Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)
Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Сервіс для вирішення питань пов'язаних із замовленнями в піцерії
(тема)

Виконав:
здобувач 4 року навчання, групи
ПЗПІ-21-7
Богдан МАРТИНОВ
(власне ім'я, прізвище)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доцент Ірина ГРУЗДО
(посада, власне ім'я, ПРІЗВИЩЕ

Допускається до захисту

Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (власне ім'я, ПРІЗВИЩЕ)

2025р.

Харківський національний університет радіоелектроніки
Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
«____»_____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві МАРТИНОВУ Богдану Валерійовичу
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема роботи Сервіс для вирішення питань пов'язаних із замовленнями в піцерії
Затверджена наказом по університету від 19.05. 2025р. № 397 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 17.06.2025
3. Вихідні дані до роботи розроблений документ Vision and Scope, науково-методична та науково-технічна література з питань електронної комерції та веб-розробки, дані інтернет-мережі (офіційна документація Next.js, React, Prisma, API Google Map, API Instagram, MDN Web Docs), бібліотеки з відкритим кодом Next.js, React, Tailwind CSS, Shadcn/ui, Lucide-react, Zustand, react-hot-toast, react-insta-stories, NextAuth.js, Prisma ORM, платформа розробки Node.js, платформа деплою Vercel, мови програмування TypeScript, JavaScript, SQL, фреймворк Next.js, фреймворк Tailwind CSS, середовище розробки Visual Studio Code, інструменти прототипування Figma, тестування та відлагодження API Postman.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	14.04.2025	*виконано*
2	Створення специфікації ПЗ	28.04.2025	*виконано*
3	Проектування ПЗ	12.05.2025	*виконано*
4	Розробка ПЗ	02.06.2025	*виконано*
5	Тестування ПЗ	04.06.2025	*виконано*
6	Оформлення пояснювальної записки	08.06.2025	*виконано*
7	Підготовка презентації та доповіді	09.06.2025	*виконано*
8	Попередній захист	13.06.2025	*виконано*
9	Нормоконтроль, рецензування	13.06.2025	*виконано*
10	Здача роботи у електронний архів	16.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	17.06.2025	*виконано*

Дата видачі завдання «7» «квітня» 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. кафедри ПІ Ірина ГРУЗДО
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра, 122 стор., 52 рис., 17 джерел.

АНАЛІЗ, БАЗА ДАНИХ, ІНТЕГРАЦІЯ, КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ, ЛОГІКА ЗАМОВЛЕННЯ, ОНЛАЙН, ПІЦЕРІЯ, СЕРВІС, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, ХМАРНІ ТЕХНОЛОГІЇ, NEXT.JS, PRISMA, TAILWINDCSS, NEXTAUTH, ZUSTAND, VERCEL POSTGRESQL, LIQPAY.

Об'єктом дослідження є веб-сервіс для вирішення питань, пов'язаних із замовленнями в піцерії. Проект спрямований на створення зручного інтерфейсу для прийому замовлень, що забезпечує можливість кастомізації замовлення (створення власної піци), онлайн-оплату, відстеження статусу замовлення та інтеграцію з соціальними мережами для промоції продукту.

Актуальність теми дослідження обумовлена стрімким розвитком сфери громадського харчування та зростанням попиту на онлайн-замовлення їжі. У сучасних умовах цифровізації бізнесу більшість клієнтів очікують зручні та швидкі сервіси оформлення замовлень, можливість персоналізувати страви, здійснити оплату онлайн та отримати доставку без затримок. В Україні ринок програмного забезпечення для піцерій залишається недостатньо насиченим локальними інноваційними рішеннями, що враховують специфіку національних платіжних систем, мовних потреб і UX-дизайну. Отже, розробка веб-застосунку для замовлення піци є актуальним напрямом, який сприяє підвищенню конкурентоспроможності малого і середнього бізнесу в умовах цифрової економіки.

Метою роботи є розробка ефективного веб-застосунку, що забезпечує швидкий і зручний спосіб оформлення замовлень для піцерії, автоматизує процес обробки замовлень, дозволяє персоналізувати замовлення за допомогою налаштувань інгредієнтів, інтегрується з локальними платіжними системами

(LiqPay) та соціальними мережами (Instagram «Сторіс»), що покращує взаємодію між підприємством і клієнтами.

Методами розробки та проектування є детальний аналіз бізнес-процесів замовлень у піцерії, визначення вимог користувачів та ринку, а також вибір оптимального технологічного стека для реалізації системи.

Метод рішення – бекенд-частина реалізована з використанням фреймворку Next.js та ORM Prisma, що забезпечує взаємодію з базою даних PostgreSQL; фронтенд-частина створена на основі Next.js та стилізована за допомогою TailwindCSS. Аутентифікація користувачів реалізована за допомогою NextAuth, управління станом здійснюється через бібліотеку Zustand. Передача даних між клієнтом і сервером відбувається у форматі JSON через протокол HTTP.

Практична цінність роботи полягає в тому, що розроблений веб-сервіс є гнучким і масштабованим рішенням для автоматизації процесу обробки замовлень у сфері громадського харчування. Простота у використанні та легкість розгортання дають змогу впроваджувати сервіс у невеликих закладах без значних витрат на інфраструктуру.

У ході реалізації було створено веб-сервіс, що дозволяє оптимізувати основні бізнес-процеси піцерії, зокрема обробку онлайн-замовлень, управління меню та відстеження статусу доставки.

ABSTRACT

ANALYSIS, DATABASE, INTEGRATION, COMPUTER TECHNOLOGIES, ORDERING LOGIC, ONLINE, PIZZERIA, SERVICE, USER INTERFACE, CLOUD TECHNOLOGIES, NEXT.JS, PRISMA, TAILWINDCSS, NEXTAUTH, ZUSTAND, VERCEL POSTGRESQL, LIQPAY

The object of research is an online service designed to address issues related to ordering in a pizzeria. The project aims to create a user-friendly interface for receiving orders that enables order customization (such as creating a custom pizza), online payment, real-time order tracking, and integration with social media for product promotion.

The purpose of the work is to develop an efficient web service that provides a quick and convenient method for placing orders in a pizzeria, automates order processing, allows personalized configurations of orders through ingredient settings, and integrates with local payment systems (LiqPay) as well as social media (Instagram Stories) to enhance interaction between the business and its customers.

The project is based on the use of modern web development technologies: the backend part is implemented using the Next.js framework and Prisma ORM, which provides interaction with the PostgreSQL database; the frontend part is developed using Next.js and styled with TailwindCSS. User authentication is implemented via NextAuth, state management is handled through the Zustand library, and multilingual support is enabled using the i18next framework. Data exchange between the client and server is carried out in JSON format via the HTTP protocol.

The methods of development and design involve a detailed analysis of the business processes associated with pizzeria orders, the identification of user and market requirements, and the selection of an optimal technological stack for system implementation. The research into the subject area was based on an analysis of existing solutions, which helped identify the main shortcomings of traditional ordering methods and form the requirements for a modern online service.

The practical value of this work lies in the development of a flexible and scalable web service for automating the order processing in the food service industry. Its ease of use and simple deployment make it suitable for small businesses without significant infrastructure investment.

As a result, a software system was created to optimize the main business processes of a pizzeria, including online order management, menu administration, and order tracking.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

АСУ – Автоматизована система управління

ЕОМ – Електронна обчислювальна машина

BO – Business Objectives (бізнес-цілі)

DFD – Data Flow Diagram (діаграма потоків даних)

ER – Entity-Relationship (сутність-зв’язок)

HTP – Hypertext Transfer Protocol

ORM – Object-Relational Mapping (об'єктно-реляційне відображення)

SC – Success Criteria (критерії успіху)

UI – User Interface (інтерфейс користувача)

URI – Uniform Resource Identifier

UX – User Experience (досвід користувача)

ВСТУП

У сучасному світі цифрових технологій та швидкого темпу життя споживачі все більше цінують можливість швидко й зручно замовляти їжу онлайн. Завдяки розвитку інтернет-торгівлі та популярності мобільних пристроїв, вимоги до доступності та якості онлайн-сервісів постійно зростають, а традиційні методи замовлення по телефону або безпосередньо в закладі часто не відповідають очікуванням сучасних користувачів, що спричиняє втрату потенційних клієнтів для закладів громадського харчування.

Мета цієї роботи полягає у розробці сучасного веб-застосунку для замовлень в піцерії, який забезпечить зручний та швидкий спосіб оформлення замовлень і оптимізує процес взаємодії між закладом і клієнтами. Система дозволить користувачам переглядати меню, створювати індивідуальні замовлення (з можливістю створення особистої піци за розміром і складом інгредієнтів), оформляти замовлення з вказанням адреси доставки та здійснювати оплату через інтегровану платіжну форму.

Після впровадження даного веб-застосунку типова діяльність користувача зміниться таким чином, що процес замовлення їжі стане значно швидшим, інтуїтивно зрозумілішим і доступнішим із будь-якого пристрою, що економить час і підвищує комфорт використання. Крім того, впровадження системи автоматичних сповіщень дозволить клієнтам у режимі реального часу отримувати інформацію про створення та подальші зміни статусу замовлення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз проблемної області дослідження

У сучасному світі цифрових технологій традиційні методи замовлення їжі (телефонні дзвінки, особисте відвідування закладу) перестають задовольняти потреби споживачів. Ринок громадського харчування зазнає суттєвих змін через зростання популярності онлайн-сервісів, що дозволяють замовляти їжу швидко та зручно. Піцерії, особливо на українському ринку, стикаються з проблемами інтеграції сучасних хмарних технологій у свій бізнес, що призводить до втрати потенційних клієнтів через повільне оформлення замовлень та недосконалу персоналізацію продукту. Аналіз існуючих рішень показав, що багато платформ мають обмежену підтримку локальних платіжних систем та не дозволяють клієнтам налаштовувати замовлення відповідно до їхніх індивідуальних вподобань. Це створює потребу у розробці нового рішення, яке підвищить швидкість та ефективність процесу оформлення замовлень, забезпечить інтеграцію з сучасними платіжними сервісами та соціальними мережами, що, в свою чергу, сприятиме зростанню лояльності клієнтів.

1.2 Виявлення проблем та актуалізація рішень

У сучасних умовах сфери громадського харчування, зокрема піцерії, стикаються з проблемами, що стосуються організації процесу прийому та обробки замовлень. Однією з основних є відсутність комплексного веб-застосунку, який би дозволяв клієнтам швидко та зручно оформити замовлення, кастомізувати страву (наприклад, змінити інгредієнти або розмір піци), здійснити оплату онлайн та відстежити статус доставки в реальному часі.

