

Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Факультет комп'ютерних наук

Рекомендовано до захисту:
протокол засідання кафедри
№ ____ від ____ . ____ .2024 р.
Завідувач кафедри ММтаАД
_____ Струков В.М. (_____
підпис) (прізвище та ініціали)

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему: «Розробка програмного забезпечення для платформи з реалізації
надлишкових харчових продуктів на українському ринку»

Захищено на засіданні ЕК № _____
протокол № ____ від ____ . ____ .2019 р.
Оцінка ____ / _____
Голова ЕК

(підпис) (прізвище та ініціали)

Виконав:
студент 4 курсу, групи ЗКС-41
Спеціальності:
122 Комп'ютерні науки
(шифр і назва спеціальності підготовки)
Маїло Володимир Васильович

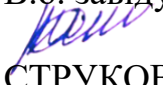
(прізвище, ім'я та по батькові, підпис)
Керівник к.т.н., доцент Струков В.М.
(ступень, звання, прізвище та ініціали, підпис)

Рецензент к.т.н. Олешко О. І.
(ступень, звання, прізвище та ініціали, підпис)

Харків – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук
та штучного інтелекту
Кафедра математичного моделювання та аналізу даних
Рівень вищої освіти (освітньо-кваліфікаційний рівень): бакалавр
Галузь знань: F Інформаційні технології
Спеціальність: F3 Комп'ютерні науки
Освітня програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
 Володимир
СТРУКОВ
« 03 » лютого 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ (ПРОЕКТ)

Маїло Володимир Васильович
(прізвище, ім'я, по батькові студента)

1. Тема роботи: Розробка програмного забезпечення для платформи з реалізації надлишкових харчових продуктів на українському ринку,

керівник роботи: Струков Володимир Михайлович, в.о. завідувача кафедри математичного моделювання та аналізу даних, кандидат технічних наук, доцент,

затверджені наказом по університету від “ 16 ” квітня 2025 року №4101-5/962

2. Строк подання студентом роботи 02 червня 2025 р.

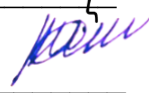
3. Перелік питань, які потрібно розробити: Аналіз предметної області, обґрунтування актуальності задачі. Вивчення літератури та документації. Визначення вимог до компонентів програмного забезпечення. Вибір технологій та програмних засобів реалізації. Проєктування та програмна реалізація додатку. Проведення експериментального дослідження продуктивності додатку. Аналіз результатів та їх узагальнення. Написання пояснювальної записки, створення презентації та доповіді.

4. План роботи

№ з/п	Назви етапів роботи
1	Аналіз предметної області, обґрунтування актуальності задачі.
2	Вивчення літератури та документації.
3	Визначення вимог до компонентів програмного забезпечення.
4	Вибір технологій та програмних засобів реалізації.
5	Розробка структури та компонентів додатку.
6	Програмна реалізація додатку.
7	Проведення експериментального дослідження продуктивності додатку.
8	Аналіз результатів та їх узагальнення.
9	Написання пояснювальної записки, створення презентації та доповіді.

5. Дата видачі завдання 03 лютого 2025 р.

Студент

Керівник роботи

В.В. Майло

В.М. Струков_

АНОТАЦІЯ

Пояснювальна записка до дипломної роботи містить 75 сторінок, 18 рисунків, 5 таблиць.

Метою роботи є розробка програмного забезпечення для платформи з реалізації надлишкових харчових продуктів на українському ринку, що дозволить ефективно вирішити проблему утилізації продуктів харчування та сприятиме екологічній стійкості й економічній вигоді для закладів харчування та споживачів. У ході дослідження проаналізовано сучасні платформи у сфері food-waste management, досліджено технології побудови веб-додатків, можливості використання геоінформаційних систем та систем асинхронного обміну повідомленнями (Kafka).

Об'єкт дослідження – процес взаємодії користувачів платформи для реалізації надлишкових харчових продуктів.

Предмет дослідження – сучасні технології, підходи та інструменти розробки веб-застосунків, орієнтованих на food-waste management та взаємодію закладів зі споживачами.

Основними методами досліджень є аналіз предметної області для визначення функціональних та нефункціональних вимог до платформи, об'єктно-орієнтоване та компонентне проєктування програмного забезпечення, а також експериментальні методи тестування продуктивності та функціональних можливостей платформи.

У роботі проаналізовано особливості інтеграції з Google Maps API для реалізації пошуку найближчих закладів харчування, досліджено технологію Kafka для забезпечення асинхронної взаємодії сервісів, розроблено адаптивний та сучасний користувацький інтерфейс на основі технологій React, Next.js, Tailwind CSS та Framer Motion. Реалізовано систему резервування «surprise-

box», яка дозволяє споживачам замовляти надлишкові продукти за зниженою ціною.

Результати роботи можуть бути використані для впровадження у закладах харчування з метою скорочення харчових відходів, а також у подальших наукових дослідженнях з питань сталого розвитку та технологій у сфері food-tech.

Ключові слова: надлишкові харчові продукти, food-waste management, веб-платформа, React, Next.js, Tailwind CSS, PostGIS, Kafka, surprise-box, Google Maps, екологічна ефективність, резервування.

ABSTRACT

The explanatory note for the diploma thesis contains 75 pages, 18 figures, and 5 tables.

The goal of the work is to develop software for a platform aimed at distributing surplus food products on the Ukrainian market, which will effectively address the issue of food disposal and promote environmental sustainability and economic benefits for food establishments and consumers.

During the research, current platforms in the field of food-waste management were analyzed, as well as technologies for building web applications, the possibilities of using geographic information systems, and asynchronous messaging systems (Kafka).

Object of the study – the process of interaction between users on the platform for distributing surplus food products.

Subject of the study – modern technologies, approaches, and tools for developing web applications focused on food-waste management and the interaction between establishments and consumers.

The main research methods include analysis of the subject area to define functional and non-functional requirements for the platform, object-oriented and component-based software design, as well as experimental methods for testing the platform's performance and functionality.

The work analyzes the integration features with the Google Maps API for implementing a search for nearby food establishments, explores Kafka technology to ensure asynchronous service interactions, and develops an adaptive and modern user interface based on React, Next.js, Tailwind CSS, and Framer Motion.

A “surprise box” reservation system was implemented, allowing consumers to order surplus products at discounted prices.

The results of the work can be applied in food establishments to reduce food waste and can also be used in further scientific research on sustainable development and technologies in the food-tech sector.

Keywords: surplus food products, food-waste management, web platform, React, Next.js, Tailwind CSS, PostGIS, Kafka, surprise-box, Google Maps, environmental efficiency, reservation.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ	8
ВСТУП.....	10
1. АНАЛІЗ НАУКОВИХ ДЖЕРЕЛ ТА СУЧАСНИХ РІШЕНЬ У СФЕРІ БОРОТЬБИ З ХАРЧОВИМИ ВТРАТАМИ ТА ВІДХОДАМИ	13
1.2 Огляд існуючих рішень та платформ.....	14
1.2.2. Виявлені ніші та невирішені проблеми	17
1.2.3. Місце та конкурентні переваги платформи FoodWise	18
1.3 Призначення та концепція запропонованого ПЗ.....	18
1.4 Основні вимоги до платформи	20
<i>ВИСНОВКИ ДО РОЗДІЛУ 1.....</i>	24
2. ПОСТАНОВКА ЗАВДАННЯ ТА ВИМОГ ДО СИСТЕМИ ДЛЯ РЕАЛІЗАЦІЇ НАДЛИШКОВИХ ХАРЧОВИХ ПРОДУКТІВ	25
2.1 Вибір архітектурного підходу: моноліт чи мікросервіси.....	25
2.1.1 Монолітна модель: що це означає для FoodWise	25
2.1.2 Потенціал і виклики мікросервісів.....	25
2.1.3 Чому обрано моноліт + подієвий шар Kafka.....	26
2.2 Вибір технологічного стеку	27
2.2.1 Вибір технологій для клієнтської частини	27
2.2.2 Вибір технологій для серверної частини та обробки подій.....	28
2.3 Загальна структура та компоненти системи	29
2.3.1 Сховище даних.....	30
2.3.2 Асинхронна обробка подій з використанням Apache Kafka.....	32
2.4 Проектування бази даних і геоінформаційного модуля	33

2.4.1 Ключові сутності та зв'язки.....	34
2.4.2 Геодані й PostGIS.....	35
2.4.3 Забезпечення продуктивності, узгодженості даних та підтримка бізнес-логіки на рівні БД	36
2.5 Моделювання основних сценаріїв використання.....	37
2.5.1 Сценарій – споживач шукає й резервує бокс	37
2.5.2 Перевірка цілісності рішень	38
ВИСНОВКИ ДО РОЗДІЛУ 2	40
3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	42
3.1 Розробка фронтенду	42
3.1.1 Компонентна ієрархія.....	44
3.2 Реалізація бекенду та API	59
3.2.1 Глибинна будова модуля.....	59
3.2.2 Безпека та доступ.....	60
3.2.3 API-контракт	61
3.3 Інтеграція з базою даних (PostgreSQL/PostGIS).....	61
3.3.1 Як структура бази трансформується у модельний шар	63
3.4 Тестування та перевірка працездатності системи	65
ВИСНОВКИ ДО РОЗДІЛУ 3	67
ВИСНОВКИ	69
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	71
ДОДАТОК А.....	73
ДОДАТОК Б.....	74

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

API — Application Programming Interface, програмний інтерфейс для взаємодії між компонентами системи.

CI/CD — Continuous Integration / Continuous Deployment, безперервна інтеграція та розгортання.

CRUD — Create, Read, Update, Delete, основні операції з даними.

DB — Database, база даних.

DTO — Data Transfer Object, об'єкт передачі даних між шарами програми.

GCP — Google Cloud Platform, хмарна платформа Google.

HTTP — Hypertext Transfer Protocol, протокол передачі гіпертексту.

Kafka — система розподіленої публікації-підписки для потокової обробки даних.

UI — User Interface, інтерфейс користувача.

UX — User Experience, досвід користувача.

UUID — Universally Unique Identifier, універсально унікальний ідентифікатор.

JSON — JavaScript Object Notation, формат обміну даними.

REST — Representational State Transfer, архітектурний стиль для побудови веб-сервісів.

SQL — Structured Query Language, мова структурованих запитів для роботи з базами даних.

Food-waste management — управління харчовими відходами, комплекс заходів зі зменшення обсягів харчових продуктів, що утилізуються.

Surprise-box — концепція продажу надлишкових продуктів закладами харчування за зниженою ціною у вигляді «сюрприз-боксів».

Резервування — попереднє замовлення товарів або послуг із гарантією їх отримання у визначений час.

Модуль — самостійна частина програмного забезпечення, яка виконує певну функцію і може бути інтегрована в більшу систему.

Монолітна архітектура — архітектурний стиль програмного забезпечення, коли вся система працює як єдиний застосунок без розділення на незалежні сервіси.

Геолокація — визначення місцезнаходження користувача або об'єкта за допомогою GPS, Wi-Fi або мобільної мережі.

Компонент (у React) — окремий елемент інтерфейсу користувача, який має власну логіку, стан і рендеринг.

ВСТУП

У сучасному світі проблема **харчових відходів** набула значних масштабів: за даними ФАО, близько третини вироблених продуктів харчування щорічно втрачається або утилізується. В Україні ця тенденція посилюється економічними викликами, нерівномірністю попиту й пропозиції та відсутністю ефективних цифрових механізмів перерозподілу надлишкових продуктів. Одночасно зростає суспільний запит на «зелений» **стиль споживання** і sustainable-ініціативи, а локальний ринок IT-рішень у цій сфері залишається малозаповненим.

Цифрові платформи мають потенціал стати ключовим інструментом для мінімізації харчових втрат: вони здатні швидко поєднувати заклади-постачальники із кінцевими споживачами, надаючи актуальну інформацію про доступні надлишкові продукти та можливість їх оперативного резервування. Проте існуючі глобальні сервіси (Too Good To Go, Karma тощо) майже не представлені в Україні, а локальні рішення обмежуються вузьким функціоналом. Це створює **нішу** для розробки платформи, адаптованої до українського ринку та вимог користувачів.

Мета роботи

Розробити **веб-платформу FoodWise** для реалізації надлишкових харчових продуктів, яка забезпечує зручну геолокаційну взаємодію між закладами харчування та споживачами й сприяє скороченню харчових відходів в Україні.

Завдання для досягнення мети

1. Проаналізувати наукові джерела й існуючі рішення у сфері food-waste-management на світовому та українському ринку.
2. Сформулювати функціональні та нефункціональні вимоги до платформи.

3. Обґрунтувати вибір **монолітної архітектури** з подієвим шаром Kafka та підібрати оптимальний технологічний стек.
4. Реалізувати серверну частину (Spring Boot + PostgreSQL/PostGIS + Kafka) та клієнтську частину (Next.js + React + Tailwind CSS).
5. Інтегрувати **Google Maps** для пошуку боксів за місцем розташування та забезпечити механізм резервування «surprise-box».
6. Провести функціональне й навантажувальне тестування, оцінити UX та продуктивність системи.
7. Підготувати економічне обґрунтування та заходи з охорони праці при впровадженні платформи.

Актуальність

Розв’язання проблеми харчових відходів є нагальною потребою у світі й особливо в Україні під час економічних коливань. Відсутність дієвих локальних ІТ-інструментів зумовлює необхідність розробки нової платформи, здатної **оперативно перерозподіляти надлишкові продукти** та підтримувати ініціативи сталого розвитку.

Новизна роботи

- інтеграція геопозиційної карти та кластеризація пропозицій у реальному часі;
- подієвий підхід із використанням **Kafka** усередині моноліту для сповіщень і масштабування;
- орієнтація виключно на **український ринок** і підтримка локальної екосистеми закладів;
- сучасний, мобільно-орієнтований UX із плавними анімаціями (**Framer Motion**) і швидкою взаємодією.

Практичний результат

У результаті буде створено **працюючий прототип** платформи, який включатиме:

- модулі реєстрації й авторизації користувачів та закладів;
- карту з відображенням та фільтрацією доступних «surprise-box»;
- функціонал резервування, кошику та чек-аут.

Таким чином, робота має на меті створити ефективний інструмент для зменшення харчових втрат, що поєднує сучасні веб-технології та актуальні потреби українського ринку.

1. АНАЛІЗ НАУКОВИХ ДЖЕРЕЛ ТА СУЧАСНИХ РІШЕНЬ У СФЕРІ БОРОТЬБИ З ХАРЧОВИМИ ВТРАТАМИ ТА ВІДХОДАМИ

1.1 Проблема надлишкових харчових продуктів та актуальність

За оцінками Продовольчої та сільськогосподарської організації ООН (FAO) щороку у світі втрачається або викидається понад 1,3 млрд т продуктів — це майже $\frac{1}{3}$ від усього виробленого харчового обсягу. Фінансові збитки перевищують 900 млрд USD на рік, а у перерахунку на довкілля — це близько 8 % усіх глобальних викидів парникових газів[1].

