

**Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення**

**КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»**

**РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ РЕСТОРАНУ З ОНЛАЙН
БРОНЮВАННЯМ СТОЛИКІВ**

**DEVELOPMENT OF A WEB APPLICATION FOR A RESTAURANT WITH
ONLINE TABLE BOOKING**

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-43
Котик Артем Андрійович

Керівник:
Ліщина Наталія Миколаївна

Кваліфікаційну роботу
допущено до захисту
«__» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних та інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Галузь знань: 12 «Інформаційні технології»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

«__» _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Котику Артему Андрійовичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Розробка вебзастосунку для ресторану з онлайн бронюванням столиків»

Керівник роботи: к.т.н, доцент Ліщина Н. М.

затверджені наказом закладу вищої освіти від «31» грудня 2025 р. № 541/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра «29» травня 2026 р.

3. Вихідні дані до роботи: вимоги до програмного продукту сформовано на основі аналізу шести аналогів у сфері ресторанного бізнесу та систем онлайн-бронювання столиків; цільова аудиторія – локальні ресторани, кафе та заклади харчування України з середнім потоком 50–200 бронювань на місяць; технологічна основа системи – Next.js 15, NestJS 11, Prisma 6, SQLite 3, інтерфейс програмного продукту реалізовано українською мовою на всіх рівнях взаємодії користувача із системою..

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): аналіз сучасного стану систем онлайн-бронювання столиків та автоматизації ресторанного бізнесу в Україні та світі, постановка завдань на розробку веб-додатку для ресторану, розробка клієнтської та серверної частин системи, проєктування та реалізація бази даних, тестування програмного продукту, забезпечення захисту даних і безпеки системи, а також економічне обґрунтування доцільності впровадження веб-додатку для ресторану з функцією онлайн-бронювання столиків.

5. Перелік графічного матеріалу: 10 рисунків, 6 лістингів, 7 таблиці.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	Ліщина Н. М.		
Специфікація вимог до розробленої системи	Ліщина Н. М.		
Розробка об'єкта проектування	Ліщина Н. М.		
Нормоконтроль	Суринович О. М.		
Гарант ОП	Ліщина Н. М.		
Показник запозичень тексту	%		
Академічна доброчесність	Ліщина Н. М.		

7. Дата видачі завдання «3» лютого 2026 р.

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 14.02.2026 р.	
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 03.03.2026 р.	
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 14.03.2026 р.	
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 14.04.2026 р.	
5	Практична реалізація об'єкта проектування та розробка бази даних	до 20.04.2026 р.	
6	Тестування та налагодження об'єкта проектування	до 04.05.2026 р.	
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедрі	до 29.05.2026 р.	

Здобувач вищої освіти _____

Артем КОТИК

Керівник кваліфікаційної роботи _____

Наталія ЛІЩИНА

АНОТАЦІЯ

Котик А. А. Розробка веб-додатку для ресторану з онлайн-бронюванням столиків із використанням сучасних технологій Next.js та Nest.JS. Рукопис.

Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків.

Спроектовано та реалізовано клієнт-серверну систему «Ресторан» із фронтомом на Next.js, бекендом на Nest.JS, базою даних SQLite з Prisma 6, автентифікацією через JWT у httpOnly-cookies та підтримкою email-сповіщень через Nodemailer із використанням Handlebars-шаблонів. Реалізовано функціонал для клієнтів і адміністраторів системи: проектування 20 моделей бази даних, створення 59 REST-ендпоінтів і 29 сторінок користувацького інтерфейсу. Система підтримує повний цикл взаємодії користувача із закладом від перегляду меню та бронювання столика до керування статусами замовлень і резервувань адміністратором.

Практична цінність роботи полягає у створенні повноцінного робочого прототипу веб-додатку, готового до запуску однією командою (`docker compose up` або `npm run dev`), із тестовими seed-даними (страви меню, демонстраційні бронювання та відгуки користувачів) і повною україномовною локалізацією інтерфейсу.

Ключові слова: веб-додаток, Next.js, Nest.JS, Prisma, TypeScript, React, JWT, автентифікація.

ABSTRACT

Kotyk A. A. Development of a Web Application for a Restaurant with Online Table Reservation Using Modern Next.js and NestJS Technologies. Manuscript.

Bachelor's qualification thesis for the Educational Program "Software Engineering" in Specialty 121 "Software Engineering". Lutsk National Technical University, Lutsk, 2026.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

A client-server system "Restaurant" was designed and implemented with a frontend based on Next.js, a backend based on NestJS, an SQLite database using Prisma 6, authentication via JWT in httpOnly cookies, and support for email notifications through Nodemailer using Handlebars templates. Functionality for both clients and system administrators was implemented, including the design of 20 database models, creation of 59 REST endpoints, and development of 29 user interface pages. The system supports the complete cycle of user interaction with the restaurant from browsing the menu and reserving a table to managing order and reservation statuses by the administrator.

The practical value of the work lies in the creation of a fully functional web application prototype, ready to run with a single command (docker compose up or npm run dev), including test seed data (menu dishes, demonstration reservations, and user reviews) and full Ukrainian localization of the interface.

Keywords: web application, Next.js, NestJS, Prisma, TypeScript, React, JWT, authentication.

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1 АНАЛІЗ МЕТОДІВ РОЗРОБКИ ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПОСТАНОВКА ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ	10
1.1 Аналіз сучасного стану проблеми	10
1.2 Постановка завдання на кваліфікаційну роботу бакалавра	12
РОЗДІЛ 2 СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ...	14
2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення	14
2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання ...	19
РОЗДІЛ 3 РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ РЕСТОРАНУ З ОНЛАЙН БРОНЮВАННЯМ СТОЛИКІВ.....	24
3.1 Практична реалізація об'єкта проєктування.....	24
3.2 Розробка бази даних	34
3.3 Захист інформаційно-комп'ютерної системи	35
3.4 Тестування та налагодження інформаційно-комп'ютерної системи	40
ВИСНОВКИ	42
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	44
ДОДАТКИ	46

ВСТУП

Актуальність теми. У сучасних умовах стрімкого розвитку інформаційних технологій та цифровізації сфери обслуговування особливої актуальності набуває автоматизація діяльності закладів ресторанного господарства. Використання веб технологій дозволяє значно покращити якість обслуговування клієнтів, оптимізувати внутрішні бізнес-процеси та забезпечити ефективну взаємодію між адміністрацією закладу і відвідувачами. Одним із важливих напрямів автоматизації є створення веб-додатків для ресторанів із можливістю онлайн-бронювання столиків, перегляду меню та оформлення замовлень у режимі реального часу.

На практиці багато ресторанів і кафе досі використовують традиційні методи обліку замовлень та бронювання столиків, зокрема телефонні дзвінки, паперові записи або окремі таблиці. Такий підхід створює низку проблем: дублювання бронювань, складність контролю доступності столиків, незручність для клієнтів при оформленні замовлень, а також збільшення навантаження на персонал закладу. Крім того, відсутність єдиної інформаційної системи ускладнює адміністрування меню, контроль замовлень та облік клієнтських даних.

Існуючі програмні рішення для ресторанного бізнесу часто є дорогими, складними у впровадженні або містять надлишкову функціональність, яка не відповідає потребам невеликих та середніх закладів харчування. Частина таких систем потребує придбання ліцензій, додаткового обладнання або спеціального навчання персоналу, що робить їх використання економічно недоцільним для багатьох підприємств.

Особливо важливим аспектом є забезпечення швидкого та зручного доступу користувачів до функцій системи через сучасний веб інтерфейс. Клієнти очікують можливості перегляду актуального меню, додавання страв до кошика, бронювання столиків та оформлення замовлень онлайн без необхідності особистого звернення до адміністратора. Для адміністрації ресторану важливими

є функції управління меню, контролю замовлень, редагування інформації про страви та керування бронюванням.

Таким чином, виникає необхідність у розробці сучасного веб-додатку для ресторану, який забезпечуватиме автоматизацію процесів онлайн-бронювання столиків, управління меню та оформлення замовлень. Використання сучасних технологій веб-розробки, зокрема фреймворків Next.js та Nest.JS, дозволяє створити продуктивний, масштабований та зручний у використанні програмний продукт із сучасним користувацьким інтерфейсом та надійною серверною частиною.

Отже, тема кваліфікаційної роботи, присвячена розробці веб-додатку для ресторану з функціями онлайн-бронювання столиків та оформлення замовлень із використанням технологій Next.js та Nest.JS, є актуальною, оскільки спрямована на підвищення ефективності роботи ресторанного закладу, покращення якості обслуговування клієнтів та автоматизацію основних бізнес-процесів.

Метою кваліфікаційної роботи є розробка веб-додатку для ресторану з можливістю онлайн-бронювання столиків, перегляду меню та оформлення замовлень із використанням сучасних технологій Next.js та Nest.JS.

Об'єктом дослідження є процес автоматизації обслуговування клієнтів у закладах ресторанного господарства.

Предметом дослідження є методи та засоби розробки веб-додатків для ресторанного бізнесу з використанням сучасних JavaScript/TypeScript технологій.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати сучасні підходи до автоматизації ресторанного бізнесу;
- визначити функціональні та нефункціональні вимоги до системи;
- обґрунтувати вибір технологій розробки веб-додатку;
- спроектувати архітектуру системи та структуру бази даних;
- реалізувати клієнтську частину веб-додатку на основі Next.js;
- реалізувати серверну частину із використанням Nest.JS;

- забезпечити функціонування системи онлайн-бронювання та кошика замовлень;
- провести тестування та налагодження програмного продукту.

Практичне значення роботи полягає у створенні сучасного веб-додатку для ресторану, який може бути використаний для автоматизації процесів бронювання столиків, управління меню та оформлення онлайн-замовлень, що сприятиме підвищенню ефективності роботи закладу та покращенню якості обслуговування клієнтів.

РОЗДІЛ 1

АНАЛІЗ МЕТОДІВ РОЗРОБКИ ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПОСТАНОВКА ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Сегмент онлайн-бронювання столиків та цифрового обслуговування у сфері ресторанного бізнесу має низку особливостей, які відрізняють його від класичних e-commerce платформ. Перша особливість полягає у необхідності роботи із динамічною інформацією в режимі реального часу: доступність столиків, кількість вільних місць та статус бронювання постійно змінюються, тому система повинна оперативнo оновлювати дані для уникнення конфліктів і подвійного резервування. Друга – критичність часу обслуговування: бронювання столика прив'язане до конкретної дати та часовго інтервалу, а затримки або помилки в роботі системи можуть негативно вплинути на репутацію закладу та задоволеність клієнтів [5]. Третя – висока роль візуальної складової та зручності інтерфейсу: користувачі очікують швидкого доступу до меню, фотографій страв, інформації про ресторан і можливості оформлення замовлення в декілька кліків, тому якісний UX/UI безпосередньо впливає на кількість замовлень і конверсію.

Учасниками процесу зазвичай є клієнт ресторану, адміністратор закладу, кухарі та персонал обслуговування. Клієнт переглядає меню, бронює столик та оформлює замовлення через веб інтерфейс. Адміністратор контролює бронювання, керує меню, редагує інформацію про страви та відстежує замовлення. Персонал ресторану отримує інформацію про нові замовлення та забезпечує їх виконання. Організація взаємодії між усіма цими ролями в межах єдиної системи є важливою задачею під час розробки веб-додатку.

Зазвичай невеликі ресторани та кафе в Україні використовують один із кількох підходів: прийом бронювань через телефонні дзвінки або соціальні мережі, використання сторінок у Instagram чи Facebook, підключення до сторонніх сервісів доставки або створення простих сайтів на конструкторах. Кожен із цих підходів має певні обмеження. Використання соціальних мереж не

забезпечує автоматизацію бронювання та централізоване зберігання даних про клієнтів і замовлення. Сторонні сервіси доставки часто стягують комісію та обмежують можливості кастомізації інтерфейсу [6]. Конструктори сайтів дозволяють швидко створити сторінку ресторану, однак не забезпечують повноцінний функціонал онлайн-бронювання, управління меню та інтеграції із серверною логікою.

Для порівняння було розглянуто кілька популярних рішень у сфері автоматизації ресторанного бізнесу: Glovo, Bolt Food, OpenTable, Poster POS, Tilda та власні вебрішення ресторанів. Зведене порівняння наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння існуючих рішень у сфері автоматизації ресторанного бізнесу

Рішення	Сильні сторони	Слабкі сторони	Висновок для проєкту
Glovo, Bolt Food	Відомість сервісу, готова логістика, велика аудиторія	Високі комісії, обмежений контроль над клієнтською базою, відсутність власного брендингу	Можуть використовуватись як додатковий канал замовлень
OpenTable	Зручна система онлайн-бронювання, сучасний UX	Обмежена доступність в Україні, орієнтація на міжнародний ринок	Використовується як приклад реалізації бронювання
Poster POS	Автоматизація ресторану, облік замовлень і персоналу	Потребує платної підписки, часткова залежність від стороннього сервісу	Підходить для великих закладів
Tilda	Простота створення сайту, сучасний дизайн	Обмежений функціонал, відсутність повноцінної серверної логіки	Підходить лише для простих сайтів-візиток
Instagram / Facebook	Швидкий старт, прямий контакт із клієнтами	Відсутність автоматизації бронювання та кошика	Ефективно лише для малого обсягу замовлень
Власна розробка вебдодатку ресторану	Повний контроль над функціоналом, можливість реалізації бронювання, кошика та управління меню, сучасний інтерфейс, адаптація під потреби закладу	Потребує часу на розробку та тестування	Саме цей підхід обрано для реалізації дипломного проєкту

Розробка власного веб-додатку для ресторану дозволяє реалізувати специфічний функціонал, необхідний для ефективної роботи закладу: онлайн-бронювання столиків, управління меню, оформлення замовлень через кошик, адміністрування контенту та зберігання інформації у базі даних. Використання сучасних технологій Next.js та Nest.JS забезпечує створення швидкого, масштабованого та зручного у використанні програмного продукту, який відповідає сучасним вимогам веб розробки та потребам ресторанного бізнесу.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Аналіз ринку, проведений у підрозділі 1.1, показав, що для ресторану середнього розміру найбільш ефективним рішенням є власний веб-додаток із декількома ролями користувачів – гість, зареєстрований користувач та адміністратор. Програмний продукт повинен забезпечувати повний цикл взаємодії користувача із закладом: від перегляду меню та бронювання столика анонімним відвідувачем до управління замовленнями, меню та бронюванням адміністратором ресторану [7].