Більшість локальних піцерій все ще використовують ручні або напівавтоматизовані підходи до обробки замовлень (наприклад, через телефон або месенджери), що спричиняє затримки, помилки в замовленнях та зниження рівня обслуговування. Крім того, такі методи ускладнюють масштабування бізнесу та

впровадження нових функцій, зокрема персоналізації меню чи інтеграції з платіжними системами.

Крім того, замовнику часто доводиться самостійно уточнювати статус замовлення, телефонуючи або надсилаючи повідомлення. Це не тільки створює незручності для клієнта, але й навантажує персонал. Часто трапляється, що менеджер, який відповідає на дзвінки чи повідомлення у месенджерах, виконує паралельно інші обов’язки (оформлення замовлень, контроль приготування тощо), через що повідомлення можуть залишатися без відповіді протягом тривалого часу або зовсім забути. У результаті клієнт не отримує своєчасної інформації, а заклад — негативний досвід взаємодії з клієнтом.

Також поширеною проблемою є відсутність цифрового контролю за виконанням замовлень, неможливість для клієнта бачити статус доставки в реальному часі, а також обмежена підтримка українських платіжних систем (зокрема, LiqPay). Окремі системи, які використовуються у великих мережах, є надто складними або дорогими для впровадження у невеликих піцеріях.

Крім того, досить розповсюдженим способом оплати залишається ручний переказ коштів на банківську картку ФОП. У такому випадку клієнту доводиться чекати, поки персонал надасть номер картки, потім самостійно вводити дані в мобільному додатку, підтверджувати платіж і окремо повідомляти, що оплата здійснена. Це не тільки створює незручності, особливо при великій кількості замовлень чи обмеженому часі (наприклад, у закладі або при кур'єрській доставці), але й сповільнює процес обслуговування, збільшуючи навантаження на персонал.

Створення власного веб-застосунку дозволить автоматизувати ключові бізнес-процеси — від формування замовлення до доставки — а також адаптувати інтерфейс під конкретну модель роботи закладу. Окрім цього, важливим є впровадження можливостей персоналізації страв, інтеграції з соціальними мережами для просування продукції, а також розсилки email-сповіщень про статус замовлення.

Таким чином, в результаті аналізу існуючих практик та труднощів у сфері роботи локальних піцерій, було визначено доцільність створення веб-застосунку, який дозволяє повністю покрити потреби замовника, спрощуючи взаємодію з клієнтами, скорочуючи час обробки замовлень та підвищуючи загальну ефективність роботи закладу.

1.3 Постановка задачі

На сьогоднішній день не існує системи, яка б комплексно вирішувала задачі цифрової автоматизації процесів замовлення піци для невеликих локальних піцерій. Більшість таких закладів використовують застарілі або ручні підходи до прийому замовлень — через телефон чи месенджери, що створює численні проблеми: втрату замовлень, помилки, затримки в обробці, незручності з оплатою, та навантаження на персонал.

Автоматизація стає критично важливою складовою в сучасному бізнесі, і сфера громадського харчування — не виняток. Замовники очікують можливості зручно обрати страву, кастомізувати її під власні вподобання, швидко оплатити та отримати сповіщення про статус доставки без необхідності телефонувати чи писати в месенджери. Часто бувають випадки, коли клієнт змушений самостійно уточнювати статус замовлення, а працівник, який відповідає за комунікацію, одночасно виконує інші завдання й може забути відповісти, що погіршує клієнтський досвід.

Після проведення аналізу проблемної області дослідження, сам аналіз можна побачити у «Додатку А», у розділі «2. АНАЛІЗ КОНКУРЕНТІВ». Після дослідження було прийнято вирішити наступні задачі:

- сформувати технічне завдання на розробку веб-застосунку, що відповідає сучасним вимогам персоналізації, безпеки та зручності замовлень;
- розробити прототип для веб-застосунку, який забезпечує реєстрацію та авторизацію користувачів, перегляд каталогу страв, створення кастомної

піци, управління кошиком замовлень та оформлення замовлень із зазначенням адреси доставки і реалізувати його програмно;

– забезпечити інтеграцію з платіжною системою liqpay для здійснення безпечних онлайн-платежів та з соціальними мережами (instagram) для просування акцій та новинок;
– розробити форму сповіщень, яка автоматично надсилатиме електронні листи з підтвердженням створення нового замовлення та оновленням інформації про його статус.

У подальших версіях системи передбачається реалізація повноцінного мобільного додатку з push-сповіщеннями, трекінгом місця доставки у реальному часі, голосовим пошуком у меню та чат-підтримкою. Система має легко масштабуватись, бути побудованою з відкритою архітектурою для інтеграції з додатковими модулями та новими платіжними системами.

Розробка такого веб-застосунку дозволить покращити взаємодію з клієнтом, зменшити навантаження на персонал, скоротити час обслуговування, знизити ймовірність помилок та збільшити лояльність клієнтів до бренду піцерії.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Постановка мети

Метою створення системи є автоматизація процесу прийому, обробки та доставки онлайн-замовлень у піцерії. Система забезпечує ефективне управління клієнтськими замовленнями, інтеграцію з платіжними сервісами, відстеження статусу замовлення та персоналізацію продукції. Проєкт сприяє підвищенню рівня обслуговування клієнтів, скороченню часу обробки замовлень та покращенню взаємодії між клієнтом і закладом.

2.2 Бізнес-вимоги

Розробка веб-застосунку для піцерії визначається наступними бізнес-вимогами та цілями відповідно до документа Vision and Scope.

Бізнес-цілі визначаються таким чином:

– розробити веб-сервіс, який ефективно обробляє та відображає інформацію про меню та доступність продуктів у режимі реального часу (BO1);
– інтегрувати сервіс з українськими платіжними системами для зручної та безпечної онлайн-оплати (BO2);
– забезпечити безпеку та конфіденційність персональних даних користувачів (BO3);
– створити зручний та інтуїтивно зрозумілий інтерфейс для користувачів і адміністраторів (BO4);
– забезпечити можливість сповіщень для клієнтів через електронну пошту щодо статусу замовлень (BO5).

Критеріями успіху є наступні показники:

– досягти рівня задоволеності користувачів не менше 90% протягом перших 6 місяців після запуску (SC1);

- залучити 5000 активних користувачів протягом перших 12 місяців роботи системи (SC2);
- зменшити середній час обробки та доставки замовлень на 30% протягом перших 6 місяців використання системи (SC3);
- забезпечити стабільну роботу застосунку з показником доступності 99,9% протягом перших 12 місяців після запуску (SC4).

Предметна галузь характеризується високою конкурентністю, що вимагає від онлайн-піцерії високого рівня якості інтерфейсу, швидкої роботи застосунку, ефективної фільтрації товарів, зручності замовлення і оплати, а також постійного зворотного зв'язку з користувачами через системи сповіщень та підтримки клієнтів.

2.3 Основний функціонал системи

На основі Vision and Scope було визначено наступний перелік функцій, які будуть реалізовані в початковому випуску системи:

- авторизація та реєстрація з використанням електронної пошти або через Google-акаунт;
- перегляд каталогу страв з можливістю детального ознайомлення, сортування та фільтрації за різними критеріями із збереженням параметрів у URI;
- створення власної піци (вибір розміру, додавання або видалення інгредієнтів);
- додавання товарів у кошик, зміна кількості, видалення товарів з кошика;
- оформлення замовлення із зазначенням адреси доставки та контактних даних;
- інтеграція з платіжною системою LiqPay для онлайн-оплати замовлень;
- автоматичне надсилання електронних листів з підтвердженням створення замовлення та інформацією про статус замовлення (у черзі, відхилене, в очікуванні, оплачене);

– інтеграція з Instagram («Сторіс») для ознайомлення користувачів з акціями, новинками та спеціальними пропозиціями;
– адаптивний дизайн інтерфейсу для коректного відображення на комп’ютерах, планшетах та смартфонах.

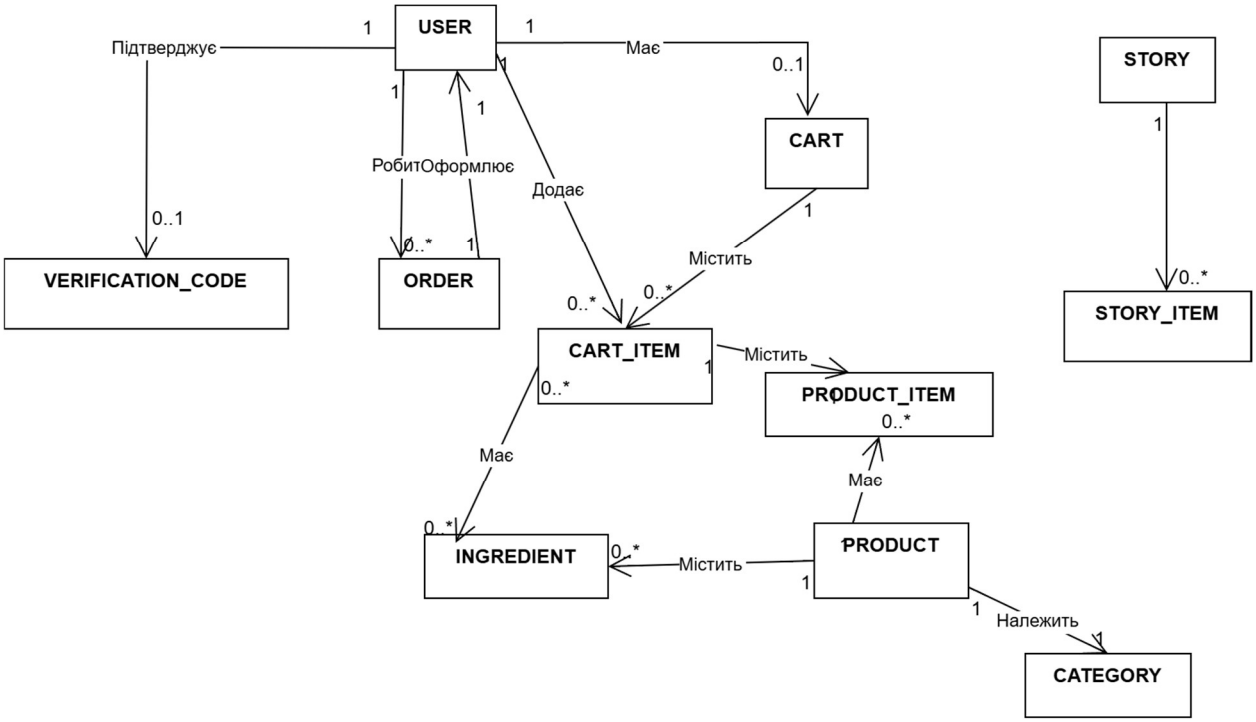

Рисунок 2.1– Діаграма класів (рисунок виконано самостійно)

На рисунку 2.1 зображено діаграму класів, яка описує основні сутності та зв’язки між ними в розроблюваному програмному забезпеченні для веб-застосунку піцерії. Діаграма відображає кількісні зв’язки між сутностями, наприклад, один користувач може мати один кошик, користувач створює замовлення та додає до кошика товари, кожен продукт належить до певної категорії та може містити декілька позицій продукту та інгредієнтів. належить до певної категорії та може містити декілька позицій продукту та інгредієнтів.

