

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Вебсистема автоматизації роботи закладів
харчування»**

Виконав студент групи ІПЗ-22-2
КРИВОЛАП Назарій Олександрович

Керівник роботи:
ЧИЖМОТРЯ Олексій Володимирович

Рецензент:
САВІЦЬКИЙ Роман Святославович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«27» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **КРИВОЛАП Назарій Олександрович**

Керівник роботи: **ЧИЖМОТРЯ Олексій Володимирович**

Тема роботи: **«Вебсистема автоматизації роботи закладів харчування»**,
затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., № 92/с**

Вихідні дані для роботи: розробка вебсистеми автоматизації роботи закладів харчування, що оптимізує комунікацію між залом та кухнею, дозволить автоматизувати рутинні завдання з прийому замовлень і повністю цифровізувати фінансово-складський облік.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В.	01.03.2026р.	01.03.2026р
2	Чижмотря О.В.	01.03.2026р	01.03.2026р
3	Чижмотря О.В.	01.03.2026р	01.03.2026р

РЕФЕРАТ

Випускна кваліфікаційна робота присвячена розробці вебсистеми автоматизації роботи закладів харчування. Робота складається з вступу, 3-х розділів, висновків, списку використаних джерел із 30 найменувань (3 стор.), 9 додатків (19 стор.), 15 таблиць та 36 ілюстрацій по тексту. Повний обсяг роботи становить 100 сторінок, з них 81 сторінка основного тексту.

Метою роботи є розробка програмного комплексу, що забезпечує керування меню, створення та обробку замовлень, передавання замовлень на кухонний дисплей, облік складських залишків, роботу з рецептурами та формування публічного QR-меню.

У роботі проведено аналіз предметної області та існуючих систем автоматизації, визначено функціональні й нефункціональні вимоги, спроектовано архітектуру, базу даних і алгоритми роботи системи. Реалізовано клієнтську частину на React і серверну частину на ASP.NET Core Web API з використанням PostgreSQL, Entity Framework Core, SignalR та Docker Compose. Проведено тестування основних функцій і CI/CD-розгортання.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗАЦІЯ, ВЕБСИСТЕМА, ЗАКЛАД ХАРЧУВАННЯ, ЗАМОВЛЕННЯ, ОБЛІ

					ІПЗ.КР.Б – 121 – 26 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Вебсистема автоматизації роботи закладів харчування	Літ.	Арк.	Аркушів
Розроб.		Криволап Н.О.						
Керівник		Чижмоторя О.В.					3	100
Рецензент		Савіцький Р.В.				Житомирська політехніка, група ІПЗ-22-2		
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The graduation qualification work is devoted to the development of a web system for automating the operation of food service establishments. The work consists of an introduction, three chapters, conclusions, a list of references containing 30 sources (3 pages), 9 appendices (19 pages), 15 tables, and 36 figures within the text. The total volume of the work is 100 pages, including 81 pages of the main text.

The purpose of the work is to develop a software system that provides menu management, order creation and processing, order transfer to the kitchen display, inventory control, recipe management, and public QR menu generation.

The thesis analyzes the subject area and existing automation systems, defines functional and non-functional requirements, and designs the system architecture, database, and operation algorithms. The client side was implemented using React, while the server side was developed using ASP.NET Core Web API with PostgreSQL, Entity Framework Core, SignalR, and Docker Compose. The main system functions and CI/CD deployment were tested.

KEYWORDS: AUTOMATION, WEB SYSTEM, FOOD SERVICE ESTABLISHMENT, ORDER, ACCOUNTING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО ВЕБСИСТЕМИ АВТОМАТИЗАЦІЇ РОБОТИ ЗАКЛАДІВ ХАРЧУВАННЯ	9
1.1. Постановка задачі	9
1.2. Аналіз існуючих систем автоматизації.....	11
1.3. Вибір архітектури вебсистеми автоматизації роботи закладів харчування.	17
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	22
Висновки до першого розділу	27
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБСИСТЕМИ АВТОМАТИЗАЦІЇ РОБОТИ ЗАКЛАДІВ ХАРЧУВАННЯ	28
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури вебсистеми.....	28
2.2. Розробка бази даних вебсистеми.....	35
2.3. Проєктування алгоритмів роботи вебсистеми	39
2.4. Реалізація вебсистеми автоматизації роботи закладів харчування	44
Висновки до другого розділу	49
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБСИСТЕМОЮ АВТОМАТИЗАЦІЇ РОБОТИ ЗАКЛАДІВ ХАРЧУВАННЯ	51
3.1. Порядок встановлення та налаштування параметрів системи	51
3.2. Структура інтерфейсу вебсистеми автоматизації роботи закладів харчування	61
3.3. Тестування роботи програмного комплексу	72
Висновки до третього розділу	76
ВИСНОВКИ	77
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	79
ДОДАТКИ	82

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Application Programming Interface, програмний інтерфейс застосунку.

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

DTO – Data Transfer Object, об'єкт передавання даних.

EF Core – Entity Framework Core, об'єктно-реляційний інструмент для роботи з базою даних.

HTTP – HyperText Transfer Protocol, протокол передавання гіпертексту.

JWT – JSON Web Token, токен для автентифікації та авторизації користувачів.

KDS – Kitchen Display System, кухонний дисплей для відображення замовлень.

POS – Point of Sale, модуль створення та обробки замовлень у точці продажу.

REST – Representational State Transfer, архітектурний стиль взаємодії клієнта і сервера.

SPA – Single Page Application, односторінковий вебзастосунок.

UI – User Interface, користувацький інтерфейс.

UML – Unified Modeling Language, уніфікована мова моделювання.

URL – Uniform Resource Locator, уніфікований покажчик ресурсу.

VPS – Virtual Private Server, віртуальний приватний сервер.

ВСТУП

Сучасний етап розвитку індустрії гостинності та громадського харчування (HoReCa) характеризується високим рівнем конкуренції, зміною споживчих звичок та постійним зростанням вимог до швидкості й якості обслуговування. В умовах динамічного ринку успішність та рентабельність закладу харчування залежить не лише від асортименту страв, але й від ефективності внутрішніх бізнес-процесів: швидкості обробки замовлень, безперебійної комунікації між залом та кухнею, а також безпомилкового складського обліку. Використання традиційних методів управління або застарілих локальних програмних рішень призводить до зниження пропускної здатності закладу, затримок в обслуговуванні, появи помилок через людський фактор та втрати фінансового контролю.

Актуальність теми зумовлена потребою закладів харчування у швидкій, зручній та надійній автоматизації основних бізнес-процесів. В умовах високої конкуренції ефективність роботи закладу залежить від швидкості обробки замовлень, якісної взаємодії між залом і кухнею, точності складського обліку та зменшення впливу людського фактора. Традиційні методи управління або застарілі локальні програми часто не забезпечують достатньої гнучкості.

Метою роботи є розробка та впровадження комплексної кросплатформної вебсистеми автоматизації закладів харчування для підвищення швидкості обслуговування клієнтів, оптимізації комунікації між персоналом та забезпечення точного фінансово-складського обліку.

Для досягнення поставленої мети необхідно виконати комплекс завдань, що охоплює повний цикл створення програмного продукту:

1. Провести аналіз предметної області закладів харчування.
2. Проаналізувати наявні системи автоматизації.
3. Визначити функціональні та нефункціональні вимоги.
4. Обґрунтувати вибір архітектури, технологій і засобів розгортання.
5. Спроектувати реляційну базу даних системи.
6. Реалізувати серверну частину з API, авторизацією та бізнес-логікою.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						7

7. Реалізувати клієнтську частину з основними модулями.
8. Налаштувати Docker-розгортання та CI/CD-процес.
9. Провести тестування функцій і підтвердити працездатність системи.

Об'єктом дослідження є процес управління та автоматизації операційної діяльності (прийом замовлень, складський облік, комунікація персоналу) у закладах громадського харчування.

Предметом дослідження виступають методи, алгоритми та програмні засоби розробки кросплатформної інформаційної системи для автоматизації бізнес-процесів закладу харчування.

Інформаційну базу дослідження склали наукові праці у галузі автоматизації сфери HoReCa та офіційна технічна документація, а для вирішення поставлених завдань застосовано методи системного аналізу, реляційного моделювання баз даних, об'єктно-орієнтованого проєктування на платформі .NET та компонентного моделювання інтерфейсів у React.