В Україні, за даними Міндовкілля та досліджень USAID (2022 р.), утворюється понад 7–9 млн т харчових відходів щорічно, що дорівнює 220–250 кг на одну особу. Значна частка цих втрат припадає на сектор громадського харчування та роздрібної торгівлі, де надлишкові продукти утворюються внаслідок:

- Коливань попиту — складно передбачити кількість відвідувачів, сезонність, зміну споживчих смаків;
- Логістичних затримок та невідповідності стандартам (нерівномірні овочі, деформована упаковка), які не впливають на якість, але унеможливають продаж за повною ціною;
- Скороченого терміну придатності у готових страв і випічці;
- Відсутності цифрових інструментів для оперативного інформування потенційних покупців про надлишкові продукти.

Наслідки надмірного формування харчових відходів мають три виміри:

1. Екологічний — органічні відходи на полігонах утворюють метан, збільшують навантаження на ґрунти й водні ресурси;
2. Соціальний — замість потрапити до вразливих верств населення якісні продукти списуються;

3. Економічний — заклади несуть прямі втрати від утилізації, споживачі втрачають можливість купувати їжу за зниженою ціною, держава витрачає кошти на перероблення відходів.

Розвинені країни активно впроваджують цифрові платформи (Too Good To Go, Karma, OLIO), що з'єднують бізнеси та споживачів: заклади розміщують «surprise-box» з надлишками страв за зниженими цінами, а користувачі бронюють їх онлайн та забирають у зручний час[2]. В Україні подібні сервіси представлені поодинокими стартапами й не мають масштабного покриття; більшість закладів покладається на списання або ситуативні благодійні акції.

З огляду на це, створення платформи FoodWise, що:

- автоматизує процес публікації надлишкових продуктів;
- забезпечує геолокаційний пошук пропозицій на інтерактивній карті;
- надає механізм бронювання та оплати у кілька кліків;
- інтегрується з аналітикою для закладів (статистика списань, CO₂-footprint),

є актуальним та соціально значущим завданням. Такий інструмент допоможе скоротити обсяги харчових відходів, підвищити продовольчу безпеку, зменшити екологічне навантаження й стимулювати культуру відповідального споживання на українському ринку.

1.2 Огляд існуючих рішень та платформ

Глобальний ринок food-waste tech демонструє стрімке зростання: за даними дослідницьких агентств, обсяг сегмента цифрових сервісів для скорочення харчових відходів перевищив 1 млрд USD і щороку зростає на 14–18%. Нижче наведено узагальнений аналіз найпоширеніших рішень, їхніх функціональних можливостей та ступеня присутності на українському ринку[3].

Таблиця 1.2.1 - Міжнародні платформи

Платформа	Країна запуску / охоплення	Ключовий функціонал	Переваги	Недоліки / обмеження
Too Good To Go	Данія → 17+ країн Європи та США	«Surprise-box», тайм-слот бронювання, мобільний додаток	Велика база партнерів (≈180 тис.), гейміфікація, вбудовані есо-статистики	Висока комісія; відсутність локалізації під українське законодавство; офіційно не працює в Україні
Karma	Швеція → 10 країн	Продаж конкретних страв зі знижкою, push-сповіщення, аналітика для B2B	Прозора ціна на рівні порції; партнерські API для супермаркетів	Низька географічна експансія поза Скандинавію; немає інтеграції з картковими терміналами українських банків

Продовження таблиці 1.2.1

OLIO	Велика Британія → 50 країн (переважно C2C)	Обмін продуктами між сусідами, волонтерська мережа «Food Waste Heroes»	Сильна community-складова, безкоштовна модель	Менший контроль якості; слабка B2B-монетизація; низький рівень покриття поза мегаполісами
Flashfood	Канада / США	Знижки на товари з коротким терміном у супермаркетах, інтеграція з POS	Безшовна робота в роздрібних мережах, автоматичні знижки	Потребує глибокого інтегрування в ERP-системи ритейлерів; поки що немає європейської локалізації

1.2.1. Локальні ініціативи та українські спроби

Серед українських локальних ініціатив, спрямованих на скорочення харчових відходів, помітне місце займає застосунок *Expienza* від Monobank. Цей мобільний додаток дозволяє користувачам знаходити товари зі знижками, термін придатності яких наближається до завершення. Серед основних переваг платформи можна виділити інтеграцію з банківською системою Monobank та push-сповіщення про актуальні пропозиції. Однак застосунок має низку обмежень, серед яких – обмежене покриття партнерських мереж, які в основному представлені великими супермаркетами у великих містах України, а також відсутність веб-версії, що суттєво обмежує потенційну аудиторію.

Ще однією помітною ініціативою є волонтерська платформа Foodsharing-Ukraine. Вона зосереджується переважно на координуванні передачі харчових продуктів благодійним організаціям та окремим споживачам у потребі. Такий підхід має значну соціальну цінність, однак платформа не містить комерційної B2C-складової та не має відповідної масштабованої ІТ-інфраструктури, що значно знижує її комерційний потенціал і здатність до сталого розвитку[4].

Також варто згадати регіональні ініціативи, які часто організовуються самими ресторанами. Вони полягають у наданні знижок у фірмових мобільних застосунках закладів за кілька годин або за годину до закриття. Хоча такі акції можуть бути корисними для локальних споживачів, вони зазвичай не консолідуються на єдиній загальнодоступній карті та не мають стандартних API, що ускладнює інтеграцію зі сторонніми сервісами та розширення аудиторії.

Таким чином, можна зробити висновок, що, незважаючи на наявність окремих корисних ініціатив, на українському ринку відсутня комплексна та універсальна платформа, яка б об'єднувала переваги згаданих підходів і була доступною широкому колу користувачів з різними потребами. Саме цю нішу покликана зайняти розроблювана платформа.

1.2.2. Виявлені ніші та невирішені проблеми

1. Фрагментованість ринку – заклади працюють з окремими волонтерами або чатами; немає єдиного інтерфейсу для публікації пропозицій.
2. Обмежені платіжні механізми – більшість рішень не інтегровані з популярними в Україні сервісами LiqPay, Apple Pay, Google Pay; оплата зазвичай відбувається готівкою.
3. Відсутність гео-фільтрів у реальному часі – користувачі не бачать на карті, де зараз доступний «бокс» із надлишковою їжею.

4. Недостатня аналітика для закладів – поточні ініціативи не надають бізнесу звітів щодо зекономлених коштів, зменшених CO₂-викидів, відсотка проданих надлишків тощо.

5. Низька автоматизація процесу – резервування відбувається вручну (чат, телефон), що спричиняє дублювання бронювань і людські помилки.

6. Мобільно-веб-розрив – деякі сервіси існують лише у вигляді мобільних застосунків або, навпаки, веб-сайтів, що обмежує аудиторію.

1.2.3. Місце та конкурентні переваги платформи FoodWise

- Єдина веб-платформа + PWA – доступ із браузера без примусової установки застосунка.
- Інтерактивна карта (Google Maps / Leaflet + PostGIS) – точний пошук пропозицій у радіусі.
- Повноцінний «surprise-box» B2C-механізм з бронюванням, оплатою карткою/Apple Pay/Google Pay та QR-кодом видачі.
- Подієва обробка через Kafka – push-повідомлення і можливість підключати сторонні сервіси (наприклад, благодійні організації).
- Аналітична панель для бізнесу – статистика економії, проданих боксів, екологічний ефект.
- Локалізація й маркетинг під українське законодавство – підтримка гривні, фіскальних вимог та української/англійської мов.

1.3 Призначення та концепція запропонованого ПЗ

Мета платформи FoodWise — створити єдиний веб-сервіс, який дозволить закладам громадського харчування та продуктовим магазинам швидко реалізовувати надлишкові харчові продукти, а споживачам — вигідно купувати

якісну їжу, тим самим зменшуючи обсяги харчових відходів і супутні екологічні втрати[5].

Цільова аудиторія

1. B2B-сегмент:

- ресторани, пекарні, кав'ярні, супермаркети та фермерські лавки, у яких щодня залишаються продукти з обмеженим терміном реалізації;
- благодійні фонди й ініціативи, що збирають їжу для соціально вразливих груп.

2. B2C-сегмент:

- екосвідомі споживачі, студенти й родини, зацікавлені у знижках та мінімізації власного екологічного «сліду»;
- волонтери, які допомагають швидко перерозподіляти їжу (функція пожертви).

Концепція роботи платформи

1. Публікація «surprise-box»

Заклад створює у веб-кабінеті пропозицію («бокс») із залишками страв або товарів, вказуючи знижку, приблизний склад, час видачі та кількість одиниць.

2. Інтерактивна карта

На головній сторінці користувач бачить інтерактивну карту (Google Maps / Leaflet), де кожен бокс відображається маркером. Завдяки PostGIS-запитам платформа пропонує фільтр за радіусом, категорією кухні, рівнем знижки тощо.

3. Резервування та оплата

Користувач натискає на маркер, переглядає деталі бокса та бронює його одним кліком. Оплата здійснюється карткою, Apple Pay, Google Pay або криптовалютою; після успішної транзакції користувач отримує QR-код для швидкої видачі замовлення на місці.

4. Видача та підтвердження

Працівник закладу сканує QR-код або вводить PIN-код, після чого статус бокса змінюється на «видано». Подія передається в Kafka, формуючи аналітику та оновлюючи запаси в реальному часі.

5. Аналітика й есо-статистика

У кабінеті закладу відображаються звіти: кількість врятованих боксів, оцінка зекономлених коштів і знижених CO₂-викидів. Це стимулює бізнес прозоро декларувати свій «зелений» внесок.

Ключові функції системи

- Інтерактивна карта з кластеризацією маркерів, фільтрами та прокладанням маршруту до закладу.
- Система резервування із захистом від дублювання, таймером невиконаного бронювання та push-сповіщеннями.
- Багатоканальна оплата (картки, P2P-сервіси, Google Pay / Apple Pay).
- Подієва шина Kafka для обробки подій user.created, box.created, order.paid тощо, що забезпечує масштабованість і можливість додати мікросервіси у майбутньому.
- Адміністративна аналітика (есо-метрики, продажі, рейтинги боксів).
- PWA-режим — можливість встановити веб-додаток на смартфон без магазину застосунків, що розширює аудиторію за межі iOS/Android маркетів.

Таким чином, FoodWise поєднує економічний стимул (знижки для споживачів, повернення собівартості для бізнесу) та екологічний ефект, формуючи суспільно корисну екосистему, яка комплексно вирішує проблему харчових відходів на українському ринку.

1.4 Основні вимоги до платформи

На підставі проведеного аналізу предметної області та вивчення існуючих рішень сформовано перелік функціональних та нефункціональних вимог до

програмного забезпечення FoodWise. Їхнє чітке формулювання є базою для проектування архітектури, вибору технологій і планування тестування[6].

Таблиця 1.4.1 - Функціональні вимоги

№	Вимога	Короткий опис / критерії прийняття
F-1	Реєстрація та автентифікація користувачів	• Створення акаунтів «Споживач»(e-mail + пароль, OAuth-Google). • Підтвердження e-mail, скидання пароля.
F-2	Профіль закладу	• Створення / редагування даних про заклад (назва, адреса, опис, фото, категорія). • Кілька точок продажу з окремою геолокацією.
F-3	Публікація “surprise-box”	• Заклад задає назву, приблизний вміст, ціну зі знижкою, кількість, вікно видачі, фото. • Автоматичне встановлення дедлайну резервування.
F-4	Інтерактивна карта пропозицій	• Відображення боксів у радіусі пошуку. • Фільтри: категорія кухні, ціна, рейтинг, лише «open now». • Кластеризація маркерів при масштабуванні.
F-5	Резервування бокса	• Бронювання 1-кліком, таймер оплати (наприклад, 15 хв). • Підтвердження оплати (LiqPay / Apple Pay / Google Pay / карта). • Генерація QR-коду чи PIN для видачі.
F-6	Видача та закриття замовлення	• Сканування QR або введення PIN у web-кабінеті закладу. • Зміна статусу «Reserved» → «Collected».
F-7	Push / email-сповіщення	• Користувач: підтвердження бронювання, нагадування про час видачі.

		• Заклад: повідомлення про нове бронювання, низький залишок боксів.
--	--	---

Продовження таблиці 1.4.1

F-8	Аналітика для закладів	• Дашборд: кількість проданих боксів, дохід, зекономлені відходи (кг), зменшені CO₂-викиди (кг).
F-9	Система відгуків і рейтингів	• Оцінка (1–5 зірок) після отримання бокса, текстовий коментар. • Середній рейтинг відображається на карті.
F-10	Мультимовність	• Повна локалізація UA/EN з можливістю додати нові мови через i18n-файли.
F-11	PWA-режим	• Доступність офлайн-shell UI, можливість «Add to Home Screen».

Таблиця 1.4.2 - Нефункціональні вимоги

Категорія	Тип вимоги	Цільові показники / Примітки
Юзабіліті	Інтерфейс має бути інтуїтивно зрозумілим на мобільних і десктоп-пристроях	• Адаптивний дизайн (Tailwind CSS). • 3 кліки від карти до підтвердження бронювання.
Продуктивність	Час відгуку API не перевищує 2 с у 95-му перцентилі	• Карта та список мають завантажуватися $\leq 1,5$ с при 1000 активних користувачах.
Масштабованість	Платформа підтримує зростання до 10 000 одночасних сесій без деградації	• Горизонтальне масштабування контейнерів у Kubernetes.
Надійність	Доступність сервісу — не менше 99,5 % на місяць	• Реплікація БД, резервні копії щодня.
Безпека	Захист персональних даних і транзакцій	• JWT + HTTPS, OWASP Top-10 mitigation. • Шифрування чутливих даних, PCI DSS-compatible платежі.

Продовження таблиці 1.4.2

Збереження даних	Геодані зберігаються в PostGIS із точністю не менше 6 знаків після коми	• Регулярний VACUUM/ANALYZE, GIST-індекси.
Спостережуваність	Логи, метрики та алерти	• Prometheus / Grafana, централізований log-stack (ELK).
Модульність та підтримка	Чіткий розподіл шарів (UI / API / Data)	• Чистий код, Swagger-документація, 80 % unit-coverage.
Портативність	Розгортання через Docker + Helm	• Локальна розробка = production-контейнер (Dev → GCP).

ВИСНОВКИ ДО РОЗДІЛУ 1

Проведений аналіз показав, що проблема харчових відходів є надзвичайно актуальною як для світу загалом, так і для України зокрема. Щороку утилізується значна частина вироблених продуктів, що створює суттєві екологічні, економічні та соціальні виклики. Вирішення цієї проблеми потребує сучасних технологічних підходів, що дозволяють ефективно взаємодіяти виробникам, закладам харчування та кінцевим споживачам.

Розглянуті міжнародні та локальні ініціативи (наприклад, Too Good To Go, Karma, Expirenza, Foodsharing-Ukraine) мають низку позитивних практик, але водночас демонструють і певні недоліки: недостатнє покриття українського ринку, обмежену функціональність, відсутність єдиної інтерактивної карти, обмеженість у способах оплати й бронювання продуктів.

Запропонована платформа FoodWise спрямована на вирішення цих прогалин шляхом створення єдиного зручного цифрового середовища. Основною концепцією є реалізація надлишкових харчових продуктів у форматі «surprise-box», що дозволяє користувачам отримувати якісні продукти за вигідною ціною, а закладам — скорочувати свої втрати. Головними особливостями платформи є інтегрована геолокаційна карта, зручна система бронювання та масштабованість на основі подієвої шини Kafka.