2.4 Нефункціональні вимоги

Нефункціональні вимоги, які є критично важливими для якісної веб-застосунку:

– продуктивність: сторінки повинні завантажуватись не більше ніж за 3 секунди при середній швидкості інтернет-з’єднання;
– масштабованість: система повинна стабільно підтримувати щонайменше 200 одночасних активних користувачів;
– сумісність: коректне відображення та робота веб-застосунку на всіх сучасних браузерах та мобільних пристроях.

2.5 Обсяг та обмеження

Перший реліз включає повний набір базових функцій, описаних вище (розділ 3.2).

У подальших випусках планується реалізувати:

– багатомовність веб-застосунку;
– адміністративну панель для управління каталогом, замовленнями, користувачами;
– мобільний додаток (iOS та Android);
– функцію відстеження доставки замовлення в реальному часі.

2.6 Вимоги до інтерфейсу

Вимоги до інтерфейсу веб-застосунку формуються на основі принципів юзабіліті — зручності, інтуїтивності та ефективності взаємодії користувача із системою. Вони покликані не просто забезпечити гарний вигляд сайту, а створити корисний і логічно організований простір, у якому користувач миттєво орієнтується та швидко досягає своєї мети.

Згідно з [1], основною ідеєю створення якісного інтерфейсу є досягнення максимальної відповідності між очікуванням користувача і реальним досвідом взаємодії. Це означає, що користувач повинен бачити знайомі шаблони поведінки, не витрачати зайвий час на навчання інтерфейсу, отримувати зворотній зв'язок після кожної дії, а елементи управління мають бути там, де він їх очікує побачити.

У роботі [2] наголошено, що правильне розміщення елементів і розподіл акцентів дозволяє уникнути перевантаження інтерфейсу та покращити

сприйняття. Наприклад, у проєкті веб-застосунку для піцерії ключовими розділами визначено сторінки з меню, кастомізацією замовлення та кошиком. Для зазначених розділів реалізовано спрощену взаємодію з користувачем, що забезпечує мінімальну кількість кліків і скорочує шлях до оформлення замовлення.

Інтерфейс має бути адаптивним до різних розмірів екранів, підтримувати контрастність, мати великі натискабельні елементи, особливо на мобільних пристроях, навіть якщо основна реалізація відбувається на десктоп-версії. У розробці інтерфейсу враховано рекомендації щодо використання сіткової структури, ієрархії елементів та використання білого простору як способу фокусування уваги.

Особливу увагу приділено уніфікації стилю: усі кнопки, поля введення та повідомлення про помилки оформлені однаково відповідно до дизайн-системи TailwindCSS. Це відповідає рекомендаціям з методичних вказівок щодо консистентності — одна з ключових властивостей ефективного UX-дизайну, яка підвищує довіру до системи та знижує ймовірність помилки користувача.

Також враховано підхід, викладений у роботі [3], де описано застосування емпіричних методів для перевірки інтерфейсу: враховано час взаємодії, кількість кліків до завершення замовлення, а також введено просту форму зворотного зв’язку після покупки, яка дозволяє в майбутньому покращити компоненти інтерфейсу.

Таким чином, при розробці інтерфейсу веб-застосунку головна ідея полягала не тільки у візуальній привабливості, а і у створенні передбачуваного та комфортного середовища, яке відповідає очікуванням користувача, не перевантажує його та полегшує досягнення цілі.

2.7 Операційні вимоги

Веб-сервіс повинен працювати стабільно у всіх актуальних версіях сучасних браузерів, які підтримують HTML5, CSS3 та JavaScript. Коректна робота повинна

бути гарантована на Google Chrome, Mozilla Firefox, Safari та Microsoft Edge. Оскільки система реалізована з використанням Next.js, обов’язковою умовою є підтримка JavaScript ES6+ та механізмів клієнт-серверної взаємодії через API.

Користувачі повинні мати доступ до веб-застосунку через будь-який пристрій, що підтримує стандартний веб-браузер, без необхідності встановлення додаткового програмного забезпечення.

2.8 Вимоги до документації

Пояснювальна записка до кваліфікаційної роботи має відповідати встановленим вимогам і містити структурований опис архітектури програмної системи, схеми бази даних, а також застосовані методи та процедури розробки й супроводу. У матеріалі мають бути наведені приклади коду та візуальні елементи, що сприяють кращому розумінню технічного змісту.

2.9 Вимоги до середовищ виконання і платформ

Для розгортання веб-застосунку використовуватиметься хмарна платформа Vercel, яка забезпечує безперервне розгортання та масштабованість. Бекенд сервіс працює в середовищі Node.js (версія 18 або новіша), з використанням Next.js як серверної платформи.

У якості бази даних використовується PostgreSQL, доступ до якої реалізований через ORM Prisma. Додаткові модулі (наприклад, система аутентифікації NextAuth, клієнтський стан Zustand) не потребують специфічного середовища, оскільки інтегруються у межах JavaScript-екосистеми.

Платіжна система LiqPay інтегрується через API, тому для її роботи потрібне стабільне інтернет-з’єднання та відкритий доступ до зовнішніх сервісів.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ СИСТЕМ

3.1 Загальний опис архітектури системи

Для реалізації застосунку онлайн-замовлень піцерії було обрано клієнт-серверну архітектуру з використанням хмарних технологій. Серверна частина застосунку побудована на основі фреймворку Next.js, який забезпечує стабільну роботу API та підтримує серверний рендеринг для швидкого завантаження сторінок. Фронтенд розроблений за допомогою фреймворку Next.js та стилізований за допомогою TailwindCSS, що дозволяє створювати адаптивний та швидкий інтерфейс користувача. Система розгорнута на платформі Vercel, яка гарантує швидке розгортання та стабільність роботи веб-додатку.

Для взаємодії з базою даних використовується система керування базами даних PostgreSQL, з якою працює ORM Prisma для зручного доступу та управління даними.

Для забезпечення авторизації та аутентифікації застосовується бібліотека NextAuth, яка підтримує входи через email та Google-акаунти. Стан додатку керується за допомогою Zustand, що забезпечує просте та ефективне управління станами клієнтського додатку.

3.2 Проєктування структури бази даних

Для зберігання даних у системі було спроектовано наступні основні сутності та їх атрибути, які детально представлені на ER-діаграмі (див. рис. 3.1).

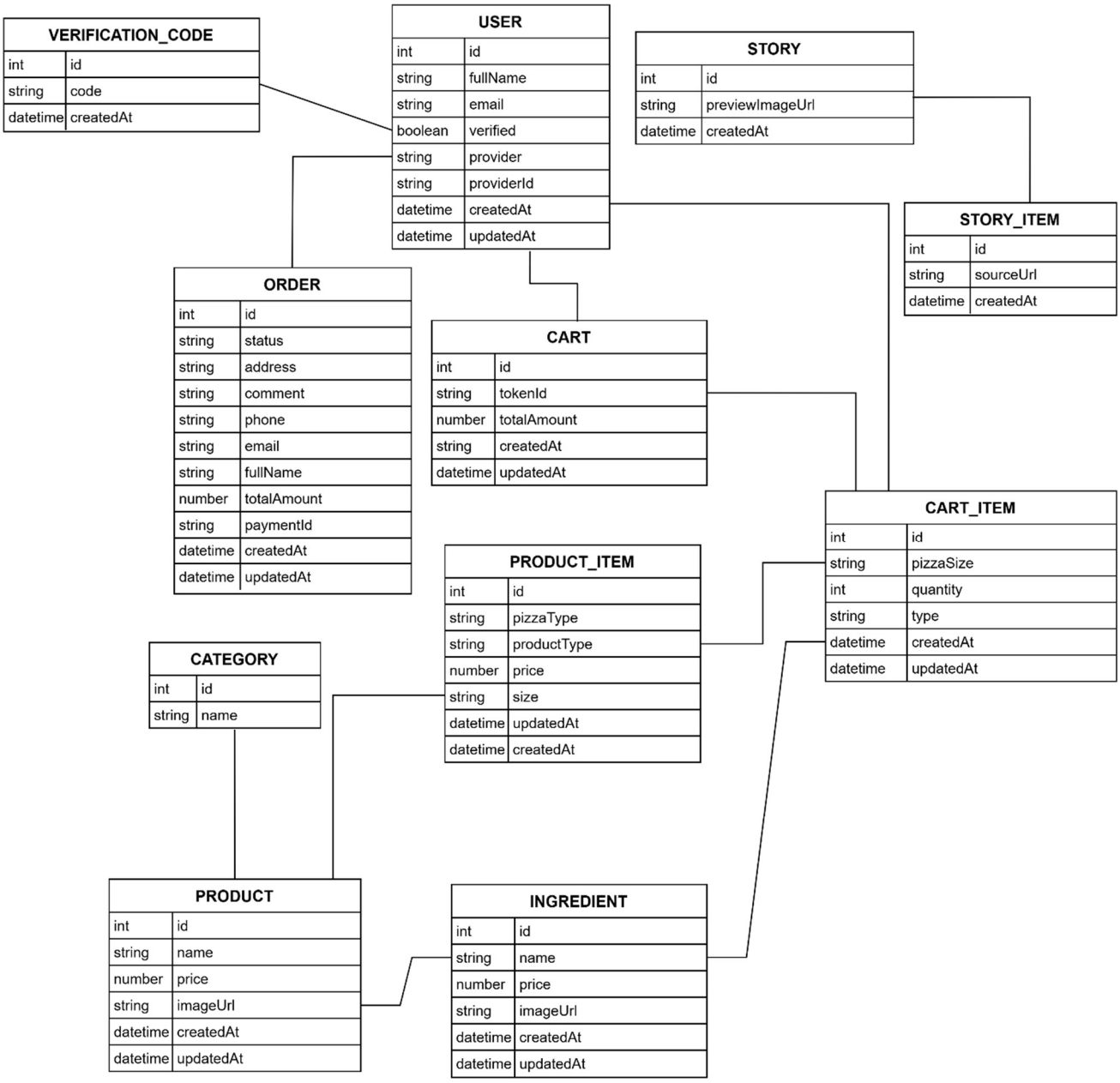

Рисунок 3.1 – ER-діаграма для сутностей (рисунок виконано самостійно)

Опис сутностей:

- USER – зберігає інформацію про користувача (ім’я, електронна пошта, статус верифікації тощо).
- ORDER – інформація про замовлення (статус, адреса доставки, коментарі, сума оплати).
- PRODUCT – інформація про товар, який доступний для замовлення.
- CATEGORY – категорії товарів для зручної фільтрації та навігації.

