МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

Перший (бакалаврський)
(рівень вищої освіти)

на тему **Клієнтський застосунок для ресторанного обслуговування**

Виконав: студент 4 курсу, групи 6.1211-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
О. О. Лиходід
(ініціали та прізвище)
Керівник асистент кафедри ЕІСПЗ О.О. Жуков
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дісітел»
П.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____перший (бакалаврський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Т. В. Критська
“_01_” __березня_____2025_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Лиходіду Олексію Олексійовичу
(прізвище, ім’я, по батькові)

1.Тема роботи Веб-застосунок для автоматизації замовлень у закладі харчування
керівник роботи _____Олександр Олексійович Жуков, асистент_____
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від _____
2. Строк подання студентом кваліфікаційної роботи___15.05.2025_
3. Вихідні дані бакалаврської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми порівняння цін та пропозицій з різних джерел;
- створення програмного продукту та його опис; • перелік вимог для роботи програми;
- розробка висновків та пропозицій.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) _____слайдів презентації

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.03.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання	Примітка
1	Аналіз предметної області (автоматизація замовлень у закладах харчування)	15.05–16.05.25	виконано
2	Формулювання мети та задач дипломної роботи, погодження з науковим керівником	16.05–17.05.25	виконано
3	Аналіз існуючих аналогів (сервіси онлайн-замовлень: Glovo, Bolt Food, Menu.ua тощо)	17.05–18.05.25	виконано
4	Проєктування структури бази даних (таблиці: Categories, Dishes, Orders, Receipts)	18.05–19.05.25	виконано
5	Узгодження подальших дій з науковим керівником	19.05–20.05.25	виконано
6	Реалізація базових CRUD-операцій для роботи з таблицями бази даних	20.05–21.05.25	виконано
7	Розробка механізму формування замовлення і збереження чеку	22.05–24.05.25	виконано
8	Розробка API-контролерів для роботи з категоріями, стравами, замовленнями	24.05–25.05.25	виконано
9	Модульна структура проєкту: розділення на контролери, репозиторії, моделі	25.05–26.05.25	виконано
10	Узгодження структури інтерфейсу користувача з науковим керівником	26.05–27.05.25	виконано
11	Створення Razor Pages для сторінок: Меню, Оплата, Успішна оплата	27.05–29.05.25	виконано
12	Реалізація сторінок адміністрування: категорії, страви	29.05–31.05.25	виконано
13	Додавання стилізації інтерфейсу з використанням Bootstrap 5	01.06–02.06.25	виконано
14	Завершальне тестування: коректність збереження, редагування, оплати	03.06–05.06.25	виконано
15	Підготовка презентації для захисту	06.06–07.06.25	виконано
16	Написання та оформлення пояснювальної записки	07.06–12.06.25	виконано

Студент _____ О.О. Лиходід
(підпис) (прізвище та ініціали)

Керівник роботи _____ О.О. Жуков
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок — 64,
Рисунків — 21,
Джерел — 8.

Лиходід О. О. Клієнтський застосунок для ресторанного обслуговування: кваліфікаційна робота бакалавра спеціальності 121 «Програмне забезпечення систем» / наук. керівник О. О. Жуков. Запоріжжя : ЗНУ, 2025. 64 с.

У роботі описано створення веб-застосунку для автоматизації замовлень у сфері громадського харчування. Мета — розробити систему, що дозволяє переглядати меню, формувати замовлення та здійснювати електронну оплату. Реалізовано серверну частину на ASP.NET Core і клієнтську — з використанням Razor Pages. Система підтримує ролі адміністратора та користувача, включає управління меню, перегляд чеків і статусів замовлень.

Результатом є повнофункціональний веб-застосунок для малого бізнесу.

Ключові слова: *веб-застосунок, автоматизація, онлайн-замовлення, громадське харчування, ASP.NET Core, Razor Pages, REST API*.

ABSTRACT

Pages — 64.

Drawings — 21.

Sources — 8.

Lykhodyd O. O. Client Application for Restaurant Service: Qualification Thesis of the Bachelor's Degree in Specialty 121 "Software Systems" / Scientific Supervisor O. O. Zhukov. Zaporizhzhia: ZNU, 2025. 64 p.

This study explores the digital transformation of customer service in the food service sector through the development of a web-based application designed to automate order management processes. The adoption of modern web technologies facilitates improved service quality, operational efficiency, and streamlined user interaction.

The objective of the research is to design and implement a web application that provides functionalities for browsing the menu, placing orders, and completing payments online. The scope of work included analysis of existing solutions, design of system architecture, development of backend components using ASP.NET Core, and frontend implementation via Razor Pages. The application incorporates role-based access for users and administrators, supporting operations such as menu management, order tracking, and receipt generation.

The resulting solution is a fully functional web application suitable for deployment in small-scale food service establishments seeking to enhance their order processing capabilities.

Keywords: *web application, online ordering, automation, food service, ASP.NET Core, Razor Pages, database, REST API*.

ЗМІСТ

ВСТУП 7
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 11
1.1 Визначення аналітичного сервісу 11
1.2 Огляд літературних джерел 14
1.3 Аналіз програмних продуктів-аналогів 16
1.4 Постановка завдання 21
2 ДОСЛІДЖЕННЯ ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ 24
2.1 Аналіз сучасних систем управління базами даних 24
2.2 Аналіз сучасних технологій для створення серверної частини 27
2.3 Аналіз сучасних технологій для створення клієнтської частини 30
2.4 Огляд інших технологій 31
3 РОЗРОБКА АНАЛІТИЧНОГО СЕРВІСУ ВИБОРУ ТОВАРІВ 33
3.1 Опис предметної області 33
3.2 Вимоги до системи 34
3.3 Архітектура системи 36
3.3.1 Огляд бази даних 37
3.4 Розробка бази даних 42
3.5 Програмна реалізація серверної частини 44
3.5.4 Розробка пакету db-core 44
3.5.5 Розробка пакету API 49
3.6 Розробка дизайну застосунку 53
3.7 Програмна реалізація застосунку 57
3.8 Практичне використання додатку 60
ВИСНОВКИ 62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 64

ВСТУП

Актуальність теми

У сучасних умовах розвитку ресторанного бізнесу впровадження цифрових технологій є необхідною передумовою підвищення ефективності управління обслуговуванням клієнтів. Автоматизація процесів оформлення замовлень дозволяє суттєво скоротити час їх обробки, підвищити точність взаємодії між персоналом і клієнтами, а також зменшити кількість помилок, спричинених людським фактором. Застосування веб-орієнтованих інформаційних систем забезпечує зручний доступ користувачів до функціоналу замовлення, перегляду меню та здійснення оплати, з одночасним удосконаленням аналітичного інструментарію для адміністраторів.

Інтеграція інформаційної системи з базою даних створює передумови для оперативного оновлення інформації про позиції меню, структуру категорій і ціни, а також для подальшого аналізу динаміки замовлень з метою підвищення якості обслуговування. Автоматизація управління замовленнями сприяє оптимізації трудових процесів, зменшенню витрат на ручне ведення обліку та підвищенню загальної рентабельності закладу. Таким чином, розробка веб-застосунку для автоматизації замовлень у сфері громадського харчування є актуальним завданням сучасної прикладної інформатики.

Мета дослідження

Розробити веб-орієнтовану інформаційну систему, що автоматизує процес управління замовленнями у закладі громадського харчування, забезпечуючи функціональність перегляду меню, оформлення замовлень, їх обробки та інтеграції з системами онлайн-оплати.

Завдання дослідження

У межах поставленої мети передбачено: здійснення аналізу предметної області та існуючих аналогів подібних систем; проєктування структури бази даних для зберігання інформації про страви, категорії та замовлення; створення прикладного інтерфейсу програмування (API) для забезпечення взаємодії клієнтської частини з сервером; реалізацію веб-застосунку з функціональністю перегляду меню, вибору страв і здійснення оплати; впровадження адміністративного інтерфейсу для управління контентом; проведення тестування системи та оцінювання її ефективності в реальному середовищі використання.

Об’єкт дослідження

Інформаційні процеси, пов’язані з автоматизацією замовлень у сфері громадського харчування.

Предмет дослідження

Методи і засоби проєктування та реалізації веб-застосунку для автоматизації оформлення замовлень та їх обробки.

Методи дослідження

Застосовано теоретичні методи аналізу літературних джерел, огляду сучасних інформаційних технологій та порівняння існуючих підходів до побудови інформаційних систем. У межах практичної частини використано методи проєктування баз даних, побудови API та розробки інтерфейсів з подальшим тестуванням функціональності створеної системи.

Практичне значення одержаних результатів

Розроблена система дозволяє автоматизувати основні процеси обробки замовлень у закладах харчування, що сприяє підвищенню якості обслуговування клієнтів, раціоналізації роботи персоналу та зростанню економічної ефективності. Отримані результати мають прикладну цінність і можуть бути впроваджені у діяльність малих та середніх підприємств у сфері громадського харчування.

Глосарій

Інформаційна система – сукупність апаратних і програмних засобів, що забезпечують обробку, зберігання та передачу даних.

Меню – структурований перелік страв із супровідною інформацією (назва, опис, ціна).

Замовлення – сукупність позицій, обраних користувачем для подальшої оплати.

Оплата – процедура здійснення розрахунків за замовлення через електронні платіжні інструменти.

База даних – централізоване сховище структурованих відомостей про страви, замовлення, категорії тощо.

Фронтенд – інтерфейсна частина веб-застосунку, що забезпечує візуалізацію даних та взаємодію з користувачем.

Бекенд – серверна частина системи, відповідальна за логіку обробки запитів і взаємодію з базою даних.

CRUD-операції – базові дії з даними: створення, читання, оновлення, видалення.

Автоматизація – процес впровадження інформаційних технологій для підвищення ефективності операцій.

Адміністратор – уповноважений користувач, що здійснює управління контентом та налаштування системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення аналітичного сервісу

Автоматизована система замовлень у закладах громадського харчування становить собою спеціалізований програмний комплекс, що забезпечує інтеграцію операцій обробки даних, пов’язаних із створенням, реєстрацією, обліком та оплатою клієнтських замовлень. З функціонального погляду така система виконує завдання збору, зберігання, обробки та візуалізації інформації про страви, категорії, транзакції та статуси виконання замовлень. Її впровадження сприяє значному підвищенню ефективності бізнес-процесів, скороченню часу обслуговування, мінімізації людських помилок і підвищенню рівня сервісу.

Головна мета реалізованого програмного забезпечення полягає в автоматизації процесу замовлення харчових позицій через інтуїтивно зрозумілий, інтерактивний веб-інтерфейс. Цей інтерфейс надає користувачеві можливість пройти повний цикл взаємодії із системою — від ознайомлення з меню та вибору бажаних страв до електронної оплати і отримання підтвердження про успішне завершення операції. Усе це реалізується з урахуванням вимог ергономіки інтерфейсу, доступності функцій та забезпечення безпечної обробки персональних і платіжних даних.

З технічного погляду система реалізована за принципами трирівневої архітектури, яка включає:

- Рівень зберігання даних — реалізований за допомогою системи керування базами даних PostgreSQL, що забезпечує високий рівень надійності, транзакційності, масштабованості та підтримку складних запитів SQL. У базі акумулюється структурована інформація про категорії страв, самі страви,

замовлення, чеки, користувачів, а також інформація про транзакції та статуси їх обробки.

- Серверний рівень (backend) — реалізований на платформі ASP.NET Core, яка забезпечує обробку HTTP-запитів, реалізацію RESTful API, логіку бізнес-процесів (створення замовлення, розрахунок суми, оплата, оновлення статусів тощо) та взаємодію з базою даних. Серверна логіка побудована відповідно до принципів SOLID, з урахуванням шаблонів проєктування MVC та Repository Pattern для кращої підтримуваності та модульності коду.
- Клієнтський рівень (frontend) — побудований з використанням Razor Pages (фреймворк ASP.NET), який забезпечує генерацію HTML-сторінок на сервері та інтеграцію з backend API. Інтерфейс користувача адаптовано під мобільні та десктопні пристрої, реалізовано можливість формування замовлення в кілька кліків, з відображенням актуального статусу замовлення та можливістю вибору платіжного методу.
- Узагальнена схема, що включає описані рівні, представлена на рисунку 1.

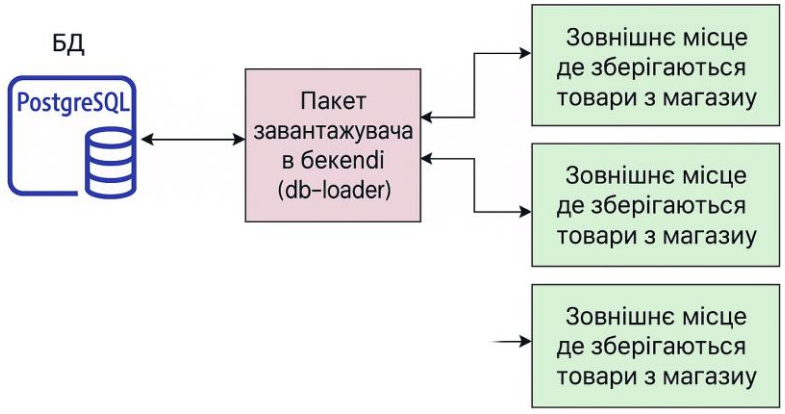


Рисунок 1 – *Схема роботи серверної частини*

Серверна частина системи виконує обробку запитів користувача, зокрема:

- отримання списку категорій та страв;
- додавання обраних позицій до кошика;
- формування замовлення на основі вмісту кошика;
- ініціалізація платіжної транзакції через інтегрований сервіс LiqPay;
- оновлення статусу замовлення після підтвердження оплати;
- формування чека і його збереження до бази даних.

Взаємодія між компонентами побудована на основі REST API, що дозволяє гнучко масштабувати систему, зокрема забезпечити підключення мобільного додатку, адміністративної панелі або зовнішніх аналітичних модулів.

Користувацький інтерфейс реалізований у вигляді SPA-подібного рішення з використанням Razor Pages, що забезпечує динамічне оновлення вмісту сторінки без необхідності повного перезавантаження. Рисунок 2 демонструє послідовність взаємодії клієнтської частини з API. Реалізовано можливість:

- перегляду категорій і страв;
- застосування фільтрів (наприклад, за доступністю, ціною);
- додавання та редагування кошика;
- вибору платіжного методу (онлайн або готівка);
- отримання підтвердження про успішне замовлення.