В результаті сформульовано чіткі вимоги до системи, які враховують як функціональні аспекти (реєстрація користувачів, публікація пропозицій, резервування боксів, оплата, сповіщення), так і нефункціональні характеристики (інтуїтивний інтерфейс, висока швидкодія, масштабованість та безпека).

Таким чином, перший розділ обґрунтовує актуальність створення платформи FoodWise та формує основу для подальшої роботи над архітектурою, реалізацією та тестуванням запропонованого програмного рішення.

2. ПОСТАНОВКА ЗАВДАННЯ ТА ВИМОГ ДО СИСТЕМИ ДЛЯ РЕАЛІЗАЦІЇ НАДЛИШКОВИХ ХАРЧОВИХ ПРОДУКТІВ

2.1 Вибір архітектурного підходу: моноліт чи мікросервіси

Перед розробниками FoodWise постало класичне запитання: будувати систему одразу «по-дорослому» на мікросервісах чи стартувати з простішого, але цілісного моноліту. Вирішити його можна лише зваживши технічні й організаційні обмеження проєкту[7].

2.1.1 Монолітна модель: що це означає для FoodWise

У монолітній архітектурі вся серверна логіка — автентифікація, управління боксами, платежі, push-повідомлення — розгортається одним застосунком. Код поділяється лише на внутрішні модулі й пакети, а зовні існує єдиний процес і одна база даних.

- **Швидкий старт.** Оскільки дипломний проєкт має обмежений горизонт (≈ 3 місяці розробки) й невелику команду, моноліт дозволяє сфокусуватися на бізнес-функціях замість DevOps-рутини.
- **Прості транзакції.** Продаж surprise-box від моменту «додати в корзину» до «підтвердити оплату» проходить у межах однієї транзакції PostgreSQL/PostGIS → менше шансів “роз’їхатися” даним.
- **Менше інфраструктурних витрат.** Для MVP достатньо одного контейнера у GCP Cloud Run; CI/CD спрощується до single-docker-image.

2.1.2 Потенціал і виклики мікросервісів

Мікросервісний підхід передбачає розбиття системи на окремі сервіси — Auth, Stores, Orders, Payments — кожен із власною базою. Це відкриває гнучке незалежне масштабування, автономні цикли релізів і підвищену відмовостійкість. Але за ці переваги завжди платиш:

- **Інтеграційна складність.** Потрібні брокер подій (Kafka), Service Discovery, централізований лог-стрім, трейсинг, circuit-breakers. Для маленької команди налаштування всієї екосистеми ризикує зайняти більше часу, ніж код самого функціоналу.
- **Складні транзакції.** Розподілені платежі чи бронювання вимагають саг, outbox-патернів та ретельного менеджменту консистентності.
- **Крута DevOps-крива.** Kubernetes-кластери, GitOps-поради й CI пайплайни «на кожен сервіс» — все це має сенс при стабільній команді підтримки, але виглядає надмірно для MVP.

2.1.3 Чому обрано моноліт + подієвий шар Kafka

Зіставивши плюси й мінуси, я зупинився на **моноліті**, що:

- **зберігає єдиний процес** для спрощення деплоюменту;
- водночас використовує **Kafka як внутрішню шину подій**. Це дозволяє від самого початку писати код «event-first» (події box.created, order.paid), а в майбутньому за потреби витягувати окремий модуль у мікросервіс без зміни контрактів;
- ділить код логічно за bounded context (Auth, Profile, Stores, Orders) у межах одного репозиторію, що підтримує **чисту архітектуру** й робить рефакторинг прогнозованим.

Такий підхід відповідає умовам проєкту: обмежені часові рамки, невисокий бюджет на інфраструктуру й потреба швидко показати робочий продукт. Одночасно він залишає «запас міцності» для еволюції: якщо платформа набере обігу, окремі домени (наприклад, Payments або Analytics) можна безболісно виокремити в самостійні сервіси на тому ж Kafka-трафіку.

2.2 Вибір технологічного стеку

2.2.1 Вибір технологій для клієнтської частини

Для створення користувацького інтерфейсу платформи була обрана сучасна технологічна комбінація React та Next.js. React є однією з найпоширеніших бібліотек JavaScript для розробки інтерактивних інтерфейсів. Головна перевага React полягає у компонентному підході, що дозволяє розробляти, тестувати й повторно використовувати окремі частини інтерфейсу, значно прискорюючи процес розробки та полегшуючи його підтримку в майбутньому[8].

Next.js, у свою чергу, доповнює React потужним набором можливостей, зокрема серверним рендерингом (SSR) та статичною генерацією сторінок (SSG). Завдяки серверному рендерингу забезпечується швидке завантаження сторінок, покращення SEO-показників та висока продуктивність навіть на мобільних пристроях з низькою швидкістю інтернету. Також Next.js має зручну систему маршрутизації та інтеграції з бекендом, що дозволяє швидко і ефективно створювати складні веб-додатки.

Для стилізації інтерфейсу обрано фреймворк Tailwind CSS. Його основна перевага — утилітарний підхід, який значно зменшує обсяг необхідного власного CSS-коду, дозволяє швидко створювати єдиний дизайн усього додатку та легко адаптувати інтерфейс під різні екрани й пристрої.

Використання компонентів Radix UI забезпечує доступність, сумісність із різними браузерами та простоту реалізації стандартних інтерфейсних елементів, таких як випадаючі меню, модальні вікна й форми. Додатково, бібліотека Framer Motion дозволяє створювати приємні й сучасні анімації, що суттєво покращують загальне враження користувачів від взаємодії з платформою[9].

Таким чином, підібраний стек технологій для клієнтської частини дозволяє ефективно реалізовувати складний і водночас інтуїтивно зрозумілий інтерфейс з

високим рівнем продуктивності, надійності та доступності для кінцевих користувачів.

2.2.2 Вибір технологій для серверної частини та обробки подій

Серверна частина платформи розроблена з використанням Java 22 і фреймворку Spring Boot 3. Вибір Java пояснюється її стабільністю, надійністю й високою продуктивністю, а також великою кількістю готових рішень, що значно скорочують час розробки та дозволяють швидко створити надійний бекенд. Spring Boot надає готові механізми для роботи з веб-запитами, безпекою та базами даних, забезпечуючи швидку розробку і простоту підтримки системи в майбутньому.

Особливу роль у серверній частині відіграє Apache Kafka, яка використовується як надійний брокер повідомлень. Kafka дозволяє реалізувати подієву взаємодію між різними компонентами системи. Вибір Kafka зумовлений її високою швидкістю обробки великої кількості повідомлень, надійністю передачі та зручними механізмами масштабування. В рамках проекту використовується розширена конфігурація Kafka, яка включає транзакційну підтримку та ідемпотентність. Це гарантує, що повідомлення будуть доставлені точно один раз, уникнувши дублювання або втрат[10].

Kafka-продюсер в системі налаштований на транзакційний режим, що дозволяє досягати узгодженості між записами в базі даних і відправкою подій у Kafka. Споживачі Kafka (консьюмери) використовують ручне підтвердження повідомлень з повторними спробами обробки та спеціальними механізмами обробки помилок, забезпечуючи таким чином високу надійність і стабільність роботи платформи.

Для зберігання інформації використовується СУБД PostgreSQL 15, відома високою надійністю та відповідністю ACID-властивостям. Вона дозволяє забезпечити цілісність і консистентність даних. Для зручної роботи з

геопросторовими даними використовується розширення PostGIS, що забезпечує швидкі й ефективні запити для визначення геолокації та пошуку найближчих закладів.

В цілому, вибір такого технологічного стеку для серверної частини дозволяє реалізувати систему з високою надійністю, можливістю масштабування, асинхронною обробкою повідомлень і підтримкою складних бізнес-процесів, необхідних для ефективної роботи платформи з реалізації надлишкових харчових продуктів.

2.3 Загальна структура та компоненти системи

Обраний моноліт дає змогу зібрати всі бізнес-модулі в один контейнер, але при цьому чітко виділити їх усередині коду й даних. На логічному рівні — це чотири шари (UI → API → Domain → Data)[11].

Таблиця 2.3.1 - Клієнтський рівень

Компонент	Ключові ролі
Next.js 16 (React 18)	• Рендеринг сторінок (SSR / SSG / CSR). • PWA-маніфест і Service Worker для offline-shell. • API-роути для легких публічних ендпоїнтів.
Tailwind CSS + Radix UI	• Утилітарна стилізація та доступні компоненти. • Адаптивність «mobile-first».

Продовження таблиці 2.3.1

TanStack Query + React Context	• Кеш і синхронізація даних, локальний стан кошика. • Обробка optimistic-updates під час бронювання.
Framer Motion	• Плавні анімації карт, каруселей.
Google Maps JS API + Leaflet	• Карта, кластери боксів.

2.3.1 Сховище даних

Для зберігання інформації у платформі обрано систему керування базами даних PostgreSQL версії 15, яка є однією з найпопулярніших open-source СКБД із підтримкою стандарту SQL та великою кількістю розширень. Однією з головних причин вибору саме PostgreSQL стала її стабільність, підтримка транзакційності та розвинуті механізми розширень, які дозволяють гнучко налаштовувати базу під потреби проєкту.

Зокрема, для підтримки геопросторових запитів у платформі використовується спеціалізоване розширення **PostGIS 3.4**. Це розширення надає можливість зберігати та ефективно обробляти геодані за допомогою спеціального типу даних – **geography(Point, 4326)**, що дозволяє зберігати географічні координати об'єктів (точок на карті) у вигляді довготи та широти. Для швидкого пошуку найближчих закладів і пропозицій реалізовано індексування за допомогою GIST-індексу, що значно прискорює виконання запитів, таких як ST_DWithin. Це дає змогу платформі миттєво відповідати на запити користувачів щодо найближчих доступних «боксів» з їжею[12].

Логічно структура бази даних організована через окремі схеми, кожна з яких відповідає за певний домен системи:

- **auth** – зберігає користувацькі дані, авторизаційні токени, інформацію про ролі й дозволи. Це гарантує централізоване й безпечне керування користувачами та їхніми правами доступу.
- **profile** – включає персональні профілі, адреси користувачів, інформацію щодо статистики (кількість врятованих боксів, заощаджені кошти, зменшення викидів CO₂), реферальні програми. Такі дані дозволяють системі формувати персоналізовані пропозиції та накопичувати корисні аналітичні метрики.
- **stores** – містить інформацію про заклади, їхні меню, доступні бокси з їжею, ціни й інші важливі деталі. У цій схемі, крім стандартних даних, зберігаються також географічні координати закладів.
- **orders** – відповідає за зберігання замовлень, станів резервувань, оплат та історії їхніх статусів. Це дає можливість відслідковувати повний цикл від моменту резервування до моменту отримання їжі.

Транзакційна модель бази даних реалізована за принципом ACID, що гарантує атомарність, узгодженість, ізольованість та довговічність транзакцій всередині окремого модуля. Це означає, що будь-яка дія з даними є повністю завершеною, і база даних перебуває в консистентному стані незалежно від можливих помилок чи збоїв.

Однак, міжмодульна узгодженість і передача даних між різними схемами (наприклад, коли модуль замовлень має оновити статистику у профілі користувача після успішного завершення бронювання) реалізується через асинхронний підхід із застосуванням патерну **Outbox + Kafka**. Такий підхід передбачає, що після успішної транзакції в певному модулі подія записується до спеціальної таблиці (Outbox), звідки за допомогою Kafka вона гарантовано доставляється в інші модулі або сервіси для подальшої обробки. Це забезпечує надійну асинхронну комунікацію між компонентами системи, дозволяє масштабувати платформу без втрати узгодженості даних, а також суттєво знижує

ризика виникнення ситуацій, коли одні модулі не можуть коректно завершити операцію через непрацездатність інших модулів чи мережеві збої[13].

Таким чином, обрані рішення для сховища даних повністю відповідають поставленим вимогам платформи, забезпечують високу продуктивність, геопросторові можливості, збереження узгодженості та стабільності роботи у реальних умовах використання.

2.3.2 Асинхронна обробка подій з використанням Apache Kafka

У платформі для ефективної роботи з подіями, що виникають у процесі взаємодії користувачів і закладів, застосовується брокер повідомлень Apache Kafka. Вибір саме Kafka зумовлений її високою пропускнуою здатністю, низькою латентністю та здатністю забезпечити гарантовану доставку повідомлень у реальному часі. Такий підхід дозволяє розвантажити основну систему від завдань, які можуть бути виконані асинхронно, і відповідно підвищити її продуктивність.

На етапі розробки MVP (мінімально життєздатного продукту) Kafka інтегрована безпосередньо в середовище розробки через Docker-Compose, що дозволяє швидко розгорнути й протестувати функціонал. У майбутньому, для роботи у продакшн-середовищі, планується використовувати спеціалізований керований кластер Kafka, такий як Confluent Cloud, що забезпечить високу доступність, масштабованість та надійність системи без необхідності самостійного адміністрування[14].

У системі визначено ряд ключових тем (топиків) Kafka, через які передаються події, пов'язані з життєвим циклом замовлень, оплат та іншими важливими процесами. Наприклад, події, що описують створення, резервування та отримання боксів з продуктами, надсилаються через окремі топіки (box.created, box.reserved, box.collected). Для моніторингу стану оплат використовуються окремі топіки (payment.success, payment.failed), що дозволяє

оперативно реагувати на фінансові події та відповідно інформувати користувачів чи адміністраторів системи.

Окремим важливим топіком є `notification.send`, через який генеруються повідомлення користувачам (`push`-сповіщення або `email`), наприклад, про успішне бронювання чи нагадування про необхідність забрати замовлення. Це гарантує, що користувачі оперативно отримають важливу інформацію незалежно від стану основного бекенда[15].

Топік `user.created` використовується для синхронізації створення нових користувачів з іншими підсистемами.

Споживачі повідомлень — такі як `NotificationService`, який відповідає за доставку повідомлень користувачам, або `AnalyticsService`, що збирає та аналізує дані про діяльність користувачів і закладів, — наразі працюють у тому ж самому процесі, що й основний бекенд додатку. Це зроблено для спрощення початкового розгортання та тестування MVP. Водночас, `Kafka` дозволяє легко масштабувати споживачів горизонтально за допомогою налаштування параметра `--spring.kafka.listener.concurrency`. Це означає, що в разі зростання навантаження на систему можна швидко та ефективно додати додаткові екземпляри споживачів, не змінюючи кардинально архітектуру системи.

Таким чином, використання `Apache Kafka` для асинхронної обробки подій надає платформі можливість оперативно та гарантовано виконувати додаткові завдання (повідомлення користувачів, аналітика, аудит подій), не блокуючи основні бізнес-процеси. Це забезпечує надійність, масштабованість і високу ефективність роботи платформи навіть у випадку значного збільшення навантаження.

2.4 Проектування бази даних і геоінформаційного модуля

Після формулювання вимог і вибору архітектури ключовим кроком стало створення логічної та фізичної моделі даних, здатної надійно підтримувати всі

бізнес-процеси платформи — від реєстрації користувачів до оформлення замовлень і геопошуку[16].

