch (error) {
            if (error.name === "TokenExpiredError") {
                return res
                    .status(401)
                    .json({ message: "Unauthorized - Access token expired" });
            }
            throw error;
        }
    } catch (error) {
        console.log("Error in verifyRoute middleware", error.message);
        return res.status(401).json({ message: "Unauthorized" });
    }
};

Windsurf: Refactor | Explain | Generate JSDoc | ×
const adminOnlyRoute = async (req, res, next) => {
    if (req.user && req.user.role === "admin") {
        next();
    } else {
        return res.status(403).json({ message: "Access denied - Admin only" });
    }
};
```

Рисунок 3.12 – Програмна реалізація middleware автентифікації та перевірки ролі

3.3.3 Реалізація API для управління товарами

Модуль управління товарами є одним із центральних у серверній частині застосунку, оскільки саме через нього реалізується основний функціонал інтернет-магазину — перегляд каталогу, фільтрація за категоріями, управління рекомендованими товарами та адміністрування асортименту. Усі маршрути модуля зареєстровані з префіксом /api/products.

Для отримання товарів передбачено кілька ендпоінтів залежно від контексту використання. Запит до /api/products повертає повний список товарів і доступний лише адміністратору. Перегляд товарів за категорією реалізовано через маршрут /api/products/category/:category, де назва категорії передається як параметр URL. Окремий маршрут /api/products/featured повертає лише

рекомендовані товари і використовується на головній сторінці застосунку. Для сторінки кошика реалізовано ендпоінт /api/products/recommendations, який щоразу повертає випадкову вибірку з трьох товарів за допомогою агрегаційного запиту MongoDB.

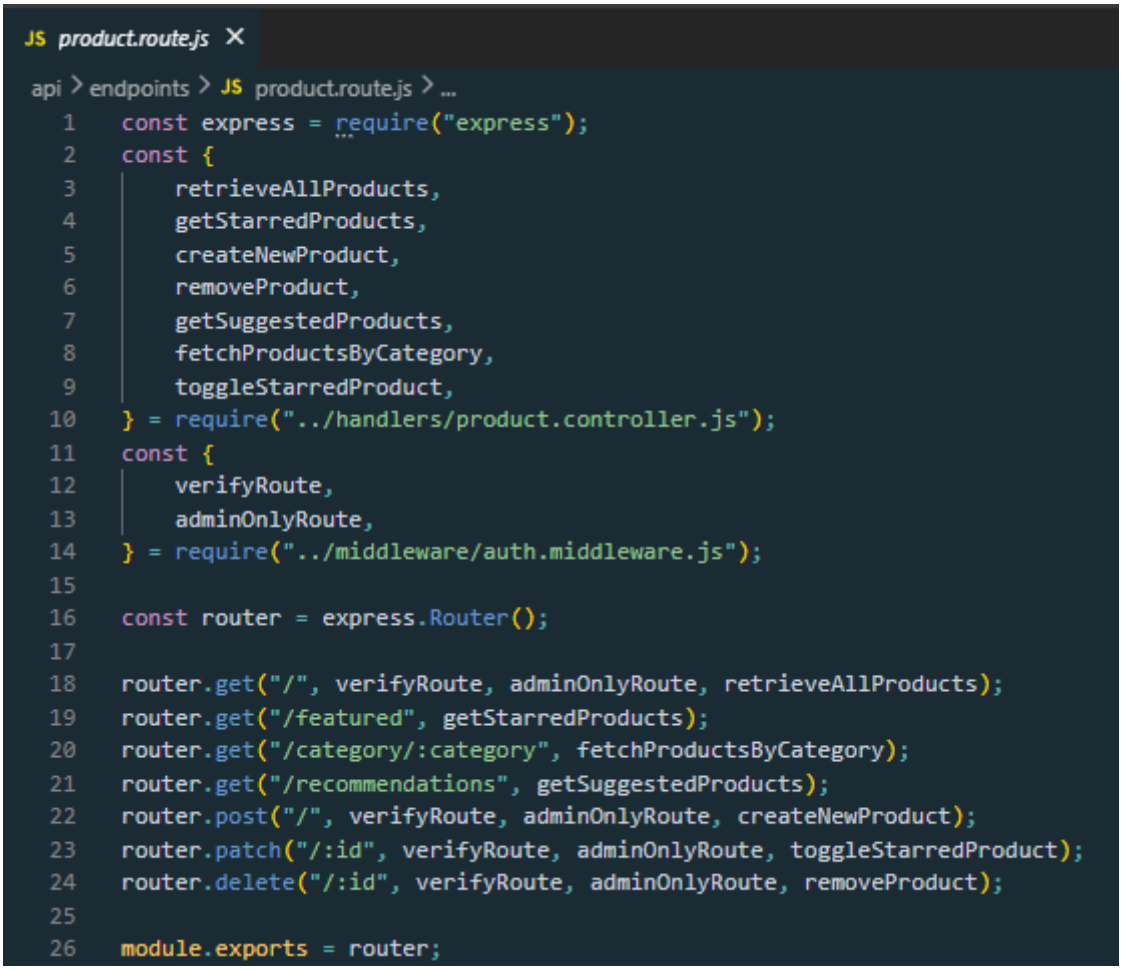


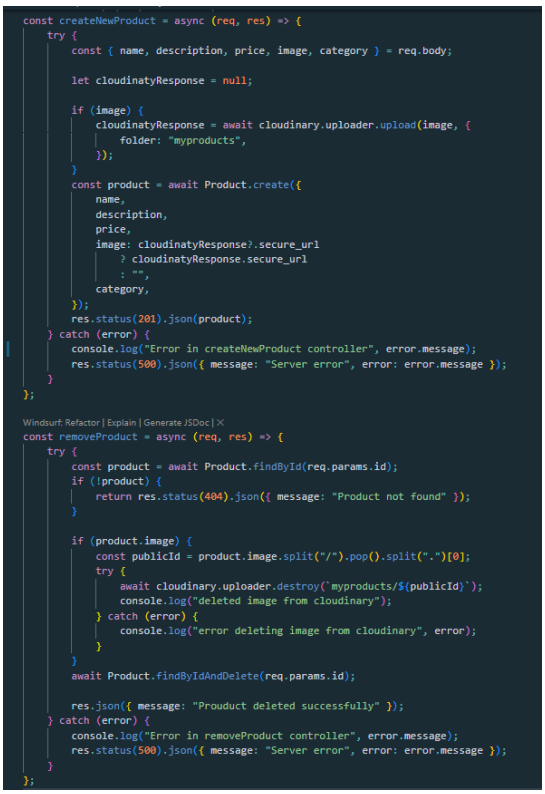
Рисунок 3.13 – Програмна реалізація маршрутів модуля товарів

Отримання рекомендованих товарів реалізовано з кешуванням через Redis. При першому запиті контролер звертається до MongoDB та зберігає результат у Redis. Усі наступні запити отримують дані напряму з кешу, не навантажуючи базу даних. Після зміни статусу будь-якого товару кеш автоматично оновлюється через функцію refreshFeaturedProductsCache, що гарантує актуальність даних.

```
const getStarredProducts = async (req, res) => {
    try {
        let featuredProducts = await redis.get("featured_products");
        if (featuredProducts) {
            return res.json(JSON.parse(featuredProducts));
        }

        //if not in redis, fetch from mongodb
        featuredProducts = await Product.find({ isFeatured: true }).lean();
        if (!featuredProducts) {
            return res
                .status(404)
                .json({ message: "No featured products found" });
        }

        await redis.set("featured_products", JSON.stringify(featuredProducts));
        res.json(featuredProducts);
    } catch (error) {
        console.log("Error in getStarredProducts controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};
```

Рисунок 3.14 – Програмна реалізація функції getStarredProducts із кешуванням

Додавання нового товару доступне лише адміністратору. Контролер отримує дані форми, завантажує зображення до Cloudinary та зберігає посилання на нього разом із рештою полів товару до бази даних. Видалення товару також передбачає попереднє видалення зображення з Cloudinary, щоб уникнути накопичення невикористаних файлів у хмарному сховищі.



```
const createNewProduct = async (req, res) => {
    try {
        const { name, description, price, image, category } = req.body;

        let cloudinatyResponse = null;

        if (image) {
            cloudinatyResponse = await cloudinary.uploader.upload(image, {
                folder: "myproducts",
            });
        }
        const product = await Product.create({
            name,
            description,
            price,
            image: cloudinatyResponse?.secure_url
                ? cloudinatyResponse.secure_url
                : "",
            category,
        });
        res.status(201).json(product);
    } catch (error) {
        console.log("Error in createNewProduct controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};

Windsurf: Refactor | Explain | Generate JSDoc | X
const removeProduct = async (req, res) => {
    try {
        const product = await Product.findById(req.params.id);
        if (!product) {
            return res.status(404).json({ message: "Product not found" });
        }

        if (product.image) {
            const publicId = product.image.split("/").pop().split(".")[0];
            try {
                await cloudinary.uploader.destroy(`myproducts/${publicId}`);
                console.log("deleted image from cloudinary");
            } catch (error) {
                console.log("error deleting image from cloudinary", error);
            }
        }
        await Product.findByIdAndDelete(req.params.id);

        res.json({ message: "Prouduct deleted successfully" });
    } catch (error) {
        console.log("Error in removeProduct controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};
```

Рисунок 3.15 – Програмна реалізація функцій createNewProduct та removeProduct

Перемикання статусу рекомендованого товару реалізовано через PATCH-запит до /api/products/:id. Контролер знаходить товар за ідентифікатором, інвертує значення поля isFeatured та зберігає зміни. Після цього автоматично оновлюється кеш рекомендованих товарів у Redis.

```
const toggleStarredProduct = async (req, res) => {
    try {
        const product = await Product.findById(req.params.id);
        if (product) {
            product.isFeatured = !product.isFeatured;
            const updatedProduct = await product.save();
            //change redis
            await refreshFeaturedProductsCache();
            res.json(updatedProduct);
        } else {
            res.status(404).json({ message: "Product not found" });
        }
    } catch (error) {
        console.log("Error in toggleStarredProduct controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};

```

Рисунок 3.16 – Програмна реалізація функції toggleStarredProduct

3.3.4 Реалізація кошика та системи купонів

Кошик у застосунку не є окремою сутністю в базі даних — він зберігається безпосередньо в документі користувача як масив cartContents. Таке рішення спрощує архітектуру та зменшує кількість запитів до бази даних, оскільки дані кошика завантажуються разом із профілем користувача.

Рисунок 3.17 – Програмна реалізація маршрутів кошика

Модуль кошика містить чотири операції. Отримання вмісту кошика реалізовано через GET-запит, який знаходить у базі даних товари за ідентифікаторами зі списку cartContents та повертає їх разом із кількістю. Додавання товару перевіряє, чи він вже є в кошику. Якщо так, кількість збільшується на одиницю, якщо ні — товар додається як новий елемент. Оновлення кількості відбувається через PUT-запит із ідентифікатором товару в URL. Якщо передана кількість дорівнює нулю, товар автоматично видаляється з кошика. DELETE-запит без вказання конкретного товару очищує кошик повністю, тоді як передача productId у тілі запиту видаляє лише один конкретний товар.

Рисунок 3.18 – Програмна реалізація контролерів кошика

Система купонів реалізована через окремий модуль із двома ендпоінтами. Перший повертає активний купон поточного користувача. Він використовується на сторінці кошика, щоб автоматично підказати користувачу про наявну знижку. Другий ендпоінт перевіряє валідність введеного коду купону: знаходить його в базі даних, звіряє належність поточному користувачу, перевіряє статус активності та дату закінчення. Якщо купон прострочений, він автоматично деактивується і зберігається оновленим у базі.

```
const checkCouponValidity = async (req, res) => {
    try {
        const { code } = req.body;

        const coupon = await Coupon.findOne({
            code,
            userId: req.user._id,
            isActive: true,
        });

        if (!coupon) {
            return res.status(404).json({ message: "Coupon not found" });
        }

        if (coupon.expirationDate < new Date()) {
            coupon.isActive = false;
            await coupon.save();
            return res.status(400).json({ message: "Coupon expired" });
        }

        res.json({
            message: "Coupon is valid",
            code: coupon.code,
            discountPercentage: coupon.discountPercentage,
        });
    } catch (error) {
        console.log("Error in checkCouponValidity controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};
```

Рисунок 3.19 – Програмна реалізація функції checkCouponValidity

Нарахування нового купону відбувається автоматично під час обробки успішного платежу. Якщо загальна сума замовлення перевищує 200 доларів, система генерує унікальний код купону з десятивідсотковою знижкою та терміном дії 30 днів. Перед створенням нового купону попередній купон користувача видаляється, що підтримує обмеження одного активного купону на користувача.

```
148  async function createNewCoupon(userId) {
149      await Coupon.findOneAndDelete({ userId });
150
151      const newCoupon = new Coupon({
152          code: "GIFT" + Math.random().toString(36).substring(2, 8).toUpperCase(),
153          discountPercentage: 10,
154          expirationDate: new Date(Date.now() + 30 * 24 * 60 * 60 * 1000), // 30 days from now
155          userId: userId,
156      });
157
158      await newCoupon.save();
159
160      return newCoupon;
161  }
```

Рисунок 3.20 – Програмна реалізація функції createNewCoupon

3.3.5 Інтеграція платіжної системи Stripe

Для обробки онлайн-платежів у застосунку використовується сервіс Stripe — одна з найпоширеніших платіжних платформ для вебзастосунків. Інтеграція реалізована через два ендпоінти: створення платіжної сесії та підтвердження успішної оплати.

При оформленні замовлення клієнтська частина надсилає на сервер список товарів із кошика та, за наявності, код купону. Контролер формує масив позицій у форматі, який очікує Stripe, та підраховує загальну суму. Якщо купон виявляється валідним, знижка застосовується як до суми на стороні сервера, так і до платіжної сесії через механізм Stripe Coupons. Після цього створюється сесія Stripe Checkout із усіма необхідними параметрами: способом оплати, переліком товарів, URL для перенаправлення після успішної або скасованої оплати. У метаданих сесії зберігаються ідентифікатор користувача, код купону

та список товарів із кількостями та цінами. Клієнту повертається URL платіжної сторінки Stripe, на яку його перенаправляють для введення платіжних даних.

Важливим аспектом інтеграції є те, що дані банківської картки користувача жодного разу не проходять через сервер застосунку. Stripe Checkout є повністю розміщеною платіжною сторінкою на стороні Stripe. Користувач вводить реквізити безпосередньо на ній, а не у формі застосунку. Це знімає з розробника відповідальність за зберігання та обробку чутливих платіжних даних і відповідає стандартам безпеки PCI DSS. Сервер лише створює сесію з параметрами замовлення та отримує підтвердження після завершення оплати.

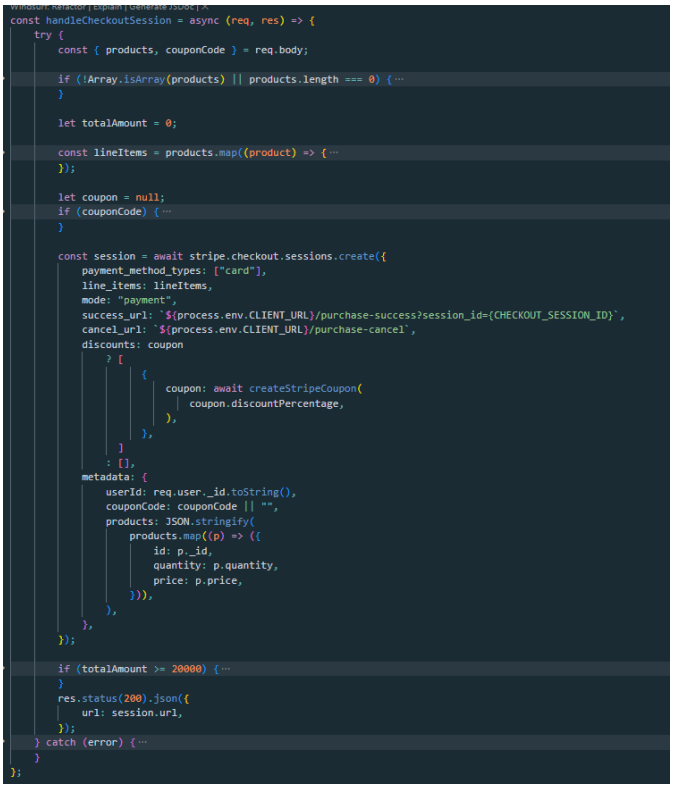


Рисунок 3.21 – Програмна реалізація функції handleCheckoutSession

Окремо реалізовано логіку автоматичного нарахування купону. Якщо загальна сума замовлення перевищує 200 доларів, після створення сесії користувачу автоматично нараховується купон на знижку 10% із терміном дії 30 днів.

Після успішної оплати Stripe перенаправляє користувача на сторінку підтвердження, яка надсилає на сервер ідентифікатор сесії. Контролер отримує деталі сесії від Stripe та перевіряє статус платежу. Якщо оплата підтверджена — використаний купон деактивується, а в базі даних створюється новий запис замовлення з переліком товарів, цінами та загальною сумою. Ідентифікатор сесії Stripe зберігається разом із замовленням, що дозволяє уникнути повторної обробки одного й того самого платежу.