На основі проведеного аналізу сформовано перелік завдань, які необхідно виконати у межах цієї кваліфікаційної роботи:

- 1) проаналізувати предметну область автоматизації ресторанного бізнесу та дослідити сучасні веб-рішення для онлайн-бронювання столиків і оформлення замовлень;
- 2) визначити функціональні та нефункціональні вимоги до програмного забезпечення на основі сценаріїв використання для різних категорій користувачів;
- 3) обґрунтувати вибір технологічного стека для клієнтської частини (фреймворк, UI-бібліотеки, управління станом, маршрутизація) та серверної частини (фреймворк, ORM, база даних, механізми автентифікації);

4) спроектувати архітектуру системи, структуру бази даних та UML-діаграми основних процесів, зокрема процесів авторизації, бронювання столиків та оформлення замовлень;

5) реалізувати клієнтську частину веб-додатку з підтримкою перегляду меню, кошика замовлень, онлайн-бронювання столиків, особистого кабінету користувача та адміністративної панелі;

6) реалізувати серверну частину з REST API, JWT-аутентифікацією, обробкою запитів користувачів, управлінням даними меню, бронюванням та замовленнями;

7) виконати тестування програмного продукту, провести перевірку основних функціональних сценаріїв та налагодження роботи системи;

8) підготувати технічну документацію, налаштувати розгортання веб-додатку та обґрунтувати практичну доцільність використання розробленої системи.

У межах проєкту достатньо реалізувати моделювання процесу оформлення замовлення та вибору способу оплати. Також до функціоналу системи не входить розробка окремого мобільного застосунку, оскільки веб-додаток створюється з адаптивним дизайном, що забезпечує коректну роботу на персональних комп'ютерах, планшетах і мобільних пристроях.

РОЗДІЛ 2

СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення

Виявлення вимог до системи проводилося у три етапи. Перший етап передбачав аналіз сучасних веб рішень у сфері ресторанного бізнесу, розглянутих у підрозділі 1.1. Було визначено перелік базових функцій, які реалізуються у більшості сучасних систем: перегляд меню, онлайн-бронювання столиків, кошик замовлень, особистий кабінет користувача та адміністративна панель. Другий етап включав аналіз потреб потенційних користувачів ресторанного веб-додатку та дослідження процесів обслуговування клієнтів у закладах харчування. У результаті було визначено специфічні вимоги ресторанної сфери: вибір дати та часу бронювання, управління доступністю столиків, оформлення замовлень із меню, контроль статусів замовлень та можливість адміністрування контенту [9]. Третій етап полягав у формуванні сценаріїв використання системи (user stories) для трьох категорій користувачів. Усі етапи стали основою для формування функціональних і нефункціональних вимог до програмного продукту.

У системі передбачено три категорії користувачів:

- гість (анонімний користувач) має доступ до перегляду головної сторінки, меню ресторану, інформації про заклад, контактів та сторінок авторизації і реєстрації. Гість може переглядати доступні страви та інформацію про ресторан, але не має можливості оформлювати замовлення або бронювати столики;

- користувач (роль USER) – зареєстрований клієнт ресторану. Окрім можливостей гостя, користувач отримує доступ до кошика, оформлення замовлень, онлайн-бронювання столиків, історії власних замовлень та особистого кабінету;

- адміністратор (роль ADMIN) має доступ до адміністративної панелі системи. Адміністратор може керувати меню ресторану, редагувати інформацію

про страви, контролювати бронювання столиків, переглядати всі замовлення користувачів, змінювати статуси замовлень та керувати користувачами системи.

Функціональні вимоги до системи структуровано наведено у таблиці 2.1.

Таблиця 2.1 – Функціональні вимоги до вебзастосунку ресторану

ID	Роль	Вимога	Реалізація
F01	Гість	Перегляд меню ресторану	Сторінка меню з отриманням даних через REST API
F02	Гість	Перегляд інформації про страву	Сторінка деталей страви
F03	Гість	Реєстрація та авторизація	POST /auth/register, POST /auth/login
F04	Користувач	Додавання страв до кошика	CartService
F05	Користувач	Зміна кількості страв у кошику	PATCH /cart
F06	Користувач	Оформлення замовлення	POST /orders
F07	Користувач	Онлайн-бронювання столика	POST /reservations
F08	Користувач	Перегляд історії замовлень	GET /orders/my
F09	Користувач	Перегляд власних бронювань	GET /reservations/my
F10	Адміністратор	Управління меню ресторану	CRUD для страв
F11	Адміністратор	Додавання та редагування категорій	CRUD для категорій
F12	Адміністратор	Контроль замовлень	GET /admin/orders
F13	Адміністратор	Зміна статусів замовлень	PATCH /admin/orders/:id
F14	Адміністратор	Управління бронюваннями	GET /admin/reservations
F15	Адміністратор	Завантаження зображень страв	POST /upload

Кожна вимога перевіряється шляхом ручного тестування або автоматизованих тестів. Нефункціональні вимоги наведено у таблиці 2.2 – вони описують загальні характеристики системи, зокрема продуктивність, безпеку, доступність та масштабованість. Дотримання нефункціональних вимог є важливим для забезпечення стабільної роботи веб-додатку в умовах одночасної взаємодії кількох користувачів, безпечного зберігання персональних даних та можливості подальшого розширення функціоналу системи без суттєвих змін архітектури. Крім того, врахування цих вимог дозволяє підвищити якість користувацького досвіду, зменшити час відгуку системи та забезпечити надійну роботу вебзастосунку на різних типах пристроїв і браузерів.

Таблиця 2.2 – Нефункціональні вимоги до системи ресторану

Категорія	Вимога	Реалізація у проєкті	Метод перевірки
Продуктивність	Швидке завантаження сторінок	SSR та оптимізація Next.js	Lighthouse
Продуктивність	Час відповіді API ≤ 200 мс	Оптимізація Prisma-запитів	DevTools
Безпека	Захист паролів	bcrypt-хешування	Code review
Безпека	JWT-аутентифікація	Access token	Тестування API
Безпека	Захист від XSS	Валідація та escaping	OWASP ZAP
Доступність	Адаптивний інтерфейс	Tailwind CSS	Тестування на різних пристроях
Масштабованість	Модульна архітектура	NestJS modules	Архітектурний аналіз
Зручність розгортання	Docker-розгортання	docker-compose	Запуск на чистій системі

Ролі користувачів і права доступу до функціоналу системи наведено у таблиці 2.3.

Таблиця 2.3 – Ролі користувачів та права доступу

Роль	Доступні frontend-маршрути	Доступні backend-ендпоінти	Обмеження
Гість (анонімний)	/, /menu, /about, /contacts, /login, /register	GET /menu, POST /auth/*	Не може оформляти замовлення
Користувач (USER)	+ /cart, /checkout, /profile, /reservations	/cart/, /orders/, /reservations/*	Доступ лише до власних даних
Адміністратор (ADMIN)	+ /admin/*	/admin/*	Повний доступ до системи

Діаграма варіантів використання демонструє взаємодію користувачів із системою та доступні для них сценарії. Архітектура системи побудована за принципом клієнт-серверної взаємодії: клієнтська частина реалізована за допомогою Next.js, серверна – із використанням NestJS та REST API.[13] Для роботи з базою даних використовується Prisma ORM [14]. Послідовність ключових процесів системи, зокрема авторизації користувача, оформлення замовлення та бронювання столика, представлена відповідними UML-

діаграмами. Діаграму варіантів використання вебзастосунку наведено на рисунку 2.1.

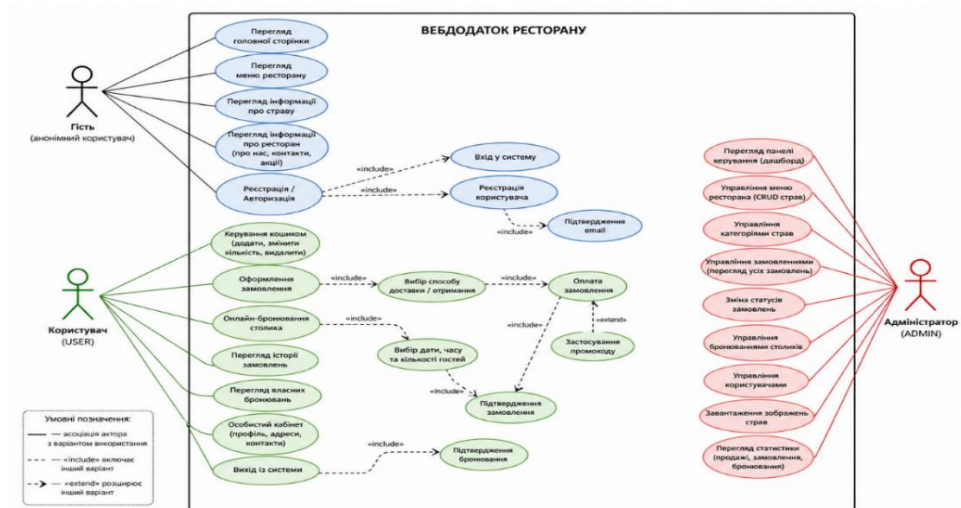


Рисунок 2.1 – Діаграма варіантів використання вебзастосунку ресторану

Дизайн веб-додатку орієнтований на створення сучасного та зручного інтерфейсу для користувачів ресторану. Основна увага приділяється візуальному представленню меню, фотографіям страв та простоті навігації. Інтерфейс побудований із використанням адаптивного дизайну, що забезпечує коректне відображення на мобільних пристроях, планшетах та персональних комп'ютерах. Для стилізації інтерфейсу використовується Tailwind [18] CSS, а дизайн-система реалізована через глобальні CSS-налаштування. Загальну архітектуру вебзастосунку наведено на рисунку 2.2.

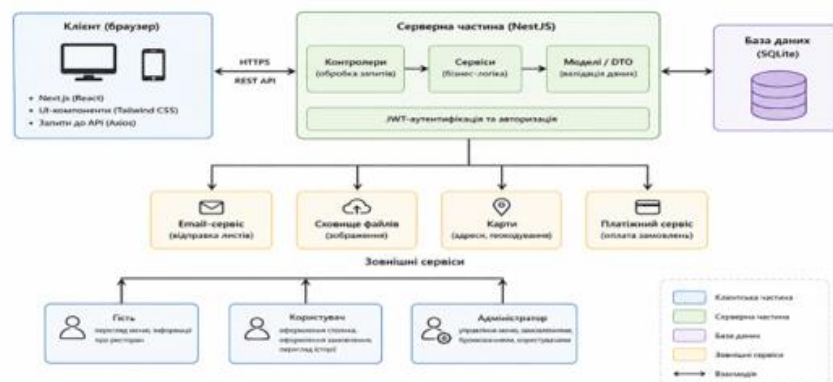


Рисунок 2.2 – Загальна архітектура вебзастосунку ресторану

У системі реалізовано кілька основних інформаційних потоків. Потік автентифікації включає введення користувачем логіна та пароля через форму входу, перевірку облікових даних на сервері та отримання JWT-токена для доступу до захищених ресурсів. Потік оформлення замовлення передбачає додавання страв у кошик, введення контактної інформації, підтвердження замовлення та збереження даних у базі.

Потік бронювання столика включає вибір дати, часу та кількості місць із подальшим створенням запису бронювання у системі. Адміністративний потік забезпечує можливість редагування меню, управління замовленнями та бронюванням через окрему адміністративну панель. Діаграму послідовності оформлення замовлення наведено на рисунку 2.3.