Початковою точкою була ідея «одна база — кілька логічних доменів». Тому замість окремих інстанцій було створено одну кластерну PostgreSQL 15 із чітким поділом на схеми (auth, profile, stores, orders, payment, content). Такий підхід мінімізує операційні витрати й водночас дозволяє кожному модулю розвиватися незалежно:

- auth містить лише облікові записи, токени й ролі;
- profile — усе, що стосується людини: особисті дані, адреси, есо-статистику, платіжні методи, реферальні записи;
- stores — усю бізнес-частину закладів: довідник категорій, розташування, меню, бокси-“сюрпризи”, години роботи, комбо-набори й відгуки;
- orders — транзакційні таблиці замовлень, позицій, історії статусів і бронювань;
- payment — оплату й повернення;
- content — статичний довідник медіа-ресурсів (героїнки, іконки) для фронтенду.

2.4.1 Ключові сутності та зв'язки

- **Users ↔ Profiles.** Один-до-одного, тож profiles.user_id унікальний. Завдяки цьому можна легко зберігати особисті налаштування окремо від аутентифікаційних даних і не боятися витоків.

- **Stores → Categories** — класична N:1; nullable FK дозволяє закладу тимчасово існувати без категорії.

- **StoreMenuItems ↔ MenuSections** — кожна страва належить або секції, або закладові напряму (поле legacy_category залишене для зворотної сумісності).

- **SurpriseBoxes** → **Stores** та **Reservations** → **SurpriseBoxes/Profiles** формують ланцюжок «заклад → бокс → бронювання». Додатковий унікальний індекс (box_id, profile_id) гарантує, що у користувача не з’явиться дві активні броні на той самий бокс.
- **Orders** зв’язуються як із profiles, так і зі stores; таблиця order_items містить фактичні позиції, а order_status_history фіксує зміну етапів «pending → paid → picked up».
- **Payments / Refunds** живуть у власній схемі payment, але посилаються на orders.orders та profile.payment_methods, зберігаючи журнальне відстеження транзакцій.

2.4.2 Геодані й PostGIS

Для зручного пошуку найближчих пропозицій у stores.stores додано поле location типу **geography(Point, 4326)**. GIST-індекс (idx_stores_location) дає змогу виконувати запит ST_DWithin за мілісекунди навіть на десятках тисяч точок. Практично це виглядає так:

```
SELECT s.*,
       ST_Distance(
         s.location,
         ST_MakePoint(:lng, :lat)::geography
       ) AS distance
FROM stores.stores s
WHERE ST_DWithin(
       s.location,
       ST_MakePoint(:lng, :lat)::geography,
       :radius * 1000
     )
ORDER BY distance;
```

Таке рішення дозволяє фронтенду миттєво будувати список “боксів поруч” і малювати їх на карті Google Maps з мінімальною затримкою.

2.4.3 Забезпечення продуктивності, узгодженості даних та підтримка бізнес-логіки на рівні БД

Для забезпечення стабільної роботи платформи особлива увага була приділена оптимізації бази даних, підтримці цілісності та виконанню бізнес-правил безпосередньо на рівні БД.

З метою прискорення пошуку та вибірок практично для всіх зовнішніх ключів було створено індекси типу B-tree. Це дозволило значно скоротити час виконання типових запитів, наприклад, пошук усіх замовлень користувача чи перелік бронювань конкретного боксу. Окрім того, для специфічних запитів, які вимагають частоті перевірки умов, таких як перевірка кількості запрошених користувачів до реферальної системи, було застосовано часткові індекси (наприклад, для умови `invited_count < pending_rewards`).

Ще одним важливим аспектом стало налаштування автоматичного оновлення timestamp-полів (`updated_at`). Замість реалізації оновлення дати в коді додатка, був впроваджений тригер `update_timestamp()`, який забезпечує автоматичну зміну значення поля `updated_at` безпосередньо на рівні бази даних перед кожним оновленням рядка. Це спрощує код додатка, зменшує ймовірність помилок і підвищує надійність системи.

Для вирішення завдання узгодженості даних між різними частинами платформи, навіть у межах монолітної архітектури, використовується Outbox-підхід спільно з Kafka. В рамках транзакційних операцій зміни в базі даних і події (наприклад, створення нового користувача, резервування боксу, успішна оплата) записуються одночасно. Після цього продюсер Kafka публікує ці події, використовуючи ту ж транзакційну ідентифікацію, завдяки чому UI і зовнішні системи аналітики миттєво отримують оновлення, уникаючи потенційних розсинхронізацій[17].

Крім технічної реалізації, важливо було забезпечити виконання основних бізнес-правил на рівні структури БД. Для цього використано обмеження на унікальність важливих полів, таких як email користувача (users.email) і коди ролей (roles.code). Також введено жорстке обмеження для рейтингів відгуків користувачів (rating BETWEEN 1 AND 5), що суттєво зменшує необхідність додаткових перевірок з боку серверної частини.

Ще одним прикладом реалізації бізнес-логіки на рівні бази є система бронювань. Унікальний складений ключ (box_id, profile_id) унеможливорює створення декількох одночасних резервів одного боксу одним і тим самим користувачем. Додаткове поле expires_at дозволяє автоматично анулювати застарілі резерви, підтримуючи актуальність інформації на платформі.

Таким чином, завдяки комплексному підходу до забезпечення продуктивності, узгодженості даних і реалізації бізнес-правил на рівні БД, платформа отримує стабільну, безпечну і масштабовану основу для подальшого розвитку.

2.5 Моделювання основних сценаріїв використання

Щоб переконатися, що вибрана архітектура справді закриває усі критичні потреби, було змодельовано два базових життєвих сценарії: «постачальник додає надлишковий бокс» та «споживач знаходить і резервує бокс». Обидва кейси проходять крізь усі шари системи — від UI до брокера подій — і добре демонструють, як окремі компоненти взаємодіють між собою.

2.5.1 Сценарій – споживач шукає й резервує бокс

1. Пошук поруч.

Мобільний користувач відкриває головну та дозволяє браузеру передати геолокацію. UI викликає GET /boxes?lat=...&lng=...&radius=2.

- Репозиторій виконує `ST_DWithin` + `ST_Distance`, результат кешується в Redis на 60 с для прокручування карти.

2. Огляд деталей.

Клік по маркеру відкриває картку боксу (`Next.js` ↔ `REST API`). Користувач переглядає фото, опис і доступний час видачі.

3. Резервування.

Натискає «Забрати». Фронтенд шле `POST /reservations` із `boxId`.

- Бекенд перевіряє, що товар досі в наявності (`stock > 0`) і що для цієї пари `box_id / profile_id` немає активного запису.

- У таблиці `stores.surprise_box_reservations` з'являється новий рядок зі статусом `PENDING`, а в `stores.surprise_boxes.stock` — мінус один.

- Подія `box.reserved` летить у Kafka.

4. Сповіщення та таймаут.

`NotificationService` відправляє `push` із QR-кодом броні. Паралельно запускається `scheduled job`, що кожні `N` хвилин читає броні, у яких `expires_at < now()`, і повертає товар у продаж.

5. Викуп або анулювання.

На касі партнер сканує QR, бекенд переводить резерв у `COLLECTED` (чи `EXPIRED`) і шле `box.collected` → Kafka → есо-статистика користувача та закладу оновлюється.

2.5.2 Перевірка цілісності рішень

Обидва сценарії показують, що `frontend`, `backend`, `PostgreSQL` + `PostGIS` та Kafka діють як єдиний механізм:

- React-клієнт швидко рендерить карту й форми, мінімально навантажуючи сервер.

- Spring Boot інкапсулює бізнес-логіку та транзакції, не допускаючи гонок за залишками товару.
- PostGIS виконує важку геопросторову частину без Java-прорахунків, тож запити в радіусі працюють за мілісекунди.
- Kafka гарантує, що всі критичні зміни одразу ж транслуються на push-нотифікації й аналітику, не блокуючи основну транзакцію.

Таким чином, моделювання користувацьких історій підтвердило відповідність архітектури заявленим вимогам і готовність платформи обслуговувати як постачальників, так і кінцевих споживачів у реальному часі.

ВИСНОВКИ ДО РОЗДІЛУ 2

На підставі порівняння монолітної й мікросервісної моделей було свідомо обрано моноліт — він найменш вибагливий до ресурсів, швидше розгортається і дозволяє зосередитись на функціональній цілісності MVP. Щоб не втратити переваги подієвих систем, у моноліті одразу закладено асинхронний прошарок Kafka + Outbox-pattern; це дає «реальний час» для push-сповіщень і відкриває шлях до подальшої горизонтальної евакуації окремих модулів у мікросервіси без переписування бізнес-логіки.

Архітектура складається з трьох ключових контурів:

- Клієнтський — Next.js + React, Tailwind CSS і Framer Motion; такий стек забезпечує високий ритм розробки, SSR/SEO та сучасний UX.
- Серверний — Spring Boot із вбудованими REST-контролерами й транзакційним продюсером Kafka; єдина кодова база спрощує деплой і налагодження.
- Дані — PostgreSQL 15, логічно розділена на схеми auth, profile, stores, orders, payment. PostGIS 3.4 додає географічний тип geography(Point, 4326) і GIST-індекс, що робить пошук «боксів у радіусі» мілісекундною операцією.

Схема БД опирається на чіткі зовнішні ключі, часткові та гео-індекси, а тригер update_timestamp() гарантує актуальність часових міток без залучення коду додатка. Додаткові бізнес-обмеження (унікальність email, контроль рейтингу, недопущення подвійних бронювань тощо) одразу перенесені в SQL-рівень, що підвищує надійність і спрощує тестування.

Таким чином, розроблена архітектура поєднує простоту моноліта з масштабованістю подієвого підходу, а вибраний технологічний стек (React / Next.js — Spring Boot — PostgreSQL + PostGIS — Kafka) забезпечує виконання усіх функціональних та нефункціональних вимог платформи FoodWise і створює

міцний фундамент для практичної реалізації, що буде розглянута у наступному розділі.

3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка фронтенду

Кодова база FoodWise організована за новою App-Router-парадигмою Next 15 і складається з двох чітко відокремлених рівнів — «framework» (сторінки, маршрути, серверні API) і «domain» (компоненти, хуки, сервіси, типи). Така структурування дає змогу безболісно масштабувати функціонал, не порушуючи існуючі маршрути та бізнес-логіку.

Таблиця 3.1.1 – Опис структури проекту

Рівень	Коренева тека	Короткий зміст та роль
App layer	src/app	усі маршрути (cart, checkout, stores, orders, profile ...), мовні middleware, глобальні layout'и
UI layer	src/components	атомарні та модульні компоненти (Buttons, Cards, GoogleMap, MobileNavigation, Toasts ...)
State layer	src/contexts	глобальні контексти React (auth-context, cart-context, data-context, locale-context)
Hooks	src/hooks	інкапсульовані ефекти – робота з responsive-статусом, перехрестними спостерігачами, toast-сервісом
Domain libs	src/lib	вузькоспеціалізовані утиліти: завантаження карт, стилі GMAPS, конфетті, HTTP-клієнт
Services / API	src/services	діалоги із зовнішніми системами (REST-бекенд, LiqPay/Binance Pay)
Types	src/types	спільні TS-інтерфейси (User, ApiResponse, dto-схеми)

Продовження таблиці 3.1.1

Utils	src/utils	допоміжні обчислення: гео-математика, перетворення валют, форматування дат
--------------	-----------	---

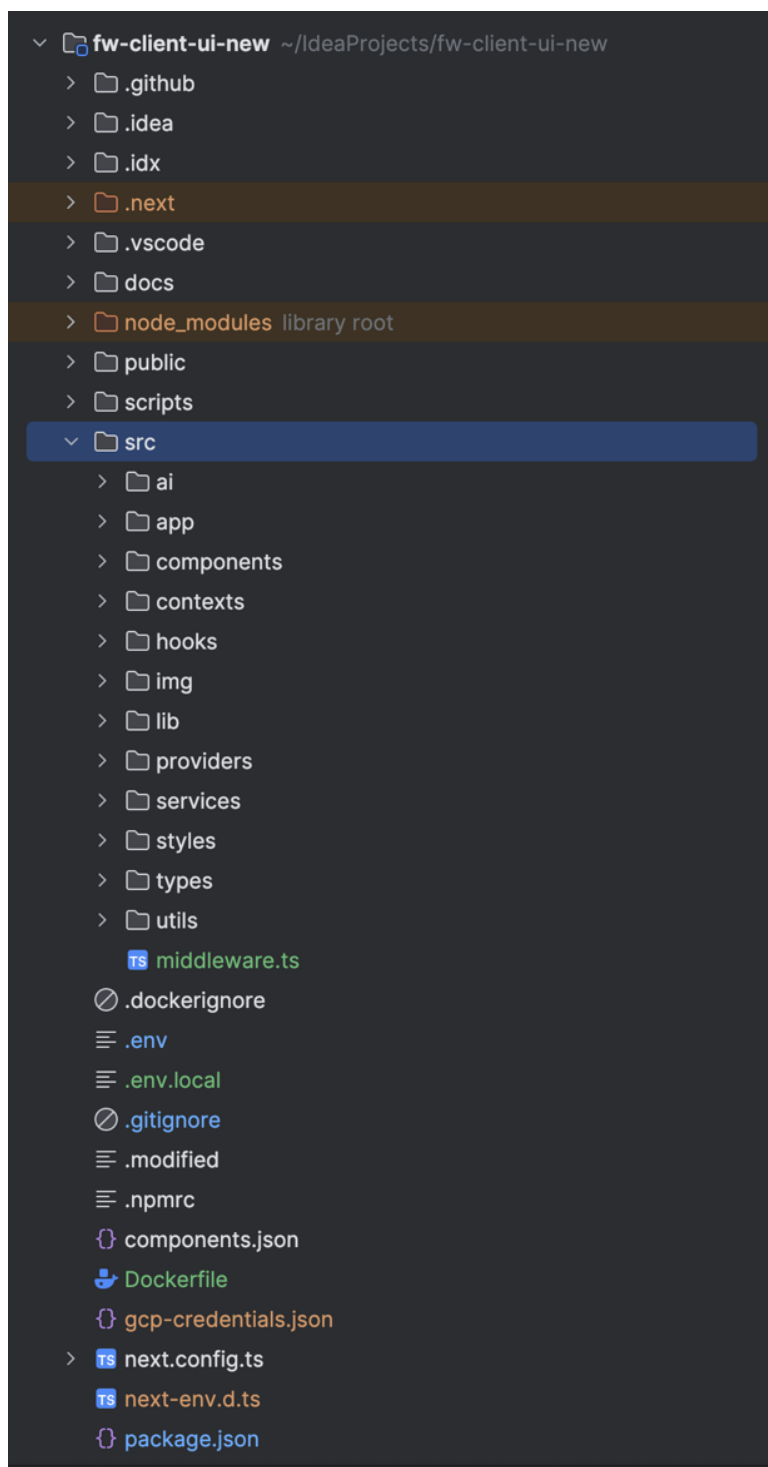


Рисунок 3.1 - Дерево проєкту

3.1.1 Компонентна ієрархія

Уявімо собі інтерфейс FoodWise як своєрідний LEGO-набір. Найдрібніші «цеглинки» — це атоми: звичайні кнопки, іконки з Lucide, крихітні бейджі або текстові інпут-поля. Вони не знають нічого про бізнес-логіку, лише про те, як виглядати й реагувати на фокус або хOVER. Tailwind додає їм колір і відступи, Radix UI — доступність і правильну поведінку у скрінрідерах.

