Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп’ютерних наук

Допущено до захисту
Завідувач кафедри комп’ютерних наук
доктор технічних наук, професор
Андрій БОНДАРЧУК
« 01 » червня 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр»
спеціальність 122 Комп’ютерні науки
освітня програма 122.00.01 Інформатика

Тема: ІНТЕРАКТИВНИЙ ЗАСТОСУНОК ІНТЕРНЕТ-МАГАЗИНУ З РЕАЛІЗАЦІЇ ПРОДУКТІВ ХАРЧУВАННЯ

Виконав
студент групи Інб-1-22-4.0д
Кравченко Кирило Андрійович

_____
(підпис)

Науковий керівник
кандидат технічних наук, доцент
Яскевич Владислав Олександрович

(підпис)

Київ – 2026

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп’ютерних наук

«Затверджую»
Завідувач кафедри комп’ютерних наук
доктор технічних наук, професор
Андрій БОНДАРЧУК
31.10.2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА «ІНТЕРАКТИВНИЙ ЗАСТОСУНОК ІНТЕРНЕТ-МАГАЗИНУ З РЕАЛІЗАЦІЇ ПРОДУКТІВ ХАРЧУВАННЯ»

Виконавець – студент групи ІНб-1-22-4.0д,
спеціальності 122 Комп’ютерні науки,
освітньої програми 122.00.01 Інформатика
Кравченко Кирило Андрійович

1. Вихідні дані: технічна документація та специфікації сучасних вебтехнологій і програмних бібліотек, Матеріали щодо інтеграції зовнішніх систем, механізмів автентифікації та cookie-based authentication, наукові публікації й методичні матеріали з питань створення вебзастосунків електронної комерції та інтеграції сервісів штучного інтелекту.

2. Основні завдання: здійснити аналіз предметної області та існуючих рішень інтернет-магазинів; обґрунтувати мету, об’єкт, предмет і методи дослідження; спроєктувати архітектуру вебзастосунку на основі клієнт-серверної моделі; розробити модулі застосунку, структуру бази даних і механізми інтеграції із зовнішніми сервісами; провести тестування функціональності, аналіз продуктивності, надійності та безпеки системи.

3. Пояснювальна записка: обсяг до 60 стор. формату А4 комп’ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.

4. Графічні матеріали: комп'ютерна презентація доповіді.

5. Додатки: фрагменти вихідного програмного коду, приклади реалізації серверних API-запитів, структура бази даних, конфігураційні файли програмного забезпечення, результати тестування системи та інструкція користувача щодо роботи з вебзастосунком.

6. Строк подання роботи на кафедру: 01.06.26

Науковий керівник
к.т.н., доцент
_____Яскевич В.О.
31.10.25 дата

Виконавець:
_____ Кравченко К.А.
31.10.25 дата

Календарний план

№	Етап виконання роботи	Термін виконання	Примітка
1	Затвердження теми кваліфікаційної роботи бакалавра	31.10.25	виконано
2	Аналіз проблеми, вивчення аналогів, формування цілей і завдань	01.11.25 - 11.01.26	виконано
3	Аналіз сучасних систем електронної комерції, існуючих архітектурних підходів до побудови веб-орієнтованих систем та обґрунтування вибору технологічного стеку (MERN)	26.01.26 – 15.03.26	виконано
4	Дослідження та проєктування загальної архітектури програмної системи, структури документоорієнтованої бази даних, REST API, моделей взаємодії користувачів та механізмів авторизації і безпеки.	16.03.26- 31.03.26	виконано
5	Розробка клієнтської (SPA) та серверної частин вебзастосунку, реалізація функціоналу управління каталогом і замовленнями, а також інтеграція AI-модуля та зовнішніх хмарних сервісів (Stripe, Cloudinary).	01.04.26 – 15.04.26	виконано
6	Тестування основних функціональних сценаріїв (End-to-End), перевірка механізмів безпеки (Security Testing), UI/UX тестування, а також аналіз продуктивності та оцінка алгоритмічної ефективності системи.	16.04.26 – 30.04.26	виконано
7	Впровадження розробленого програмного продукту, фінальне налаштування взаємодії з хмарною інфраструктурою та підготовка системи інтернет-магазину до дослідної експлуатації.	01.05.26 – 15.05.26	виконано
8	Підготовка пояснювальної записки, графічних матеріалів, додатків.	25.05.25 - 12.06.26	виконано
9	Захист кваліфікаційної роботи	19.06.26	виконано

«Затверджую»
Науковий керівник
к.т.н., доцент, Яскевич В.О.

Індивідуальний план склав
Кравченко К.А.

РЕФЕРАТ

Кваліфікаційна робота бакалавра: 86 с., 32 рис., 63 джерело, 3 додатки.

Актуальність: у сучасних умовах розвитку електронної комерції значно зростає попит на онлайн-сервіси для придбання товарів через мережу Інтернет. Інтернет-магазини стають основним каналом взаємодії між продавцями та покупцями, забезпечуючи швидкий доступ до товарів, можливість дистанційного оформлення замовлень та здійснення безпечних онлайн-платежів. Водночас важливим є створення ефективних, надійних та масштабованих вебзастосунків, які забезпечують зручний інтерфейс користувача, захист персональних даних, інтеграцію із зовнішніми сервісами та підтримку сучасних технологій, зокрема систем рекомендацій і багатомовності. Тому розробка вебзастосунку інтернет-магазину Greencart з використанням сучасних вебтехнологій та інтеграцією платіжних і хмарних сервісів є актуальною задачею у сфері інформаційних технологій.

Об'єкт дослідження: процеси функціонування інформаційних систем електронної комерції та взаємодія користувачів із вебзастосунком інтернет-магазину.

Предмет дослідження: архітектура та програмна реалізація клієнт-серверної системи інтернет-магазину з механізмами безпечної взаємодії та обробки замовлень.

Мета роботи: розробка архітектури та програмна реалізація вебзастосунку інтернет-магазину з використанням сучасних вебтехнологій, що забезпечує можливість перегляду каталогу товарів, оформлення замовлень, здійснення онлайн-оплати та управління товарами через адміністративну панель.

Завдання дослідження:

- проаналізувати сучасні системи електронної комерції;
- дослідити архітектурні підходи до створення вебсистем;
- спроєктувати структуру бази даних інтернет-магазину;
- розробити REST API для взаємодії компонентів системи;

- реалізувати клієнтську та серверну частину вебзастосунку;
- провести тестування функціональних сценаріїв роботи системи;
- дослідити та реалізувати систему рекомендацій товарів.

Основні результати: розроблений вебзастосунок інтернет-магазину, який забезпечує повний цикл роботи з товарами та замовленнями. Реалізована клієнтська частина застосунку з використанням сучасних технологій веброзробки та серверну частину, що забезпечує обробку запитів користувачів і взаємодію з базою даних. У застосунку реалізовано механізми автентифікації користувачів, управління каталогом товарів, формування замовлень і здійснення онлайн-оплати через інтеграцію з платіжною системою. Забезпечено можливість завантаження та зберігання зображень товарів у хмарному середовищі, а також підтримку багатомовного інтерфейсу користувача та функцій рекомендації товарів.

Практичне значення дослідження: створенні працездатної веборієнтованої системи електронної комерції, що забезпечує реалізацію основних бізнес-процесів інтернет-магазину та може бути використана як основа для подальшого розширення функціональності.

Ключові слова: вебзастосунок, інтернет-магазин, електронна комерція, клієнт-серверна архітектура, react, node.js, mongodb, rest api, stripe, cloudinary.

ЗМІСТ

Вступ

Актуальність теми. У сучасному цифровому суспільстві електронна комерція стала одним із ключових напрямів розвитку інформаційних технологій [1]. З поширенням мережі Інтернет та мобільних пристроїв значна частина торгівлі перемістилася в онлайн-простір. Це дозволяє користувачам швидко знаходити необхідні товари, порівнювати ціни, оформлювати замовлення та здійснювати оплату без фізичного відвідування магазинів.

Системи електронної комерції представляють собою складні програмні комплекси, які забезпечують взаємодію між продавцями та покупцями через веб-інтерфейс. Такі системи включають модулі керування товарами, обробки замовлень, інтеграції платіжних сервісів, а також системи аналітики та рекомендацій.

З розвитком веб-технологій та хмарних сервісів сучасні інтернет-магазини будуються за клієнт-серверною архітектурою та використовують REST API [2] для взаємодії між компонентами системи. Крім того, дедалі більшого поширення набувають системи рекомендацій, що дозволяють автоматично підбирати товари відповідно до інтересів користувачів, тому вивчення та розуміння цієї технології є доцільною.

Об’єкт дослідження: процеси функціонування інформаційних систем електронної комерції та взаємодія користувачів із вебзастосунком інтернет-магазину.

Предмет дослідження: архітектура та програмна реалізація клієнт-серверної системи інтернет-магазину з механізмами безпечної взаємодії та обробки замовлень.

Мета дослідження: розробка веб-орієнтованої системи інтернет-магазину продуктів харчування з використанням сучасних технологій веб-розробки.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати сучасні системи електронної комерції;
- дослідити архітектурні підходи до створення веб-систем;

- спроєктувати структуру бази даних інтернет-магазину;
- розробити REST API для взаємодії компонентів системи;
- реалізувати клієнтську та серверну частину вебзастосунку;
- провести тестування функціональних сценаріїв роботи системи;
- дослідити та реалізувати систему рекомендацій товарів.

Методи дослідження: системний аналіз, аналіз наукових та технічних джерел, об’єктно-орієнтоване моделювання, методи вебпроєктування та тестування програмного забезпечення.

Практичне значення одержаних результатів. Створення працездатної веборієнтованої системи електронної комерції, що забезпечує реалізацію основних бізнес-процесів інтернет-магазину та може бути використана як основа для подальшого розширення функціональності

Деякі результати дослідження доповідалися на студентській науковій конференції: XIII Всеукраїнська науково-практична конференція молодих науковців «ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ - 2026» (ІТ-2026) (м. Київ). Опубліковані у матеріалах конференції [3].

РОЗДІЛ 1
АНАЛІЗ СУЧАСНИХ СИСТЕМ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

1.1 Поняття та особливості систем електронної комерції

Електронна комерція є одним із ключових напрямів цифрової трансформації сучасного бізнесу та охоплює процеси купівлі, продажу, просування товарів і послуг, обробки замовлень, здійснення електронних платежів та взаємодії з клієнтами через мережу Інтернет [4]. На відміну від традиційної торгівлі, електронна комерція передбачає використання програмних систем, які забезпечують дистанційний доступ користувачів до товарів, автоматизовану обробку бізнес-процесів і підтримку електронних транзакцій.

Однією з найпоширеніших моделей електронної комерції є B2C, тобто взаємодія між бізнесом і кінцевим споживачем. У межах цієї моделі компанія або продавець пропонує товари чи послуги безпосередньо покупцеві через вебсайт, мобільний застосунок або іншу цифрову платформу [5]. Саме B2C-модель найчастіше використовується в інтернет-магазинах, оскільки вона передбачає наявність електронного каталогу товарів, кошика, особистого кабінету користувача, системи оформлення замовлень, оплати та доставки.

Системи електронної комерції зазвичай мають багаторівневу архітектуру, що включає клієнтську частину, серверну логіку, базу даних та зовнішні сервіси. Клієнтська частина відповідає за взаємодію користувача з вебзастосунком: перегляд каталогу, пошук і фільтрацію товарів, додавання позицій до кошика та оформлення замовлення. Серверна частина забезпечує обробку запитів, реалізацію бізнес-логіки, автентифікацію користувачів, перевірку даних, керування замовленнями та взаємодію з базою даних.

Важливою складовою сучасних систем електронної комерції є інтеграція платіжних сервісів. Такі сервіси дають змогу здійснювати онлайн-оплату банківськими картками або іншими електронними способами, не зберігаючи чутливі платіжні дані безпосередньо в інформаційній системі інтернет-магазину. Для цього використовуються спеціалізовані платіжні платформи, які

забезпечують створення платіжних сесій, обробку транзакцій, підтвердження платежів і взаємодію із банківською інфраструктурою [6].

Окрему роль у функціонуванні інтернет-магазинів відіграє зберігання та обробка мультимедійного контенту. Оскільки товари зазвичай супроводжуються зображеннями, описами та іншими візуальними матеріалами, системи електронної комерції часто використовують хмарні сервіси для зберігання, оптимізації та доставки медіафайлів. Такі сервіси дозволяють зменшити навантаження на основний сервер, прискорити завантаження сторінок і забезпечити коректне відображення зображень на різних пристроях [7].

Безпека є однією з ключових вимог до систем електронної комерції, оскільки такі системи працюють із персональними даними користувачів, історією замовлень і платіжною інформацією. Для захисту даних застосовуються механізми автентифікації та авторизації, шифрування з’єднання, хешування паролів, валідація вхідних даних, розмежування прав доступу та захист від типових вебзагроз. Особливо важливим є відокремлення платіжного контуру від основної логіки інтернет-магазину, оскільки це зменшує ризики, пов’язані з обробкою фінансової інформації.

Крім базових торговельних функцій, сучасні системи електронної комерції дедалі частіше використовують додаткові інтелектуальні механізми. До них належать рекомендаційні системи, персоналізований пошук, аналіз поведінки користувачів, прогнозування попиту та автоматизація взаємодії з клієнтами. Такі можливості підвищують зручність користування інтернет-магазином і дозволяють адаптувати пропозиції до потреб конкретного користувача.

1.2 Архітектурні підходи до побудови веб-орієнтованих систем

Архітектура програмної системи визначає структурну організацію її компонентів, протоколи їхньої взаємодії та стратегії масштабування. Для систем електронної комерції архітектурні рішення безпосередньо впливають на показники продуктивності, відмовостійкості та ефективність інтеграції із зовнішніми сервісами.

Еволюція вебсистем характеризується переходом від монолітних структур до модульних та сервісно орієнтованих підходів. Монолітна архітектура, де інтерфейс та бізнес-логіка реалізовані в межах єдиного модуля, має обмеження щодо масштабованості та тестування. Натомість розподілені системи дозволяють ізолювати функціональні блоки, що спрощує супровід складних проєктів.

Базовим принципом побудови сучасних вебзастосунків є клієнт-серверна архітектура [15], яка передбачає логічний поділ на рівень представлення та рівень обробки даних. Такий підхід забезпечує незалежність розробки фронтенду і бекенду, дозволяючи повторно використовувати серверну логіку для різних типів клієнтів та розділяти програму на 2 незалежні складові та працювати у команді розробки ефективніше. У проєкті GreenCart цей принцип реалізовано через фізичний поділ коду на директорії client/ та server/, де взаємодія відбувається шляхом обміну JSON-даними через RESTful API.

Проєкт базується на багаторівневій архітектурі (Multi-tier Architecture) [16], що включає три основні шари:

Перший шар – це рівень представлення (Presentation Layer): реалізований на React, відповідає за рендеринг інтерфейсу та обробку подій користувача.

Другий шар – це івень бізнес-логіки (Logic Layer): контролери та сервіси Express.js, що забезпечують виконання функціональних сценаріїв.

І останній - рівень даних (Data Layer): СУБД MongoDB compass [17] та моделі Mongoose у node.js, які відповідають за збереження стану системи.

Архітектурний стиль REST (Representational State Transfer) використовується для побудови інтерфейсів взаємодії, де кожна сутність (товари, замовлення, користувачі) розглядається як окремий ресурс [3]. Система GreenCart використовує уніфіковані HTTP-методи (GET, POST, PUT, DELETE)[18] для маніпуляції ресурсами, що забезпечує високу сумісність із вебстандартами [19]. Структура ендпоінтів (наприклад, /api/product/list або /api/order/stripe) відображає ресурсну декомпозицію системи.

Реалізація клієнтської частини за моделлю Single Page Application (далі - SPA)[20] за допомогою бібліотеки React [21] дозволяє динамічно оновити інтерфейс без перезавантаження сторінок. Це мінімізує обсяг переданих даних та знижує затримки при взаємодії з каталогом чи кошиком. Використання NoSQL-бази даних MongoDB замість реляційних СУБД обумовлене гнучкістю документоорієнтованої моделі, що критично для систем із динамічною схемою товарних характеристик.

Для забезпечення масштабованості інтелектуальних функцій, AI-модуль GreenCart винесено в окремий функціональний блок (ендпоінт POST /api/ai/query). Це дозволяє ускладнювати алгоритми семантичного пошуку та рекомендацій без втручання в основну транзакційну логіку.

Компонентний підхід [22] до побудови фронтенду базується на принципах повторного використання коду та декомпозиції інтерфейсу на незалежні одиниці (у випадку Greencart - jsx файли react коду). Це спрощує тестування та розширення функціоналу, наприклад, при додаванні нових форм оплати чи адміністративних панелей. Безпека архітектури реалізується через розмежування публічних і захищених маршрутів із застосуванням JWT-авторизації та механізмів контролю доступу на рівні Middleware.

Таким чином, поєднання клієнт-серверної моделі, REST API та SPA-підходу в межах MERN-стеку [23] забезпечує необхідний баланс між продуктивністю системи та швидкістю розробки. Таке архітектурне рішення є оптимальним для сучасних B2C-платформ, оскільки дозволяє гнучко масштабувати систему відповідно до зростання навантаження.

1.3 Аналіз існуючих платформ інтернет-магазинів

Порівняльний аналіз архітектурних рішень [24] існуючих платформ дозволив визначити оптимальні технологічні підходи для побудови масштабованих веб-орієнтованих систем. Сучасні рішення для електронної комерції класифікуються за трьома основними моделями: SaaS-рішення, CMS-

платформи та кастомні full-stack системи. Кожна модель має специфічні обмеження щодо гнучкості архітектури та можливостей інтеграції.

CMS-орієнтовані рішення (OpenCart [25], Magento [26], WooCommerce [27]) базуються на модульній архітектурі, що дозволяє розширювати функціонал через плагіни. Проте надлишковість коду типових CMS негативно впливає на швидкість рендерингу та ускладнює супровід при глибокій кастомізації. Наприклад, Magento підтримує складні бізнес-процеси, але вимагає значних обчислювальних ресурсів та складної конфігурації серверного середовища. Використання CMS часто зміщує фокус розробки з проєктування архітектури на адаптацію існуючого шаблону під обмеження платформи.

SaaS-платформи (наприклад, Shopify[28]) забезпечують швидке розгортання торговельних майданчиків за моделлю «програмне забезпечення як послуга». Бізнес-орієнтованість таких систем дозволяє мінімізувати витрати на інфраструктуру, проте суттєво обмежує контроль над внутрішньою логікою, структурами даних та механізмами оптимізації продуктивності. Закритість вихідного коду та жорстка схема даних роблять неможливою реалізацію нетипових алгоритмів рекомендацій або специфічних моделей взаємодії, що критично для систем із високим ступенем інтелектуалізації.

Кастомні full-stack системи, що створюються з нуля, забезпечують максимальну відповідність специфічним вимогам проєкту. Такий підхід дозволяє обґрунтовано обрати архітектурні патерни, методи збереження даних та алгоритми обробки запитів. Для реалізації GreenCart було обрано стратегію кастомної розробки на базі MERN-стеку, що забезпечує:

- повний контроль над життєвим циклом об’єктів у MongoDB;
- проєктування гнучкого REST API на базі Express.js [29];
- реалізацію складних клієнтських сценаріїв за допомогою React;
- ізольовану інтеграцію AI-модуля для семантичного пошуку.

Функціональні відмінності GreenCart від типових «коробкових» рішень проявляються у підтримці специфічних інтелектуальних сценаріїв:

semantic_search, recipe_from_products та dinners_budget. Реалізація такого функціоналу в межах готових CMS вимагала б докорінної перебудови ядра системи. Кастомна архітектура дозволяє інтегрувати платіжні шлюзи Stripe та хмарні сервіси Cloudinary на рівні нативної логіки застосунку, мінімізуючи затримки (latency).

Важливим критерієм вибору є стратегія масштабування. На відміну від SaaS-рішень, кастомна система дозволяє впроваджувати вибіркове кешування, балансування навантаження та оптимізацію асимптотичної складності окремих API-запитів. Це відкриває можливості для дослідження поведінки системи при зростанні обсягу транзакцій та бази товарів.

Таким чином, розробка власної full-stack системи є найбільш доцільною для Greencart, бо вона потребує високої гнучкості, специфічної логіки та інтеграції інтелектуальних сервісів.

1.4 Технології розробки сучасних вебзастосунків

Ефективність систем електронної комерції визначається вибором технологічного стеку, який повинен забезпечувати високу пропускну здатність, масштабованість та надійність обробки транзакцій [15]. Сучасна веброзробка базується на розподілі функцій між клієнтською та серверною частинами, де кожен рівень використовує спеціалізовані фреймворки та інструменти.

Клієнтські частини (Frontend) будуються на базі екосистеми JavaScript [30]. Використання бібліотек React, Vue, Angular[21,31,32] реалізує декларативну модель побудови інтерфейсу через компонентний підхід, що дозволяє декомпозувати UI на незалежні модулі: картки товарів, кошик, блоки рекомендацій та панелі адміністрування. У проєкті GreenCart React-архітектура забезпечує архітектурний підхід Single Page Application (SPA), у межах якого синхронізація інтерфейсу з даними відбувається без повного перезавантаження(або ререндера, перемалювання) сторінок, що зменшує обсяг мережевого трафіку. Водночас початкове завантаження клієнтського застосунку

може потребувати більше часу порівняно з традиційними багатосторінковими вебзастосунками, але компенсується меншим навантаженням на мережу.

Для оптимізації розробки та збирання проєкту використано такі інструменти:

Vite [33]: сучасний інструмент збирання, що забезпечує швидке гаряче оновлення модулів (HMR) та ефективну мініфікацію коду для production-режиму.

Tailwind CSS [34]: утилітарний фреймворк для побудови адаптивного дизайну, що дозволяє стандартизувати стилі та зменшити об'єм CSS-файлів.

React Router [35]: бібліотека для організації клієнтської маршрутизації, що забезпечує навігацію між функціональними доменами системи (каталог, профілі, замовлення).

Axios [36]: клієнт для виконання асинхронних HTTP/s-запитів до REST API, що підтримує централізовану обробку помилок, автоматичну трансформацію JSON-даних та надає більш зручний програмний інтерфейс порівняно з використанням стандартного API fetch.

Серверна частина (Backend) реалізована на платформі Node.js [37], яка використовує неблокуючу модель введення-виведення, що є оптимальним для систем із великою кількістю одночасних запитів. Вебфреймворк Express.js забезпечує маршрутизацію та впровадження Middleware[38] для валідації запитів і керування доступом. В GreenCart серверна логіка декомпозована на окремі API-контролери за предметними областями: користувачі, товари, замовлення та адміністративні функції. Прикладом до цього може слугувати Додаток Б - реалізація Stripe контролера.

Рівень збереження даних базується на СУБД MongoDB compass та ODM-бібліотеці Mongoose [39]. Документоорієнтована модель дозволяє зберігати сутності зі складною вкладеною структурою (наприклад, замовлення з переліком продуктів та адресами доставки) у форматі BSON [40]. Використання Mongoose забезпечує сувору валідацію схем даних на рівні серверного коду, що підвищує цілісність інформаційної системи.

Безпека та автентифікація реалізовані через механізм JSON Web Tokens (JWT). Використання моделі безстанної (stateless) автентифікації у поєднанні з httpOnly cookies дозволяє захистити маркери доступу від XSS-атак та забезпечити контроль сесій на рівні Middleware.

Інтеграція зовнішніх сервісів є ключовим елементом функціональності GreenCart:

Stripe API використовується для обробки фінансових транзакцій, реалізуючи безпечний кліринг через захищені вебхуки.

Cloudinary забезпечує хмарне зберігання та динамічну оптимізацію зображень товарів, що розвантажує основний сервер застосунку.

AI-модуль оркестратора n8n виконує інтелектуальні функції семантичного пошуку та генерації рекомендацій на основі моделей глибокого навчання. Модуль інтегрований як окремий сервіс через API-ендпоінт.

Додатково в системі реалізовано механізми локалізації та мультивалютності завдяки API Нацбанку України [41], що вимагає динамічної адаптації інтерфейсу та обробки даних з урахуванням мови інтерфейсу користувача.

Таким чином, обраний технологічний стек (MERN) у поєднанні з інтегрованими сервісами Cloudinary та Stripe формує цілісне середовище для розробки сучасної, масштабованої системи електронної комерції, що відповідає вимогам програмної інженерії.