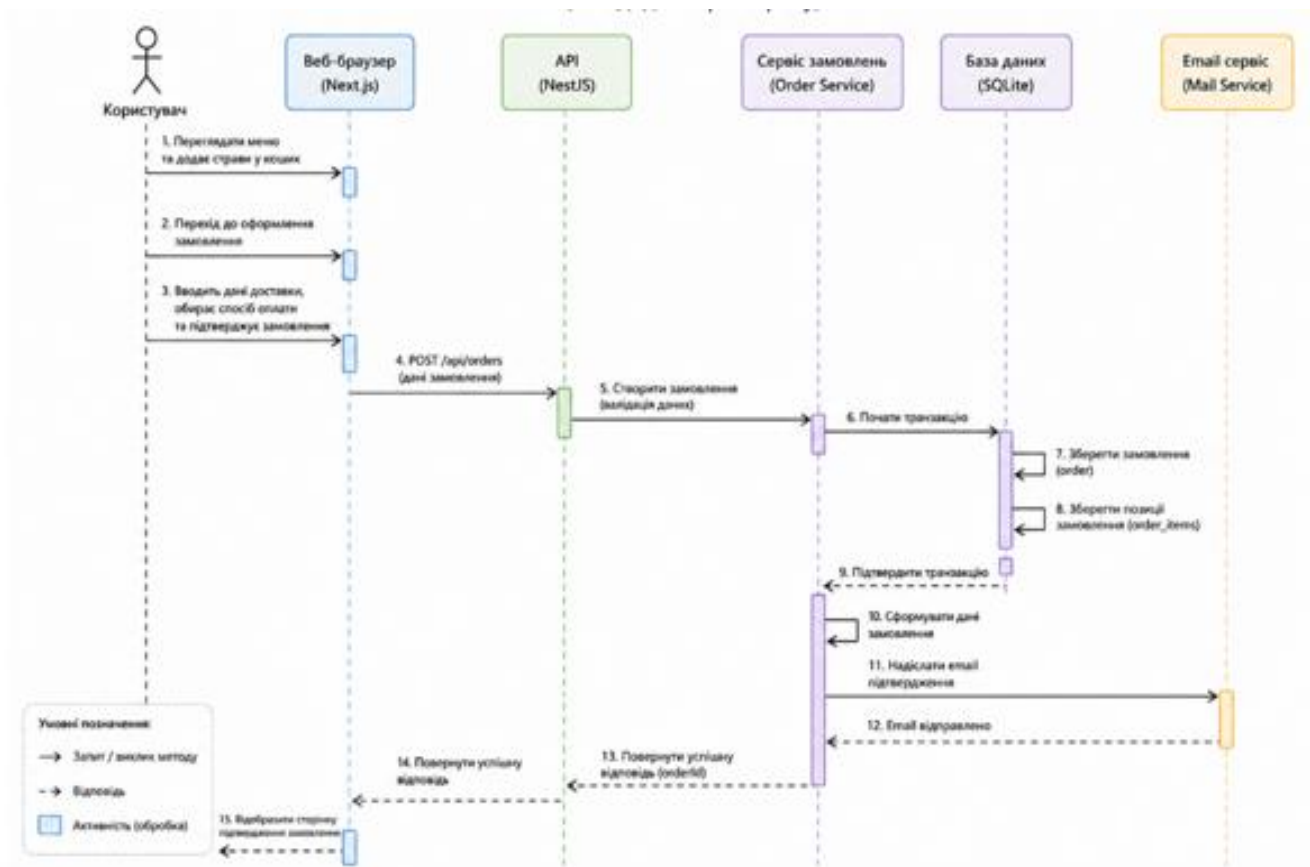


Рисунок 2.3 – Діаграма послідовності оформлення замовлення

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Для побудови клієнтської частини веб-додатку ресторану було обрано фреймворк Next.js версії 15 із використанням App Router. Під час вибору технології розглядалися також альтернативні рішення: Vite + React SPA, Remix та SvelteKit. Основними причинами вибору Next.js стали підтримка серверного рендерингу (SSR), статичної генерації сторінок (SSG) та клієнтського рендерингу (CSR) в межах одного фреймворку. Це дозволяє оптимізувати швидкість завантаження сторінок, покращити SEO та забезпечити високу продуктивність веб-додатку. Наприклад, статичні сторінки ресторану («Про нас», «Контакти») можуть генеруватися як SSG, сторінки меню – через SSR, а інтерактивні елементи, такі як кошик або бронювання столиків, працюють на клієнтській стороні. Додатковою перевагою є використання React Server Components, що зменшує обсяг JavaScript-коду, який передається браузеру користувача.

Як основну UI-бібліотеку обрано React 19 у поєднанні з TypeScript. React забезпечує компонентний підхід до побудови інтерфейсу та має широку екосистему бібліотек і готових рішень. Використання TypeScript у strict mode дозволяє забезпечити безпечність проєкту, що є важливим для складної структури веб-додатку ресторану, де реалізовано взаємодію між меню, кошиком, бронюванням та системою авторизації. На етапі компіляції TypeScript дозволяє виявляти помилки типів, що значно спрощує процес розробки та підтримки програмного забезпечення.

Для стилізації інтерфейсу використовується Tailwind CSS. Під час вибору технології також розглядалися CSS Modules та styled-components, однак Tailwind CSS був обраний завдяки високій швидкості розробки, зручній системі utility-класів та підтримці адаптивного дизайну [18]. Використання Tailwind CSS дозволяє створити сучасний інтерфейс із мінімальним обсягом окремих CSS-

файлів та забезпечує зручну підтримку стилів у межах компонентної архітектури React.

Для роботи із серверним станом та асинхронними запитами використовується TanStack Query (React Query). Альтернативою був Redux Toolkit, проте для вебдодатку ресторану використання Redux виявилось надлишковим. TanStack Query забезпечує автоматичне кешування даних, оновлення інформації після змін та спрощує роботу з REST API [19]. Для локального стану, зокрема даних авторизованого користувача та кошика, використовується React Context API [20].

Для реалізації форм та валідації даних використовується React Hook Form у поєднанні з бібліотекою Zod. Такий підхід забезпечує ефективну обробку форм без надмірних перерендерів компонентів та дозволяє використовувати єдині схеми валідації як для перевірки даних під час виконання програми, так і для генерації типів TypeScript [21].

На серверній стороні обрано Nest.JS – сучасний Node.js-фреймворк для створення масштабованих серверних застосунків. Під час аналізу альтернатив розглядалися Express та Fastify. Використання Nest.JS дозволило організувати серверну частину за модульним принципом із чітким розділенням контролерів, сервісів та моделей даних. Додатковими перевагами є підтримка dependency injection, використання декораторів та вбудовані механізми валідації й авторизації. Зведений технологічний стек наведено у таблиці 2.4.

Таблиця 2.4 – Технологічний стек вебзастосунку ресторану

Рівень	Технологія	Версія	Призначення
Runtime	Node.js	22.x LTS	Серверне середовище виконання
Frontend	Next.js	15.1	App Router, SSR/CSR, оптимізація сторінок
Frontend	React	19.0	Побудова користувацького інтерфейсу
Frontend	TypeScript	5.9	Типізація та strict mode
Frontend	Tailwind CSS	4.0	Стилізація та адаптивний дизайн
Frontend	TanStack Query	5.62	Робота із серверним станом та кешування

Продовження таблиці 2.4

Рівень	Технологія	Версія	Призначення
Frontend	React Hook Form + Zod	7.54 / 3.23	Форми та валідація даних
Frontend	axios	1.7	HTTP-клієнт для роботи з API
Frontend	sonner	1.7	Toast-сповіщення
Frontend	date-fns	4.1	Робота з датами та часом
Backend	NestJS	11.0	Серверний фреймворк, модульна архітектура
Backend	Prisma ORM	6.19	Робота з базою даних
Backend	Passport + JWT	11.0	Автентифікація користувачів
Backend	bcrypt	5.1	Хешування паролів
Backend	class-validator	0.14	Валідація DTO
Backend	Multer + Sharp	.4 / 0.33	Завантаження та обробка зображень
База даних	SQLite	3.x	Реляційна база даних
DevOps	Docker	24+	Контейнеризація застосунку
Тестування	Jest, Vitest	29.7 / 2.1	Модульне тестування

Для взаємодії з базою даних використовується Prisma ORM. Альтернативою була TypeORM, однак Prisma забезпечує більш зручну систему міграцій, автоматичну генерацію типів та окремий файл `schema.prisma`, який виступає єдиним джерелом опису структури бази даних. Prisma також значно спрощує роботу зі зв'язками між таблицями та формування SQL-запитів.

Як систему керування базами даних для розробки та демонстрації проєкту обрано SQLite. Основною причиною такого вибору є простота налаштування та відсутність необхідності запуску окремого сервера бази даних. SQLite дозволяє швидко розгорнути систему без додаткових залежностей, що є доцільним для дипломного проєкту. При цьому структура Prisma-схеми дозволяє у майбутньому без значних змін перейти на PostgreSQL або MySQL.

Для реалізації автентифікації використовується JWT (JSON Web Token) із зберіганням токенів у `httpOnly cookies`. Такий підхід забезпечує stateless-автентифікацію та дозволяє підвищити безпеку системи завдяки захисту токенів від XSS-атак. Альтернативний підхід із серверними сесіями не використовувався через необхідність додаткового сервісу для зберігання сесій.

Для хешування паролів використовується бібліотека `bcrypt` із параметром `cost factor 12`. Використання `bcrypt` дозволяє забезпечити безпечне зберігання паролів користувачів відповідно до сучасних вимог інформаційної безпеки та рекомендацій OWASP [22].

Таким чином, обраний технологічний стек забезпечує створення сучасного, масштабованого та безпечного веб-додатку ресторану з підтримкою онлайн-бронювання столиків, оформлення замовлень та адміністрування контенту. Перелік основних API-ендпоінтів – у таблиці 2.5.

Таблиця 2.5 – Основні API-ендпоінти вебзастосунку ресторану

Метод	Endpoint	Guard / роль	Опис
POST	/api/v1/auth/register	–	Реєстрація нового користувача
POST	/api/v1/auth/login	Throttler 5/хв	Авторизація користувача та видача JWT
POST	/api/v1/auth/refresh	refresh token	Оновлення access token
POST	/api/v1/auth/logout	JwtAuthGuard	Вихід із системи та очищення cookies
GET	/api/v1/auth/me	JwtAuthGuard	Отримання даних поточного користувача
POST	/api/v1/auth/verify-email	–	Підтвердження email
POST	/api/v1/auth/forgot-password	Throttler 3/хв	Запит на скидання пароля
POST	/api/v1/auth/reset-password	–	Встановлення нового паролю
GET	/api/v1/menu	–	Отримання списку страв із фільтрами та пагінацією
GET	/api/v1/menu/:id	–	Отримання детальної інформації про страву
GET	/api/v1/categories	–	Отримання списку категорій меню
GET	/api/v1/cart	JwtAuthGuard	Отримання поточного кошика користувача
POST	/api/v1/cart/items	JwtAuthGuard	Додавання страви до кошика
POST	/api/v1/orders	JwtAuthGuard	Створення замовлення з кошика
PATCH	/api/v1/cart/items/:id	JwtAuthGuard	Оновлення кількості страв у кошику
DELETE	/api/v1/cart/items/:id	JwtAuthGuard	Видалення страви з кошика
POST	/api/v1/orders	JwtAuthGuard	Створення нового замовлення
GET	/api/v1/orders/my	JwtAuthGuard	Перегляд історії власних замовлень
GET	/api/v1/orders/:id	JwtAuthGuard	Перегляд деталей замовлення
POST	/api/v1/reservations	JwtAuthGuard	Онлайн-бронювання столика

Продовження таблиці 2.5

Метод	Endpoint	Guard / роль	Опис
GET	/api/v1/reservations/my	JwtAuthGuard	Перегляд власних бронювань
DELETE	/api/v1/reservations/:id	JwtAuthGuard	Скасування бронювання
GET	/api/v1/admin/stats	@Roles(ADMIN)	Отримання статистики для адміністративної панелі
GET	/api/v1/admin/orders	@Roles(ADMIN)	Перегляд усіх замовлень

РОЗДІЛ 3

РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ РЕСТОРАНУ З ОНЛАЙН БРОНЮВАННЯМ СТОЛИКІВ

3.1 Практична реалізація об'єкта проєктування

Реалізація веб-додатку ресторану виконувалась ітеративно, із поступовим додаванням функціональних модулів та фіксацією кожного завершеного етапу у системі контролю версій Git. Основними етапами розробки стали: створення базової структури проєкту, реалізація серверної частини з автентифікацією та початковими даними, створення клієнтської частини з основними сторінками, інтеграція функціоналу замовлень і бронювання столиків, додавання тестування та Docker-контейнеризації, а також виправлення помилок конфігурації та мережевих портів [26]. Такий підхід дозволив ізолювати окремі зміни та забезпечив можливість швидкого повернення до стабільних версій системи у разі виникнення регресій. Головну сторінку вебзастосунку можна побачити на рисунку 3.1.

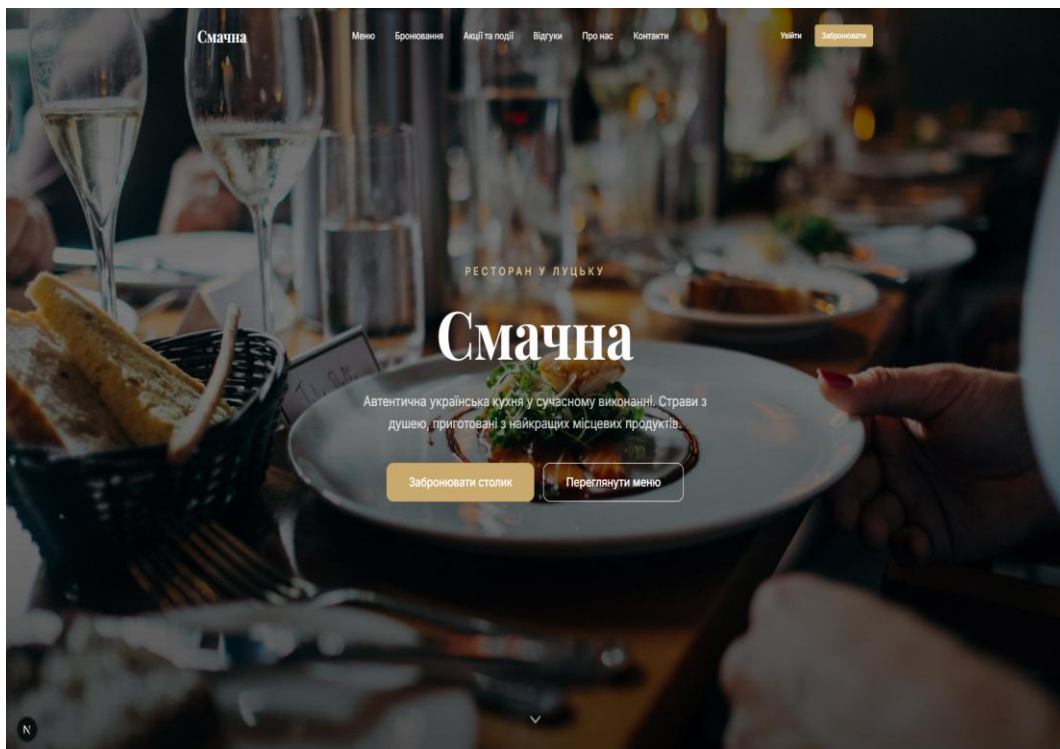


Рисунок 3.1 – Головна сторінка вебзастосунку Ресторану

Кодова база проєкту поділена на два окремі пакети у межах monorepo-структури: backend/ для серверної частини на Nest.JS та frontend/ для клієнтської частини на Next.js. У кореновому package.json використано npm workspaces та налаштовано спільний скрипт npm run dev, який запускає обидва сервери паралельно через бібліотеку concurrently. На рівні TypeScript обидві частини проєкту налаштовані у strict mode із параметрами strict: true, noImplicitAny: true та strictNullChecks: true, що забезпечує сувору типізацію та зменшує кількість помилок під час розробки.

Точкою входу серверної частини є файл backend/src/main.ts. У ньому виконується ініціалізація Nest.JS-застосунку та послідовне підключення middleware безпеки. Для захисту HTTP-заголовків використовується Helmet із налаштованою Content Security Policy, для роботи з cookies – cookie-parser, а для забезпечення взаємодії між frontend і backend налаштовано CORS з обмеженням доступу лише для FRONTEND_URL та параметром credentials: true для підтримки httpOnly-cookies.