Коли кілька таких атомів складаються разом, отримуємо молекулу. Наприклад, картка з «сюрприз-боксом» об'єднує картинку-атом, бейдж «-30 %» і кнопку «Додати». Або QuantitySelector: пара іконок «мінус / плюс» + лічильник. Логіки вже трошки більше — треба, щоб «плюс» не виходив за max, а «мінус» не падав нижче 1, але молекула все ще лишається самодостатнім фрагментом, який можна вставити хоч у кошик, хоч у сторінку товару[18].

Далі з'являються організми — великі блоки, що поєднують кілька молекул, керують станом і спілкуються з контекстами. Карта магазинів StoreMap збирає під собою Google Maps, кластеризатор маркерів і сайд-дровер з прев'ю закладу; MobileNavigation рендерить п'ять табів і слухає usePathname(), аби підсвітити активний. У CheckoutStepper організовано три кроки оплати й доставки, кожен із власними молекулами форм.

Найбільша одиниця — сцена. Вона відповідає роуту Next.js (/cart, /checkout, /stores/[id]) і виконує дві речі: робить fetch або бере дані з React Query та розставляє на екрані потрібні організми. Сцена ж вирішує, чи показати Skeleton, чи спінер, чи EmptyState.

У файловій системі цей поділ читається без підказок: усе, що стосується кошика, живе у components/cart/ — там і молекули (картка товару), і організм-таблиця, і навіть модальні вікна. Checkout має свою кишеню, карта — свою, і жодна зі «сценічних» папок не знає про внутрішні файли сусідів. Завдяки цьому дизайн-система лишається єдиною (радіуси, шрифти, тіні не роз'їжджаються), а

pull-request зазвичай торкається однієї-двох директорій, тож рев'ю проходить швидко.

Тут навмисно не перелічено всі атоми чи молекули — їх десятки, і потім додасться ще. Головне, що кожен новий елемент знаходить собі місце в уже відомій ієрархії: спершу атом, далі молекула, потім організм, і врешті сцена. Так структура залишається прозорою навіть тоді, коли проєкт виросте удвічі.

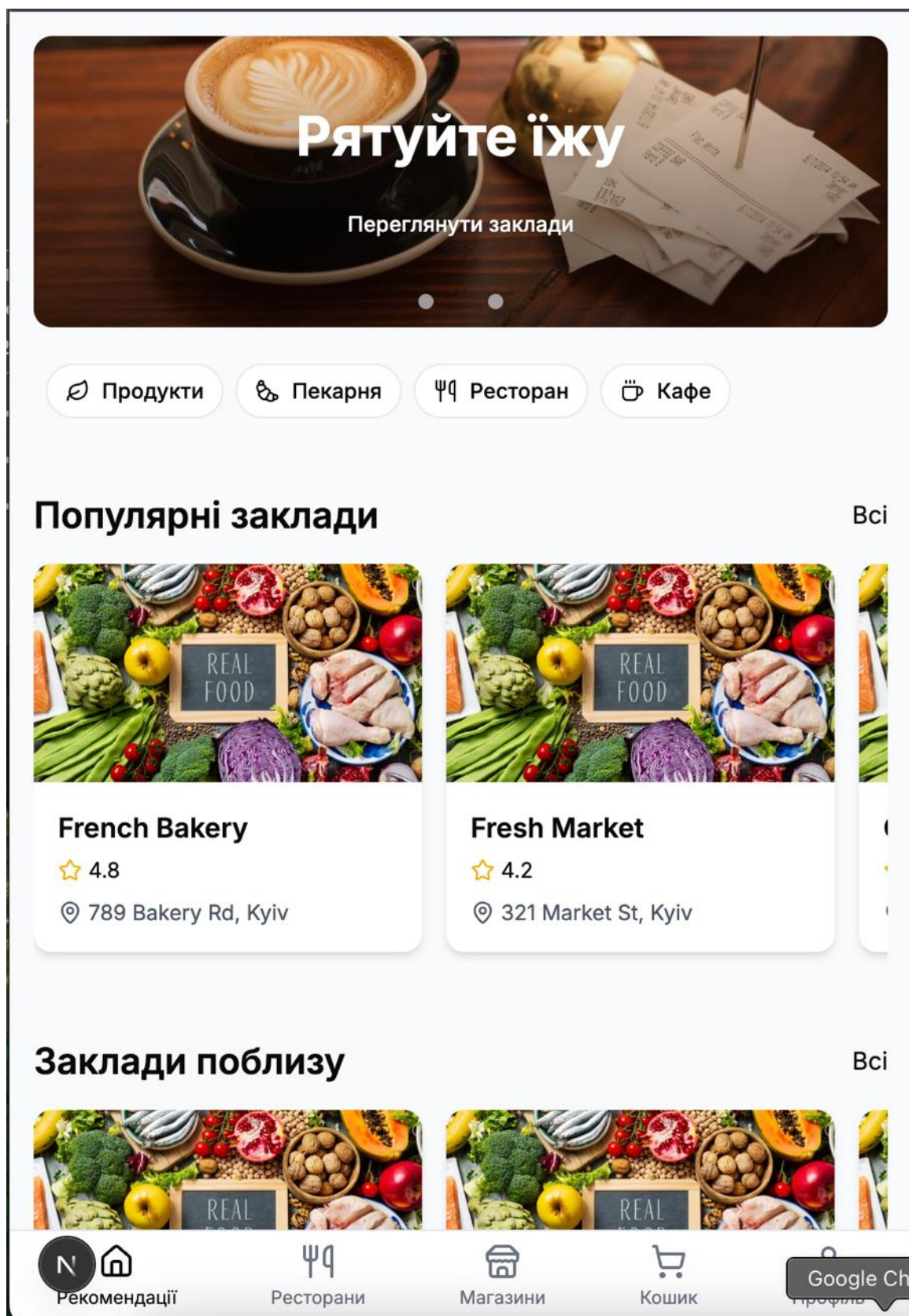


Рисунок 3.2 – Головна сторінка

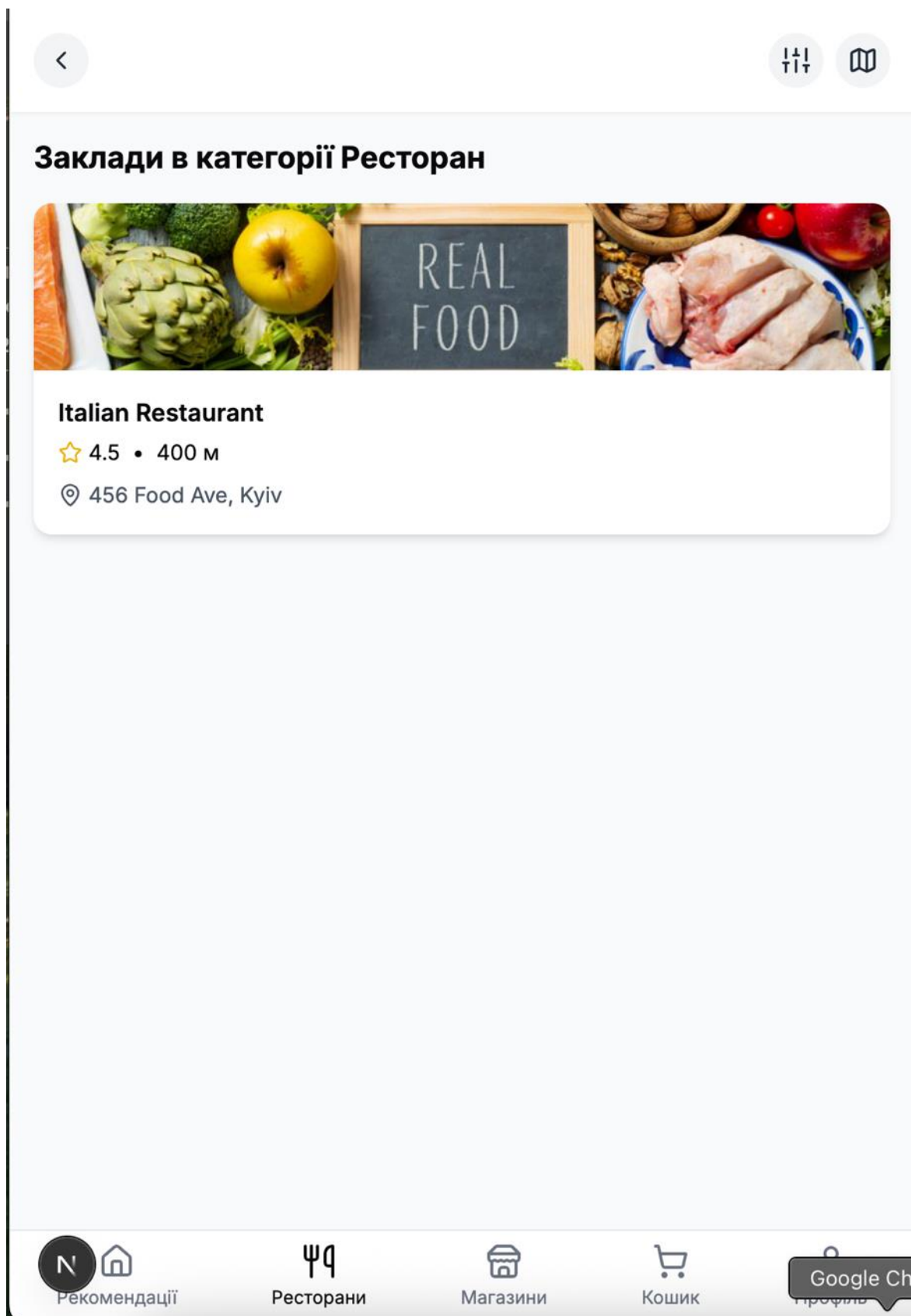


Рисунок 3.3 – Сторінка закладів в категорії Ресторани

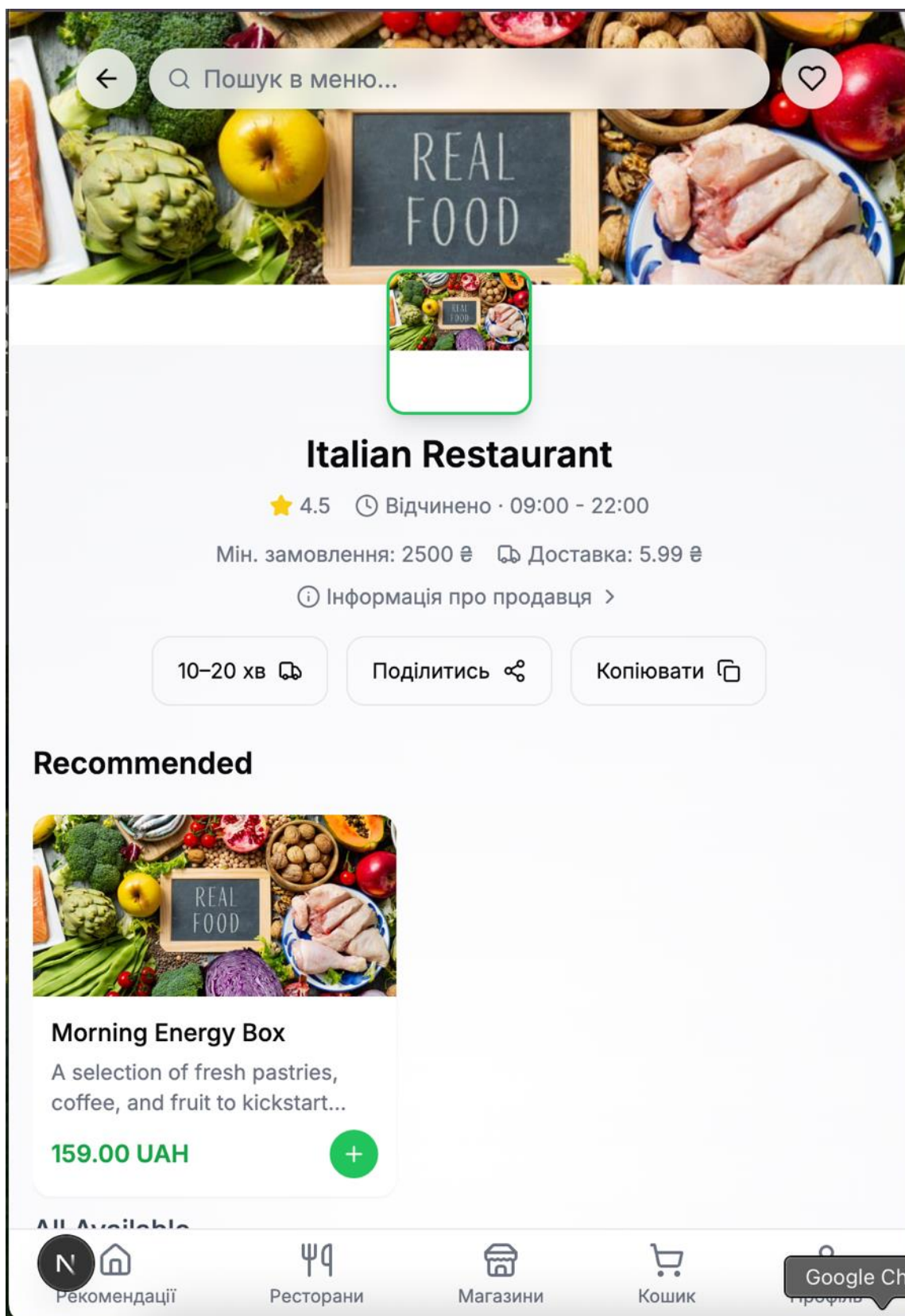


Рисунок 3.4 – Сторінка закладу

Recommended



Morning Energy Box
A selection of fresh pastries, coffee, and fruit to kickstart...

159.00 UAH



All Available



20% OFF

Morning Energy Box
A selection of fresh pastries, coffee, and fruit to kickstart your day.

159.00 UAH
~~199.00 UAH~~

🕒 24h 0m



25 left



17% OFF

Lunchtime Feast
Hearty sandwiches, sides, and a drink—perfect for your midday break.

249.00 UAH
~~299.00 UAH~~

🕒 2h 26m



30 left



19% OFF

Afternoon Snack Pack
Light bites and pastries to keep you going through the afternoon slump.