```
Windsurf: Refactor | Explain | Generate JSDoc | ×
const handleCheckoutSuccess = async (req, res) => {
    try {
        const { sessionId } = req.body;
        const session = await stripe.checkout.sessions.retrieve(sessionId);

        if (session.payment_status === "paid") {
            if (session.metadata.couponCode) {
                await Coupon.findOneAndUpdate(
                    {
                        code: session.metadata.couponCode,
                        userId: session.metadata.userId,
                    },
                    {
                        isActive: false,
                    },
                );
            }

            const products = JSON.parse(session.metadata.products);
            const newOrder = new Order({
                user: session.metadata.userId,
                products: products.map((product) => ({
                    product: product.id,
                    quantity: product.quantity,
                    price: product.price,
                })),
                totalAmount: session.amount_total / 100,
                stripeSessionId: sessionId,
            });

            await newOrder.save();

            res.status(200).json({
                success: true,
                message:
                    "Payment successful, order created, and coupon deactivated if used.",
                orderId: newOrder._id,
            });
        }
    } catch (error) {
        console.error("Error processing successful checkout:", error);
        res.status(500).json({
            message: "Error processing successful checkout",
            error: error.message,
        });
    }
};
```

Рисунок 3.22 – Програмна реалізація функції handleCheckoutSuccess

3.4 Розробка клієнтської частини

Клієнтська частина застосунку реалізована за концепцією односторінкового застосунку SPA (Single Page Application), де навігація між сторінками відбувається без перезавантаження браузера. Це забезпечує плавну та швидку взаємодію користувача з інтерфейсом. В основі лежить бібліотека React, яка дозволяє будувати інтерфейс із незалежних компонентів, кожен із

яких відповідає за окрему частину сторінки. Усі звернення до серверного API виконуються через попередньо налаштований екземпляр axios, який автоматично визначає базову URL-адресу залежно від середовища розробки чи продакшну та передає cookie з кожним запитом. Для відображення повідомлень про успішні дії або помилки використовується бібліотека react-hot-toast, що забезпечує зручний зворотний зв'язок із користувачем без додаткових модальних вікон.

3.4.1 Структура React-застосунку та маршрутизація

Точкою входу клієнтської частини є файл main.jsx, де кореневий компонент App монтується у DOM. Застосунок обгортається у BrowserRouter, що активує механізм клієнтської маршрутизації через React Router.

При кожному завантаженні сторінки компонент App першочергово перевіряє стан автентифікації користувача, звертаючись до серверного API. Поки відповідь не отримана — відображається екран завантаження, що виключає ситуацію, коли захищений контент на мить з'являється перед неавторизованим користувачем. Одночасно з перевіркою автентифікації завантажується вміст кошика.

Маршрути застосунку поділено на три групи залежно від рівня доступу. Перша група — публічні сторінки, доступні всім: головна сторінка та каталоги за категоріями. Друга група — сторінки для неавторизованих користувачів: реєстрація та вхід. Якщо авторизований користувач намагається їх відкрити, його автоматично перенаправляє на головну. Третя група — захищені сторінки, які вимагають входу: кошик, підтвердження та скасування замовлення. Адміністративна панель виділена окремо, адже вона доступна лише користувачам із роллю admin, решта отримує перенаправлення на сторінку входу.

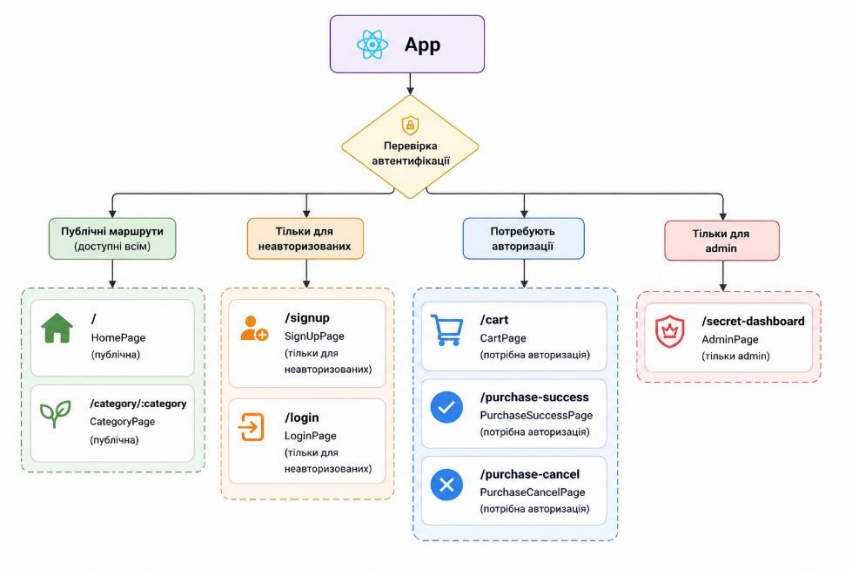


Рисунок 3.23 – Схема маршрутизації вебзастосунку

3.4.2 Управління станом через Zustand

Для управління глобальним станом у клієнтській частині застосунку використовується бібліотека Zustand. На відміну від Redux, який вимагає значної кількості допоміжного коду, Zustand дозволяє описати стан і логіку його зміни в одному компактному файлі. Кожне сховище створюється через функцію create і є повністю незалежним, що спрощує структуру коду та полегшує його підтримку.

У застосунку визначено три окремі сховища. Сховище useUserStore відповідає за стан автентифікації — зберігає дані поточного користувача, індикатор завантаження та прапорець перевірки автентифікації. Тут зосереджено всі операції, пов'язані з обліковим записом: реєстрація, вхід, вихід, перевірка автентифікації та оновлення токена. Окремої уваги заслуговує реалізація оновлення access-токену. У файлі сховища налаштовано axios-інтерцептор, який автоматично перехоплює відповіді з кодом 401 і намагається оновити токен без втручання користувача. Якщо оновлення не вдається — виконується автоматичний вихід із системи.

Рисунок 3.24 – Програмна реалізація useUserStore та axios-інтерцептора

Сховище useProductStore керує даними про товари. Воно містить масив продуктів, індикатор завантаження та набір асинхронних функцій для отримання товарів із сервера за категорією, рекомендованих або повного списку для адміністратора. Адміністративні операції, такі як створення, видалення та перемикання статусу рекомендованого товару, також реалізовані тут і після виконання автоматично оновлюють локальний стан без додаткового запиту до сервера.

Рисунок 3.25 – Програмна реалізація useProductStore

Найбільш розгалуженим є сховище useCartStore, яке керує вмістом кошика, купонами та підрахунком загальної суми. Після кожної зміни кошика, додавання, видалення або оновлення кількості, автоматично викликається функція calculateMyTotal, яка перераховує суму з урахуванням активного купону. Застосування та видалення купону також реалізовані у цьому сховищі — після зміни стану купону сума одразу перераховується.

```
import { create } from "zustand";  864 (gzipped: 483)
import axios from "../lib/axios";
import { toast } from "react-hot-toast";  9k (gzipped: 3.6k)

export const useCartStore = create((set, get) => ({
    cart: [],
    coupon: null,
    total: 0,
    subtotal: 0,
    isCouponApplied: false,

    Windsurf: Refactor | Explain | Generate JSDoc | X
    receiveMyCoupon: async () => {···
    },
    Windsurf: Refactor | Explain | Generate JSDoc | X
    useMyCoupon: async (code) => {···
    },
    Windsurf: Refactor | Explain | Generate JSDoc | X
    deleteCoupon: () => {···
    },

    Windsurf: Refactor | Explain | Generate JSDoc | X
    receiveCartItems: async () => {···
    },
    Windsurf: Refactor | Explain | Generate JSDoc | X
    emptyCart: async () => {···
    },
    Windsurf: Refactor | Explain | Generate JSDoc | X
    addItemToCart: async (product) => {···
    },
    Windsurf: Refactor | Explain | Generate JSDoc | X
    deleteFromCart: async (productId) => {···
    },
    Windsurf: Refactor | Explain | Generate JSDoc | X
    updateMyQuantity: async (productId, quantity) => {···
    },
    Windsurf: Refactor | Explain | Generate JSDoc | X
    calculateMyTotal: () => {
        const { cart, coupon } = get();
        const subtotal = cart.reduce(
            (sum, item) => sum + item.price * item.quantity,
            0,
        );
        let total = subtotal;

        if (coupon) {
            const discount = subtotal * (coupon.discountPercentage / 100);
            total = subtotal - discount;
        }

        set({ subtotal, total });
    },
}));
```

Рисунок 3.26 – Програмна реалізація useCartStore

3.4.3 Реалізація основних сторінок та компонентів

Клієнтська частина застосунку складається з восьми сторінок та набору багаторазових компонентів. Кожна сторінка є окремим React-компонентом, який отримує дані зі сховищ Zustand та відображає відповідний інтерфейс. Для анімації елементів використовується бібліотека Framer Motion — вона

забезпечує плавну появу компонентів при завантаженні сторінки, що покращує загальне сприйняття інтерфейсу.

Головна сторінка HomePage відображає сітку категорій товарів та підбірку рекомендованих рослин. При завантаженні автоматично виконується запит до сервера для отримання рекомендованих товарів. Категорії визначені статично у вигляді масиву об'єктів із назвою, зображенням та посиланням. Рекомендовані товари відображаються у вигляді каруселі з можливістю перегортання, яка адаптується до розміру екрана, на мобільних пристроях показується один товар, на планшетах — два, на десктопі — три.

Рисунок 3.27 – Програмна реалізація головної сторінки застосунку

Сторінка CategoryPage завантажує товари обраної категорії з сервера при кожній зміні параметра URL. Назва категорії береться з параметрів маршруту та відображається у заголовку сторінки з першою великою літерою. Товари відображаються у адаптивній сітці через компонент ProductCard, який показує зображення, назву, опис та ціну товару з кнопкою додавання до кошика. Якщо користувач не авторизований, при спробі додати товар він отримує відповідне повідомлення.

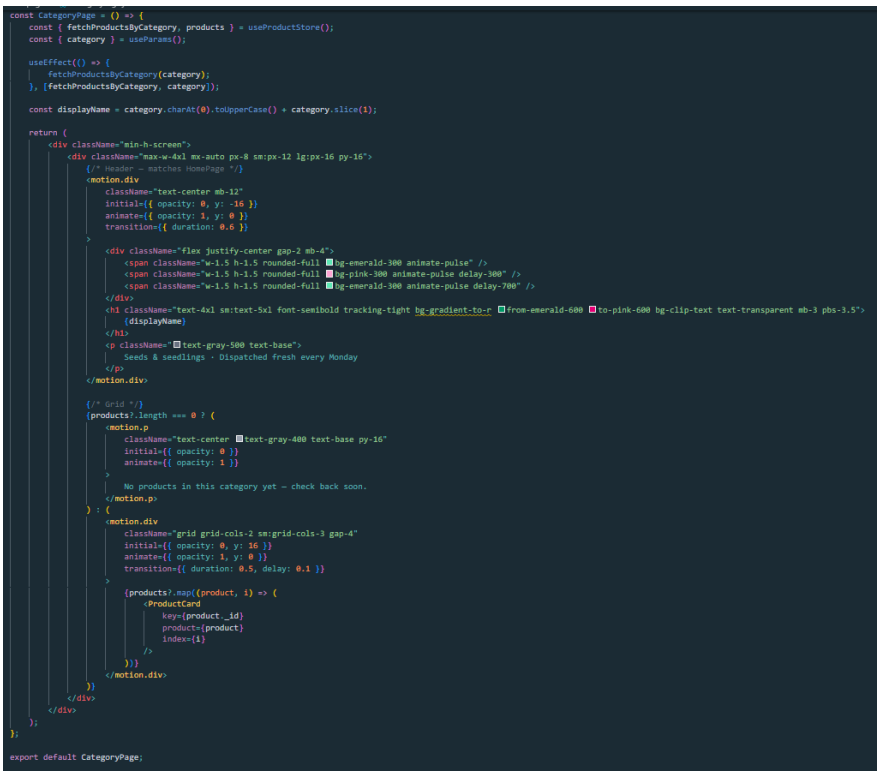


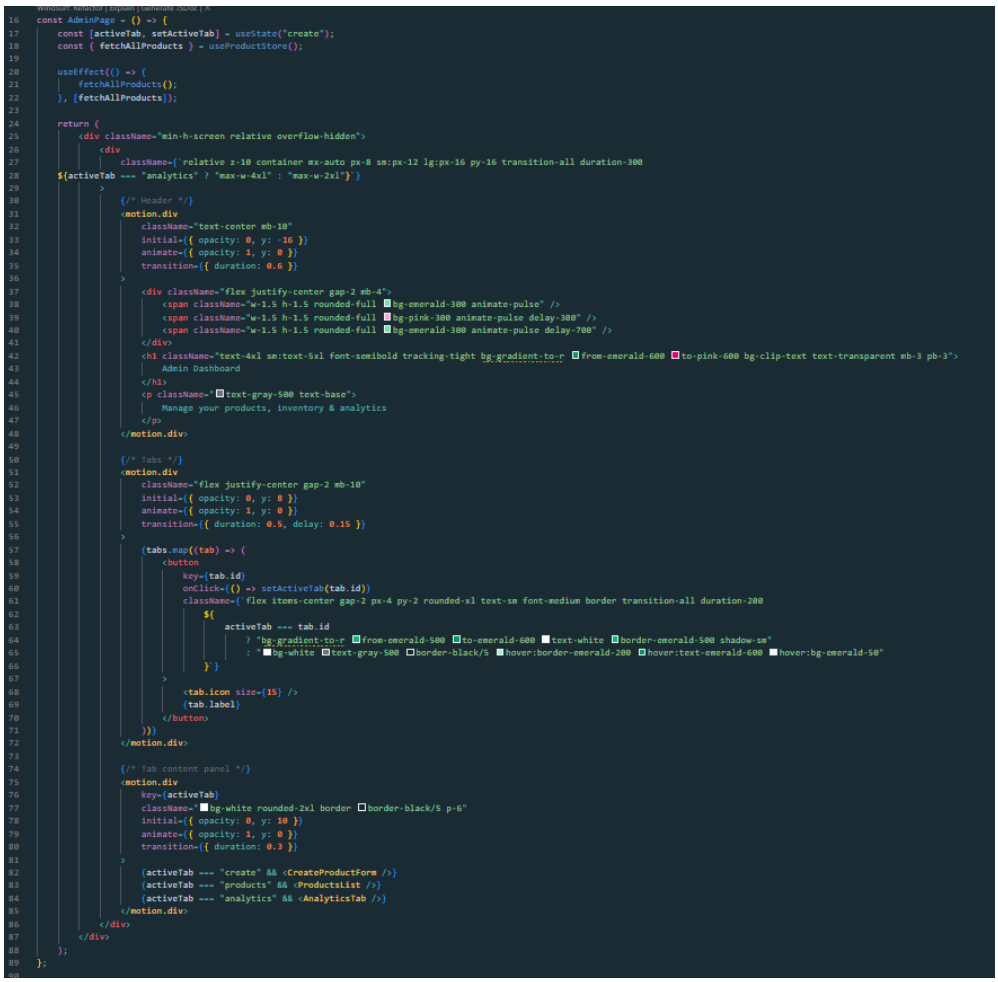
Рисунок 3.28 – Програмна реалізація сторінки категорії товарів

Сторінка CartPage відображає вміст кошика у вигляді списку елементів CartItem, кожен із яких дозволяє змінити кількість або видалити товар. Поруч розташовані компоненти GiftCouponCard для введення купону та OrderSummary із підсумком замовлення та кнопкою оплати. При натисканні на кнопку оплати клієнт звертається до сервера для створення платіжної сесії Stripe та перенаправляє користувача на сторінку оплати. Якщо кошик порожній — відображається відповідне повідомлення з посиланням на каталог. Нижче списку товарів розташований компонент PeopleAlsoBought, який показує випадкові рекомендації.

```
Windsurf: Refactor | Explain | Generate JSDoc | ×
const CartPage = () => {
    const { cart } = useCartStore();

    return (
        <div className="min-h-screen">
            <div className="max-w-2xl mx-auto px-8 sm:px-12 lg:px-16 py-16">
                <motion.div
                    className="mb-8"
                    initial={{ opacity: 0, y: -12 }}
                    animate={{ opacity: 1, y: 0 }}
                    transition={{ duration: 0.5 }}
                >
                    <div className="flex items-baseline gap-2">
                        <h1 className="text-2xl font-semibold text-gray-800">
                            Your cart
                        </h1>
                        <span className="text-sm text-gray-400">
                            {cart.length} {cart.length === 1 ? "item" : "items"}
                        </span>
                    </div>
                </motion.div>

                {cart.length === 0 ? (
                    <EmptyCartUI />
                ) : (
                    <>
                        <motion.div
                            initial={{ opacity: 0, y: 12 }}
                            animate={{ opacity: 1, y: 0 }}
                            transition={{ duration: 0.4, delay: 0.1 }}
                        >
                            <div className="flex flex-col gap-2.5 mb-6">
                                {cart.map((item, i) => (
                                    <CartItem
                                        key={item._id}
                                        item={item}
                                        index={i}
                                    />
                                ))}
                            </div>
                        </motion.div>

                        <motion.div
                            className="mx-auto mt-6 max-w-4xl flex-1 space-y-6 lg:mt-0 lg:w-full"
                            initial={{ opacity: 0, x: 20 }}
                            animate={{ opacity: 1, x: 0 }}
                            transition={{ duration: 0.5, delay: 0.4 }}
                        >
                            <GiftCouponCard />
                            <OrderSummary />
                            <PeopleAlsoBought />
                        </motion.div>
                    </>
                )}
            </div>
        </div>
    );
};

export default CartPage;
```

Рисунок 3.29 – Програмна реалізація сторінки кошика

Адміністративна панель AdminPage містить три вкладки: створення товару, список товарів та аналітика. Перемикання між вкладками реалізовано через локальний стан компонента. Форма створення товару дозволяє завантажити зображення через FileReader, який конвертує файл у формат base64 перед відправкою на сервер. Вкладка аналітики відображає чотири показники: кількість користувачів, товарів, загальну кількість продажів та виручку, а також лінійний графік динаміки продажів і виручки за останні 7 днів, побудований за допомогою бібліотеки Recharts.