Для глобальної перевірки даних використовується ValidationPipe з параметрами whitelist: true, forbidNonWhitelisted: true та transform: true. Це дозволяє автоматично відкидати поля, які не описані у DTO, запобігаючи mass-assignment атакам, а також автоматично перетворювати query-параметри у відповідні типи даних. Усі REST API-ендпоінти мають спільний префікс /api/v1, а документація Swagger підключена за маршрутом /api/docs, що спрощує тестування та налагодження серверної частини веб-додатку [27]. Фрагмент коду наведено у лістингу 3.1.

Лістинг 3.1 – Bootstrap Nest.JS-додатку з налаштуванням безпеки і Swagger

```
import 'reflect-metadata';
import { NestFactory, Reflector } from '@nestjs/core';
import { ClassSerializerInterceptor,
Logger,
ValidationPipe,
} from '@nestjs/common';
```

```

import { DocumentBuilder, SwaggerModule } from '@nestjs/swagger';
import helmet from 'helmet';
import cookieParser from 'cookie-parser';
import { AppModule } from './app.module';
async function bootstrap(): Promise<void> {
  const app = await NestFactory.create(AppModule, {
    logger: ['log', 'error', 'warn', 'debug'],
  });
  app.use(
    helmet({
      crossOriginResourcePolicy: {
        policy: 'cross-origin',
      },
    }),
  );
  app.use(cookieParser());
  app.enableCors({
    origin: process.env.FRONTEND_URL ?? 'http://localhost:3000',
    credentials: true,
  });
}

```

кінець лістингу 3.1

Кореневий модуль AppModule об'єднує основні функціональні модулі веб-додатку ресторану: AuthModule, UsersModule, MenuModule, CategoriesModule, CartModule, OrdersModule, ReservationsModule та UploadModule. Також підключено глобальні провайдери: AllExceptionsFilter для єдиного формату помилок, TransformInterceptor для стандартизації відповідей API, ThrottlerGuard для rate-limit, JwtAuthGuard для авторизації та RolesGuard для перевірки ролей користувачів.

PrismaService реалізовано як глобальний сервіс через PrismaModule, що дозволяє використовувати єдине підключення до бази даних у всіх модулях системи. Сервіс наслідується від PrismaClient і автоматично виконує \$connect при запуску застосунку та \$disconnect при його завершенні.

Модуль автентифікації реалізовано у `auth.service.ts`. Він містить методи `register`, `login`, `refresh`, `logout`, `forgotPassword` і `resetPassword`. Під час реєстрації система перевіряє унікальність `email`, виконує хешування пароля через `bcrypt` з `cost factor 12` та створює користувача у базі даних.

Для авторизації використовується JWT-автентифікація з `access` і `refresh token`. `Access token` має короткий термін дії, а `refresh token` додатково хешується через `SHA-256` перед збереженням у базі даних, що підвищує безпеку системи. Лістинг 3.2 показує цей фрагмент.

Лістинг 3.2 – Видача `access`- і `refresh`-токенів з збереженням хешу

```

async generateTokens(
  userId: number,
  email: string,
  role: UserRole,
): Promise<{
  accessToken: string;
  refreshToken: string;
}> {
  const payload = {
    sub: userId,
    email,
    role,
  };
  const accessToken = await this.jwtService.signAsync(payload, {
    secret: this.configService.get<string>('JWT_ACCESS_SECRET'),
    expiresIn: '15m',
  });
  const refreshToken = await this.jwtService.signAsync(payload, {
    secret: this.configService.get<string>('JWT_REFRESH_SECRET'),
    expiresIn: '7d',
  });
  await this.prisma.refreshToken.create({
    data: {
      userId,

```

```

    token: createHash('sha256')
      .update(refreshToken)
      .digest('hex'),
    expiresAt: new Date(Date.now() + 7 * 24 * 60 * 60 * 1000),
  },
});
return {
  accessToken,
  refreshToken,
};
}

```

кінець лістингу 3.2

Оформлення замовлення є одним із ключових бізнес-процесів веб-додатку ресторану. Метод `OrdersService.createOrder` працює як єдина транзакція через `prisma.$transaction()`: створюється запис замовлення з інформацією про користувача, адресу доставки, спосіб оплати та загальну суму. Одночасно створюються записи `order_items`, у яких зберігаються `snapshot`-поля (`dishName`, `dishPrice`, `dishImage`), що гарантує незмінність історії замовлення навіть після редагування або видалення страв із меню. Після успішного створення замовлення кошик користувача автоматично очищується. Надсилення email-повідомлення виконується окремо від транзакції, оскільки не впливає на цілісність даних. Окремої уваги потребує реалізація кошика.

Початкова версія використовувала звичайний `create()` для додавання страв, що викликало помилки при повторному додаванні однакової позиції через порушення `unique`-індексу (`cartId`, `productId`). Для вирішення цієї проблеми реалізацію змінено на Prisma `upsert`, який або створює новий запис, або збільшує кількість страв через `{ increment: quantity }`, як показано у лістингу 3.3.

Лістинг 3.3 – Додавання страви у кошик через Prisma `upsert`

```

async addToCart(userId: number, productId: number, quantity = 1) {
  if (quantity < 1) {

```

```
throw new BadRequestException('Кількість повинна бути більшою за 0');
}
const product = await this.prisma.product.findUnique({
  where: { id: productId },
});
if (!product || !product.isAvailable) {
  throw new NotFoundException('Страва недоступна');
}
const cart = await this.prisma.cart.upsert({
  where: { userId },
  create: { userId },
  update: {},
});
await this.prisma.cartItem.upsert({
  where: {
    cartId_productId: {
      cartId: cart.id,
      productId,
    },
  },
  create: {
    cartId: cart.id,
    productId,
    quantity,
  },
  update: {
    quantity: {
      increment: quantity,
    },
  },
});
return this.getCart(userId);
}
```

кінець лістингу 3.3

Клієнтська частина вебзастосунку ресторану побудована на Next.js App Router із використанням route groups для логічного розділення інтерфейсу. Папка (public) містить публічні сторінки меню, головної сторінки, контактів та інформації про ресторан із загальними компонентами Header і Footer. Меню можна переглянути на рисунку 3.2.

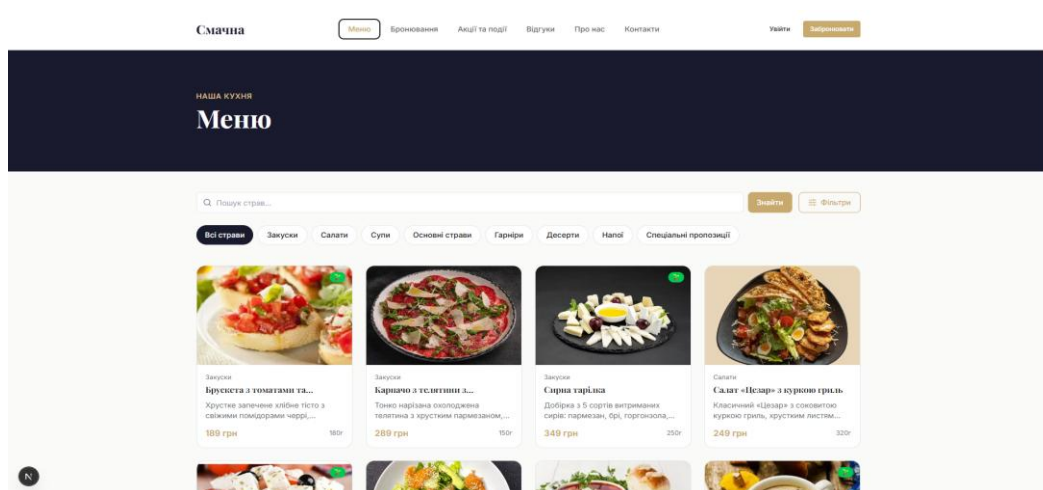


Рисунок 3.2 – Меню

Інформації про ресторан із загальними компонентами Header і Footer можна побачити на рисунку 3.3.

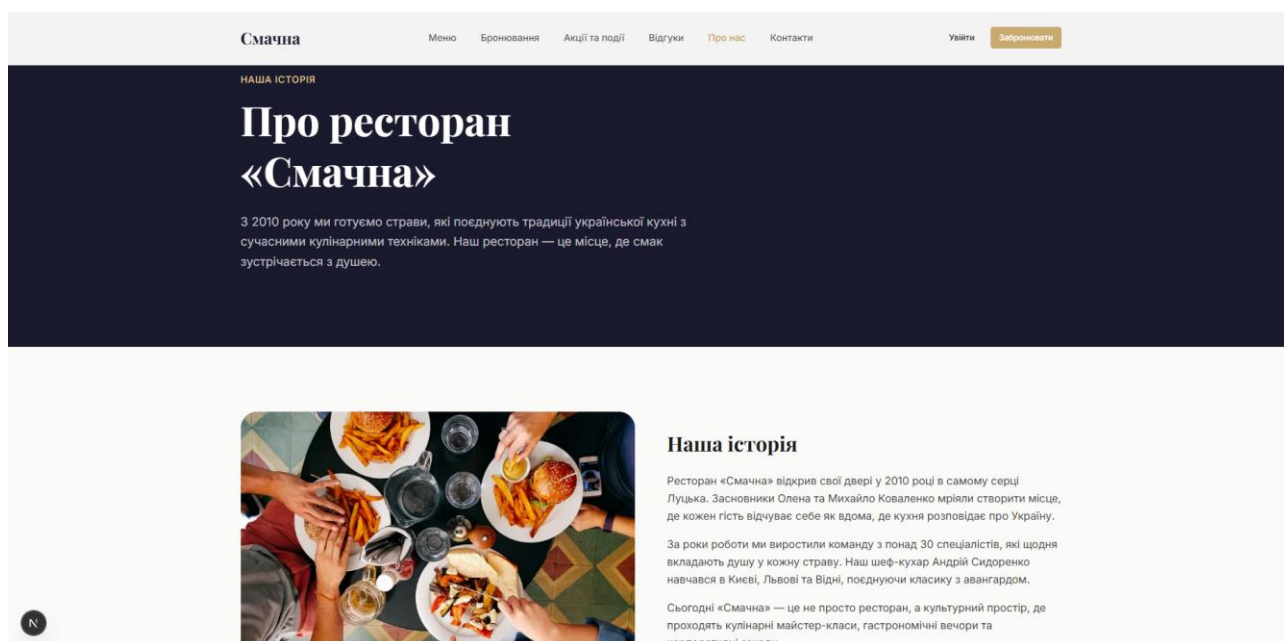


Рисунок 3.3 – Сторінка з інформацією про ресторан

Група (auth) використовується для сторінок входу та реєстрації й має окремий мінімалістичний layout без навігаційної панелі. Маршрути account/ та admin/ є захищеними й містять окремі бічні меню для користувача та адміністратора. Кожна група використовує власний layout.tsx, що дозволяє розділити структуру інтерфейсу залежно від ролі користувача.

У кореневому layout.tsx підключено шрифти через next/font/google, що забезпечує оптимізоване завантаження та підтримку кирилиці. Для стилізації використовується Tailwind CSS v4 із глобальними дизайн-токенами, винесеними у globals.css.

Глобальний клієнтський стан розділено на окремі провайдери. AuthProvider відповідає за збереження інформації про авторизованого користувача та автоматичне відновлення сесії через запит GET /auth/me. CartProvider керує станом кошика і синхронізує його з сервером після кожної зміни. Для серверного стану (меню, категорії, замовлення, бронювання) використовується TanStack Query, який забезпечує кешування, автоматичне оновлення даних та інвалідацію запитів. Для відображення повідомлень про успішні дії або помилки використовується бібліотека sonner.

Взаємодія клієнта із сервером реалізована через окремий API-клієнт на базі axios. У файлі lib/api/client.ts axios-інстанс налаштований із withCredentials: true, що дозволяє автоматично передавати httpOnly-cookies. Response-інтерсептор перехоплює помилки 401 Unauthorized та виконує автоматичне оновлення access token через POST /auth/refresh. Після успішного оновлення початковий запит повторюється повторно, що забезпечує безперервну роботу користувача без необхідності повторного входу в систему. Це наведено у лістингу 3.4

Лістинг 3.4 – Axios-інтерсептор для автоматичного оновлення JWT-токена

```
import axios, { AxiosError } from 'axios';
export const api = axios.create({
  baseURL:
    process.env.NEXT_PUBLIC_API_URL ??
    'http://localhost:4000/api/v1',
```

```

withCredentials: true,
headers: {
  'Content-Type': 'application/json',
},
});
let refreshing: Promise<unknown> | null = null;
api.interceptors.response.use(
  (response) => response,
  async (error: AxiosError) => {
    const originalRequest = error.config;
    if (
      error.response?.status === 401 &&
      originalRequest &&
      !(originalRequest as { _retry?: boolean })._retry &&
      !originalRequest.url?.includes('/auth/')
    ) {
      (originalRequest as { _retry?: boolean })._retry = true;
      try {
        if (!refreshing) {
          refreshing = api
            .post('/auth/refresh')
            .finally(() => {
              refreshing = null;
            });
        }
      }
    }
  }
}

```

кінець лістингу 3.4

Інтерфейс бронювання столика демонструє інтеграцію клієнтських і серверних компонентів системи. У файлі форма побудована через `useForm<FormData>` із використанням `zodResolver(schema)`. Схема Zod описує всі обов'язкові поля: ім'я клієнта, номер телефону, дату, час бронювання та кількість гостей. Наприклад, поле номера телефону перевіряється через регулярний вираз із повідомленням українською мовою у випадку помилки. Валідація реалізована на двох рівнях: на клієнті через `React Hook Form` для

миттєвого зворотного зв'язку та на сервері через class-validator у DTO. Це дозволяє зменшити кількість помилкових запитів і забезпечує додатковий рівень безпеки. Сторінку замовлень можна побачити на рисунку 3.4.