Висновки до розділу 1

На основі аналізу теоретичних та технологічних засад побудови систем електронної комерції зроблено такі висновки:

Специфіка інформаційної системи.

Сучасний інтернет-магазин формату B2C є складною розподіленою системою. Її функціонування вимагає не лише реалізації базових торговельних операцій, а й забезпечення високої надійності, інтеграції інтелектуальних

сервісів (AI-рекомендацій), підтримки багатомовності та захисту транзакційних даних.

Архітектурна модель.

Оптимальним підходом до побудови масштабованої системи є багаторівнева клієнт-серверна архітектура з використанням REST API. Такий розподіл логічно ізолює рівень представлення (SPA) від серверної обробки та бази даних, що підвищує модульність коду та спрощує подальший супровід системи.

Обґрунтування кастомної розробки.

Аналіз готових SaaS-платформ та CMS-рішень виявив їхню структурну обмеженість щодо впровадження нестандартної логіки (наприклад, семантичного пошуку чи планування бюджету покупок). Розробка власного full-stack вебзастосунку забезпечує необхідний рівень гнучкості, повний контроль над оптимізацією бази даних та можливість ізольованого масштабування окремих компонентів.

Вибір технологічного стеку.

Обґрунтовано доцільність використання MERN-стеку для реалізації проєкту. React і Tailwind CSS визначено як засоби для створення адаптивного та оптимізованого користувацького інтерфейсу. Node.js та Express.js обрано для побудови високопродуктивного серверного застосунку завдяки подієво-орієнтованій моделі роботи. MongoDB у поєднанні з ODM Mongoose забезпечує гнучке збереження сутностей зі змінною структурою у форматі BSON. Безпека та підтримка транзакційності досягаються шляхом використання безстанної JWT-автентифікації, а також інтеграції сервісів Stripe для обробки платежів і Cloudinary для роботи з медіаконтентом.

Отримані результати формують методологічну базу для розробки проєкту GreenCart. Визначені архітектурні підходи та технологічний стек будуть використані в наступному розділі під час проєктування моделі даних, розробки структури REST API та реалізації механізмів безпеки вебзастосунку.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ GREEN CART

2.1 Вимоги до системи

Для розроблення програмної системи GreenCart було визначено функціональні та нефункціональні вимоги. Функціональні вимоги описують основні можливості системи, які повинні бути реалізовані для забезпечення роботи інтернет-магазину продуктів харчування. Нефункціональні вимоги визначають якісні характеристики системи, зокрема продуктивність, безпеку, надійність, масштабованість та зручність використання.

Функціональні вимоги програмної системи GreenCart мають: забезпечення роботи з обліковими записами користувачів. Система повинна надавати можливість реєстрації нового користувача, авторизації за допомогою email та пароля, а також виходу із системи після завершення сеансу роботи.

Робота з каталогом товарів. Користувач повинен мати можливість переглядати список товарів, здійснювати пошук за назвою або категорією, а також відкривати картку товару для перегляду детальної інформації. Це забезпечує зручну навігацію інтернет-магазином та швидкий доступ до потрібних продуктів.

Робота з кошиком. Система повинна дозволяти користувачу додавати товари до кошика, видаляти їх, а також змінювати кількість обраних товарів. Після формування кошика користувач повинен мати можливість вказати адресу доставки та оформити замовлення.

Програмна система GreenCart також повинна підтримувати різні способи оплати. Зокрема, система має забезпечувати онлайн-оплату замовлення через сервіс Stripe, а також підтримувати можливість оплати при отриманні. Для коректної роботи платіжної підсистеми необхідно реалізувати обробку підтверджень платежів через webhook.

Для зареєстрованих користувачів має бути передбачена можливість перегляду історії власних замовлень. Крім того, система повинна підтримувати

перемикання мови інтерфейсу між українською та англійською, а також відображення цін у різних валютах.

Адміністративна частина системи має забезпечувати керування товарами та замовленнями. Адміністратор повинен мати можливість додавати нові товари, редагувати інформацію про них, видаляти товари, змінювати статус їх наявності, а також переглядати всі замовлення користувачів. Для товарів також має бути реалізовано завантаження та зберігання зображень.

Додатковою функціональною можливістю системи є робота AI-модуля. Він повинен забезпечувати рекомендації товарів і страв, що дозволяє покращити користувацький досвід та спростити пошук релевантних продуктів.

Нефункціональні вимоги до програмної системи GreenCart визначають якість її роботи. Наприклад, система повинна забезпечувати достатній рівень продуктивності, тобто обробляти запити користувача не більше ніж за 2-3 секунди. Також вона має бути масштабованою, щоб підтримувати збільшення кількості користувачів без зміни основної архітектури.

Важливою вимогою є надійність системи, тобто програмна система повинна забезпечувати стабільну роботу без втрати даних, коректно обробляти помилки сервера та підтримувати резервне копіювання інформації. Це дозволяє зменшити ризик втрати важливих даних у разі технічних збоїв.

Особлива увага приділяється безпеці. Система повинна використовувати механізми автентифікації користувачів та захисту їхніх даних. Паролі користувачів мають зберігатися у зашифрованому вигляді, а платіжні операції повинні виконуватися через захищений сервіс Stripe.

Система повинна бути доступною для користувачів у режимі 24/7 та сумісною із сучасними веббраузерами. Інтерфейс програмної системи має бути зрозумілим і зручним для користувача, що забезпечує належний рівень юзабіліті.

Для подальшої підтримки та розвитку GreenCart код системи повинен бути модульним і легко змінюваним. Система також має підтримувати можливість додавання нових модулів у майбутньому. Крім того, необхідно забезпечити

ведення журналу подій та помилок, що спрощує моніторинг роботи системи та пошук причин можливих збоїв.

2.2 Загальна архітектура програмної системи

Архітектура програмної системи формалізує організацію її компонентів, протоколи взаємодії та стратегії обробки даних. Для інтернет-магазину GreenCart архітектурне рішення (рис. 2.1) базується на забезпеченні масштабованості, модульності та безперебійної інтеграції із зовнішніми сервісами.

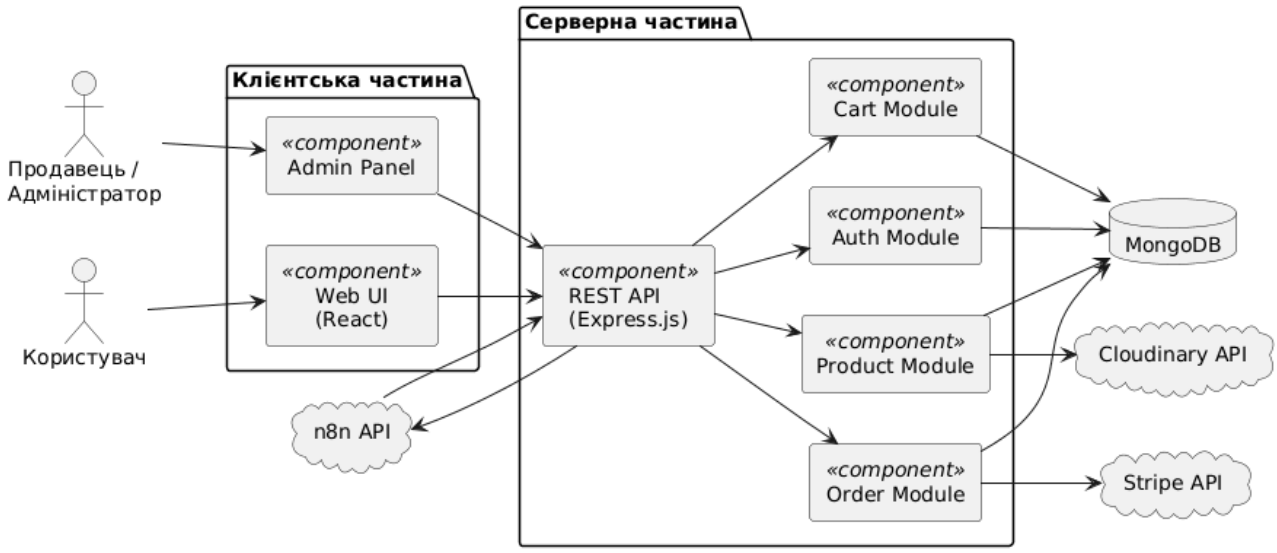

Рисунок 2.1. Структурна схема архітектури системи

Система реалізує багаторівневу клієнт-серверну архітектуру, яка фізично розділена на два незалежні компоненти: клієнтський та серверний застосунки.

Клієнтський рівень (Frontend) відповідає за рендеринг інтерфейсу за моделлю Single Page Application (SPA) та управління станом на стороні браузера. Цей рівень забезпечує інтерактивну взаємодію користувача з каталогом, кошиком, профілем і панеллю адміністратора. Клієнт генерує асинхронні HTTP-запити до API та динамічно оновлює DOM на основі отриманих відповідей.

Серверний рівень (Backend) реалізує логіку предметної області електронної комерції. Він виконує валідацію вхідних даних, автентифікацію, взаємодію з документоорієнтованою базою даних (MongoDB) та координацію

транзакцій. Сервер також виступає шлюзом для зовнішніх платформ (Stripe, Cloudinary), формуючи уніфіковані відповіді у форматі JSON.

Логічно архітектура декомпозована на такі незалежні підсистеми:

Модуль автентифікації та контролю доступу Identity and Access Management (Далі - IAM): забезпечує керування сесіями (через JWT) та реалізує рольову модель доступу (RBAC). Логіка жорстко розмежовує привілеї користувача (робота з каталогом і замовленнями) та адміністратора (керування контентом і моніторинг продажів).

Модуль керування каталогом: базовий домен системи, що інкапсулює CRUD-операції з товарами, категоріями та метаданими.

Модуль транзакцій (кошик та замовлення) управляє життєвим циклом покупки. Кошик функціонує як тимчасова сесійна структура, тоді як замовлення (Order) є персистентною сутністю, що фіксує фінансовий стан. Модуль підтримує розгалуження бізнес-логіки залежно від методу оплати (післяплата або онлайн-транзакція).

Модуль інтеграції зовнішніх сервісів ізолює роботу зі сторонніми API. Делегування обробки платежів сервісу Stripe усуває необхідність зберігання чутливих фінансових даних у власній базі, а використання Cloudinary як CDN оптимізує доставку медіаконтенту.

AI-модуль, який виділений у самостійний API-ендпоінт для забезпечення слабкого зв'язування компонентів. Модуль реалізує алгоритми семантичного пошуку та генерації рекомендацій без впливу на продуктивність основної транзакційної системи.

Взаємодія між клієнтом та сервером здійснюється за архітектурним стилем REST. Ця модель гарантує незалежність (stateless) обробки кожного запиту, що є критичним для горизонтального масштабування бекенду.

Архітектура системи також нативно підтримує механізми міжнародної адаптації (i18n) [42]. Локалізація інтерфейсу (українська/англійська мови) та мультивалютність реалізовані через динамічне кешування курсів з API нацбанку локалізовані пули текстових констант.

2.3 Проєктування структури бази даних

Проєктування бази даних (далі - БД) формалізує предметну область інтернет-магазину у вигляді структурованої моделі сутностей, забезпечуючи цілісність даних та ефективність операцій. Для реалізації GreenCart обрано документоорієнтовану СУБД MongoDB у зв’язці з ODM-бібліотекою Mongoose.

Вибір NoSQL-рішення обґрунтовано такими архітектурними чинниками:

Природне моделювання агрегатів: об'єкти електронної комерції (наприклад, замовлення) інкапсулюють вкладені масиви товарів та платіжні дані, що ідеально лягає на документоорієнтовану модель.

Гнучкість схеми (Schema-less): динамічний каталог товарів із локалізованими атрибутами та специфічними характеристиками не потребує складних міграцій бази даних при зміні структури.

Уніфікація форматів: структура BSON-документів[40] нативно узгоджується з JSON-відповідями REST API у середовищі MERN-стеку.

Фізична модель даних системи GreenCart декомпозована на чотири базові колекції (сутності):

1. User (Користувач): містить ідентифікаційні дані (email), хешований пароль та поточний стан кошика (cartItems). Зберігання кошика як вбудованого масиву (embedded array) в документі користувача мінімізує кількість звернень до БД під час сесії, усуваючи потребу у створенні окремої колекції для тимчасових даних.

2. Product (Товар): центральний напівструктурований об’єкт каталогу. Включає скалярні атрибути (ціна, категорія, статус inStock) та масиви (зображення, багатомовні описи). Нативна підтримка української та англійської локалізації інтегрована безпосередньо у схему документа, що оптимізує запити при багатомовному рендерингу.

3. Address (Адреса доставки): винесена у самостійну колекцію для збереження адрес користувачів. Таке відокремлення реалізує відношення «один-

до-багатьох» (1:N) між User та Address, дозволяючи користувачеві багаторазово використовувати збережені адреси без дублювання даних.

4. Order (Замовлення): агрегатна сутність, що фіксує факт транзакції. Містить масив придбаних товарів (з їхньою кількістю), загальну суму, статус (зокрема маркер isPaid), метод оплати (COD або ONLINE) та зафіксовану адресу доставки.

Програмні реалізації схем подані в Додатку А.

Модель зв'язків та збереження цілісності: при проєктуванні використано гібридний підхід до моделювання зв'язків (Embedding vs. Referencing). Зв’язок замовлення з профілем користувача реалізується через посилання (Reference), тоді як дані про товари та адресу на момент покупки копіюються (Embedding) у документ Order. Це архітектурно правильне рішення, оскільки воно гарантує історичну незмінність транзакції: подальша зміна ціни товару чи редагування адреси в профілі не вплине на вже зафіксовані чеки замовлень.

Індексування та аналітична придатність: для оптимізації швидкодії CRUD-операцій створено унікальний індекс для поля email у колекції User. Для пришвидшення фільтрації та роботи алгоритмів пошуку передбачено індексування полів category, name та inStock у колекції Product. Крім того, наявність структурованих локалізованих описів та чіткої категоризації формує якісний датасет, необхідний для роботи інтегрованого AI-модуля (семантичний пошук та генерація рекомендацій страв).

Запропонована структура даних (рис. 2.2) є масштабованою: вона забезпечує високу швидкість виконання операцій та допускає еволюційне розширення (наприклад, додавання колекцій відгуків, купонів чи журналу аудиту) без рефакторингу існуючого ядра системи.

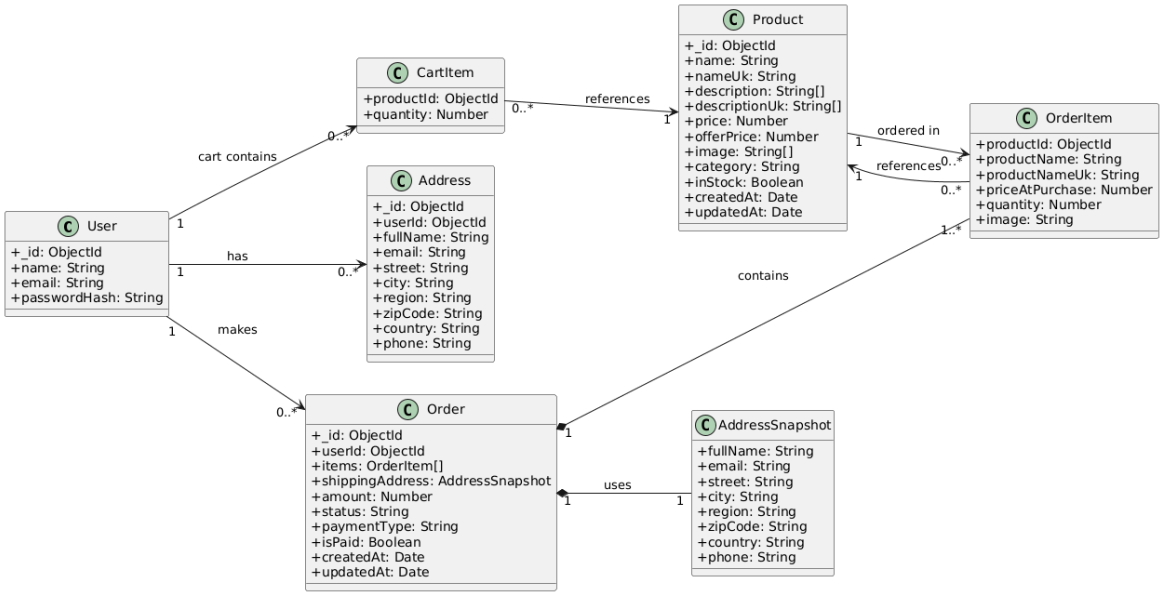

Рисунок 2.2. Модель класів для структури даних

2.4 Моделювання взаємодії користувача із системою

Моделювання взаємодії формалізує поведінкову модель системи (рис 2.3), пов'язуючи абстрактну архітектуру та модель даних із конкретними бізнес-процесами електронної комерції. Основою цієї моделі є ідентифікація акторів (ролей) та опис типових сценаріїв їхньої взаємодії з програмним продуктом.

У межах системи GreenCart виділено дві основні ролі: користувач та продавець. Користувач взаємодіє із системою через сценарії реєстрації, входу в систему, перегляду каталогу, пошуку товарів, додавання товарів до кошика, оформлення замовлення, онлайн-оплати та перегляду власних замовлень. Продавець, своєю чергою, має доступ до функцій керування товарами та замовленнями, зокрема додавання, редагування й видалення товарних позицій, перегляду замовлень та зміни їхнього статусу.

Діаграма варіантів використання демонструє, що частина дій доступна лише після авторизації користувача або продавця в системі. Це відповідає рольовій моделі доступу, за якої кожна категорія користувачів має власний набір дозволених операцій.

Рисунок 2.3. UML use case діаграма

У системі GreenCart реалізовано рольову модель доступу (RBAC), яка виділяє двох основних акторів із чітко розмежованими привілеями:

Користувач (Клієнт) взаємодіє з публічним інтерфейсом, споживаючи торговельні та інтелектуальні сервіси системи.

Адміністратор (Продавець) працює у захищеному контурі системи (адміністративній панелі), керуючи контентом та операційною діяльністю.

Функціональна модель системи для ролі користувача описує основні дії, які він може виконувати під час роботи з вебзастосунком. Передусім користувач має можливість зареєструватися або увійти до системи. Під час входу введені облікові дані перевіряються на сервері, після чого формується JWT-токен і створюється захищена сесія за допомогою httpOnly cookie. Це дає змогу користувачу отримувати доступ до власного кошика, адрес доставки та історії замовлень.

Після авторизації або без неї користувач може переглядати каталог товарів. Клієнтська частина застосунку надсилає запити до сервера для отримання списку продукції, а також дозволяє використовувати фільтрацію, пагінацію та відкривати сторінку окремого товару. На сторінці товару відображається детальна інформація: назва, опис, ціна, наявність і зображення.

Окремим сценарієм є робота з кошиком. Користувач може додавати товари, змінювати їх кількість або видаляти позиції. Дані кошика зберігаються на сервері, що дозволяє синхронізувати його стан між різними пристроями після входу в обліковий запис.

Також користувач може керувати адресами доставки: додавати нові адреси, редагувати вже збережені або видаляти непотрібні. Це спрощує процес оформлення замовлення, оскільки під час покупки можна швидко вибрати одну з раніше доданих адрес.

Основним сценарієм взаємодії є оформлення замовлення, яке подано на рис. 2.4. На цьому етапі користувач підтверджує склад кошика, обирає адресу доставки та спосіб оплати. Якщо вибрано післяплату, замовлення одразу зберігається в базі даних зі статусом активного. Якщо користувач обирає онлайн-оплату через Stripe, система створює платіжну сесію та перенаправляє його на захищену сторінку оплати. Після успішної транзакції статус оплати оновлюється автоматично через механізм Webhook [43].

Програмну реалізацію логіки створення Stripe Checkout Session та обробки онлайн-замовлення наведено в Додатку Б.

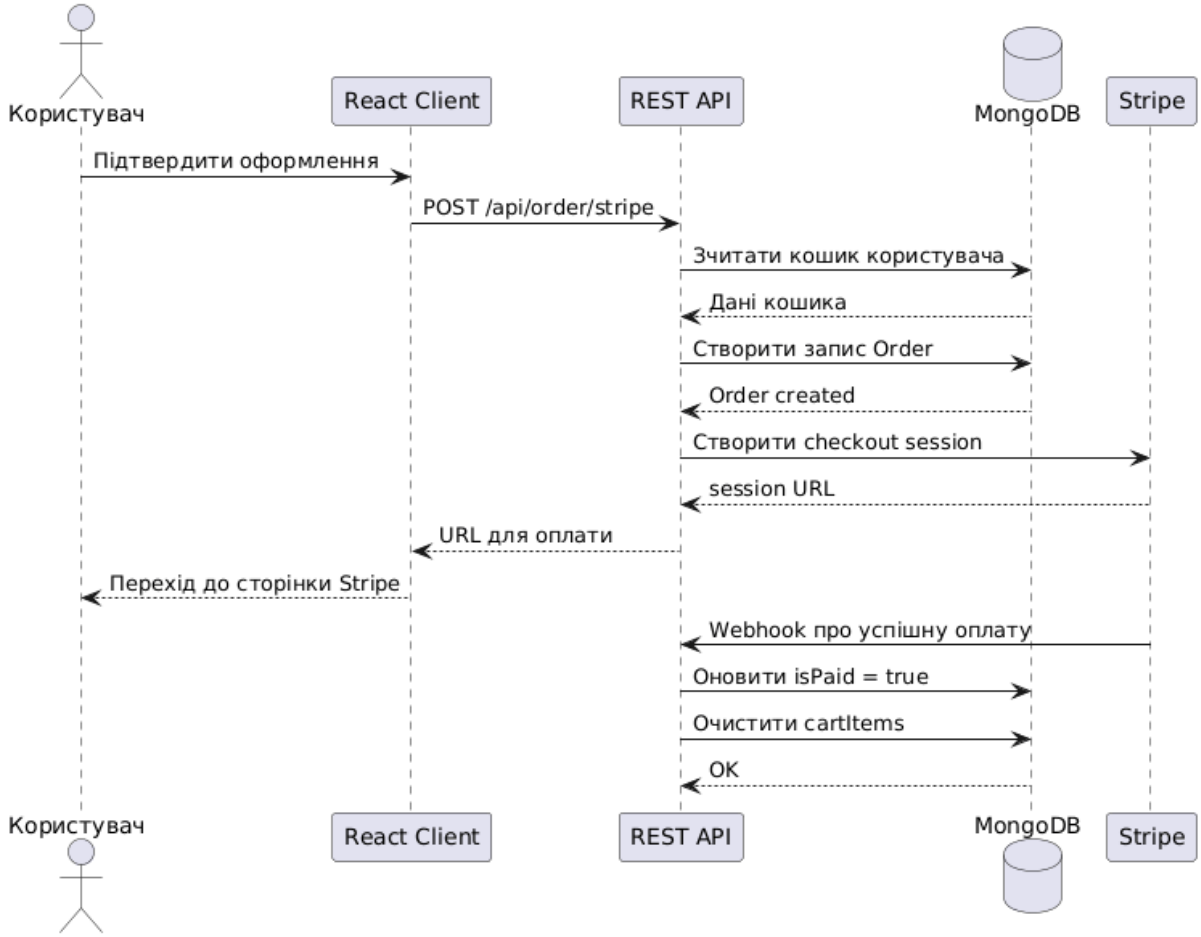

Рисунок 2.4. UML sequence діаграма для оформлення замовлення

Моніторинг замовлень.

Користувач отримує доступ до історії власних покупок (read-only доступ до колекції Order з фільтрацією за userId).

Взаємодія з AI-модулем.

користувач формулює запит природною мовою (наприклад, планування бюджету чи підбір інгредієнтів). Клієнт надсилає запит до окремого ендпоінта, який аналізує контекст і повертає структуровану рекомендаційну вибірку товарів.

Для ролі Адміністратора визначено операційні сценарії керування каталогом:

CRUD-операції з товарами (рис 2.5): додавання нових позицій, редагування атрибутів (цін, локалізованих описів) та повне видалення товару з бази.