Практичне значення одержаних результатів полягає у створенні кросплатформеного хмарного програмного комплексу, який завдяки роботі через веббраузер функціонує на будь-яких пристроях персоналу і суттєво знижує витрати на IT-інфраструктуру закладу.

Впровадження системи дозволяє підвищити якість сервісу за допомогою інтерактивного публічного QR-меню для гостей, оптимізувати комунікацію з кухнею через систему коментарів до замовлень, а також забезпечити точність складського обліку завдяки механізму архівації даних (Soft Delete).

Налаштовані конфігурації Docker Compose та автоматичні конвеєри CI/CD забезпечують безперебійне розгортання та оновлення вебсистеми в хмарному середовищі Microsoft Azure, роблячи продукт повністю готовим до інтеграції в реальний сектор ресторанного бізнесу.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО ВЕБСИСТЕМИ АВТОМАТИЗАЦІЇ РОБОТИ ЗАКЛАДІВ ХАРЧУВАННЯ

1.1. Постановка задачі

Робота закладів харчування пов'язана з постійною обробкою замовлень, керуванням меню, взаємодією між працівниками залу, кухні та адміністрації. У невеликих і середніх закладах частина цих процесів часто виконується вручну або за допомогою окремих інструментів, які не об'єднані в єдину систему. Це може призводити до затримок під час передавання замовлень, помилок у їхньому складі, складності контролю поточного стану кухні, а також до неузгодженості між інформацією в меню та фактичною доступністю страв.

У зв'язку з цим виникає потреба у створенні вебсистеми, яка дасть змогу автоматизувати основні операції закладу харчування та забезпечити швидку взаємодію між ключовими робочими зонами. Під час розробки такої системи важливо враховувати не лише її функціональні можливості, а й зручність використання, надійність, захищеність та можливість подальшого розширення. Відповідно до підходів програмної інженерії, перед початком реалізації програмного комплексу необхідно визначити вимоги до системи, структуру її основних модулів та способи взаємодії між ними [1].

У межах випускної роботи необхідно розробити програмний комплекс, який складатиметься з клієнтської частини, серверної частини та бази даних. Такий підхід відповідає поширеним принципам побудови вебзастосунків, коли інтерфейс користувача відокремлюється від бізнес-логіки, а дані зберігаються в окремій системі керування базами даних [2]. Це дозволяє зробити систему більш гнучкою, зручною для підтримки та придатною для подальшого розвитку.

Першочерговим завданням є аналіз основних процесів закладу харчування: формування меню, приймання замовлень, передавання їх на кухню, зміна статусів виконання та завершення обслуговування. На основі цього визначаються користувачі системи та їхні права доступу. Потрібно передбачити ролі адміністратора, працівника POS-зони та працівника кухні, що дозволить упорядкувати роботу й обмежити доступ до окремих функцій [3].

Важливою частиною системи є адміністративний модуль, який забезпечує керування категоріями, стравами, цінами та доступністю позицій меню. Це дає змогу підтримувати актуальність даних і зменшити кількість помилок під час створення замовлень.

POS-модуль призначений для швидкого формування замовлень: вибору страв, зміни кількості позицій, розрахунку суми та підтвердження продажу. Після цього замовлення зберігається в базі даних і передається до KDS-модуля, де працівники кухні переглядають активні замовлення та змінюють їхні статуси.

Під час реалізації серверної частини необхідно забезпечити обробку запитів від клієнтського застосунку, перевірку вхідних даних, виконання бізнес-логіки та взаємодію з базою даних. Для збереження інформації потрібно спроектувати структуру бази даних, яка міститиме дані про користувачів, ролі, категорії, страви, замовлення та позиції замовлень. Використання об'єктно-реляційного підходу до роботи з даними дозволяє спростити взаємодію між серверною частиною та базою даних [4].

Також необхідно передбачити авторизацію користувачів і розмежування доступу відповідно до їхніх ролей. Це є важливою умовою безпечної роботи системи, оскільки адміністративні функції не повинні бути доступними звичайним працівникам. Окрім цього, система має відповідати базовим вимогам до якості програмного забезпечення: бути зручною, надійною, зрозумілою для користувача, придатною до супроводу та подальшого розширення [5].

На завершальному етапі потрібно виконати тестування розробленого програмного комплексу. Необхідно перевірити коректність входу користувачів до системи, роботу рольового доступу, створення та редагування позицій меню, формування замовлень, їхнє відображення на кухні та зміну статусів. Результатом виконання поставлених завдань має стати вебсистема, яка забезпечує автоматизацію основних процесів роботи закладу харчування, зменшує кількість ручних операцій і покращує взаємодію між працівниками

1.2. Аналіз існуючих систем автоматизації

На ринку програмного забезпечення для закладів харчування існує багато систем для автоматизації продажів, обліку замовлень, керування меню, складу, персоналу та фінансової звітності. Такі рішення зазвичай поєднують POS-модуль, адміністративну панель, аналітику й інтеграції з обладнанням.

Для аналізу було обрано Poster POS, Skyservice POS та SERVIO POS. Їх порівняння дозволяє визначити типові можливості систем цього класу та обґрунтувати особливості власної розробки.

Poster POS – це хмарна система автоматизації для закладів громадського харчування, зокрема кафе, ресторанів, кав'ярень, барів, фаст-фудів і пекарень. Основне призначення системи полягає в організації роботи каси (рис. 1.1), прийманні замовлень, керуванні меню, веденні складського обліку, контролі фінансів і формуванні звітності [6].

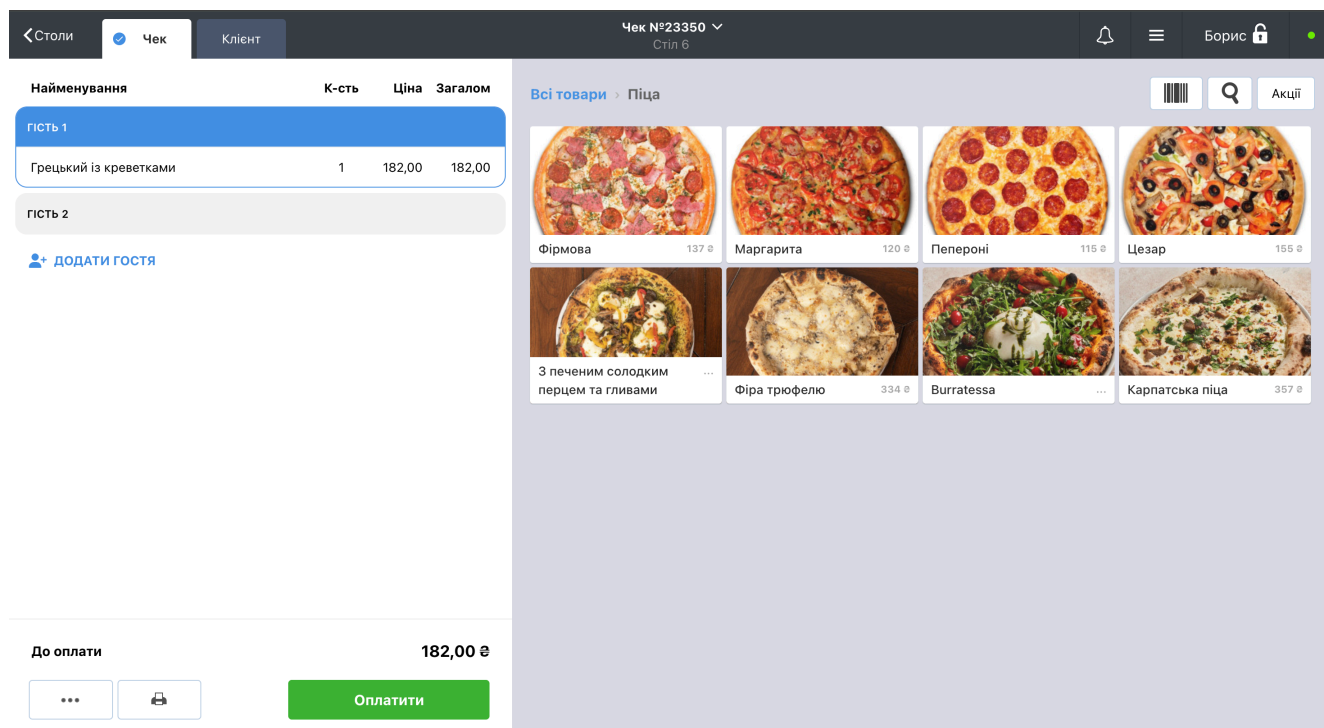


Рис. 1.1. Вигляд каси в системі Poster POS

Poster POS орієнтована на зручність щоденної роботи працівників закладу: касир або офіціант може швидко створювати замовлення (рис. 1.2), додавати позиції з меню, працювати з різними форматами продажу та передавати інформацію на кухню.