```jsx
const AdminPage = () => {
    const [activeTab, setActiveTab] = useState("create");
    const { fetchAllProducts } = useProductStore();

    useEffect(() => {
        fetchAllProducts();
    }, [fetchAllProducts]);

    return (
        <div className="min-h-screen relative overflow-hidden">
            <div
                className={`relative z-10 container mx-auto px-8 sm:px-12 lg:px-16 py-16 transition-all duration-300
${activeTab === "analytics" ? "max-w-4xl" : "max-w-2xl"}`}
            >
                {/* Header */}
                <motion.div
                    className="text-center mb-10"
                    initial={{ opacity: 0, y: -16 }}
                    animate={{ opacity: 1, y: 0 }}
                    transition={{ duration: 0.6 }}
                >
                    <div className="flex justify-center gap-2 mb-4">
                        <span className="w-1.5 h-1.5 rounded-full bg-emerald-300 animate-pulse" />
                        <span className="w-1.5 h-1.5 rounded-full bg-pink-300 animate-pulse delay-300" />
                        <span className="w-1.5 h-1.5 rounded-full bg-emerald-300 animate-pulse delay-700" />
                    </div>
                    <h1 className="text-4xl sm:text-5xl font-semibold tracking-tight bg-gradient-to-r from-emerald-600 to-pink-600 bg-clip-text text-transparent mb-3 pb-1">
                        Admin Dashboard
                    </h1>
                    <p className="text-gray-500 text-base">
                        Manage your products, inventory & analytics
                    </p>
                </motion.div>

                {/* Tabs */}
                <motion.div
                    className="flex justify-center gap-2 mb-10"
                    initial={{ opacity: 0, y: 0 }}
                    animate={{ opacity: 1, y: 0 }}
                    transition={{ duration: 0.5, delay: 0.15 }}
                >
                    {tabs.map((tab) => (
                        <button
                            key={tab.id}
                            onClick={() => setActiveTab(tab.id)}
                            className={`flex items-center gap-2 px-4 py-2 rounded-xl text-sm font-medium border transition-all duration-200
                                ${
                                    activeTab === tab.id
                                        ? "bg-gradient-to-r from-emerald-500 to-emerald-600 text-white border-emerald-500 shadow-sm"
                                        : "bg-white text-gray-500 border-black/5 hover:border-emerald-200 hover:text-emerald-600 hover:bg-emerald-50"
                                }`}
                        >
                            <tab.icon size={15} />
                            {tab.label}
                        </button>
                    ))}
                </motion.div>