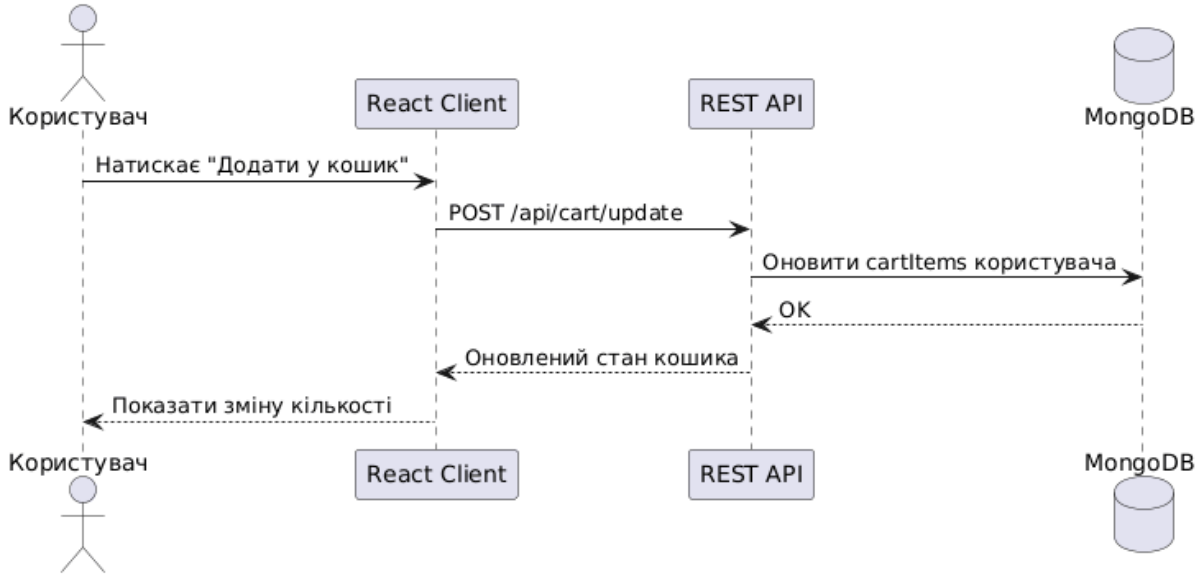

Рисунок 2.5. UML sequence діаграма для додавання товару в кошик

Управління запасами: динамічна зміна статусу наявності товару (inStock).

Глобальний моніторинг замовлень: перегляд усіх транзакцій у системі незалежно від користувача для організації логістики та контролю оплат.

2.5 Проєктування REST API

Програмний інтерфейс (API) виступає сполучною ланкою між клієнтським рівнем, бізнес-логікою та базою даних. У системі GreenCart інтеграція компонентів реалізована за архітектурним стилем REST (Representational State Transfer). Цей підхід забезпечує слабке зв’язування (loose coupling) між фронтендом та бекендом, уніфікований доступ до ресурсів через стандартні HTTP-методи та формат обміну даними JSON.

Архітектура API GreenCart базується на принципі відсутності стану (stateless): кожен запит містить вичерпну інформацію для його обробки, а контекст автентифікації передається через токени (JWT), що є критичним для горизонтального масштабування бекенду.

Програмний інтерфейс логічно декомпозовано на функціональні домени, кожен з яких інкапсулює операції над відповідними сутностями (рис 2.6):

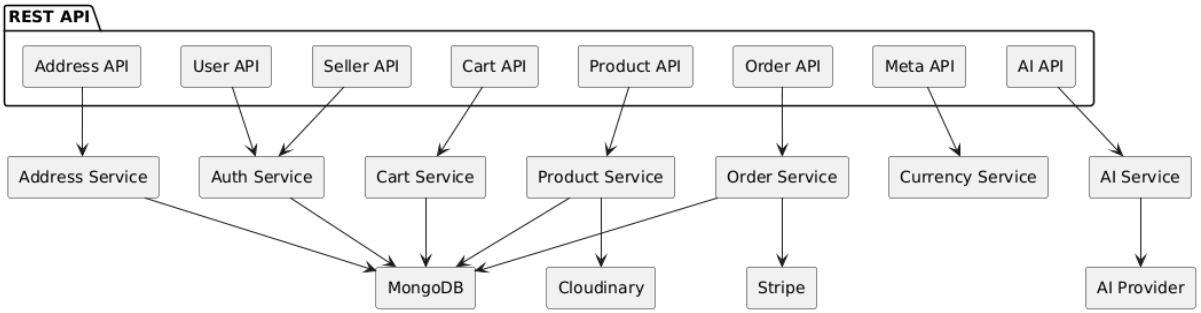

Рисунок 2.6. Структурна схема компонентів REST API

1. Модуль IAM (Identity and Access Management):

Ендпоінти (/api/user/register, /api/user/login, /api/user/logout, /api/user/is-auth) керують життєвим циклом сесій клієнтів.

Виділення окремого маршруту /api/seller/login реалізує рольову модель доступу (RBAC) на рівні маршрутизації, ізолюючи адміністративний контур від користувацького.

2. Модуль керування каталогом (/api/product/*назва товару*) (рис 2.7):

Забезпечує CRUD-операції з товарами. API поєднує публічні маршрути для читання (отримання списку та деталей товару) із захищеними маршрутами модифікації (додавання, зміна статусу, видалення), які вимагають авторизації продавця.

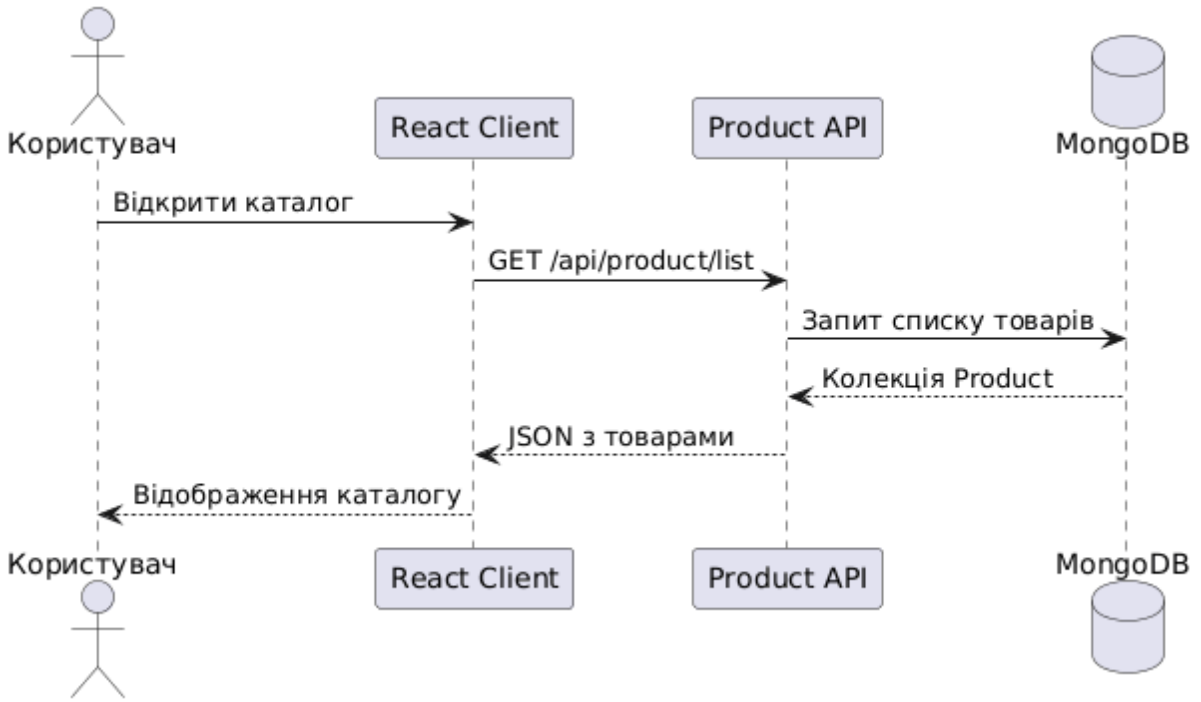

Рисунок 2.7. UML sequence діаграма для отримання каталогу товарів

1. Модуль транзакцій (Кошик та Замовлення):

Кошик (/api/cart/update): реалізовано як операцію синхронізації стану масиву товарів у профілі користувача.

Замовлення (/api/order/*назва замовлення*): центральний бізнес-домен. Архітектурно виправданим є розділення маршрутів оформлення замовлення за типом транзакції: POST /api/order/cod (післяплата) та POST /api/order/stripe (онлайн-оплата). Це дозволяє ізолювати логіку ініціалізації платіжних сесій та обробки вебхуків від стандартного процесу фіксації замовлення [44].

Приклад програмної реалізації відповідних серверних маршрутів і контролера обробки онлайн-оплати наведено в Додатку В**.**

2. Модуль профілю користувача (/api/address/*):

Забезпечує управління масивом адрес доставки незалежно від процесу оформлення замовлення, реалізуючи принцип повторного використання даних.

3. Сервісні та інтелектуальні модулі:

Метадані (GET /api/meta/exchange-rate): службовий ендпоінт для отримання актуальних курсів валют з нацбанку України, що підтримує нефункціональну вимогу локалізації та мультивалютності.

AI-модуль (POST /api/ai/query): на відміну від класичних REST-ресурсів, цей маршрут реалізує RPC-подібний підхід (Remote Procedure Call). Він приймає контекст запиту природною мовою та повертає згенеровані рекомендації або семантичну вибірку товарів. (рис 2.8)

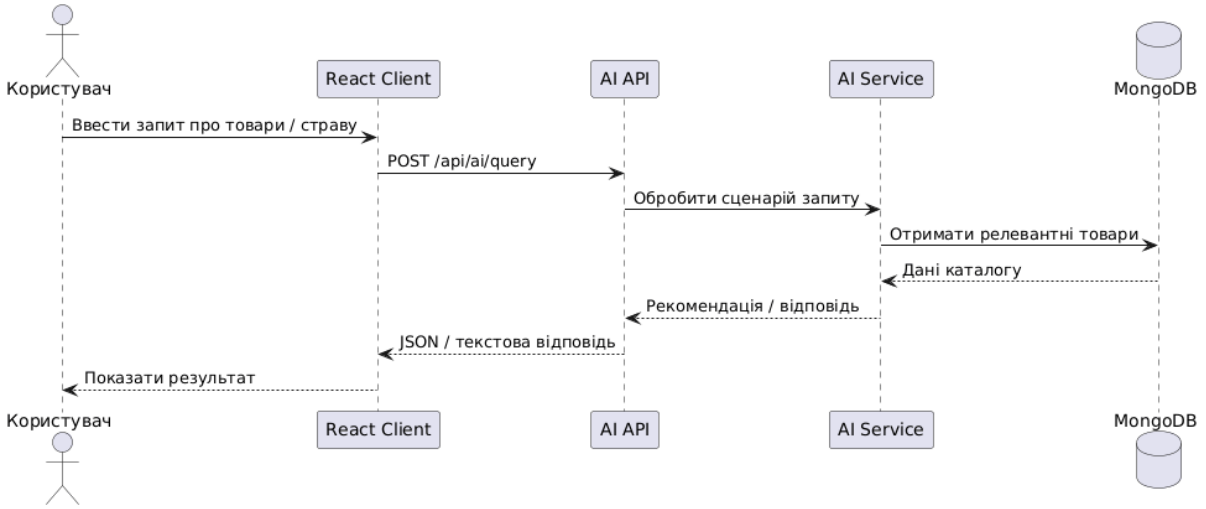

Рисунок 2.8. UML sequence діаграма для AI-запиту

Стандартизація та оптимізація.

Для забезпечення передбачуваності взаємодії всі ендпоінти використовують стандартизовану структуру відповіді: ознака успішності (success), корисне навантаження (data або message), а у випадку помилок – відповідний HTTP-статус (наприклад, 401 Unauthorized, 404 Not Found, 500 Internal Server Error). Для оптимізації мережевого навантаження API списків (наприклад, каталог) повертає лише базові атрибути, тоді як детальна інформація завантажується окремим запитом при відкритті картки товару.

Критичний аналіз та архітектурні вразливості.

Аналіз спроєктованої структури маршрутів виявляє відхилення від строгих стандартів REST, які потребують рефакторингу в процесі подальшого розвитку системи:

Передача ідентифікатора товару в тілі (body) GET-запиту (GET /api/product/id) порушує специфікацію HTTP. Для ідентифікації ресурсів у GET-запитах необхідно використовувати параметри шляху (path parameters, наприклад /api/product/:id) або параметри запиту (query parameters).

У маршруті /api/cart/update ідентифікатор користувача (userId) береться з тіла запиту, надісланого клієнтом. З точки зору інформаційної безпеки (захист від IDOR-вразливостей[45]), ідентифікатор користувача повинен вилучатися

виключно з розшифрованого та валідованого JWT на рівні серверного Middleware.

Незважаючи на виявлені точки для оптимізації, розроблений REST API покриває всі функціональні вимоги інтернет-магазину та створює надійний, слабкозв'язаний фундамент для інтеграції клієнтського застосунку з базою даних та зовнішніми сервісами.

2.6 Проєктування механізмів авторизації та безпеки

Безпека системи електронної комерції є критичним нефункціональним атрибутом, що охоплює забезпечення конфіденційності, цілісності даних та стійкості до зовнішніх загроз (OWASP[45]). У системі GreenCart архітектура безпеки базується на криптографічному захисті сесій, рольовій моделі доступу та суворій валідації вхідних даних.

Механізм автентифікації (рис 2.9) та захисту сесій Для ідентифікації користувачів реалізовано механізм на основі JWT. Процес автентифікації спроєктовано за таким алгоритмом:

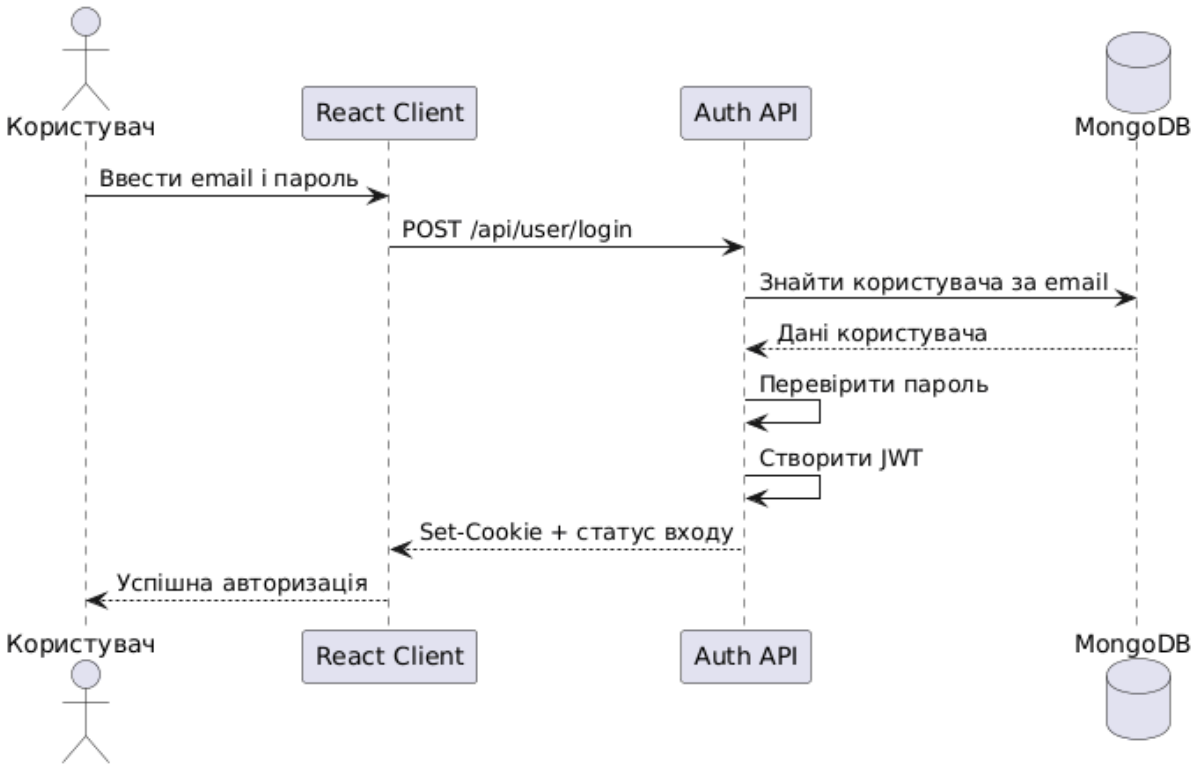

Рисунок 2.9. UML sequence діаграма процесу автентифікації

Під час реєстрації сервер хешує пароль користувача з використанням криптографічної солі алгоритмом bcrypt [46] перед збереженням у базу даних.

При успішному вході сервер генерує підписаний JWT, який інкапсулює ідентифікатор користувача (userId) та його роль.

Токен передається клієнту виключно через httpOnly cookie.

Використання httpOnly усуває можливість читання токена за допомогою клієнтського JavaScript, що є ефективним превентивним заходом проти атак типу Cross-Site Scripting (XSS). Для захисту від Cross-Site Request Forgery (CSRF) cookies конфігуруються з атрибутом SameSite=Strict або Lax та флагом Secure (у production-середовищі).

Рольова модель доступу (RBAC) Система реалізує дворівневу авторизацію:

Користувач (User): має доступ до публічного каталогу та власних захищених ресурсів (кошик, адреси, власні замовлення).

Адміністратор/Продавець (Admin): має ексклюзивний доступ до модифікації каталогу (CRUD-операції з товарами) та глобального моніторингу транзакцій через шлях …/seller. Дані продавця а саме логін та пароль зберігаються у env файлі серверу

Валідація та цілісність даних.

Будь-які дані від клієнта розглядаються як потенційно небезпечні. Система вимагає суворої валідації на стороні сервера:

Санітизація вводу: екранування спеціальних символів для запобігання ін'єкційним атакам (NoSQL Injection) [45]. Використання Mongoose частково вирішує цю проблему через сувору типізацію схем.

Логічна консистентність: перевірка типів даних, довжини рядків та діапазонів числових значень (особливо цін та кількості товарів у кошику), щоб унеможливити маніпуляції з фінансовими атрибутами на стороні клієнта.

Безпека платіжного контуру (рис 2.10).

Критично важливим архітектурним рішенням є делегування обробки фінансових даних платформі Stripe. Сервер GreenCart не збирає, не обробляє і не зберігає реквізити банківських карток (PCI DSS compliance). Під час оплати створюється унікальна checkout session, після чого користувач перенаправляється на захищений шлюз Stripe. Завершення транзакції підтверджується через криптографічно підписаний Webhook, що гарантує захист від підробки статусу оплати (isPaid).

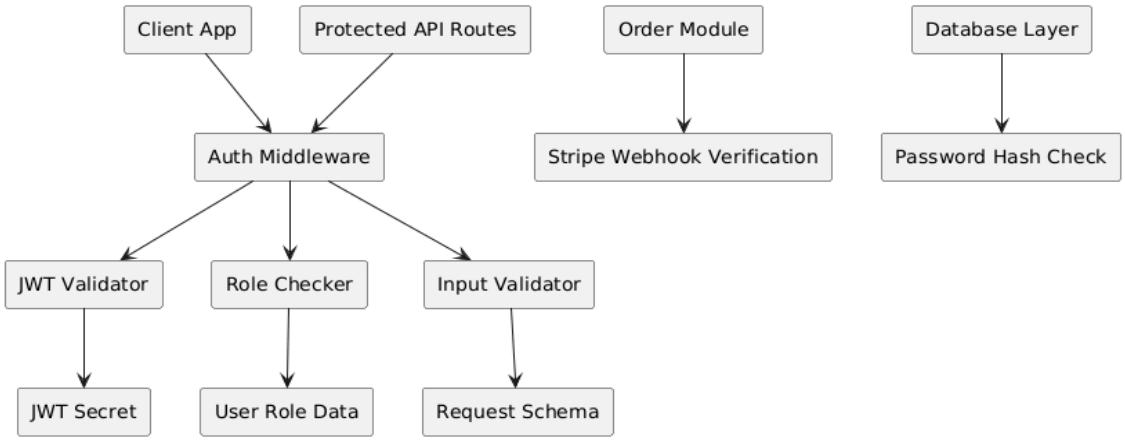

Рисунок 2.10. UML component діаграма безпекового контуру

Архітектурні вразливості та точки вдосконалення.

Аналіз реалізації виявив архітектурні дефекти, які потребують усунення для забезпечення повної цілісності системи:

1. IDOR-вразливість (Insecure Direct Object Reference): У маршруті cart/update ідентифікатор користувача отримується з тіла запиту (req.body.userId), а не з перевіреного контексту сесії. Це створює критичну загрозу: зловмисник може модифікувати кошики інших користувачів, підмінивши ID. Надійне рішення: вилучати userId виключно з розшифрованого JWT на рівні Middleware req.user.id.

2. Некоректна термінація сесій: виявлено проблеми з механізмом clearCookie при виконанні logout. Неправильне налаштування параметрів очищення cookie може призвести до збереження активної сесії у браузері, що є загрозою при використанні публічних комп'ютерів.

Висновки до розділу 2

У другому розділі виконано комплексне проєктування архітектури, моделі даних та логіки взаємодії програмної системи GreenCart. За результатами проведеного етапу проєктування сформульовано такі висновки:

1. Архітектура програмної системи: Обґрунтовано та спроєктовано багаторівневу клієнт-серверну архітектуру. Строгий логічний розподіл на рівень представлення (SPA фронтенд), рівень бізнес-логіки (Node.js/Express) та рівень збереження даних (MongoDB) забезпечує модульність, горизонтальну масштабованість системи та можливість ізольованого оновлення компонентів.

2. Структура бази даних: Розроблено фізичну модель даних на базі документоорієнтованої СУБД. Виділення базових колекцій (User, Product, Address, Order) та використання BSON-документів зі змінною структурою нативно інтегрується з MERN-стеком, дозволяє реалізувати багатомовність (через локалізовані атрибути) і гарантує історичну незмінність замовлень (через механізм вбудовування даних на момент транзакції).

3. Поведінкова модель: за допомогою UML-методології формалізовано сценарії взаємодії користувачів із системою. У межах рольової моделі (Клієнт / Адміністратор) спроєктовано повний спектр бізнес-процесів: від базових CRUD-операцій і транзакційного процесу (чекаут) до інтеграції інтелектуальних сценаріїв AI-пошуку.

4. Програмні інтерфейси (REST API): спроєктовано ресурсно-орієнтований API, декомпозований за функціональними доменами (IAM, каталог, кошик, замовлення, AI-сервіси). Модель взаємодії без збереження стану (stateless) на сервері з уніфікованими JSON-відповідями формує надійний та прозорий шлюз для зв'язку фронтенду з бекендом.

5. Архітектура безпеки: визначено та закладено в проєкт критичні механізми захисту: криптографічну автентифікацію через JWT із використанням httpOnly cookies, строгу верифікацію прав доступу (RBAC) на рівні Middleware та безпечне делегування обробки платежів (Stripe). У процесі проєктування

також виявлено потенційні вразливості (зокрема IDOR-ризики в окремих маршрутах), які потребують обов'язкового рефакторингу на етапі реалізації.

.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ВЕБ-ОРІЄНТОВАНОЇ СИСТЕМИ ІНТЕРНЕТ-МАГАЗИНУ

3.1 Реалізація клієнтської частини застосунку

Клієнтська частина вебзастосунку.

GreenCart реалізована за архітектурою SPA (Single Page Application). Вона функціонує як автономне програмне середовище у браузері, де маршрутизація, управління станом, рендеринг компонентів та взаємодія із сервером відбуваються асинхронно, без перезавантаження сторінки.

Технологічний стек фронтенд-рівня базується на сучасних інструментах JavaScript-екосистеми:

React: базова бібліотека для побудови інтерфейсу на основі декларативної компонентної моделі. Дозволяє декомпозувати UI на незалежні, повторно використовувані блоки (картки товарів, форми, навігаційні панелі), що спрощує підтримку та масштабування коду.

Vite: інструмент збирання проєкту, який забезпечує швидкий старт розробницького сервера та підтримку Hot Module Replacement (HMR), значно прискорюючи ітеративний цикл розробки.

React Router: бібліотека для клієнтської маршрутизації, що керує переходами між логічними сторінками (каталог, кошик, чекаут, адмін-панель) шляхом підміни візуальних компонентів без звернення до сервера за новими HTML-документами.