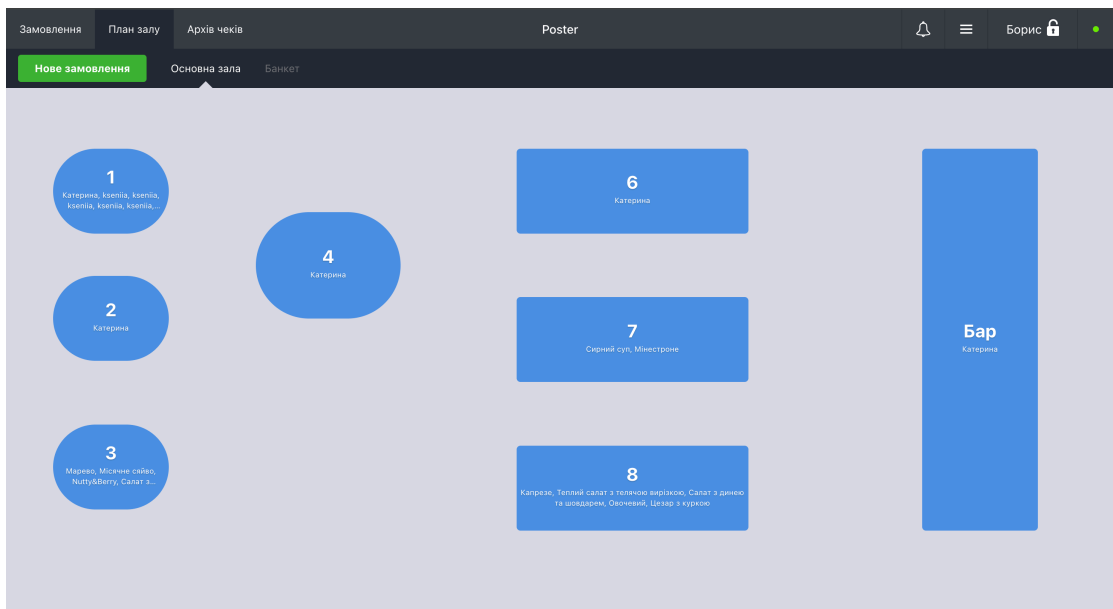


Рис. 1.2. Сторінка створення замовлення для конкретного столу

Важливою перевагою Poster POS є розвинена екосистема додаткових сервісів. Система підтримує QR-меню, онлайн-оплату, чайові, відгуки, аналітику (рис. 1.3), маркетингові інструменти, програми лояльності та інтеграції з іншими сервісами. QR-меню може формуватися на основі страв, доданих в адміністративній панелі, що спрощує оновлення інформації для клієнтів [7].

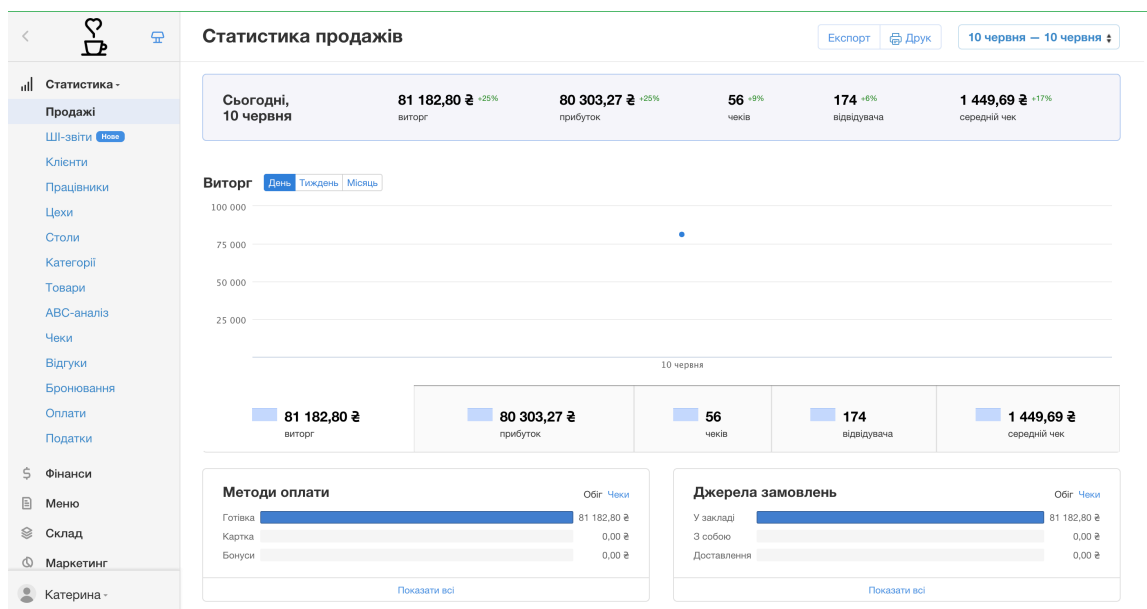


Рис. 1.3. Сторінка з аналітикою продажів

Також Poster має окремі рішення для кухні, зокрема Kitchen Kit, що дає змогу кухарям бачити замовлення, склад страв, номер столу, час надходження замовлення та статуси приготування [8]. Таким чином, Poster POS є комплексним

рішенням, яке охоплює не лише касові операції, а й значну частину управлінських процесів закладу.

Skyservice POS – це хмарна касова система, призначена для автоматизації продажів, складського обліку та фінансів. Система може використовуватися на планшеті (рис. 1.4.), телефоні або комп’ютері, що робить її гнучкою для різних форматів бізнесу [9]. Для закладів харчування Skyservice POS пропонує можливості обліку продажів, контролю залишків, роботи з товарами, фінансової аналітики, підтримки ПРРО та формування звітів. Система також позиціонується як інструмент для контролю бізнесу з будь-якої точки, оскільки керівник може переглядати фінансові операції, складські дані, статистику та аналітику через хмарний доступ [10].