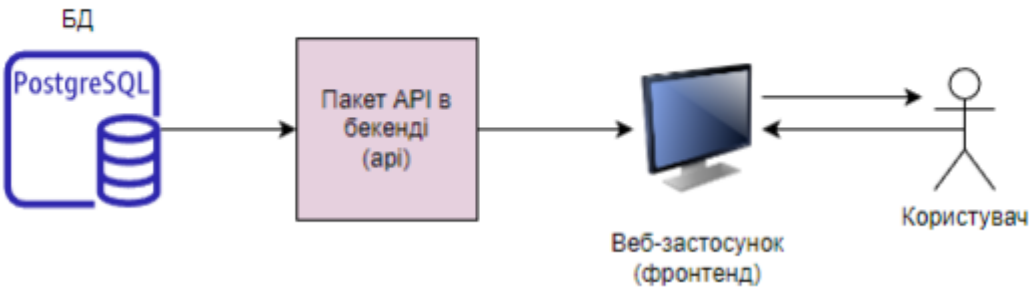


Рисунок 2 – *Схема взаємодії клієнтської частини з API*

Таким чином, запропонована система є прикладом інтегрованого підходу до автоматизації обслуговування в закладах громадського харчування. Реалізована архітектура дозволяє не лише ефективно обробляти замовлення, але й забезпечує розширюваність, масштабованість та захист даних. Урахування кращих практик з розробки веб-застосунків, а також використання відкритих технологій забезпечують зниження вартості володіння системою та відкривають можливості для її подальшого розвитку.

1.2 Огляд літературних джерел

У межах виконання кваліфікаційної роботи було здійснено комплексний огляд наукової, навчально-методичної та аналітичної літератури, яка висвітлює проблематику створення інформаційних систем, автоматизації бізнес-процесів у сфері громадського харчування, а також сучасні підходи до проєктування веб орієнтованих програмних систем. Метою цього аналізу було не лише ідентифікація сучасних технологічних трендів, але й виявлення обмежень існуючих рішень та формування концептуальних основ для реалізації власного застосунку.

Автор роботи [1] досліджує вплив автоматизованих інформаційних систем на всі аспекти діяльності ресторанних підприємств, від автоматизації

замовлень та управління запасами до аналітики та прогнозування попиту. У роботі обґрунтовується, що такі системи є комплексним набором інформаційних, програмно-технічних засобів та спеціалістів, призначених для оптимізації управлінських процесів та прийняття обґрунтованих рішень.

Дослідження включає детальну класифікацію існуючих систем управління ресторанним бізнесом, виокремлюючи сім основних класів від систем управління ресурсами підприємства до систем управління ланцюгами постачання. Практична значущість роботи підкріплюється аналізом провідних розробників програмного забезпечення, включаючи міжнародних лідерів та українських виробників технологічних рішень.

Автори роботи [2] представили дослідження ролі сучасних інформаційних технологій у контексті четвертої промислової революції в сфері громадського харчування. Автори розглядають POS-системи як життєво важливий інструмент ведення ресторанного бізнесу, що поєднує в собі функції касового апарату, системи управління запасами, інструменту звітності та аналітики, а також платформи управління співробітниками.

Практичну цінність роботи підкреслює представлене емпіричне дослідження компанії myservername.com, яке виявило пріоритети власників ресторанного бізнесу щодо функцій POS-систем. Згідно з результатами, найбільш затребуваною функцією є управління запасами, яку вимагають 48% респондентів, за якою слідують відстеження звітів про продажі та керування співробітниками. Автори детально аналізують переваги хмарних POS-систем над локальними рішеннями, особливо підкреслюючи такі аспекти як безкоштовні оновлення, простота використання та зменшення фінансових інвестицій. Стаття також розглядає практичні аспекти комп'ютеризації ресторанів, включаючи мобільність обслуговування через використання планшетів та впровадження систем самостійного замовлення, що сприяє

підвищенню ефективності обслуговування клієнтів та мінімізації помилок у процесі прийняття замовлень.

Дослідження функціональності комерційних рішень, таких як Poster POS [3] і R-Keeper [4], дало змогу здійснити порівняльний аналіз промислових інформаційних систем, які активно використовуються в ресторанній галузі. Аналіз функцій управління меню, обліку товарів, аналітики продажів, підтримки безготівкових оплат, ролей користувачів і підключення зовнішніх сервісів дозволив сформулювати мінімально необхідний набір функціональних можливостей, що мають бути реалізовані у власному програмному рішенні. Особливу увагу було приділено обмеженням зазначених систем — зокрема, високій вартості впровадження, складності інтеграції з іншими сервісами, жорсткій прив’язаності до фірмового обладнання — що стало підґрунтям для обґрунтування розробки відкритої вебплатформи, адаптованої до потреб малого бізнесу.

У результаті узагальнення проведеного аналізу літературних джерел було підтверджено актуальність створення спеціалізованого вебзастосунку для автоматизації процесів замовлення в закладах громадського харчування. Виявлені у дослідженнях обмеження комерційних рішень, зокрема низька адаптивність до малого бізнесу, складність кастомізації, відсутність відкритого доступу до вихідного коду – засвідчують доцільність створення власної системи, побудованої з урахуванням гнучких вимог користувача та орієнтованої на ефективне адміністрування замовлень, оплат і меню.

1.3 Аналіз програмних продуктів-аналогів

Автоматизація процесів оформлення замовлень у закладах громадського харчування є одним із ключових напрямів цифрової трансформації індустрії HoReCa (Hotel, Restaurant, Catering). Ефективна організація інформаційних

потоків, оперативне обслуговування клієнтів, гнучкість адміністрування меню та інтеграція з платіжними системами сьогодні розглядаються не лише як конкурентна перевага, а як обов’язкова умова забезпечення стабільної роботи підприємства в умовах високої ринкової конкуренції.

На сучасному етапі розвитку IT-інфраструктури на ринку представлено низку як універсальних, так і вузькоспеціалізованих рішень для автоматизації ресторанного бізнесу. Ці програмні продукти відрізняються між собою за такими параметрами, як обсяг функціональності, архітектурні особливості, можливості масштабування, ступінь інтеграції з зовнішніми сервісами, а також моделлю монетизації (одноразова ліцензія, підписка, комісійна модель тощо). Крім того, важливим критерієм вибору системи для підприємства громадського харчування є простота впровадження та навчання персоналу, що безпосередньо впливає на швидкість окупності інвестицій у цифрову інфраструктуру.

Одним із найпоширеніших рішень серед малих та середніх підприємств є система Poster POS[3]. Це хмарна платформа для автоматизації ресторанного обліку, яка надає широкий спектр функцій: управління асортиментом і меню, прийом та обробка замовлень, облік залишків товарів на складі, ведення фінансової аналітики, інтеграція з терміналами для безготівкових розрахунків, а також підтримка функцій лояльності. Використання архітектури типу "Software-as-a-Service" (SaaS) спрощує розгортання рішення, оскільки не потребує встановлення серверного обладнання на боці закладу. Проте модель оплати за передплатою передбачає регулярні фінансові витрати, що для малого бізнесу може бути критичним фактором при виборі системи. Додатковим обмеженням є залежність від інтернет-з’єднання та необхідність використання сертифікованих POS-пристроїв (рис. 3).

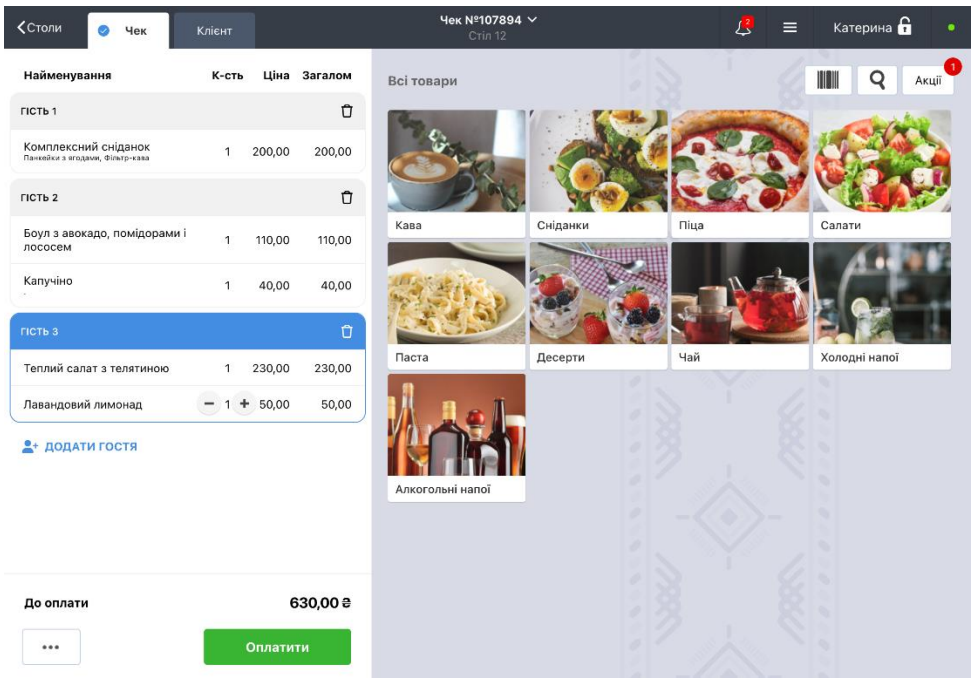


Рисунок 3 – *Інтерфейс системи Poster*

Іншим відомим комерційним продуктом є система R-Keeper, яка позиціонується як високонавантажена POS-платформа для великих мереж ресторанів та закладів швидкого обслуговування. Вона підтримує інтеграцію з корпоративними інформаційними системами, зокрема ERP-рішеннями, CRM-платформами, аналітичними модулями, сервісами доставки та електронними платіжними шлюзами. Однією з ключових переваг R-Keeper є гнучка модульна архітектура, що дозволяє адаптувати функціонал системи під конкретні бізнес-процеси. Проте складність налаштувань, потреба в сертифікованих інтеграторах та високі початкові витрати значно обмежують можливість використання цього рішення у малих і середніх закладах. Крім того, інтерфейс системи орієнтований на досвідчених користувачів і вимагає попереднього навчання персоналу (рис. 4).

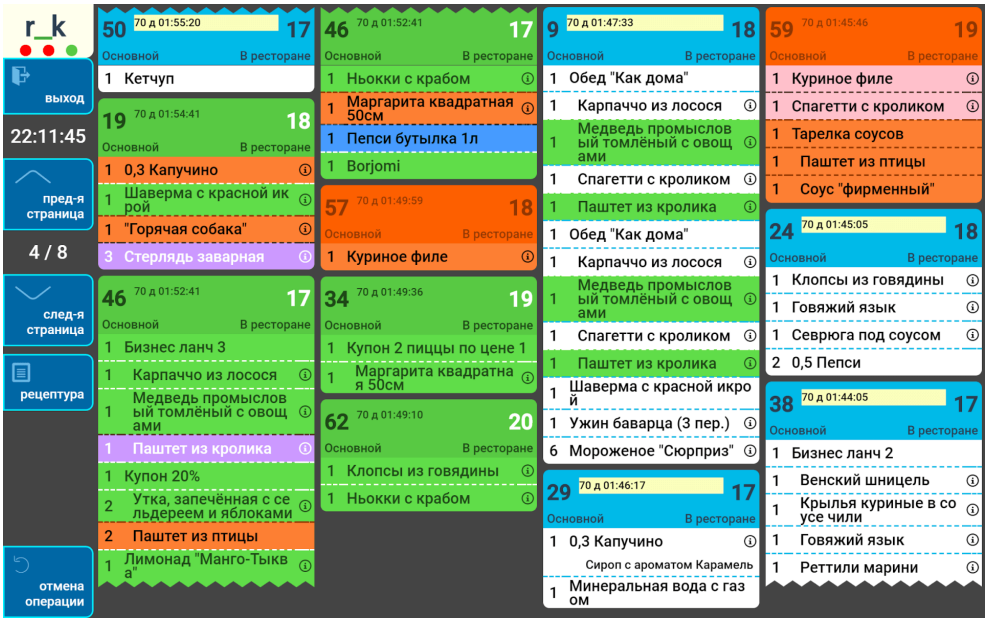


Рисунок 4 – *Інтерфейс R-Keeper*

Паралельно з професійними POS-рішеннями, стрімкого розвитку набувають мобільні сервіси доставки їжі, зокрема такі платформи, як Glovo, Bolt Food, Uber Eats. Вони орієнтовані переважно на кінцевого споживача, надаючи можливість переглядати меню, здійснювати онлайн-замовлення та проводити оплату через мобільні застосунки. Хоча такі сервіси суттєво розширюють аудиторію закладів, вони не включають функціональність для внутрішньої автоматизації, як-от налаштування меню, аналітика продажів чи інтеграція з фінансовими обліковими системами. Крім того, комісії за транзакції в цих системах можуть сягати 30% від вартості замовлення, що робить їхню економічну доцільність для малих гравців сумнівною (рис. 5).

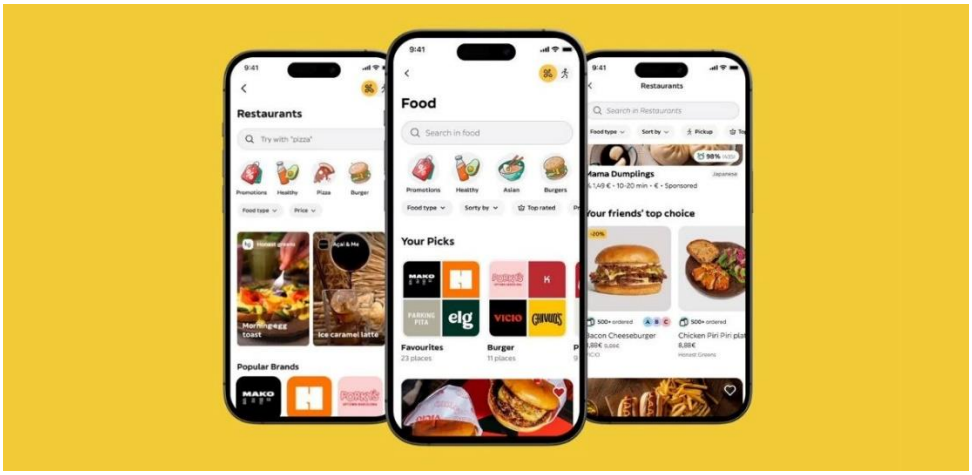


Рисунок 5 — *Мобільний інтерфейс Glovo*

У практиці мікропідприємств, таких як локальні кав’ярні чи фастфуди, досі використовуються імпровізовані системи обліку, створені на базі Google Forms, Microsoft Excel, або навіть паперових замовлень. Ці рішення не потребують витрат на ліцензії або обслуговування, однак мають обмежену функціональність, не забезпечують захищене зберігання інформації, не підтримують інтеграцію з платіжними системами та не дозволяють відстежувати статуси замовлень у реальному часі. Такі підходи можуть бути виправдані лише на стартовому етапі діяльності, але у довгостроковій перспективі істотно знижують ефективність бізнес-процесів.