КОД	ГІСТЬ	СТОЛИК	ДАТА/ЧАС	ГОСТЕЙ	СТАТУС	Дії
REZ-DEPO-815	Марія Шевченко +380503333333	Столик №6	11 червня 2026 р. 18:00 – 20:00	4	Очікує	Змінити...
REZ-DEPO-814	Богдан Коваль +380501414141	Столик №1	21 травня 2026 р. 12:00 – 14:00	2	Завершено	
REZ-DEPO-813	Яна Ковальчук +380501313131	Тераса №3	4 червня 2026 р. 19:00 – 21:00	6	Підтверджено	Змінити...
REZ-DEPO-812	Андрій Лисенко +380501212121	Тераса №2	16 травня 2026 р. 19:00 – 21:00	4	Завершено	
REZ-DEPO-811	Іван Петренко +380502222222	Столик №4	25 травня 2026 р. 13:00 – 15:00	2	Очікує	Змінити...
REZ-DEPO-818	Людмила Сидоренко +380501010101	Тераса №1	18 травня 2026 р. 12:00 – 14:00	2	Не з'явився	
REZ-DEPO-809	Василь Ткаченко +380509999999	Приватна кімната	31 травня 2026 р. 18:00 – 20:00	12	Очікує	Змінити...
REZ-DEPO-808	Олена Коваленко +380501111111	Столик №7	28 травня 2026 р. 20:00 – 22:00	4	Підтверджено	Змінити...
REZ-DEPO-807	Оксана Мельник +380507777777	Столик №3	19 травня 2026 р. 19:00 – 21:00	2	Скасовано	
REZ-DEPO-806	Павло Кравченко +380506666666	Столик №6	20 травня 2026 р. 13:00 – 15:00	3	Завершено	
REZ-DEPO-805	Наталія Гриценко +380505555555	VIP-1	26 травня 2026 р. 18:00 – 20:00	8	Підтверджено	Змінити...
REZ-DEPO-804	Тарас Бондаренко +380504444444	Столик №2	21 травня 2026 р. 19:00 – 21:00	2	Розміщено	Змінити...

Рисунок 3.4 – Адмінська таблиця замовлень з можливістю зміни статусу

Під час реалізації сторінки перегляду бронювань виникла проблема надлишкових запитів до бази даних. Початкова версія окремо завантажувала інформацію про бронювання та пов'язані столи, що створювало проблему $n+1$ queries. Для її усунення було використано Prisma include, завдяки чому дані про бронювання та пов'язані записи завантажуються одним SQL-запитом.

Для адміністратора реалізовано окрему панель /admin, у якій доступне управління меню, замовленнями, бронюванням та користувачами зображено на рисунку 3.5. Адміністративна панель побудована у вигляді окремого захищеного інтерфейсу з перевіркою ролі користувача через RolesGuard та JWT-аутентифікацію. У межах панелі адміністратор може змінювати статуси замовлень, підтверджувати або скасовувати бронювання, редагувати інформацію про страви та переглядати статистичні дані системи.

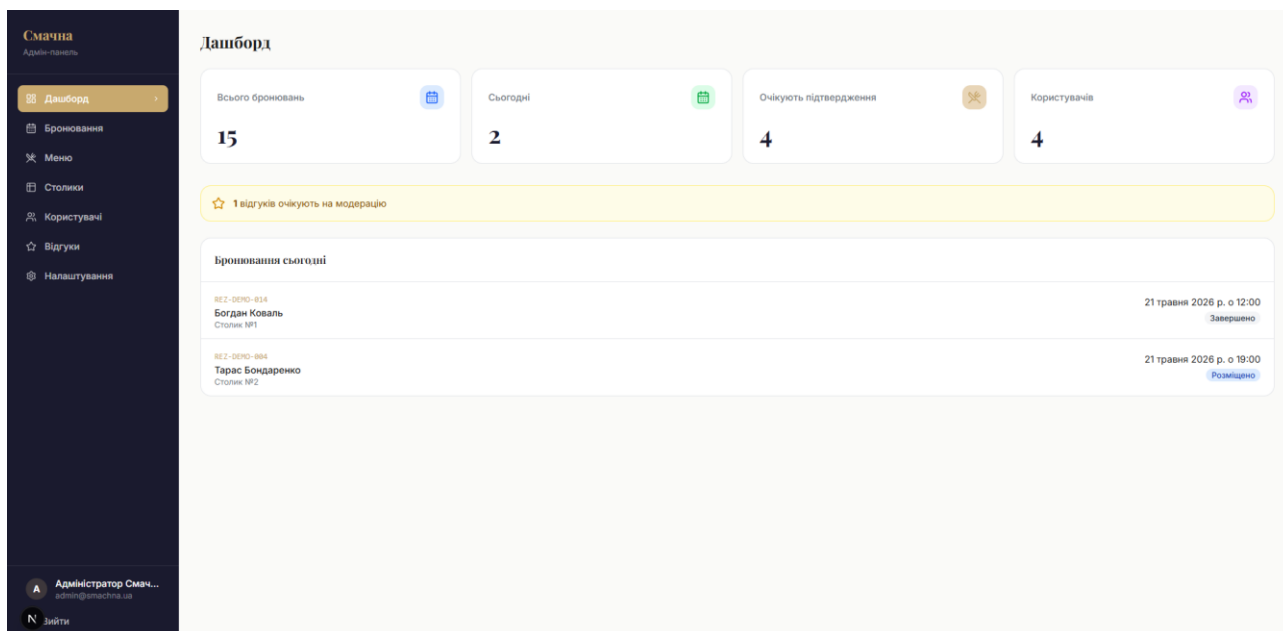


Рисунок 3.5 – Адмінський дашборд зі статистикою

Частина CRUD-операцій виконується через Swagger-документацію API та Prisma Studio, що спрощує розробку і дозволяє зосередитись на бізнес-логіці застосунку.

Email-сповіщення реалізовано через MailService із використанням Nodemailer і шаблонів Handlebars. Після успішного бронювання користувач отримує лист-підтвердження із деталями резервування столика. У режимі розробки використовується jsonTransport, який виводить листи у консоль без потреби підключення реального SMTP-сервера.

Завантаження зображень для меню ресторану реалізовано через Multer і Sharp. Файли приймаються у форматі multipart/form-data, після чого автоматично оптимізуються через Sharp: змінюється розмір зображення та виконується конвертація у формат WebP. Оброблені файли зберігаються у директорії backend/uploads, а доступ до них забезпечується через ServeStaticModule.

3.2 Розробка бази даних

Концептуальна модель даних системи онлайн-бронювання столиків базується на основних сутностях, що забезпечують роботу веб-додатку

ресторану: користувачі та їхні ролі, столики, бронювання, часові слоти, меню, категорії страв, замовлення, відгуки, повідомлення та журнал email-сповіщень. Модель охоплює як клієнтську частину (бронювання, перегляд меню, історія замовлень), так і адміністративну (керування столиками, меню, бронюванням та користувачами).

Основними сутностями системи є: User, Reservation, Table, MenuCategory, MenuItem, Order, OrderItem, Review, Notification, RefreshToken та EmailLog. Зв'язки між ними забезпечують цілісність даних і підтримують бізнес-логіку ресторану. Наприклад, один користувач може мати багато бронювань і замовлень, а кожне бронювання прив'язане до конкретного столика, дати та часового слоту.

Для забезпечення історичності даних у замовленнях використовуються snapshot-поля (назва страви, ціна на момент замовлення), що дозволяє зберігати коректну інформацію навіть після редагування меню адміністратором. Також у системі реалізовано зв'язки для відгуків користувачів про заклад і окремі страви, а таблиця EmailLog використовується для журналювання відправлених повідомлень щодо підтвердження бронювання або зміни його статусу.

Повна ER-діаграма системи наведена на рисунку 3.6.

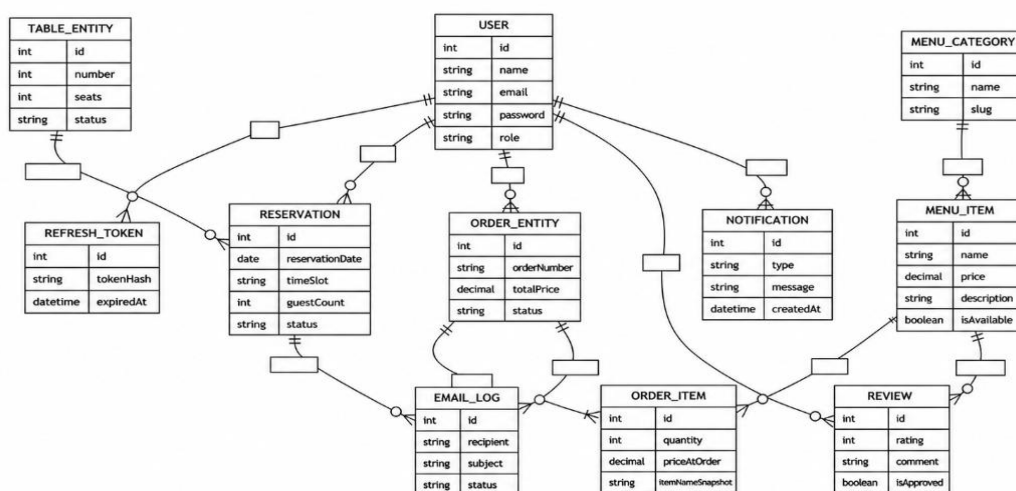


Рисунок 3.6 – Спрощена діаграма бази даних

Лістинг 3.5 – Модель Order у схемі Prisma зі snapshot-полями і історією

```

model Reservation {
  id Int @id @default(autoincrement())
  reservationCode String @unique
  userId Int?
  tableId Int
  guestName String
  guestEmail String
  guestPhone String
  reservationDate DateTime
  timeSlot String
  guestsCount Int
  status String @default("PENDING") // ReservationStatus
  comment String?
  adminNote String?
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  confirmedAt DateTime?
  cancelledAt DateTime?
  user User? @relation(fields: [userId], references: [id])
  table Table @relation(fields: [tableId], references: [id])
  history ReservationStatusHistory[]
  @@index([userId])
  @@index([tableId])
  @@index([status])
  @@index([reservationDate])
  @@map("reservations")
}

model ReservationStatusHistory {
  id Int @id @default(autoincrement())
  reservationId Int
  oldStatus String?
}

```

кінець лістингу 3.5

3.3 Захист інформаційно-комп'ютерної системи

Безпека веб-додатку ресторану з онлайн-бронюванням столиків побудована на багаторівневому підході відповідно до рекомендацій OWASP ASVS Level 1. Основний акцент зроблено на захисті аутентифікації, персональних даних користувачів та адміністративної панелі.

Для автентифікації використовується JWT-механізм із access- і refresh-токенами, що зберігаються у httpOnly-cookies. Access-токен має короткий термін дії, а refresh-токен використовується для автоматичного оновлення сесії без повторного введення пароля. Усі refresh-токени зберігаються у базі даних у вигляді SHA-256 хешів, що унеможлиблює їхнє використання навіть у випадку компрометації БД.

Лістинг 3.6 – Реалізація механізмів безпеки

```
// app.module.ts
import { ThrottlerModule } from '@nestjs/throttler';
import { APP_GUARD } from '@nestjs/core';
@Module({
  imports: [
    ThrottlerModule.forRoot([
      {
        name: 'short',
        ttl: 60000,
        limit: 20,
      }
    ]),
  ],
  providers: [
    { provide: APP_GUARD, useClass: JwtAuthGuard },
    { provide: APP_GUARD, useClass: RolesGuard },
  ]
})
export class AppModule {}

// main.ts
async function bootstrap() {
```

```

const app = await NestFactory.create(AppModule);
app.use(helmetMiddleware());
app.useGlobalPipes(
  new ValidationPipe({
    whitelist: true,
    transform: true,
    forbidNonWhitelisted: true,
  }),
);
app.enableCors({
  origin: process.env.FRONTEND_URL,
  credentials: true,
});
}
// auth.service.ts
async login(dto: LoginDto, userAgent?: string, ip?: string) {
  const user = await this.usersService.findByEmail(dto.email);
  const isPasswordValid = await bcrypt.compare(
    dto.password,
    user.passwordHash,
  );
  if (!isPasswordValid) {
    throw new UnauthorizedException();
  }
  const tokens = await this.generateTokens(
    user.id,
    user.email,
    user.role,
  );
  await this.saveRefreshToken(
    user.id,
    tokens.refreshToken,
    userAgent,
    ip,
  );
};

```

Додатково реалізовано захист від brute-force атак через ThrottlerGuard, валідацію всіх DTO за допомогою class-validator, автоматичне очищення небезпечних полів через ValidationPipe та захист HTTP-заголовків через Helmet. Ролі USER та ADMIN перевіряються через RolesGuard, а доступ до адміністративних маршрутів дозволений лише адміністраторам.

Послідовність аутентифікації та ротації токенів у системі онлайн-бронювання столиків наведена на рисунку 3.7.