Axios: HTTP-клієнт для стандартизованої асинхронної взаємодії з REST API. Забезпечує передачу JWT-токенів, обробку JSON-відповідей та глобальне перехоплення помилок мережі.

Tailwind CSS: утилітарний CSS-фреймворк, який використовується для стилізації компонентів безпосередньо через класи, гарантуючи єдність візуальної системи та адаптивність інтерфейсу для мобільних пристроїв.

1. Публічний інтерфейс (Користувач): навігація каталогом, детальне подання товарів, управління кошиком, форми адрес, процес оформлення замовлення (чекаут) та особистий кабінет з історією покупок.

2. Адміністративний інтерфейс (Продавець): захищений простір для виконання CRUD-операцій з товарами (додавання, редагування локалізованих описів, завантаження зображень, зміна статусу наявності) та моніторингу глобального списку замовлень.

Реалізація ключових клієнтських сценаріїв.

Реалізація ключових клієнтських сценаріїв здійснюється за наступною схемою(рис 3.1):

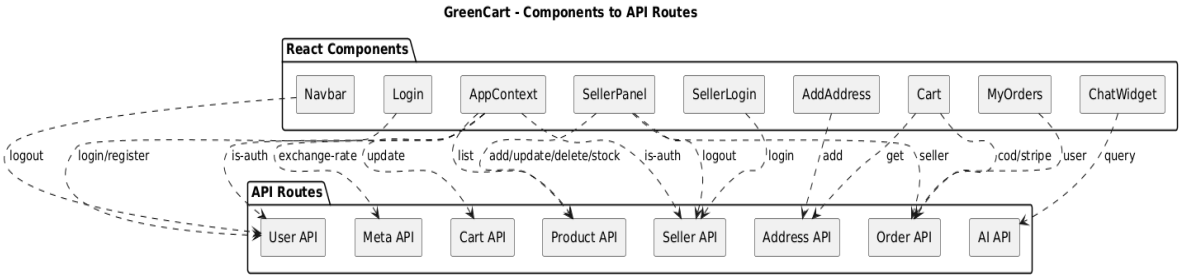

Рисунок 3.1. Діаграма взаємодії компонентів GreenCart з API-маршрутами

Каталог та картка товару (рис 3.2, 3.3): При ініціалізації компонента каталогу виконується асинхронний запит до GET /api/product/list. Отриманий масив об'єктів перетворюється на ієрархію UI-компонентів (картки товарів). Детальна сторінка товару додатково відтворює локалізовані атрибути, галерею зображень та елементи керування кошиком.

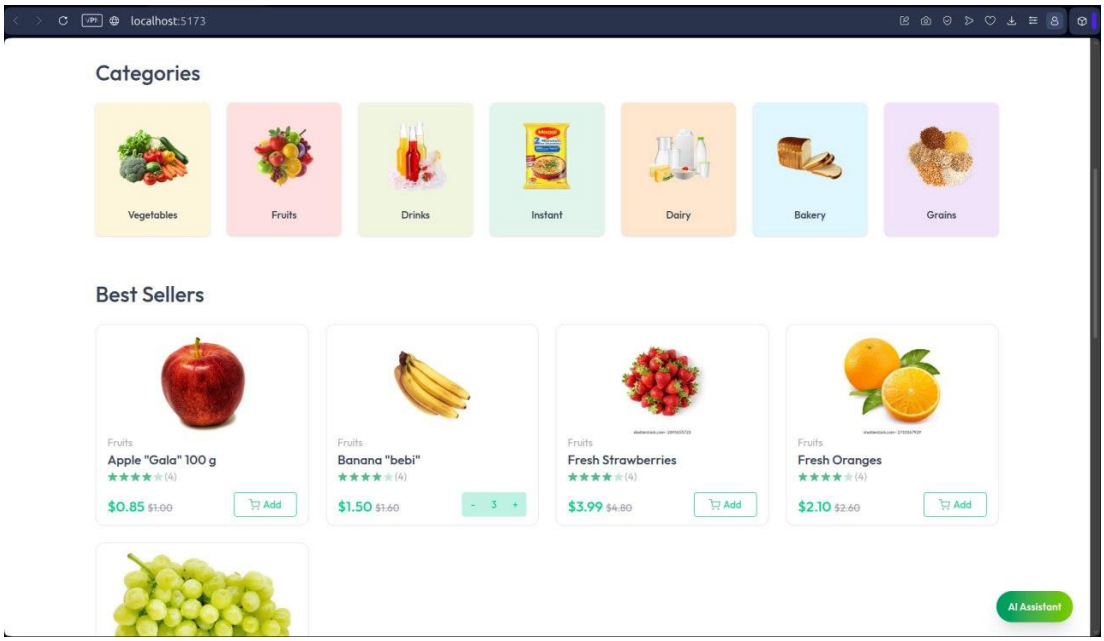

Рисунок 3.2. Скріншот каталогу товарів

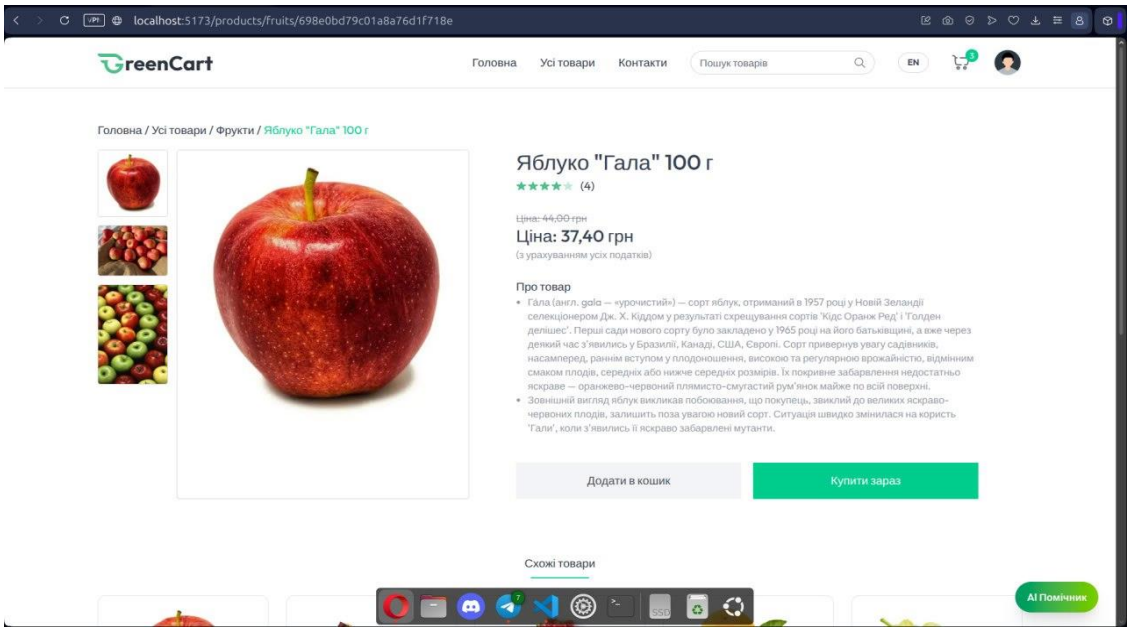

Рисунок 3.3. Скріншот картки товару

Синхронізація кошика: Робота з кошиком (рис. 3.4) вимагає миттєвого відгуку інтерфейсу. Додавання товару або зміна його кількості ініціює локальне оновлення стану (оптимістичний UI) з паралельною фоновою синхронізацією із сервером через API.

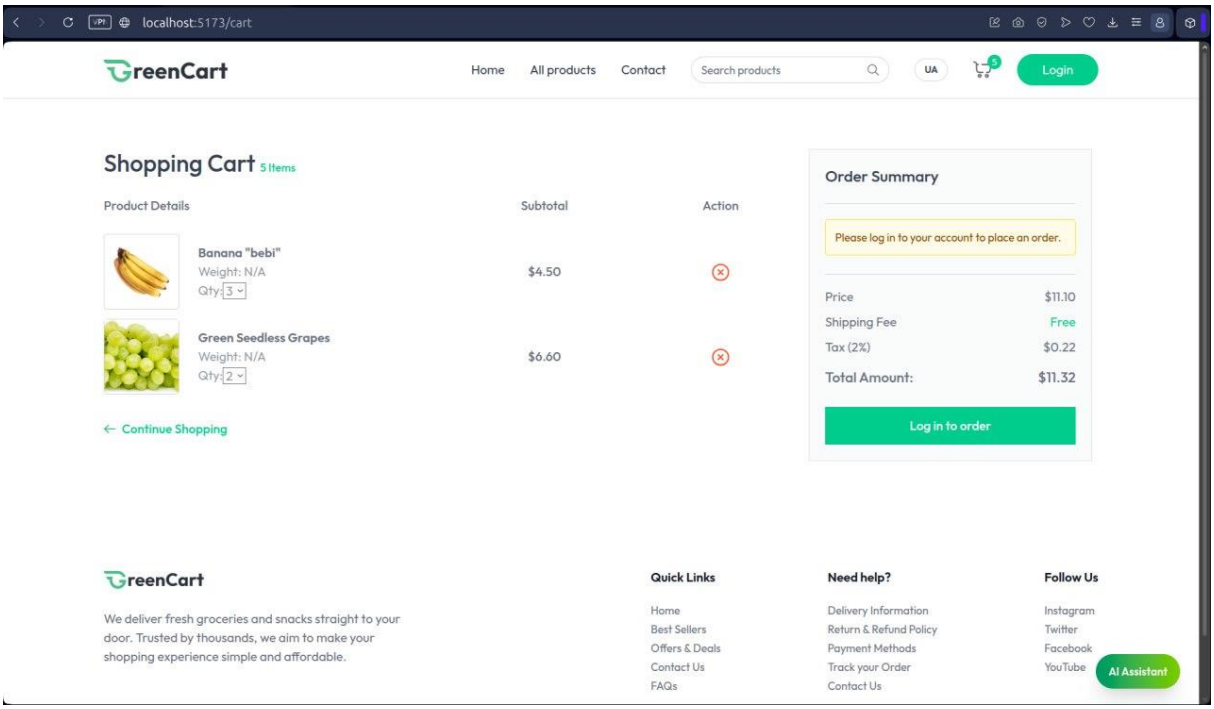

Рисунок 3.4. Скріншот кошику

Оформлення замовлення (рис. 3.5): Складений сценарій, що інтегрує локальний стан (сума кошика), введення адресних даних та взаємодію із зовнішнім платіжним провайдером. У разі вибору онлайн-оплати, інтерфейс обробляє отриманий від бекенда URL платіжної сесії та перенаправляє користувача на захищений шлюз Stripe.

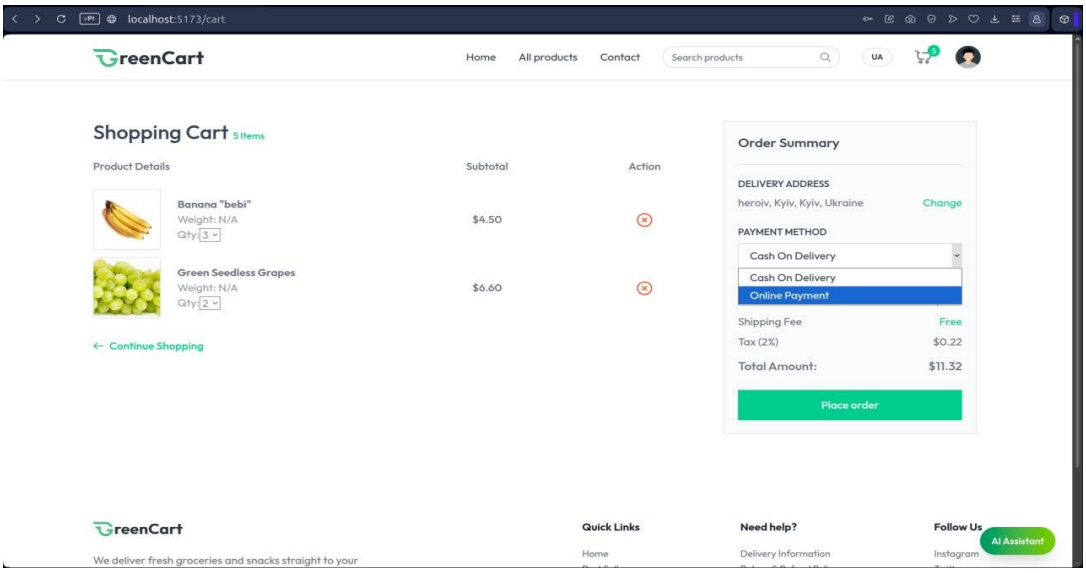

Рисунок 3.5. Оформлення замовлення

Локалізація та мультивалютність: інтерфейс підтримує динамічне перемикання між українською та англійською мовами. Клієнтська частина керує поточним контекстом локалізації, відображаючи відповідні поля з об'єкта Product (наприклад, nameUa або nameEn), та виконує конвертацію цін на льоту, спираючись на дані службового ендпоінта курсів валют нацбанку України.

Взаємодія з AI-модулем (рис. 3.6): реалізовано окремий UI-компонент (чат або форма пошуку), який приймає природномовні запити користувача, надсилає їх до POST /api/ai/query та рендерить отримані рекомендації (рецепти, бюджетні добірки) у вигляді структурованих карток товарів.

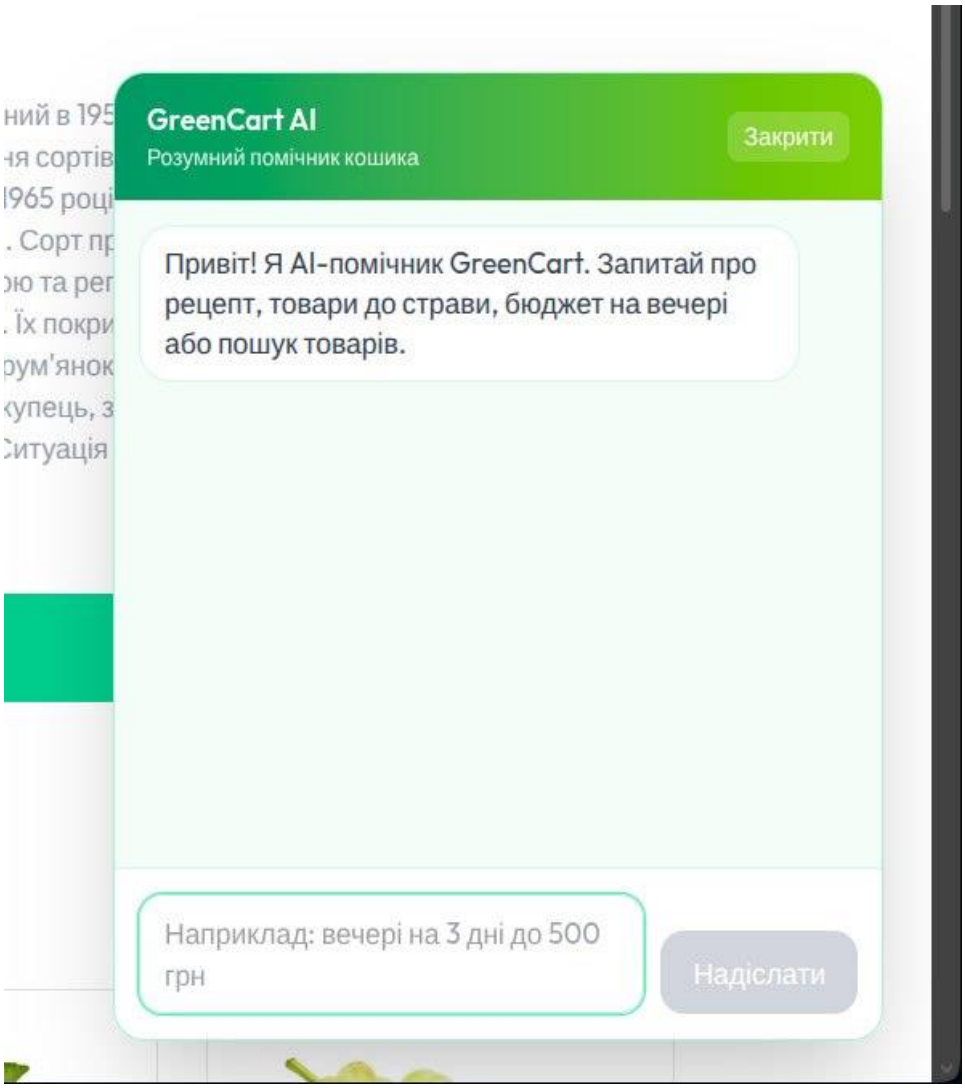

Рисунок 3.6. Скріншот AI модуля

Важливим аспектом інженерної якості клієнтської частини є обробка UX-станів (рис. 3.7, 3.8, 3.9). Застосунок коректно опрацьовує життєвий цикл асинхронних запитів: відображає індикатори завантаження (спінери, скелетони) під час очікування відповіді, рендерить інформативні заглушки при порожніх результатах пошуку та перехоплює мережеві збої за допомогою механізмів Error Boundaries і сповіщень-тостів (toast notifications).

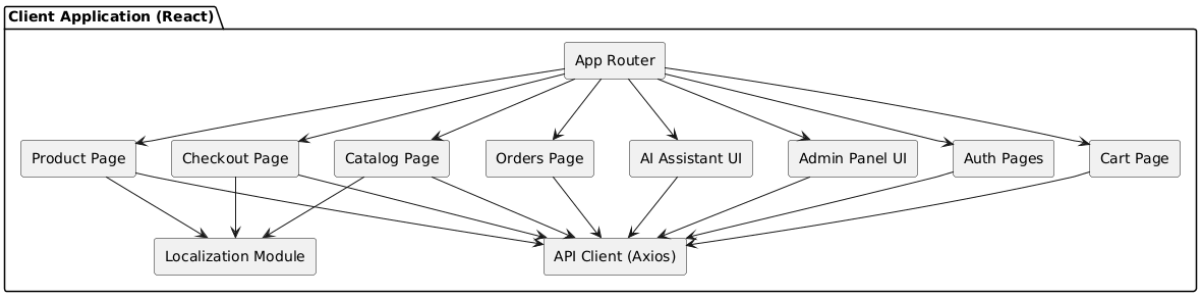

Рисунок 3.7. UML component діаграма клієнтської частини

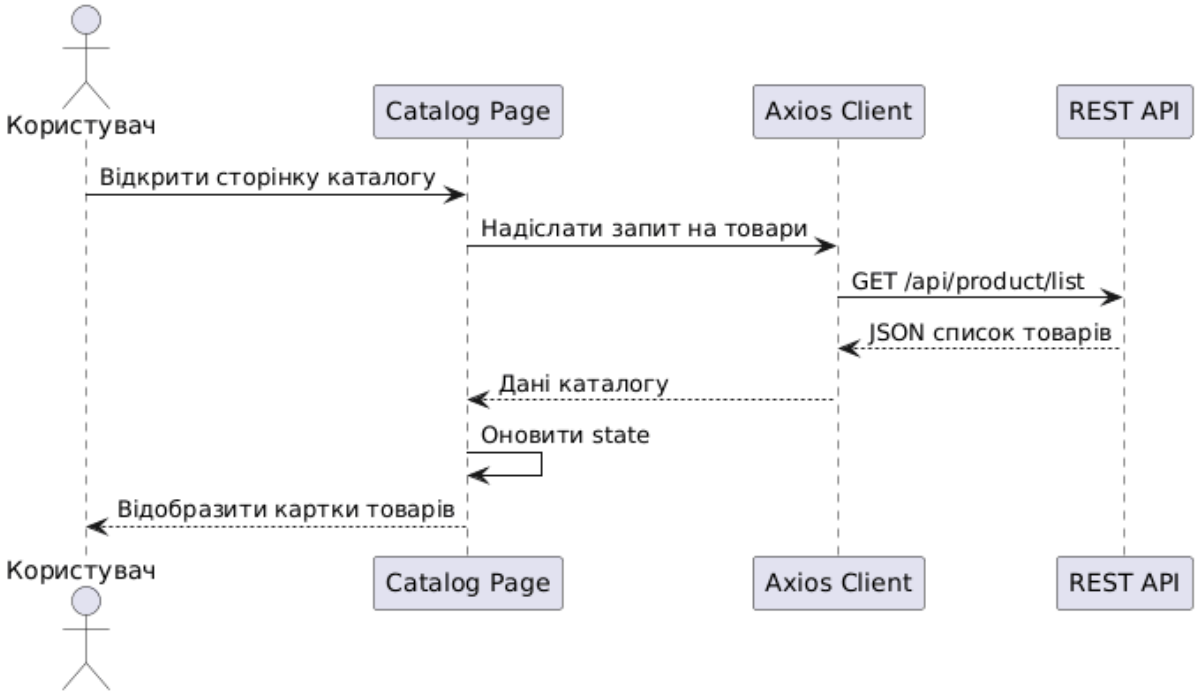

Рисунок 3.8. UML sequence діаграма завантаження каталогу на клієнті

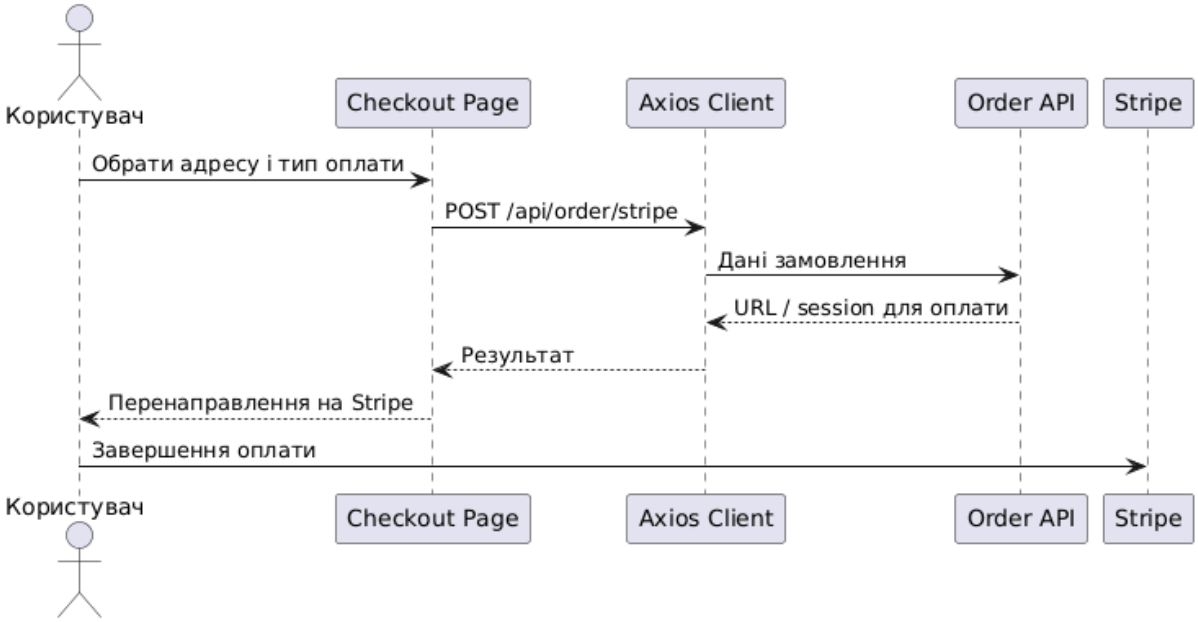

Рисунок 3.9. UML sequence діаграма клієнтського сценарію оформлення замовлення

3.2 Реалізація серверної частини та бізнес-логіки

Серверна частина (бекенд) системи GreenCart реалізована на платформі Node.js із використанням вебфреймворку Express.js. Вибір Node.js обґрунтовано його асинхронною подієво-орієнтованою моделлю введення-виведення (non-blocking I/O)[47]. Це забезпечує високу пропускну здатність і дозволяє бекенду ефективно обробляти велику кількість конкурентних мережевих запитів (наприклад, масовий перегляд каталогу або одночасне оформлення замовлень) без блокування основного потоку виконання.

Програмний код серверної частини побудовано за патерном Layered Architecture (Багаторівнева архітектура)[16], що суворо реалізує принцип Separation of Concerns (розділення відповідальності):

Маршрути (Routes): визначають кінцеві точки REST API та направляють вхідні HTTP-запити до відповідних контролерів.

Контролери (Controllers): інкапсулюють бізнес-логіку застосунку (реєстрація, управління кошиком, ініціалізація замовлень тощо).