Враховуючи результати проведеного аналізу, у межах даної кваліфікаційної роботи було розроблено власне рішення — веб-застосунок, який поєднує ключову функціональність сучасних POS-систем, орієнтований на потреби малих закладів харчування, та водночас не вимагає суттєвих фінансових інвестицій. Його функціональна модель включає:

- гнучке управління меню (додавання, редагування, категоризація позицій);
- формування замовлень користувачами через веб-інтерфейс;
- інтеграцію з платіжним сервісом LiqPay для здійснення електронних платежів;
- моніторинг статусу замовлень у реальному часі;

- адміністрування позицій через окрему панель керування;
- збереження даних у реляційній базі даних PostgreSQL з підтримкою регулярного резервного копіювання (рис. 6).

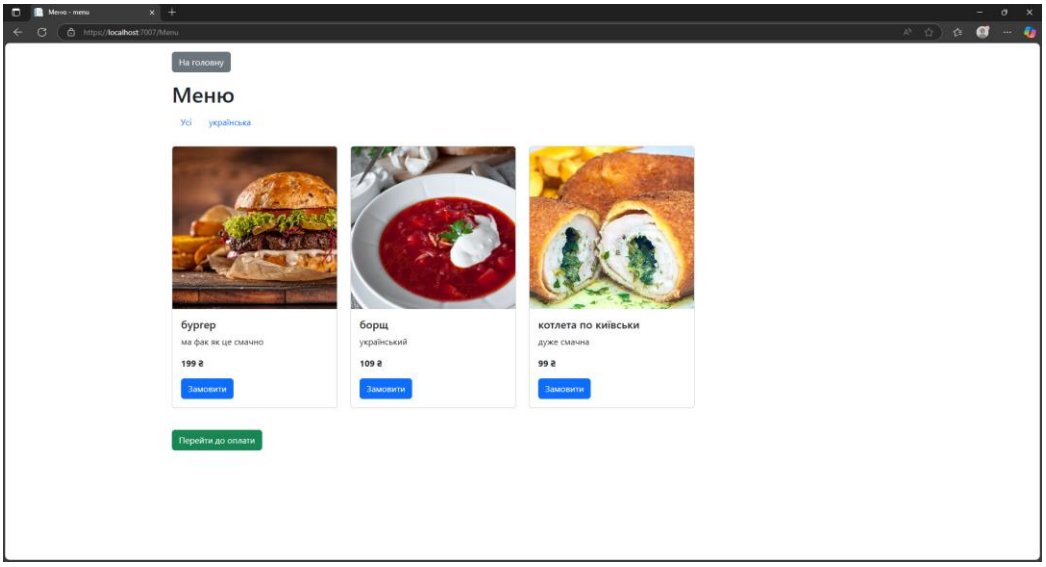


Рисунок 6 – *Інтерфейс веб-застосунку для користувача*

Таким чином, запропоноване рішення заповнює нішу між надто складними комерційними системами та примітивними саморобними інструментами, надаючи власникам невеликих закладів доступну та ефективну платформу для автоматизації процесів обслуговування клієнтів, управління меню та обробки платежів. Такий підхід дозволяє не лише оптимізувати поточну діяльність підприємства, але й створює основу для подальшого масштабування бізнесу, включаючи інтеграцію з мобільними застосунками, підключення аналітичних модулів або розширення до формату доставки.

1.4 **Постановка завдання**

Метою даної кваліфікаційної роботи є розробка веб-застосунку для автоматизації процесів прийому, обробки та оплати замовлень у закладах громадського харчування з урахуванням актуальних вимог до зручності

використання, безпеки обробки транзакцій та ефективності управління даними. Програмний продукт має забезпечувати безперервну взаємодію між клієнтом і підприємством шляхом реалізації інструментів для перегляду меню, формування замовлення, здійснення онлайн-оплати та подальшого адміністрування процесу обслуговування.

Досягнення поставленої мети передбачає реалізацію низки послідовних етапів:

1. Аналіз предметної області та вивчення аналогічних рішень. Першим кроком є вивчення специфіки предметної галузі та існуючих комерційних рішень (зокрема POS-систем, мобільних застосунків, онлайн-сервісів доставки). У результаті аналізу визначаються типові функціональні компоненти, які є доцільними для включення до цільового застосунку. Здійснюється порівняння систем з точки зору гнучкості, вартості впровадження, архітектури, рівня автоматизації та масштабованості. Виявлені переваги та обмеження комерційних систем лягають в основу специфікації вимог до створюваного продукту.

2. Проєктування структури бази даних. Важливою передумовою коректної роботи системи є раціональна модель зберігання даних. Було обґрунтовано вибір реляційної СУБД PostgreSQL як основи для централізованого управління інформацією. Запроєктовано структуру бази даних, яка охоплює таблиці для зберігання інформації про страви, категорії, замовлення, платіжні чеки та пов’язані метадані. Особливу увагу приділено забезпеченню цілісності даних через зовнішні ключі, нормалізації структури, вибору відповідних типів даних та індексації з метою підвищення продуктивності запитів.

3. Реалізація серверної частини. На основі технологій ASP.NET Core реалізовано серверну логіку у вигляді RESTful API, що забезпечує централізовану обробку запитів від клієнтської частини. Основні

функціональні блоки охоплюють створення, читання, оновлення та видалення ресурсів (CRUD-операції), обробку замовлень, розрахунок сум, логіку оплати та оновлення статусів. Процес проектування API відбувався відповідно до принципів REST і OpenAPI-специфікацій, що забезпечує простоту інтеграції з майбутніми модулями або сторонніми сервісами.

4. Розробка клієнтського інтерфейсу. Інтерфейс користувача побудовано з використанням Razor Pages. Реалізовано низку представлень (views), що забезпечують навігацію по меню, додавання позицій до кошика, виведення інформації про поточне замовлення та ініціацію оплати. Інтерфейс адаптований до мобільних і десктопних пристроїв, дотримується принципів UX/UI дизайну, забезпечує перевірку введених даних (валидацію), обрахунок загальної вартості та формування візуального чеку. Передбачено асинхронну взаємодію з сервером у режимі реального часу через AJAX-запити.

5. Інтеграція платіжної системи. Для реалізації онлайн-оплати в систему інтегровано зовнішній платіжний сервіс LiqPay. Реалізовано обробку платіжних даних з використанням шифрування та токенізації. Підтримано механізм callback-повідомлень для автоматичного підтвердження транзакцій та оновлення статусу відповідного замовлення. У такий спосіб гарантується безпечна, прозора та швидка обробка фінансових операцій.

6. Очікувані результати впровадження. У результаті реалізації описаних етапів очікується створення функціонального, масштабованого та економічно обґрунтованого рішення, яке дозволить автоматизувати ключові бізнес-процеси невеликих підприємств у сфері харчування. Застосунок забезпечує централізований облік замовлень, оптимізує роботу персоналу, скорочує час обслуговування клієнтів та сприяє цифровій трансформації підприємства. Запропоноване рішення також знижує потребу у дорогому обладнанні та дозволяє адаптувати систему під індивідуальні потреби конкретного закладу.

2 ДОСЛІДЖЕННЯ ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Аналіз сучасних систем управління базами даних

Процес зберігання та опрацювання інформації є надзвичайно важливим для функціонування будь-якої інформаційної системи, особливо у сфері автоматизації обслуговування. Ефективність роботи програмного забезпечення, його стабільність, адаптивність до навантажень і довгострокова підтримка великою мірою залежать від правильно підібраної системи управління базами даних (СУБД). СУБД виступає як серце інформаційного застосунку, забезпечуючи централізоване управління структурованими даними, що переміщуються між модулями програми, інтерфейсами користувачів та зовнішніми сервісами.

СУБД — це програмне забезпечення, розроблене для створення, редагування, супроводу та керування базами даних. Основні функції СУБД включають:

- забезпечення структурного зберігання інформації відповідно до визначеної схеми;
- підтримку транзакційної цілісності;
- організацію одночасного доступу для кількох користувачів;
- захист та резервне копіювання;
- оптимізацію запитів та індексування;
- масштабовану обробку запитів при збільшенні обсягів даних.

Сучасні СУБД класифікуються за моделлю представлення та організації даних. Найбільш поширеними в розробницькому середовищі є реляційні СУБД, які працюють з табличною моделлю, та нереляційні (NoSQL) системи, що підтримують документоорієнтовану, графову, колонкову або ключ-значення структуру. Вибір тієї чи іншої категорії визначається специфікою

предметної області, вимогами до структурованості даних та передбачуваним масштабом системи.

Під час вибору бази даних для системи автоматизації закладів громадського харчування було визначено наступні основні критерії:

- Підтримка ACID-властивостей — забезпечення цілісності даних під час одночасної роботи з кількома транзакціями;
- Продуктивність — висока швидкість обробки запитів до таблиць меню, замовлень та рахунків;
- Масштабованість — здатність системи впоратися з збільшенням обсягів даних без значного зниження продуктивності;
- Гнучкість в проектуванні схеми даних — підтримка складних зв'язків між таблицями, зовнішніх ключів, індексів;
- Наявність активної спільноти — що спрощує розробку, супровід і вирішення типових проблем;
- Вартість використання — особливо критично для невеликих підприємств з обмеженим бюджетом.

Таблиця *1 — Порівняльний аналіз популярних СУБД*

СУБД	Тип	Особливості	Недоліки
PostgreSQL	Реляційна	Відкритий код, ACID, підтримка JSON, складних запитів, геоданих	Вимагає більше ресурсів при великій кількості одночасних запитів
MySQL	Реляційна	Простота у використанні, висока стабільність, хороша документація	Менш гнучкий синтаксис запитів, обмежена робота з транзакціями в InnoDB
Microsoft SQL Server	Реляційна	Глибока інтеграція з екосистемою Microsoft,	Комерційна ліцензія, складність у масштабуванні без Azure

СУБД	Тип	Особливості	Недоліки
		зручна адміністраторська панель	
Oracle DB	Реляційна	Потужна платформа для корпорацій, підтримка великих обсягів даних	Висока вартість, складність впровадження
MongoDB	NoSQL	Гнучка структура документів, JSON-подібні дані, висока масштабованість	Відсутність транзакційності, складна реалізація реляційних зв’язків

На основі вищевказаних критеріїв та проведеного аналізу, для реалізації цільового веб-застосунку було обрано PostgreSQL. Це повнофункціональна реляційна СУБД з відкритим вихідним кодом, яка поєднує гнучкість, безпеку, розширюваність та активну підтримку спільноти. PostgreSQL забезпечує:

- високу продуктивність при складних SQL-запитах завдяки ефективному планувальнику запитів;
- гарантовану цілісність даних через строге дотримання ACID-властивостей;
- гнучкість структури з можливістю комбінування реляційного і JSON-подібного зберігання (hybrid-schema);
- засоби безпеки — контроль доступу, SSL-з’єднання, аудит;
- легке масштабування через підтримку реплікації, партиціювання, шардінгу.

Крім того, PostgreSQL є повністю безкоштовною у використанні, що зменшує загальні витрати на реалізацію системи та робить її особливо привабливою для малих та середніх підприємств.

Таким чином, PostgreSQL було обрано як основну СУБД для веб-застосунку, оскільки вона повністю відповідає вимогам проекту щодо

надійності, швидкодії, підтримки складної схеми даних та адаптації до зростання навантаження. Її впровадження дозволяє забезпечити ефективне функціонування системи у довгостроковій перспективі, підтримувати високу якість обслуговування клієнтів, а також гарантувати збереження й захист всіх транзакцій та замовлень в цифровому середовищі.

2.2 Аналіз сучасних технологій для створення серверної частини

Серверна частина веб-застосунку є ключовим компонентом, що забезпечує реалізацію всієї бізнес-логіки системи, обробку даних, збереження інформації в базі даних, а також взаємодію з клієнтською частиною через інтерфейс прикладного програмування (API). Її ефективність безпосередньо впливає на загальну продуктивність застосунку, швидкість обробки запитів, надійність збереження замовлень, а також масштабованість системи в умовах зростання користувацької бази.

Серверна частина виконує наступні задачі:

- прийом та обробка HTTP-запитів;
- виконання CRUD-операцій з базою даних (створення, читання, оновлення, видалення);
- реалізація бізнес-логіки: обробка замовлень, формування чеків, інтеграція з платіжною системою;
- валідація даних, перевірка доступів та обробка помилок;
- надання уніфікованих відповідей у форматі JSON для клієнтської частини;
- забезпечення логування, аутентифікації, обробки помилок та масштабованості.

Після аналізу сучасних рішень для створення REST API-сервісів було обрано ASP.NET 8.0 як основну платформу для реалізації серверної логіки. Це сучасна версія популярного фреймворку від Microsoft, яка поєднує

продуктивність, стабільність і широкі можливості інтеграції. Серед ключових технічних переваг ASP.NET 8.0:

- Покращена продуктивність: оптимізоване середовище виконання .NET 8.0 забезпечує високу швидкість обробки запитів, зменшення часу відповіді (latency) та ефективне керування ресурсами.
- Асинхронність: підтримка async/await дозволяє реалізовувати неблокуючі операції з базою даних або сторонніми сервісами, що критично важливо для роботи з високонавантаженими застосунками.
- Безпечність: сучасні механізми захисту від SQL-ін’єкцій, CSRF, XSS забезпечуються "з коробки".
- Модульність і масштабованість: платформа підтримує Clean Architecture, Dependency Injection, CQRS, мікросервісну архітектуру, що спрощує майбутню підтримку.
- Інтеграція з EF Core: Entity Framework Core забезпечує ORM-рівень взаємодії з базою, суттєво спрощуючи створення, оновлення та тестування моделей даних.

У межах проєкту були реалізовані контролери, що обробляють запити за такими сутностями:

- Категорії страв — відображення ієрархії меню;
- Страви — CRUD-операції, збереження, оновлення, фільтрація;
- Замовлення — реєстрація нових замовлень, обчислення загальної суми;
- Чеки — збереження історії оплат, формування інформації для клієнта.

PostgreSQL як складова архітектури зберігання даних

Для збереження інформації обрана реляційна СУБД PostgreSQL, яка успішно поєднується з ASP.NET 8.0 через драйвер Npgsql та підтримує ORM через Entity Framework Core. Основні переваги PostgreSQL, актуальні для розробки:

- Підтримка транзакцій ACID, що гарантує цілісність і надійність збереження даних;

- Розширена підтримка типів даних (ENUM, JSONB, TIMESTAMP, UUID), що підходить для зберігання гнучких структур;
- Потужний рушій запитів, що дозволяє реалізовувати фільтрацію, агрегацію та складну аналітику (наприклад, частота замовлення страв);
- Механізми безпеки — контроль прав доступу, SSL-з’єднання, політики доступу на рівні рядків;
- Інтеграція з бекендом через ORM — за рахунок Entity Framework Core розробник працює безпосередньо з C#-об'єктами, а не з сирим SQL-кодом.