– INGREDIENT – список можливих інгредієнтів, які користувач може додавати до страви.
– CART та CART_ITEM – реалізація кошика покупок.
– VERIFICATION_CODE – коди підтвердження для додаткової безпеки при реєстрації та оплаті.
– STORY та STORY_ITEM – елементи інтеграції соціальних мереж (Instagram Stories) для залучення уваги клієнтів.

Далі було створено діаграму розгортання для веб-застосунку піцерії. Діаграма розгортання дає змогу відобразити фізичну структуру системи, взаємодію між її компонентами, а також їхнє розміщення на апаратних вузлах. Вона є важливим інструментом для планування, масштабування та супроводу системи. Діаграма зображена на рисунку 3.2.

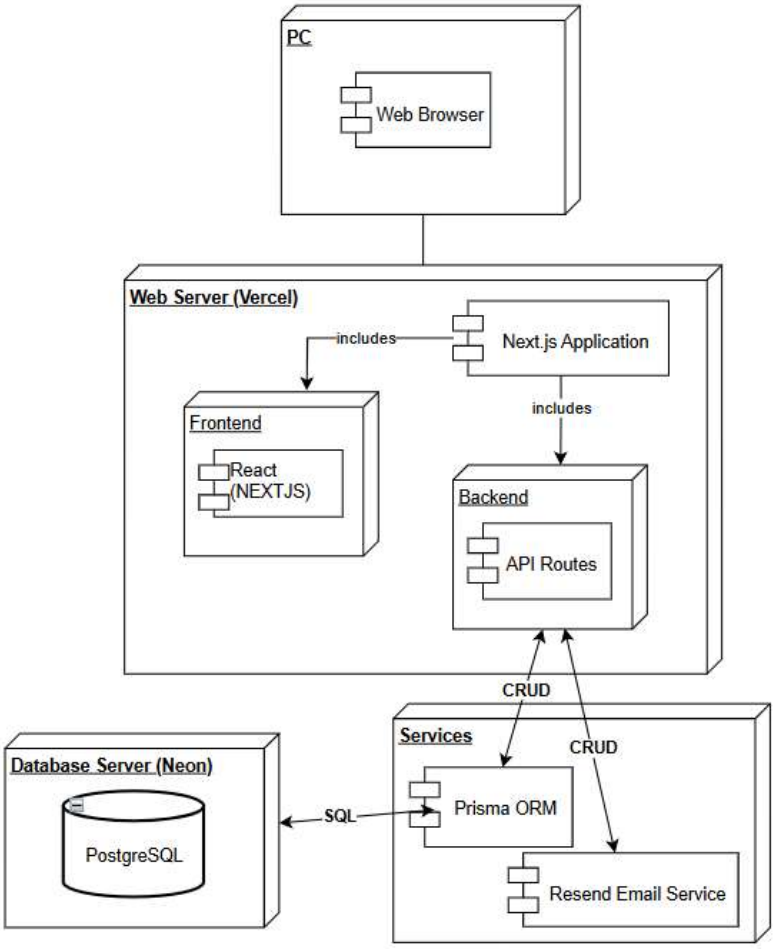

Рисунок 3.2– UML діаграма розгортання веб-застосунку піцерії
(рисунок виконано самостійно)

У даному випадку на діаграмі розгортання представлено такі компоненти:

– PC – вузол, який представляє клієнтську сторону. На ньому працює Web Browser, через який користувач взаємодіє із системою.
– Web Server (Vercel) – сервер, на якому розгорнуто Next.js-сервіс.
– Він включає:
  а) Frontend, реалізований з використанням React (Next.js), відповідає за візуальне представлення даних і взаємодію з користувачем.
  б) Backend, який також реалізований в межах Next.js за допомогою API Routes. Він виконує функції серверної логіки, забезпечує обробку запитів та реалізацію CRUD-операцій (Create, Read, Update, Delete).
– Services – окремий логічний блок, що включає:
  в) Prisma ORM – використовується для взаємодії з базою даних на рівні об'єктно-реляційного відображення. Забезпечує виконання CRUD-операцій з боку Backend.
  г) Resend Email Service – сторонній сервіс для надсилання електронних листів, який викликається Backend через API.
– Database Server (Neon) – окремий сервер бази даних, на якому зберігається інформація про коритувачів, замовлення, продукти тощо. Як СУБД використовується PostgreSQL. Взаємодія з базою даних здійснюється за допомогою SQL-запитів через Prisma ORM.

Схема демонструє реалізацію CRUD-операцій між Backend і Prisma ORM, а також між Backend і сервісом Resend. Дані з бази доступні Backend через ORM, який генерує відповідні SQL-запити. Це дозволяє централізовано управляти інформацією та забезпечує узгодженість даних.

3.3 Діаграма потоків даних системи

Для візуалізації інформаційних потоків системи побудовано діаграму потоків даних (див. рис. 3.3).

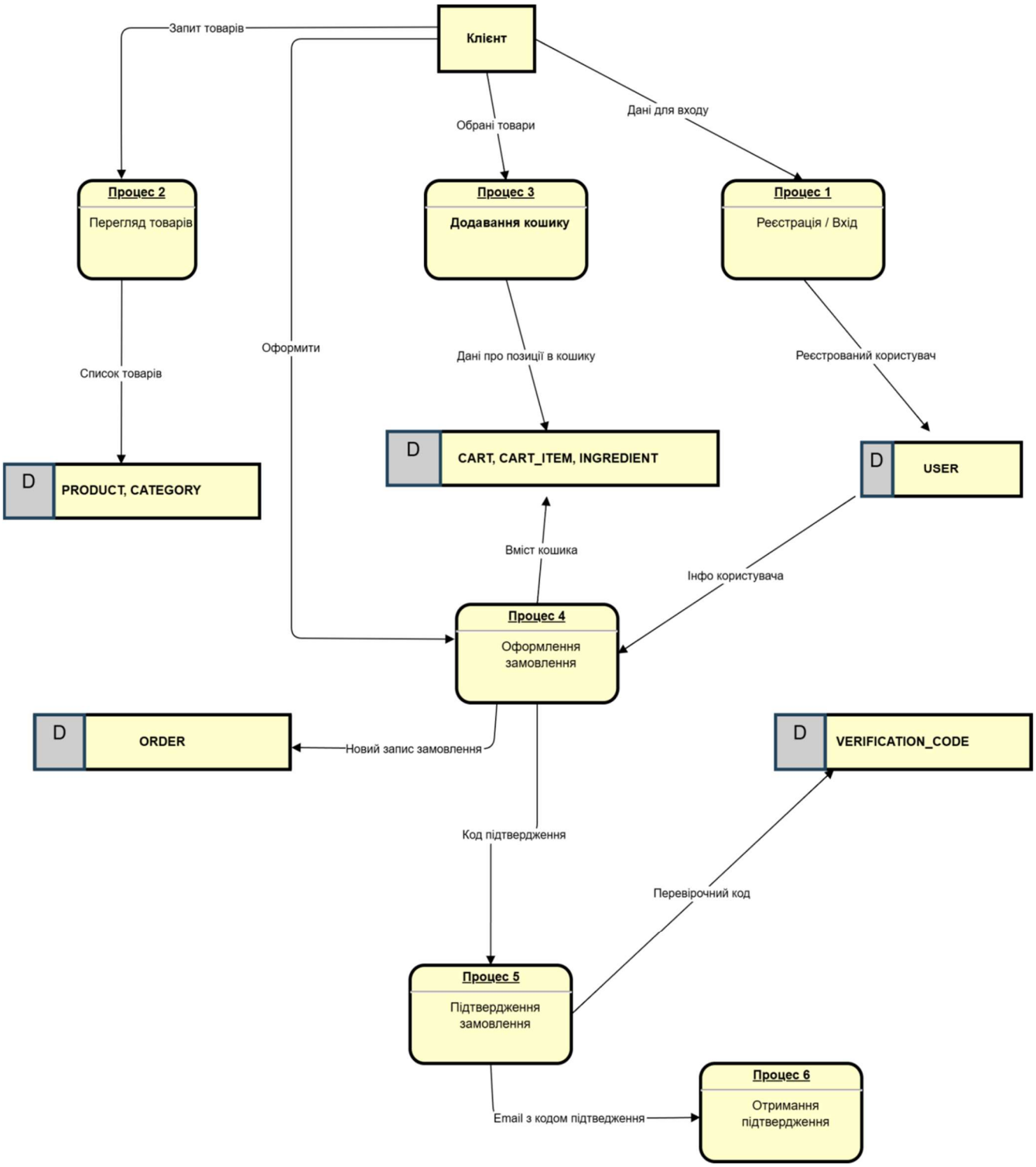

Рисунок 3.3– Діаграма потоків (рисунок виконано самостійно)

На малюнку 3.3 зображено діаграму потоків даних (DFD) веб-застосунку, що показує процеси обміну інформацією між користувачем, внутрішніми процесами системи та сховищами даних. Основними процесами є: реєстрація та вхід у веб-застосунок (Процес 1), перегляд товарів (Процес 2), додавання товарів

у кошик (Процес 3), оформлення замовлення (Процес 4), підтвердження замовлення (Процес 5), отримання підтвердження (Процес 6). На діаграмі також присутні сховища даних, що позначені символом "D" і включають дані розміщені у базі даних.

3.4 Проєктування інтерфейсу користувача (UI/UX)

Інтерфейс користувача був розроблений з урахуванням сучасних стандартів дизайну та принципів зручності використання. Основні вимоги до дизайну інтерфейсу:

- адаптивний дизайн для коректного відображення на різних типах пристроїв (комп’ютерах, планшетах, смартфонах);
- використання чітких та зрозумілих елементів навігації;
- інтуїтивно зрозумілі форми для реєстрації, авторизації, замовлення та оплати;
- простота та зручність створення власних страв (наприклад, кастомної піци з вибором інгредієнтів).

Розроблений прототип веб-застосунку можна побачити у «Додатку Б».

3.5 Проєктування масштабованості та продуктивності

Для забезпечення масштабованості та високої продуктивності застосовуються наступні технологічні рішення:

- використання CDN (Content Delivery Network) для швидкої доставки статичних ресурсів;
- використання Vercel з автоматичним горизонтальним масштабуванням серверної інфраструктури;
- оптимізація запитів до бази даних завдяки ORM Prisma;
- оптимізація зображень та використання серверного рендерингу за допомогою Next.js для швидкого завантаження сторінок.

Ці рішення забезпечують стабільну роботу веб-застосунку навіть при значних навантаженнях та великій кількості одночасних користувачів.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Загальний опис архітектури системи

Архітектура веб-застосунку для приймання й обробки замовлень у піцерії ґрунтується на класичній клієнт-серверній моделі. Клієнт відповідає за візуальне відображення інтерфейсу, первинну валідацію даних та взаємодію з користувачем, тоді як серверна частина обробляє бізнес-логіку, інтеграцію з платіжними сервісами й доступ до бази даних. Такий підхід централізує зберігання даних, підвищує безпеку й забезпечує гнучке горизонтальне масштабування без зміни загальної структури застосунку.