Моделі (Models): описують фізичну структуру даних і забезпечують взаємодію з БД.

Сервіси (Services): ізольовані модулі для інтеграції із зовнішніми API.

Життєвий цикл обробки запитів та Middleware (рис. 3.10).

Express.js дозволяє гнучко керувати конвеєром обробки запитів через проміжне програмне забезпечення (Middleware). Стандартизований алгоритм обробки в системі виглядає так:

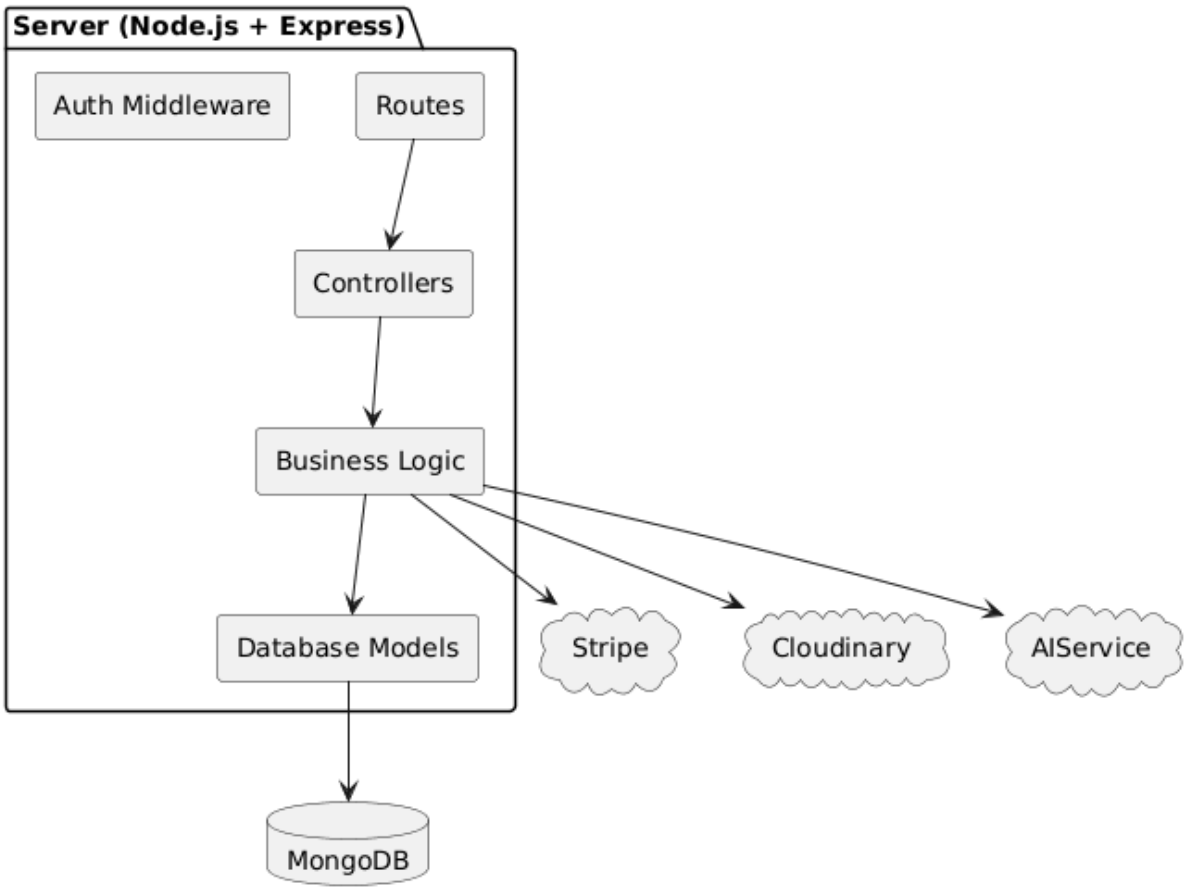

Рисунок 3.10. UML діаграма компонентів серверної частини

1. Запит потрапляє до маршрутизатора.
2. Проходить через глобальні та специфічні Middleware: парсинг JSON-тіла, перевірка наявності та валідності JWT-токена, контроль прав доступу (RBAC) та санітизація вхідних даних.
3. У разі успішної валідації керування передається контролеру для виконання бізнес-логіки.

4. Контролер взаємодіє з базою даних або зовнішнім сервісом.

5. Клієнту повертається уніфікована JSON-відповідь.

У разі порушення бізнес-правил або помилок (некоректні дані, брак привілеїв), глобальний обробник помилок перериває ланцюг і повертає відповідний HTTP-статус (наприклад, 400 Bad Request, 401 Unauthorized, 403 Forbidden, 500 Internal Server Error), що забезпечує стабільність системи та полегшує клієнтську обробку винятків.

Взаємодія з базою даних (ODM).

Для роботи з MongoDB на серверному рівні інтегровано бібліотеку Mongoose (Object Data Modeling). Mongoose дозволяє абстрагуватися від сирих запитів до бази даних, описуючи базові сутності (User, Product, Address, Order) у вигляді жорстко типізованих схем. Це забезпечує валідацію типів на рівні застосунку перед спробою запису в базу даних та дозволяє виконувати CRUD-операції в об'єктно-орієнтованому стилі.

Інтеграція зовнішніх сервісів.

Серверна частина діє як безпечний посередник (проксі) між клієнтським застосунком та сторонніми інфраструктурними платформами:

Cloudinary: використовується сервісом керування товарами для завантаження, оптимізації та хмарного зберігання медіафайлів (зображень продуктів).

Stripe: обробка онлайн-транзакцій. Контролер замовлень створює захищену платіжну сесію (Checkout Session) та повертає клієнту ідентифікатор сесії. Сервер оновлює фінансовий статус замовлення виключно після отримання криптографічно підтвердженого Webhook від Stripe.

AI-ендпоінт: контролер штучного інтелекту приймає природномовні запити користувачів, обробляє їх (через локальні евристики або звернення до зовнішніх LLM-провайдерів) і формує структуровану рекомендаційну вибірку товарів.

Така модульна серверна архітектура відповідає сучасним стандартам веброзробки, забезпечуючи високий рівень безпеки, масштабованості та легкості в подальшому супроводі коду.

3.3 Реалізація функціоналу управління товарами

Модуль управління товарами є базовою CRUD-підсистемою (Create, Read, Update, Delete) електронної комерції. Він інкапсулює логіку взаємодії адміністратора (продавця) з каталогом, а також забезпечує клієнтський застосунок даними для рендерингу вітрини. На серверному рівні цей функціонал реалізовано через набір захищених і публічних REST-ендпоінтів, які взаємодіють із колекцією Product у базі даних MongoDB.

Модель даних.

Основою підсистеми є ODM-схема Product (через Mongoose), яка визначає структуру документа та містить такі логічні блоки атрибутів:

Базова інформація: універсальний ідентифікатор, категорія товару.

Локалізований контент: назва та опис двома мовами (наприклад, nameUa, nameEn, descriptionUa, descriptionEn) для підтримки мультимовного інтерфейсу.

Ціноутворення: стандартна ціна (price) та акційна ціна (promotionalPrice).

Медіа: масив URL-адрес зображень товару.

Стан: булевий прапорець наявності inStock, який керує видимістю або можливістю замовлення товару.

Реалізація REST-операцій.

Функціонал управління каталогом охоплює повний життєвий цикл товарної позиції:

Створення (Create): Ініціюється через захищений маршрут POST /api/product/add. Контролер валідує вхідні дані від адміністратора, асинхронно завантажує файли зображень до хмарного сховища, після чого формує новий BSON-документ і зберігає його в MongoDB.

Читання (Read): Колекція: GET /api/product/list повертає масив об'єктів Product для відтворення загального каталогу. Для оптимізації продуктивності

бази даних у майбутньому цей маршрут передбачає інтеграцію механізмів пагінації (limit/skip) та фільтрації.

Окремий ресурс: GET /api/product/id (або /:id) використовується для отримання вичерпної інформації при переході користувача на сторінку детального перегляду товару.

Оновлення (Update): Виконується через маршрут POST /api/product/update/:id (архітектурно для оновлення зазвичай використовуються методи PUT або PATCH, проте в межах поточного проєкту застосовано POST). Операція дозволяє здійснювати часткове або повне оновлення атрибутів документа. Окремим важливим бізнес-кейсом є динамічна зміна статусу наявності (inStock = false), що блокує можливість додавання товару до кошика без його фактичного видалення з бази.

Видалення (Delete): Реалізовано через DELETE /api/product/:id. У поточній конфігурації виконується фізичне видалення документа з колекції. (Варто зазначити, що для збереження цілісності історичних фінансових даних та статистики замовлень у промислових системах частіше застосовується патерн «м'якого видалення» - soft delete, який зводиться до маркування товару як неактивного).

Інтеграція хмарного сховища Content delivery network(далі - CDN).

Оскільки інтернет-магазин генерує значний обсяг статичного графічного трафіку, збереження зображень локально на сервері є неефективним з точки зору масштабування та пропускної здатності. Для вирішення цього інженерного завдання модуль управління товарами інтегровано з хмарним сервісом **Cloudinary**. Під час додавання або редагування товару сервер працює як проксі (рис. 3.11): перенаправляє бінарні дані файлів до Cloudinary, отримує у відповідь оптимізовані URL-адреси (що роздаються через CDN) та зберігає лише ці посилання у масиві images документа Product.

Таким чином, реалізований модуль є автономною, масштабованою підсистемою, яка надійно забезпечує адміністративне керування контентом та безперебійне постачання даних для клієнтського інтерфейсу.

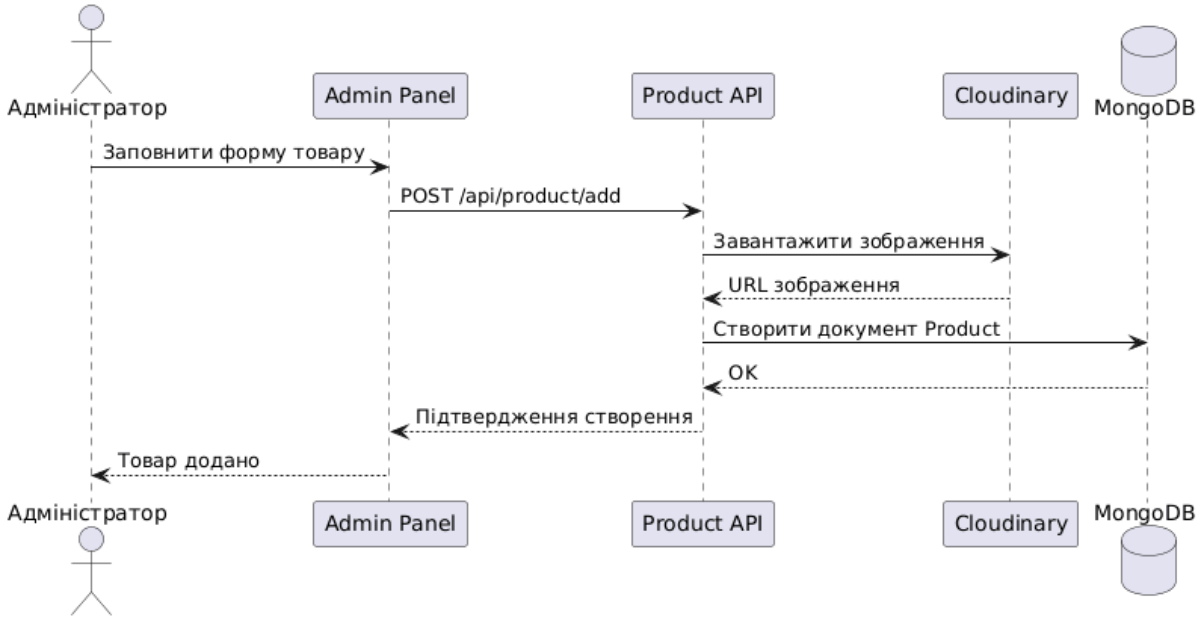

Рисунок 3.11. UML sequence діаграма взаємодії з CDN Cloudinary

3.4 Реалізація механізму обробки замовлень

Механізм обробки замовлень є транзакційним ядром системи електронної комерції. Цей модуль відповідає за конвертацію тимчасового стану кошика користувача у персистентний запис (документ) бази даних, фіксуючи фінансові та логістичні атрибути угоди. У системі GreenCart реалізація цього процесу охоплює серверну валідацію, обчислення вартості, маршрутизацію платіжних сценаріїв та взаємодію з базою даних MongoDB.

Модель даних та принцип історичної незмінності.

Сутність Order інкапсулює повний контекст транзакції. Її схема містить такі обов'язкові атрибути:

- Ідентифікатор користувача (Reference на колекцію User).
- Масив придбаних товарів.
- Об'єкт адреси доставки.
- Фінансові показники: загальна сума замовлення та вартість доставки.
- Статус оплати (наприклад, pending, paid, failed).

- Статус виконання (наприклад, processing, shipped, delivered).
- Часові мітки (Timestamps).

Критичним архітектурним рішенням є збереження масиву товарів як історичного зрізу (snapshot). Замість збереження лише посилань (ObjectIDs) на колекцію Product, у документ замовлення копіюються назви, ціни та зображення товарів на момент здійснення покупки. Це гарантує цілісність фінансової історії: якщо адміністратор змінить ціну в каталозі або видалить товар, це не вплине на дані в уже оформлених замовленнях.

Алгоритм ініціалізації транзакції.

Процес оформлення замовлення (Checkout) ініціюється клієнтом, але критична бізнес-логіка виконується виключно на сервері для запобігання маніпуляціям із даними:

Валідація наявності: сервер перевіряє, чи існують обрані товари в базі даних і чи є вони доступними (inStock === true).

Синхронізація цін: ігноруючи ціни, передані з клієнтського застосунку (payload), бекенд самостійно витягує актуальні ціни з колекції Product та перераховує загальну суму. Це унеможливлює атаку через підміну вартості.

Маршрутизація платіжних сценаріїв.

Залежно від вибору користувача, система спрямовує обробку замовлення за одним із двох незалежних алгоритмів:

Післяплата (Cash on Delivery): Обробляється синхронно через маршрут POST /api/order/cod. Замовлення одразу зберігається в базі даних зі статусом оплати pending та статусом виконання processing. Користувач миттєво отримує підтвердження успішного оформлення.

Онлайн-оплата (Stripe Integration): Обробляється асинхронно через POST /api/order/stripe. Для дотримання стандарту PCI DSS сервер GreenCart не обробляє реквізити карток. Замість цього контролер звертається до API Stripe, формує платіжну сесію (Checkout Session) та повертає клієнту її URL. Після здійснення транзакції на боці Stripe, платіжний шлюз надсилає криптографічно

підписаний HTTP-запит (Webhook) на сервер GreenCart. Лише після валідації вебхука сервер оновлює статус замовлення на paid.

Моніторинг та управління замовленнями(рис 3.12, 3.13).

Для забезпечення користувацького досвіду та адміністративного контролю реалізовано два роздільні інтерфейси читання:

GET /api/order/user: повертає виключно ті замовлення, які належать авторизованому користувачу (фільтрація за userId з токена автентифікації).

GET /api/order/seller: адміністративний ендпоінт, що надає доступ до глобального пулу замовлень для подальшого оновлення статусів доставки (shipped, delivered).

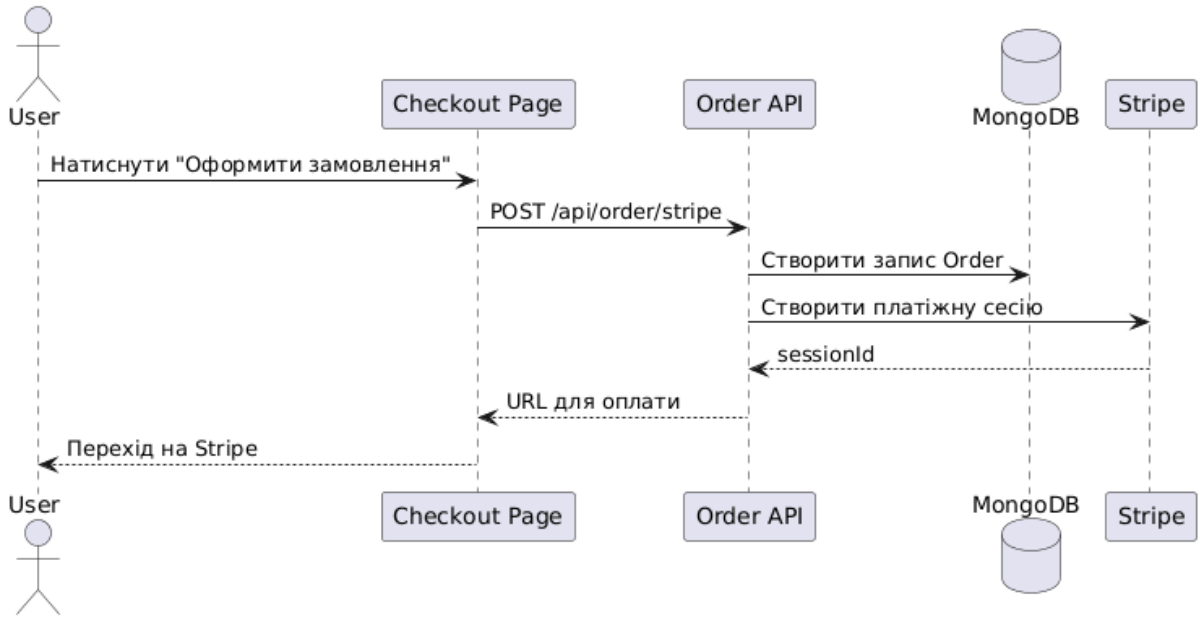

Рисунок 3.12. UML sequence діаграма оформлення замовлення

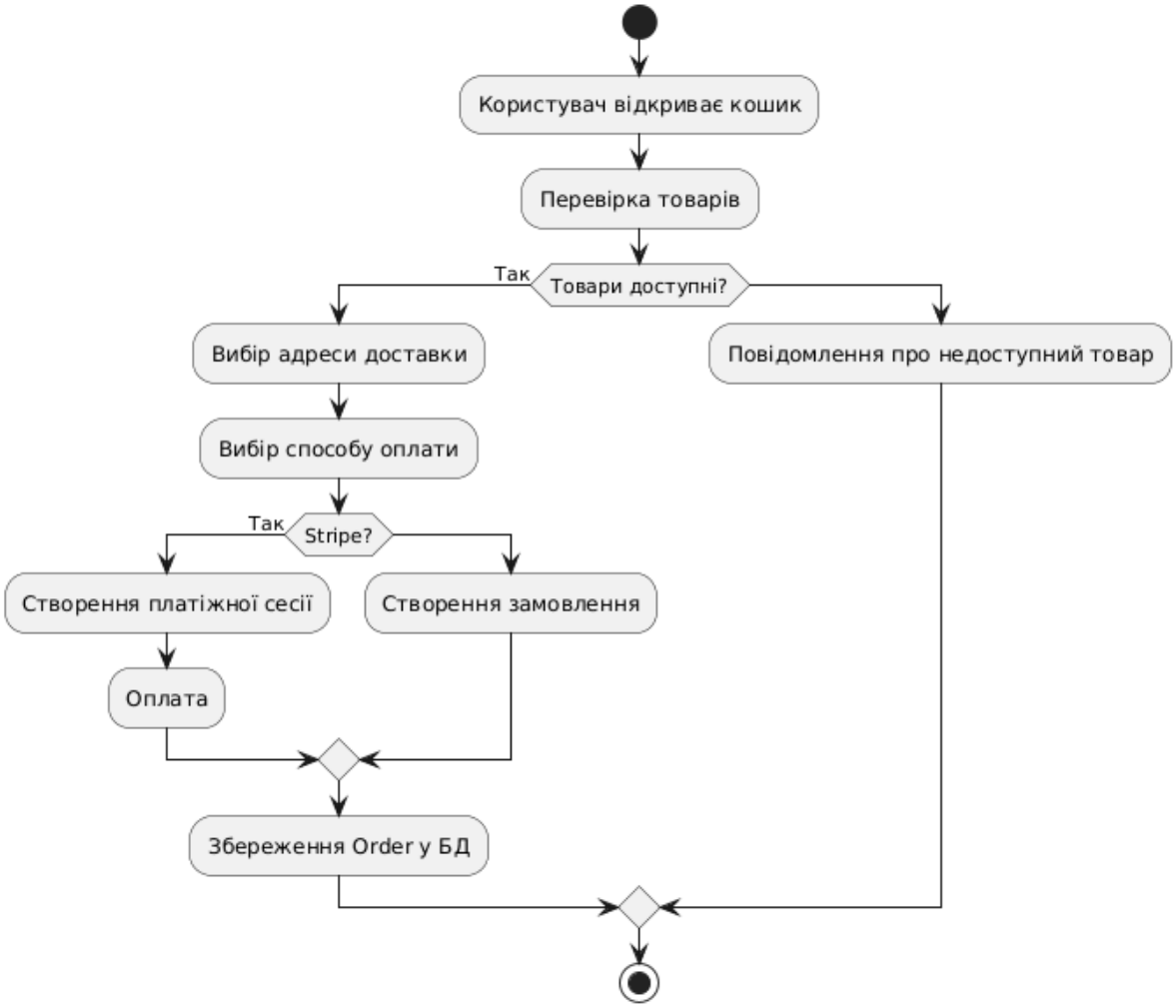

Рисунок 3.13. UML activity діаграма процесу замовлення

3.5 Реалізація системи рекомендацій та пошуку товарів

Інтелектуальний підбір товарів є розширеним компонентом сучасних систем електронної комерції, спрямованим на оптимізацію користувацького досвіду (UX) та спрощення навігації неструктурованими масивами каталогу. У системі GreenCart цей функціонал (рис. 3.14) реалізовано у вигляді гібридного модуля пошуку та рекомендацій, який поєднує класичні базисні методи з технологіями штучного інтелекту.

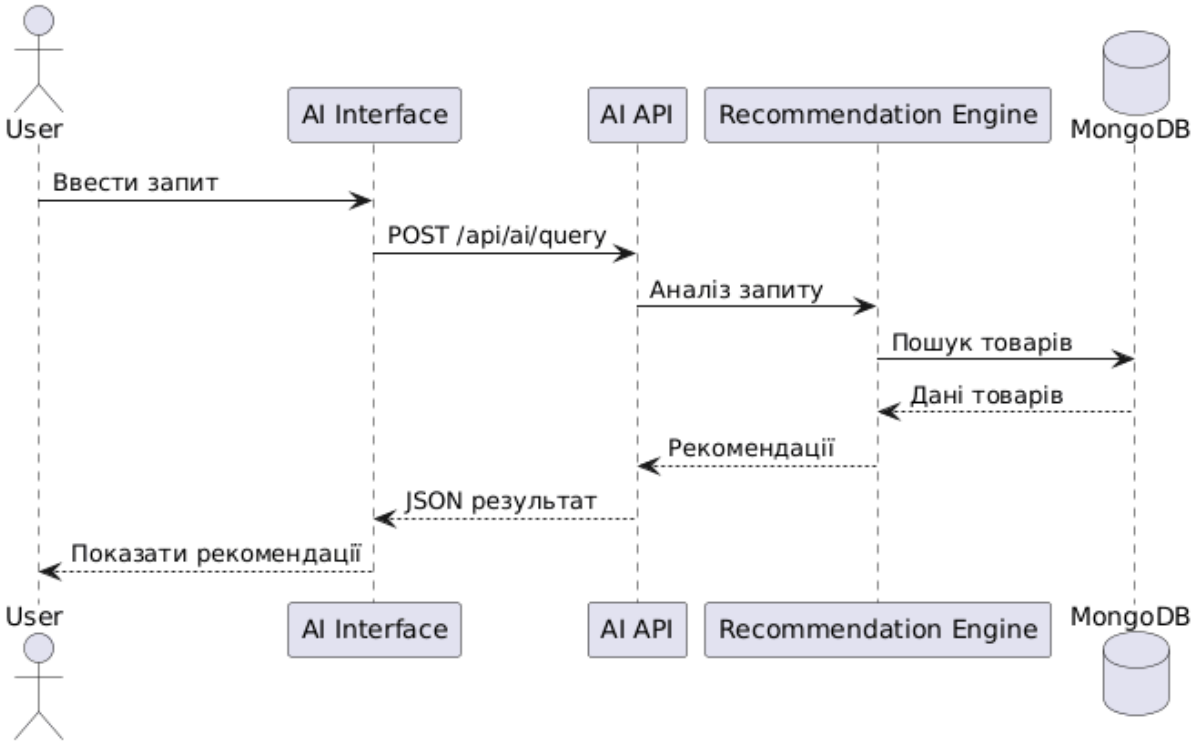

Рисунок 3.14. UML sequence діаграма AI-запиту

Архітектура обробки запитів.