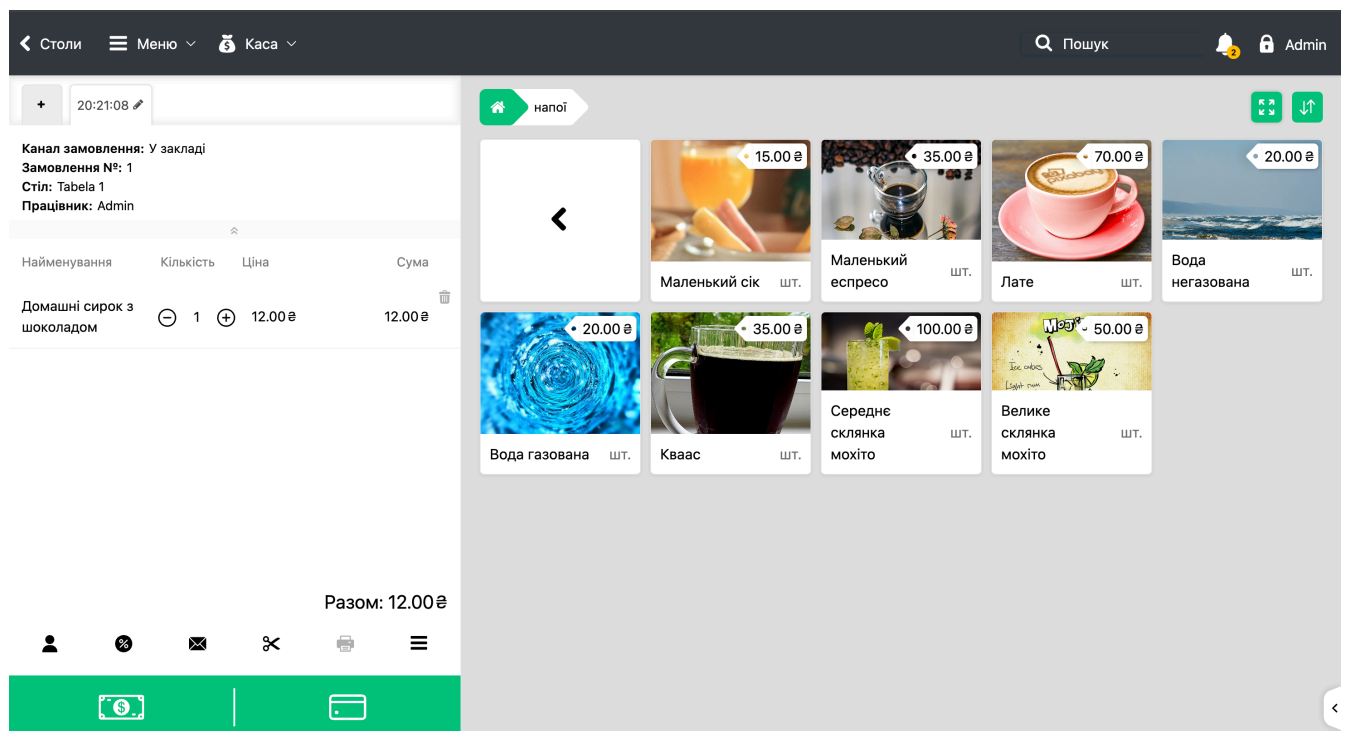


Рис. 1.4. Вигляд каси Skyservice Pos на планшеті

До переваг Skyservice POS можна віднести широкий набір функцій для обліку товарів (рис. 1.5) і руху продукції.

Пошук по меню

POS - Термінал

Перейти до каси

Початкова

Товари

Товари

Модифікатори

Залишки товарів

Статистика

Фінанси

Маркетинг

Налаштування

Мої додатки

Допомога

Вихід

Kalbasa industries

42

Залишки товарів

Контроль залишки товарів на складах та торгових точках в одному місці.

Додати

🔍

📍 Студентська робота

🏠 Обрано кілька складів

📦 Залишки по складах

🔧 Усі фільтри

☐ Найменування	Кількість	Категорія	Собівартість	Σ витрат	Σ роздрібної ціни	
☐ *Карп. Джерельна*2л н/г	0,00 шт.	—	▼ 0,00 ₪	0,00 ₪	0,00 ₪	
☐ Сирол Фісташка	0,000 г	—	▼ 0,00 ₪	0,00 ₪	0,00 ₪	
☐ Капучино	-4,00 шт.	- Еспрессо	▼ 0,00 ₪	0,00 ₪	-320,00 ₪	
☐ Бамбл	0,000 кг	—	▼ 30,00 ₪	0,00 ₪	0,00 ₪	
☐ Медіатор Fender 351 Shape Premium Celluloid Pic...	-6,00 шт.	—	▼ 0,00 ₪	0,00 ₪	-1830,00 ₪	
☐ Фірмовий Бургер	-54,00 шт.	- Какао	▼ 0,00 ₪	0,00 ₪	-3240,00 ₪	
☐ Безалкогольний напій Moxito	-24,00 шт.	- Какао	▼ 0,00 ₪	0,00 ₪	-960,00 ₪	
☐ джинси	-1,00 шт.	—	▼ 0,00 ₪	0,00 ₪	-500000,00 ₪	
☐ 1233456	-70,00 шт.	—	▼ 0,00 ₪	0,00 ₪	-14000,00 ₪	
☐ ваза	-3,00 шт.	—	▼ 0,00 ₪	0,00 ₪	-4497,00 ₪	
☐ зірка на ялинку	-22,00 шт.	—	▼ 0,00 ₪	0,00 ₪	-10978,00 ₪	
☐ монс	-21,00 шт.	—	▼ 0,00 ₪	0,00 ₪	-210000,00 ₪	
☐ Капучино з сиропом і вершками	-10,00 шт.	- Американо з молоком	▼ 0,00 ₪	0,00 ₪	-1000,00 ₪	
☐ чоботи демисезонні	-4,00 шт.	—	▼ 0,00 ₪	0,00 ₪	-14000,00 ₪	

Знайдено: 194

Сума витрат: -283436.58 ₪

Загальна вартість: -24112564.36 ₪

Загальна кількість товарів: -8621.63

Рис. 1.5. Облік товарів на складі

Система дозволяє фіксувати прихід товарів, переміщення, повернення, списання (рис. 1.6) та виробництво [11]. Для закладів харчування це важливо, оскільки складський облік безпосередньо пов'язаний із контролем собівартості, залишків інгредієнтів і плануванням закупівель. Також Skyservice POS підтримує ПРРО, що є важливою вимогою для реального комерційного використання в Україні [12]. Водночас система має більш універсальну орієнтацію, оскільки застосовується не лише в ресторанах і кафе, а й у торгівлі, сфері послуг та інших видах бізнесу.

Skyservice

Пошук по меню

POS - Термінал
Перейти до каси

Початкова

Товари

Статистика

Основне

Рух товару

Продажі

Надходження

Повернення

Списання

Переміщення

Зміни

Витрата товару

Топ продажів

Kalbasa industries

Списання

Відстежуй списання товарів — зіпсовані, втрачені чи інвентаризовані.

Додати

За весь часТоргова точкаСкладЗмінаПродавецьЦех

Продавець	Дата час	Торгова точка	Сума	Причина списання	
Admin	2026-06-06 16:11:57	ВМагазин 1	—	—	1
Admin	2026-04-23 10:58:10	Студентська робота	—	—	1
Admin	2026-04-21 18:18:43	Клктд	100.00 ₪	—	1
Admin	2026-03-25 23:55:13	Студентська робота	—	—	1
Admin	2026-03-12 10:00:18	Test-shop	—	прострочка	1
Admin	2026-02-28 15:33:38	—	—	—	1
Admin	2026-01-14 18:38:41	Клктд	—	—	1

Знайдено: 7 Показано: 7

Рис. 1.6. Сторінка перегляду історії списання товару

SERVIO POS – це комплексна система автоматизації ресторанного бізнесу, орієнтована на кафе, ресторани, бари, готельно-ресторанні комплекси та інші заклади. SERVIO позиціонується як програмний інструмент для вирішення різних завдань ресторанного бізнесу: обслуговування гостей, проведення банкетів, складського й управлінського обліку (рис. 1.7), ведення калькуляційних і технологічних карт, фінансів, дисконтних програм, акцій та обліку робочого часу персоналу [13]. На відміну від простіших POS-рішень, SERVIO POS має більш виражену орієнтацію на комплексне управління закладом.

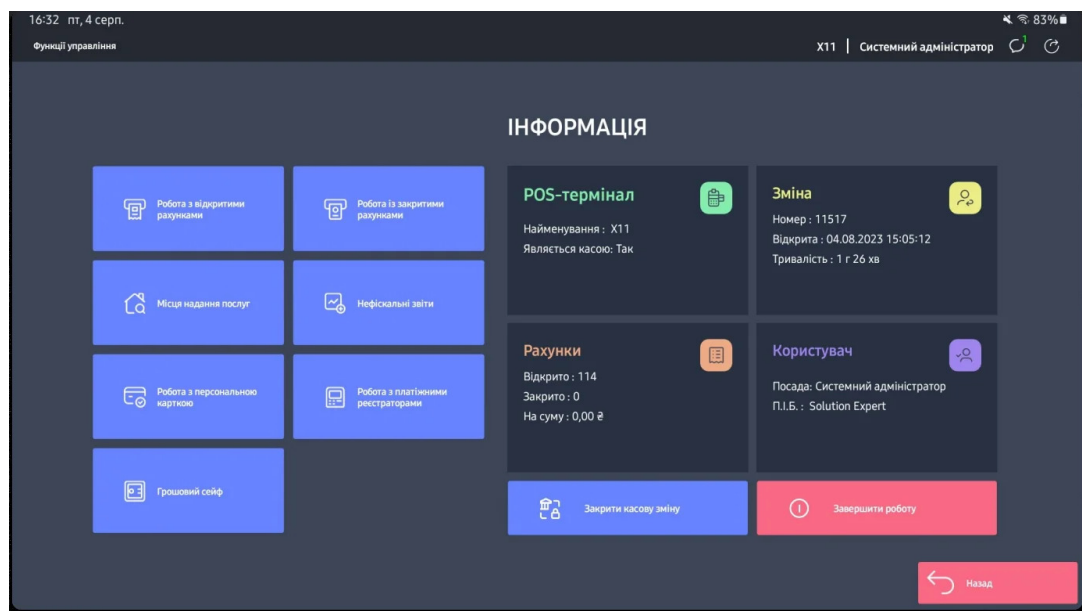


Рис. 1.7. Функції управління в системі SERVIO POS

Система підтримує роботу з касою (рис. 1.8), рахунками, меню, персоналом, інтеграціями з РРО/ПРРО та банківськими терміналами [14]. Для ресторанного бізнесу важливими є також можливості налаштування доступів, контролю дій працівників, роботи з банкетами, передоплатою, черговою подачі страв і програмами лояльності. SERVIO POS більше підходить для закладів, які потребують глибокої автоматизації та мають складніші внутрішні процеси. До таких закладів можна віднести ресторани з повним обслуговуванням, мережеві заклади, готельно-ресторанні комплекси та підприємства, де необхідна інтеграція декількох модулів в одну систему.