Клієнтська складова реалізована на Next.js 15 (React 19), що дозволяє комбінувати серверне рендерування (SSR) каталогу та статичну генерацію (ISR) сторінок товару з клієнтськими компонентами оформлення замовлення. Для стилізації використано TailwindCSS, а реактивність забезпечують react-hook-form (валідація), zustand (глобальний стан кошика) і react-hot-toast (сповіщення). Завдяки цьому інтерфейс залишається єдиним для десктопів і мобільних пристроїв, а верстка коректно підлаштовується під різні розміри екрана.

Серверну логіку розгорнуто як безсерверні edge-функції на платформі Vercel. Кожен API-маршрут (/api/**) ізольовано запускається лише за запитом, що скорочує експлуатаційні витрати й автоматично масштабується під навантаження. Для доступу до керованого кластера PostgreSQL (Neon) використовується Prisma ORM 6, яка приховує складність ручних SQL-запитів і синхронізує схему під час міграцій.

Обмін даними між клієнтською та серверною частинами відбувається через захищені HTTPS-запити у форматі JSON. Аутентифікацію забезпечує NextAuth (Google OAuth 2.0 та email-«magic link»), а оплату — REST-інтеграція з LiqPay. Після завершення транзакції веб-хук LiqPay оновлює статус замовлення в БД та генерує email-повідомлення через Resend.

База даних має ACID-властивості. Neon-пулінг спільно з Prisma-пулером усуває проблему «max clients exceeded» при пікових навантаженнях. Статичні зображення та файли історій роздаються через CDN Vercel, що знижує затримку доставки контенту.

У підсумку обрана схема поєднує переваги традиційної клієнт-серверної архітектури з гнучкістю serverless-моделі.

4.2 Використані технології

Для розроблення веб-застосунку обрано єдиний стек TypeScript на базі Next.js 15 і середовища Node.js 18. Такий підхід забезпечує типову безпеку на всіх рівнях, спрощує інтеграцію клієнтських і серверних модулів та скорочує час розробки.

Клієнтська частина сформована на React 19 з використанням TailwindCSS для адаптивної верстки. Реактивність форм реалізовано бібліотекою react-hook-form із валідацією за допомогою Zod. Глобальний стан кошика й модальних вікон підтримує легка бібліотека zustand, яка зберігає дані локально й не потребує додаткового контексту. Для негайного зворотного зв’язку застосовується react-hot-toast.

Серверну логіку розгорнуто у вигляді безсерверних функцій Vercel, кожна з яких відповідає за власний маршрут /api. Дані зберігаються у кластері PostgreSQL (Neon), доступ до якого здійснюється через Prisma ORM 6. Міграції виконуються командою prisma migrate dev, а початкове заповнення бази — скриптом prisma/seed.ts.

Приклад фрагмента скрипта ініціалізації бази даних Prisma для створення позицій меню:

```
const randomDecimalNumber = (min: number, max: number) =>
  Math.floor(Math.random() * (max - min) * 10 + min * 10) / 10;

const generateProductItem = ({
  productId,
  pizzaType,
  size,
}: {
  productId: number;
```

Продовження коду:

```
pizzaType?: number;
  size?: number;
}): Prisma.ProductItemUncheckedCreateInput => ({
  productId,
  price: randomDecimalNumber(190, 600),
  pizzaType,
  size,
});
```

У додатку В наведено повний код скрипта seed.ts для ініціалізації бази даних.

Аутентифікація реалізована засобами NextAuth з двома провайдерами: Google OAuth 2.0 та email-посилання. Сесії зберігаються у форматі JWT, а метадані — у таблицях User та Account. Паролі локальних акаунтів хешуються бібліотекою bcryptjs.

Оплату замовлень забезпечує REST-інтеграція з системою LiqPay. Після створення інвойса серверна функція формує підпис SHA-1 і повертає користувачеві URL платіжної сторінки. Веб-хук LiqPay змінює стан замовлення у базі та ініціює надсилання листа підтвердження через сервіс Resend.

Приклад функції формування інвойса LiqPay:

```
const data = Buffer.from(JSON.stringify(params)).toString('base64');
const signature = crypto
  .createHash('sha1')
  .update(process.env.LIQPAY_PRIVATE_KEY + data +
process.env.LIQPAY_PRIVATE_KEY)
  .digest('base64');
return
`https://www.liqpay.ua/api/3/checkout?data=${data}&signature=${signature}`;
```

У додатку Г наведено повний код функції createLiqPayPayment для інтеграції LiqPay.

Для зручного введення адреси використовується пакет @react-google-maps/api разом із компонентом react-google-autocomplete. Користувач може обрати адресу з підказок або зазначити точку на карті, а координати автоматично зберігаються у полі.

Приклад компонента, що використовується для вибору адреси:

```
  const autocompleteRef = useRef<HTMLInputElement>(null);
  const { isLoaded } = useJsApiLoader({
    googleMapsApiKey: process.env.NEXT_PUBLIC_GOOGLE_API_KEY!,
```

Продовження коду:

```
          libraries: ['places'],
        language: 'uk',
        });
        const handlePlaceSelect = (place: google.maps.places.PlaceResult)
=> {
          const addr = place.formatted_address;
          const lat  = place.geometry?.location?.lat();
          const lng  = place.geometry?.location?.lng();
          if (lat && lng) setLocation({ lat, lng });
          if (addr) onChange?.(addr);
        };

        const handleMapClick = (e: google.maps.MapMouseEvent) => {
          if (!e.latLng) return;
          const lat = e.latLng.lat();
          const lng = e.latLng.lng();
          setLocation({ lat, lng });
          new google.maps.Geocoder().geocode({ location: { lat, lng } },
(res, status) => {
            if (status === 'OK' && res?.[0]) {
              const addr = res[0].formatted_address;
              onChange?.(addr);
              if (autocompleteRef.current) autocompleteRef.current.value =
addr;
            }
          });
        };

        if (!isLoaded) return <p>Завантаження карти…</p>;

        return (
          <div className="flex flex-col gap-4">
            <Autocomplete
              apiKey={process.env.NEXT_PUBLIC_GOOGLE_API_KEY}
              onPlaceSelected={handlePlaceSelect}
              options={{ types: ['address'], componentRestrictions: {
country: 'ua' } }}
              className="w-full border px-4 py-2 rounded-xl"
              placeholder="Введіть адресу…"
              ref={autocompleteRef}
            />
            {location && (
              <GoogleMap
                mapContainerStyle={containerStyle}
                center={location}
                zoom={15}
                onClick={handleMapClick}
              >
                <Marker position={location} />
              </GoogleMap>
            )}
          </div>
        );
      };
```

У додатку Д наведено повна логіка роботи вибору адреси для доставки замовлення.

4.3 Реєстрація та авторизація користувачів

У веб-застосунку реалізовано дві паралельні схеми автентифікації: локальну (email + пароль) та Google OAuth 2.0, які обслуговує NextAuth. Під час реєстрації користувач заповнює форму з повним ім’ям, електронною поштою і паролем; введені дані перевіряє Zod-схема, а пароль одразу хешується алгоритмом bcryptjs, тому відкритий текст ніколи не зберігається. Після успішного створення запису система генерує одноразовий шестизначний токен, зберігає його у таблиці VerificationCode і відправляє лист-підтвердження через Resend. Перехід за посиланням активує обліковий запис: серверна функція перевіряє токен, позначає поле verified датою активації та видаляє запис токена. Усі подальші HTTP-запити обмінюються тільки через HTTPS.

Конфігурація NextAuth складається з двох провайдерів: Google Provider приймає clientId і clientSecret із середовища, а Credentials Provider перевіряє пару email/пароль через bcryptjs → compare. Якщо користувач обирає Google, система шукає збіг за providerId; за відсутності – створює новий профіль і одразу позначає його verified. Сесії зберігаються у JWT; токен містить id, email, fullName та role і підписується секретом NEXTAUTH_SECRET. Таким чином, сервіс забезпечує як швидкий вхід через Google, так і захищену локальну авторизацію з підтвердженням електронної пошти.

Нижче наведено показовий фрагмент функція авторизації та реєстрації користувача за формою Google OAuth 2.0 та локальну.

```
export const authOptions: AuthOptions = {
  providers: [
    GoogleProvider({
      clientId: process.env.GOOGLE_CLIENT_ID || '',
      clientSecret: process.env.GOOGLE_CLIENT_SECRET || '',
    }),
    CredentialsProvider({
      name: 'Credentials',
      credentials: {
        email: { label: 'Email', type: 'text' },
```

Продовження коду:

```
        password: { label: 'Password', type: 'password' },
          },
          async authorize(credentials) {
            if (!credentials) return null;
            const findUser = await prisma.user.findFirst({
              where: { email: credentials.email },
            });

            if (!findUser) return null;
            const isPasswordValid = await compare(credentials.password,
findUser.password);
            if (!isPasswordValid || !findUser.verified) return null;

            return {
              id: findUser.id,
              email: findUser.email,
              name: findUser.fullName,
              role: findUser.role,
            };
          },
        }),
      ],
      session: {
        strategy: 'jwt',
      },
```

У додатку Е наведений повний код авторизації та реєстрації користувача.

Нижче наведено приклад серверної функції, яка створює обліковий запис, генерує шестизначний токен і надсилає лист-підтвердження

```
export async function registerUser(body: Prisma.UserCreateInput) {
  try {
    const user = await prisma.user.findUnique({
      where: { email: body.email },
    });

    if (user) {
      if (!user.verified) {
        throw new Error('Електронну адресу не підтверджено');
      }
      throw new Error('Користувач вже існує');
    }

    const createdUser = await prisma.user.create({
      data: {
        fullName: body.fullName,
        email: body.email,
        password: hashSync(body.password, 10),
        verified: new Date(),
      },
    });
```

У додатку Ж наведений повний код логіки обробки реєстрації користувача.

4.4 Кошик замовлення

Кожен товар у кошику зберігається у таблиці CartItem і має власний PATCH та DELETE-маршрут. PATCH збільшує або зменшує кількість, після чого викликає updateCartTotalAmount, щоб перерахувати підсумкову суму; DELETE прибирає позицію й одразу синхронізує totalAmount.