                {/* Tab content panel */}
                <motion.div
                    key={activeTab}
                    className="bg-white rounded-2xl border border-black/5 p-6"
                    initial={{ opacity: 0, y: 10 }}
                    animate={{ opacity: 1, y: 0 }}
                    transition={{ duration: 0.3 }}
                >
                    {activeTab === "create" && <CreateProductForm />}
                    {activeTab === "products" && <ProductsList />}
                    {activeTab === "analytics" && <AnalyticsTab />}
                </motion.div>
            </div>
        </div>
    );
};
```

Рисунок 3.30 – Програмна реалізація адміністративної панелі

Після успішної оплати користувач потрапляє на сторінку PurchaseSuccessPage, яка надсилає на сервер ідентифікатор платіжної сесії для підтвердження замовлення та очищає кошик. Для святкового ефекту використовується бібліотека react-confetti, яка відображає конфеті на весь екран. У разі скасування платежу відображається сторінка PurchaseCancelPage з відповідним повідомленням та посиланням для повернення до магазину.

ВИСНОВОК ДО РОЗДІЛУ 3

У третьому розділі описано повний цикл розробки вебзастосунку електронної комерції для продажу рослин — від аналізу вимог до реалізації серверної та клієнтської частини.

На початку розділу сформульовано функціональні та нефункціональні вимоги до системи, які визначили напрям подальшої розробки. На їх основі спроєктовано структуру бази даних із чотирьох колекцій — користувачів, товарів, замовлень та купонів — та визначено зв'язки між ними.

Серверну частину реалізовано на основі Node.js та Express.js із дотриманням принципу розділення відповідальності між роутерами, контролерами та моделями. Детально описано механізм автентифікації на основі JWT із парою access та refresh токенів, захист маршрутів через middleware та розмежування доступу між користувачами і адміністраторами. Реалізовано API для управління товарами з кешуванням рекомендованих позицій через Redis, модуль кошика зі збереженням стану в документі користувача, систему купонів із автоматичним нарахуванням знижки та інтеграцію з платіжним сервісом Stripe.

Клієнтську частину побудовано на React за концепцією односторінкового застосунку з маршрутизацією через React Router та розмежуванням доступу до сторінок залежно від ролі користувача. Управління глобальним станом реалізовано через три незалежні сховища Zustand. Описано реалізацію основних сторінок та компонентів застосунку.

Результатом розробки є вебзастосунок електронної комерції, готовий до подальшого тестування.

РОЗДІЛ 4. ОГЛЯД ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОЄКТУ

У цьому розділі представлено функціональність розробленого вебзастосунку електронної комерції для продажу рослин. Кожен модуль системи проілюстровано скріншотами ключових екранів із коротким описом реалізованого функціоналу. Розглянуто взаємодію користувача із застосунком від моменту реєстрації до завершення покупки, а також можливості адміністративної панелі.

4.1 Реєстрація та автентифікація користувача

Новий користувач починає роботу із застосунком зі сторінки реєстрації. Щоб створити обліковий запис, потрібно вказати ім'я, електронну пошту та пароль, який вводиться двічі для підтвердження. Якщо паролі не збігаються, система одразу повідомляє про це — ще до відправки даних на сервер.

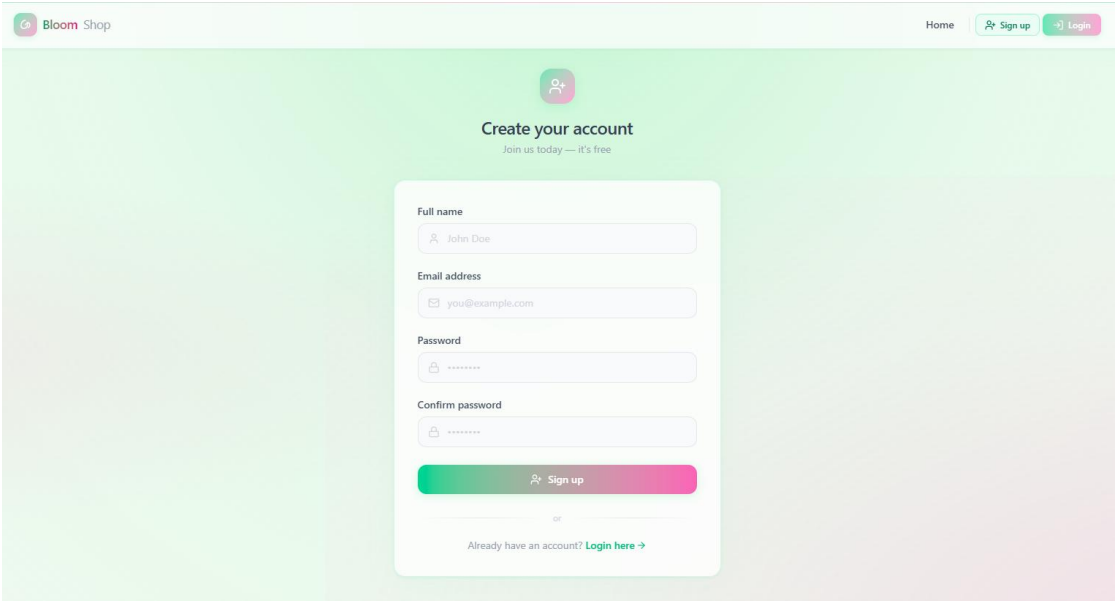


Рисунок 4.1 – Сторінка реєстрації нового користувача

Після успішної реєстрації користувач автоматично входить у систему — токени встановлюються у cookie, і він одразу потрапляє на головну сторінку. Якщо обліковий запис із вказаною поштою вже існує — відображається відповідна помилка.

Для входу в систему користувач вводить електронну пошту та пароль. У разі введення невірних даних сервер повертає помилку, яка відображається у вигляді сповіщення. Авторизований користувач, який намагається відкрити сторінку входу повторно, автоматично перенаправляється на головну.

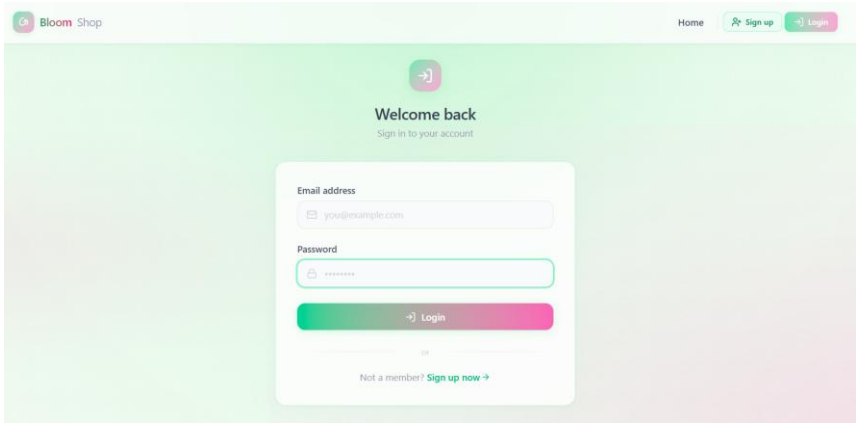


Рисунок 4.2 – Сторінка входу в систему

Стан авторизації зберігається між сесіями завдяки механізму автоматичного оновлення токенів. При кожному завантаженні застосунок перевіряє наявність валідного токена і відновлює сесію без повторного входу.

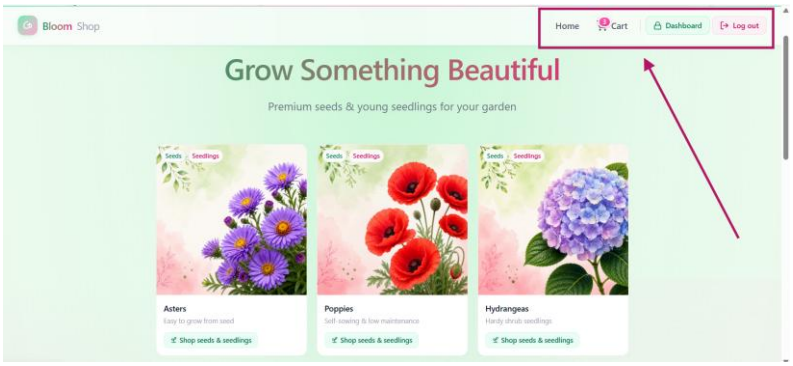


Рисунок 4.3 – Відображення стану авторизованого користувача у навігаційній панелі

4.2 Головна сторінка та каталог товарів

Після входу користувач потрапляє на головну сторінку, де розміщено сітку із шести категорій товарів — айстри, маки, гортензії, дельфініуми, піони та герані. Кожна категорія представлена карткою із зображенням, назвою та коротким описом. Натискання на картку переводить користувача до відповідного каталогу.

Рисунок 4.4 – Головна сторінка застосунку з категоріями товарів

Нижче категорій відображається підбірка рекомендованих товарів у форматі каруселі. Кількість видимих карток залежить від розміру екрана. Гортання здійснюється кнопками ліворуч і праворуч. Кожна картка містить зображення товару, його назву, опис та ціну, а також кнопку додавання до кошика.

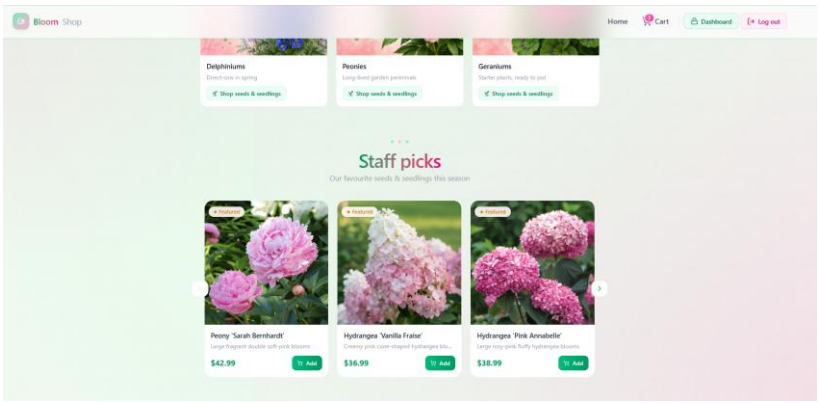


Рисунок 4.5 – Підбірка рекомендованих товарів на головній сторінці

При переході до будь-якої категорії відкривається сторінка каталогу з усіма товарами обраного типу. Товари відображаються в адаптивній сітці — по два або три в рядку залежно від ширини екрана. Картка товару показує зображення, назву, короткий опис та ціну. Неавторизований користувач також може переглядати каталог, однак при спробі додати товар до кошика отримує пропозицію увійти в систему.

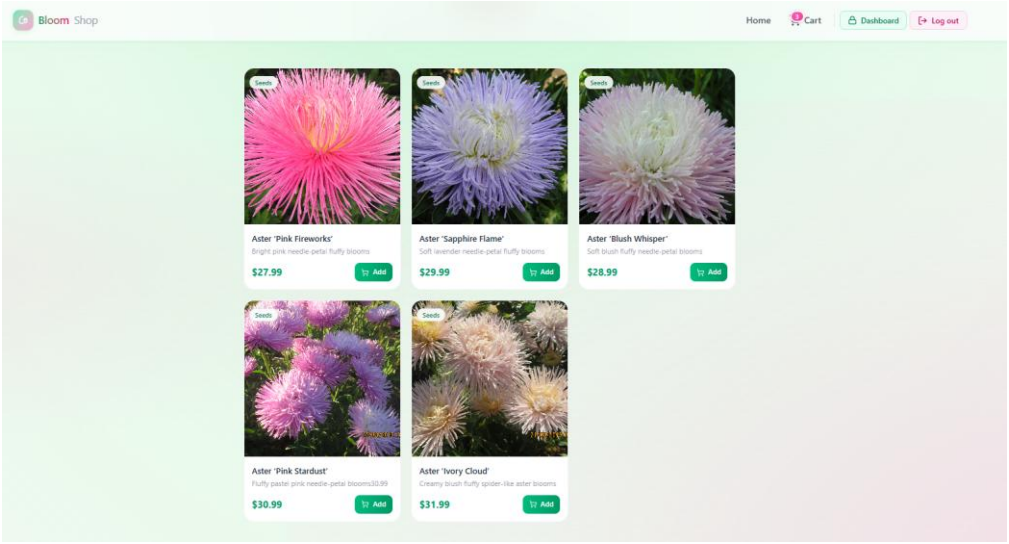


Рисунок 4.6 – Сторінка каталогу товарів за категорією

4.3 Кошик та оформлення замовлення

Сторінка кошика відображає перелік доданих товарів у вигляді списку карток. Кожна картка містить зображення, назву та категорію товару, елементи керування кількістю, підсумкову ціну за позицію та кнопку видалення. Зменшення кількості до нуля автоматично видаляє товар із кошика. У шапці сторінки зазначається загальна кількість позицій. Якщо кошик порожній, відображається відповідне повідомлення з посиланням на каталог.

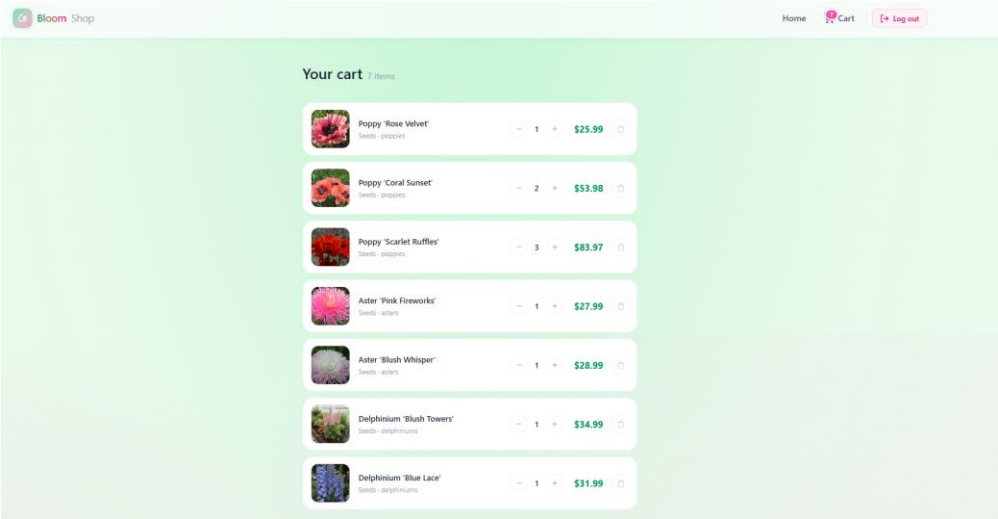


Рисунок 4.7 – Сторінка кошика з переліком доданих товарів

Нижче списку товарів розміщено компонент введення купону. Якщо у користувача є активний купон, система автоматично підказує його код. Після введення коду та натискання кнопки Apply застосунок перевіряє валідність купону на сервері. У разі успіху відображається повідомлення про застосовану знижку, а активований купон можна видалити окремою кнопкою.

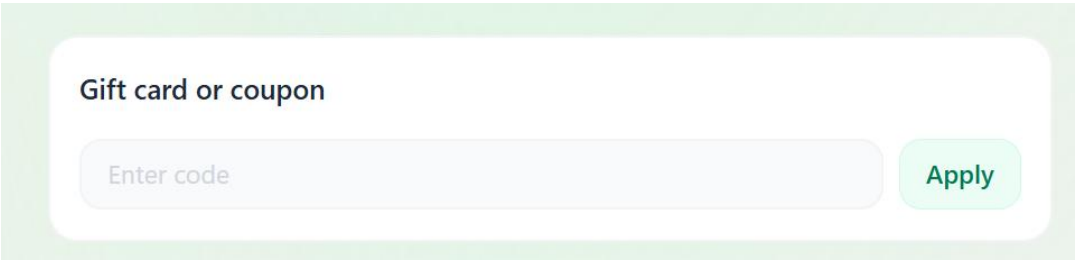


Рисунок 4.8 – Компонент введення та застосування купону

Нижче розташований блок підсумку замовлення. Він відображає підсумкову суму, розмір знижки при активному купоні, позначку безкоштовної доставки та загальну суму до сплати. Кнопка Place order ініціює оформлення покупки. Після натискання застосунок створює платіжну сесію на сервері та перенаправляє користувача на сторінку оплати Stripe.

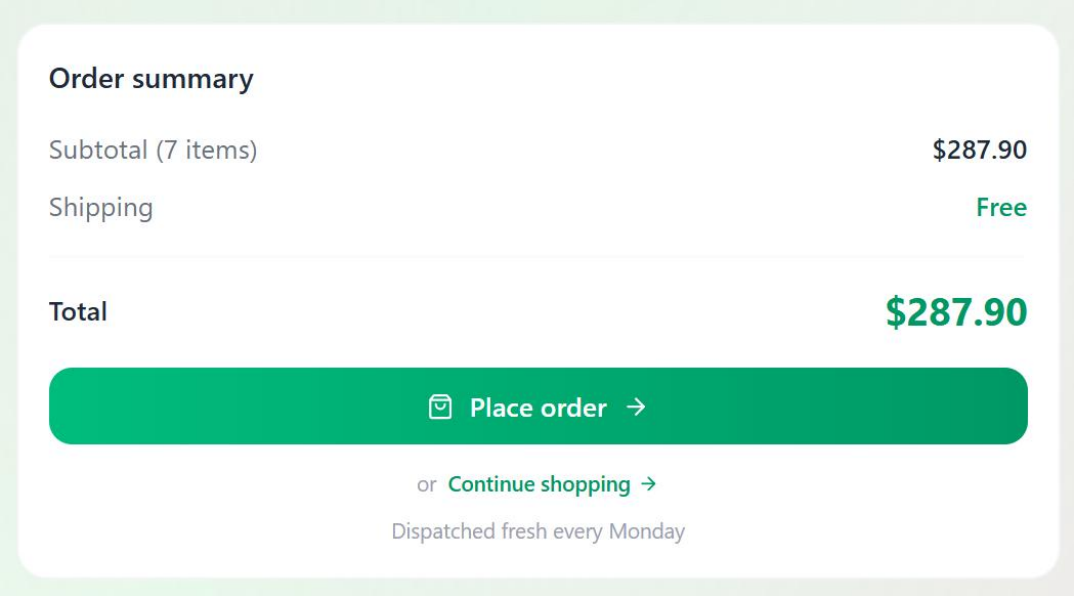


Рисунок 4.9 – Блок підсумку замовлення з підрахунком суми

Нижче підсумку відображається блок рекомендацій — випадкова вибірка з трьох товарів, яка формується на сервері при кожному відкритті сторінки.

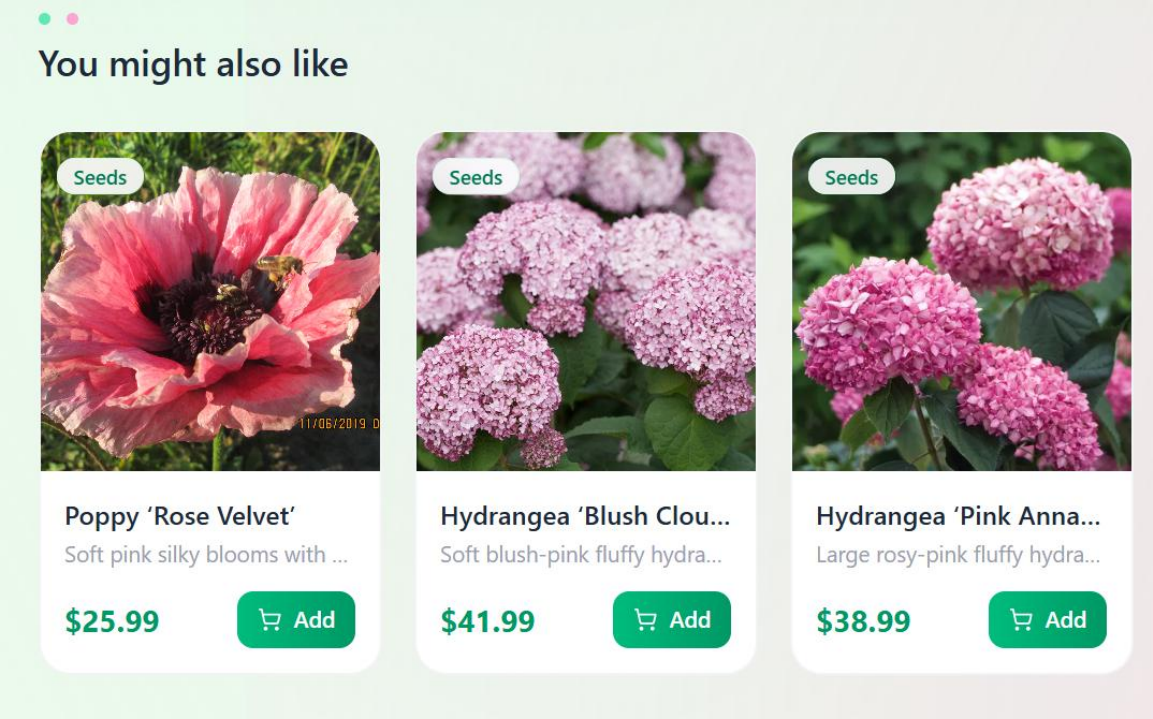


Рисунок 4.10 – Блок рекомендацій на сторінці кошика

4.4 Платіжна система

Після натискання кнопки Place order користувач перенаправляється на захищену сторінку оплати Stripe Checkout. Ця сторінка розміщена безпосередньо на інфраструктурі Stripe і містить перелік товарів, загальну суму, а також поля для введення платіжних даних. Дані банківської картки жодного разу не проходять через сервер застосунку, що забезпечує відповідність стандартам безпеки PCI DSS. Якщо при оформленні було застосовано купон, знижка відображається окремим рядком у підсумку.

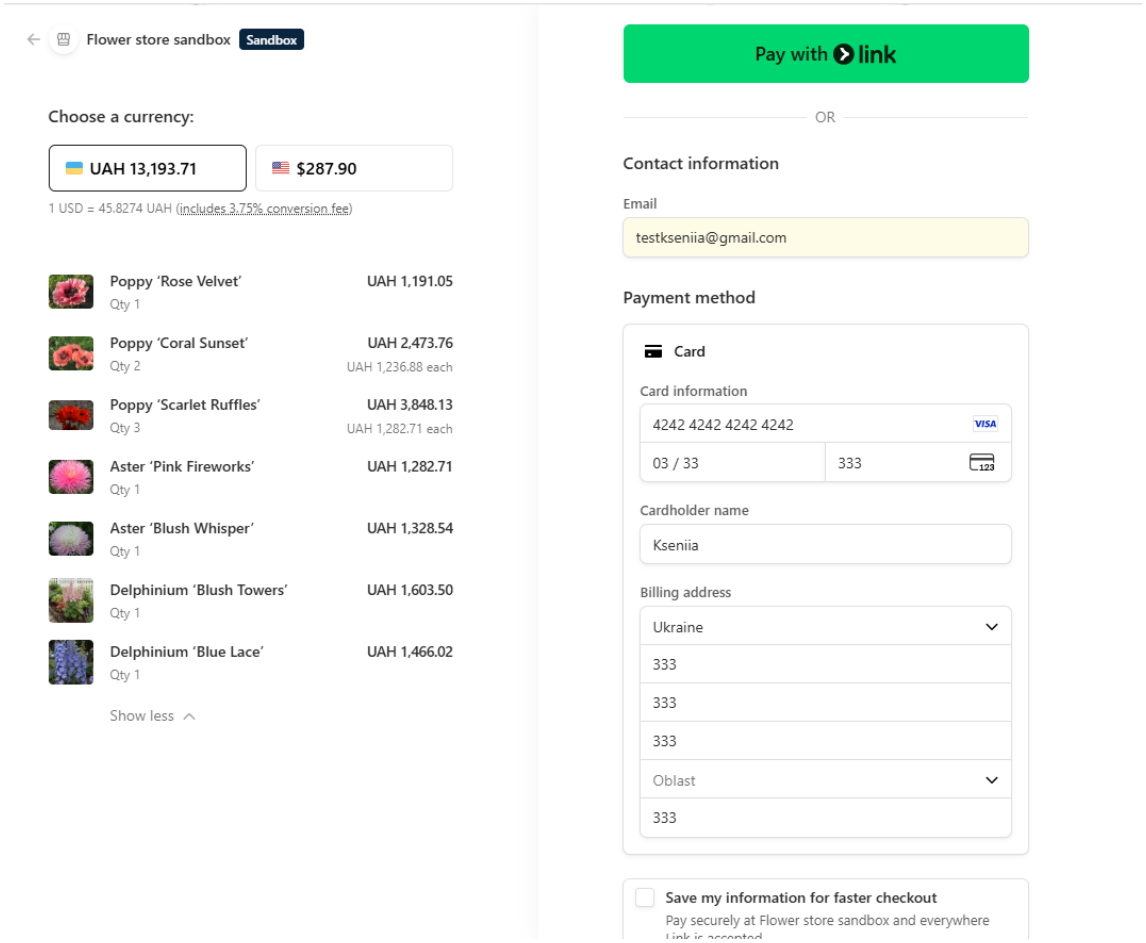


Рисунок 4.11 – Сторінка оплати Stripe Checkout

Після успішного завершення платежу Stripe перенаправляє користувача на сторінку підтвердження замовлення. Сторінка відображає анімацію конфеті та повідомлення про успішну оплату. Одночасно застосунок надсилає на сервер

ідентифікатор платіжної сесії, на основі якого створюється запис замовлення у базі даних, деактивується використаний купон, а кошик автоматично очищується. Якщо сума покупки перевищила 200 доларів, користувачу автоматично нараховується купон на знижку 10% терміном дії 30 днів.

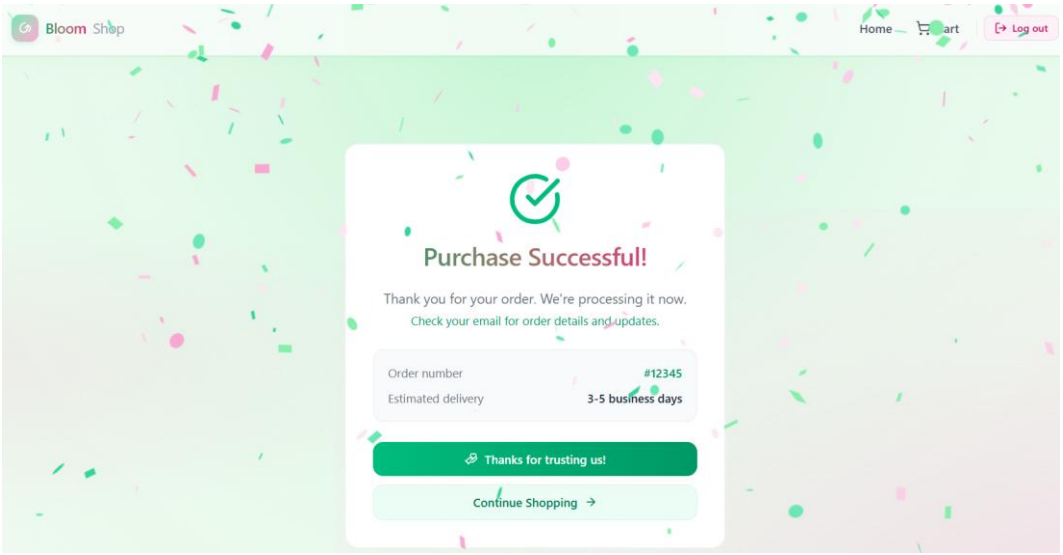


Рисунок 4.12 – Сторінка підтвердження успішного замовлення

У разі скасування платежу на сторінці Stripe користувач перенаправляється на сторінку скасування замовлення. Вона містить відповідне повідомлення про те, що жодних коштів не було знято, та посилання для повернення до магазину. Вміст кошика при цьому зберігається.

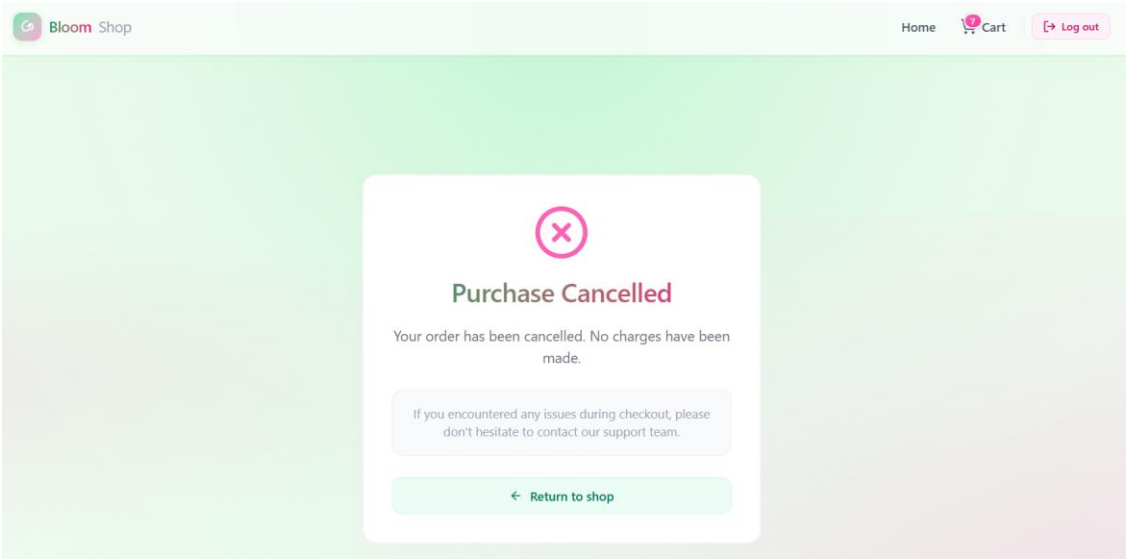


Рисунок 4.13 – Сторінка скасування замовлення

4.5 Адміністративна панель

Адміністративна панель доступна лише користувачам із роллю «admin» через окремий захищений маршрут. Звичайний користувач при спробі перейти за адресою панелі автоматично перенаправляється на сторінку входу. Панель містить три вкладки: Create Product, Products та Analytics.

Вкладка Create Product містить форму для додавання нового товару. Адміністратор вводить назву, опис, ціну, обирає категорію зі списку та завантажує зображення. Завантажений файл конвертується у формат base64 на клієнтській стороні та передається на сервер, де завантажується до хмарного сховища Cloudinary. Кнопка підтвердження блокується на час виконання запиту та відображає індикатор завантаження.

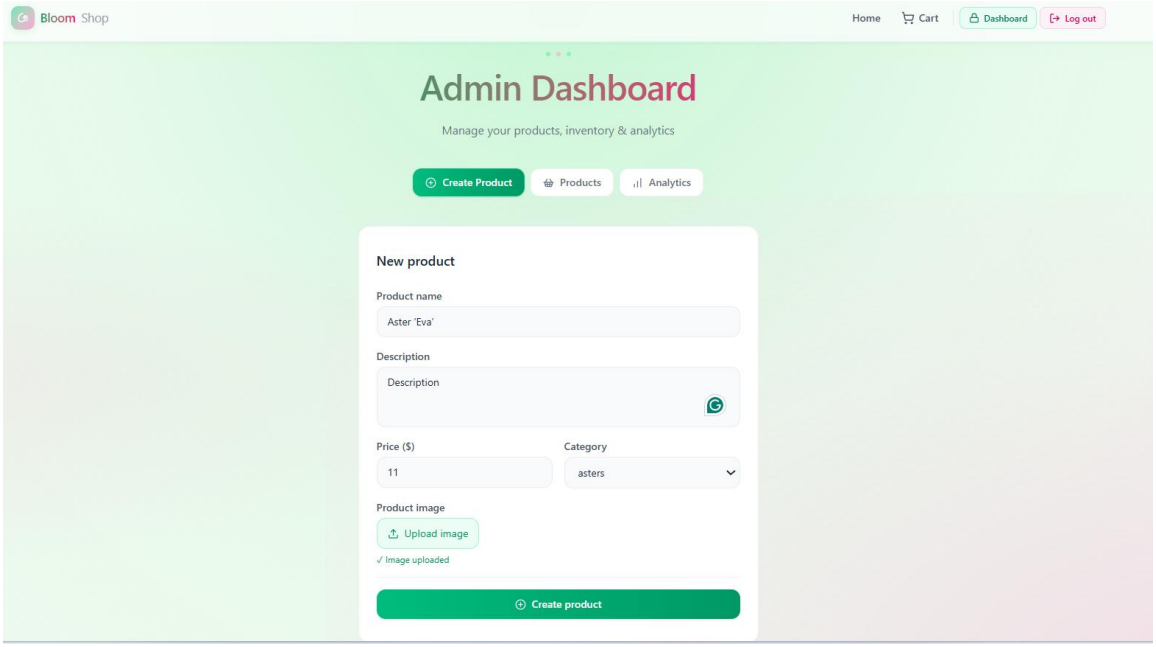


Рисунок 4.14 – Форма створення нового товару

Вкладка Products відображає повний список усіх товарів магазину. Кожен запис містить мініатюру зображення, назву, позначку категорії та ціну. Для кожного товару доступні дві дії: перемикання статусу рекомендованого (зірочка — якщо активна, товар відображається у підбірці рекомендацій на

головній сторінці) та видалення. При видаленні зображення автоматично видаляється з Cloudinary. Зміна статусу рекомендованого товару одразу оновлює кеш Redis без перезавантаження сторінки.

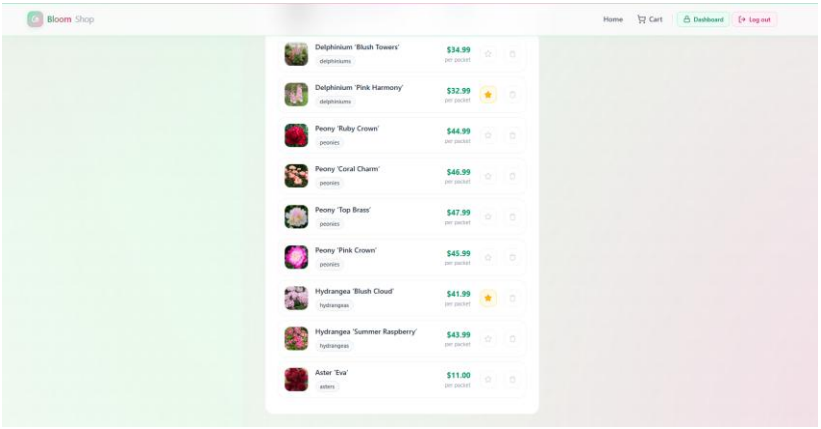


Рисунок 4.15 – Список усіх товарів у адміністративній панелі

Вкладка «Analytics» відображає чотири ключові показники: загальну кількість зареєстрованих користувачів, кількість товарів у каталозі, загальну кількість оформлених замовлень та сукупну виручку. Нижче показників розміщено лінійний графік динаміки продажів та виручки за останні 7 днів, побудований за допомогою бібліотеки Recharts. Графік містить дві лінії — зелену для кількості продажів та рожеву для виручки з підписами по осях і спливаючими підказками при наведенні на точки.

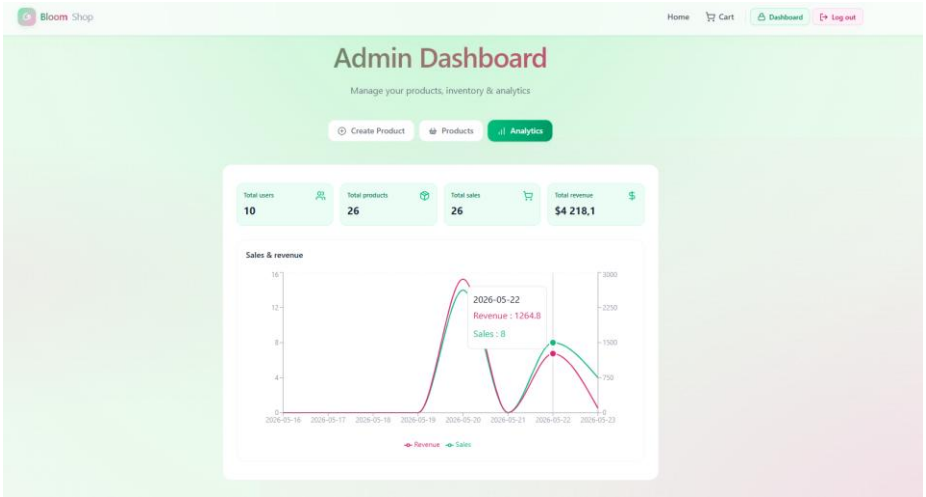


Рисунок 4.16 – Вкладка аналітики з показниками та графіком

ВИСНОВОК ДО РОЗДІЛУ 4

У четвертому розділі представлено повний огляд розробленого вебзастосунку електронної комерції для продажу рослин. Розглянуто всі основні модулі системи та описано взаємодію користувача з інтерфейсом у рамках ключових сценаріїв використання.

Показано, що система реєстрації та автентифікації забезпечує зручний процес створення облікового запису та входу в систему, а механізм JWT-токенів підтримує стан авторизації між сесіями без повторного входу.

Огляд головної сторінки та каталогу підтверджує коректне відображення категорій товарів та рекомендованих позицій. Навігація між сторінками відбувається плавно завдяки клієнтській маршрутизації, а адаптивна сітка товарів коректно відображається на пристроях різних розмірів.

Описано повний процес оформлення замовлення — від формування кошика через застосування купону до проведення оплати через Stripe Checkout. Підтверджено коректну роботу автоматичного нарахування купону при перевищенні порогової суми покупки та очищення кошика після успішної оплати.