При проєктуванні бази даних була реалізована чітка схема з наступними таблицями:

- Categories — категорії страв;
- Dishes — окремі страви з цінами, доступністю, вагою, зображенням;
- Orders — замовлення з прив’язкою до користувача, дати та статусу;
- Receipts — чеки з даними про оплату, суму та метод.

Таким чином, обрана архітектура на основі ASP.NET 8.0 + PostgreSQL дозволяє забезпечити високу продуктивність, масштабованість, надійність та гнучкість у реалізації серверної логіки інформаційної системи для закладів громадського харчування. Її впровадження створює передумови для подальшого розвитку — підключення мобільних клієнтів, додавання аналітики, автоматичної генерації звітів або інтеграції з обліковими системами (наприклад, 1С або M.E.Doc).

За рахунок ефективного розподілу обов’язків між серверною частиною, клієнтським інтерфейсом і базою даних, реалізована система відповідає сучасним вимогам до цифрових рішень у сфері HoReCa (Hotel, Restaurant, Catering) і може бути адаптована до будь-якого масштабу бізнесу — від локального кафе до мережевого закладу.

2.3 Аналіз сучасних технологій для створення клієнтської частини

У структурі розробленого веб-застосунку клієнтська частина виконує не лише функцію відображення даних, але і забезпечує повноцінну інтерактивну взаємодію користувача з системою. До основних функціональних можливостей фронтенду належать формування замовлення, перегляд меню, ініціація оплати, отримання підтвердження про оформлення та збереження стану кошика в межах користувацької сесії. Особливу увагу також було приділено адаптації інтерфейсу для мобільних пристроїв, що відповідає сучасним вимогам до зручності використання.

Для реалізації клієнтської частини було обрано технологію Razor Pages, яка входить до складу ASP.NET 8.0. Такий вибір обумовлений можливістю побудови єдиного технологічного рішення без необхідності виділення окремого фронтенд-застосунку, що, у свою чергу, дозволяє уникнути складної конфігурації взаємодії між клієнтом і сервером, зокрема у контексті авторизації, передачі сесій та запитів до API.

Razor Pages забезпечує спрощену модель програмування, що дозволяє чітко розділяти логіку та представлення в межах однієї сторінки. Завдяки серверному рендерингу HTML-контент передається клієнту без затримок, пов’язаних з ініціалізацією SPA-фреймворків. Це особливо важливо для невеликих бізнес-застосунків, де критичними є швидкість завантаження сторінок та мінімальне споживання ресурсів клієнтського пристрою [5].

Інтеграція JavaScript і AJAX дозволяє реалізовувати часткове оновлення сторінок — наприклад, під час додавання страви до кошика, оновлення суми замовлення або підтвердження дій користувача без повного перезавантаження. Завдяки тісній взаємодії Razor Pages з модельно-орієнтованим підходом ASP.NET, передача даних між клієнтським інтерфейсом і серверною логікою здійснюється напряму, що суттєво спрощує реалізацію складних сценаріїв взаємодії.

Важливим чинником на користь Razor Pages є зручна інтеграція із системами авторизації ASP.NET Core, що дозволяє ефективно реалізувати сесійну підтримку, зберігання кошика замовлень, а також контроль доступу до адміністративних функцій відповідно до ролі користувача.

На етапі вибору технології було також проведено порівняльний аналіз з іншими сучасними фреймворками, зокрема React, Angular та Vue. Попри їхню функціональну потужність і широке розповсюдження в розробці SPA, використання цих фреймворків у рамках цього проєкту передбачало б значне ускладнення архітектури, потребу у розгортанні окремого клієнтського застосунку, реалізації CORS-політик, налаштуванні механізмів взаємодії між API та клієнтом, а також синхронізації сесій.

Таким чином, Razor Pages виявляється найбільш ефективним рішенням, що забезпечує високу швидкість розробки, простоту підтримки, мінімальні накладні витрати та гнучку взаємодію з іншими компонентами системи.

Реалізація клієнтської частини за допомогою Razor Pages виявилася оптимальним рішенням у межах поставлених вимог. Такий підхід забезпечує ефективну інтеграцію з бекендом, високу продуктивність, зручну роботу з сесіями та авторизацією, а також значно спрощує розгортання і підтримку системи. У результаті було створено надійний, швидкий і функціональний інтерфейс, адаптований до потреб користувачів закладу громадського харчування.

2.4 Огляд інших технологій

У межах розробки програмного забезпечення використовувалися також допоміжні інструменти, які не брали безпосередньої участі у створенні програмного коду, проте відіграли важливу роль на етапах проєктування, планування та комунікації. Зокрема, до таких технологій належать системи візуального моделювання та інструменти для розробки інтерфейсів користувача.

Одним із ключових засобів візуального проєктування стала система Diagrams.net (раніше відома як *draw.io*) . Це веборієнтований застосунок, що забезпечує можливість створення структурних діаграм, UML-схем, прототипів архітектури та інших графічних представлень. Його використання дозволило візуалізувати логічну структуру системи, зокрема взаємозв'язки між сутностями, модель бази даних та загальну архітектуру програмного сервісу. Такий підхід значною мірою полегшив розуміння складних технічних концепцій, підвищив ефективність планування та сприяв продуктивнішому обговоренню технічних рішень із науковим керівником. Наявність зручного інтерфейсу, інструментів для спільної роботи та підтримка імпорту/експорту у різних форматах зробили цей застосунок особливо придатним для використання в освітніх і дослідницьких цілях.

Для розробки дизайну інтерфейсу майбутнього веб-застосунку використовувався інструмент Figma, що є сучасним вебзастосунком для створення прототипів, макетів інтерфейсів та інтерактивних елементів. Його функціональні можливості включають векторне редагування, побудову інтерактивних прототипів, підтримку бібліотек компонентів і стилів, що дозволяє досягти високої консистентності та візуальної привабливості інтерфейсу. Особливу увагу було приділено адаптивності дизайну, що забезпечило коректне відображення інтерфейсу як на десктопних, так і на мобільних пристроях. У процесі розробки Figma дозволила швидко візуалізувати структуру майбутнього інтерфейсу, провести початкову валідацію UX-рішень та забезпечити безперервний зворотний зв'язок під час реалізації проєкту.

Таким чином, використання Diagrams.net та Figma дало змогу ефективно реалізувати етапи концептуального проєктування та візуального оформлення інформаційної системи, що є важливою складовою її загального успіху.

3 РОЗРОБКА АНАЛІТИЧНОГО СЕРВІСУ ВИБОРУ ТОВАРІВ

3.1 Опис предметної області

Предметна область, в межах якої реалізується даний проєкт, охоплює сферу автоматизації бізнес-процесів у закладах громадського харчування, що входить до галузі HoReCa (Hotel/Restaurant/Café). Основним вектором дослідження є створення інформаційної системи, здатної оптимізувати процеси обробки замовлень, керування асортиментом продукції, здійснення оплат та обліку замовлень, що, у свою чергу, сприяє підвищенню ефективності функціонування таких закладів.

З огляду на стрімку цифрову трансформацію сфери обслуговування, автоматизація діяльності кафе, ресторанів та інших об’єктів харчування набуває особливої актуальності. Відсутність технологічних рішень у цій площині часто зумовлює виникнення системних проблем, таких як зниження швидкості обслуговування, зростання кількості помилок у замовленнях, незадовільна координація між персоналом та клієнтами, що безпосередньо впливає на рівень задоволеності відвідувачів і фінансові показники закладу.

Розглянута предметна область передбачає реалізацію низки ключових функціональних складових, серед яких:

- управління номенклатурою меню, включаючи створення, редагування та категоризацію страв;
- забезпечення можливості формування замовлення через клієнтський веб-інтерфейс;
- інтеграція з платіжними сервісами з метою здійснення безготівкових оплат;
- формування чеків та надання інформації про статус виконання замовлення;

- збереження та обробка інформації у базі даних, що включає деталі про страви, замовлення та платежі;
- адміністративна панель для моніторингу замовлень, аналізу активності та управління контентом меню.

Метою розробки є створення багатофункціонального, масштабованого та зручного у користуванні веб-застосунку, здатного забезпечити ефективну взаємодію між клієнтами та обслуговуючим персоналом. Реалізація такого рішення покликана автоматизувати ключові бізнес-процеси, мінімізувати ручну працю персоналу, скоротити час обробки замовлень та покращити загальну якість сервісу.

Впровадження подібних систем, що інтегрують функціональність онлайн-замовлень, цифрової обробки платежів і зручного управління меню, стає не лише конкурентною перевагою для великих гравців ринку, а й необхідною умовою ефективної діяльності малих і середніх закладів громадського харчування, які прагнуть відповідати сучасним стандартам обслуговування.

3.2 Вимоги до системи

Розробка програмного забезпечення передбачає формалізацію чітко визначених вимог, що охоплюють як функціональні, так і нефункціональні характеристики системи. Вони слугують вихідною точкою для проєктування архітектури, визначення технічних засобів реалізації, планування етапів впровадження та подальшого тестування. Визначення таких вимог є критичним для досягнення узгодженості між очікуваннями кінцевих користувачів, бізнес-цілями та технічними обмеженнями.

У контексті створення веб-застосунку для автоматизації процесу замовлень у закладах громадського харчування сформульовано перелік

функціональних та нефункціональних вимог, що забезпечують його коректну та ефективну роботу. Функціональні вимоги:

- Перегляд меню — система повинна забезпечувати доступ користувачів до перегляду категорій та страв із відображенням опису, ціни та доступності.
- Формування замовлення — користувач має змогу додавати страви до кошика, змінювати їх кількість та переглядати загальну суму до оплати.
- Оформлення замовлення — реалізується можливість підтвердження замовлення із заповненням необхідної контактної інформації.
- Онлайн-оплата — необхідна інтеграція з платіжною системою (зокрема, LiqPay) для забезпечення безпечної та зручної оплати.
- Генерація чеку — після успішної оплати користувач отримує цифровий чек із детальною інформацією про замовлення.
- Панель адміністратора — передбачена функціональність для управління контентом меню (додавання, редагування, видалення страв і категорій).
- Збереження замовлень — система зберігає усі замовлення в базі даних з повним набором реквізитів: статус, сума, перелік страв, дата та час.
- Керування статусами — адміністратор має змогу змінювати статус замовлення (наприклад, «прийнято», «виконано») через веб-інтерфейс.
- Підтримка сесій — дані кошика повинні зберігатися між HTTP-запитами без необхідності авторизації за допомогою механізмів сесій ASP.NET.
- Адаптивний дизайн — користувацький інтерфейс має коректно відображатися як на стаціонарних, так і на мобільних пристроях.

Нефункціональні вимоги:

- Продуктивність — застосунок повинен забезпечувати високу швидкодію та мінімальні затримки навіть за умов інтенсивного навантаження.

- Надійність — система має функціонувати стабільно, без збоїв, протягом тривалого часу експлуатації.
- Безпека — реалізація має включати захист переданих даних (зокрема при оплаті) та контроль доступу до адміністративних функцій.
- Масштабованість — архітектура повинна дозволяти подальше розширення функціоналу (наприклад, авторизація, аналітика).
- Інтуїтивність інтерфейсу — користувач повинен мати можливість легко орієнтуватися в застосунку, виконувати базові операції без додаткових інструкцій.
- Модульність — серверна частина повинна мати чітко визначену структуру з розділенням логіки на окремі компоненти (контролери, сервіси, репозиторії).
- Відокремлення важкої логіки — ресурсоємні операції, як-от масове імпортування страв з CSV-файлів, виконуються у фоновому режимі, не блокуючи основний потік API.

3.3 Архітектура системи

Система веб-застосунку для автоматизації замовлень у закладах харчування реалізована за архітектурною моделлю клієнт–сервер. Такий підхід забезпечує розподіл відповідальностей між клієнтською частиною (веб-інтерфейсом) і серверною частиною (бізнес-логікою, API, базою даних), що сприяє підвищенню гнучкості, масштабованості, безпеки та зручності розробки й обслуговування системи.

У клієнт-серверній моделі клієнтська частина надсилає HTTP-запити до сервера, який обробляє ці запити, взаємодіє з базою даних, виконує необхідні обчислення або логіку та формує відповідь. Така структура дозволяє легко реалізувати додаткові функціональні можливості, здійснювати оновлення без зміни клієнтського коду, а також централізовано забезпечувати безпеку та обробку помилок. Основні складові системи:

База даних (PostgreSQL): Відповідає за зберігання всіх структурованих даних: інформації про страви, категорії, замовлення, платежі, статуси тощо. Організована у вигляді зв’язаних реляційних таблиць із підтримкою індексації, транзакцій та цілісності даних.

Серверна частина (бекенд на ASP.NET 8.0): Реалізує бізнес-логіку застосунку, обробляє запити клієнта, взаємодіє з базою даних через ORM (Entity Framework Core), формує відповіді у форматі HTML або JSON. Включає API-контролери, сервіси, репозиторії, механізми обробки сесій, автентифікації (для адміністративного інтерфейсу) та інтеграції з платіжними сервісами.

Клієнтська частина (фронтенд на Razor Pages): Відповідає за відображення інтерфейсу, взаємодію з користувачем, надсилання запитів до бекенду (включно з AJAX-оновленням без перезавантаження сторінки) та візуалізацію результатів. Завдяки Razor Pages логіка представлення тісно інтегрована з контролерами та моделями ASP.NET

3.3.1 Огляд бази даних

У системі збереження інформації про товари ключову роль відіграє база даних, що забезпечує централізоване зберігання даних про продукти, магазини, категорії та їх взаємозв’язки. Відповідна структура даних зображена на рисунку 7.

База даних містить чотири основні сутності, що формують логічну модель системи: торговельні мережі (markets), категорії товарів (category), унікальні товари (unique_products) та відповідність товарів конкретним магазинам (market_product). Таблиця, яка зберігає інформацію про магазини, містить унікальний ідентифікатор та назву кожного з них. Категорії представлені у вигляді окремої таблиці з власним ідентифікатором, назвою та унікальним шляхом, що може використовуватись для побудови ієрархії чи фільтраційного пошуку.