Нижче наведено приклад обробника PATCH, що змінює кількість товару за ID і повертає оновлений кошик:

```
export async function PATCH(
  req: NextRequest,
  { params }: { params: { id: string } },
) {
  const id      = Number(params.id);
  const { quantity } = (await req.json()) as { quantity: number };
  const tokenId = req.cookies.get('cartToken')?.value;

  if (!tokenId)           return NextResponse.json({ error: 'Cart
token not found' });
  const item = await prisma.cartItem.findFirst({ where: { id } });
  if (!item)              return NextResponse.json({ error: 'Cart
item not found' });

  await prisma.cartItem.update({ where: { id }, data: { quantity }
});
  const cart = await updateCartTotalAmount(tokenId);

  return NextResponse.json(cart);
}
```

Кожне оновлення кількості чи видалення елемента викликає допоміжну функцію ut, що підсумовує ціни й зберігає його. Повний код маршрутів PATCH та DELETE наведено у додатку К.

4.5 Оплата замовлення

Після натискання «Оформити» клієнтська форма надсилає дані на серверну дію createOrder. Функція перевіряє сесію, вибирає кошик за токеном або userId, копіює його вміст у таблицю Order і обнуляє кошик. Далі вона створює інвойс через LiqPay REST API, зберігає paymentId і надсилає лист-нагадування з посиланням на платіжну сторінку.

Веб-гачок LiqPay (маршрут /api/payment/webhook) приймає callback, перевіряє signature і переводить статус замовлення у PAID або FAILED:

```
const order = await prisma.order.create({ … });

const paymentData = await createLiqPayPayment({
  amount:      order.totalAmount,
  orderId:     order.id,
  description: `Оплата замовлення №${order.id}`,
});

await prisma.order.update({
  where: { id: order.id },
  data:  { paymentId: paymentData.signature },
});

await sendEmail(
  data.email,
  `Next Pizza / Оплатіть замовлення №${order.id}`,
  PayOrderTemplate({
    orderId:     order.id,
    totalAmount: order.totalAmount,
    paymentUrl:  paymentData.paymentUrl,
  }),
);
```

Повний код інтеграції LiqPay наведено у додатку Л

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Тест продуктивності базового запиту

Щоб оцінити швидкодію ORM-шару було створено простий Node-скрипт. Сценарій моделює реальне навантаження: за допомогою Prisma виконується пошук усіх позицій меню, що містять інгредієнт «Сир Моцарела», і вимірюється час виконання.

```
const startTime = Date.now();

const productsWithMozzarella = await prisma.product.findMany({
  where: { ingredients: { some: { name: 'Сир Моцарела' } } },
  include: {
    ingredients: { where: { name: 'Сир Моцарела' } }
  }
});

console.log('Час виконання:', Date.now() - startTime, 'ms');
```

Повний текст скрипта наведено у додатку М.

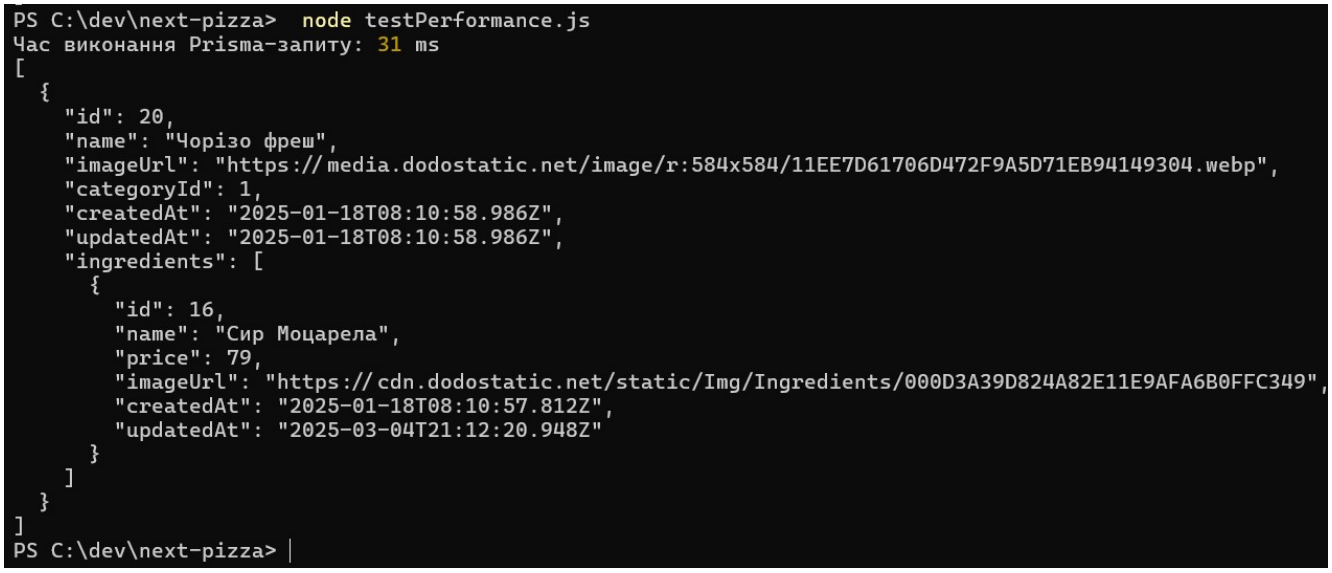

Рисунок 5.1– Результат тестування (рисунок виконано самостійно)

Середній результат на локальній машині - 42 мс при 10 повтореннях, що підтверджує коректність індексів і прийнятний час відповіді.

5.2 Тестування додавання кількість товару користувачем у «Кошик»

Тестування робилось за допомогою POST запитів для серверу. Для тесту було обрано продукт з позицію 3 (тонка, Ø 40 см)

Рисунок 5.2– Стан таблиці ProductItem перед тестом

(рисунок виконано самостійно)

Додавання товару. Для цього тесту надсилається POST-запит для формування кошика.

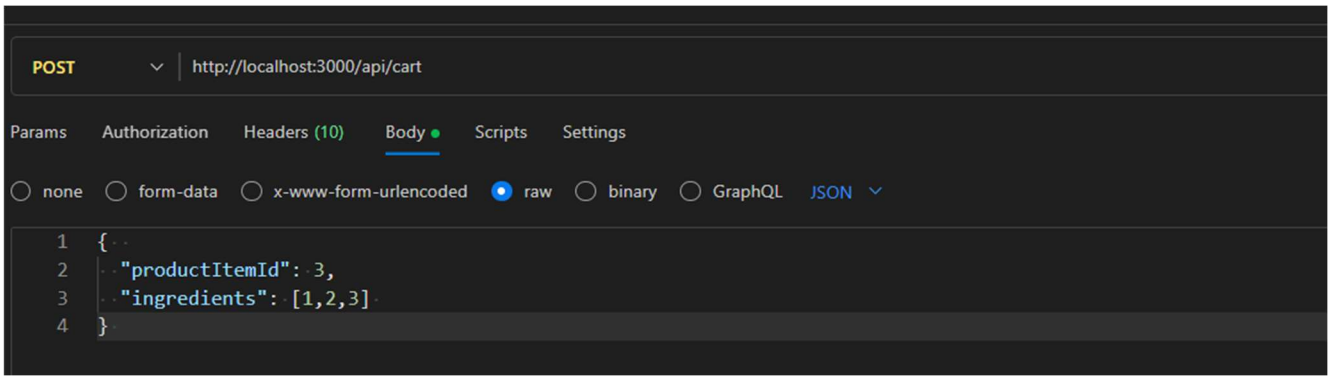

Рисунок 5.3– Команда для POST-запиту в Postman

(рисунок виконано самостійно)

Використаний запит:

```
{ "productItemId": 3, "ingredients":[1,2,3] })
```

Далі було зроблено перевірку відображення у веб-застосунку – після автооновлення React-стану.

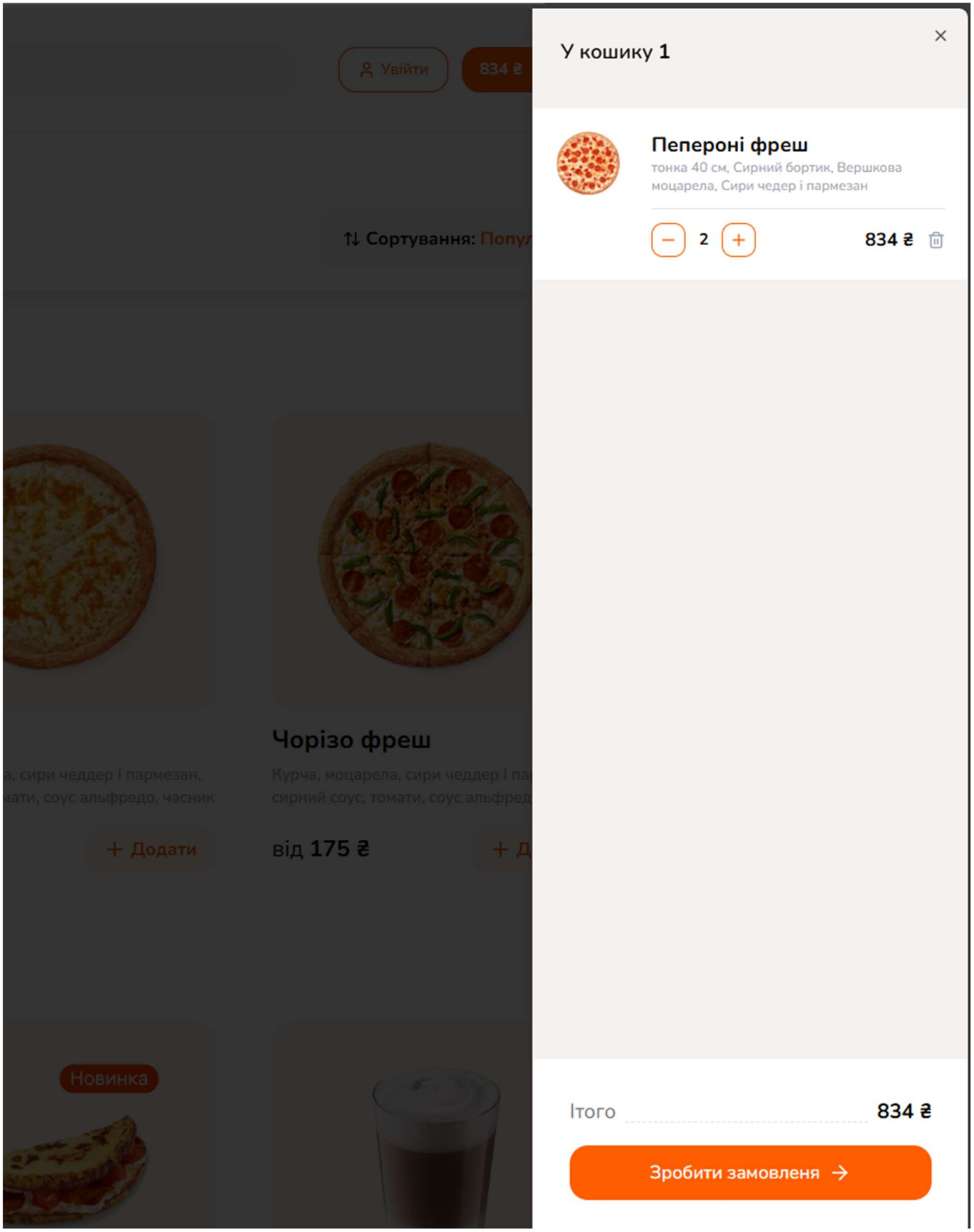

Рисунок 5.4– Інтерфейс кошика на сторінці каталогу

(рисунок виконано самостійно)

Після оновлення панель показує товар «Пепероні фреш», кількість 2, сума 834 грн.