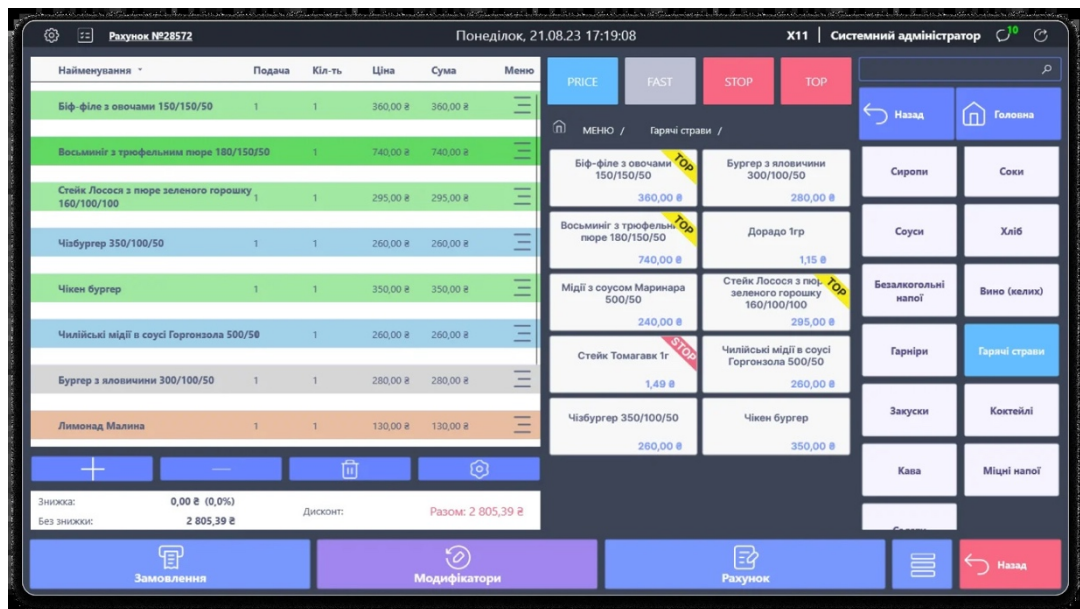


Рис. 1.8. Вигляд каси SERVIO POS

Для узагальнення результатів аналізу було сформовано порівняльну таблицю основних функціональних можливостей розглянутих систем (табл.1.1). У таблиці також наведено функції власної розробки, що дозволяє визначити її місце серед аналогічних програмних продуктів.

Таблиця 1.1

Порівняння функціональних можливостей систем автоматизації закладів харчування

№	Функціональна можливість	Poster POS	Skyservice POS	SERVIO POS	Власна розробка
1	Створення замовлень через POS-інтерфейс	+	+	+	+
2	Керування категоріями та стравами меню	+	+	+	+
3	Адміністративна панель для керування системою	+	+	+	+
4	Рольовий доступ для різних працівників	+	+	+	+
5	Кухонний дисплей / монітор активних замовлень	+	-	+	+
6	Оновлення замовлень для кухні в реальному часі	+	-	+	+
7	Зміна статусів замовлення	+	+	+	+
8	Публічне QR-меню для клієнтів	+	-	-	+
9	Генерація та друк QR-коду меню	+	-	-	+
10	Складський облік і контроль залишків	+	+	+	+

продовж. табл. 1.1

1.	2.	3.	4.	5.	6.
11	Рецептури / технологічні карти страв	+	+	+	+
12	Автоматичне списання інгредієнтів при продажу	+	+	+	+
13	Фінансова аналітика та звітність	+	+	+	+
14	Друк чеків, X/Z-звітів або касових звітів	+	+	+	+
15	Підтримка ПРРО / фіскалізації	+	+	+	-
16	Інтеграція з банківськими терміналами	+	+	+	-

Як видно з таблиці, усі розглянуті комерційні системи мають широкий функціонал і призначені для практичного використання в реальних закладах харчування. Вони підтримують створення замовлень, керування меню, складський облік, технологічні карти, аналітику, роботу з касовими звітами, програми лояльності, доставку, фіскалізацію та інтеграцію з обладнанням. Такі можливості є важливими для закладів, які потребують повноцінного управління всіма бізнес-процесами.

1.3. Вибір архітектури вебсистеми автоматизації роботи закладів харчування

Вибір архітектури є важливим етапом проєктування вебсистеми автоматизації закладів харчування, оскільки він визначає організацію компонентів, їхню взаємодію, розподіл відповідальностей, збереження даних, масштабування та захист доступу. Архітектура має враховувати одночасну роботу різних користувачів, створення замовлень, передавання їх на кухню, керування меню, облік залишків і звітність. Серед поширених підходів розглядаються монолітна, клієнт-серверна, багаторівнева, MVC, SPA, мікросервісна та подієво-орієнтована архітектури.

Монолітна архітектура є простою в розробці й розгортанні, однак має обмеження щодо масштабування окремих частин системи. Мікросервісний підхід забезпечує високу гнучкість і незалежність сервісів, але потребує складнішої інфраструктури, моніторингу, міжсервісної взаємодії та керування

транзакціями, тому для системи середнього рівня складності є надлишковим [15].

MVC-архітектура дозволяє розділити дані, логіку та представлення, однак класичний варіант більше орієнтований на серверне формування сторінок. Для інтерактивної системи з POS-модулем, кухонним дисплеєм, адміністративною панеллю та публічним меню доцільніше використовувати SPA-підхід, за якого клієнтська частина працює як окремий застосунок без повного перезавантаження сторінок.

Для майбутньої вебсистеми автоматизації роботи закладів харчування оберемо клієнт-серверну багаторівневу архітектуру з односторінковим клієнтським застосунком та RESTful API. У межах такого підходу система буде складатися з трьох основних рівнів: клієнтського рівня, серверного рівня та рівня бази даних (рис. 1.9).

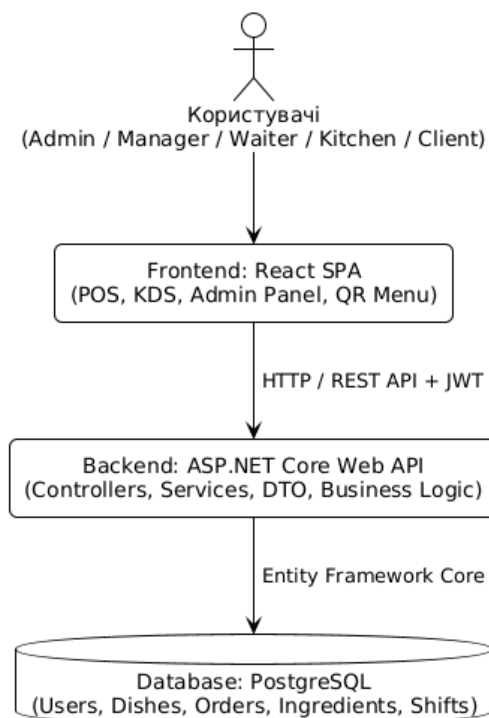


Рис. 1.9. Загальна клієнт-серверна архітектура вебсистеми автоматизації роботи закладів харчування

Клієнтський рівень буде представлений вебзастосунком типу SPA. Він відповідатиме за відображення інтерфейсу користувача та взаємодію з API. У межах клієнтської частини передбачається створення кількох функціональних

зон: POS-інтерфейсу для формування замовлень, KDS-інтерфейсу для кухні, адміністративної панелі для керування системою та публічного QR-меню для клієнтів (рис. 1.10). Такий розподіл дозволить адаптувати інтерфейс до конкретної ролі користувача й уникнути перевантаження окремих екранів зайвими функціями.