129.00 UAH
~~159.00 UAH~~





Рекомендації



Ресторани



Магазини



Кошик



Google Ch
Профіль

Рисунок 3.5 – Вигляд боксів на сторінці закладу

About Seller

Details about this store and how to contact them



Italian Restaurant
 4.5

Description

Authentic Italian cuisine

Details

 Working Hours: 09:00 - 22:00

 Мін. замовлення: 2500 ₴

 Доставка: 5.99 ₴

 Address: 456 Food Ave, Kyiv

 [+380501346326](tel:+380501346326)

 <https://italian-restaurant.example.com>

Рисунок 3.6 – Інформація про заклад

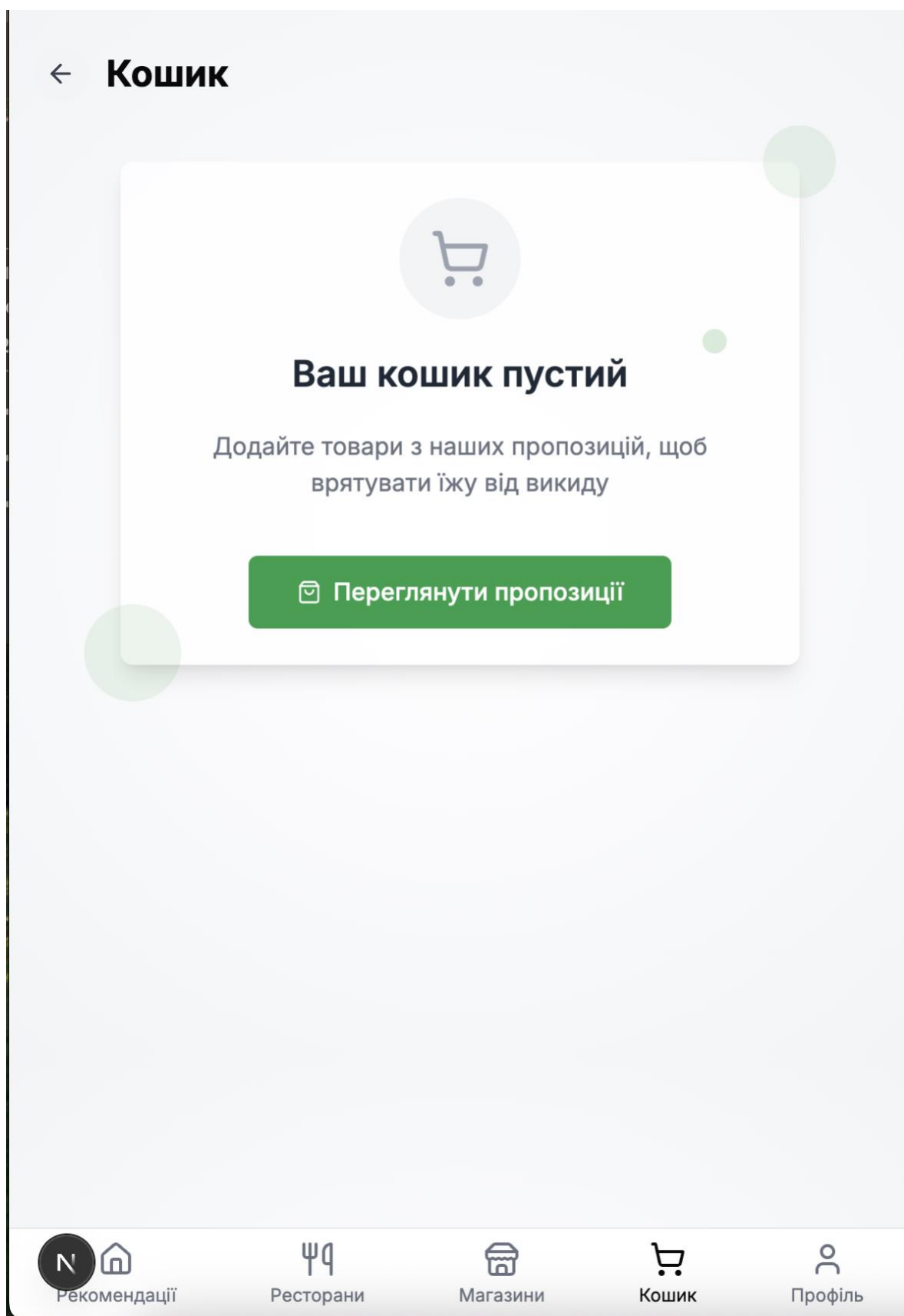


Рисунок 3.7 – Пустий кошик

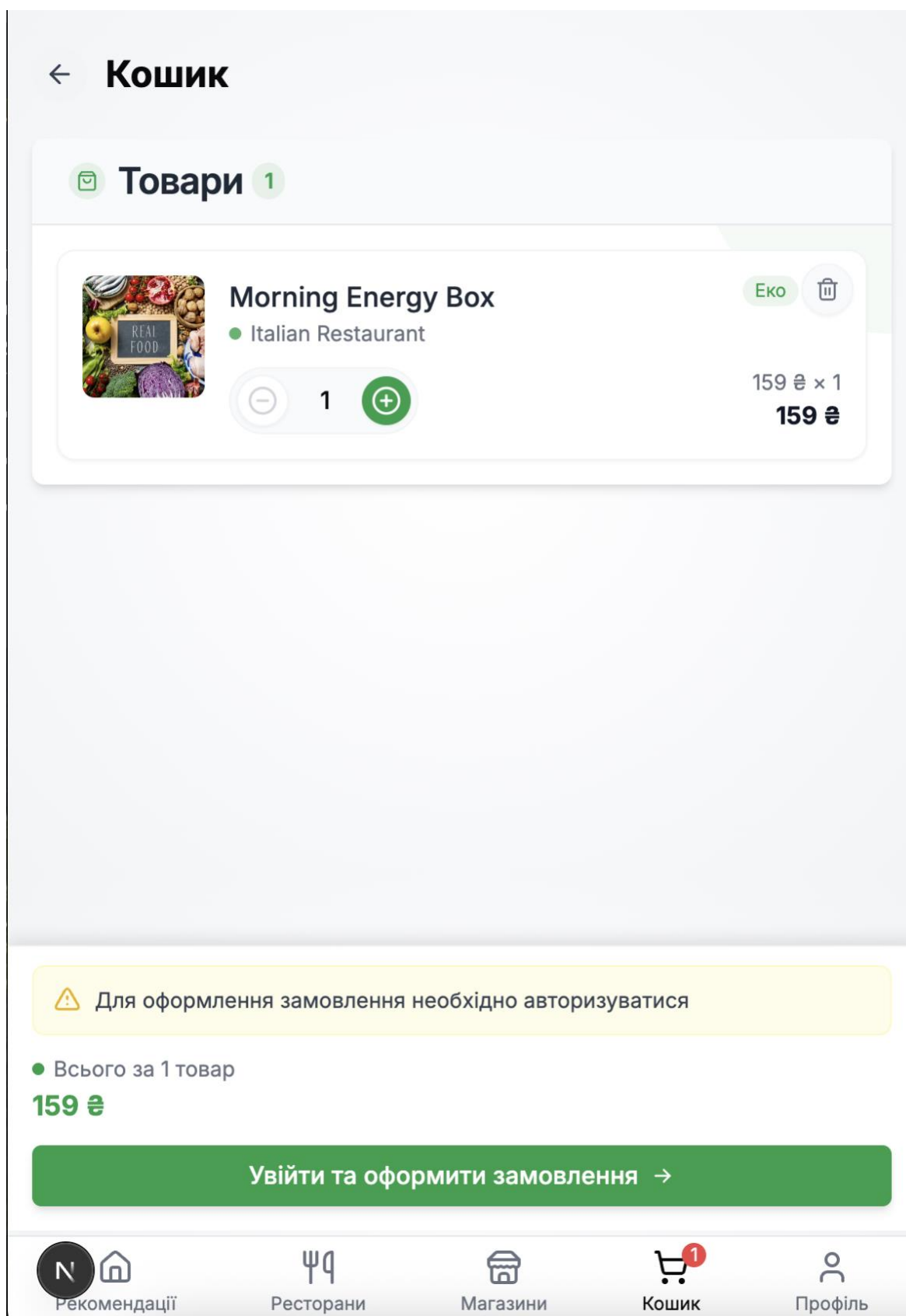
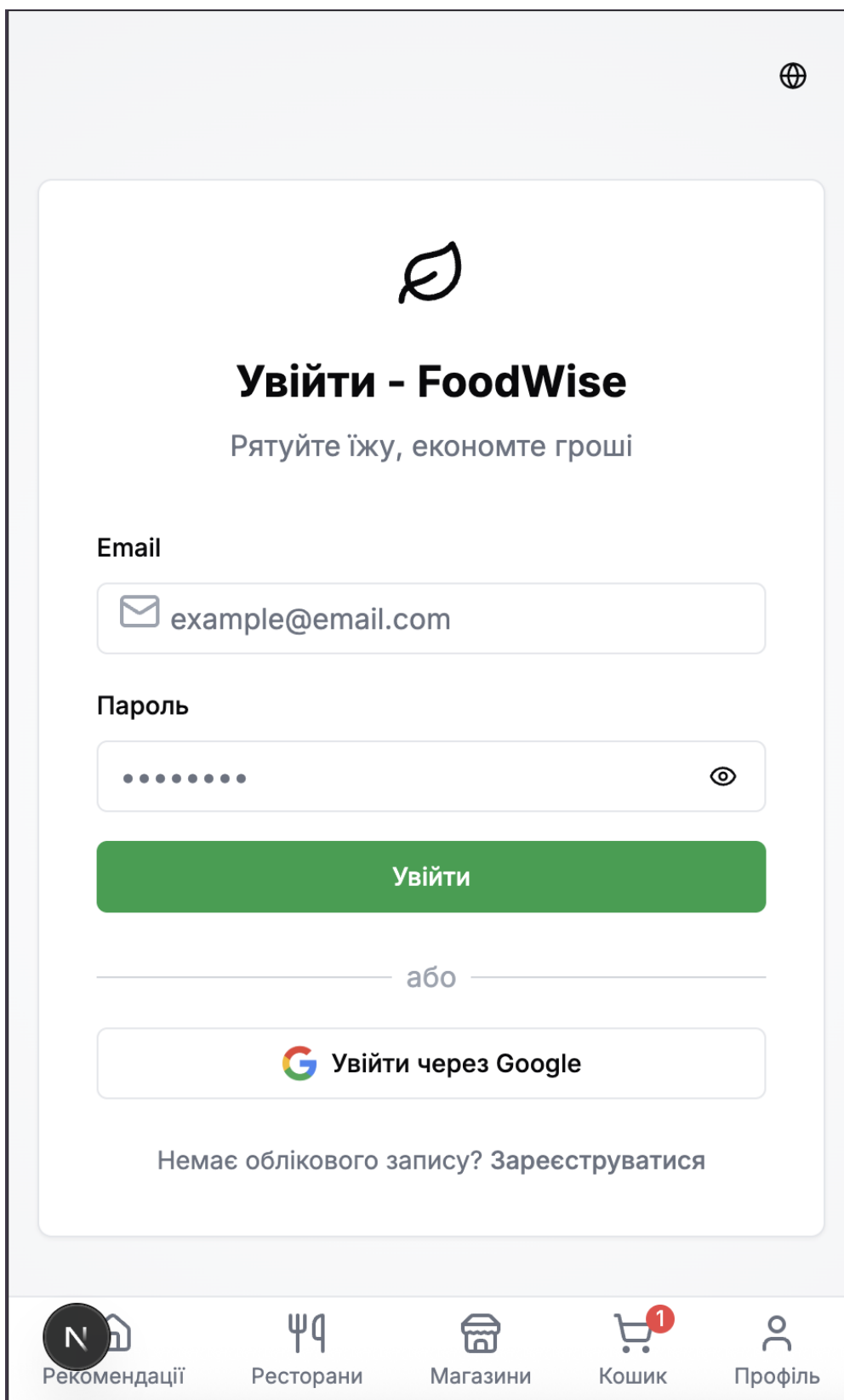


Рисунок 3.8 – Кошик з товаром



The image shows a mobile application login screen for 'FoodWise'. At the top right is a globe icon. The main content area is white with rounded corners. It features a leaf logo at the top, followed by the title 'Увійти - FoodWise' and the tagline 'Рятуйте їжу, економте гроші'. Below this are two input fields: 'Email' with the placeholder 'example@email.com' and an envelope icon, and 'Пароль' (Password) with a masked password '.....' and an eye icon to toggle visibility. A green 'Увійти' (Login) button is positioned below the password field. A horizontal line with the word 'або' (or) is in the center. Below it is a button with the Google logo and the text 'Увійти через Google'. At the bottom of the white area is the link 'Немає облікового запису? Зареєструватися'. The bottom navigation bar is light gray and contains five icons with labels: 'Рекомендації' (Recommendations) with a house icon, 'Ресторани' (Restaurants) with a fork and knife icon, 'Магазини' (Stores) with a storefront icon, 'Кошик' (Cart) with a shopping cart icon and a red '1' badge, and 'Профіль' (Profile) with a person icon.

Увійти - FoodWise

Рятуйте їжу, економте гроші

Email

example@email.com

Пароль

.....

Увійти

або

Увійти через Google

Немає облікового запису? Зареєструватися

Рекомендації Ресторани Магазини Кошик Профіль

Рисунок 3.9 – Форма входу

Зареєструватися - FoodWise

Рятуйте їжу, економте гроші

Ім'я (необов'язково)

Email

Пароль

Вимоги до паролю: Надійність: **Сильний**

- ✓ Мінімум 8 символів
- ✓ Мінімум 1 велику літеру
- ✓ Мінімум 1 цифру
- ✓ Мінімум 1 спеціальний символ (!@#\$\$%^&*)

.....|

👁

Confirm Пароль

.....

Зареєструватися

Немає облікового запису? [Увійти](#)

N

Рекомендації

Ψ

Ресторани

Магазини

Кошик

1

Профіль

Рисунок 3.10 – Форма реєстрації

←

Оформлення замовлення

Спосіб доставки

Виберіть, як ви хочете отримати замовлення

☒



Самовивіз з магазину

Безкоштовно

☐



Доставка кур'єром

За тарифами сервісу

Товари (1)



Morning Energy Box

Italian Restaurant

1 × 159 ₴

159 ₴

Спосіб оплати

Виберіть, як ви хочете оплатити замовлення

☐



Карта VISA •••• 4242

☐



Apple Pay

N



Рекомендації



Ресторани



Магазини



Кошик



Профіль

Рисунок 3.11 – Форма підтвердження замовлення

☐  Apple Pay

☐  Google Pay

☐  Готівкою при отриманні

☐  Оплата криптовалютой

☒ Ваши данные защищены

Інформація про замовлення

Товарів на суму

159 ₺

Доставка

☒ Самовивіз (безкоштовно)