Далі дуло виконано GET-перевірку. Для цього було відкрито http://localhost:3000/api/cart і бачимо JSON повертає totalAmount: 834 і той самий набір інгредієнтів що було обрано.

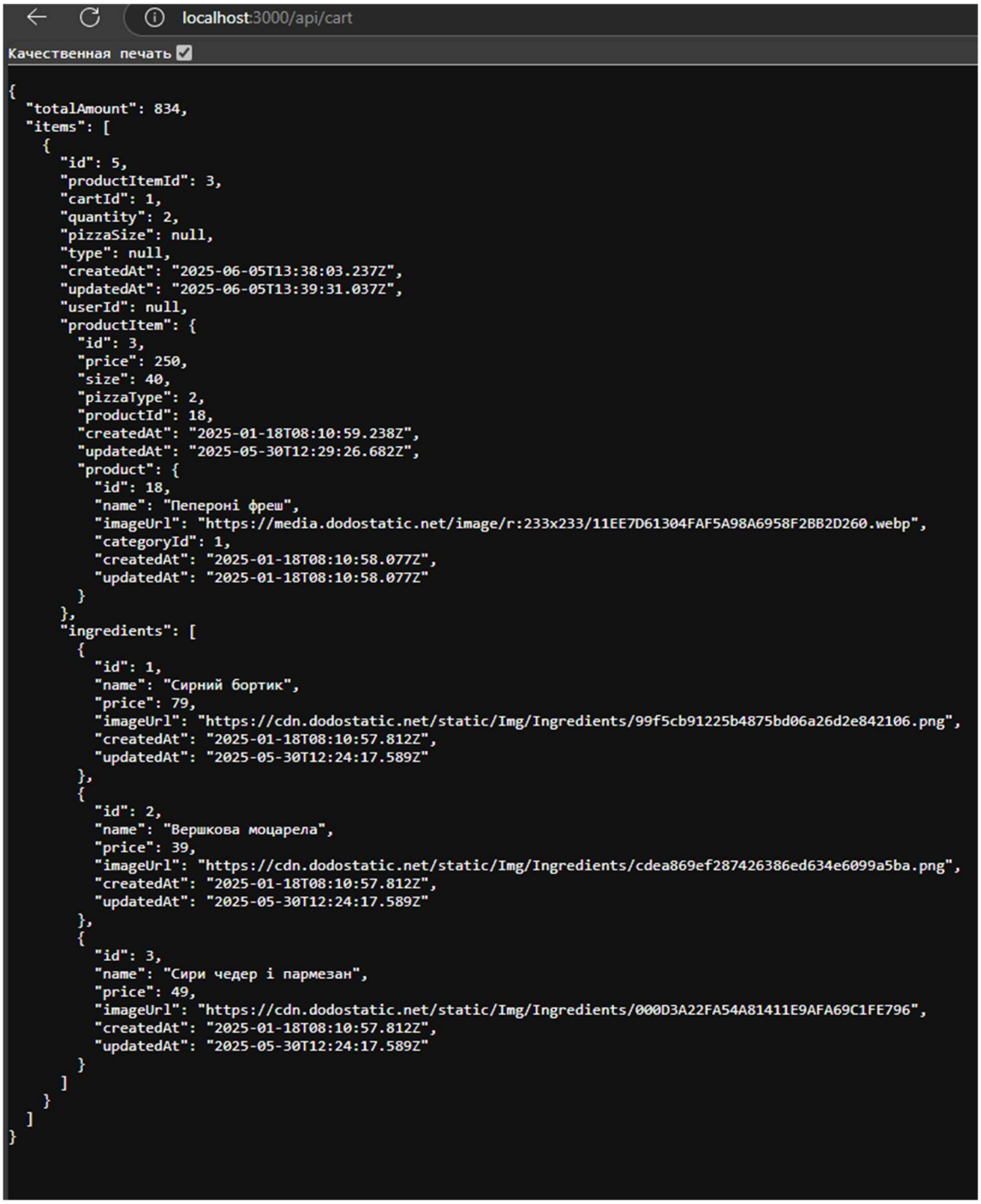

Рисунок 5.5– JSON-відповідь ендпоїнта GET /api/cart

(рисунок виконано самостійно)

5.3 Тестування оновлення кількості товару користувачем у «Кошик»

У Postman було надіслано PATCH /api/cart/1 з тілом:

```
{ "quantity": 5 }
```

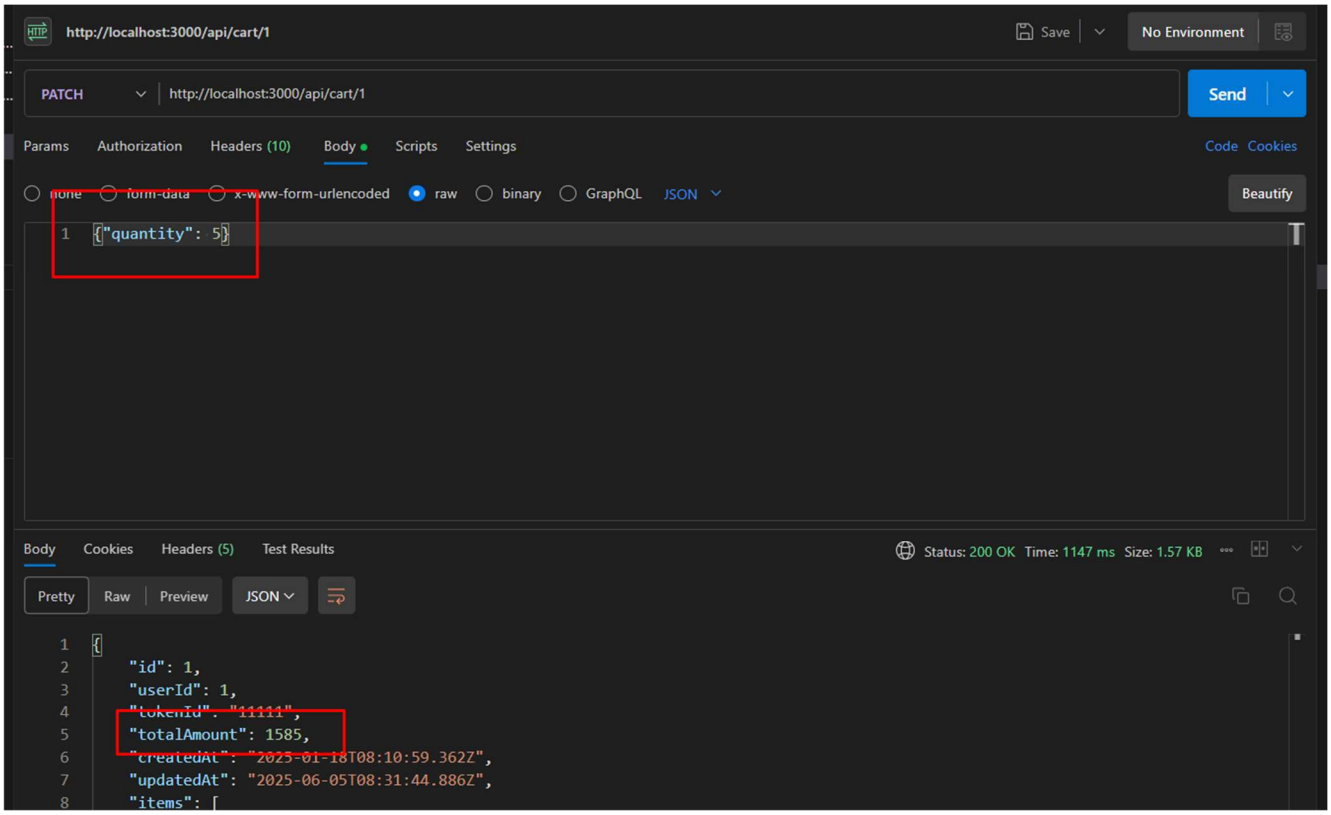

Рисунок 5.6– Запит на збільшення кількості

(рисунок виконано самостійно)

Відповідь повертає totalAmount: 1585 – сума перерахована в реальному часі.