Адміністративна панель надає адміністратору повний набір інструментів для управління каталогом товарів та моніторингу ключових показників продажів. Обмеження доступу до панелі на рівні маршрутизації гарантує, що звичайні користувачі не можуть отримати доступ до адміністративного функціоналу.

Таким чином, розроблений вебзастосунок відповідає визначеним у третьому розділі функціональним та нефункціональним вимогам і є готовим програмним продуктом для організації онлайн-продажу рослин.

ВИСНОВКИ

У даному дипломному проєкті розроблено вебзастосунок електронної комерції для продажу рослин. У процесі виконання роботи було досягнуто поставлених цілей та вирішено наступні завдання.

У першому розділі проведено аналіз існуючих рішень у сфері онлайн-продажу рослин. Розглянуто три вебзастосунки — Флоріум.ua, Agro-Market та The Sill. Аналіз показав, що більшість існуючих платформ має застарілий інтерфейс, перевантажену навігацію, відсутність гнучкого адміністрування та обмежену підтримку мов. На основі виявлених недоліків сформовано вимоги до власного застосунку, що поєднує сучасний дизайн, зручний користувацький досвід та адміністративний функціонал.

У другому розділі обґрунтовано вибір архітектурних рішень та технологічного стеку. Обрано клієнт-серверну архітектуру та MERN-стек (MongoDB, Express.js, React.js, Node.js), що дозволяє використовувати єдину мову програмування JavaScript на всіх рівнях системи. Для забезпечення безпеки реалізовано автентифікацію на основі JWT з парою access та refresh токенів із зберіганням у httpOnly cookie, що унеможливлює XSS-атаки. Додатково інтегровано Redis для кешування, Cloudinary для зберігання зображень та Tailwind CSS для стилізації інтерфейсу.

У третьому розділі описано повний цикл розробки застосунку. Спроєктовано структуру бази даних із чотирьох колекцій: User, Product, Order та Coupon. Реалізовано серверну частину на Node.js та Express.js із дотриманням принципу розділення відповідальності між роутерами, контролерами та моделями. Розроблено REST API для управління товарами з кешуванням рекомендованих позицій через Redis, модуль кошика зі збереженням стану в документі користувача, систему купонів із автоматичним нарахуванням знижки при перевищенні порогової суми замовлення, а також інтеграцію з платіжним сервісом Stripe Checkout, що відповідає стандартам безпеки PCI DSS.

Клієнтську частину побудовано на React за концепцією SPA з маршрутизацією через React Router та управлінням глобальним станом через три незалежні сховища Zustand.

У четвертому розділі представлено огляд усіх модулів розробленого застосунку та проілюстровано ключові сценарії взаємодії користувача із системою. Підтверджено коректну роботу реєстрації та автентифікації, перегляду товарів за категоріями, формування кошика, застосування купонів, оформлення та оплати замовлень через Stripe Checkout, а також функціонал адміністративної панелі з управління товарами та аналітикою продажів.

Розроблений вебзастосунок відповідає визначеним функціональним та нефункціональним вимогам. Він забезпечує зручний сучасний інтерфейс, безпечну автентифікацію, ефективне кешування, надійну обробку платежів та інструменти адміністрування. Застосунок може бути використаний для автоматизації онлайн-продажу рослин, а також слугувати основою для розширення функціоналу — додавання системи відгуків, рекомендацій на основі машинного навчання або мобільного додатку.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Flower-Store [Електронний ресурс]. – Режим доступу: https://github.com/KseniiaHorai/Flower-Store (дата звернення: 22.05.2026).
2. What is e-commerce? [Електронний ресурс] / IBM. – Режим доступу: https://www.ibm.com/think/topics/ecommerce (дата звернення: 22.05.2026).
3. History and evolution of web development [Електронний ресурс] / GeeksforGeeks. – Режим доступу: https://www.geeksforgeeks.org/blogs/history-and-evolution-of-web-development/ (дата звернення: 22.05.2026).
4. Вебзастосунок [Електронний ресурс] / Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Вебзастосунок (дата звернення: 22.05.2026).
5. What is an online store? [Електронний ресурс] / Wix. – Режим доступу: https://www.wix.com/blog/what-is-an-online-store (дата звернення: 22.05.2026).
6. Що таке UI та як користувацький інтерфейс впливає на продажі інтернет-магазину [Електронний ресурс] / WEZOM. – Режим доступу: https://wezom.com.ua/ua/blog (дата звернення: 22.05.2026).
7. OWASP Top 10:2025 [Електронний ресурс] / OWASP Foundation. – Режим доступу: https://owasp.org/Top10/2025/ (дата звернення: 22.05.2026).
8. Оптимізація та продуктивність бази даних [Електронний ресурс] / Hostragons. – Режим доступу: https://www.hostragons.com/uk/блог/оптимізація-та-продуктивність-бази-д/ (дата звернення: 22.05.2026).

9. Florium.ua — інтернет-магазин рослин та товарів для садівництва [Електронний ресурс]. – Режим доступу: https://florium.ua/ua/ (дата звернення: 22.05.2026).
10. Agro-Market — інтернет-магазин рослин та товарів для садівництва [Електронний ресурс]. – Режим доступу: https://agro-market.net/ua/ (дата звернення: 22.05.2026).
11. The Sill — an online store of houseplants and plant care products [Електронний ресурс]. – Режим доступу: https://www.thesill.com/ (дата звернення: 22.05.2026).
12. Клієнт-серверна архітектура [Електронний ресурс] / QATestLab. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення: 22.05.2026).
13. What is a Tech Stack and How Do They Work? [Електронний ресурс] / MongoDB. – Режим доступу: https://www.mongodb.com/resources/basics/technology-stack (дата звернення: 22.05.2026).
14. MERN Stack Explained [Електронний ресурс] / MongoDB. – Режим доступу: https://www.mongodb.com/resources/languages/mern-stack (дата звернення: 22.05.2026).
15. React Documentation [Електронний ресурс] / React. – Режим доступу: https://react.dev/learn (дата звернення: 22.05.2026).
16. Node.js Documentation [Електронний ресурс] / Node.js. – Режим доступу: https://nodejs.org/en/docs (дата звернення: 22.05.2026).
17. Express.js — Fast, unopinionated, minimalist web framework for Node.js [Електронний ресурс]. – Режим доступу: https://expressjs.com/ (дата звернення: 22.05.2026).
18. MongoDB Documentation [Електронний ресурс] / MongoDB. – Режим доступу: https://www.mongodb.com/docs/ (дата звернення: 22.05.2026).

19. JSON and BSON [Електронний ресурс] / MongoDB. – Режим доступу: https://www.mongodb.com/resources/basics/json-and-bson (дата звернення: 22.05.2026).
20. Mongoose ODM v8 [Електронний ресурс] / Mongoose. – Режим доступу: https://mongoosejs.com/docs/ (дата звернення: 22.05.2026).
21. Stack Overflow Developer Survey 2025 [Електронний ресурс] / Stack Overflow. – Режим доступу: https://survey.stackoverflow.co/2025/technology#most-popular-technologies-database-database (дата звернення: 22.05.2026).
22. PYPL PopularitY of Programming Language Index — Database Trends [Електронний ресурс] / PYPL. – Режим доступу: https://pypl.github.io/DB.html (дата звернення: 22.05.2026).
23. CRUD Explained [Електронний ресурс] / DEV Community. – Режим доступу: https://dev.to/iammtander/crud-explained-36pe (дата звернення: 22.05.2026).
24. What is middleware? [Електронний ресурс] / IBM. – Режим доступу: https://www.ibm.com/think/topics/middleware (дата звернення: 22.05.2026).
25. React Components [Електронний ресурс] / W3Schools. – Режим доступу: https://www.w3schools.com/react/react_components.asp (дата звернення: 22.05.2026).
26. What is the Virtual DOM in React? [Електронний ресурс] / freeCodeCamp. – Режим доступу: https://www.freecodecamp.org/news/what-is-the-virtual-dom-in-react/ (дата звернення: 22.05.2026).
27. Overview of Blocking vs Non-Blocking [Електронний ресурс] / Node.js. – Режим доступу: https://nodejs.org/en/learn/asynchronous-work/overview-of-blocking-vs-non-blocking (дата звернення: 22.05.2026).
28. Introduction to JSON Web Tokens [Електронний ресурс] / JWT.io. – Режим доступу: https://jwt.io/introduction (дата звернення: 22.05.2026).

29. Mahindrakar P., Pujeri U. Insights of JSON Web Token [Електронний ресурс] // International Journal of Recent Technology and Engineering (IJRTE). – 2020. – Vol. 8, Issue 6. – Режим доступу: https://www.ijrte.org/wp-content/uploads/papers/v8i6/F7689038620.pdf (дата звернення: 22.05.2026).
30. Understanding JSON Web Tokens [Електронний ресурс] / EmbeddedInn. – Режим доступу: https://embeddedinn.com/articles/tutorial/understanding-JSON-web-tokens/ (дата звернення: 22.05.2026).
31. Access Tokens [Електронний ресурс] / Auth0. – Режим доступу: https://auth0.com/docs/secure/tokens/access-tokens (дата звернення: 22.05.2026).
32. Refresh Tokens [Електронний ресурс] / Auth0. – Режим доступу: https://auth0.com/docs/secure/tokens/refresh-tokens (дата звернення: 22.05.2026).
33. Stateless Authentication [Електронний ресурс] / Double Octopus. – Режим доступу: https://doubleoctopus.com/security-wiki/network-architecture/stateless-authentication/ (дата звернення: 22.05.2026).
34. What is Redis? [Електронний ресурс] / Redis. – Режим доступу: https://redis.io/docs/latest/get-started/ (дата звернення: 22.05.2026).
35. Cloudinary Documentation [Електронний ресурс] / Cloudinary. – Режим доступу: https://cloudinary.com/documentation (дата звернення: 22.05.2026).
36. What is a CDN? [Електронний ресурс] / Cloudflare. – Режим доступу: https://www.cloudflare.com/learning/cdn/what-is-a-cdn/ (дата звернення: 22.05.2026).
37. Tailwind CSS Documentation [Електронний ресурс] / Tailwind CSS. – Режим доступу: https://tailwindcss.com/docs/installation (дата звернення: 22.05.2026).

38. Responsive Design [Електронний ресурс] / Tailwind CSS. – Режим доступу: https://tailwindcss.com/docs/responsive-design (дата звернення: 22.05.2026).
39. Функціональні вимоги [Електронний ресурс] / Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Функціональні_вимоги (дата звернення: 22.05.2026).
40. Нефункціональні вимоги [Електронний ресурс] / Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Нефункціональні_вимоги (дата звернення: 22.05.2026).

ДОДАТОК 1

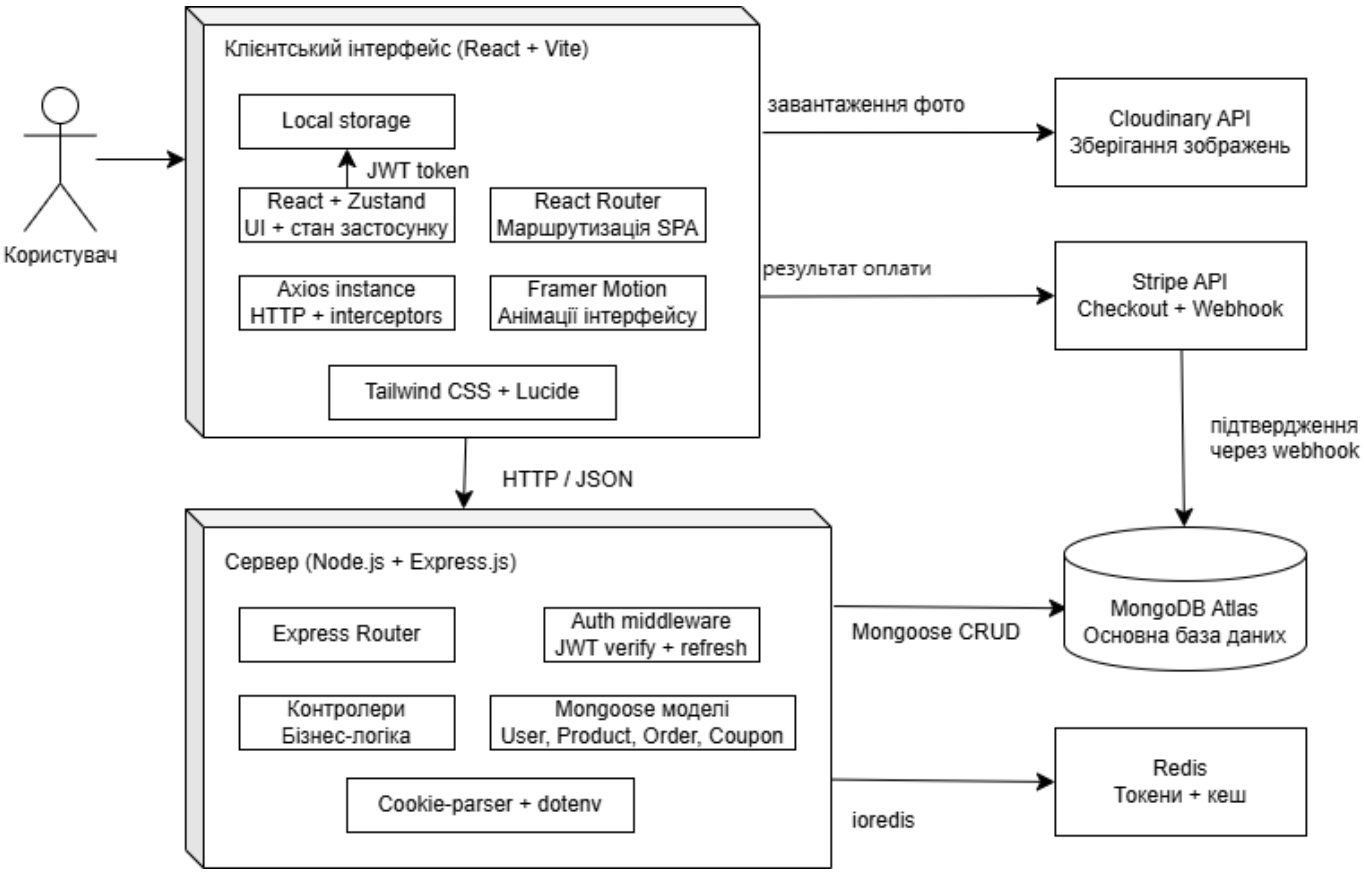
Вебзастосунок електронної комерції для продажу рослин

Структурна схема системи

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2026 р



					ІАЛЦ.467200.004 Д1	
		№ докум.	Підпис	Дата		
Розробив		Горай К.О.			**Вебзастосунок електронної комерції для продажу рослин** Структурна схема системи	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Роковий О.П.				
Реценз.		Деведжіогуллари А. В.				КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-21
Н. Контр.		Міщенко Л. Д.				
Затвердив		Волокита А.М.				

ДОДАТОК 2

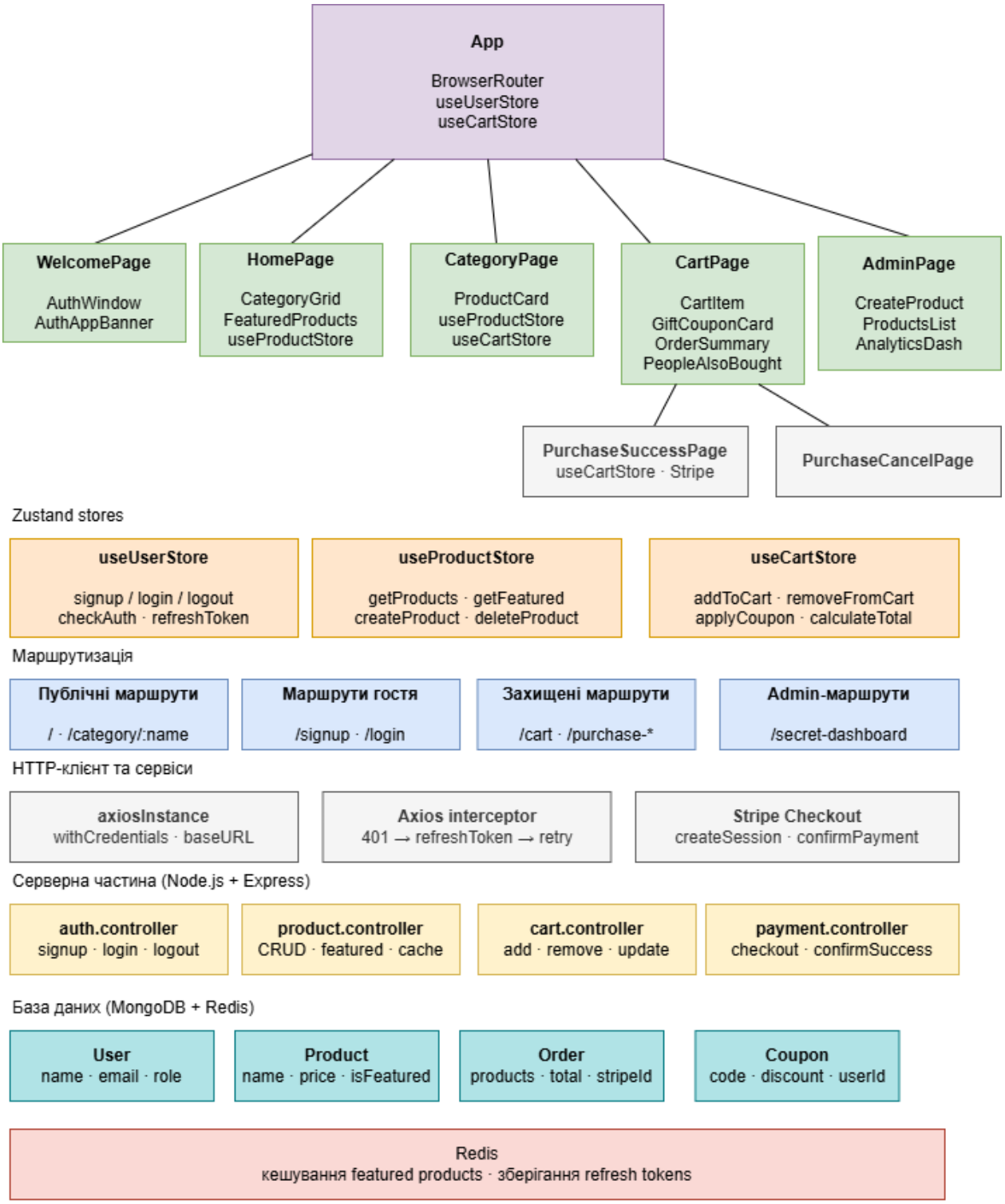
Вебзастосунок електронної комерції для продажу рослин

Функціональна схема

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2026 р



					ІАЛЦ.467200.005 Д2	
		№ докум.	Підпис	Дата		
Розробив		Горай К.О.			**Вебзастосунок електронної комерції для продажу рослин** Функціональна схема	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Роковий О.П.				
Реценз.		Деведжіоуллари А. В.				КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-21
Н. Контр.		Міщенко Л. Д.				
Затвердив		Волокита А.М.				

ДОДАТОК 3

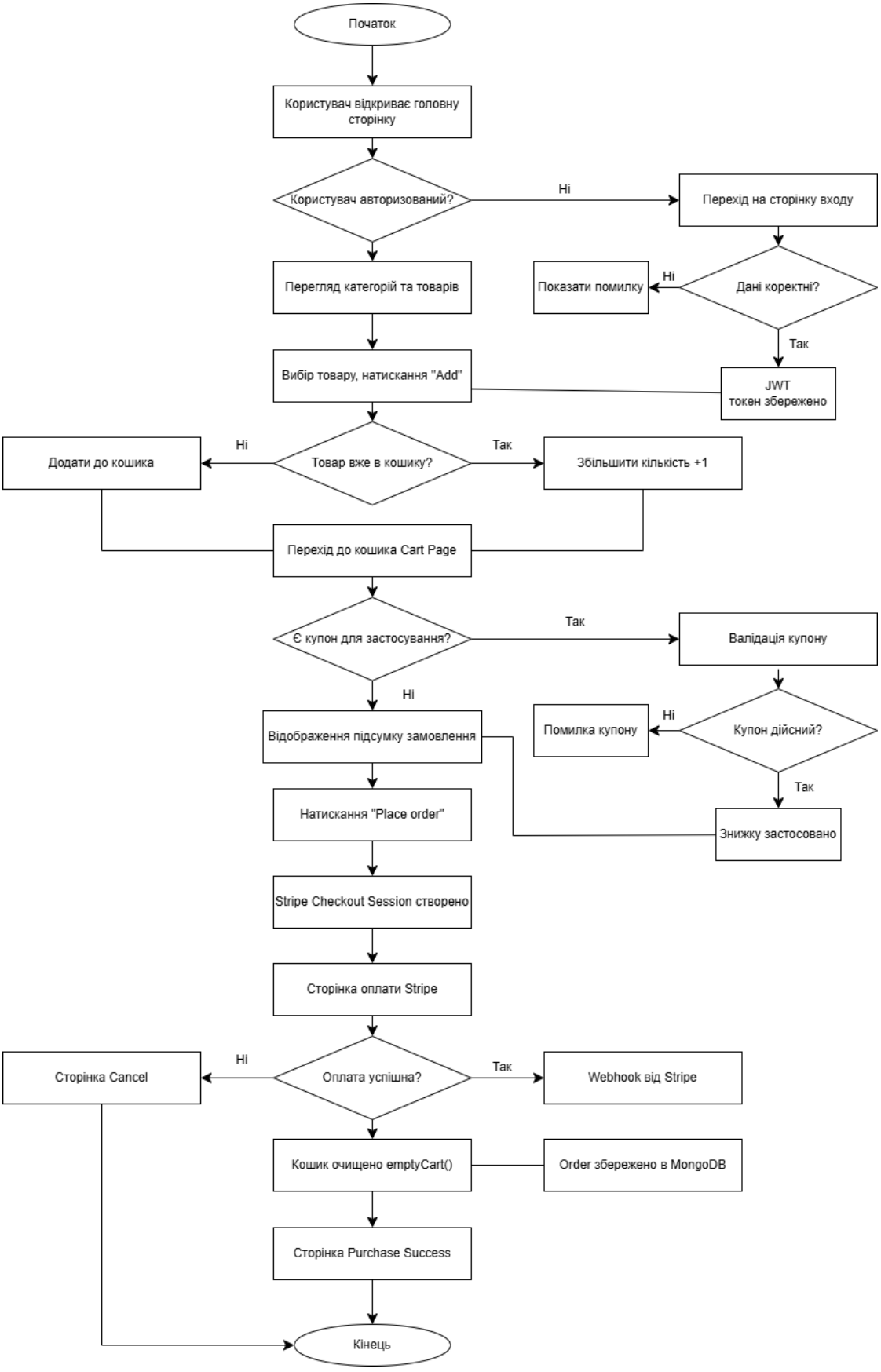
Вебзастосунок електронної комерції для продажу рослин

Алгоритм дій програмного забезпечення

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2026 р



					ІАЛЦ.467200.006 Д3	
		№ докум.	Підпис	Дата		
Розробив		Горай К.О.			**Вебзастосунок електронної комерції для продажу рослин** Алгоритм дій програмного забезпечення	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Роковий О.П.				
Реценз.		Деведжіогуллари А. В.				КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-21
Н. Контр.		Міщенко Л.Д.				
Затвердив		Волокита А.М.				

ДОДАТОК 4

Вебзастосунок електронної комерції для продажу рослин

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 47

Київ 2026 р

app.js