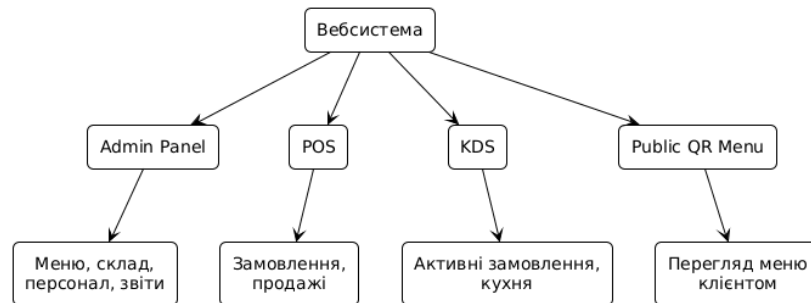


Рис. 1.10. Функціональні зони вебсистеми автоматизації роботи закладу харчування

Серверний рівень буде реалізований у вигляді Web API, який оброблятиме запити клієнтського застосунку, перевірятиме права доступу, виконуватиме бізнес-логіку, працюватиме з базою даних і повертатиме результати у форматі JSON. Для організації серверної частини найкраще використати контролери, сервіси, DTO-об'єкти, доменні моделі та контекст бази даних.

Рівень бази даних забезпечуватиме збереження інформації про користувачів, меню, страви, інгредієнти, рецептури, замовлення, складські операції та касові зміни. Для взаємодії клієнта й сервера буде використовуватися RESTful API зі стандартними HTTP-методами GET, POST, PUT, PATCH і DELETE, що забезпечує відокремлення клієнтської частини від серверної та можливість подальшого підключення інших клієнтів [16].

Окремим архітектурним рішенням стане використання механізму оновлення даних у реальному часі для кухонного дисплея. У процесі роботи закладу харчування важливо, щоб нові оплачені замовлення одразу з'являлися на екрані кухні без ручного оновлення сторінки. Для цього необхідно передбачити WebSocket-комунікацію або надбудову над нею, наприклад SignalR [17] (рис.1.11). У такій моделі сервер після створення або зміни статусу замовлення зможе надсилати повідомлення підключеним клієнтам кухні. Це

відповідає подієво-орієнтованому підходу, у якому певна дія в системі породжує подію, а зацікавлені клієнти отримують оновлення.

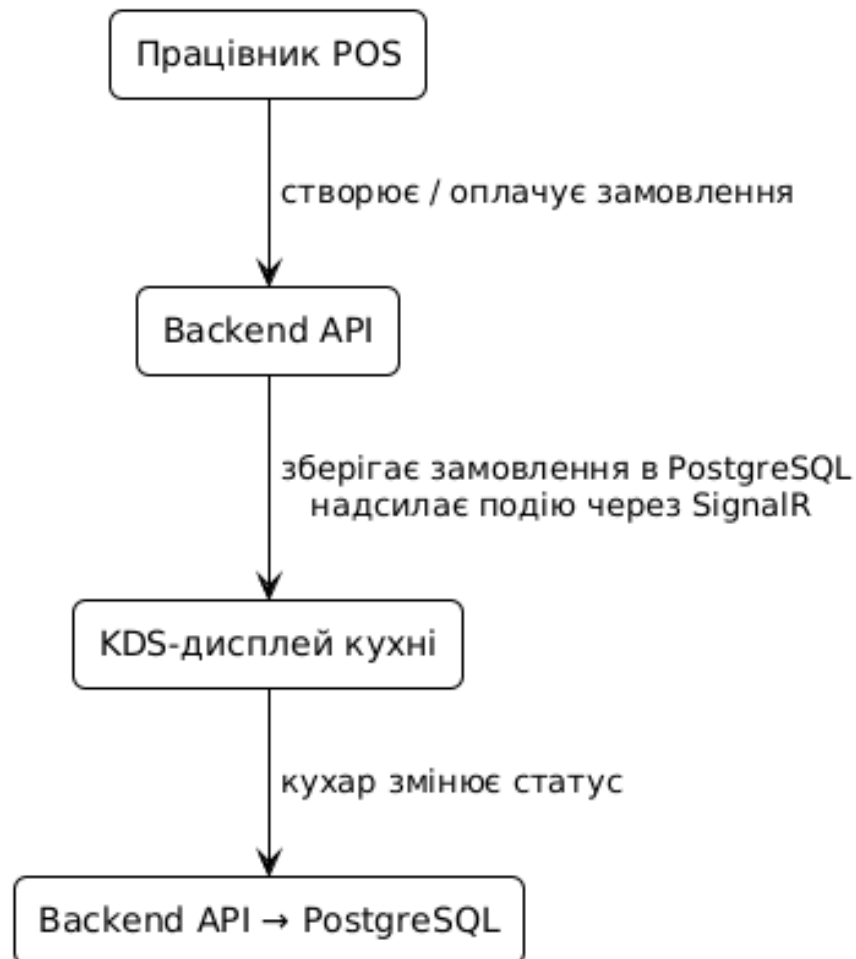


Рис. 1.11. Схема передавання замовлення від POS-модуля до кухонного дисплея

Для внутрішньої структури серверної частини використаємо Service Layer Pattern, за якого бізнес-логіка виноситься з контролерів в окремі сервісні класи. Наприклад, операції із замовленнями реалізуються в сервісі замовлень, складські операції – у складському сервісі, а авторизація – в сервісі автентифікації. Це забезпечує розділення відповідальності, спрощує тестування та зменшує дублювання коду.

Для передавання даних між клієнтом і сервером використовуються DTO-об'єкти. Вони дозволяють не передавати внутрішні доменні сутності напряму, а формувати окремі структури запитів і відповідей. Це підвищує безпеку та зменшує залежність frontend від структури бази даних.

Важливою складовою є механізм автентифікації та авторизації. У системі застосуємо рольову модель доступу з ролями адміністратора, менеджера, офіціанта та працівника кухні. Для захисту API використовуються JWT-токени, а для швидкого входу персоналу – PIN-код.

Для збереження історичної цілісності даних передбачено soft delete, тобто архівацію записів без їх фізичного видалення. Для оптимізації роботи з великими списками застосовується серверна пагінація та фільтрація. Розгортання системи доцільно виконувати в контейнерах за допомогою Docker Compose [18], а процес збирання, тестування та розгортання автоматизувати через CI/CD-конвеєр.

Порівняльний опис розглянутих архітектурних підходів наведено в таблиці 1.2.

Таблиця 1.2

Порівняння архітектурних підходів для вебсистеми автоматизації закладів харчування

Архітектурний підхід	Переваги	Недоліки	Доцільність для системи, що проєктується
Монолітна архітектура	Простота розробки, тестування та розгортання	Складніше масштабувати окремі частини системи	Доцільна для серверної частини середнього масштабу
Мікросервісна архітектура	Висока масштабованість, незалежність сервісів	Складна інфраструктура, складніше тестування й розгортання	Надлишкова для меж випускної роботи
Класичний MVC	Чіткий поділ на модель, представлення і контролер	Менш зручний для інтерактивного SPA-інтерфейсу	Може використовуватися частково на рівні API-контролерів
SPA + REST API	Швидкий інтерфейс, розділення frontend і backend, зручність підтримки	Потребує окремої організації клієнтської маршрутизації та захисту маршрутів	Найбільш доцільний варіант
Подієво-орієнтована архітектура	Швидке оновлення даних, зручна для KDS	Потребує додаткового механізму обміну повідомленнями	Доцільна частково для оновлення кухонного дисплея
Багаторівнева архітектура	Чіткий розподіл відповідальностей, зручність супроводу	Потребує продуманого проєктування шарів	Доцільна як основа всієї системи

Отже, для вебсистеми автоматизації роботи закладів харчування найбільш підходящою є клієнт-серверна багаторівнева архітектура з SPA-клієнтом, RESTful API та реляційною базою даних. Вона дозволяє розділити інтерфейс, бізнес-логіку й збереження даних, підтримати рольовий доступ і декілька функціональних зон. Використання REST API забезпечує стандартизовану взаємодію клієнта й сервера, SignalR – оновлення KDS у реальному часі, а контейнеризація спрощує розгортання системи.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для системи, що проєктується, потрібно використати сучасний веборієнтований технологічний стек, який забезпечить розділення клієнтської та серверної частин, роботу з реляційною базою даних, підтримку авторизації користувачів, оновлення даних у реальному часі та можливість контейнеризованого розгортання. З урахуванням функціональних вимог системи планується використання таких основних технологій: React, TypeScript, Vite, Tailwind CSS для клієнтської частини; ASP.NET Core Web API, Entity Framework Core та SignalR для серверної частини; PostgreSQL як СКБД; Docker Compose, Nginx, GitLab CI/CD та Microsoft Azure для розгортання і супроводу системи.