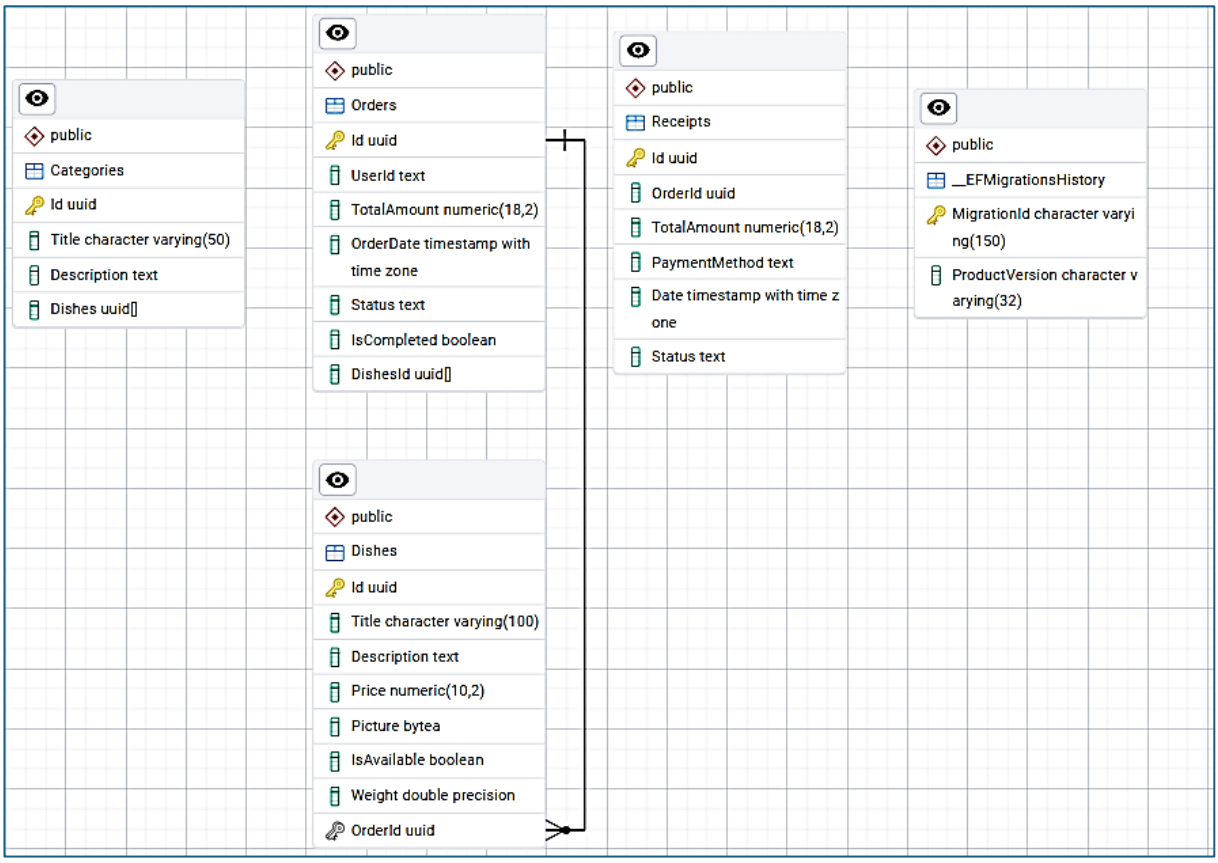


Рисунок 7 – *Схема бази даних для зберігання інформації про товари*

Ключовим елементом є таблиця унікальних товарів, яка містить повну інформацію про кожну товарну позицію, включаючи назву, бренд, ознаку обмеження за віком, шлях до зображення, а також масив категорій, до яких належить товар. Такий підхід дозволяє забезпечити коректну класифікацію товарів та їх подальшу обробку в рамках сервісу.

Зв’язок між товарами та магазинами реалізується через таблицю відповідностей, яка окрім ідентифікаторів магазину та товару містить інформацію про ціну в конкретному магазині, одиницю виміру, а також приведену ціну у стандартизованих одиницях для можливості подальшого порівняння.

Особливістю такої структури є її придатність до масштабування, підтримка порівняння цін між різними торговельними мережами, а також можливість гнучкої фільтрації та пошуку. Саме завдяки таблиці

unique_products, яка зберігає узагальнені позиції товарів, система здатна здійснювати порівняльний аналіз асортименту між різними магазинами, незалежно від їхніх внутрішніх категоризацій. Це підвищує точність обробки даних і робить систему ефективним інструментом для аналітики.

3.3.2 Огляд серверної частини

Серверна частина програмного забезпечення реалізована здебільшого у вигляді монолітного застосунку, в якому всі основні компоненти та логічні шари об’єднані в єдину кодову базу та функціонують як один цілісний процес. Такий підхід дозволяє суттєво спростити початкову фазу розробки, усунути складнощі, пов’язані з міжсервісною взаємодією, а також забезпечити централізований контроль над бізнес-логікою системи.

Монолітна архітектура передбачає, що вся система компілюється як єдине ціле, а обмін інформацією між модулями відбувається через прямі виклики методів. База даних функціонує як спільний компонент, що обслуговує всі модулі, а розгортання виконується у формі єдиного програмного екземпляра.

Водночас у проєкті реалізовано окремі елементи, притаманні мікросервісній архітектурі, шляхом логічного розділення структури застосунку на модулі з чітко визначеними сферами відповідальності. Таке структурування забезпечує кращу організацію коду, спрощує тестування та створює підґрунтя для можливого масштабування у майбутньому.

Структура серверної частини передбачає наявність модулів для обробки API-запитів, роботи з моделями даних, взаємодії з базою даних за допомогою ORM (Entity Framework Core), а також компоненти для опису інтерфейсів, конфігураційних розширень, міграцій бази даних і сторінок користувацького інтерфейсу. Усі ці елементи впорядковані відповідно до архітектурних принципів модульності та інверсії залежностей, що наведено на рис. 8.

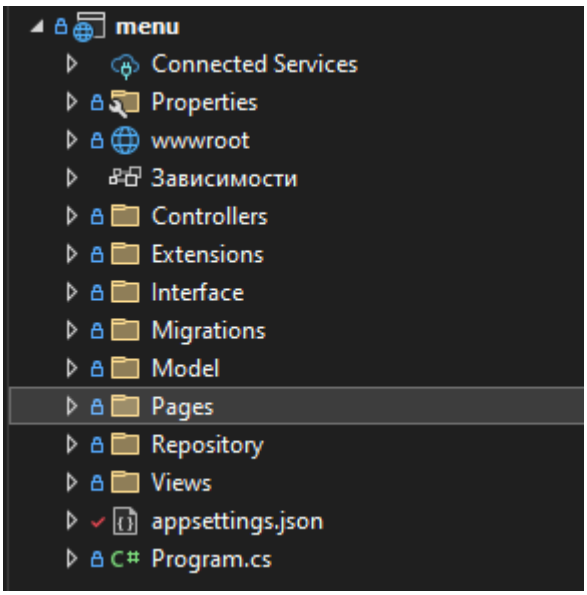


Рисунок 8 – *Структура проєкту серверної частини*

На даному етапі такий гібридний підхід — поєднання централізованої структури з елементами логічної декомпозиції — є доцільним з огляду на продуктивність, зручність розгортання і перспективу майбутньої еволюції системи. Перехід до повноцінної мікросервісної моделі, у разі потреби, передбачатиме винесення окремих функціональних підсистем (наприклад, модуля управління замовленнями або аутентифікації) в самостійні сервіси з подальшою реалізацією взаємодії через HTTP або gRPC. Також це вимагатиме автономного розгортання кожного компонента із використанням інструментів контейнеризації, таких як Docker чи Kubernetes.

3.3.3 Огляд клієнтської частини

Під час розробки клієнтської частини веб-застосунку були використані принципи шаблону MVVM (Model–View–ViewModel), адаптовані до функціональних можливостей фреймворку ASP.NET Razor Pages. Попри те, що Razor Pages не реалізує повноцінну MVVM-модель у класичному розумінні, структура проєкту забезпечує логічне розділення коду на представлення, модель та обробку подій користувача.

Роль представлення виконує .cshtml-розмітка, яка відображає інтерфейс користувача та підтримує прив’язку до моделі через допоміжні елементи (Tag

Helpers). У ролі ViewModel виступають .cshtml.cs-файли, які реалізують логіку обробки сторінки, взаємодіють із сервісами, обробляють введення та відповідають за стан сторінки. Класи моделей, представлені у вигляді DTO, забезпечують передачу структурованих даних між шарами застосунку.

Такий підхід дозволяє ефективно підтримувати код, забезпечує можливість модульного тестування сторінкової логіки (PageModel), а також сприяє розділенню обов'язків між інтерфейсом та бізнес-логікою. Механізми прив'язки даних та серверного рендерингу забезпечують двосторонню взаємодію між формами введення та об'єктами моделей.

Усі сторінки клієнтської частини згруповано в межах папки Pages (рис. 8), що забезпечує структурну організацію за функціональними ознаками. Це включає представлення меню, формування замовлення, оформлення та підтвердження оплати, а також інтерфейс адміністрування для керування даними.

Збереження стану реалізовано із використанням стандартного механізму ISession або через ін'єкцію залежностей для сервісів, що відповідають за роботу з відповідними даними. Ключова логіка взаємодії з такими даними, як поточне замовлення, категорії, перелік товарів або вибір магазину, винесена до окремих сервісів, які використовуються у відповідних сторінкових моделях. Таке рішення дозволяє зберігати слабке зв'язування між компонентами, підтримує масштабованість системи та забезпечує повторне використання логіки в межах різних сторінок інтерфейсу.

Загалом, адаптація принципів MVVM у контексті Razor Pages продемонструвала свою ефективність. Чітке розділення обов'язків, ізоляція логіки від представлення, використання сервісів для роботи з даними та централізоване управління станом суттєво підвищили стабільність, читабельність і підтримуваність застосунку.

3.4 Розробка бази даних

Під час розробки програмного забезпечення для автоматизації роботи закладу харчування як систему управління базами даних було обрано PostgreSQL.

Структура бази даних відображає логіку предметної області та складається з чотирьох основних сутностей: категорії страв, страви, замовлення та чеки. Для побудови таблиць використовувалась мова SQL. У наведених нижче лістингах представлено основні команди для створення відповідних структур.

Лістинг *1 — створення таблиці категорій страв*

```
create table categories (
    id uuid primary key,
    title varchar(50) not null,
    description text,
    dishes uuid[]
);
```

Таблиця categories містить перелік категорій, до яких належать страви. Поле dishes містить масив ідентифікаторів страв, що дозволяє відобразити зв’язок «один-до-багатьох».

Лістинг 2 — створення таблиці страв

```
create table dishes (
    id uuid primary key,
    title varchar(100) not null,
    description text,
    price decimal(18,2) not null,
    picture bytea,
    is_available boolean default true,
    weight double precision
);
```

У таблиці dishes зберігається інформація про кожну страву, включаючи назву, опис, ціну, наявність, зображення та вагу.

Лістинг 3 — створення таблиці замовлень

```
create table orders (
    id uuid primary key,
    user_id text,
    total_amount decimal(18,2),
    order_date timestamp default now(),
    status text default 'В обробці',
    is_completed boolean default false,
    dishes_id uuid[],
);
```

Таблиця orders фіксує інформацію про замовлення користувачів. Крім основних полів, зберігається масив ідентифікаторів страв, що були включені до кожного замовлення.

Лістинг 4 — створення таблиці чеків

```
create table receipts (
    id uuid primary key,
    order_id uuid not null references orders(id),
    total_amount decimal(18,2),
    payment_method text not null,
    date timestamp default now(),
    status text default 'Оплачено'
);
```

Таблиця receipts використовується для обліку оплати кожного замовлення. Вона містить посилання на відповідне замовлення, суму, метод оплати, дату та статус.

Така структура забезпечує ефективне зберігання та обробку інформації про замовлення, дозволяє швидко отримувати агреговані дані, підтримує

масштабованість та служить основою для реалізації серверної логіки й API-запитів, необхідних для інтеграції з інтерфейсом користувача.

3.5 Програмна реалізація серверної частини

Для реалізації серверної частини було обрано мову програмування C#, яка функціонує на сучасній платформі .NET 8.0. C# є строго типізованою, об’єктно-орієнтованою мовою, що забезпечує високу надійність, масштабованість та безпеку під час розробки вебзастосунків.

У якості вебфреймворку використано ASP.NET Core, який надає потужні засоби для побудови RESTful API, інтеграції з базами даних, реалізації механізмів аутентифікації та авторизації, а також підтримує створення клієнтського інтерфейсу за допомогою Razor Pages. Завдяки цьому фреймворку вдалося реалізувати ефективну серверну логіку, що відповідає сучасним вимогам до продуктивності та підтримки.

3.5.1 Розробка пакету db-core

Модуль db-core виконує ключову роль у серверній частині програмного забезпечення, оскільки відповідає за взаємодію з базою даних PostgreSQL. Через цей модуль реалізовано логіку створення, читання, оновлення та видалення (CRUD-операції) даних для основних сутностей системи: категорій, страв, замовлень і чеків.

Модуль інтегровано до основного ASP.NET Core Web Application, при цьому його структура організована відповідно до принципів розділення відповідальностей. Основними складовими модуля є:

- Model/ — класи сутностей, що відповідають таблицям бази даних;
- Interface/ — інтерфейси доступу до даних;
- Repository/ — реалізації інтерфейсів з використанням Entity Framework Core;

- AppDbContext.cs — контекст бази даних, через який здійснюється доступ до таблиць.

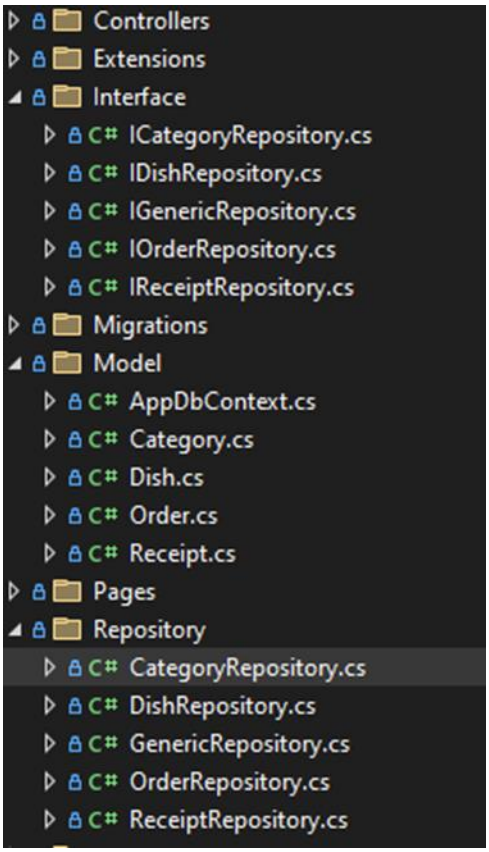


Рисунок 9 – *Структура модуля db-core*

Кожна сутність (наприклад, Category) представлена:

- відповідною моделлю (Category.cs);
- інтерфейсом для репозиторію (ICategoryRepository.cs);
- реалізацією репозиторію (CategoryRepository.cs).