```
const express = require("express");
const cookieParser = require("cookie-parser");
const dotenv = require("dotenv");

const authRoutes = require("./endpoints/auth.route.js");
const productRoutes = require("./endpoints/product.route.js");
const cartRoutes = require("./endpoints/cart.route.js");
const couponRoutes = require("./endpoints/coupon.route.js");
const paymentRoutes = require("./endpoints/payment.route.js");
const analyticsRoutes = require("./endpoints/analytics.route.js");

const setupDatabase = require("./lib/database.js");

dotenv.config();

const app = express();
const PORT = process.env.PORT || 4000;

app.use(express.json({ limit: "10mb" }));
app.use(cookieParser());

app.use("/api/auth", authRoutes);
app.use("/api/products", productRoutes);
app.use("/api/cart", cartRoutes);
app.use("/api/coupons", couponRoutes);
app.use("/api/payments", paymentRoutes);
app.use("/api/analytics", analyticsRoutes);

app.listen(PORT, () => {
    console.log("Listening on port 4000");

    setupDatabase();
});
```

auth.middleware.js

```
const User = require("../models/User.js");
const jwt = require("jsonwebtoken");

const verifyRoute = async (req, res, next) => {
    try {
        const accessToken = req.cookies.accessToken;

        if (!accessToken) {
            return res
                .status(401)
                .json({ message: "Unauthorized - No access token provided" });
        }

        try {
            const decoded = jwt.verify(
                accessToken,
                process.env.SECRET_ACCESS_TOKEN,
            );
            const user = await User.findById(decoded.userId).select(
```

```
                "-password",
            );

            if (!user) {
                return res.status(401).json({ message: "User not found" });
            }

            req.user = user;

            next();
        } catch (error) {
            if (error.name === "TokenExpiredError") {
                return res
                    .status(401)
                    .json({ message: "Unauthorized - Access token expired" });
            }
            throw error;
        }
    } catch (error) {
        console.log("Error in verifyRoute middleware", error.message);
        return res.status(401).json({ message: "Unauthorized" });
    }
};

const adminOnlyRoute = async (req, res, next) => {
    if (req.user && req.user.role === "admin") {
        next();
    } else {
        return res.status(403).json({ message: "Access denied - Admin only" });
    }
};

module.exports = { verifyRoute, adminOnlyRoute };
```

auth.controller.js

```
const User = require("../models/User.js");
const redis = require("../lib/redis.js");
const jwt = require("jsonwebtoken");
```

```
const getMyTokens = function (userId) {
    const accessToken = jwt.sign({ userId }, process.env.SECRET_ACCESS_TOKEN, {
        expiresIn: "15m",
    });
    const refreshToken = jwt.sign(
        { userId },
        process.env.SECRET_REFRESH_TOKEN,
        {
            expiresIn: "10d",
        },
    );
    return { accessToken, refreshToken };
};

const saveRefreshToken = async (userId, refreshToken) => {
    await redis.set(
        `refresh_token:${userId}`,
        refreshToken,
        "EX",
        10 * 24 * 60 * 60,
    );
};

const setCookies = (res, accessToken, refreshToken) => {
    res.cookie("accessToken", accessToken, {
        httpOnly: true, //against XSS hacking
        secure: process.env.NODE_ENV === "production",
        sameSite: "strict", //against CSRF attacks
        maxAge: 10 * 60 * 1000,
    });

    res.cookie("refreshToken", refreshToken, {
        httpOnly: true, //against XSS hacking
        secure: process.env.NODE_ENV === "production",
        sameSite: "strict", //against CSRF attacks
        maxAge: 10 * 24 * 60 * 60 * 1000,
    });
};
```

```
const signup = async (req, res) => {
    const { email, password, name } = req.body;
    try {
        const userIsInDB = await User.findOne({ email });

        if (userIsInDB) {
            return res.status(400).json({ message: "User already exists" });
        }
        const newUser = await User.create({ name, email, password });

        //AUTHENTICATION
        const { accessToken, refreshToken } = getMyTokens(newUser._id);
        await saveRefreshToken(newUser._id, refreshToken);

        setCookies(res, accessToken, refreshToken);

        res.status(201).json({
            newUser: {
                _id: newUser._id,
                name: newUser.name,
                email: newUser.email,
                role: newUser.role,
            },
            message: "User created successfully",
        });
    } catch (error) {
        console.log("Signup error occured", error.message);
        res.status(500).json({ message: error.message });
    }
};

const login = async (req, res) => {
    try {
        const { email, password } = req.body;
        const newUser = await User.findOne({ email });

        if (newUser && (await newUser.comparePassword(password))) {
            const { accessToken, refreshToken } = getMyTokens(newUser._id);
```

```
            await saveRefreshToken(newUser._id, refreshToken);
            setCookies(res, accessToken, refreshToken);

            res.json({
                _id: newUser._id,
                name: newUser.name,
                email: newUser.email,
                role: newUser.role,
            });
        } else {
            res.status(400).json({ message: "Invalid credentials" });
        }
    } catch (error) {
        console.log("Login error occured", error.message);
        res.status(500).json({ message: error.message });
    }
};

const logout = async (req, res) => {
    try {
        const refreshToken = req.cookies.refreshToken;
        if (refreshToken) {
            const decodedData = jwt.verify(
                refreshToken,
                process.env.SECRET_REFRESH_TOKEN,
            );
            await redis.del(`refresh_token:${decodedData.userId}`);
        }
        res.clearCookie("accessToken");
        res.clearCookie("refreshToken");
        res.json({ message: "Logout was successful" });
    } catch (error) {
        console.log("Logout error occured", error.message);
        res.status(500).json({
            message: "something is wrong with the server",
            error: error.message,
        });
    }
```

```
};
const refreshToken = async (req, res) => {
    try {
        const refreshToken = req.cookies.refreshToken;

        if (!refreshToken) {
            return res
                .status(401)
                .json({ message: "Refresh token is not found" });
        }

        const decoded = jwt.verify(
            refreshToken,
            process.env.SECRET_REFRESH_TOKEN,
        );
        const storedToken = await redis.get(`refresh_token:${decoded.userId}`);

        if (storedToken !== refreshToken) {
            return res.status(401).json({ message: "Invalid refresh token" });
        }

        const accessToken = jwt.sign(
            { userId: decoded.userId },
            process.env.SECRET_ACCESS_TOKEN,
            { expiresIn: "15m" },
        );

        res.cookie("accessToken", accessToken, {
            httpOnly: true, //against XSS hacking
            secure: process.env.NODE_ENV === "production",
            sameSite: "strict", //against CSRF attacks
            maxAge: 10 * 60 * 1000,
        });

        res.json({ message: "Token refreshed successfully" });
    } catch (error) {
        console.log("Error in refreshToken controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
```

```
};

const getProfile = async (req, res) => {
    try {
        res.json(req.user);
    } catch (error) {
        res.status(500).json({ message: "Server error", error: error.message });
    }
};

module.exports = {
    signup,
    login,
    logout,
    refreshToken,
    getProfile,
};
```

product.controller.js

```
const Product = require("../models/Product.js");
const redis = require("../lib/redis.js");
const cloudinary = require("../lib/cloudinary.js");

const retrieveAllProducts = async (req, res) => {
    try {
        const products = await Product.find({});
        res.json({ products });
    } catch (error) {
        console.log("Error in retrieveAllProducts controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};

const getStarredProducts = async (req, res) => {
    try {
        let featuredProducts = await redis.get("featured_products");
        if (featuredProducts) {
            return res.json(JSON.parse(featuredProducts));
```

```
        }

        //if not in redis, fetch from mongodb
        featuredProducts = await Product.find({ isFeatured: true }).lean();
        if (!featuredProducts) {
            return res
                .status(404)
                .json({ message: "No featured products found" });
        }

        await redis.set("featured_products", JSON.stringify(featuredProducts));
        res.json(featuredProducts);
    } catch (error) {
        console.log("Error in getStarredProducts controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};

const createNewProduct = async (req, res) => {
    try {
        const { name, description, price, image, category } = req.body;

        let cloudinatyResponse = null;

        if (image) {
            cloudinatyResponse = await cloudinary.uploader.upload(image, {
                folder: "myproducts",
            });
        }
        const product = await Product.create({
            name,
            description,
            price,
            image: cloudinatyResponse?.secure_url
                ? cloudinatyResponse.secure_url
                : "",
            category,
        });
        res.status(201).json(product);
```