Клієнтську частину вебсистеми буде оптимально реалізувати у вигляді односторінкового застосунку, тобто SPA. Такий підхід дозволить забезпечити швидку взаємодію користувача з інтерфейсом без постійного повного перезавантаження сторінок. Це особливо важливо для POS-інтерфейсу, кухонного дисплея та адміністративної панелі, де користувач повинен швидко виконувати типові операції: додавати страви до замовлення, змінювати статуси, переглядати залишки, керувати меню або друкувати службові документи.

Для реалізації frontend-частини використаємо React, оскільки ця бібліотека дозволяє створювати інтерфейс із незалежних багаторазових компонентів [19]. Це зручно для вебсистеми автоматизації закладу харчування, де повторно використовуються кнопки, модальні вікна, картки страв, таблиці, форми, повідомлення та навігаційні елементи. Для підвищення надійності коду варто

застосувати TypeScript, який забезпечує статичну типізацію та зменшує ризик помилок під час роботи з даними користувачів, замовлень, страв, інгредієнтів, ролей і статусів.

Як інструмент збирання клієнтської частини буде використовуватись Vite, що забезпечує швидкий запуск середовища розробки та ефективне формування production-збірки [20]. Для стилізації інтерфейсу варто застосувати Tailwind CSS, який дає змогу швидко створювати адаптивні сторінки за допомогою utility-класів і підтримувати єдиний стиль POS-модуля, адміністративної панелі, KDS та QR-меню [21].

Для взаємодії з backend API буде використано Axios із централізованим налаштуванням адреси API, JWT-токена та обробки помилок. Для керування клієнтським станом можна застосувати Zustand, а для генерації QR-коду меню – бібліотеку qrcode.react із підтримкою SVG-формату.

Серверну частину системи буде реалізовано за допомогою ASP.NET Core Web API. Ця технологія підтримує створення RESTful API, маршрутизацію, middleware, dependency injection, автентифікацію, авторизацію та інтеграцію з різними системами збереження даних [22]. Для вебсистеми автоматизації закладів харчування ASP.NET Core забезпечує продуктивність, зрозумілу структуру проєкту та зручну побудову багаторівневої серверної логіки.

Серверний рівень поділяється на контролери, сервіси, DTO-об'єкти, доменні моделі та контекст бази даних. Для роботи з реляційною базою використовується Entity Framework Core, який забезпечує роботу з об'єктною моделлю, міграціями та зменшує потребу в ручному SQL-коді.

Як систему керування базами даних обрано PostgreSQL – об'єктно-реляційну СКБД з відкритим кодом, що підтримує SQL, транзакції, зовнішні ключі, індекси та обмеження цілісності [23]. Вона підходить для збереження пов'язаних даних про замовлення, страви, рецептури, інгредієнти, складські операції та касові зміни, а також забезпечує збереження історичної цілісності інформації.

Для оновлення кухонного дисплея в реальному часі використаємо SignalR, що дозволяє серверу надсилати повідомлення клієнтам без постійного опитування API. Завдяки цьому нові або оплачені замовлення автоматично з'являються в KDS-модулі, що покращує взаємодію між POS-зоною та кухнею.

Для захисту системи застосуємо JWT-токени та швидкий вхід працівників за PIN-кодом. JWT забезпечує перевірку доступу до API, а PIN-код спрощує авторизацію персоналу. PIN-коди зберігаються у хешованому вигляді.

Для документування й тестування API використаємо Swagger / OpenAPI, що спрощує розробку та перевірку HTTP-запитів.

Для розгортання системи використаємо контейнеризацію. Docker і Docker Compose дозволяють запускати клієнтську частину, серверну частину та базу даних як окремі сервіси в одному середовищі (рис. 1.12).

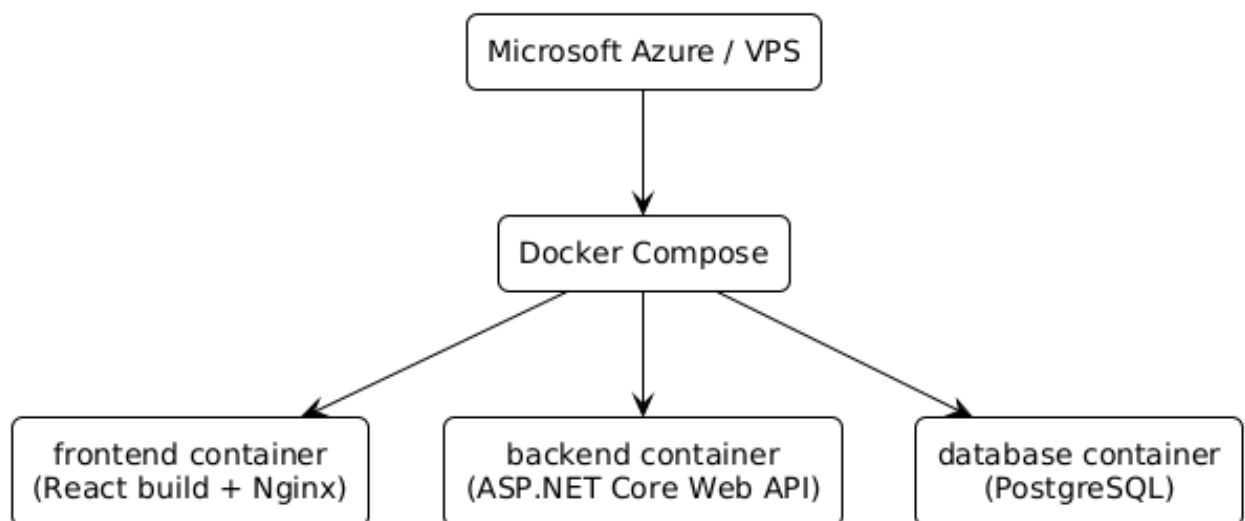


Рис. 1.12. Схема контейнеризованого розгортання вебсистеми

Це спрощує налаштування залежностей, зменшує відмінності між локальним і серверним середовищами та полегшує перенесення системи. Для обслуговування frontend-частини у production-середовищі використаємо Nginx, який працює як вебсервер для статичних файлів SPA-застосунку [24].

Для автоматизації збирання, тестування та розгортання буде використано GitLab CI/CD. Конвеєр автоматично перевіряє код, збирає frontend і backend, запускає тести та розгортає систему на VPS або хмарному сервері з підтримкою Docker-контейнерів [25].

Узагальнений вибір інструментальних засобів наведено в таблиці 1.3.

Таблиця 1.3

Обґрунтування вибору інструментальних засобів

Компонент системи	Обраний інструмент / технологія	Призначення
Клієнтський застосунок	React	Побудова компонентного SPA-інтерфейсу
Мова frontend-розробки	TypeScript	Статична типізація та зменшення кількості помилок
Збірка frontend	Vite	Швидке середовище розробки та production-збірка
Стилізація	Tailwind CSS	Швидке створення адаптивного інтерфейсу
HTTP-запити	Axios	Взаємодія клієнта з REST API
Клієнтський стан	Zustand	Збереження стану авторизації, кошика, замовлень
QR-коди	qrcode.react	Генерація QR-коду для публічного меню
Серверна частина	ASP.NET Core Web API	Реалізація REST API та бізнес-логіки
ORM	Entity Framework Core	Робота з реляційною БД через об'єктну модель
Real-time	SignalR	Оновлення кухонного дисплея в реальному часі
СКБД	PostgreSQL	Збереження структурованих даних системи
Документація API	Swagger / OpenAPI	Опис і тестування API-ендпоінтів
Контейнеризація	Docker Compose	Запуск frontend, backend і БД як окремих сервісів
Вебсервер	Nginx	Обслуговування статичних файлів frontend
CI/CD	GitLab CI/CD	Автоматизація збірки, тестування та розгортання
Платформа розгортання	Microsoft Azure / VPS	Розміщення системи у серверному середовищі

Окрім вибору програмних засобів, необхідно визначити вимоги до апаратного забезпечення. Оскільки система буде веборієнтованою, основне навантаження припадатиме на сервер, де працюватимуть backend, база даних і вебсервер. Клієнтські пристрої повинні мати лише сучасний браузер і стабільне мережеве з'єднання.