Точкою входу для підсистеми є спеціалізований маршрут POST /api/ai/query. На відміну від стандартних пошукових API, що оперують жорсткими фільтрами, цей ендпоінт виступає інтерфейсом обробки природної мови Natural Language Processing (далі - NLP) [48]. Він приймає неструктуровані текстові запити, наприклад «що купити для приготування борщу», «набір продуктів до 500 гривень», і класифікує інтенцію користувача. Залежно від виявленого патерну, серверний контролер ініціює відповідний пайплайн обробки: генерацію рецепта, пошук інгредієнтів для конкретної страви або оптимізацію кошика під заданий бюджет.

Для підвищення достовірності рекомендацій використано підхід retrieval-augmented generation (RAG), за якого мовна модель не формує відповідь ізольовано, а спирається на попередньо відібраний контекст із власної бази товарів. Такий підхід зменшує ризик генерації неіснуючих позицій і дозволяє будувати рекомендації лише на основі актуального асортименту GreenCart. У межах цієї логіки запит користувача та описи товарів можуть перетворюватися у

векторні представлення, після чого здійснюється пошук найбільш релевантних позицій у базі даних.

Окремим елементом реалізації є використання оркестратора n8n, який забезпечує побудову послідовності обробки AI-запиту через webhook, формування структурованого промпта, передавання параметрів фільтрації та взаємодію з мовною моделлю у форматі JSON. Це дозволяє винести частину логіки AI-процесів у гнучкий workflow-рівень і спростити подальше масштабування або зміну сценаріїв рекомендацій.

Алгоритмічні підходи.

Для формування релевантної вибірки товарів AI-модуль спирається на комбінацію кількох методів інформаційного пошуку:

Лексичний пошук (Keyword Search [49]): базова токенізація запиту та повнотекстовий пошук за індексованими атрибутами колекції Product: назви, описи, категорії.

Семантичний аналіз (Semantic Search [50]): використання AI-моделей для інтерпретації контексту. Це дозволяє системі знаходити релевантні товари навіть за відсутності прямих текстових збігів, враховуючи синоніми та категорійні зв’язки.

Евристичне зіставлення (Rule-based Mapping [11]): застосування правил предметної області. Наприклад, декомпозиція комплексного запиту «рецепт борщу» на масив обов’язкових інгредієнтів: буряк, капуста, м’ясо, із подальшим автоматичним пошуком цих сутностей у базі даних.

Оптимізація за обмеженнями (Budget Filtering [50]): математичний відбір товарів, що максимізує релевантність вибірки за умови дотримання жорсткого фінансового ліміту, що є практичною варіацією задачі про рюкзак [51; 52].

Для забезпечення стабільної роботи модуля також передбачено постобробку результатів: нормалізацію відповіді, виділення ключових рекомендацій, кешування повторюваних запитів і застосування обмеження частоти звернень. Це дозволяє зменшити затримку відповіді, уникнути

надмірного навантаження на AI-сервіс і забезпечити більш прогнозовану роботу системи під час одночасного використання кількома користувачами.

Оцінка ефективності роботи модуля.

Ефективність AI-модуля може оцінюватися не лише за швидкістю відповіді, а й за якістю сформованих рекомендацій. Для цього доцільно використовувати метрики Precision, Recall та F1-score, які дозволяють визначити частку релевантних товарів серед запропонованих позицій, повноту охоплення правильних результатів і збалансовану оцінку якості рекомендацій. У межах експериментального тестування, описаного в тезах за темою: «Розвиток технологій штучного інтелекту у вебдодатках електронної комерції»[3], комбінований підхід Vector Search + LLM продемонстрував достатній рівень релевантності рекомендацій із показником F1-score близько 0.79.

Окрім якості рекомендацій, важливим критерієм є продуктивність системи. Моделювання навантаження показало, що система зберігає стабільність при обробці запитів до 200 одночасних користувачів[3], після чого затримка відповіді починає зростати через вплив зовнішнього API мовної моделі, витрати на векторний пошук і мережеві затримки. Це підтверджує необхідність подальшої оптимізації кешування, обмеження частоти запитів і масштабування пошукової підсистеми.

У перспективі масштабування архітектура пошукової підсистеми створює базу для впровадження механізмів автодоповнення (Autocomplete [53]), фасетної фільтрації та персоналізованих рекомендацій на основі алгоритмів колаборативної фільтрації (Collaborative Filtering [54]). З точки зору програмної інженерії, реалізація цього модуля трансформує стандартний CRUD-додаток у семантично-орієнтоване середовище, демонструючи ефективну інтеграцію інтелектуальних сервісів у прикладну комерційну розробку.

3.6 Інтеграція зовнішніх сервісів

У сучасній інженерії вебзастосунків делегування спеціалізованих обчислювальних, логістичних або транзакційних процесів стороннім

інфраструктурним платформам (SaaS/PaaS) є галузевим стандартом, який пришвидшує швидкість розробки[55]. У системі GreenCart реалізовано інтеграцію трьох зовнішніх сервісів, що дозволило суттєво знизити навантаження на основний сервер, уникнути розробки складних вузькоспеціалізованих модулів та підвищити загальну відмовостійкість архітектури.

Інтеграція платіжного шлюзу (Stripe).

Для забезпечення фінансових транзакцій систему інтегровано з платформою Stripe (рис 3.15).

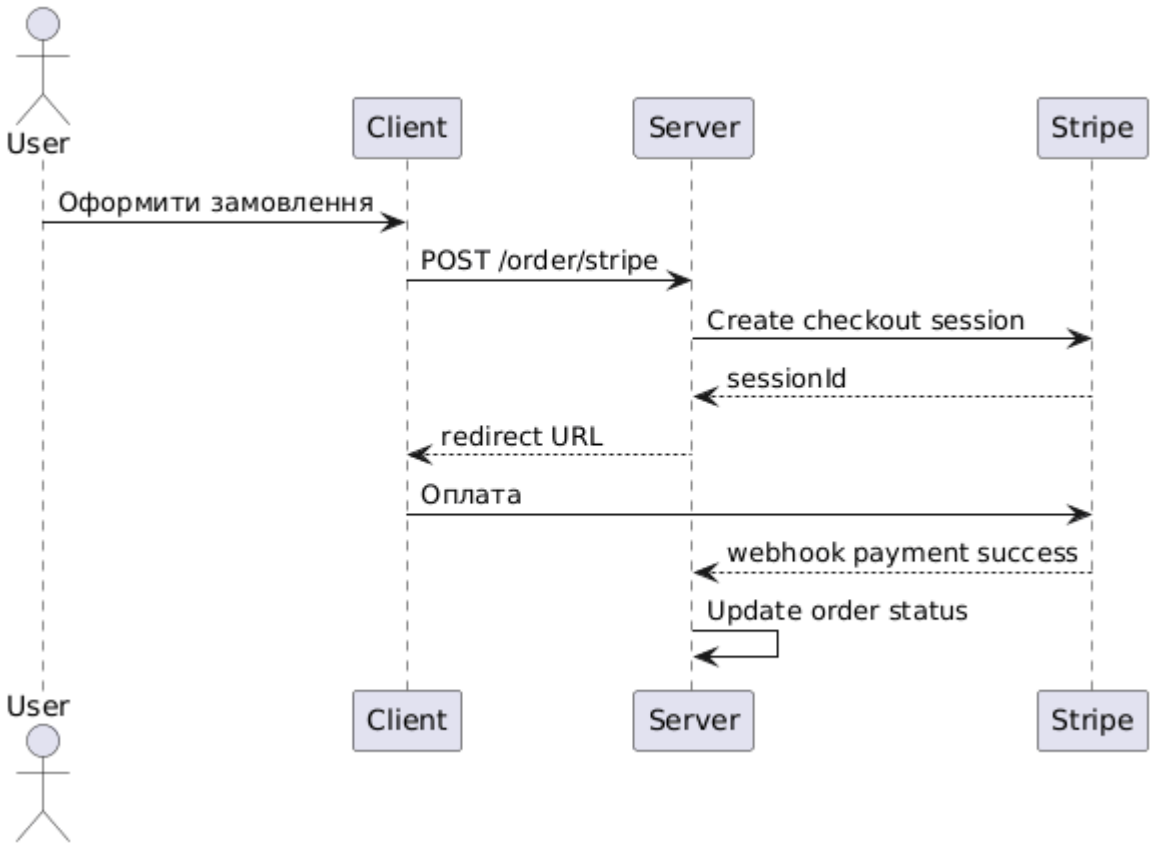

Рисунок 3.15. UML sequence діаграма платежу Stripe

Таке архітектурне рішення гарантує повну відповідність міжнародним стандартам безпеки платіжних карток (PCI DSS[44]), оскільки сервер GreenCart не збирає, не обробляє і не зберігає чутливі реквізити (PAN, CVV). Процес інтеграції реалізує патерн асинхронного підтвердження платежів:

1. Ініціалізація: Під час оформлення замовлення бекенд звертається до Stripe API та генерує унікальну платіжну сесію (Checkout Session), передаючи параметри кошика.

2. Делегування: Клієнтський застосунок отримує ідентифікатор сесії та перенаправляє користувача на захищений хостований шлюз Stripe для введення даних картки.

3. Асинхронне підтвердження (Webhook): Після завершення транзакції Stripe генерує криптографічно підписаний HTTP-запит (webhook) на спеціальний службовий ендпоінт сервера GreenCart.

4. Синхронізація стану: Сервер валідує сигнатуру вебхука (для запобігання підробленим запитам) та атомарно оновлює статус відповідного документа Order у MongoDB на paid.

Хмарне зберігання та дистрибуція медіаданих (Cloudinary).

Зберігання статичного графічного контенту на локальній файловій системі Node.js-сервера є архітектурним антипатерном, що ускладнює горизонтальне масштабування та створює вузькі місця в I/O операціях. Для вирішення цього завдання імплементовано сервіс Cloudinary. Під час додавання або редагування товару адміністратором (рис. 3.16), сервер виконує роль проксі-шлюзу: він приймає бінарні дані файлу та асинхронно відправляє їх до хмарного сховища. У відповідь Cloudinary генерує постійну URL-адресу, яка зберігається в масиві images колекції Product. Крім ізоляції файлової системи бекенда, використання Cloudinary,як і було зазначено раніше, виконує роль CDN забезпечуючи кешування на edge-серверах та динамічну оптимізацію розміру й формату зображень «на льоту» (наприклад, конвертацію у WebP), що критично важливо для продуктивності клієнтського рендерингу.

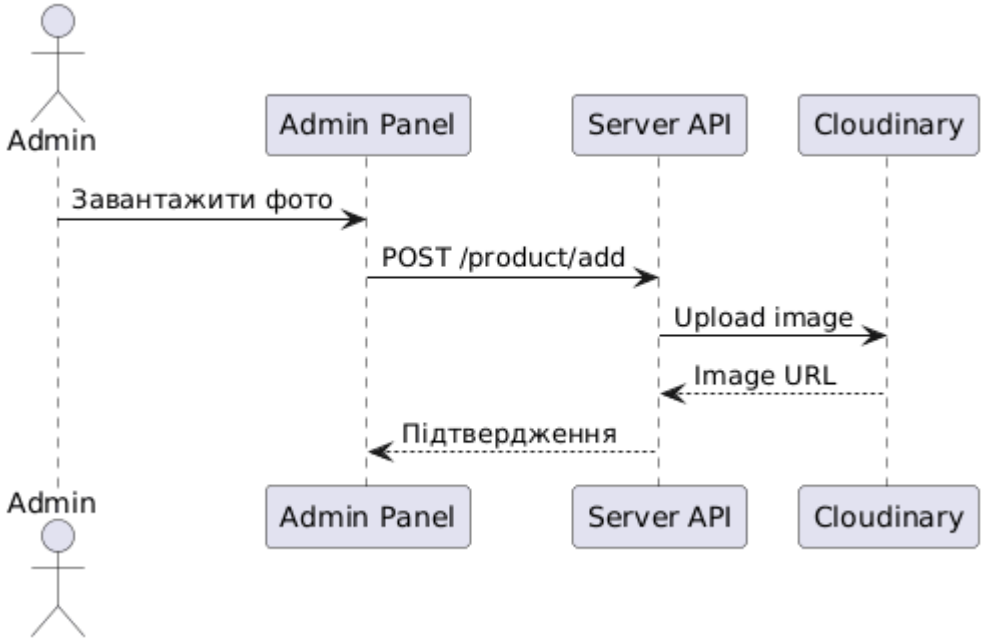

Рисунок 3.16. UML sequence діаграма завантаження зображення

Інтеграція сервісів штучного інтелекту (AI API).

Для функціонування підсистеми семантичного пошуку та рекомендацій (розглянутої в підрозділі 3.5), серверний рівень GreenCart діє як API-шлюз (API Gateway) між клієнтським застосунком та зовнішніми моделями штучного інтелекту (Large Language Models - LLM). Замість розгортання власних нейромереж, що потребувало б значних обчислювальних ресурсів, сервер приймає природномовний запит користувача, агрегує його із системним промптом (який задає контекст інтернет-магазину, правила обмеження бюджету або форматування рецептів) і направляє до зовнішнього AI API. Отриманий результат парситься бекендом, зіставляється з наявною базою товарів і повертається клієнту у вигляді структурованого JSON. Ця інтеграція трансформує застосунок у семантично обізнану систему без перевантаження власної інфраструктури.

Підсумовуючи, модульна інтеграція зовнішніх провайдерів створює розподілену, гнучку архітектуру, де сервер GreenCart зосереджений виключно на виконанні бізнес-логіки маркетплейсу, делегуючи вузькоспеціалізовані завдання профільним хмарним рішенням.

Висновки до розділу 3

У третьому розділі було виконано практичну реалізацію веборієнтованої інформаційної системи GreenCart відповідно до архітектури, спроєктованої на попередньому етапі. Основну увагу приділено розробці клієнтської та серверної частин застосунку, реалізації роботи з товарами й замовленнями, а також інтеграції зовнішніх сервісів, необхідних для повноцінного функціонування інтернет-магазину.

Клієнтську частину системи реалізовано як SPA-застосунок із використанням React, Vite та Tailwind CSS. Це дозволило створити динамічний інтерфейс користувача з підтримкою навігації між сторінками, оновленням компонентів без повного перезавантаження сторінки та зручним керуванням станом кошика. Для обміну даними із серверною частиною використано HTTP-запити через Axios, що забезпечує взаємодію клієнта з REST API системи. Також у клієнтській частині передбачено підтримку локалізації інтерфейсу та відображення цін у різних валютах.

Серверну частину реалізовано на базі Node.js та Express.js. У процесі розробки було розділено маршрути, middleware та контролери, що дало змогу структурувати бізнес-логіку системи й спростити подальший супровід коду. Для збереження даних використано MongoDB у поєднанні з Mongoose, що забезпечило роботу з основними сутностями системи: користувачами, товарами, адресами доставки та замовленнями.

Окремо було реалізовано функціонал керування каталогом товарів. Адміністративна частина системи дає змогу додавати нові товари, редагувати їхні характеристики, змінювати статус наявності та видаляти позиції з каталогу. Для кожного товару передбачено збереження основних атрибутів, зображень, ціни, категорії та локалізованих описів, що забезпечує коректне відображення товарів у клієнтському інтерфейсі.

Механізм оформлення замовлень реалізовано з урахуванням двох сценаріїв оплати: післяплати та онлайн-оплати через Stripe. Під час створення

замовлення система зберігає інформацію про склад кошика, адресу доставки, суму замовлення та спосіб оплати. Для онлайн-оплати використовується створення платіжної сесії Stripe, а підтвердження успішної транзакції обробляється за допомогою webhook-механізму. Такий підхід дозволяє не зберігати чутливі платіжні дані безпосередньо в системі GreenCart.

У межах розділу також було реалізовано модуль інтелектуального пошуку та рекомендацій. Його призначення полягає в обробці природномовних запитів користувача та формуванні добірки товарів відповідно до змісту запиту. На відміну від звичайного пошуку за назвою або категорією, цей модуль дає змогу враховувати контекст запиту, наприклад підбір продуктів для страви або пошук товарів у межах певного бюджету.

Для розширення функціональності системи було інтегровано зовнішні сервіси. Stripe використовується для обробки онлайн-платежів, Cloudinary — для зберігання й оптимізації зображень товарів, а AI API — для обробки запитів, пов’язаних із рекомендаціями та семантичним пошуком. Використання таких сервісів дозволило винести частину складних операцій за межі основного серверного застосунку та зменшити навантаження на власну інфраструктуру.

Отже, у третьому розділі було реалізовано основні програмні компоненти системи GreenCart, які забезпечують роботу інтернет-магазину: клієнтський інтерфейс, серверну логіку, взаємодію з базою даних, керування товарами, оформлення замовлень, онлайн-оплату, роботу із зображеннями та інтелектуальний пошук. Отримана програмна реалізація відповідає визначеним функціональним вимогам і створює основу для подальшого тестування, перевірки безпеки та оцінки продуктивності системи.

РОЗДІЛ 4
ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 Методика тестування вебзастосунку

Забезпечення якості (Quality Assurance) є критичним етапом життєвого циклу розробки вебзастосунку Greencart. Для систем електронної комерції цей етап має підвищену пріоритетність, оскільки застосунок опрацьовує персональні дані користувачів, виконує фінансові транзакції та керує консистентністю даних товарного каталогу. Основною метою тестування інформаційної системи GreenCart є верифікація відповідності реалізованого продукту специфікованим функціональним і нефункціональним вимогам, а також виявлення та усунення дефектів коду.

У межах даного проєкту застосовано комплексний підхід до тестування (Comprehensive Testing Strategy), який базується на перевірці як ізольованих компонентів, так і системи в цілому.

У процесі розробки тестування виконувалося переважно у форматі ручної та напівавтоматизованої перевірки, що є типовим підходом для індивідуального студентського проєкту. Для цього використовувалися локальний запуск клієнтської та серверної частин, перевірка REST-ендпоінтів через HTTP-клієнт, аналіз відповідей сервера в консолі розробника браузера, а також відстеження змін у базі даних після виконання ключових бізнес-сценаріїв. Окрему увагу було приділено не лише сценаріям коректної роботи системи, а й ситуаціям, у яких користувач вводить неповні дані, повторно надсилає форми або переходить між сторінками в неочікуваному порядку.

Під час тестування я зіткнувся з низкою практичних труднощів, характерних для full-stack розробки. Найскладнішими виявилися перевірка узгодженості стану кошика між клієнтом і сервером, налагодження сценарію онлайн-оплати через Stripe webhook, а також забезпечення коректної адаптивності інтерфейсу на мобільних екранах. Саме тому тестування виконувалося ітеративно: після кожного виявленого дефекту здійснювалося

локальне виправлення, повторний запуск сценарію та регресійна перевірка суміжних модулів.

Життєвий цикл процесу тестування (STLC [56]).

Процес валідації системи реалізовано за стандартизованим ітеративним алгоритмом:

1. Підготовка тестового середовища: розгортання локальних інстансів Node.js-сервера, підключення до тестової бази даних MongoDB та ініціалізація клієнтського застосунку.

2. Проєктування тест-кейсів: розробка тестових сценаріїв (Test Cases) на основі бізнес-вимог для позитивних (happy path) та негативних (edge cases) варіантів використання.

3. Виконання тестів: імітація дій користувача через UI та безпосереднє відправлення HTTP-запитів до REST API за допомогою спеціалізованих інструментів.

4. Аналіз результатів: зіставлення фактичної поведінки системи (Actual Result) з очікуваною (Expected Result).

5. Дебагінг та усунення дефектів: локалізація знайдених аномалій, внесення виправлень у програмний код та проведення регресійного тестування.

Рівні та види тестування Для досягнення максимального тестового покриття (Test Coverage) застосовано такі види тестування:

Функціональне тестування (Functional Testing): Перевірка коректності реалізації ключових бізнес-правил. Об'єктами тестування на цьому рівні були: CRUD-операції товарного каталогу, алгоритми управління станом кошика, розрахунок загальної вартості (включно зі зміною валют) та робота AI-модуля пошуку.

Інтеграційне тестування (Integration Testing): Верифікація коректності взаємодії між незалежними підсистемами. Зокрема, перевірявся обмін даними між клієнтським SPA (React) та бекендом (Express.js), транзакційні запити до MongoDB, а також стабільність двосторонньої комунікації із зовнішніми

сервісами - платіжним шлюзом Stripe (генерація сесій, обробка вебхуків) і хмарним сховищем Cloudinary.

Тестування інтерфейсу користувача (UI/UX Testing): Валідація клієнтської частини на предмет коректного рендерингу компонентів (рис. 4.1), адаптивності (рис. 4.2, 4.3)верстки Tailwind CSS на різних розмірах екранів, а також перевірка навігаційних потоків (Routing) та обробки станів завантаження і помилок на боці клієнта.

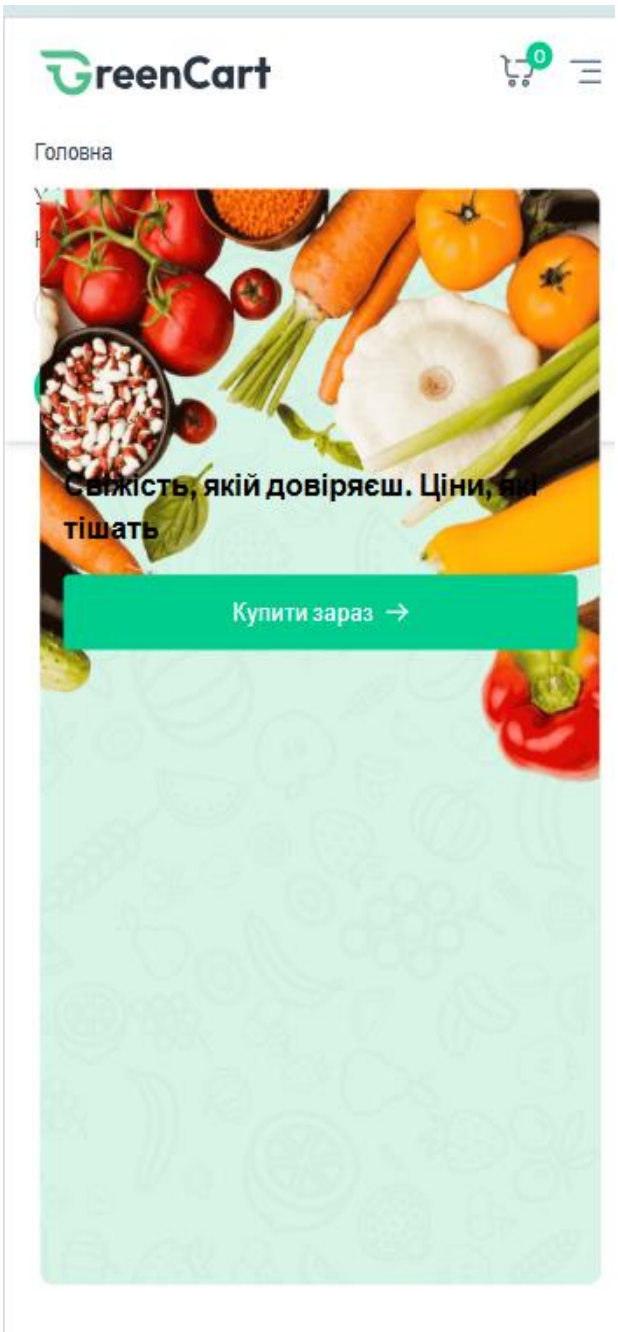

Рисунок 4.1. Скріншот інтерфейсу до змін адаптивності ч.1

Рисунок 4.2. Скріншот інтерфейсу до змін адаптивності ч.2

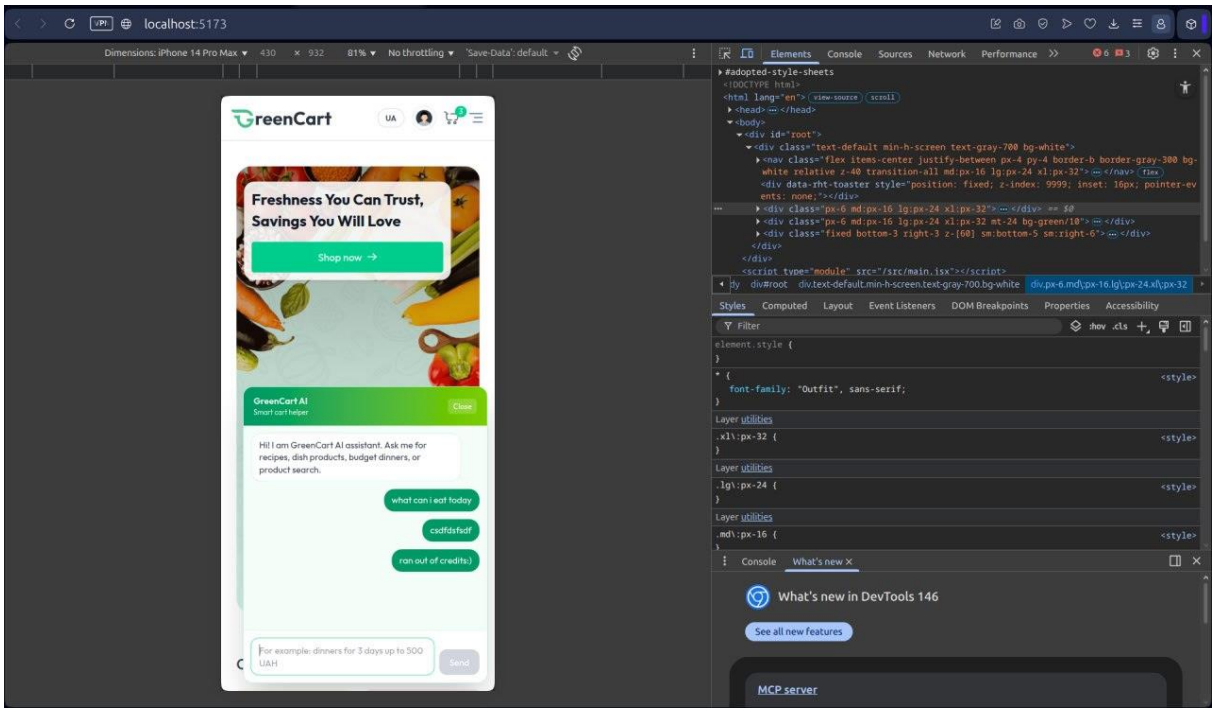

Рисунок 4.3. Скріншот інтерфейсу після змін адаптивності ч.2

Тестування безпеки (Security Testing): Аудит механізмів захисту інформації. Фокус було спрямовано на перевірку надійності автентифікації (JWT-токени), рольового контролю доступу (RBAC) до адміністративних маршрутів, санітизації вхідних даних для запобігання ін'єкціям та безпеки обробки платіжних ініціатив.