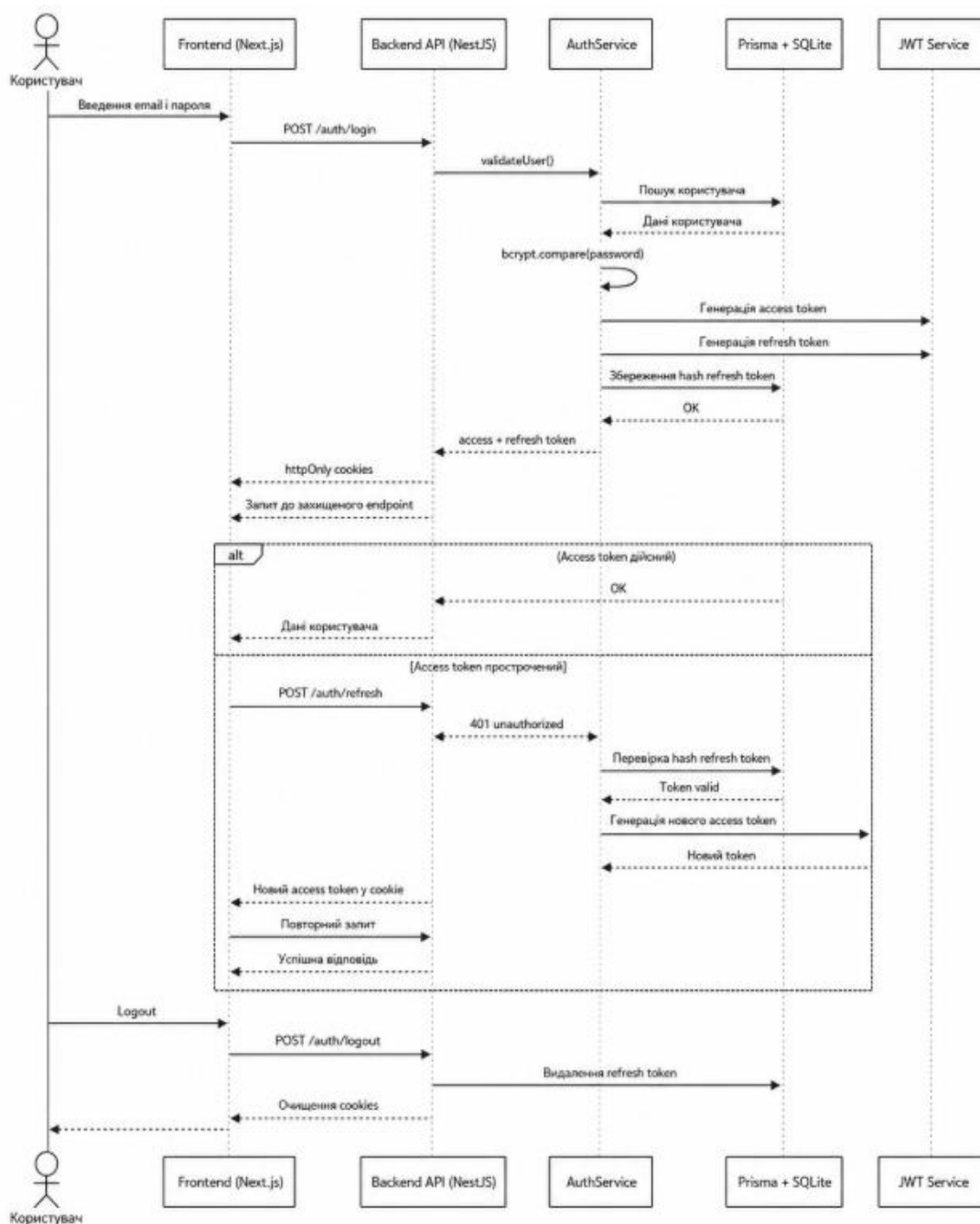


Рисунок 3.7 – Послідовність аутентифікації з ротацією refresh-токенів

3.4 Тестування та налагодження інформаційно-комп'ютерної системи

Стратегія тестування веб-додатку ресторану з онлайн-бронюванням столиків базується на класичній піраміді тестування: юніт-тести → інтеграційні тести → end-to-end перевірка. У межах проєкту реалізовано набір автоматизованих тестів для backend і frontend частини, а також ручний E2E-сценарій, що охоплює повний процес бронювання столика – від реєстрації користувача до підтвердження бронювання адміністратором. Основна увага приділялась критичним бізнес-сценаріям і модулям з найбільшим ризиком виникнення помилок.

Для серверної частини використано Jest 29.7 – стандартний фреймворк тестування для Nest.JS. Тестами покрито модулі AuthService та ReservationService. Перевіряються сценарії реєстрації користувача, перевірки пароля, створення бронювання, перевірки доступності столика та зміни статусу бронювання. Усі тести використовують mock-об'єкти PrismaService, що дозволяє ізолювати бізнес-логіку від реальної бази даних.

Таблиця 3.1 – Рівні та покриття тестування веб-додатку ресторану

Рівень	Інструмент	Покриті модулі	Кількість тестів
Юніт (backend)	Jest 29.7 + ts-jest	AuthService, ReservationService	6
Юніт (frontend)	Vitest 2.1 + happy-dom	utils.ts, компоненти бронювання	6
Інтеграційний smoke	curl + Python	login, create reservation, list reservations, update status	8
E2E (ручний)	Browser	Реєстрація → login → бронювання → підтвердження адміністратором	1 сценарій
TypeScript	tsc --noEmit	Усі файли frontend/backend	0 помилок
ESLint	eslint	Усі файли проєкту	0 порушень
Production build	next build, nest build	Frontend і backend	Білд успішний

Для frontend використано Vitest 2.1 разом із Testing Library та happy-dom. Тестування охоплює утиліти форматування дати й часу бронювання, функції перевірки статусів, а також React-компоненти форм бронювання. Конфігурація Vitest узгоджена зі структурою Next.js через alias @/ → src/.

Інтеграційне тестування виконувалось через HTTP-запити до запущеного API. Перевірялись сценарії авторизації, створення бронювання, перегляд списку бронювань, зміна статусу адміністратором та надсилення email-повідомлень.

Таблиця 3.2 – Мінімальні системні вимоги

Категорія	Компонент	Мінімум	Рекомендовано
Клієнт (браузер)	Браузер	Chrome 120, Firefox 120, Safari 17	Останні версії
Клієнт	Роздільність екрана	360×640 (мобільний)	1920×1080
Сервер (production)	CPU	1 vCPU	2 vCPU
Сервер	RAM	512 MB	1 GB
Сервер	Диск	200 MB (база + uploads)	5 GB
Сервер	Node.js	22.x LTS	24.x LTS
Розробник	ОС	Windows 10 / Ubuntu 22.04 / macOS 12	Останні версії
Розробник	RAM	4 GB	16 GB
Розробник	Node.js + npm	22.x + 10.x	24.x + 11.x

ВИСНОВКИ

У ході виконання кваліфікаційної роботи бакалавра розроблено повноцінний веб-додаток для онлайн-бронювання столиків у ресторані з використанням сучасних технологій Next.js та Nest.JS. Система забезпечує повний цикл взаємодії користувача із закладом – від перегляду меню та вибору столика до підтвердження бронювання і керування замовленнями адміністратором ресторану.

Проведено аналіз сучасних систем онлайн-бронювання ресторанів. Розглянуто популярні рішення-аналогі: OpenTable, TableCheck, Poster POS, ChoiceQR, Restaron та EasyWeek. Виявлено типові обмеження: висока вартість SaaS-платформ, недостатня адаптація під локальні ресторани, обмеження кастомізації інтерфейсу та залежність від сторонніх сервісів. На основі цього обґрунтовано доцільність створення власного веб-додатку.

Визначено функціональні та нефункціональні вимоги до системи. Сформовано 20 функціональних вимог (бронювання столиків, управління меню, аутентифікація, система ролей, email-сповіщення тощо) і 10 нефункціональних вимог (безпека, продуктивність, масштабованість, адаптивність, доступність). Визначено три ролі користувачів: гість, авторизований користувач та адміністратор ресторану. Усі вимоги формалізовано у відповідних таблицях і UML-діаграмах.

Обґрунтовано вибір технологічного стека. Для клієнтської частини використано Next.js 15, React 19, TypeScript 5.9, Tailwind CSS v4, TanStack Query та React Hook Form із Zod. Для серверної частини обрано Nest.JS 11, Prisma ORM 6, SQLite з можливістю переходу на MySQL, JWT-аутентифікацію та bcrypt для захисту паролів [28]. Кожне рішення порівнювалося з альтернативами, що дозволило сформулювати оптимальний стек для веб-додатку ресторану.

Спроектовано архітектуру системи та базу даних. Побудовано UML-діаграми варіантів використання, компонентної архітектури, ER-діаграму бази даних, діаграми послідовності бронювання столика та аутентифікації

користувача. База даних включає сутності користувачів, столиків, бронювань, меню, категорій страв, замовлень, відгуків та журналу email-повідомлень.

Реалізовано клієнтську та серверну частини системи. Клієнтський застосунок містить сторінки меню, бронювання столиків, особистого кабінету та адміністративної панелі. Серверна частина реалізує REST API для роботи з користувачами, бронюванням, столиками, меню та замовленнями. Інтеграція компонентів продемонстрована через приклади bootstrap-конфігурації NestJS, JWT-аутентифікації, Prisma ORM та axios-інтерсепторів.

Виконано тестування системи. Для backend використано Jest, для frontend – Vitest. Реалізовано автоматизовані юніт-тести та ручний E2E-сценарій, який перевіряє повний процес: реєстрація → авторизація → бронювання столика → підтвердження → керування бронюванням адміністратором. У ході тестування було виявлено та усунуто проблеми з обробкою токенів, валідацією форм і запитами до бази даних.

Підготовлено документацію та налаштовано Docker-розгортання. Створено конфігурації docker-compose, Dockerfile для frontend і backend, а також Swagger-документацію API. Це дозволяє швидко розгорнути систему у середовищі розробки або production.

У результаті розроблено сучасний веб-додаток ресторану, який забезпечує автоматизацію процесу онлайн-бронювання столиків, управління меню та адміністрування замовлень. Система готова до використання у закладах громадського харчування та може бути масштабована для мережі ресторанів.

Перспективами подальшого розвитку є інтеграція онлайн-оплати (LiqPay, Fondy), push-сповіщень, мобільного застосунку на React Native, аналітики відвідуваності, системи лояльності та CI/CD-процесу з GitHub Actions і Docker Registry.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Інженерія програмного забезпечення» / уклад. кафедра інженерії програмного забезпечення ЛНТУ. Луцьк: ЛНТУ, 2025. 56 с.
2. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ: ДП «УкрНДНЦ», 2016. 26с.
3. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ: ДП «УкрНДНЦ», 2016. 16 с.
4. Стандарт вищої освіти України першого (бакалаврського) рівня за спеціальністю 121 «Інженерія програмного забезпечення». Затверджено наказом МОН України від 29.10.2018 № 1166.
5. OpenTable – online restaurant reservation platform. URL: <https://www.opentable.com> (дата звернення: 18.02.2026).
6. Poster POS – автоматизація ресторанів та кафе. URL: <https://joinposter.com/ua> (дата звернення: 20.02.2026).
7. ChoiceQR – QR-меню та автоматизація ресторанів. URL: <https://choiceqr.com/ua> (дата звернення: 24.02.2026).
8. EasyWeek – система онлайн-бронювання. URL: <https://easyweek.com.ua> (дата звернення: 01.03.2026).
9. TableCheck – reservation and guest experience platform. URL: <https://www.tablecheck.com> (дата звернення: 03.03.2026).
10. Statista Research Department. Online Food Delivery and Restaurant Industry Report 2025. URL: <https://www.statista.com/topics/1135/food-delivery/> (дата звернення: 08.03.2026).
11. Nielsen J. 10 Usability Heuristics for User Interface Design. Nielsen Norman Group. URL: <https://www.nngroup.com/articles/ten-usability-heuristics/> (дата звернення: 11.03.2026).

12. BuiltWith Technology Usage Statistics. URL: <https://trends.builtwith.com> (дата звернення: 15.03.2026).
13. Next.js Documentation. App Router and React Server Components. URL: <https://nextjs.org/docs> (дата звернення: 21.03.2026).
14. React Documentation. URL: <https://react.dev> (дата звернення: 25.03.2026).
15. NestJS Documentation. URL: <https://docs.nestjs.com> (дата звернення: 27.03.2026).
16. Prisma Documentation. ORM concepts and migrations. URL: <https://www.prisma.io/docs> (дата звернення: 30.03.2026).
17. TypeScript Handbook. URL: <https://www.typescriptlang.org/docs/> (дата звернення: 02.04.2026).
18. Tailwind CSS v4 Documentation. URL: <https://tailwindcss.com/docs> (дата звернення: 05.04.2026).
19. TanStack Query v5 Documentation. URL: <https://tanstack.com/query/latest/docs/framework/react/overview> (дата звернення: 08.04.2026).
20. React Hook Form Documentation. URL: <https://react-hook-form.com/docs> (дата звернення: 11.04.2026).
21. Zod Documentation. URL: <https://zod.dev> (дата звернення: 13.04.2026).
22. OWASP Application Security Verification Standard 4.0.3. The OWASP Foundation, 2024. URL: <https://owasp.org/ASVS> (дата звернення: 16.04.2026).
23. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. Internet Engineering Task Force. URL: <https://datatracker.ietf.org/doc/html/rfc7519> (дата звернення: 18.04.2026).
24. Nodemailer Documentation. URL: <https://nodemailer.com/about/> (дата звернення: 22.04.2026).
25. Sharp – high-performance image processing for Node.js. URL: <https://sharp.pixelplumbing.com> (дата звернення: 24.04.2026).
26. Docker Documentation. URL: <https://docs.docker.com> (дата звернення: 26.04.2026).

27. Swagger OpenAPI Documentation. URL: <https://swagger.io/docs/> (дата звернення: 28.04.2026).

28. SQLite Documentation. URL: <https://www.sqlite.org/docs.html> (дата звернення: 30.04.2026).

ДОДАТКИ

ДОДАТОК А

Діаграми проєктування веб-додатку «Ресторану»

У додатку А наведено повнорозмірні версії діаграм, які використовуються в основному тексті, та додаткові діаграми, що деталізують архітектуру і потоки даних у системі.