Такий підхід дозволяє чітко відокремити бізнес-логіку від логіки доступу до даних, спрощує масштабування системи та її підтримку.

Лістинг 1 — Модель Category

```
public class Category
{
    public Guid Id { get; set; }

    [Required, MaxLength(50)]
    public string Title { get; set; }
```

```
        public string Description { get; set; }
        public List<Guid> Dishes { get; set; } = new();
    }
```

Клас Category описує сутність категорії страв. Завдяки суворій типізації властивостей забезпечується цілісність даних.

Лістинг 2 — Інтерфейс ICategoryRepository

```
    public interface ICategoryRepository
    {
        Task<IEnumerable<Category>> GetAllAsync();
        Task<Category?> GetByIdAsync(Guid id);
        Task AddAsync(Category category);
        Task UpdateAsync(Category category);
        Task DeleteAsync(Guid id);
    }
```

Цей інтерфейс визначає стандартні методи для доступу до категорій, забезпечуючи узгоджений контракт для реалізації.

Лістинг 3 — Реалізація CategoryRepository

```
    public class CategoryRepository : ICategoryRepository
    {
        private readonly AppDbContext _context;

        public CategoryRepository(AppDbContext context)
        {
            _context = context;
        }

        public async Task<IEnumerable<Category>>
GetAllAsync()
        {
```

```
            return await
_context.Categories.ToListAsync();
        }

        public async Task<Category?> GetByIdAsync(Guid id)
        {
            return await
_context.Categories.FindAsync(id);
        }

        public async Task AddAsync(Category category)
        {
            await _context.Categories.AddAsync(category);
            await _context.SaveChangesAsync();
        }

        public async Task UpdateAsync(Category category)
        {
            _context.Categories.Update(category);
            await _context.SaveChangesAsync();
        }

        public async Task DeleteAsync(Guid id)
        {
            var category = await
_context.Categories.FindAsync(id);
            if (category is not null)
            {
                _context.Categories.Remove(category);
                await _context.SaveChangesAsync();
            }
        }
    }
```

Реалізація використовує асинхронні методи для забезпечення масштабованості та ефективної роботи з базою даних. Застосування DbContext дозволяє уникнути прямої залежності від фізичної структури БД та реалізувати ORM-стиль взаємодії.

Лістинг 4 — Контекст бази даних (AppDbContext)

```
public class AppDbContext : DbContext
{
    public DbSet<Category> Categories { get; set; }
    public DbSet<Dish> Dishes { get; set; }
    public DbSet<Order> Orders { get; set; }
    public DbSet<Receipt> Receipts { get; set; }

    public  AppDbContext(DbContextOptions<AppDbContext>
options) : base(options) { }
}
```

Контекст конфігурується в Program.cs за допомогою впровадження залежностей (Dependency Injection), що спрощує інтеграцію з іншими частинами системи.

Основні переваги реалізації модуля db-core:

- централізація логіки доступу до даних;
- підтримка повного спектру CRUD-операцій;
- можливість повторного використання коду в інших частинах системи;
- зручність модульного тестування;
- розширюваність — можливість додавання нових репозиторіїв, фільтрацій, пагінації тощо.

Таким чином, модуль db-core є фундаментом зберігання й обробки даних у межах програмного комплексу. Його реалізація відповідає принципам чистої архітектури та сприяє підвищенню надійності й гнучкості програмного забезпечення.

3.5.2 Розробка пакету API

Після створення категорій, страв і замовлень за допомогою адміністративної частини системи виникає необхідність надати доступ до цієї інформації зовнішнім модулям або інтерфейсу користувача. Для цього в межах вебзастосунку реалізовано REST API за допомогою ASP.NET Core MVC, де контролери виступають посередниками між клієнтськими запитами та логікою доступу до бази даних.

У проєкті передбачено окрему директорію Controllers, яка містить такі основні контролери:

- CategoryController.cs — робота з категоріями;
- DishController.cs — обробка запитів щодо страв;
- OrderController.cs — створення та перегляд замовлень;
- ReceiptController.cs — отримання інформації про чеки.

Кожен із контролерів взаємодіє з відповідним репозиторієм, що інкапсулює доступ до бази даних через AppDbContext. Така структура відповідає принципам архітектури MVC та розділення відповідальностей (SoC).

Структуру API-контролерів, а також відповідні репозиторії, наведено на рис. 10. Вона демонструє логічну організацію коду у вигляді дерева файлів Visual Studio.

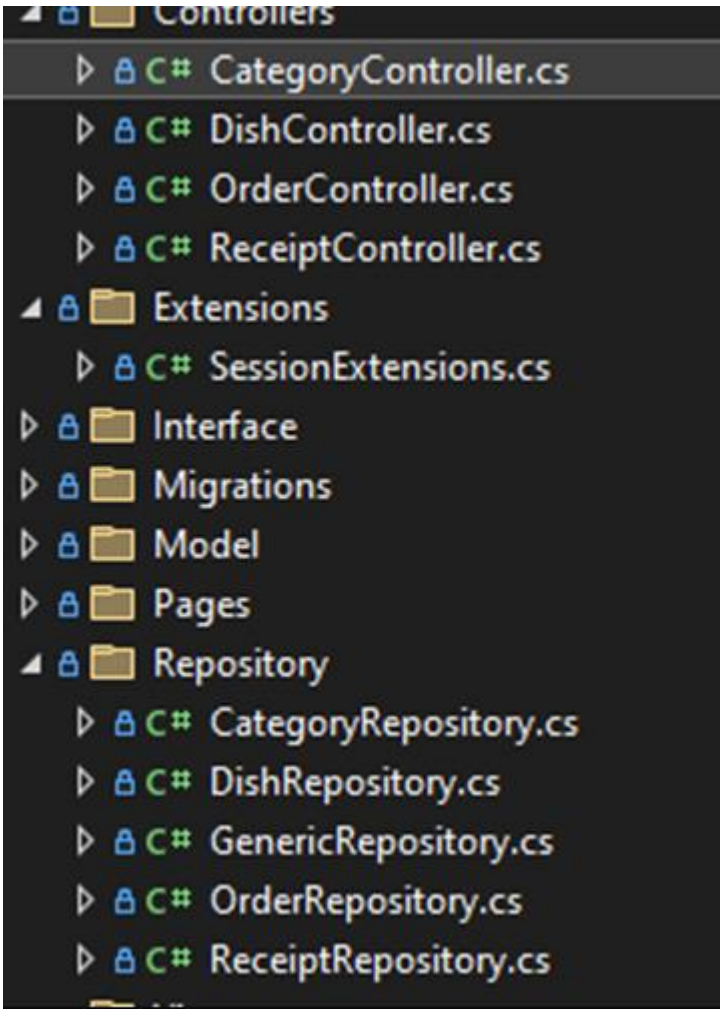


Рисунок 10 – *Структура API-контролерів системи*

Контролер CategoryController надає ендпоінти для роботи з категоріями. Один із базових методів виглядає так:

```
[HttpGet]
public async Task<IActionResult> GetAll()
{
    var categories = await
_categoryRepository.GetAllAsync();
    return Ok(categories);
}
```

Цей метод повертає перелік усіх категорій у форматі JSON у відповідь на HTTP GET-запит.

Аналогічно, у DishController реалізовано метод для отримання конкретної страви за ідентифікатором:

```
[HttpGet("{id}")]
public async Task<IActionResult> GetById(Guid id)
{
    var dish = await _dishRepository.GetByIdAsync(id);
    return dish != null ? Ok(dish) : NotFound();
}
```

Для розширення функціональності можуть бути реалізовані додаткові методи, наприклад, пошук:

```
[HttpGet("search")]
public async Task<IActionResult> Search(string name)
{
    var matches = await
_categoryRepository.SearchByNameAsync(name);
    return Ok(matches);
}
```

Усі запити до бази даних обробляються через відповідні репозиторії, наприклад:

```
public async Task<IEnumerable<Dish>> GetAllAsync()
{
    return await _context.Dishes.ToListAsync();
}
```

Це забезпечує інкапсуляцію доступу до даних та полегшує тестування і супровід.

Основні можливості, які реалізуються за допомогою контролерів, відповідають *use-case-діаграмі*, представленій на рис. 11. У цій діаграмі відображено взаємодію користувача із системою: створення та видалення категорій і страв, додавання до замовлення та його завершення.

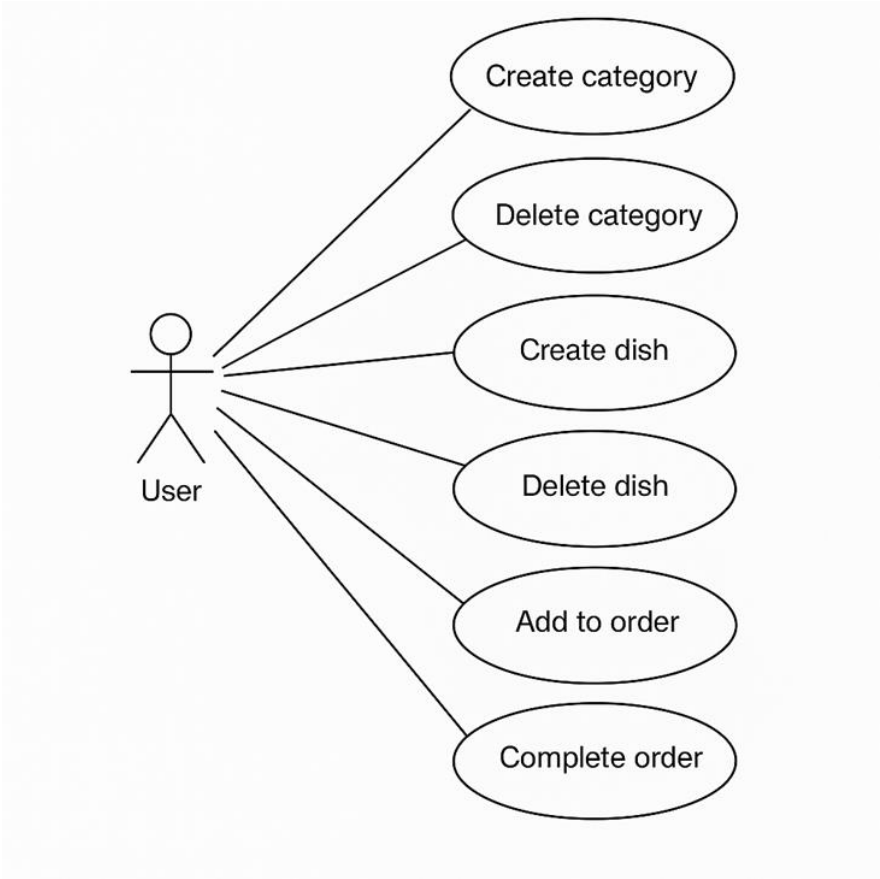


Рисунок 11 – *Відповідність до Use Case-діаграми API*

Переваги реалізованої архітектури API:

- Безпечний доступ до даних — безпосередній доступ до бази обмежено, усі дії проходять через контролери;
- Масштабованість — легко додавати нові ендпоінти для обслуговування нових сценаріїв;
- Гнучкість — можливість реалізації фільтрів, пагінації, DTO-об’єктів відповіді тощо;
- Прозорість і зрозуміла структура — чітке відображення кожного ресурсу (категорії, страви, замовлення, чеки) у вигляді окремого контролера.

Таким чином, реалізоване API забезпечує повноцінну, безпечну та ефективну взаємодію між клієнтською частиною та серверною логікою, дозволяючи обслуговувати запити користувачів та зовнішніх сервісів у межах сучасної архітектури вебзастосунку.

3.6 Розробка дизайну застосунку

Проєктування інтерфейсу користувача здійснювалося з урахуванням вимог юзабіліті, адаптивності та візуальної простоти. Основною метою було створення інтуїтивно зрозумілого середовища для взаємодії з меню, оформлення замовлень та здійснення оплати, що є критично важливим для закладів громадського харчування.

Для реалізації інтерфейсу було використано технологію Razor Pages, яка входить до складу ASP.NET Core. Такий підхід дозволяє чітко розділити логіку обробки запитів (у класах PageModel) від представлення (шаблони .cshtml). Це сприяє зменшенню складності коду, підвищенню модульності та полегшенню супроводу системи.

Головна сторінка вебзастосунку, зображена на *рисунку 12*, виконує роль точки входу до системи. Звідси користувач може перейти до перегляду меню ресторану.

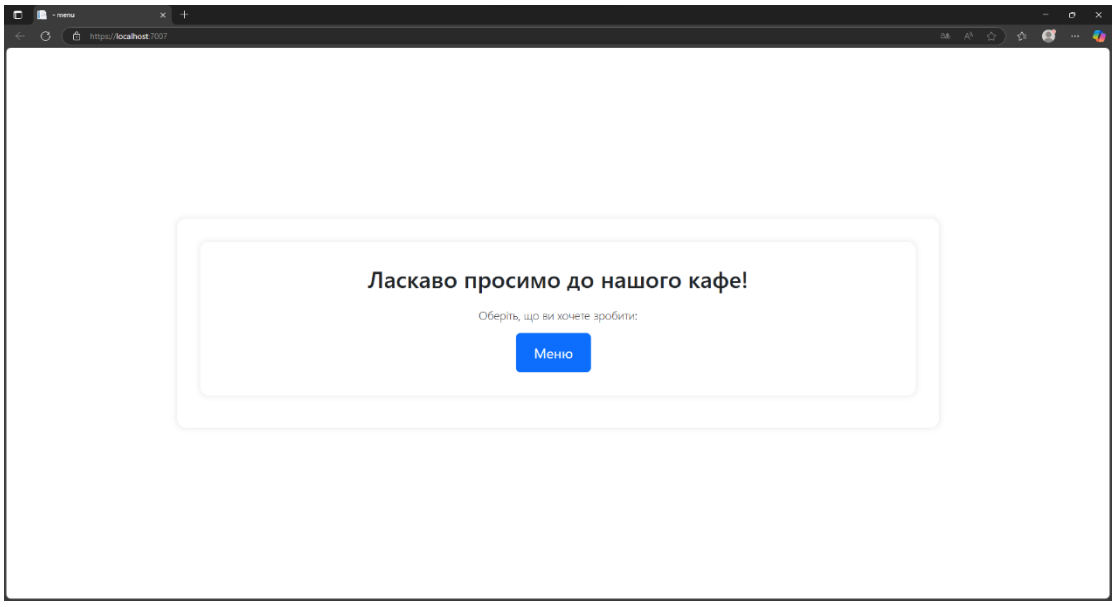


Рисунок 12 – *Головна сторінка застосунку*

Основна функціональність представлена на сторінці меню (*рис. 13*), яка виводить перелік доступних страв. Кожна страва супроводжується зображенням, назвою, описом і ціною. Для взаємодії передбачено кнопку «Замовити», яка додає відповідну позицію до поточного кошика користувача.