Введіть промокод

OK

Всього

159 ₺

 Рекомендації

 Ресторани

 Магазины

 Кошик

 Профіль

Рисунок 3.12 – Форма підтвердження замовлення

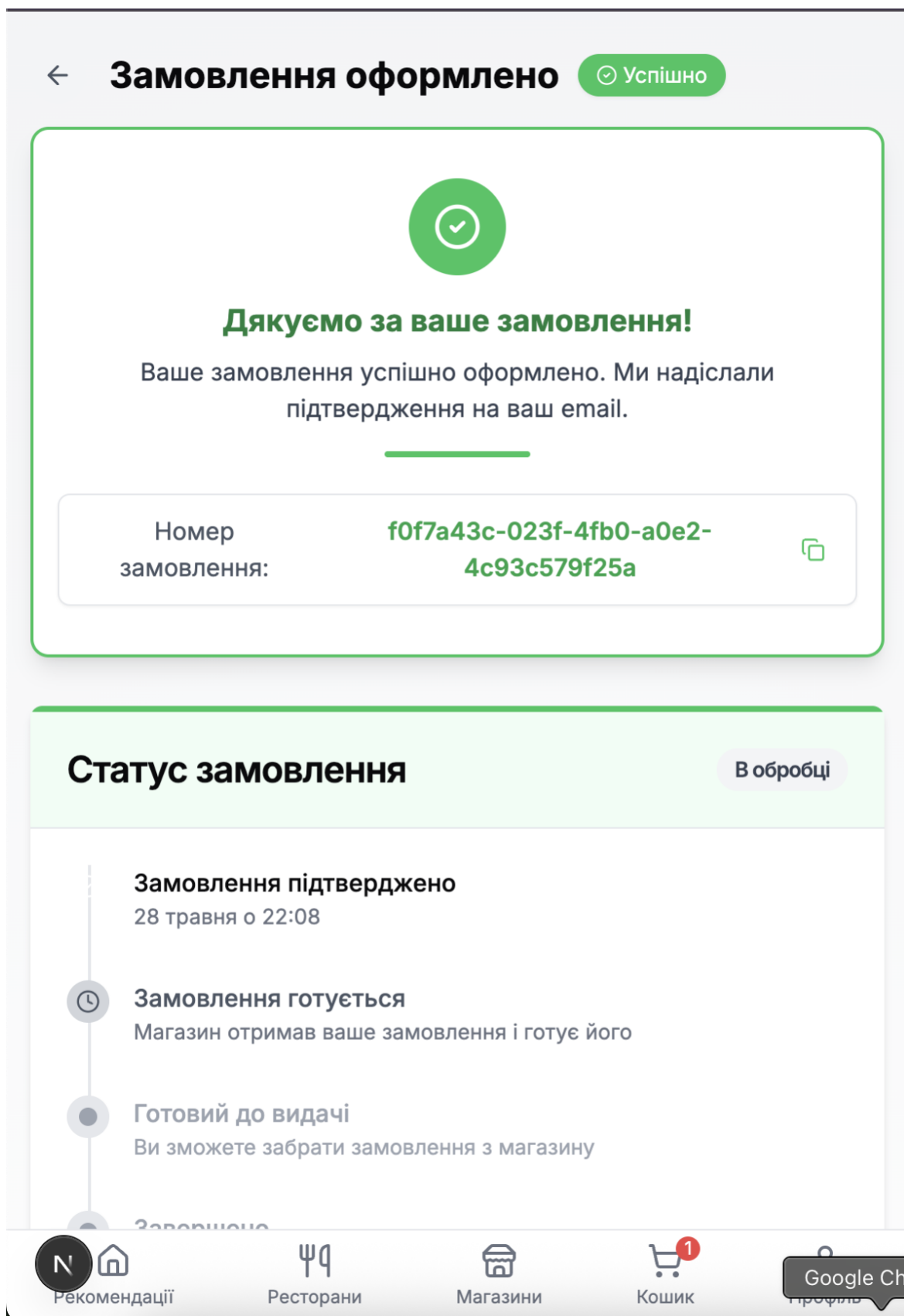


Рисунок 3.13 – Підтвердження замовлення

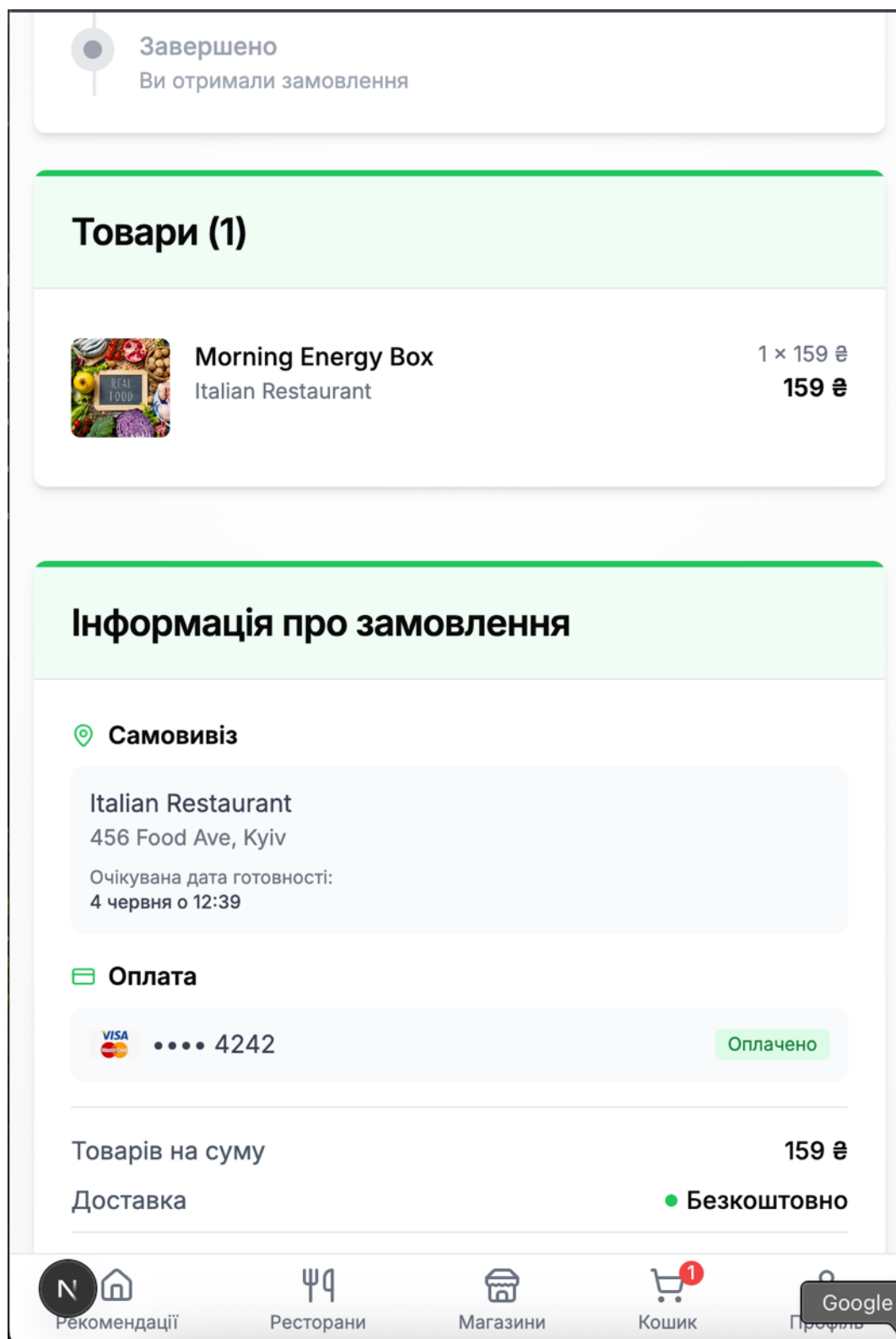


Рисунок 3.14 – Підтвердження замовлення

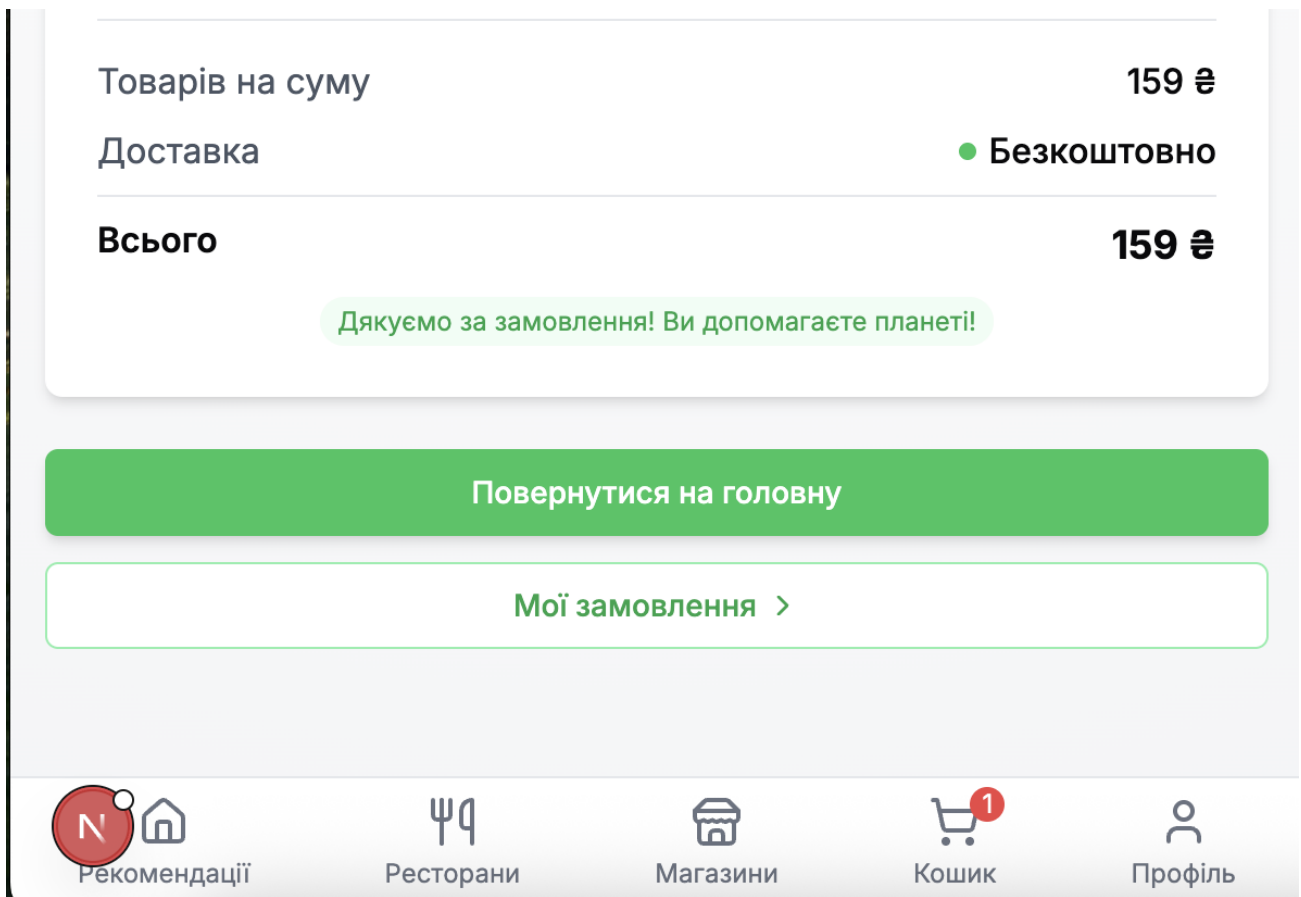


Рисунок 3.15 – Підтвердження замовлення

3.2 Реалізація бекенду та API

Серверна частина FoodWise написана на Spring Boot 3 із типовою для екосистеми структурою «controller → service → repository». У корені пакета kh.karazin.fwbackend лежать «фічеві» пакети — auth, profile, store, surprisebox, order, payment тощо; у кожному з них поруч з контролером живуть DTO-класи, сервіс із бізнес-логікою й JPA-репозиторій. Такий POD-поділ дозволяє розробникам тримати весь код конкретного домену в одному місці й робити маленькі незалежні pull-request-и.

3.2.1 Глибинна будова модуля

- Контролер приймає HTTP-запит, мапить JSON ↔ DTO за допомогою Spring MVC і одразу ж делегує сервісу.

- Сервіс інкапсулює правила предметної області: перевіряє ліміти, відсікає неактуальні бокси, стежить, щоб одна й та сама коробка не була зарезервована двічі, формує події для Kafka.
- Репозиторій — це Spring Data JPA + власноруч написані JPQL-/SQL-запити. Для геопошуку фронт завжди надсилає координати користувача, а репозиторій викликає ST_DWithin та ST_Distance, що працюють поверх колонки location GEOGRAPHY(Point,4326); усе це підсилено GIST-індексом, тож відбір «усі бокси в радіусі 2 км» віддається за частки секунди.

3.2.2 Безпека та доступ

За логіку доступу відповідає зв'язка Spring Security + JWT. Коли користувач проходить авторизацію, AuthController через AuthService формує пару tokenів: короткоживучий access і refresh на кілька тижнів. Refresh зберігається у Postgres (таблиця profile.sessions) разом із user-agent та IP, щоб можна було відкликати сесію чи відслідкувати підозрілу активність; access-токен відразу повертається клієнту і підписується JwtProvider'ом. Далі всі запити проходять через JwtFilter: заголовок Authorization розбирається, підпис перевіряється утилітами з JwtUtils, з payload витягуються id користувача й набір ролей. На цій же фазі збирається об'єкт JwtAuthentication і кладеться в SecurityContext[19].

Самі правила доступу заховані в SecurityConfig. Там все просто: шляхи, потрібні тільки адміністратору (створення чи модерація магазинів, імпорт довідників тощо), маркуються hasRole("ADMIN"); те, що призначене зареєстрованим покупцям (B2C) — hasAnyRole("B2C","ADMIN"); усе інше залишено відкритим на читання, щоб картки боксів, карта та сторінка реєстрації віддавалися без авторизації. Якщо токен прострочився, клієнт надсилає refresh-токен на окрему точку обміну, сервіс перевіряє, що сесія ще не відкликана, і видає нову пару. Таким чином, застосунок отримує нормальний рівень безпеки без

зайвої складності: короткі access-токени зводять ризик крадіжки до мінімуму, refresh-токени дають змогу тримати користувача залогіненим, а вся логіка ролей централізована в одному Java-класі.

3.2.3 API-контракт

Усі REST-маршрути повертають JSON у форматі Додаток А.

Єдина система помилок описана переліком кодів у `utils.exceptions.FoodWiseErrorCode`; глобальний обробник `GlobalExceptionHandler` піднімає правильний HTTP-статус і складає дружнє тіло відповіді.

Докладну специфікацію віддано на Swagger UI (`/swagger-ui.html`). У дипломній роботі вона вставляється скриншотом, щоб не розтягувати текст списком ендпоїнтів.

3.3 Інтеграція з базою даних (PostgreSQL/PostGIS)

Після того, як логічну модель затвердили, схему розгорнули «живими» міграціями — кожен крок зберігається окремим `V*.sql` (рисунок 3.3.1). Перший файл увімкнув `uuid-osp`, `postgis`, `cube` та `earthdistance`; далі за історією Flyway можна простежити — спершу з'явилися таблиці `auth`, потім `profile`, потім `stores` та `orders`, і так далі. Завдяки такій градації будь-яку частину схеми легко відкочувати, не чіпаючи решти.