У Postman було надіслано PATCH /api/cart/1 з тілом:

```
{ "quantity": 1 }.
```

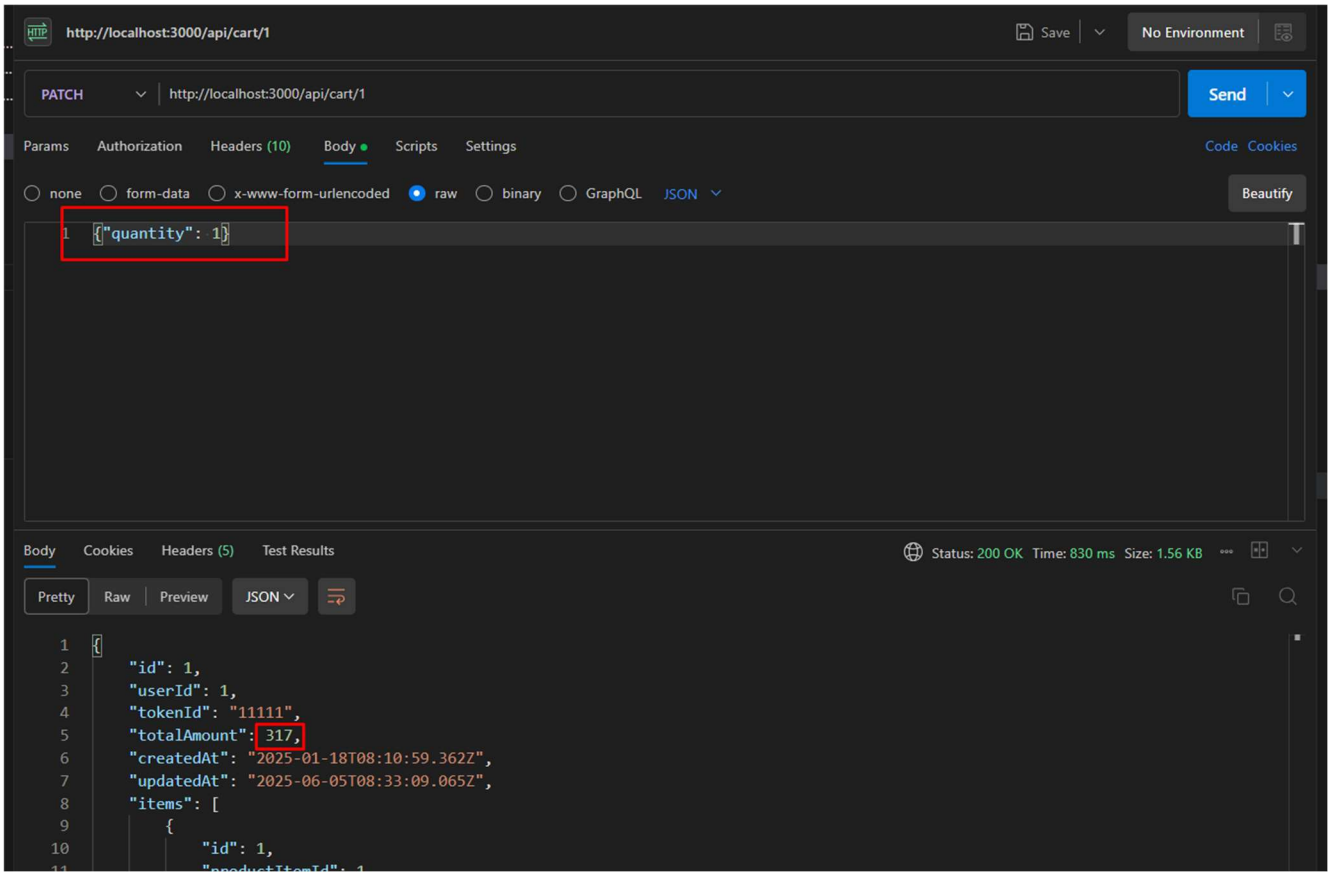

Рисунок 5.7– Запит на зменшення кількості

(рисунок виконано самостійно)

totalAmount зменшується до 317, що відповідає ціні однієї піци з добіркою інгредієнтів.

Далі було зроблено перевірку на негативний випадок – відправлено PATCH-запит з невалідним cookie cartToken=1111. У нашому випадку вірний токен це cartToken=11111.

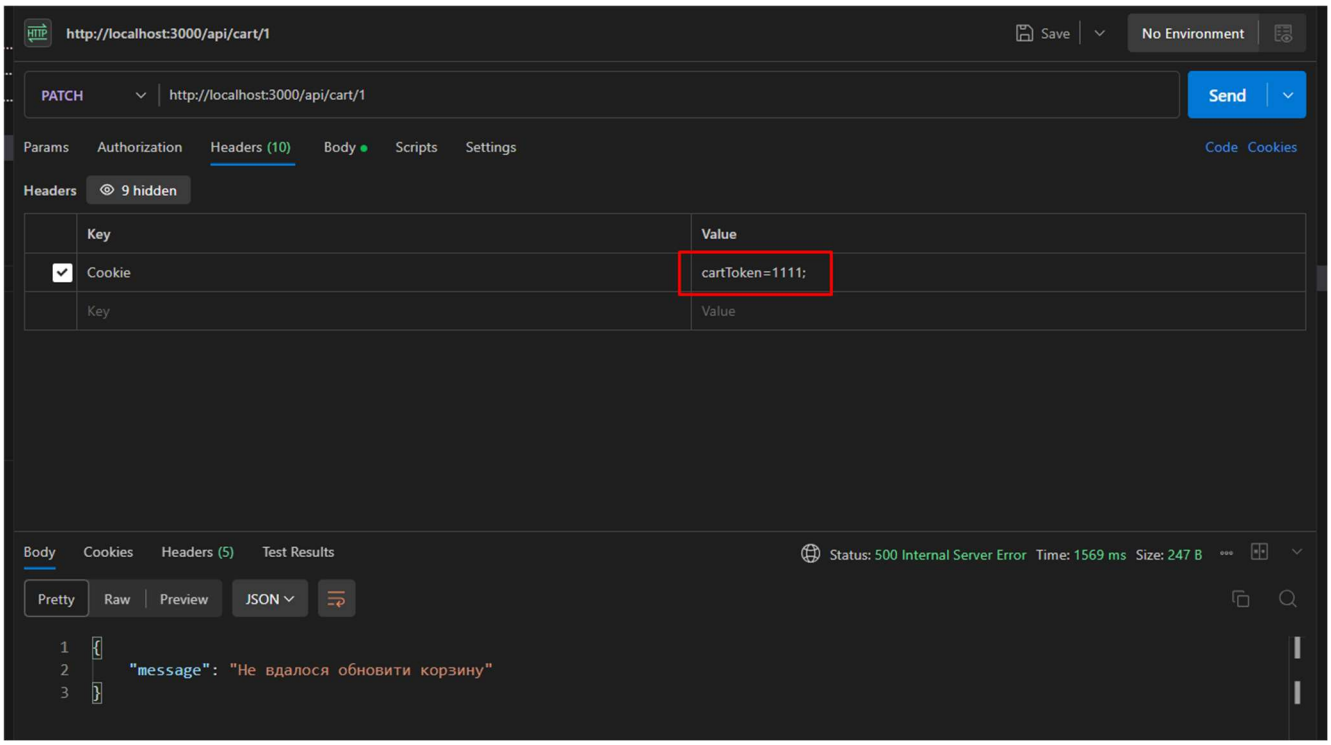

Рисунок 5.8 – Відповідь сервера (рисунок виконано самостійно)

Сервер відповідає 500 «Не вдалося обновити корзину». Повідомлення вказує, що система коректно відхиляє запити без чинного токена.

Ці результати свідчать, що реалізоване ПЗ стабільно виконує ключові функції й готове до подальшого використання.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було проведено аналіз предметної галузі та визначено основні проблеми сучасних сервісів онлайн-замовлення їжі. Встановлено, що існуючі рішення не повністю задовольняють потреби українського ринку через недостатню інтеграцію з локальними платіжними системами, обмежені можливості персоналізації замовлень та не завжди зручний інтерфейс користувача.

Було сформовано вимоги до веб-застосунку, які враховують актуальні потреби користувачів та особливості українського ринку громадського харчування. Розроблено детальні сценарії використання веб-застосунку, описано необхідний функціонал та визначено обмеження для першого випуску продукту та подальших версій.

Веб-сервіс побудовано за архітектурним стилем REST API. Взаємодія між клієнтською та серверною частинами реалізована через HTTP-запити до API, з передачею даних у форматі JSON. Серверна частина реалізована з використанням фреймворку Next.js, база даних — PostgreSQL, доступ до якої здійснюється через ORM Prisma. Фронтенд створено на основі Next.js із застосуванням TailwindCSS для стилізації інтерфейсу. Управління станом реалізовано за допомогою бібліотеки Zustand.

Було створено прототип веб-застосунку для автоматизації процесів оформлення замовлень у піцерії, з підтримкою кастомізації страв, інтеграцією платіжної системи LiqPay та соціальних мереж. У наступних версіях планується розширення функціональності веб-застосунку, зокрема — реалізація панелі адміністрування, розширені можливості керування меню та інтеграція з аналітичними модулями.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Next.js Documentation. URL: https://nextjs.org/docs (дата звернення: 06.05.2024).

2. Prisma ORM Documentation. URL: https://www.prisma.io/docs (дата звернення: 06.05.2024Next.js Documentation. URL: https://nextjs.org/docs (дата звернення: 06.05.2024).

3. Prisma ORM Documentation. URL: https://www.prisma.io/docs (дата звернення: 06.05.2024).

4. Vercel PostgreSQL. URL: https://vercel.com/docs/storage/vercel-postgres (дата звернення: 06.05.2024).

5. TailwindCSS Documentation. URL: https://tailwindcss.com/docs/installation (дата звернення: 06.05.2024).

6. NextAuth.js Documentation. URL: https://next-auth.js.org/getting-started/introduction (дата звернення: 06.05.2024).

7. Zustand State Management Documentation. URL: https://github.com/pmndrs/zustand (дата звернення: 06.05.2024).

8. LiqPay Documentation. URL: https://www.liqpay.ua/doc (дата звернення: 06.05.2024).

9. Instagram Graph API Documentation. URL: https://developers.facebook.com/docs/instagram-platform/instagram-api-with-instagram-login (дата звернення: 06.05.2024).

10. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/current/ (дата звернення: 06.05.2024).

11. Salunkhe G., Nagpurkar S., Kengale J. A. Boosting Productivity through Deep Learning: Strategies for Enhanced Efficiency. International Journal of Intelligent Systems and Applications in Engineering. 2024. 12(13). P. 396–406

12. Методичні вказівки до лабораторних робіт з дисципліни «Емпіричні методи програмної інженерії» для студентів усіх форм навчання першого (бакалаврського) рівня вищої освіти спеціальності 121 – Інженерія програмного

забезпечення, освітня програма Програмна інженерія / Упоряд.: І.В. Груздо, О.С. Назаров. – Електронне видання. – Харків: ХНУРЕ, 2024. – 127 с.

13. Методичні вказівки до практичних робіт з дисципліни «Емпіричні методи програмної інженерії» для студентів усіх форм навчання першого (бакалавського) рівня вищої освіти спеціальності 121 – Інженерія програмного забезпечення, освітня програма Програмна інженерія / Упоряд.: І.В. Груздо, О.С.Назаров. – Харків: ХНУРЕ, 2021. – 80 с.

14. «Системні технології» 4 (153) 2024 «System technologies», с. 47-57 DOI: 10.34185/1562-9945-4-153-2024-06

15. Пироженко, С. С. Дослідження продуктивності методів та засобів контролю глобального стану компонентів React додатків / С. С. Пироженко, Н. С. Лєсна // Наука онлайн : міжнародний електронний науковий журнал. – 2020. – №10. – С. 13. – URL: https://nauka-online.com/publications/informatsionnye-tehnologii/2020/10/doslidzhennya-produktivnosti-metodiv-ta-zasobiv-kontrolyu-globalnogo-stanu-komponentiv-react-dodatkiv/.

16. Соловей, І. В. Розробка програмної системи для ведення цифрової картки домашньої тварини / І. В. Соловей // Системні технології. – 2024. – № 4 (153). – С. 47–57. – DOI: 10.34185/1562-9945-4-153-2024-06.

17. Методичні вказівки до курсового проектування з дисципліни «Бази даних» для студентів усіх форм навчання спеціальності 121 – «Інженерія програмного забезпечення» (освітньо-професійна програма «Програмна інженерія») / упоряд.: О. О. Мазурова, М. С. Широкопетлєва, Ю. Ю. Черепанова, Д. О. Колесников. – Харків : ХНУРЕ, 2020. – 52 с. – URL: https://catalogue.nure.ua/download=237424.