З точки зору програмної інженерії, застосована методика тестування розглядає вебзастосунок як систему кінцевих автоматів (State Machines)[57], де перевіряється детермінованість переходів між станами у відповідь на валідні та невалідні вхідні дані. Такий багаторівневий підхід дозволяє гарантувати стабільність архітектури, цілісність даних та високий рівень надійності системи GreenCart перед етапом її дослідної експлуатації.

4.2 Тестування основних функціональних сценаріїв

Функціональне тестування (Functional Testing) було спрямоване на верифікацію ключових бізнес-процесів (End-to-End сценаріїв) інтернет-магазину. Головна мета цього етапу - підтвердження того, що всі модулі системи

коректно обробляють транзакції, зберігають консистентність даних та відповідають специфікованим вимогам експлуатації.

У процесі тестування було перевірено такі базові тестові сценарії :

1. Управління доступом: Реєстрація та Автентифікація (IAM) а саме:

Проведено перевірку обробки форм. При передачі валідного payload (електронна пошта, пароль) сервер коректно хешує пароль, створює запис у колекції User та повертає статус 201 Created. При спробі реєстрації з наявним email або невалідними даними система успішно генерує виняток і повертає 400 Bad Request.

Тестування підтвердило, що при введенні правильних облікових даних сервер успішно генерує сесійний JWT-токен (200 OK). Спроби входу з хибним паролем блокуються (401 Unauthorized). Також підтверджено роботу Middleware: захищені маршрути (наприклад, профіль користувача або адмін-панель) недоступні для неавторизованих сесій (403 Forbidden).

У ході ручної перевірки сценаріїв входу та виходу було також виявлено, що коректна робота cookie-based автентифікації залежить не лише від серверної логіки, а й від правильних параметрів cookie у браузері. Це особливо проявлялося під час тестування в локальному середовищі, де різниця між localhost, портами клієнта і сервера та налаштуваннями credentials у запитах безпосередньо впливала на стабільність сесії. Після уточнення конфігурації було підтверджено коректний доступ до захищених маршрутів.

2. Взаємодія з каталогом товарів (Catalog Browsing).

Перевірено механізм читання ресурсів каталогу. Запити клієнтського застосунку до API успішно повертають масив товарів.

Результати перевірки: Локалізовані назви та описи відтворюються коректно. Ціни відображаються відповідно до обраної валюти. Медіафайли (зображення) успішно завантажуються через CDN Cloudinary. Сторінка детального перегляду товару коректно відпрацьовує запит за унікальним ідентифікатором (ID).

Під час перевірки сторінок каталогу та картки товару особливу увагу було приділено мультимовності інтерфейсу. Практика показала, що навіть незначна неузгодженість у назвах полів моделі товару між сервером і клієнтом може призводити до часткового зникнення назв або описів після перемикання мови. Саме тому після кожного редагування схеми товару виконувалася повторна перевірка відображення локалізованих полів у всіх основних вітринах застосунку.

3. Управління станом кошика (Cart State Management) Кошик є найбільш динамічним компонентом клієнтської частини. Проведено тестування мутацій стану:

Результати перевірки: Додавання нових товарів, зміна їхньої кількості (інкремент/декремент) та видалення позицій працюють без затримок. Алгоритм розрахунку загальної вартості (Subtotal) динамічно і безпомилково перераховує суму при будь-якій зміні стану кошика, враховуючи можливі акційні ціни.

У процесі тестування оформлення замовлення було встановлено, що цей сценарій є найбільш чутливим до помилок, оскільки поєднує відразу кілька підсистем: кошик, адресу доставки, перевірку авторизації, створення замовлення та платіжний модуль. Тому будь-яка локальна зміна в одній з цих частин вимагала повторного проходження повного сценарію покупки, щоб переконатися у відсутності регресій.

4. Пайплайн оформлення замовлення (рис. 4.4) Тестування критичного бізнес-сценарію - конвертації стану кошика у фіксоване замовлення.

5. Результати перевірки: Форма оформлення коректно валідує адресу доставки. Після підтвердження (Submit) сервер успішно формує документ у колекції Order. Історичний зріз даних (ціни та назви товарів на момент покупки) зберігається незмінним. Клієнт отримує підтвердження успішного створення замовлення.

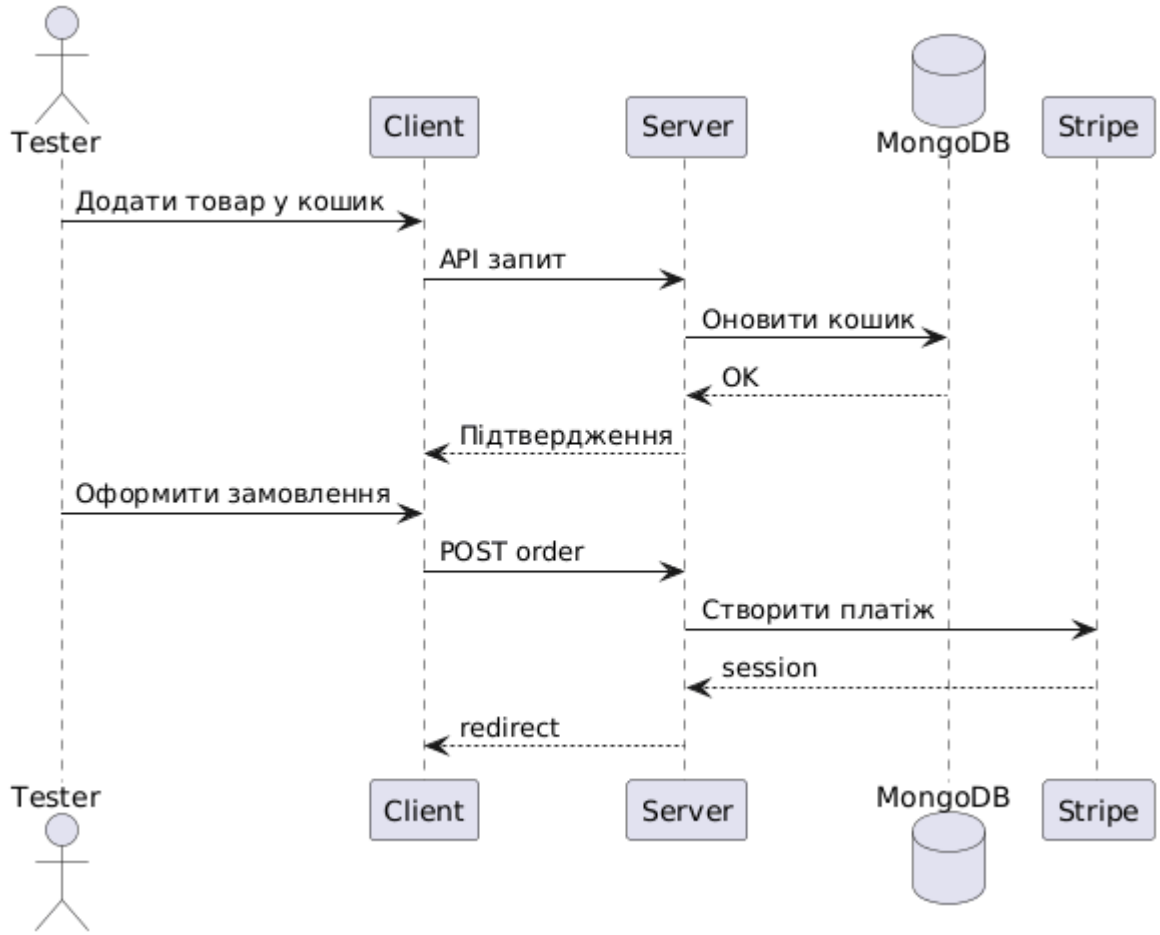

Рисунок 4.4. UML sequence діаграма тестування сценарію покупки

6. Інтеграція платіжного шлюзу (Payment Gateway) Окремо перевірено асинхронний сценарій онлайн-оплати через Stripe.

Результати перевірки: При виборі відповідного методу оплати сервер безпомилково генерує Stripe Checkout Session. Перенаправлення користувача на платіжну сторінку відбувається коректно. Після імітації успішної транзакції (через Stripe CLI у тестовому режимі) сервер успішно приймає Webhook, валідує його сигнатуру та атомарно оновлює статус замовлення в базі даних на paid.

Окрім перевірки позитивних сценаріїв, у ході функціонального тестування було виявлено кілька практичних проблем, що мали значення для стабільності системи. По-перше, під час налагодження кошика з’ясувалося, що найменші розбіжності між локальним станом React і даними, збереженими на сервері, одразу призводять до некоректного відображення кількості товарів після повторного входу користувача. Для усунення цієї проблеми було виконано

додаткову перевірку сценаріїв синхронізації після оновлення сторінки, авторизації та повторного завантаження профілю.

По-друге, окрему складність викликала перевірка сценарію онлайн-оплати. Основна проблема полягала в тому, що в локальному середовищі недостатньо лише створити платіжну сесію Stripe; необхідно також переконатися, що webhook дійсно доходить до сервера, проходить перевірку сигнатури та коректно змінює стан замовлення в базі даних. Через це тестування платіжного модуля виконувалося в декілька етапів: перевірка створення Checkout Session, перевірка редіректу користувача, емуляція успішної транзакції та контроль факту оновлення поля `isPaid`.

По-третє, при перевірці API було помічено, що окремі ендпоінти потребують додаткової уваги з точки зору архітектурної строгості. Зокрема, частина бізнес-логіки кошика покладається на передавання `userId` у тілі запиту, хоча більш надійним підходом є використання ідентифікатора користувача безпосередньо з токена автентифікації. Таке спостереження не зупиняло роботу системи, проте було зафіксоване як важливий результат тестування та як напрям подальшого вдосконалення безпеки.

За результатами функціонального тестування підтверджено високий рівень надійності системи. Всі ключові End-to-End сценарії відпрацьовують у штатному режимі, алгоритми розрахунків не містять логічних помилок, а інтеграційні точки із зовнішніми API функціонують стабільно.

4.3 Аналіз продуктивності та часу відповіді системи

Оцінка продуктивності є обов'язковою складовою перевірки нефункціональних вимог (Non-Functional Requirements) до вебзастосунків. У контексті електронної комерції такі метрики, як час затримки (Latency), пропускна здатність (Throughput) та швидкість рендерингу на стороні клієнта, безпосередньо впливають на користувацький досвід (UX) та показники конверсії.

Аналіз продуктивності системи GreenCart здійснювався на основі профілювання основних API-ендпоінтів та вимірювання часу відгуку сервера (Server Response Time). За результатами тестування, базовий час відповіді бекенда становить від 100 до 300 мілісекунд (мс), що повністю відповідає сучасним галузевим стандартам для вебзастосунків. А час відповіді клієнта - від 50 до 70 мс (рис. 4.6).

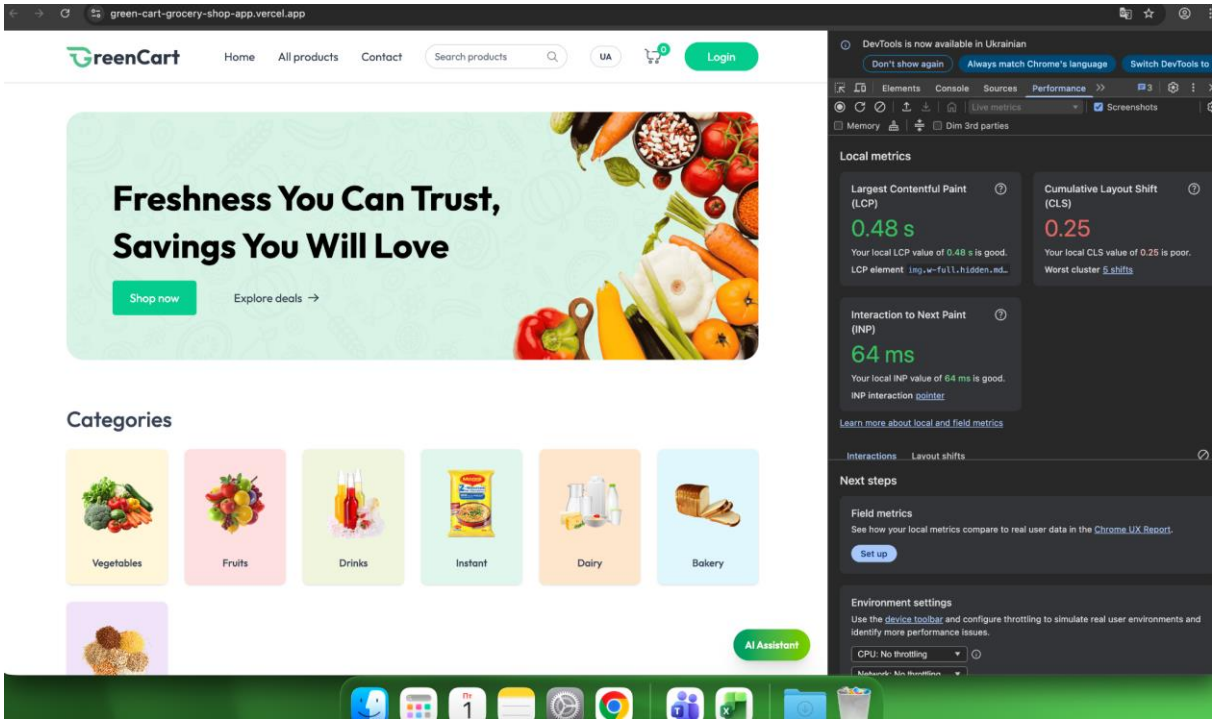

Рисунок 4.5. Час відгуку клієнта, отриманий за допомогою вбудованих інструментів розробника

З точки зору обчислювального навантаження, результати тестування диференційовано за трьома категоріями операцій:

1.Операції читання (Read-Intensive I/O)

Найвищу швидкодію демонструють ресурси, що відповідають за постачання даних: GET /api/product/list та GET /api/product/id. Мінімальна затримка пояснюється відсутністю складної бізнес-логіки та ефективністю MongoDB у виконанні прямих запитів на читання.

Транзакційні операції запису (Write-Intensive Operations).

Помірне зростання часу відповіді зафіксовано для ендпоінтів мутації стану: POST /api/cart/update та POST /api/order/cod. Обробка цих запитів потребує додаткових процесорних тактів для десеріалізації вхідних даних (JSON payload), виконання критеріїв валідації Mongoose, математичних обчислень (перерахунок вартості кошика) та запису оновленого документа в базу даних.

2. Операції з високою затримкою.

(External API Dependencies) Найбільш ресурсоємними виявилися процеси, що покладаються на сторонні сервіси (Third-party integrations):

- Генерація платіжних сесій Stripe.
- Асинхронне завантаження та оптимізація зображень через Cloudinary.
- Звернення до AI-модуля для обробки природномовних запитів.
- Час виконання таких операцій варіюється від 500 мс до кількох секунд, оскільки до часу обробки на власному сервері додаються мережеві затримки (Network Latency) та час виконання алгоритмів на стороні SaaS-провайдерів.

3. Фактори архітектурної оптимізації.

Досягнутий рівень продуктивності є результатом застосування комплексу сучасних інженерних патернів:

Асинхронність бекенда: Використання подієвого циклу (Event Loop[59]) та неблокуючої моделі вводу-виводу (Non-blocking I/O) платформи Node.js дозволяє одному потоку ефективно обробляти десятки одночасних підключень без блокування ресурсів сервера.

Делегування навантаження (CDN): Передача графічного трафіку хмарній інфраструктурі Cloudinary нівелює проблему вузького місця пропускної здатності локального сервера.

Оптимізація клієнтського рендерингу: Архітектура Single Page Application (SPA) на базі React зводить до мінімуму обсяг переданих даних, оскільки браузер завантажує DOM-дерево лише один раз, а надалі здійснює асинхронний обмін виключно легкими JSON-пакетами.

Підсумкові результати навантажувального тестування доводять, що реалізована архітектура GreenCart здатна стабільно обслуговувати конкурентні запити типового інтернет-магазину середнього масштабу без критичної деградації швидкодії.

4.4 Оцінка ефективності реалізованих алгоритмів

Продуктивність вебзастосунку безпосередньо залежить від обчислювальної складності (Computational Complexity, Big “O” [60]) реалізованих алгоритмів та ефективності обраних структур даних. У процесі розробки системи GreenCart основний фокус було спрямовано на оптимізацію процесів пошуку, обробки стану кошика, формування замовлень та генерації інтелектуальних рекомендацій. Оцінка цих алгоритмів проводилася за критеріями часової складності (Time Complexity[60]).

1. Алгоритм пошуку товарів (Catalog Search).

Пошук товарів за назвою або описом реалізовано на рівні бази даних MongoDB.

Без використання оптимізації такий пошук вимагає повного сканування колекції (Collection Scan), що має лінійну асимптотичну складність O(n), де n - загальна кількість документів у каталозі.

Для забезпечення масштабованості системи алгоритм покладається на механізми індексування MongoDB (B-Tree індекси та текстові індекси). Використання індексів знижує часову складність операції пошуку до логарифмічної O(log n), що гарантує високу швидкодію навіть при значному розширенні асортименту інтернет-магазину.

2. Алгоритм управління станом кошика (Cart State Management).

Алгоритм обробки кошика відповідає за мутації стану: додавання нових позицій (SKU), інкремент/декремент кількості та видалення.

Перевірка наявності товару у масиві кошика та зміна його кількості виконується за час O(n), де n - кількість унікальних позицій у кошику користувача.

Алгоритм перерахунку загальної вартості (Subtotal Calculation) також потребує одного проходу по масиву товарів, маючи лінійну складність O(n). Оскільки в контексті електронної комерції розмір кошика (n) є статистично малим значенням, ця операція виконується практично миттєво (за мікросекунди) і не створює обчислювального навантаження ані на клієнті, ані на сервері.

3. Алгоритм формування замовлення (Order Processing).

Процес конвертації кошика у фіксоване замовлення включає кілька послідовних кроків: отримання актуального масиву товарів, валідацію наявності (Inventory Check), фінальний розрахунок суми та запис документа в базу даних.

Валідація та створення історичного зрізу (snapshot) товарів виконується за час O(n) (де n - кількість товарів у замовленні).

Операція запису (Insert) сформованого документа Order у MongoDB має константну складність O(1). Таким чином, загальна ефективність алгоритму є лінійною і цілком задовольняє вимоги транзакційних систем.

4. Алгоритм інтелектуальних рекомендацій (AI Recommendation Engine).

Алгоритм формування персоналізованих рекомендацій та семантичного підбору інгредієнтів (наприклад, для рецептів) є найбільш комплексним з точки зору обчислень.

Локальний семантичний аналіз тексту вимагав би використання ресурсоємних NLP-моделей. Однак архітектурне рішення делегувати цю задачу зовнішньому AI API перетворює локальну обчислювальну складність для сервера GreenCart на O(1), додаючи лише константний час мережевої затримки (Network Latency).

Після отримання структурованої відповіді від AI, сервер виконує пошук відповідних товарів (за масивом згенерованих ключових слів) у базі даних, що, завдяки індексам, займає час O(k log m), де k - кількість рекомендованих позицій, а m - загальний розмір каталогу.

Загалом, оцінка ефективності реалізованих алгоритмів підтверджує, що ядро системи побудовано з урахуванням оптимальних патернів проєктування.

Уникання алгоритмів із квадратичною O(n^2) або експоненціальною O(2^n) складністю гарантує стабільну та передбачувану роботу інтернет-магазину при зростанні навантаження.

Логічний алгоритм роботи AI-модуля.

Алгоритм роботи AI-модуля рекомендацій (рис. 4.7) у системі GreenCart складається з послідовності етапів обробки запиту користувача, кожен з яких виконує окрему функцію у процесі формування рекомендацій.

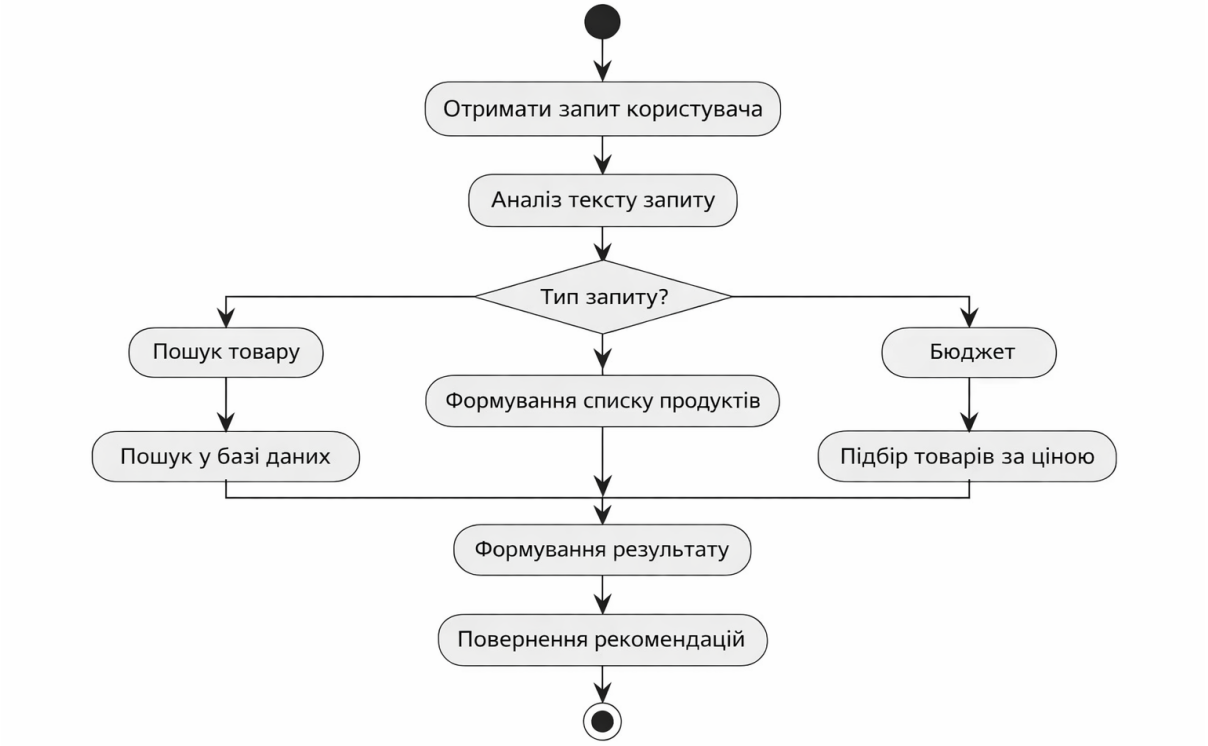

Рисунок 4.6. Блок-схема алгоритму рекомендацій

Отримання запиту користувача.