Рисунок А.1 – Повна діаграма варіантів використання (розгорнута версія рис. 2.1)

Рисунок А.2 – Повна архітектура з модулями NestJS (розгорнута версія рис. 2.2)

Рисунок А.3 – Повна ER-діаграма бази даних з усіма атрибутами (розгорнута версія рис. 3.1)

Рисунок А.4 – Діаграма станів замовлення

Рисунок А.5 – Послідовність реєстрації з верифікацією email

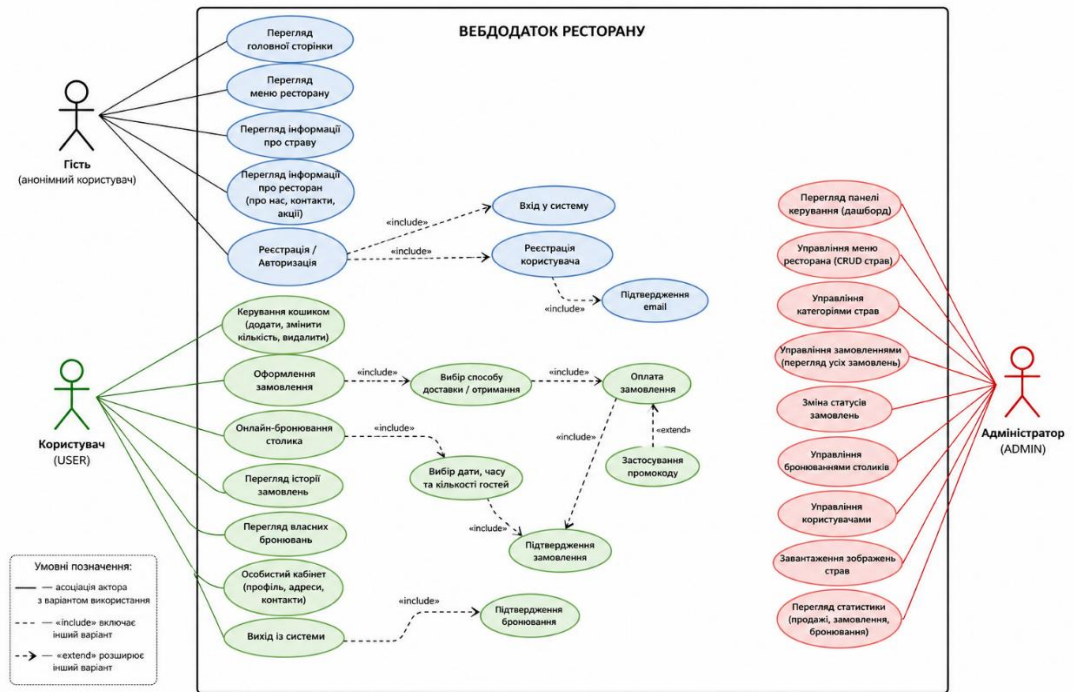


Рисунок А.1 – Повна діаграма варіантів використання веб-додатку «Ресторану»



Рисунок А.2 – Повна архітектура з модулями Nest.JS

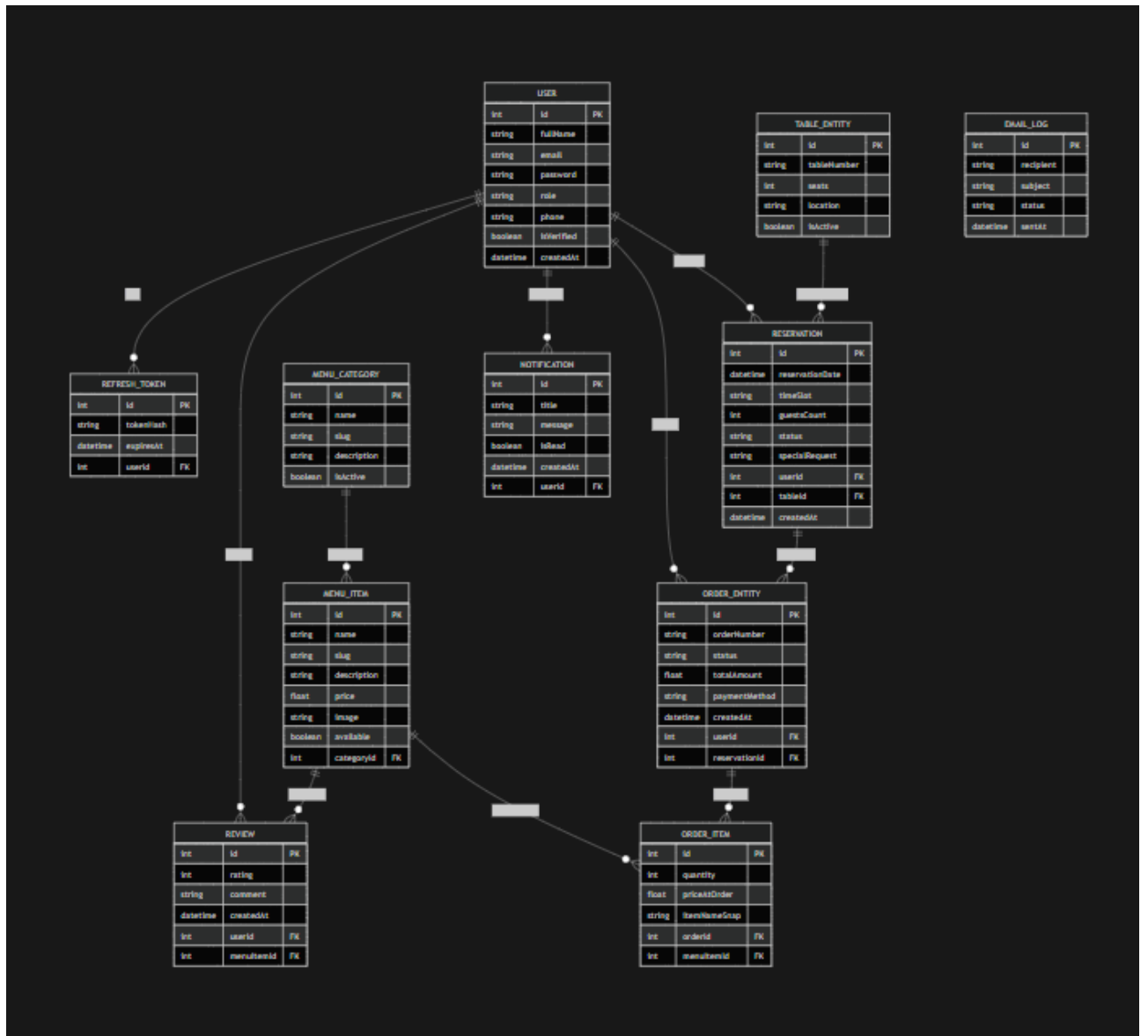


Рисунок А.3 – Повна ER-діаграма бази даних

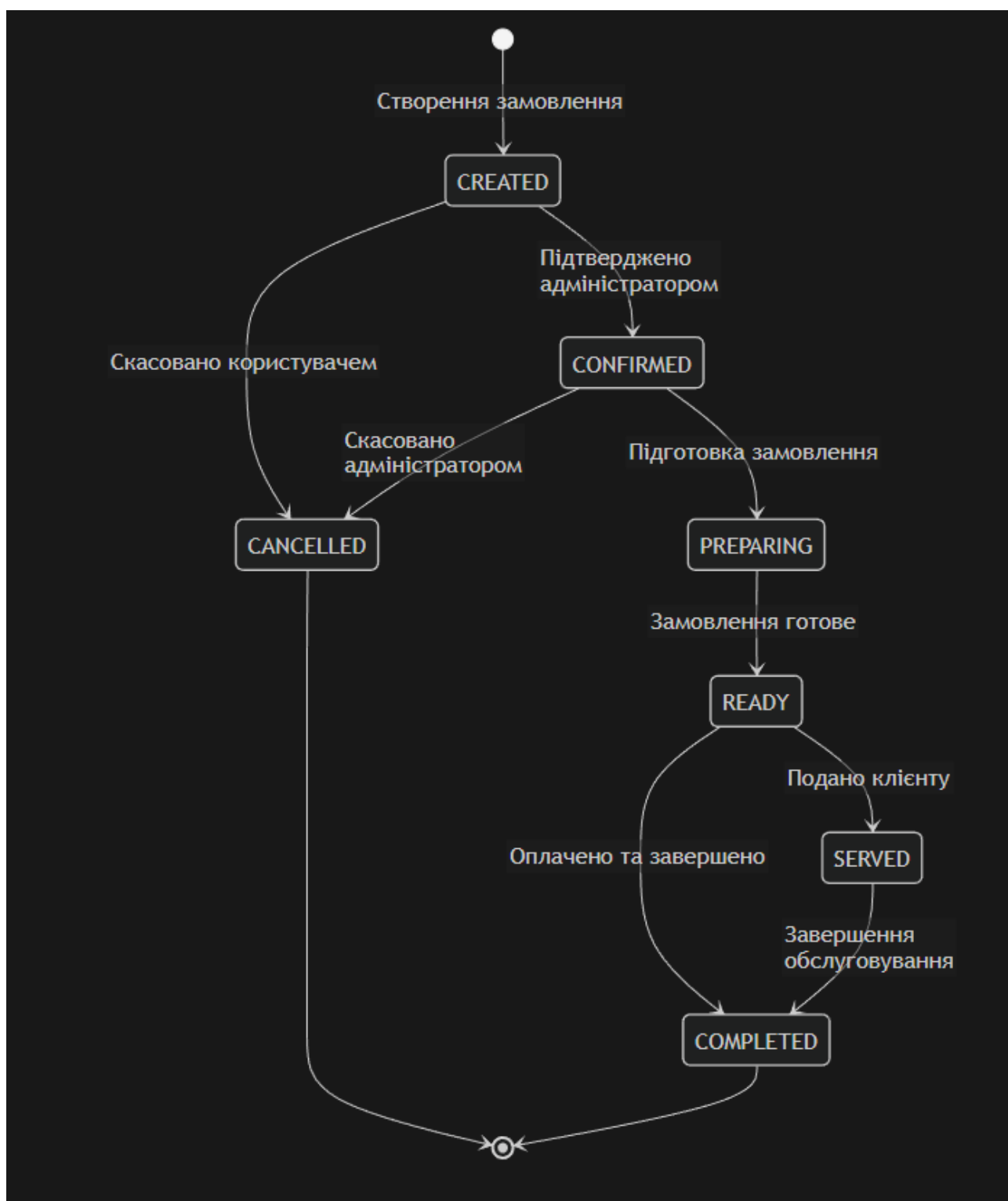


Рисунок А.4 – Діаграма станів замовлення

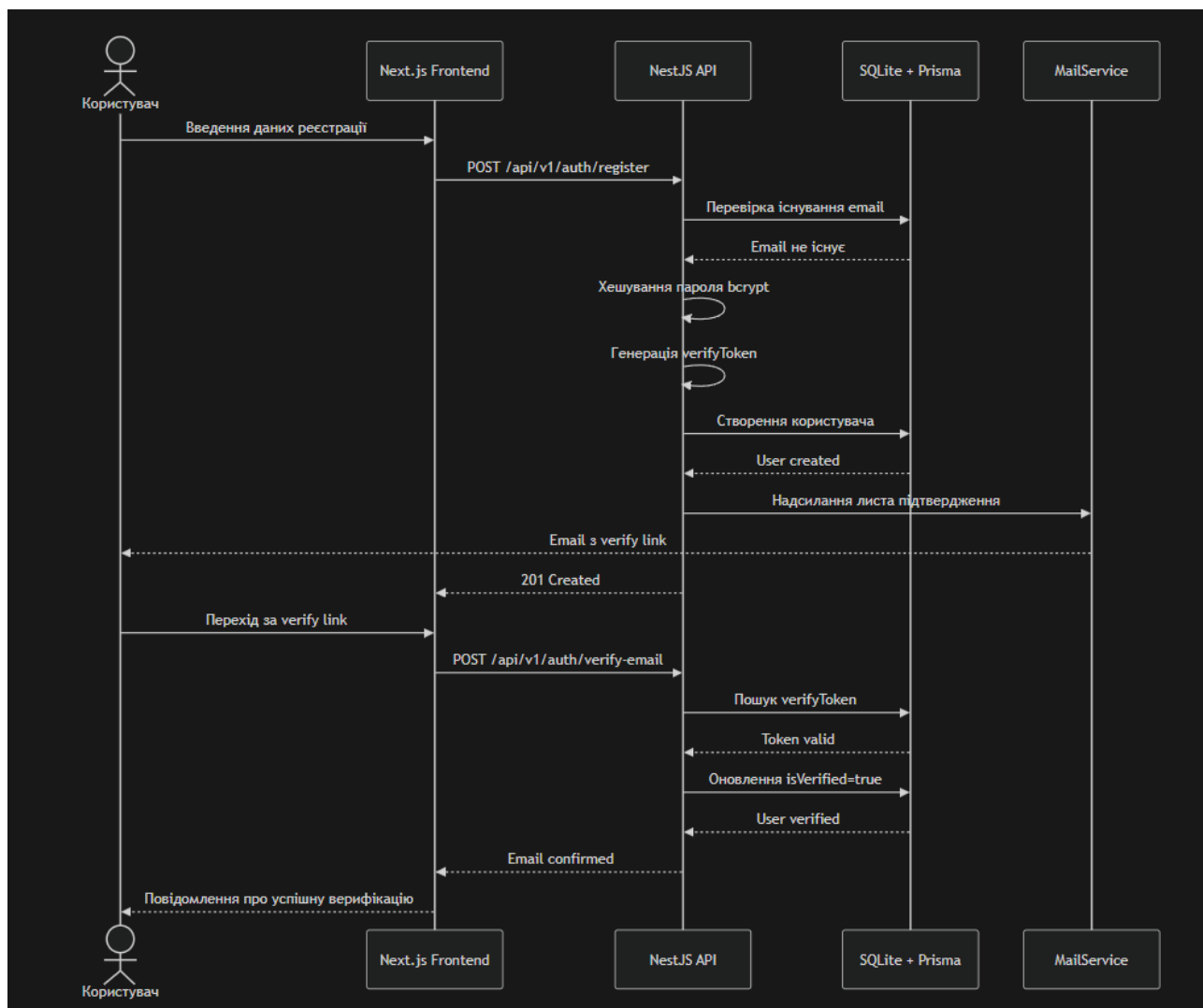


Рисунок А.5 – Послідовність реєстрації з верифікацією email

Додаток Б

Лістинги основних програмних модулів

У додатку Б наведено повні лістинги ключових модулів системи, скорочені фрагменти яких подано в основному тексті.