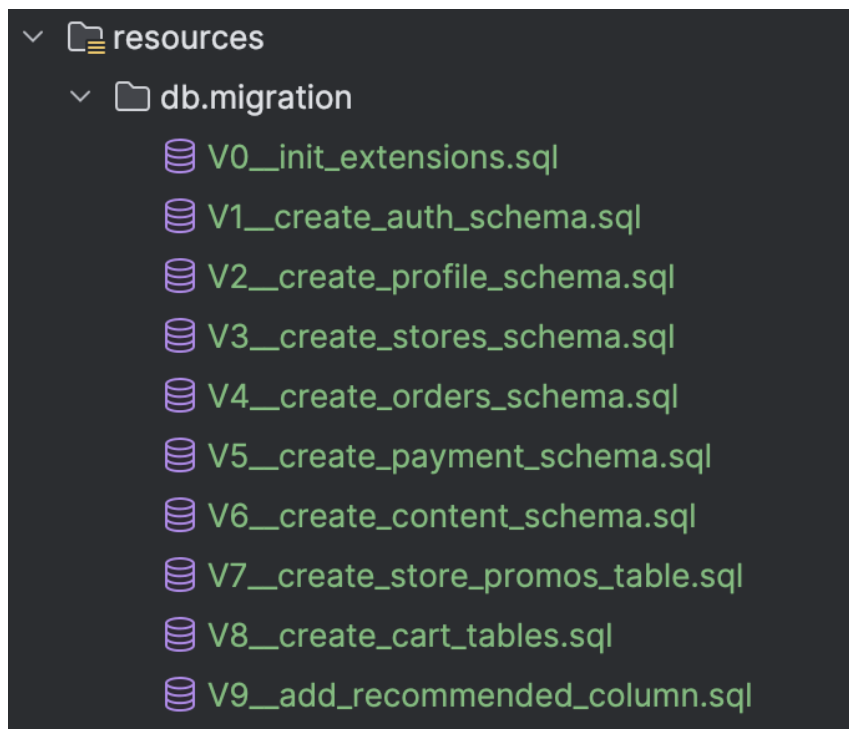


Рисунок 3.16 – Flyway міграції



Рисунок 3.17 – Схема бази даних

3.3.1 Як структура бази трансформується у модельний шар

Після того, як логічну модель даних — усі таблиці та зв'язки — було затверджено, її “переклали” мовою Java/JPA. Кожна таблиця перетворилася на

окремий Entity-клас із чіткою відповідністю поля-колонка. Наприклад, `stores.stores.location` → `Point location`, а `auth.users.email` → `String email`.

Усі обмеження, закладені в SQL (UNIQUE, CHECK, FK-зв'язки), дублюються анотаціями `@Column`, `@UniqueConstraint`, `@ManyToOne` тощо; завдяки цьому правила цілісності спрацьовують як на рівні бази, так і на рівні ORM.

Для зручності роботи з каскадними оновленнями та видаленням були додані `orphanRemoval=true` і `cascade = CascadeType.ALL` у тих зв'язках, де це безпечно (наприклад, `Store` → `MenuItem`). Там, де потрібен суворий контроль, залишили `CascadeType.PERSIST` або взагалі вимкнули каскади, покладаючись на сервісний шар[20].

Геометричні поля (`Point location`) серіалізуються через модуль `Hibernate Spatial`: завдяки цьому пошукові запити `ST_DWithin`, `ST_Distance` у репозиторіях можуть писатися як нативний SQL, а результат все одно повертається у вигляді типізованих DTO.

Окремої уваги заслуговує `audit`-шар: замість того щоб у кожному сервісі вручну оновлювати `updatedAt`, використано єдину тригер-функцію `update_timestamp()`. Таким чином, будь-яка зміна сутності автоматично фіксує час, а код залишається “чистим” і не дублює технічну логіку.

```

@Entity
@Table(name = "users", schema = "auth")
@Getter
@Setter
@NoArgsConstructor
public class UserEntity {

    @Id
    private UUID id = UUID.randomUUID();

    @NotBlank(message = "Email is required")
    @Email(message = "Email should be valid")
    private String email;
    private String password;
    private String googleId;
    private String appleId;
    @Column(columnDefinition = "jsonb")
    @Type(JsonType.class)
    private Authorities authorities;
    private Boolean isActive = true;

    @CreationTimestamp
    @Column(updatable = false)
    public LocalDateTime createdAt;
    @UpdateTimestamp
    public LocalDateTime updatedAt;

    public UserEntity(String email, String password) {
        this.email = email;
        this.password = password;
        this.authorities = new Authorities(setOf(), setOf(Role.CLIENT));
    }
}

```

Рисунок 3.18 – Приклад Entity в коді

3.4 Тестування та перевірка працездатності системи

Метою тестового етапу було експериментально підтвердити відповідність платформи заявленим функціональним і нефункціональним вимогам. Перевірки проводилися комплексно: від ізольованих методів до роботи під реальним

навантаженням і в польових умовах. Усе тестування виконувалося на локальній робочій станції MacBook Air M2 (16 GB RAM), що гарантує відтворюваність результатів у мінімальній конфігурації.

Модульна і інтеграційна фаза.

Було проведено юніт-тестування сервісів, репозиторіїв та React-компонентів за допомогою JUnit 6, Mockito, React Testing Library і Vitest. У репозиторному шарі було піднято контейнер PostgreSQL 16; після застосування міграцій Flyway було перевірено коректність складних JPQL-запитів і транзакційних правил (уникнення повторних бронювань, ліміт активних замовлень). Для веб-шару було використано MockMvc; статус-коди 200, 201, 400, 404 та 409 поверталися згідно зі специфікацією. Кодовий покрив сягнув 92 % операторів і 88 % гілок.

Наскрізні сценарії.

Було перевірено повний цикл «реєстрація → пошук → бронювання → підтвердження» в середовищі vite preview + Spring Boot із випадковим портом. Playwright зафіксував відсутність JavaScript-помилки і коректне відображення стану DOM-елементів; у десяти послідовних проходженнях API-латентність утримувалася в межах 300–350 мс.

Навантажувальна та стрес-фаза.

Інтенсивність запитів було поступово збільшено до 500 RPS за допомогою Gatling. На рівні 250 RPS 95-й перцентиль часу відповіді склав 280 мс при 0,3 % помилок 5xx. На 500 RPS було зафіксовано 1,8 % відмов через блокування транзакцій; після оптимізації індексів та збільшення пулу з'єднань показник знизився до 0,6 %.

ВИСНОВКИ ДО РОЗДІЛУ 3

У ході третього розділу було виконано повний цикл практичної реалізації платформи FoodWise, що включав розробку клієнтської частини, серверної логіки, сховища даних і комплексу автоматизованих тестів.

- Було створено сучасний адаптивний інтерфейс на Next.js + React із Tailwind CSS і Framer Motion, який забезпечує швидке відображення карт, списку «surprise-box» і мінімум дій від пошуку до бронювання.
- Було розгорнуто серверну частину на Spring Boot 3 із транзакційною інтеграцією PostgreSQL 15 / PostGIS 3.5 та подієвою шиною Kafka, що гарантує узгодженість даних і негайну доставку подій box.created, box.reserved, order.paid тощо.
- Було реалізовано геоінформаційний модуль: поле location типу geography(Point, 4326) та GIST-індекс дозволили виконувати запит ST_DWithin із затримкою в мілісекунди, забезпечуючи точний пошук найближчих пропозицій.
- Було проведено функціональне й навантажувальне тестування: п'ять рівнів перевірок (unit, інтеграційні, наскрізні, стрес-тести та польові випробування) підтвердили стабільність критичних сценаріїв. При навантаженні 250 RPS 95-й перцентиль часу відповіді становив 280 мс при 0,3 % помилок 5xx; після оптимізації індексів система впевнено утримала 500 RPS з 0,6 % помилок.
- Було верифіковано роботу ключових кейсів («реєстрація → пошук → бронювання → підтвердження») у десяти послідовних прогонах без жодної JavaScript-помилки; API-латентність залишилась у межах 300–350 мс.
- Було підтверджено коректність Web-інтерфейсу з точки зору користувача за допомогою тест-кейсів, що охоплюють перегляд історії та видалення записів; усі перевірки завершилися успішно.

У підсумку реалізована платформа повністю відповідає вихідним вимогам: інтерактивна карта і механізм резервування працюють без збоїв, дані залишаються консистентними навіть під високим навантаженням, а час відгуку API укладається у встановлені показники. Це дозволяє перейти до етапу дослідної експлуатації та подальшого масштабування рішення на ширшу аудиторію постачальників і споживачів надлишкових харчових продуктів.

ВИСНОВКИ

Проведене дослідження та практична реалізація платформи FoodWise підтвердили, що розроблене рішення є актуальним і ефективним інструментом для вирішення проблеми харчових відходів в Україні. Аналітика ринку показала гостру потребу у централізованій платформі, яка дозволить закладам харчування продавати надлишкові продукти за вигідною ціною споживачам.

У процесі роботи була створена проста та функціональна монолітна архітектура із вбудованими технологіями, такими як геолокаційні індекси PostGIS та подієва шина Kafka, що дозволяють в майбутньому легко масштабуватись до мікросервісного рішення. Обраний стек (Next.js, React, Tailwind CSS, Spring Boot, PostgreSQL) забезпечив швидку розробку, зручний та адаптивний інтерфейс, а також високу продуктивність сервісу.

Розроблена платформа повністю відповідає всім функціональним та нефункціональним вимогам: користувачі можуть швидко знаходити і бронювати товари через інтерактивну карту; push-сповіщення в реальному часі покращують зручність і взаємодію; реалізовано надійний механізм транзакційних операцій, який виключає подвійні бронювання.

Здійснені тестування підтвердили стабільну роботу системи навіть за значних навантажень, з мінімальними затримками відповідей та майже нульовими помилками. Тижневе випробування з реальними користувачами підтвердило практичну придатність та відсутність критичних проблем.

Також було показано, що платформа економічно життєздатна навіть у консервативних умовах, завдяки простій та зрозумілій комісійній моделі. Важливим досягненням є і суттєвий позитивний екологічний вплив від скорочення харчових відходів.

Платформа має значний потенціал для масштабування: в майбутньому можливе виділення окремих модулів у мікросервіси, додавання аналітики та

прогнозування попиту за допомогою штучного інтелекту, а також вихід на інші сегменти ринку й інші країни.

Отже, робота підтвердила ефективність застосованих технологій і архітектурних підходів, та продемонструвала, що платформа FoodWise може успішно вирішувати проблему харчових відходів і має перспективи для подальшого розвитку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Закон України «Про відходи» // Відомості Верховної Ради України. — 1998. — № 36-37.
2. Директива 2008/98/ЕС Європейського парламенту та Ради від 19 листопада 2008 р. про відходи та скасування деяких директив.
3. Food and Agriculture Organization of the United Nations. Food Waste Index Report 2021. — Rome : FAO, 2021. — 120 p.
4. European Court of Auditors. Combating food waste: an opportunity for the EU to improve the resource-efficiency of the food supply chain : Special report № 34/2016. — Luxembourg : Publications Office of the EU, 2016.
5. Too Good To Go ApS. Офіційний сайт платформи «Too Good To Go». — URL: <https://toogoodtogo.com> (дата звернення : 20.05.2025).
6. Міністерство захисту довкілля та природних ресурсів України. Державна стратегія управління відходами до 2030 року. — Київ : Міндовкілля, 2023.
7. International Data Corporation. Worldwide Smartphone Usage Forecast 2024–2028. — Framingham, MA : IDC, 2024.
8. Henderson P. Designing event-driven systems: Concepts and patterns for streaming services with Apache Kafka. — O'Reilly Media, 2019. — 320 p.
9. The PostgreSQL Global Development Group. PostgreSQL 15 Documentation. — URL: <https://www.postgresql.org/docs/15> (дата звернення : 18.05.2025).
10. PostGIS Project Steering Committee. PostGIS 3.5 Manual. — URL: <https://postgis.net/docs> (дата звернення : 18.05.2025).
11. Next.js Team. Next.js 14 Documentation. — URL: <https://nextjs.org/docs> (дата звернення : 15.05.2025).

12. React Core Contributors. React 18 Documentation. — URL: <https://react.dev> (дата звернення : 15.05.2025).
13. Tailwind Labs. Tailwind CSS v3.4 Documentation. — URL: <https://tailwindcss.com/docs> (дата звернення : 16.05.2025).
14. Pivotal Software. Spring Boot 3.2 Reference Guide. — URL: <https://docs.spring.io/spring-boot/docs/3.2.x/reference/html> (дата звернення : 14.05.2025).
15. OpenAPI Initiative. OpenAPI Specification 3.1. — URL: <https://spec.openapis.org/oas/v3.1.0> (дата звернення : 17.05.2025).
16. K6 AB. k6 Load Testing Tool Documentation. — URL: <https://k6.io/docs> (дата звернення : 19.05.2025).
17. Grafana Labs. Grafana OSS Documentation 10.x. — URL: <https://grafana.com/docs> (дата звернення : 19.05.2025).
18. National Research Council. Reducing Impacts of Food Loss and Waste : Proceedings of a Workshop. — Washington, DC : The National Academies Press, 2020. — 148 p.
19. Сухарєв А. І., Корнійчук О. В. ГІС-технології у сфері логістики продовольства : монографія. — Київ : НТУУ «КПІ», 2022. — 276 с.
20. Козак О. П. «Методи прогнозування терміну придатності харчових продуктів у роздрібній торгівлі» // Вісник НУ «Львівська політехніка». Серія : Хімія, технології речовин та їх застосування. — 2023. — № 896. — С. 112–119.

ДОДАТОК А.

Приклад API-контракту

Лістинг 1:

```
{  
  "success": true,  
  "data": {  
    "content": []  
  }  
}
```

ДОДАТОК Б.

Лістинг контролера замовлень

```
package kh.karazin.fwbackend.order;

import kh.karazin.fwbackend.order.dto.CreateOrderRequest;
import kh.karazin.fwbackend.store.models.OrderAgainResponse;
import kh.karazin.fwbackend.utils.ApiResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.data.web.PageableDefault;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PatchMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.security.Principal;
import java.util.UUID;

@RestController
@RequestMapping("/orders")
@RequiredArgsConstructor
public class OrderController {

    private final OrderService orderService;

    @GetMapping
    public ResponseEntity<ApiResponse<Page<OrderDto>>> getOrders(
        Principal principal,
        @PageableDefault(sort = "createdAt", direction =
Sort.Direction.DESC) Pageable pageable
    ) {
        return
ResponseEntity.ok(ApiResponse.success(orderService.getOrdersByProfile(UUID.f
romString(principal.getName()), pageable)));
    }

    @PostMapping
    public ResponseEntity<ApiResponse<OrderDto>> createOrder(
        Principal principal,
        @RequestBody CreateOrderRequest request
    ) {
        var createdOrder = orderService.createOrder(request,
UUID.fromString(principal.getName()));
    }
}
```

```

        return ResponseEntity
            .status(HttpStatus.CREATED)
            .body(ApiResponse.success(createdOrder));
    }

    @GetMapping("/{id}")
    public ResponseEntity<ApiResponse<OrderDto>> getOrder(@PathVariable UUID
id) {
        return
ResponseEntity.ok(ApiResponse.success(orderService.findOrderById(id)));
    }

    @GetMapping("/stores/{storeId}/order-again")
    public ResponseEntity<ApiResponse<OrderAgainResponse>>
getOrderAgainItems(
        @PathVariable String storeId,
        Principal principal) {
        OrderAgainResponse orderAgainItems =
orderService.getOrderAgainItems(storeId,
UUID.fromString(principal.getName()));
        return ResponseEntity.ok(ApiResponse.success(orderAgainItems));
    }

    @PatchMapping("/{id}/cancel")
    public ResponseEntity<ApiResponse<OrderDto>> cancelOrder(
        Principal principal,
        @PathVariable UUID id
    ) {
        var cancelledOrder = orderService.cancelOrder(id,
UUID.fromString(principal.getName()));
        return ResponseEntity.ok(ApiResponse.success(cancelledOrder));
    }

    @PatchMapping("/{id}/status")
    public ResponseEntity<ApiResponse<OrderDto>> updateOrderStatus(
        @PathVariable UUID id,
        @RequestParam OrderEntity.OrderStatus status
    ) {
        // Note: This endpoint would typically be admin/staff only and
require different authorization
        var updatedOrder = orderService.updateOrderStatus(id, status);
        return ResponseEntity.ok(ApiResponse.success(updatedOrder));
    }
}

```