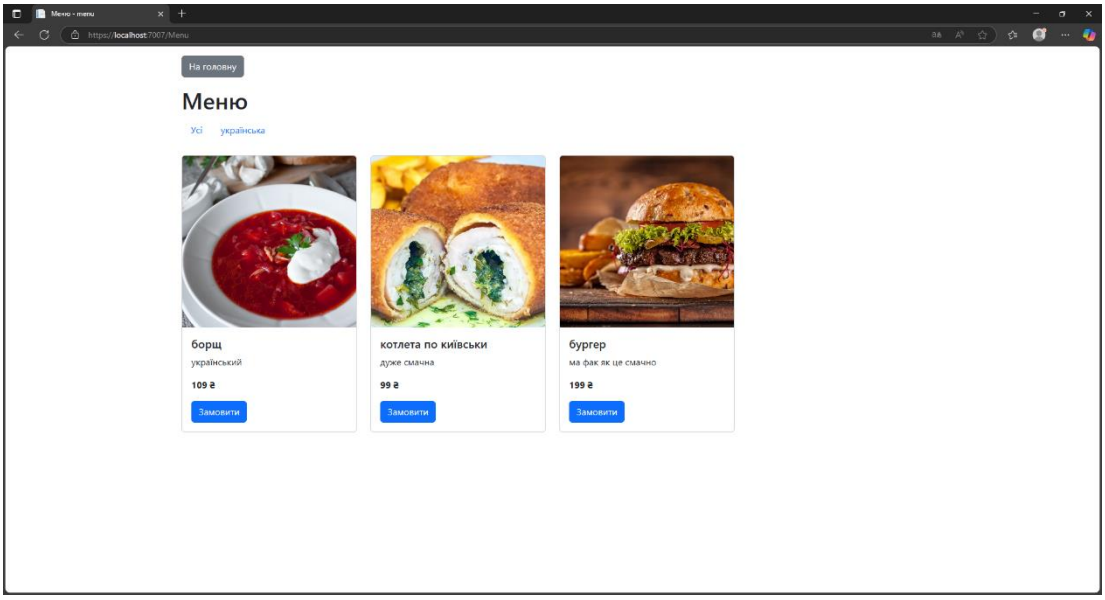


Рисунок 13 – *Перелік страв у меню*

Сторінка оформлення замовлення (рис. 14) дозволяє переглянути деталі сформованого кошика, обрати спосіб оплати (готівкою або банківською картою) та підтвердити замовлення. Це реалізує зручний механізм завершення покупки.

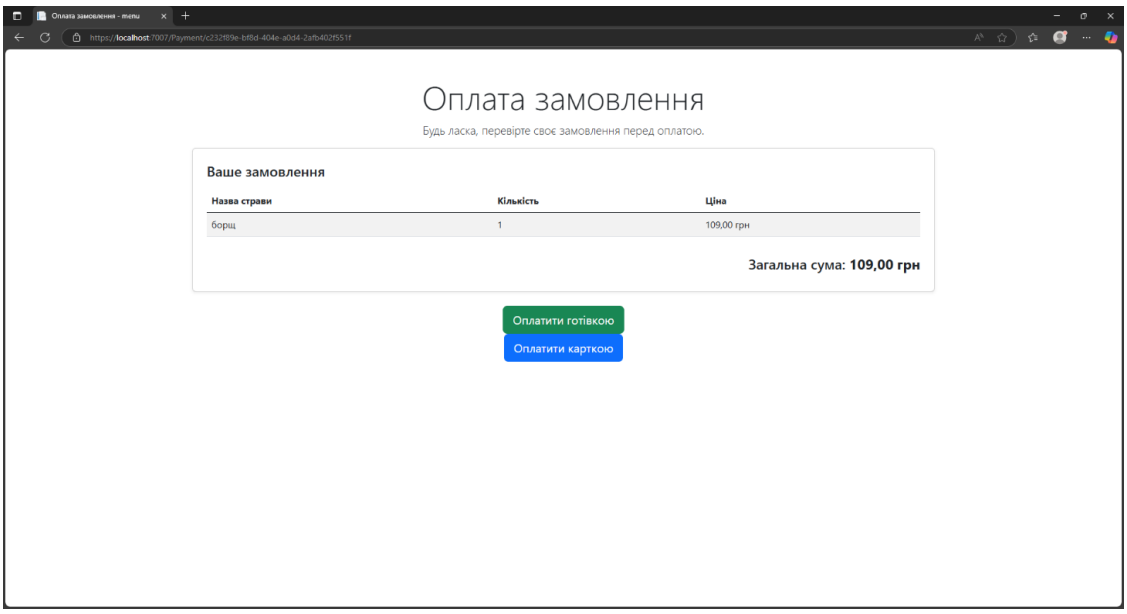


Рисунок 14 – *Сторінка оплати замовлення*

Для адміністративного персоналу реалізовано панель керування, яка дозволяє створювати, редагувати та видаляти як страви, так і категорії. Інтерфейс адмін-панелі подано на *рисунку 15*, а окремі її функціональні компоненти — на *рисунках 26* та *27*.

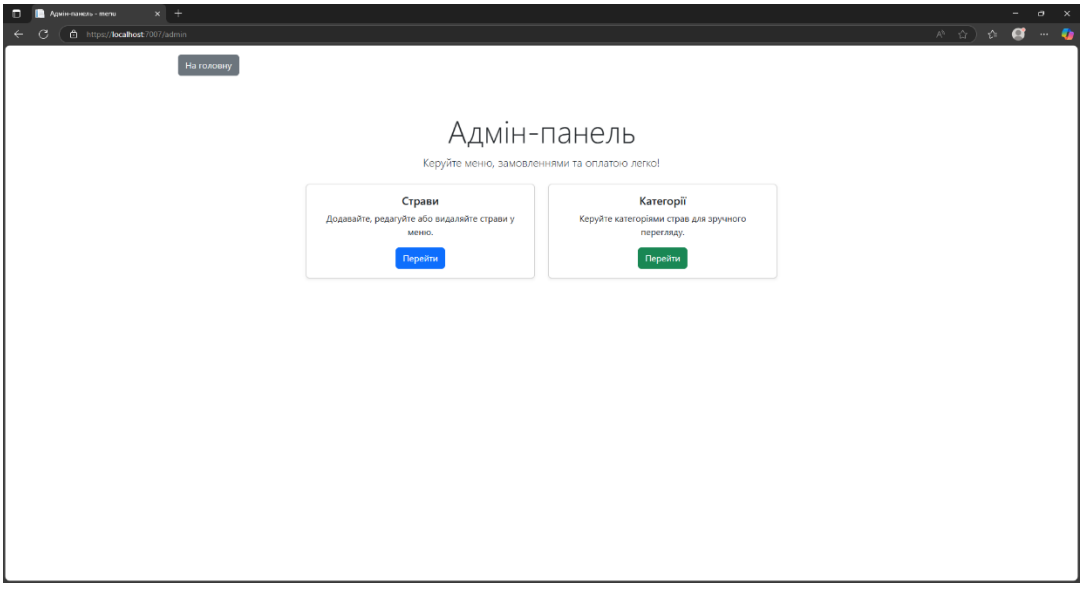


Рисунок 15 – *Адмін-панель керування даними*

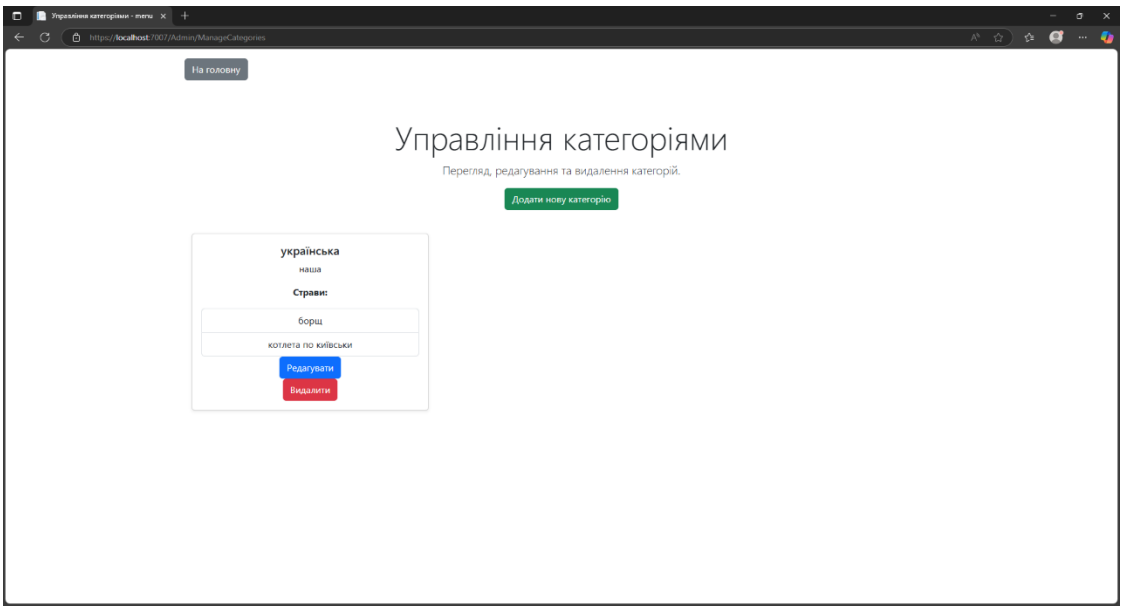


Рисунок 16 – Управління категоріями

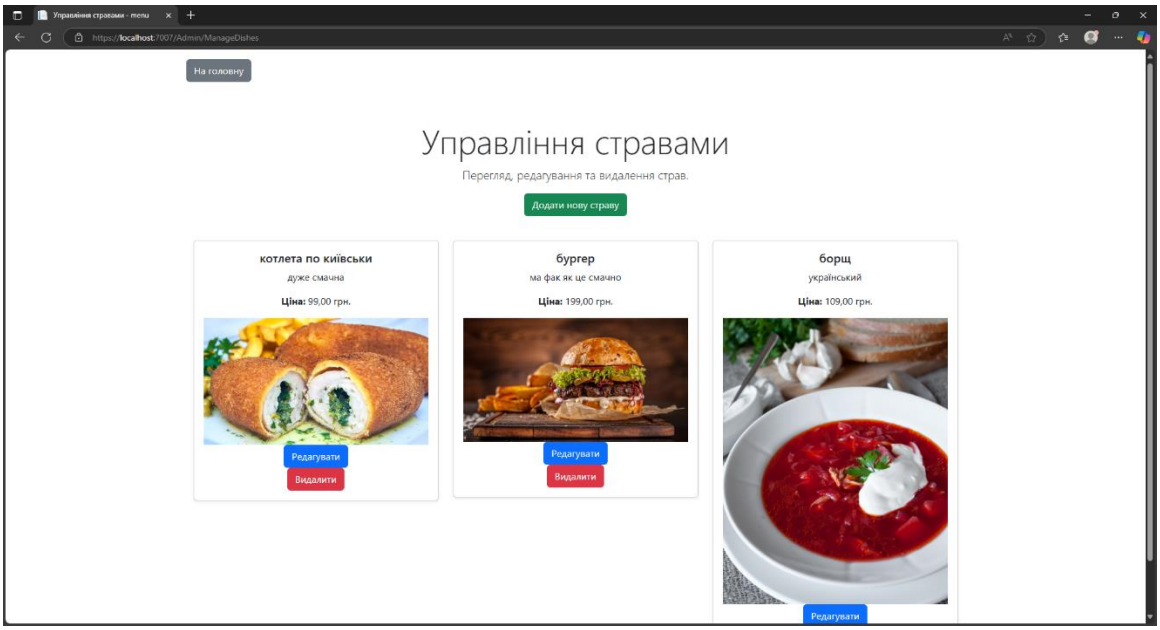


Рисунок 17 – *Перелік страв з можливістю редагування*

В основі дизайну застосунку використано CSS-фреймворк Bootstrap [6], який забезпечує адаптивну верстку та коректне відображення інтерфейсу на різних пристроях — як десктопних, так і мобільних. Структура інтерфейсу побудована за принципами логічної ієрархії та візуальної легкості: усі кнопки та форми мають зрозумілі підписи, поля введення розміщено згідно з очікуваннями користувачів, а елементи керування виділені візуально.

У підсумку, розроблений інтерфейс є сучасним, адаптивним та ергономічним, забезпечуючи ефективну взаємодію з системою як для клієнтів закладу, так і для персоналу, відповідального за наповнення меню та обробку замовлень.

3.7 Програмна реалізація застосунку

Клієнтська частина інформаційної системи реалізована за допомогою технології Razor Pages у межах фреймворку ASP.NET Core. Обраний підхід дозволяє досягти чіткого розділення логіки обробки запитів (PageModel) та представлення (шаблони .cshtml), що сприяє створенню масштабованого, підтримуваного й легко модифікованого вебінтерфейсу.

Уся клієнтська логіка згрупована в директорії Pages, де кожна сторінка має власну модель (.cshtml.cs) та відповідний шаблон (.cshtml). Така організація забезпечує логічне розділення функціональних блоків. Структура директорії представлена на рисунку 18.

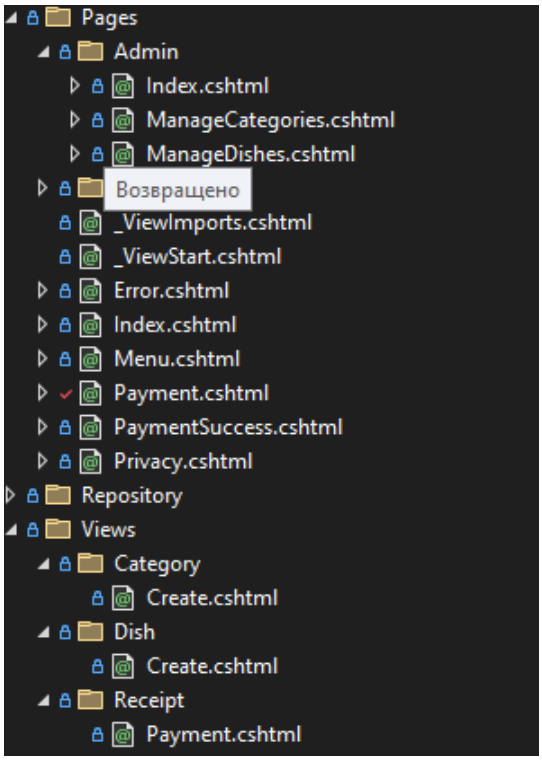


Рисунок 18 – *Файлова структура додатку*

З наведеної файлової структури додатку можна виділити наступні ключові директорії:

- /Pages/Admin — інтерфейс для адміністративного керування стравами та категоріями;
- /Pages/Menu — відображення меню ресторану для користувачів;
- /Pages/Payment — оформлення та оплата замовлення.
- Ключові елементи інтерфейсу
- Інтерфейс застосунку містить низку логічно структурованих сторінок:
- Головна сторінка — привітання та навігація до основних розділів.
- Меню — виведення переліку страв із можливістю додавання до замовлення.
- Оплата — підтвердження та завершення замовлення з вибором способу оплати.
- Адмін-панель — керування даними (категоріями та стравами) через форми CRUD.