На цьому етапі система приймає HTTP-запит від клієнтського застосунку через API-ендпоінт /api/ai/query, який містить текст запиту користувача та, за необхідності, додаткові параметри, такі як бюджет, назва страви або список наявних продуктів. Отриманий запит передається до серверного модуля обробки рекомендацій для подальшого аналізу.

Аналіз тексту запиту.

Система виконує аналіз тексту запиту користувача, що включає нормалізацію тексту, видалення службових символів і визначення ключових слів

або фраз. Метою цього етапу є виділення змістовних параметрів запиту, які дозволяють визначити намір користувача та підготувати дані для наступного етапу обробки.

Визначення типу запиту.

На основі результатів аналізу тексту система визначає тип запиту користувача. Зокрема, система перевіряє, чи стосується запит формування рецепта, підбору продуктів для конкретної страви або планування покупок у межах заданого бюджету. Від визначеного типу запиту залежить подальша логіка виконання алгоритму.

Обробка запиту типу «Рецепт».

Якщо система визначає, що запит користувача пов’язаний із приготуванням страви або отриманням рецепта, виконується пошук відповідних продуктів у базі даних. Система аналізує складники рецепта та підбирає товари з каталогу, які можуть бути використані для приготування зазначеної страви.

Обробка запиту типу «Бюджет».

Якщо запит користувача містить обмеження за бюджетом, система отримує значення доступних коштів і виконує підбір товарів таким чином, щоб загальна вартість сформованого списку не перевищувала заданий бюджет. У процесі відбору враховується ціна товарів, їх доступність та відповідність категорії продуктів.

Формування списку продуктів.

Після виконання відповідної гілки алгоритму система формує підсумковий список продуктів, які відповідають умовам запиту користувача. До списку включаються назви товарів, їх кількість, ціна та інші характеристики, необхідні для подальшого відображення результату в інтерфейсі системи.

Повернення результату користувачу.

На завершальному етапі алгоритму система формує структуровану відповідь у форматі JSON і надсилає її клієнтському застосунку через HTTP-протокол. Отримані дані відображаються користувачу у вигляді списку рекомендованих продуктів або рецепта відповідно до виконаного сценарію.

Висновки до розділу 4

У четвертому розділі було проведено тестування розробленого вебзастосунку GreenCart, а також виконано оцінку його продуктивності та ефективності основних алгоритмів. Перевірка охоплювала як окремі функціональні можливості системи, так і повні сценарії роботи користувача з інтернет-магазином.

На етапі тестування було перевірено роботу основних модулів системи: реєстрацію та автентифікацію користувачів, перегляд каталогу товарів, додавання товарів до кошика, зміну їх кількості, оформлення замовлення та обробку платежів. Окрему увагу приділено перевірці взаємодії клієнтської частини із сервером через REST API, а також коректності збереження й оновлення даних у MongoDB.

Під час тестування бізнес-логіки підтверджено, що основні сценарії використання системи виконуються коректно. Користувач може пройти повний шлях від перегляду товару до створення замовлення, а адміністратор має можливість керувати товарами та переглядати замовлення. Також було перевірено роботу механізму онлайн-оплати через Stripe, зокрема створення платіжної сесії та оновлення статусу замовлення після підтвердження платежу через webhook.

Аналіз продуктивності показав, що серверна частина системи забезпечує прийнятний час відповіді для основних API-запитів. У межах проведених вимірювань середній час відповіді становив приблизно 100–300 мс для типових операцій, що є достатнім для комфортної роботи користувача з вебзастосунком. Операції, пов’язані із зовнішніми сервісами, зокрема Stripe, Cloudinary та AI API, можуть виконуватися довше, оскільки залежать не лише від серверної логіки GreenCart, а й від швидкодії сторонніх платформ.

Оцінка алгоритмічної ефективності показала, що реалізовані механізми пошуку, роботи з кошиком і створення замовлень відповідають потребам системи на поточному етапі її розвитку. Використання індексів у базі даних дає

змогу пришвидшити пошук і фільтрацію товарів, а збереження стану кошика та замовлень у структурованому вигляді спрощує обробку основних користувацьких дій. Для подальшого масштабування системи доцільно передбачити додаткову оптимізацію запитів, кешування окремих даних і розширене логування помилок.

Результати тестування показали, що розроблений вебзастосунок GreenCart виконує основні функціональні вимоги та забезпечує коректну роботу ключових модулів. Система може бути використана для дослідної експлуатації, а отримані результати тестування створюють основу для подальшого вдосконалення продуктивності, безпеки та зручності користування.

ВИСНОВКИ

У кваліфікаційній роботі досягнуто поставленої мети - розроблено веборієнтовану систему інтернет-магазину продуктів харчування GreenCart з використанням сучасних технологій веброзробки. Створений застосунок забезпечує перегляд каталогу товарів, роботу з кошиком, оформлення замовлень, онлайн-оплату, керування товарами через адміністративну частину та використання рекомендаційного модуля.

У процесі виконання роботи було досягнуто наступні цілі:

Проаналізовано сучасні системи електронної комерції, їхні особливості, архітектурні підходи та технології розробки вебзастосунків. Визначено, що для реалізації інтернет-магазину доцільним є використання клієнт-серверної архітектури, REST API та MERN-стеку, оскільки такий підхід забезпечує модульність, масштабованість і зручність подальшого розвитку системи.

Визначено функціональні та нефункціональні вимоги до GreenCart, розроблено загальну архітектуру застосунку, спроєктовано структуру бази даних, моделі взаємодії користувачів із системою, REST API та механізми авторизації і безпеки. Для збереження даних використано документоорієнтовану базу даних MongoDB, що дозволило гнучко організувати зберігання інформації про користувачів, товари, адреси доставки та замовлення.

Реалізовано клієнтську частину вебзастосунку у вигляді SPA-інтерфейсу, що забезпечує зручну взаємодію користувача з каталогом, кошиком, замовленнями та адміністративною панеллю. Серверну частину реалізовано на основі Node.js та Express.js, що забезпечило обробку запитів, взаємодію з базою даних, автентифікацію користувачів і виконання основної бізнес-логіки інтернет-магазину.

Реалізовано функціонал керування товарами, обробки замовлень, завантаження зображень, підтримки багатомовності та конвертації валют. Для забезпечення онлайн-оплати інтегровано сервіс Stripe, а для зберігання зображень товарів використано хмарний сервіс Cloudinary. Окремо досліджено

та реалізовано систему рекомендацій товарів, що розширює функціональні можливості інтернет-магазину та покращує користувацький досвід.

Проведено тестування основних функціональних сценаріїв роботи системи, зокрема реєстрації та авторизації користувача, перегляду каталогу, роботи з кошиком, оформлення замовлення, оплати, керування товарами та взаємодії з адміністративною частиною. Результати тестування підтвердили коректність роботи основних модулів системи та відповідність розробленого застосунку визначеним вимогам.

Практичне значення : створення працездатної веборієнтованої системи електронної комерції, яка реалізує основні бізнес-процеси інтернет-магазину продуктів харчування. Розроблений застосунок може бути використаний як основа для подальшого розширення функціональності, зокрема додавання нових платіжних методів, розширеної аналітики, персоналізованих рекомендацій та інструментів адміністрування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Розвиток електронної комерції в Україні в сучасних умовах URL: https://www.researchgate.net/publication/379595355_Rozvitok_elektronnoi_komerci_v_Ukraini_v_sucasnih_umovah (дата звернення: 24.04.2026).

2. REST API Architectural Style. RESTful API Tutorial. URL: https://restfulapi.net (дата звернення: 24.04.2026).

3. Розвиток технологій штучного інтелекту у вебдодатках електронної комерції URL : https://zcit.kubg.edu.ua/index.php/journal/article/view/83 (дата звернення: 21.05.2026).

4. First global e-commerce business report: AI, social commerce & sustainability lead 2025 trends URL : https://group.dhl.com/en/media-relations/press-releases/2025/dhl-e-commerce-trends-report-2025-business-edition.html?utm_source=chatgpt.com (дата звернення: 01.05.2026).

5. TechTarget. B2C (business-to-consumer).URL : https://www.techtarget.com/searchcustomerexperience/definition/B2C (дата звернення: 01.05.2026).

6. Stripe. Official Documentation. URL: https://docs.stripe.com (дата звернення: 24.04.2026).

7. Cloudinary. Official Documentation. URL: https://cloudinary.com/documentation (дата звернення: 24.04.2026).

8. n8n. Official Documentation. URL: https://docs.n8n.io (дата звернення: 24.04.2026).

9. ISO/IEC 25002:2024 documentation. URL : https://www.iso.org/ru/standard/78175.html (дата звернення: 29.04.2026).

10. MongoDB. Official Documentation. URL: https://www.mongodb.com/docs (дата звернення: 24.04.2026).

11. Rule-Based Systems in AI // GeeksforGeeks. URL: https://www.geeksforgeeks.org/rule-based-system-in-artificial-intelligence/ (дата звернення: 01.05.2026).

12. HTTPS Protocol. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Glossary/HTTPS (дата звернення: 24.04.2026).

13. JSON Web Token (JWT). Official Documentation. URL: https://jwt.io (дата звернення: 24.04.2026).

14.MDN Web Docs. HTTP cookies [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies (дата звернення: 30.04.2026).

15. Web Services: Concepts, Architectures and Applications. Springer, 2004.

16. Understanding the Layered Architecture Pattern: A Comprehensive Guide URL : https://dev.to/yasmine_ddec94f4d4/understanding-the-layered-architecture-pattern-a-comprehensive-guide-1e2j (дата звернення: 01.05.2026).

17. Compass documentation (official) URL: https://www.mongodb.com/docs/compass/

18. HTTP METHODS. URL: https://developer.mozilla.org/ru/docs/Web/HTTP/Reference/Methods (дата звернення: 29.04.2026).

19. Best practices for REST-api URL : https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design (дата звернення: 24.04.2026).

20.Що таке SPA URL: https://wezom.com.ua/ua/blog/chto-takoe-spa-prilozheniya (дата звернення: 24.04.2026).

21.React. Official Documentation. URL: https://react.dev (дата звернення: 24.04.2026).

22.Component-based approach URL: https://www.mendix.com/blog/what-is-component-based-architecture/ (дата звернення: 24.04.2026)

23. MERN-stack explained. URL: https://www.mongodb.com/resources/languages/mern-stack (дата звернення: 29.04.2026).

24. How to choose between Ecommerce platforms. URL: https://www.namaait.com/en/articles/97/how-to-choose-between-ecommerce-solutions (дата звернення: 29.04.2026).

25. OpenCart. URL: https://www.opencart.com (дата звернення: 29.04.2026).

26. Magento Open Source. URL: https://magento.com (дата звернення: 29.04.2026).

27. WooCommerce. URL: https://woocommerce.com (дата звернення: 29.04.2026).

28. Shopify. URL: https://www.shopify.com (дата звернення: 29.04.2026).

29. Express.js. Official Documentation. URL: https://expressjs.com (дата звернення: 24.04.2026).

30. Сучасний підручник по Javascript URL: https://uk.javascript.info/

31. Angular Documentation. URL: https://angular.io (дата звернення: 29.04.2026).

32 Vue official website URL: https://vuejs.org/ (дата звернення: 29.04.2026).

33. Vite. Official Documentation. URL: https://vitejs.dev/guide (дата звернення: 24.04.2026).

34. Tailwind CSS. Official Documentation. URL: https://tailwindcss.com/docs (дата звернення: 24.04.2026).

35. React Router. Official Documentation. URL: https://reactrouter.com (дата звернення: 24.04.2026).

36. Axios. Official Documentation. URL: https://axios-http.com (дата звернення: 24.04.2026).

37. Node.js. About Node.js. URL: https://nodejs.org/en/about/ (дата звернення: 01.05.2026).

38. Express.js. Using middleware URL:https://expressjs.com/en/guide/using-middleware.html (дата звернення: 01.05.2026).

39. Mongoose. Official Documentation. URL: https://mongoosejs.com/docs (дата звернення: 24.04.2026).

40. MongoDB bson documentation URL: https://www.mongodb.com/docs/php-library/current/reference/bson/

41.Національний банк України. Open Data API for developers. URL: https://bank.gov.ua/ua/open-data/api-dev (дата звернення: 29.04.2026).

42. W3C. Internationalization (i18n) URL:https://www.w3.org/International/ (дата звернення: 01.05.2026).

43. Webhooks. Mozilla Developer Network (MDN) Documentation. URL: https://developer.mozilla.org/en-US/docs/Web/API/Webhooks_API (дата звернення: 30.04.2026).

44. PCI Security Standards Council URL : https://www.pcisecuritystandards.org/pci_security/ (дата звернення: 01.05.2026)

45. OWASP Foundation. OWASP Top Ten Web Application Security Risks [Електронний ресурс]. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 30.04.2026).

46. bcrypt. Official Documentation (NPM bcrypt package) [Електронний ресурс]. URL: https://www.npmjs.com/package/bcrypt (дата звернення: 30.04.2026).

47. Node.js. Official Documentation.Blocking vs non-blocking URL: https://nodejs.org/learn/asynchronous-work/overview-of-blocking-vs-non-blocking (дата звернення: 24.04.2026).

48. Jurafsky D., Martin J. Speech and Language Processing. 3rd ed. Draft. URL: https://web.stanford.edu/~jurafsky/slp3/ (дата звернення: 01.05.2026).

49. Information Retrieval Overview // Elastic. URL: https://www.elastic.co/what-is/information-retrieval (дата звернення: 01.05.2026).

50. Semantic Search // Google Cloud. URL: https://cloud.google.com/learn/what-is-semantic-search (дата звернення: 01.05.2026).

51Knapsack Problem // cp-algorithms. URL: https://cp-algorithms.com/dynamic_programming/knapsack.html (дата звернення: 01.05.2026).

52. 0/1 Knapsack Problem // GeeksforGeeks. URL: https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/ (дата звернення: 01.05.2026).

53. Autocomplete Feature Explained // Algolia. URL: https://www.algolia.com/doc/guides/building-search-ui/what-is-autocomplete/ (дата

звернення: 01.05.2026).

54. Collaborative Filtering Explained // IBM. URL: https://www.ibm.com/topics/collaborative-filtering (дата звернення: 01.05.2026

55. Cloud Computing Models (SaaS, PaaS) // Microsoft Azure.

https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-saas/ (дата звернення: 01.05.2026.)

56. Software Testing Life Cycle Url : https://www.geeksforgeeks.org/software-testing-life-cycle-stlc/ (дата звернення: 01.05.2026.)

57. State Design Pattern // Refactoring.Guru. URL: https://refactoring.guru/design-patterns/state (дата звернення: 01.05.2026).

58. Multer. Official Documentation. URL: https://www.npmjs.com/package/multer (дата звернення: 24.04.2026).

59. Сучасний підручник js. Цикл подій (event loop): мікрозавдання (microtasks) та макрозавдання (macrotasks) URL : https://uk.javascript.info/event-loop (дата звернення: 01.05.2026)

60.Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. Introduction to Algorithms - 3rd ed. - Cambridge (MA): MIT Press, 2009. - 1312 p.

61. Бондарчук, А. П., & Глушак, О. М. (2025). Предиктивне управління оновленнями програмного забезпечення в інтернеті речей. Зв'язок, (5), 13-17.

62. Dmytro M. Bodnenko, Oleksandra V. Lokaziuk, Sergiy Radchenko, Volodymyr Proshkin, Serhiy D Shymchyk. The development and application of VBA macros in the teaching of science and mathematics. 8th Workshop for Young Scientists in Computer Science & Software Engineering (CS&SE@SW 2025), Kryvyi Rih, Ukraine, November 21, 2025

63. Яскевич В. О. JavaScript-бібліотека React : навч. посіб. для студентів спеціальності «Комп’ютерні науки». Київ : Київський університет імені Бориса Грінченка, 2023.

Додаток А

Реалізація Mongoose-моделей.

Фрагмент опису моделі користувача User.js, що використовується для збереження облікових даних користувача та вмісту кошика:

```
import mongoose from "mongoose";
const userSchema = new mongoose.Schema({
name:{type: String, required: true},
email:{type: String, required: true, unique:true},
password:{type: String, required: true},
cartItems:{type: Object, default: {}},
}, {minimize: false})
const User = mongoose.models.user || mongoose.model('user', userSchema)
export default User
```

Фрагмент моделі товару Product.js, що визначає мультимовні поля, ціну, акційну вартість, категорію та наявність:

```
import mongoose from "mongoose";
const productSchema = new mongoose.Schema({
name:{type: String, required: true},
nameUk:{type: String, required: true},
description:{type: Array, required: true},
descriptionUk:{type: Array, required: true},
price:{type: Number, required: true},
offerPrice:{type: Number, required: true},
image:{type: Array, required: true},
category:{type: String, required: true},
inStock:{type: Boolean, default: true},
}, {timestamps: true})
const Product = mongoose.models.product || mongoose.model('product', productSchema)
export default Product
```

Фрагмент моделі замовлення Order.js, яка зв'язує користувача, товари, адресу доставки та статус оплати:

```
import mongoose from "mongoose";
const orderSchema= new mongoose.Schema({
userId: { type: mongoose.Schema.Types.ObjectId, required: true, ref: 'user' },
items: [
{
```

```
      product: { type: mongoose.Schema.Types.ObjectId, required: true, ref:
'product' },
      quantity:{type: Number, required: true}
      }
      ],
      amount:{type: Number, required: true},
      address: { type: mongoose.Schema.Types.ObjectId, required: true, ref:
'address' },
      status:{type: String, default: 'Order Placed'},
      paymentType:{type: String, required: true},
      isPaid:{type: Boolean, required: true, default:false },
      }, {timestamps: true })
      const Order = mongoose.models.order || mongoose.model('order', orderSchema)
      export default Order
```

Додаток Б

Логіка роботи Stripe-контролера.

Нижче наведено фрагмент функції placeOrderStripe з файла orderController.js, яка формує замовлення, обчислює суму, створює Checkout Session у Stripe та повертає клієнту URL для переходу до оплати:

```
export const placeOrderStripe = async (req, res) => {
try {
const { items, address } = req.body;
const userId = req.user?._id;
const { origin } = req.headers;
if (!userId) {
return res.json({ success: false, message: "Not authorized" });
}
if (!address || !items || items.length === 0) {
return res.json({ success: false, message: "Invalid data" });
}
const productData = [];
let amount = 0;
for (const item of items) {
const product = await Product.findById(item.product);
if (!product) {
return res.json({ success: false, message: "Product not found" });
}
productData.push({
name: product.name,
price: product.offerPrice,
quantity: item.quantity,
});
amount += product.offerPrice * item.quantity;
}
amount += Math.floor(amount * 0.02);
const order = await Order.create({
userId,
items,
amount,
address,
paymentType: "ONLINE",
});
const stripe = new Stripe(process.env.STRIPE_SECRET_KEY);
const line_items = productData.map((item) => ({
price_data: {
```

```
currency: "usd",
product_data: { name: item.name },
unit_amount: Math.floor(item.price + item.price * 0.02) * 100,
},
quantity: item.quantity,
}));
const session = await stripe.checkout.sessions.create({
line_items,
mode: "payment",
success_url: `${origin}/loader?next=my-orders`,
cancel_url: `${origin}/cart`,
metadata: {
orderId: order._id.toString(),
userId: userId.toString(),
},
});
return res.json({
success: true,
url: session.url,
});
} catch (error) {
return res.json({
success: false,
message: error.message,
});
}
};
```

Додаток В

Підключення AI модуля.

Підключення AI-модуля виконується через контролер aiController.js, який приймає запит користувача, визначає сценарій взаємодії, формує структуровану відповідь і, за наявності ключа OPENAI_API_KEY, звертається до OpenAI API:

```
import { loadCatalog, productToChatItem, rankProductsBySemanticQuery } from "../services/ai/aiCatalogService.js";
import {
buildRecipeFromCatalog,
decideAction,
planDinnersByBudget,
suggestProductsForDish,
} from "../services/ai/aiPlannerService.js";
import { buildPlainReply } from "../services/ai/aiResponseService.js";
import { canUseOpenAI, generateOpenAIReply } from "../services/ai/openAiChatService.js";
export const queryAssistant = async (req, res) => {
try {
const {
message = "",
action,
mode = "chat",
locale = "uk",
includeProductIds = [],
dinnersCount = 3,
budget = 500,
dish = "",
} = req.body || {};
if (!String(message).trim() && !String(dish).trim() && !action) {
return res.json({ success: false, message: "message is required" });
}
const catalog = await loadCatalog({ includeProductIds });
const resolvedAction = decideAction(action, message || dish);
let payload;
if (resolvedAction === "recipe_from_products") {
payload = buildRecipeFromCatalog(catalog, message || dish, locale);
} else if (resolvedAction === "products_for_dish") {
payload = suggestProductsForDish(catalog, dish || message, locale);
} else if (resolvedAction === "dinners_budget") {
payload = planDinnersByBudget(catalog, dinnersCount, budget);
} else {
```

```
const found = rankProductsBySemanticQuery(catalog, message || dish, 8).map((product) =>
productToChatItem(product, 1),
);
payload = {
action: "semantic_search",
products: found,
};
}
const fallbackReply = buildPlainReply(payload, locale);
let reply = fallbackReply;
if (mode === "chat" && canUseOpenAI()) {
try {
const aiReply = await generateOpenAIReply({
locale,
message: message || dish,
structuredPayload: payload,
});
if (aiReply) {
reply = aiReply;
}
} catch (error) {
console.log(error.message);
}
}
return res.json({
success: true,
action: payload.action,
mode,
reply,
data: payload,
});
} catch (error) {
console.log(error.message);
return res.json({ success: false, message: error.message });
}
};
```

Безпосереднє підключення до OpenAI API реалізоване в сервісі openAiChatService.js:

```
const OPENAI_URL = "https://api.openai.com/v1/responses";
export const canUseOpenAI = () => Boolean(process.env.OPENAI_API_KEY);
```

```
export const generateOpenAIReply = async ({ locale = "uk", message = "", structuredPayload }) => {
const apiKey = process.env.OPENAI_API_KEY;
if (!apiKey) {
return null;
}
const model = process.env.OPENAI_MODEL || "gpt-4.1-mini";
const languageRule = locale === "uk" ? "Відповідай українською." : "Reply in English.";
const prompt = [
"You are GreenCart AI assistant.",
languageRule,
"Use only facts from STRUCTURED_DATA, do not invent products or prices.",
"Be concise and practical.",
`USER_MESSAGE: ${message}`,
`STRUCTURED_DATA: ${JSON.stringify(structuredPayload)}`,
].join("\\n");
const response = await fetch(OPENAI_URL, {
method: "POST",
headers: {
Authorization: `Bearer ${apiKey}`,
"Content-Type": "application/json",
},
body: JSON.stringify({
model,
input: prompt,
temperature: 0.2,
}),
});
if (!response.ok) {
const text = await response.text();
throw new Error(`OpenAI error: ${response.status} ${text}`);
}
const data = await response.json();
return data.output_text?.trim() || null;
};
```

Маршрутизація AI-модуля підключається у головному серверному файлі server.js через імпорт роутера та реєстрацію маршруту app.use('/api/ai', aiRouter), що інтегрує AI-функціональність у загальну архітектуру REST API.