```
    } catch (error) {
        console.log("Error in createNewProduct controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};

const removeProduct = async (req, res) => {
    try {
        const product = await Product.findById(req.params.id);
        if (!product) {
            return res.status(404).json({ message: "Product not found" });
        }

        if (product.image) {
            const publicId = product.image.split("/").pop().split(".")[0];
            try {
                await cloudinary.uploader.destroy(`myproducts/${publicId}`);
                console.log("deleted image from cloudinary");
            } catch (error) {
                console.log("error deleting image from cloudinary", error);
            }
        }
        await Product.findByIdAndDelete(req.params.id);

        res.json({ message: "Prouduct deleted successfully" });
    } catch (error) {
        console.log("Error in removeProduct controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};

const getSuggestedProducts = async (req, res) => {
    try {
        const products = await Product.aggregate([
            {
                $sample: { size: 3 },
            },
            {
                $project: {
```

```
                    _id: 1,
                    name: 1,
                    description: 1,
                    image: 1,
                    price: 1,
                },
            },
        ]);
        res.json(products);
    } catch (error) {
        console.log("Error in getSuggestedProducts controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};

const fetchProductsByCategory = async (req, res) => {
    const { category } = req.params;
    try {
        const products = await Product.find({ category });
        res.json({ products });
    } catch (error) {
        console.log(
            "Error in fetchProductsByCategory controller",
            error.message,
        );
        res.status(500).json({ message: "Server error", error: error.message });
    }
};

const toggleStarredProduct = async (req, res) => {
    try {
        const product = await Product.findById(req.params.id);
        if (product) {
            product.isFeatured = !product.isFeatured;
            const updatedProduct = await product.save();
            //change redis
            await refreshFeaturedProductsCache();
            res.json(updatedProduct);
        } else {
```

```
            res.status(404).json({ message: "Product not found" });
        }
    } catch (error) {
        console.log("Error in toggleStarredProduct controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};

const refreshFeaturedProductsCache = async (req, res) => {
    try {
        const featuredProducts = await Product.find({
            isFeatured: true,
        }).lean();
        await redis.set("featured_products", JSON.stringify(featuredProducts));
    } catch (error) {
        console.log("Error in refreshFeaturedProductsCache", error.message);
    }
};

module.exports = {
    retrieveAllProducts,
    getStarredProducts,
    createNewProduct,
    removeProduct,
    getSuggestedProducts,
    fetchProductsByCategory,
    toggleStarredProduct,
};
```

payment.controller.js

```
const Coupon = require("../models/Coupon.js");
const Order = require("../models/Order.js");
const stripe = require("../lib/stripe.js");

const handleCheckoutSession = async (req, res) => {
    try {
        const { products, couponCode } = req.body;
```

```
if (!Array.isArray(products) || products.length === 0) {
    return res
        .status(400)
        .json({ error: "Invalid or empty products array" });
}

let totalAmount = 0;

const lineItems = products.map((product) => {
    const amount = Math.round(product.price * 100);
    totalAmount += amount * product.quantity;

    return {
        price_data: {
            currency: "usd",
            product_data: {
                name: product.name,
                images: [product.image],
            },
            unit_amount: amount,
        },
        quantity: product.quantity || 1,
    };
});

let coupon = null;
if (couponCode) {
    coupon = await Coupon.findOne({
        code: couponCode,
        userId: req.user._id,
        isActive: true,
    });
    if (coupon) {
        totalAmount -= Math.round(
            (totalAmount * coupon.discountPercentage) / 100,
        );
    }
}
```

```
        const session = await stripe.checkout.sessions.create({
            payment_method_types: ["card"],
            line_items: lineItems,
            mode: "payment",
            success_url:                                    `${process.env.CLIENT_URL}/purchase-
success?session_id={CHECKOUT_SESSION_ID}`,
            cancel_url: `${process.env.CLIENT_URL}/purchase-cancel`,
            discounts: coupon
                ? [
                      {
                          coupon: await createStripeCoupon(
                              coupon.discountPercentage,
                          ),
                      },
                  ]
                : [],
            metadata: {
                userId: req.user._id.toString(),
                couponCode: couponCode || "",
                products: JSON.stringify(
                    products.map((p) => ({
                        id: p._id,
                        quantity: p.quantity,
                        price: p.price,
                    })),
                ),
            },
        });

        if (totalAmount >= 20000) {
            await createNewCoupon(req.user._id);
        }
        res.status(200).json({
            url: session.url,
        });
    } catch (error) {
        console.error("Error processing checkout:", error);
        res.status(500).json({
            message: "Error processing checkout",
```

```
            error: error.message,
        });
    }
};

const handleCheckoutSuccess = async (req, res) => {
    try {
        const { sessionId } = req.body;
        const session = await stripe.checkout.sessions.retrieve(sessionId);

        if (session.payment_status === "paid") {
            if (session.metadata.couponCode) {
                await Coupon.findOneAndUpdate(
                    {
                        code: session.metadata.couponCode,
                        userId: session.metadata.userId,
                    },
                    {
                        isActive: false,
                    },
                );
            }

            const products = JSON.parse(session.metadata.products);
            const newOrder = new Order({
                user: session.metadata.userId,
                products: products.map((product) => ({
                    product: product.id,
                    quantity: product.quantity,
                    price: product.price,
                })),
                totalAmount: session.amount_total / 100,
                stripeSessionId: sessionId,
            });

            await newOrder.save();

            res.status(200).json({
                success: true,
```

```
                message:
                    "Payment successful, order created, and coupon deactivated if used.",
                orderId: newOrder._id,
            });
        }
    } catch (error) {
        console.error("Error processing successful checkout:", error);
        res.status(500).json({
            message: "Error processing successful checkout",
            error: error.message,
        });
    }
};

async function createStripeCoupon(discountPercentage) {
    const coupon = await stripe.coupons.create({
        percent_off: discountPercentage,
        duration: "once",
    });

    return coupon.id;
}

async function createNewCoupon(userId) {
    await Coupon.findOneAndDelete({ userId });

    const newCoupon = new Coupon({
        code: "GIFT" + Math.random().toString(36).substring(2, 8).toUpperCase(),
        discountPercentage: 10,
        expirationDate: new Date(Date.now() + 30 * 24 * 60 * 60 * 1000), // 30 days from now
        userId: userId,
    });

    await newCoupon.save();

    return newCoupon;
}

module.exports = { handleCheckoutSession, handleCheckoutSuccess };
```

cart.controller.js

```
const Product = require("../models/Product.js");
const addItemToCart = async (req, res) => {
    try {
        const { productId } = req.body;
        const user = req.user;

        const existingItem = user.cartContents.find(
            (item) => item.id === productId,
        );

        if (existingItem) {
            existingItem.quantity += 1;
        } else {
            user.cartContents.push(productId);
        }

        await user.save();
        res.json(user.cartContents);
    } catch (error) {
        console.log("Error in addItemToCart controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};

const deleteAllFromCart = async (req, res) => {
    try {
        const productId = req.body?.productId;
        const user = req.user;

        if (!productId) {
            user.cartContents = [];
        } else {
            user.cartContents = user.cartContents.filter(
                (item) => item.product?.toString() !== productId,
            );
        }
```

```
        await user.save();
        res.json(user.cartContents);
    } catch (error) {
        console.log("Error in deleteAllFromCart controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};
const updateMyQuantity = async (req, res) => {
    try {
        const { id: productId } = req.params;
        const { quantity } = req.body;
        const user = req.user;
        const existingItem = user.cartContents.find(
            (item) => item.id === productId,
        );

        if (existingItem) {
            if (quantity === 0) {
                user.cartContents = user.cartContents.filter(
                    (item) => item.id !== productId,
                );
                await user.save();
                return res.json(user.cartContents);
            }

            existingItem.quantity = quantity;
            await user.save();
            return res.json(user.cartContents);
        } else {
            res.status(404).json({ message: "Item not found in cart" });
        }
    } catch (error) {
        console.log("Error in updateMyQuantity controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};

const receiveCartItems = async (req, res) => {
```

```
    try {
        const products = await Product.find({
            _id: { $in: req.user.cartContents.map((item) => item.id || item) },
        });

        const cartContents = products.map((product) => {
            const item = req.user.cartContents.find(
                (cartItem) =>
                    (cartItem.id || cartItem).toString() ===
                    product._id.toString(),
            );

            return {
                ...product.toJSON(),
                quantity: item?.quantity || 1,
            };
        });

        res.json(cartContents);
    } catch (error) {
        console.log("Error in receiveCartItems controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};

module.exports = {
    addItemToCart,
    deleteAllFromCart,
    updateMyQuantity,
    receiveCartItems,
};
```

coupon.controller.js

```
const Coupon = require("../models/Coupon.js");

const getUserCoupon = async (req, res) => {
    try {
        const coupon = await Coupon.findOne({
            userId: req.user._id,
```

```
            isActive: true,
        });

        res.json(coupon || null);
    } catch (error) {
        console.log("Error in getUserCoupon controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};
const checkCouponValidity = async (req, res) => {
    try {
        const { code } = req.body;

        const coupon = await Coupon.findOne({
            code,
            userId: req.user._id,
            isActive: true,
        });

        if (!coupon) {
            return res.status(404).json({ message: "Coupon not found" });
        }

        if (coupon.expirationDate < new Date()) {
            coupon.isActive = false;
            await coupon.save();
            return res.status(400).json({ message: "Coupon expired" });
        }

        res.json({
            message: "Coupon is valid",
            code: coupon.code,
            discountPercentage: coupon.discountPercentage,
        });
    } catch (error) {
        console.log("Error in checkCouponValidity controller", error.message);
        res.status(500).json({ message: "Server error", error: error.message });
    }
};
```

```
module.exports = { getUserCoupon, checkCouponValidity };
```

analytics.controller.js

```
const Order = require("../models/Order.js");
const User = require("../models/User.js");
const Product = require("../models/Product.js");

const loadAnalyticsData = async () => {
    const allUsers = await User.countDocuments();
    const allProducts = await Product.countDocuments();

    const salesData = await Order.aggregate([
        {
            $group: {
                _id: null,
                totalSales: { $sum: 1 },
                totalRevenue: { $sum: "$totalAmount" },
            },
        },
    ]);

    const { totalSales, totalRevenue } = salesData[0] || {
        totalSales: 0,
        totalRevenue: 0,
    };

    return {
        users: allUsers,
        products: allProducts,
        totalSales,
        totalRevenue,
    };
};

const loadDailySalesData = async (startDate, endDate) => {
    try {
        const dailySalesData = await Order.aggregate([
```

```
            {
                $match: {
                    createdAt: {
                        $gte: startDate,
                        $lte: endDate,
                    },
                },
            },
            {
                $group: {
                    _id: {
                        $dateToString: {
                            format: "%Y-%m-%d",
                            date: "$createdAt",
                        },
                    },
                    sales: { $sum: 1 },
                    revenue: { $sum: "$totalAmount" },
                },
            },
            { $sort: { _id: 1 } },
        ]);

        const dateArray = listDatesInRange(startDate, endDate);

        return dateArray.map((date) => {
            const foundData = dailySalesData.find((item) => item._id === date);

            return {
                date,
                sales: foundData?.sales || 0,
                revenue: foundData?.revenue || 0,
            };
        });
    } catch (error) {
        throw error;
    }
};
```

```
function listDatesInRange(startDate, endDate) {
    const dates = [];
    let currentDate = new Date(startDate);

    while (currentDate <= endDate) {
        dates.push(currentDate.toISOString().split("T")[0]);
        currentDate.setDate(currentDate.getDate() + 1);
    }

    return dates;
}

module.exports = {
    loadAnalyticsData,
    loadDailySalesData,
};
```

User.js

```
const mongoose = require("mongoose");
const bcrypt = require("bcryptjs");

const UserSchema = new mongoose.Schema(
    {
        name: {
            type: String,
            required: [true, "name cannot be empty"],
        },
        email: {
            type: String,
            required: [true, "email cannot be empty"],
            unique: true,
            lowercase: true,
            trim: true,
        },
        role: {
            type: String,
            enum: ["user", "admin"],
            default: "user",
```

```
        },
        password: {
            type: String,
            required: [true, "password cannot be empty"],
            minlength: [8, "please use a password with 8+ characters"],
        },
        cartContents: [
            {
                quantity: {
                    type: Number,
                    default: 1,
                },
                product: {
                    type: mongoose.Schema.Types.ObjectId,
                    ref: "Product",
                },
            },
        ],
    },
    {
        timestamps: true,
    },
);

UserSchema.pre("save", async function (next) {
    if (!this.isModified("password")) return;

    try {
        const salt = await bcrypt.genSalt(10);
        this.password = await bcrypt.hash(this.password, salt);
    } catch (error) {
        next(error);
    }
});

UserSchema.methods.comparePassword = async function (password) {
    return bcrypt.compare(password, this.password);
};
```

```
const User = mongoose.model("User", UserSchema);

module.exports = User;
```

Order.js

```
const mongoose = require("mongoose");

const OrderSchema = new mongoose.Schema(
    {
        user: {
            type: mongoose.Schema.Types.ObjectId,
            ref: "User",
            required: true,
        },
        products: [
            {
                product: {
                    type: mongoose.Schema.Types.ObjectId,
                    ref: "Product",
                    required: true,
                },
                quantity: {
                    type: Number,
                    required: true,
                    min: 1,
                },
                price: {
                    type: Number,
                    required: true,
                    min: 0,
                },
            },
        ],
        totalAmount: {
            type: Number,
            required: true,
            min: 0,
        },
```

```
        stripeSessionId: {
            type: String,
            unique: true,
        },
    },
    { timestamps: true },
);

const Order = mongoose.model("Order", OrderSchema);

module.exports = Order;
```

Product.js

```
const mongoose = require("mongoose");

const ProductSchema = new mongoose.Schema(
    {
        name: {
            type: String,
            required: true,
        },
        description: {
            type: String,
            required: true,
        },
        price: {
            type: Number,
            min: 0,
            required: true,
        },
        image: {
            type: String,
            required: [true, "Image is required"],
        },
        category: {
            type: String,
            required: true,
        },
```

```
        isFeatured: {
            type: Boolean,
            default: false,
        },
    },
    {
        timestamps: true,
    },
);

const Product = mongoose.model("Product", ProductSchema);

module.exports = Product;
```

Coupon.js

```
const mongoose = require("mongoose");

const CouponSchema = new mongoose.Schema(
    {
        code: {
            type: String,
            required: true,
            unique: true,
        },
        discountPercentage: {
            type: Number,
            required: true,
            min: 0,
            max: 100,
        },
        expirationDate: {
            type: Date,
            required: true,
        },
        isActive: {
            type: Boolean,
```

```
            default: true,
        },
        userId: {
            type: mongoose.Schema.Types.ObjectId,
            ref: "User",
            required: true,
            unique: true,
        },
    },
    {
        timestamps: true,
    },
);

const Coupon = mongoose.model("Coupon", CouponSchema);

module.exports = Coupon;
```

App.jsx

```
import { useEffect, useState } from "react";
import { Route, Routes, Navigate } from "react-router-dom";
import HomePage from "./pages/HomePage.jsx";
import SignUpPage from "./pages/SignUpPage.jsx";
import AdminPage from "./pages/AdminPage.jsx";
import LoginPage from "./pages/LoginPage.jsx";
import CategoryPage from "./pages/CategoryPage.jsx";
import CartPage from "./pages/CartPage.jsx";
import LoadingSpinner from "./components/LoadingSpinner.jsx";
import PurchaseSuccessPage from "./pages/PurchaseSuccessPage.jsx";
import PurchaseCancelPage from "./pages/PurchaseCancelPage.jsx";

import Navbar from "./components/Navbar.jsx";
import { Toaster } from "react-hot-toast";
import { useUserStore } from "./stores/useUserStore.js";
import { useCartStore } from "./stores/useCartStore.js";
```

```
function App() {
    const [count, setCount] = useState(0);
    const { user, checkAuth, checkingAuth } = useUserStore();
    const { receiveCartItems } = useCartStore();

    useEffect(() => {
        checkAuth();
    }, [checkAuth]);

    useEffect(() => {
        receiveCartItems();
    }, [receiveCartItems]);

    if (checkingAuth) {
        return <LoadingSpinner />;
    }

    return (
        <div
            className="min-h-screen text-gray-800 relative overflow-hidden"
            style={{ backgroundColor: "#f0faf0" }}
        >
            {/* Background gradient */}
            <div className="absolute inset-0 overflow-hidden">
                <div className="absolute inset-0">
                    {/* Світло-зелений градієнт зверху */}
                    <div
                        className="absolute top-0 left-1/2 -translate-x-1/2 w-full h-full"
                        style={{
                            background:
                                "radial-gradient(ellipse at top, rgba(134,239,172,0.4) 0%,
rgba(187,247,208,0.2) 45%, transparent 100%)",
                        }}
                    />
                    {/* Рожевий акцент знизу праворуч */}
                    <div
                        className="absolute bottom-0 right-0 w-2/3 h-2/3"
                        style={{
                            background:
```

```
                                "radial-gradient(ellipse       at       bottom       right,
rgba(249,168,212,0.35) 0%, rgba(253,213,236,0.2) 50%, transparent 100%)",
                        }}
                    />
                    {/* М'який рожевий зліва */}
                    <div
                        className="absolute top-1/3 left-0 w-1/2 h-1/2"
                        style={{
                            background:
                                "radial-gradient(ellipse at left, rgba(244,194,218,0.25) 0%,
transparent 70%)",
                        }}
                    />
                </div>
            </div>

            <div className="relative z-50 pt-20">
                <Navbar />
                <Routes>
                    <Route path="/" element={<HomePage />} />
                    <Route
                        path="/signup"
                        element={!user ? <SignUpPage /> : <Navigate to="/" />}
                    />
                    <Route
                        path="/login"
                        element={!user ? <LoginPage /> : <Navigate to="/" />}
                    />
                    <Route
                        path="/secret-dashboard"
                        element={
                            user?.role === "admin" ? (
                                <AdminPage />
                            ) : (
                                <Navigate to="/login" />
                            )
                        }
                    />
                    <Route
```