Хоча Razor Pages не передбачає компонентну модель у стилі React або Vue, кожна сторінка реалізує окремий функціональний сценарій з чітким розмежуванням обов’язків. Наприклад, сторінка /Pages/Admin/Category/Create.cshtml містить форму створення нової категорії з вибором пов’язаних страв (рис. 19), а сторінка /Pages/Admin/Dish/Create.cshtml дозволяє адміністратору створити або змінити страву, завантажити її зображення та задати параметри доступності.

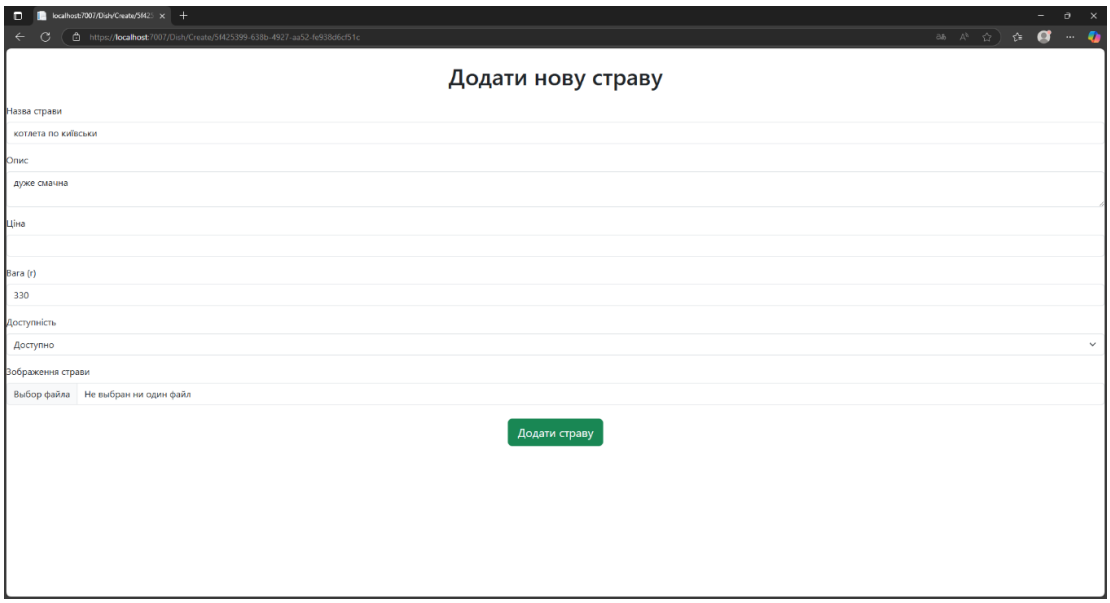


Рисунок 19 – *Форма для створення або редагування страви*

Усі взаємодії з даними здійснюються через моделі сторінок, які викликають відповідні методи з Repository-класів. Адміністративна частина надає можливість:

- створювати нові категорії з асоційованими стравами;
- додавати, редагувати або видаляти страви із вказаними параметрами: назва, опис, ціна, вага, зображення, наявність

Адаптивність і візуальне оформлення

Застосунок використовує Bootstrap, що дозволяє реалізувати адаптивну верстку, сумісну з мобільними пристроями та різними розмірами екранів. Візуальна концепція витримана у мінімалістичному стилі: чиста типографіка, чітка ієрархія елементів та зручне компонування керуючих кнопок і форм.

Підсумки реалізації клієнтської частини

Розроблений інтерфейс забезпечує структуровану архітектуру з можливістю масштабування, зручне адміністрування без надмірної складності, інтуїтивну взаємодію для клієнтів ресторану, швидкий обмін даними без використання важкого клієнтського JavaScript:

Використання Razor Pages дозволило побудувати ефективну клієнтську частину, що є логічно завершеним і гнучким елементом вебзастосунку, який відповідає сучасним вимогам як з точки зору юзабіліті, так і продуктивності.

3.8 Практичне використання додатку

Після завершення реалізації ключових сторінок вебзастосунку, таких як меню страв та сторінка оплати, система стала повністю готовою до практичного використання. На основі інтерфейсних прикладів, наведених на рис. 20 та 21 проілюстровано основні сценарії взаємодії кінцевого користувача із застосунком.

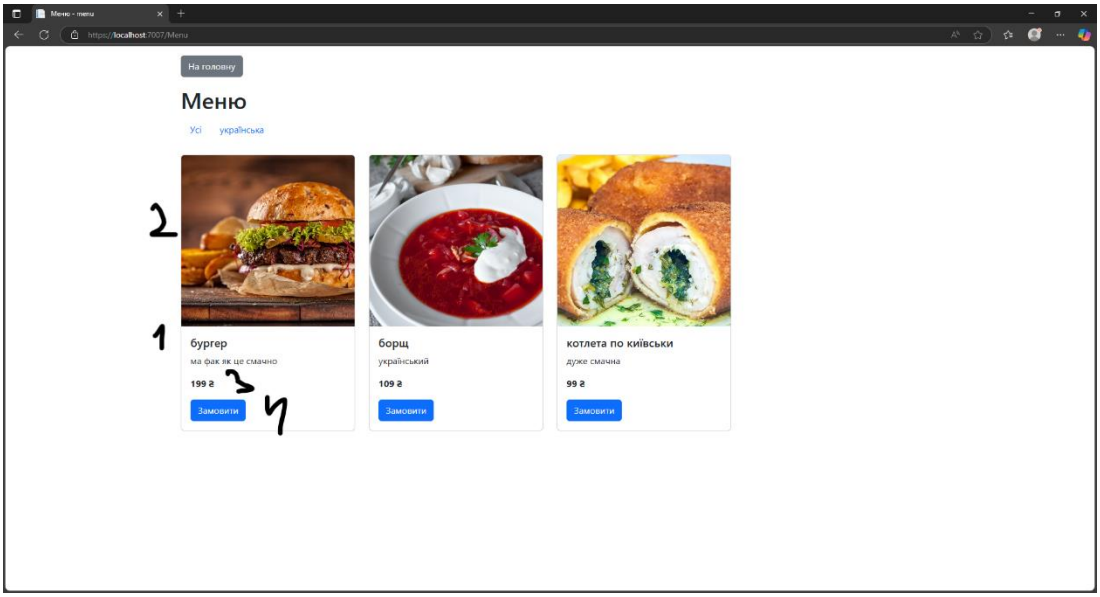


Рисунок *20 – Відображення списку страв у меню*

На цій сторінці реалізовано візуалізацію меню у вигляді окремих карток. Кожна картка страви містить такі елементи:

1. Назву страви та її короткий опис.
2. Зображення для візуального ознайомлення.
3. Вартість однієї порції.
4. Кнопку «Замовити», яка додає відповідну страву до кошика замовлення.

Цей інтерфейс дозволяє користувачеві швидко сформувати персональне замовлення без потреби проходити додаткові кроки або реєстрацію. Сторінка оптимізована для зручного перегляду на мобільних і десктопних пристроях.

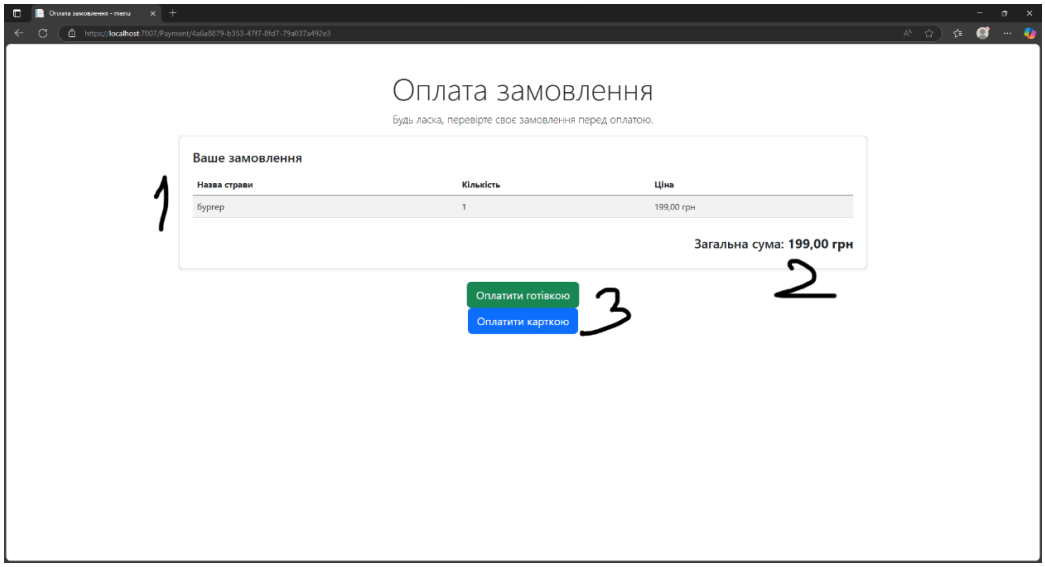


Рисунок *21 – Сторінка оплати замовлення*

Після додавання страв до кошика, користувач автоматично переходить до етапу підтвердження та оплати замовлення. Інтерфейс цієї сторінки передбачає:

- Виведення переліку всіх вибраних позицій із зазначенням кількості та вартості кожної.
- Розрахунок загальної суми замовлення.
- Вибір способу оплати — готівкою або банківською карткою.

Розміщення елементів управління є логічним і простим: підсумок і кнопки підтвердження розташовані в нижній частині сторінки, що забезпечує зручність навігації.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи бакалавра було розроблено прототип інформаційної системи, призначеної для збору, обробки та аналітичного опрацювання даних про ціни на товари у різних торговельних мережах. Система дозволяє здійснювати порівняльний аналіз, формувати рейтинги товарів та забезпечувати користувачів персоналізованими рекомендаціями щодо найвигідніших покупок. Робота охоплює повний цикл розробки інформаційної системи — від аналізу предметної області та вибору технологічного стеку до побудови інтерфейсу, реалізації серверної логіки та валідації функціональності в умовах експериментального середовища.

На першому етапі було здійснено комплексний огляд наукових джерел, що стосуються створення інформаційних систем, аналітичних модулів та підходів до роботи з великими обсягами даних. Розглянуто питання оптимізації процесів, стратегії ціноутворення та засоби інтеграції систем на основі сучасних архітектурних підходів.

У ході роботи проведено аналіз актуальних технологій, призначених для створення як клієнтської, так і серверної частини вебзастосунків. Вибір таких інструментів, як ASP.NET Core, Razor Pages та PostgreSQL, обґрунтовано на підставі їхньої високої продуктивності, масштабованості, зручності у розробці, активної підтримки спільнотою та сумісності з мікросервісною архітектурою.

Було реалізовано користувацький інтерфейс із використанням Razor Pages, який забезпечує інтуїтивну взаємодію, відповідну сучасним вимогам UX/UI-дизайну. Архітектура інтерфейсу передбачає модульність, адаптивність та подальшу інтеграцію з мобільними або десктопними застосунками.

Серверна логіка реалізована у вигляді RESTful API, що відповідає сучасним підходам до побудови розподілених систем. Сервер обробляє запити, зберігає інформацію про товари та магазини в базі даних, виконує

агрегацію та сортування, що забезпечує гнучкість у роботі з даними та можливість масштабування функціональності.

Одним з центральних компонентів системи став аналітичний модуль, який реалізує можливість автоматизованого порівняння товарів між супермаркетами, аналізу змін цін, формування рейтингів та генерації персоналізованих рекомендацій для кінцевого користувача. У межах модуля впроваджено алгоритми фільтрації, нормалізації та зваженого ранжування, що суттєво покращує точність та релевантність результатів.

Система була протестована в експериментальному середовищі із залученням реальних наборів даних. Результати тестування підтвердили її працездатність, стабільність і відповідність функціональним вимогам, сформованим на етапі постановки задачі. Особливу увагу приділено оцінці точності рекомендацій, часу відповіді API та зручності користувацького інтерфейсу.

Таким чином, у межах роботи було створено ефективний приклад застосування сучасних вебтехнологій і методів обробки даних у сфері електронної комерції. Отримані результати можуть бути використані як основа для подальшого розвитку системи — зокрема, інтеграції зовнішніх джерел даних, реалізації мобільного клієнта, впровадження алгоритмів машинного навчання для покращення якості рекомендацій, або розширення функціоналу в напрямку динамічного ціноутворення та моніторингу ринку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бутельський Я. Ю. Автоматизовані інформаційні системи управління підприємствами ресторанного бізнесу. Академічні візії. 2024. Вип. 29. С. 1–8. DOI: https://doi.org/10.5281/zenodo.10691831
2. Стойко І., Долубовська О. Індустрія 4.0: ресторанні комп'ютерні POS-системи. Матеріали конференції 23-24 листопада 2022 року. Тернопіль : ТНТУ, 2022. С. 62–64.
3. Poster POS : програма автоматизації громадського харчування на планшеті. URL: https://joinposter.com (дата звернення: 13.06.2025).
4. R-Keeper : система автоматизації ресторанів. URL: https://rkeeper.com (дата звернення: 13.06.2025).
5. Anderson R., Brock D., Larkin K. Introduction to Razor Pages in ASP.NET Core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-9.0&tabs=visual-studio (дата звернення: 13.06.2025).
6. Build fast, responsive sites with Bootstrap. Powerful, extensible, and feature-packed frontend toolkit. URL: https://getbootstrap.com (дата звернення: 13.06.2025).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Лиходід Олексій Олексійович, студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення освітньої програми “Програмне забезпечення систем”, адреса електронної пошти se121-06@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Клієнтський застосунок для ресторанного обслуговування**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 02.06.2025 _____ Лиходід Олексій Олексійович
(підпис) (прізвище та ініціали) (студент)

Дата 02.06.2025 _____ Олександр Олексійович Жуков
(підпис) (прізвище та ініціали) (керівник)