Лістинг Б.1 – Повний код AuthService (фрагмент із трьома методами)

```
async register(dto: RegisterDto) {
  const email = dto.email.toLowerCase();
  const exists = await this.prisma.user.findUnique({
    where: { email },
  });
  if (exists) {
    throw new ConflictException('Користувач уже існує');
  }
  const passwordHash = await bcrypt.hash(dto.password, 12);
  const verifyToken = randomBytes(32).toString('hex');
  const user = await this.prisma.user.create({
    data: {
      fullName: dto.fullName,
      email,
      phone: dto.phone,
      passwordHash,
      verifyToken,
      role: 'USER',
    },
  });
  await this.mailService.sendVerificationEmail(user.email, verifyToken);
  return {
    id: user.id,
    email: user.email,
    fullName: user.fullName,
  };
}

async login(dto: LoginDto) {
  const user = await this.prisma.user.findUnique({
    where: { email: dto.email.toLowerCase() },
```

```

});
if (!user) {
  throw new UnauthorizedException('Невірний email або пароль');
}
const passwordValid = await bcrypt.compare(
  dto.password,
  user.passwordHash,
);
if (!passwordValid) {
  throw new UnauthorizedException('Невірний email або пароль');
}
const accessToken = await this.jwtService.signAsync({
  sub: user.id,
  email: user.email,
  role: user.role,
});
const refreshToken = await this.jwtService.signAsync(
  { sub: user.id },
  { expiresIn: '7d' },
);
await this.prisma.refreshToken.create({
  data: {
    userId: user.id,
    token: refreshToken,
  },
});
return {
  accessToken,
  refreshToken,
  user,
};
}

async refreshToken(token: string) {
  const storedToken = await this.prisma.refreshToken.findFirst({
    where: { token },
  });
};

```

```

if (!storedToken) {
  throw new UnauthorizedException('Недійсний токен');
}
const payload = await this.jwtService.verifyAsync(token);
const user = await this.prisma.user.findUnique({
  where: { id: payload.sub },
});
if (!user) {
  throw new UnauthorizedException('Користувача не знайдено');
}
const newAccessToken = await this.jwtService.signAsync({
  sub: user.id,
  email: user.email,
  role: user.role,
});
return {
  accessToken: newAccessToken,};}

```

кінець лістингу Б.1

Лістинг Б.2 – Повний код OrdersService.createFromCart (транзакційне створення замовлення)

```

async createReservation(userId: number | null, dto: CreateReservationDto) {
  const table = await this.prisma.table.findUnique({
    where: { id: dto.tableId },
  });
  if (!table || !table.isActive) {
    throw new NotFoundException('Столик недоступний');
  }
  const reservationDate = new Date(dto.reservationDate);
  const exists = await this.prisma.reservation.findFirst({
    where: {
      tableId: dto.tableId,
      reservationDate,
      timeSlot: dto.timeSlot,

```

```

status: {
  in: ['PENDING', 'CONFIRMED'],
},
},
});

if (exists) {
  throw new BadRequestException('Обраний часовий слот уже зайнятий');
}

const reservationNumber = await this.generateReservationNumber();
const reservation = await this.prisma.$transaction(async (tx) => {
  const created = await tx.reservation.create({
    data: {
      reservationNumber,
      userId,
      tableId: dto.tableId,
      guestName: dto.guestName,
      guestPhone: dto.guestPhone,
      guestEmail: dto.guestEmail,
      guestsCount: dto.guestsCount,
      reservationDate,
      timeSlot: dto.timeSlot,
      specialRequest: dto.specialRequest,
      status: 'PENDING',
      statusHistory: {
        create: {
          toStatus: 'PENDING',
          note: 'Бронювання створено',
        },
      },
    },
    include: {
      table: true,
      statusHistory: true,
    },
  });
});

```

```
    },  
  });  
  await tx.table.update({  
    where: { id: dto.tableId },  
    data: {  
      reservationCount: {  
        increment: 1,  
      },  
    },  
  });  
  return created;  
});  
  
// Email надсилається поза транзакцією  
await this.mailService.sendReservationConfirmation(reservation);  
return reservation;}
```

кінець лістингу Б.2

Додаток В

Додаткові скріншоти інтерфейсу

У додатку В наведено додаткові скріншоти інтерфейсу, які не увійшли в основний текст роботи через обмеження обсягу.

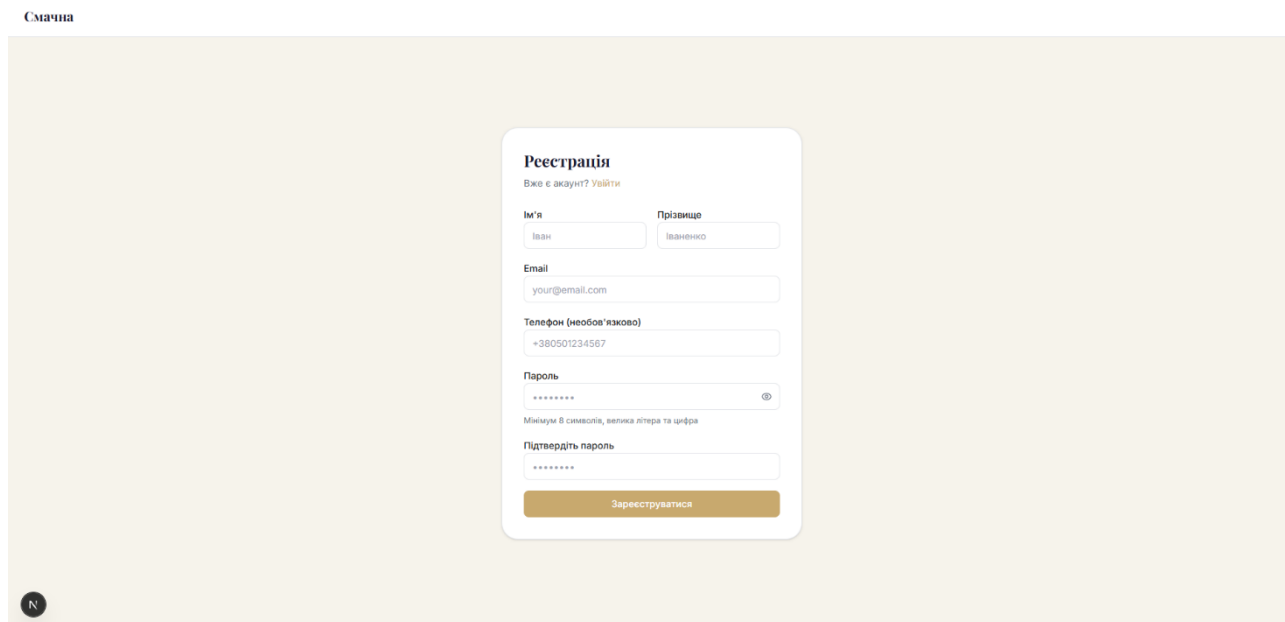


Рисунок В.1 – Сторінка реєстрації з валідацією форми

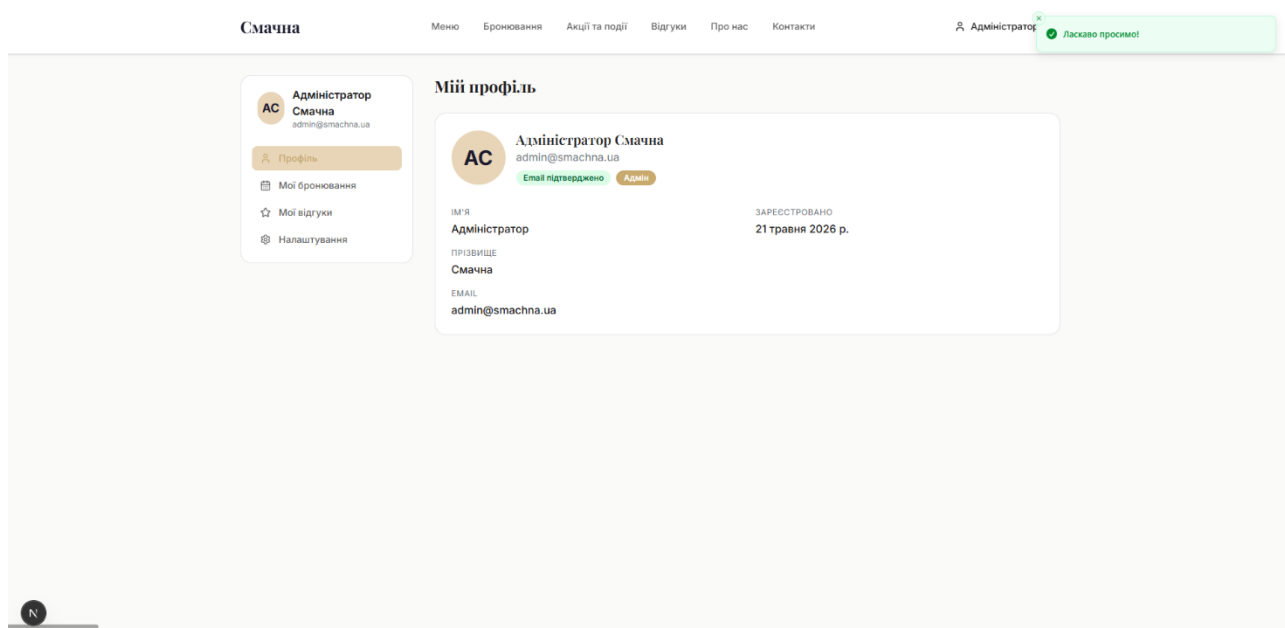


Рисунок В.2 – Особистий кабінет користувача

Смачна

Адмін-панель

Дашборд

Бронювання

Меню

Столики

Користувачі

Відгуки

Налаштування

Адміністратор Смач...

admin@smachna.ua

Зійти

Бронювання

Всі статуси

КОД	ГІСТЬ	СТОЛИК	ДАТА/ЧАС	ГОСТЕЙ	СТАТУС	Дії
REZ-009D-005	Марія Шевченко +380503333333	Столик №6	11 червня 2026 р. 18:00 – 20:00	4	Очікує	Змінити...
REZ-009D-004	Богдан Коваль +38050184141	Столик №1	21 травня 2026 р. 12:00 – 14:00	2	Завершено	
REZ-009D-003	Яна Ковальчук +380501313131	Тераса №3	4 червня 2026 р. 19:00 – 21:00	6	Підтверджено	Змінити...
REZ-009D-002	Андрій Лисенко +380501212121	Тераса №2	16 травня 2026 р. 19:00 – 21:00	4	Завершено	
REZ-009D-001	Іван Петренко +380502222222	Столик №4	25 травня 2026 р. 13:00 – 15:00	2	Очікує	Змінити...
REZ-009D-009	Людмила Сидоренко +380501010101	Тераса №1	18 травня 2026 р. 12:00 – 14:00	2	Не з'явився	
REZ-009D-009	Василь Ткаченко +380509999999	Приватна кімната	31 травня 2026 р. 18:00 – 20:00	12	Очікує	Змінити...
REZ-009D-008	Олена Коваленко +380501111111	Столик №7	28 травня 2026 р. 20:00 – 22:00	4	Підтверджено	Змінити...
REZ-009D-007	Оксана Мельник +380507777777	Столик №3	19 травня 2026 р. 19:00 – 21:00	2	Скасовано	
REZ-009D-006	Павло Краченко +380506666666	Столик №6	20 травня 2026 р. 13:00 – 15:00	3	Завершено	
REZ-009D-005	Наталія Гриценко +380505555555	VIP-1	26 травня 2026 р. 18:00 – 20:00	8	Підтверджено	Змінити...
REZ-009D-004	Тарас Бондаренко +380504444444	Столик №2	21 травня 2026 р. 19:00 – 21:00	2	Розміщено	Змінити...

Рисунок В.3 – Адмінська таблиця бронювання столиків

Смачна

Адмін-панель

Дашборд

Бронювання

Меню

Столики

Користувачі

Відгуки

Налаштування

Адміністратор Смач...

admin@smachna.ua

Зійти

Відгуки

На модераторії

★★★★☆

Іван Петренко

21 травня 2026 р.

Замовив стейк Well Done — отримав суху підшву. Соус терпів занадто солодий. Буду чекати покращень.

Опублікувати

Відповісти

Опубліковані (10)

★★★★★

Іван Петренко

21 травня 2026 р.

Опубліковано

Святкували весілля — ресторан перевершив усі очікування. Меню для банкету, обслуговування, декор — все ідеально. Сердечно дякуємо!

Зняти

★★★★☆

Олена Коваленко

21 травня 2026 р.

Опубліковано

Прекрасна атмосфера, гарний дизайн. Цезар з куркою смачний, порція гарна. Офіціант порадив вино — відмінний вибір.

Зняти

★★★★★

Марія Шевченко

21 травня 2026 р.

Опубліковано

Панна-котта — найкраща, яку куштувала. Ягідний соус яскравий та свіжий. Обожнюю десертне меню цього ресторану!

Зняти

★★★★★

Олена Коваленко

21 травня 2026 р.

Опубліковано

Найкращий ресторан Луцька! Ходимо вже 3 роки поспіль. Якість не падає, асортимент оновлюється. Дякуємо команді!

Зняти

★★★★☆

Марія Шевченко

21 травня 2026 р.

Опубліковано

Їжа дуже смачна, особливо дегустаційний сет. Але паркування поруч ускладнене. В цілому рекомендую!

Зняти

★★★★★

Іван Петренко

21 травня 2026 р.

Опубліковано

Шоколадний фондан — просто неймовірний! Тепле шоколадне серце всередині, морозиво ванільне, полуниця — ідеальна комбінація.

Зняти

Рисунок В.4 – Адмінська модерація відгуків

Додаток Г

Матеріали тестування та запуску системи

У додатку Г наведено команди для встановлення і запуску системи, а також приклади звіту автоматизованих тестів.

Таблиця Г.1 – Команди для запуску та перевірки веб-додатку «Ресторану»

Команда	Призначення	Очікуваний результат
npm install (y backend/ i frontend/)	Встановлення залежностей проєкту	Створення папок node_modules в обох частинах системи
npm run db:generate (y backend/)	Генерація Prisma Client	Успішне створення Prisma Client без помилок
npm run db:migrate (y backend/)	Створення SQLite бази даних і застосування міграцій	Створення файлу backend/prisma/dev.db
npm run db:seed (y backend/)	Заповнення БД тестовими даними	Додавання користувачів, категорій, товарів, замовлень і відгуків
npm run start:dev (y backend/)	Запуск серверної частини NestJS	Backend доступний на http://localhost:3001
npm run dev (y frontend/)	Запуск клієнтської частини Next.js	Frontend доступний на http://localhost:3000