```
                    path="/category/:category"
                    element={<CategoryPage />}
                />
                <Route
                    path="/cart"
                    element={user ? <CartPage /> : <Navigate to="/login" />}
                />
                <Route
                    path="/purchase-success"
                    element={
                        user ? (
                            <PurchaseSuccessPage />
                        ) : (
                            <Navigate to="/login" />
                        )
                    }
                />
                <Route
                    path="/purchase-cancel"
                    element={
                        user ? (
                            <PurchaseCancelPage />
                        ) : (
                            <Navigate to="/login" />
                        )
                    }
                />
            </Routes>
        </div>
        <Toaster />
    </div>
  );
}

export default App;
```

useUserStore.js

```
import { create } from "zustand";
```

```
import axios from "../lib/axios";
import { toast } from "react-hot-toast";

export const useUserStore = create((set, get) => ({
    user: null,
    loading: false,
    checkingAuth: true,

    signup: async ({ name, email, password, confirmPassword }) => {
        set({ loading: true });

        if (password !== confirmPassword) {
            set({ loading: false });
            return toast.error("Passwords do not match");
        }

        try {
            const res = await axios.post("/auth/signup", {
                name,
                email,
                password,
            });
            set({ user: res.data, loading: false });
        } catch (error) {
            set({ loading: false });
            toast.error(error.response.data.message || "An error occurred");
        }
    },
    login: async (email, password) => {
        set({ loading: true });

        try {
            const res = await axios.post("/auth/login", { email, password });

            set({ user: res.data, loading: false });
        } catch (error) {
            set({ loading: false });
            toast.error(error.response.data.message || "An error occurred");
        }
```

```
    },

    logout: async () => {
        try {
            await axios.post("/auth/logout");
            set({ user: null });
        } catch (error) {
            toast.error(
                error.response?.data?.message ||
                    "An error occurred during logout",
            );
        }
    },

    checkAuth: async () => {
        set({ checkingAuth: true });
        try {
            const response = await axios.get("/auth/profile");
            set({ user: response.data, checkingAuth: false });
        } catch (error) {
            console.log(error.message);
            set({ checkingAuth: false, user: null });
        }
    },
    refreshToken: async () => {
        // Prevent multiple simultaneous refresh attempts
        if (get().checkingAuth) return;

        set({ checkingAuth: true });
        try {
            const response = await axios.post("/auth/refresh-token");
            set({ checkingAuth: false });
            return response.data;
        } catch (error) {
            set({ user: null, checkingAuth: false });
            throw error;
        }
    },
}));
```

```
let refreshPromise = null;

axios.interceptors.response.use(
    (response) => response,
    async (error) => {
        const originalRequest = error.config;
        if (error.response?.status === 401 && !originalRequest._retry) {
            originalRequest._retry = true;

            try {
                if (refreshPromise) {
                    await refreshPromise;
                    return axios(originalRequest);
                }

                refreshPromise = useUserStore.getState().refreshToken();
                await refreshPromise;
                refreshPromise = null;

                return axios(originalRequest);
            } catch (refreshError) {
                useUserStore.getState().logout();
                return Promise.reject(refreshError);
            }
        }
        return Promise.reject(error);
    },
);
```

useCartStore.js

```
import { create } from "zustand";
import axios from "../lib/axios";
import { toast } from "react-hot-toast";

export const useCartStore = create((set, get) => ({
    cart: [],
    coupon: null,
```

```
total: 0,
subtotal: 0,
isCouponApplied: false,

receiveMyCoupon: async () => {
    try {
        const response = await axios.get("/coupons");
        set({ coupon: response.data });
    } catch (error) {
        console.error("Error fetching coupon:", error);
    }
},
useMyCoupon: async (code) => {
    try {
        const response = await axios.post("/coupons/validate", { code });
        set({ coupon: response.data, isCouponApplied: true });
        get().calculateMyTotal();
        toast.success("Coupon applied successfully");
    } catch (error) {
        toast.error(
            error.response?.data?.message || "Failed to apply coupon",
        );
    }
},
deleteCoupon: () => {
    set({ coupon: null, isCouponApplied: false });
    get().calculateMyTotal();
    toast.success("Coupon removed");
},

receiveCartItems: async () => {
    try {
        const res = await axios.get("/cart");
        set({ cart: res.data });
        get().calculateMyTotal();
    } catch (error) {
        set({ cart: [] });
        toast.error(error.response.data.message || "An error occurred");
    }
```

```
},
emptyCart: async () => {
    try {
        await axios.delete("/cart");

        set({
            cart: [],
            coupon: null,
            total: 0,
            subtotal: 0,
            isCouponApplied: false,
        });
    } catch (error) {
        console.error("Error clearing cart:", error);
        toast.error("Failed to clear cart");
    }
},
addItemToCart: async (product) => {
    try {
        await axios.post("/cart", { productId: product._id });
        toast.success("Product added to cart");

        set((prevState) => {
            const existingItem = prevState.cart.find(
                (item) => item._id === product._id,
            );
            const newCart = existingItem
                ? prevState.cart.map((item) =>
                      item._id === product._id
                          ? { ...item, quantity: item.quantity + 1 }
                          : item,
                  )
                : [...prevState.cart, { ...product, quantity: 1 }];
            return { cart: newCart };
        });
        get().calculateMyTotal();
    } catch (error) {
        toast.error(error.response.data.message || "An error occurred");
    }
```

```
    },
    deleteFromCart: async (productId) => {
        await axios.delete(`/cart`, { data: { productId } });
        set((prevState) => ({
            cart: prevState.cart.filter((item) => item._id !== productId),
        }));
        get().calculateMyTotal();
    },
    updateMyQuantity: async (productId, quantity) => {
        if (quantity === 0) {
            get().deleteFromCart(productId);
            return;
        }

        await axios.put(`/cart/${productId}`, { quantity });
        set((prevState) => ({
            cart: prevState.cart.map((item) =>
                item._id === productId ? { ...item, quantity } : item,
            ),
        }));
        get().calculateMyTotal();
    },
    calculateMyTotal: () => {
        const { cart, coupon } = get();
        const subtotal = cart.reduce(
            (sum, item) => sum + item.price * item.quantity,
            0,
        );
        let total = subtotal;

        if (coupon) {
            const discount = subtotal * (coupon.discountPercentage / 100);
            total = subtotal - discount;
        }

        set({ subtotal, total });
    },
}));
```

useProductStore.js

```
import { create } from "zustand";
import toast from "react-hot-toast";
import axios from "../lib/axios";

export const useProductStore = create((set) => ({
    products: [],
    loading: false,

    setProducts: (products) => set({ products }),
    createNewProduct: async (productData) => {
        set({ loading: true });
        try {
            const res = await axios.post("/products", productData);
            set((prevState) => ({
                products: [...prevState.products, res.data],
                loading: false,
            }));
        } catch (error) {
            toast.error(error.response.data.error);
            set({ loading: false });
        }
    },
    fetchAllProducts: async () => {
        set({ loading: true });
        try {
            const response = await axios.get("/products");
            set({ products: response.data.products, loading: false });
        } catch (error) {
            set({ error: "Failed to fetch products", loading: false });
            toast.error(
                error.response.data.error || "Failed to fetch products",
            );
        }
    },
    fetchProductsByCategory: async (category) => {
        set({ loading: true });
        try {
```

```
            const response = await axios.get(`/products/category/${category}`);
            set({ products: response.data.products, loading: false });
        } catch (error) {
            set({ error: "Failed to fetch products", loading: false });
            toast.error(
                error.response.data.error || "Failed to fetch products",
            );
        }
    },
    removeProduct: async (productId) => {
        set({ loading: true });
        try {
            await axios.delete(`/products/${productId}`);
            set((prevProducts) => ({
                products: prevProducts.products.filter(
                    (product) => product._id !== productId,
                ),
                loading: false,
            }));
        } catch (error) {
            set({ loading: false });
            toast.error(
                error.response.data.error || "Failed to delete product",
            );
        }
    },
    toggleStarredProduct: async (productId) => {
        set({ loading: true });
        try {
            const response = await axios.patch(`/products/${productId}`);
            // this will update the isFeatured prop of the product
            set((prevProducts) => ({
                products: prevProducts.products.map((product) =>
                    product._id === productId
                        ? { ...product, isFeatured: response.data.isFeatured }
                        : product,
                ),
                loading: false,
            }));
```

```
        } catch (error) {
            set({ loading: false });
            toast.error(
                error.response.data.error || "Failed to update product",
            );
        }
    },
    fetchFeaturedProducts: async () => {
        set({ loading: true });
        try {
            const response = await axios.get("/products/featured");
            set({ products: response.data, loading: false });
        } catch (error) {
            set({ error: "Failed to fetch products", loading: false });
            console.log("Error fetching featured products:", error);
        }
    },
}));
```

OrderSummary.jsx

```
import { motion } from "framer-motion";
import { useCartStore } from "../stores/useCartStore";
import { Link } from "react-router-dom";
import { ArrowRight, ShoppingBag } from "lucide-react";
import { loadStripe } from "@stripe/stripe-js";
import axios from "../lib/axios";

const stripePromise = loadStripe(

"pk_test_51TVVgSDfVHQowCb5eznkJbfzAuGVaOTFvse35vBtGldXYyADgvhzUbnWZNZ9CMFLhtcSSReCQfzqbU7uBl
RBrHXY00pKYQuhMI",
);

const OrderSummary = () => {
    const { total, subtotal, coupon, isCouponApplied, cart } = useCartStore();

    const savings = subtotal - total;
```

```
const formattedSubtotal = subtotal.toFixed(2);
const formattedTotal = total.toFixed(2);
const formattedSavings = savings.toFixed(2);

const handlePayment = async () => {
    try {
        const res = await axios.post("/payments/create-checkout-session", {
            products: cart,
            couponCode: coupon ? coupon.code : null,
        });

        window.location.href = res.data.url;
    } catch (error) {
        console.error("Payment error:", error);
    }
};

return (
    <motion.div
        className="bg-white border-[1.5px] border-gray-100 rounded-2xl p-4 mt-6"
        initial={{ opacity: 0, y: 10 }}
        animate={{ opacity: 1, y: 0 }}
        transition={{ duration: 0.4 }}
    >
        <h2 className="text-[15px] font-semibold text-gray-800 mb-4">
            Order summary
        </h2>

        <div className="space-y-2.5 mb-4">
            <div className="flex justify-between items-center">
                <span className="text-sm text-gray-500">
                    Subtotal ({cart.length}{" "}
                    {cart.length === 1 ? "item" : "items"})
                </span>
                <span className="text-sm font-medium text-gray-800">
                    ${formattedSubtotal}
                </span>
            </div>
```

```
                {savings > 0 && (
                    <div className="flex justify-between items-center">
                        <span className="text-sm text-pink-500">Savings</span>
                        <span className="text-sm font-medium text-pink-500">
                            -${formattedSavings}
                        </span>
                    </div>
                )}

                {coupon && isCouponApplied && (
                    <div className="flex justify-between items-center">
                        <span className="text-sm text-emerald-600 flex items-center gap-1.5">
                            Coupon
                            <span  className="text-[11px]  font-medium  bg-emerald-50  border
border-emerald-100 text-emerald-700 px-1.5 py-0.5 rounded-full">
                                {coupon.code}
                            </span>
                        </span>
                        <span className="text-sm font-medium text-emerald-600">
                            -{coupon.discountPercentage}%
                        </span>
                    </div>
                )}

                <div className="flex justify-between items-center">
                    <span className="text-sm text-gray-500">Shipping</span>
                    <span className="text-sm font-medium text-emerald-600">
                        Free
                    </span>
                </div>
            </div>

            <div className="border-t border-gray-50 pt-3.5 flex justify-between items-center
mb-4">
                <span className="text-sm font-semibold text-gray-800">
                    Total
                </span>
                <span className="text-xl font-bold text-emerald-600">
                    ${formattedTotal}
```

```
                </span>
            </div>

            <button
                onClick={handlePayment}
                className="w-full flex items-center justify-center gap-2 py-2.5 rounded-xl
text-sm font-medium text-white bg-gradient-to-r from-emerald-500 to-emerald-600
hover:opacity-90 hover:-translate-y-px transition-all duration-200"
            >
                <ShoppingBag size={14} />
                Place order
                <ArrowRight size={14} />
            </button>

            <div className="flex items-center justify-center gap-1.5 mt-3">
                <span className="text-[11px] text-gray-400">or</span>
                <Link
                    to="/"
                    className="text-[11px] font-medium text-emerald-600 hover:text-emerald-
700 flex items-center gap-1 transition-colors duration-200"
                >
                    Continue shopping
                    <ArrowRight size={11} />
                </Link>
            </div>

            <p className="text-center text-[11px] text-gray-400 mt-2">
                Dispatched fresh every Monday
            </p>
        </motion.div>
    );
};

export default OrderSummary;
```

AnalyticsTab.jsx

```
import { motion } from "framer-motion";
```

```
import { useEffect, useState } from "react";
import axios from "../lib/axios";
import { Users, Package, ShoppingCart, DollarSign } from "lucide-react";
import {
    LineChart,
    Line,
    XAxis,
    YAxis,
    CartesianGrid,
    Tooltip,
    Legend,
    ResponsiveContainer,
} from "recharts";

const AnalyticsTab = () => {
    const [analyticsData, setAnalyticsData] = useState({
        users: 0,
        products: 0,
        totalSales: 0,
        totalRevenue: 0,
    });
    const [isLoading, setIsLoading] = useState(true);
    const [dailySalesData, setDailySalesData] = useState([]);

    useEffect(() => {
        const fetchAnalyticsData = async () => {
            try {
                const response = await axios.get("/analytics");
                setAnalyticsData(response.data.analyticsData);
                setDailySalesData(response.data.dailySalesData);
            } catch (error) {
                console.error("Error fetching analytics data:", error);
            } finally {
                setIsLoading(false);
            }
        };
        fetchAnalyticsData();
    }, []);
```

```
if (isLoading) {
    return (
        <div className="text-sm text-gray-400 text-center py-8">
            Loading…
        </div>
    );
}

return (
    <motion.div
        className="max-w-4xl mx-auto py-2"
        initial={{ opacity: 0, y: 16 }}
        animate={{ opacity: 1, y: 0 }}
        transition={{ duration: 0.4 }}
    >
        {/* Metrics grid */}
        <div className="grid grid-cols-2 lg:grid-cols-4 gap-3 mb-6">
            <AnalyticsCard
                title="Total users"
                value={analyticsData.users.toLocaleString()}
                icon={Users}
                color="emerald"
            />
            <AnalyticsCard
                title="Total products"
                value={analyticsData.products.toLocaleString()}
                icon={Package}
                color="emerald"
            />
            <AnalyticsCard
                title="Total sales"
                value={analyticsData.totalSales.toLocaleString()}
                icon={ShoppingCart}
                color="emerald"
            />
            <AnalyticsCard
                title="Total revenue"
                value={`$${analyticsData.totalRevenue.toLocaleString()}`}
                icon={DollarSign}
```

```
            color="emerald"
        />
    </div>

    {/* Chart */}
    <motion.div
        className="bg-white rounded-2xl border-[1.5px] border-gray-100 p-4"
        initial={{ opacity: 0, y: 10 }}
        animate={{ opacity: 1, y: 0 }}
        transition={{ duration: 0.4, delay: 0.1 }}
    >
        <h3 className="text-sm font-semibold text-gray-800 mb-4">
            Sales & revenue
        </h3>
        <ResponsiveContainer width="100%" height={320}>
            <LineChart data={dailySalesData}>
                <CartesianGrid
                    strokeDasharray="3 3"
                    stroke="#f3f4f6"
                    vertical={false}
                />
                <XAxis
                    dataKey="date"
                    stroke="#9ca3af"
                    style={{ fontSize: "12px" }}
                />
                <YAxis
                    yAxisId="left"
                    stroke="#9ca3af"
                    style={{ fontSize: "12px" }}
                />
                <YAxis
                    yAxisId="right"
                    orientation="right"
                    stroke="#9ca3af"
                    style={{ fontSize: "12px" }}
                />
                <Tooltip
                    contentStyle={{
```

```
                            background: "#fff",
                            border: "1px solid #e5e7eb",
                            borderRadius: "8px",
                        }}
                        labelStyle={{ color: "#111827" }}
                    />
                    <Legend
                        wrapperStyle={{
                            fontSize: "12px",
                            paddingTop: "16px",
                        }}
                    />
                    <Line
                        yAxisId="left"
                        type="monotone"
                        dataKey="sales"
                        stroke="#10b981"
                        strokeWidth={2}
                        activeDot={{ r: 6 }}
                        name="Sales"
                        dot={false}
                    />
                    <Line
                        yAxisId="right"
                        type="monotone"
                        dataKey="revenue"
                        stroke="#db2777"
                        strokeWidth={2}
                        activeDot={{ r: 6 }}
                        name="Revenue"
                        dot={false}
                    />
                </LineChart>
            </ResponsiveContainer>
        </motion.div>
    </motion.div>
  );
};
```

```
export default AnalyticsTab;

const AnalyticsCard = ({ title, value, icon: Icon, color }) => {
    const colorMap = {
        emerald: {
            bg: "bg-emerald-50",
            border: "border-emerald-100",
            text: "text-emerald-700",
            icon: "text-emerald-500",
        },
    };
    const styles = colorMap[color];

    return (
        <motion.div
            className={`${styles.bg}  rounded-xl  border-[1.5px]  ${styles.border}  p-3  flex
flex-col justify-between`}
            initial={{ opacity: 0, y: 8 }}
            animate={{ opacity: 1, y: 0 }}
            transition={{ duration: 0.3 }}
        >
            <div className="flex items-start justify-between">
                <div>
                    <p
                        className={`text-[11px] font-medium ${styles.text} mb-1`}
                    >
                        {title}
                    </p>
                    <h3 className="text-lg font-bold text-gray-800">{value}</h3>
                </div>
                <Icon size={18} className={styles.icon} />
            </div>
        </motion.div>
    );
};
```