Для локальної розробки системи необхідно використовувати комп'ютер із процесором не нижче Intel Core i5 або аналогічним, 8–16 ГБ оперативної пам'яті та SSD-накопичувачем. Така конфігурація дозволить одночасно запускати середовище розробки, frontend, backend, базу даних і Docker-контейнери. Для

комфортної роботи з контейнеризованим середовищем бажано мати не менше 16 ГБ оперативної пам'яті.

Для серверного розгортання мінімально достатньою буде віртуальна машина з 2 vCPU, 4 ГБ оперативної пам'яті та 30–40 ГБ SSD-дискового простору [26]. Така конфігурація підходить для демонстраційного або навчального використання, а також для невеликого закладу з помірною кількістю замовлень. Для реального комерційного використання з великою кількістю одночасних користувачів потрібно збільшити обсяг оперативної пам'яті до 8 ГБ, використовувати швидший SSD-накопичувач і передбачити регулярне резервне копіювання бази даних.

Вимоги до клієнтських пристроїв є помірними. Для POS-зони може використовуватися планшет, ноутбук або моноблок із сучасним браузером. Для кухонного дисплея достатньо планшета або екрана, підключеного до комп'ютера, який відкриватиме KDS-інтерфейс. Для адміністративної панелі бажано використовувати ноутбук або персональний комп'ютер із більшим екраном, оскільки робота з таблицями, складом, меню та звітами потребує більшого робочого простору.

Таблиця 1.4

Орієнтовні вимоги до апаратного забезпечення

Середовище / пристрій	Мінімальні вимоги	Рекомендовані вимоги
Комп'ютер розробника	4-ядерний процесор, 8 ГБ RAM, 20 ГБ вільного місця	Intel Core i5 / Apple M1 або вище, 16 ГБ RAM, SSD
Сервер розгортання	2 vCPU, 4 ГБ RAM, 30 ГБ SSD	4 vCPU, 8 ГБ RAM, 50+ ГБ SSD
POS-пристрій	Планшет або ПК із сучасним браузером	Сенсорний монітор / планшет, стабільний Wi-Fi
KDS-пристрій	Планшет або екран із браузером	Окремий дисплей на кухні, постійне живлення
Адміністративне робоче місце	Ноутбук або ПК із браузером	Екран від 13–15", стабільне підключення до мережі
Мережа	Локальна мережа або Wi-Fi	Стабільний Wi-Fi / Ethernet для POS і KDS

Отже, для реалізації вебсистеми автоматизації роботи закладів харчування обрано сучасний клієнт-серверний стек технологій, що поєднує React SPA, ASP.NET Core Web API, PostgreSQL, SignalR та Docker Compose. Такий набір

інструментів дозволить створити зручний інтерфейс користувача, надійну серверну логіку, структуроване збереження даних, оновлення замовлень у реальному часі та відтворюване розгортання. Вибрані технології відповідають функціональним вимогам системи, є достатньо поширеними, мають активну документацію та забезпечують можливість подальшого розвитку програмного комплексу.

Висновки до першого розділу

У першому розділі було розглянуто передумови створення вебсистеми автоматизації роботи закладів харчування. Визначено, що сучасні заклади потребують програмних рішень для керування меню, замовленнями, кухнею, персоналом, складом і звітністю. Такі системи зменшують кількість ручних операцій, пришвидшують обслуговування клієнтів і покращують контроль внутрішніх процесів.

У межах постановки задачі було визначено потребу в реалізації адміністративної панелі, POS-інтерфейсу, кухонного дисплея, рольового доступу, засобів керування меню, складського обліку та формування звітної інформації. Також враховано вимоги до зручності використання, захисту даних і можливості подальшого розширення системи.

Під час аналізу аналогів розглянуто Poster POS, Skyservice POS та SERVIO POS. Встановлено, що комерційні системи мають широкий функціонал, однак власна розробка дозволяє зосередитися на ключових процесах закладу та забезпечити гнучкість подальшого розвитку.

У результаті порівняння архітектурних підходів обрано клієнт-серверну багаторівневу архітектуру з SPA-клієнтом, RESTful API та реляційною базою даних. Для реалізації обґрунтовано використання React, TypeScript, Vite, Tailwind CSS, ASP.NET Core Web API, Entity Framework Core, SignalR, PostgreSQL, Docker Compose, Nginx і GitLab CI/CD. Такий набір технологій забезпечує зручну розробку, підтримку, розгортання та модернізацію програмного комплексу.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБСИСТЕМИ АВТОМАТИЗАЦІЇ РОБОТИ ЗАКЛАДІВ ХАРЧУВАННЯ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури вебсистеми

Вебсистема призначена для автоматизації основних процесів закладу харчування: керування меню, створення й обробки замовлень, передавання їх на кухню, контролю виконання, складського обліку, роботи з рецептурами, формування звітності та забезпечення рольового доступу. Інтерфейс системи поділено на функціональні зони відповідно до потреб користувачів.

Основними ролями є адміністратор, менеджер, працівник POS-зони, працівник кухні та клієнт. Адміністратор і менеджер працюють з управлінськими функціями, POS-працівник створює замовлення, кухар використовує KDS-модуль, а клієнт переглядає QR-меню (табл. 2.1).

Таблиця 2.1

Функціональні вимоги до вебсистеми

№	Функціональна вимога	Користувачі
1	Авторизація працівників у системі за PIN-кодом	Адміністратор, менеджер, офіціант, кухня
2	Розмежування доступу відповідно до ролі користувача	Усі авторизовані користувачі
3	Керування працівниками та їхніми ролями	Адміністратор
4	Керування категоріями меню	Адміністратор, менеджер
5	Додавання, редагування та архівація страв	Адміністратор, менеджер
6	Перегляд публічного QR-меню	Клієнт
7	Генерація та друк QR-коду меню	Адміністратор, менеджер
8	Створення замовлення через POS-інтерфейс	Офіціант, менеджер, адміністратор
9	Додавання позицій до замовлення та розрахунок суми	Офіціант, менеджер, адміністратор
10	Оплата замовлення та передавання його на кухню	Офіціант, менеджер, адміністратор
11	Перегляд активних замовлень на кухонному дисплеї	Працівник кухні
12	Зміна статусу замовлення	Працівник кухні, менеджер, адміністратор
13	Ведення складського обліку інгредієнтів	Адміністратор, менеджер
14	Створення рецептур для страв	Адміністратор, менеджер
15	Автоматичне списання інгредієнтів за рецептурою	Система
16	Формування касових і складських звітів	Адміністратор, менеджер
17	Друк чеків, QR-кодів та звітів	Адміністратор, менеджер, офіціант

Крім функціональних вимог, необхідно визначити нефункціональні вимоги, що характеризують якість роботи системи. Для вебсистеми автоматизації закладів харчування важливими є швидкодія, надійність, зручність використання, захищеність, масштабованість і можливість супроводу (табл. 2.2)

Таблиця 2.2

Нефункціональні вимоги до вебсистеми

№	Нефункціональна вимога	Опис
1	Швидкодія	Основні дії користувача мають виконуватися без помітних затримок
2	Зручність використання	Інтерфейс має бути зрозумілим для працівників без спеціальної технічної підготовки
3	Захищеність	Доступ до функцій має надаватися відповідно до ролі користувача
4	Надійність	Система має коректно обробляти помилки та не втрачати критичні дані
5	Масштабованість	Архітектура має дозволяти подальше розширення функціональності
6	Супроводжуваність	Код має бути структурований за модулями та шарами відповідальності
7	Кросплатформеність	Система має працювати у сучасних браузерях на різних пристроях
8	Цілісність даних	Дані про замовлення, складські операції та користувачів мають зберігатися узгоджено
9	Відновлюваність	Система має бути придатною до резервного копіювання та повторного розгортання
10	Розширюваність	У майбутньому має бути можливо додати ПРРО, онлайн-оплату, доставку або програму лояльності

Окрему увагу під час проектування необхідно приділити часовим характеристикам реакції системи на дії користувача. Система повинна швидко реагувати на дії працівників, оскільки затримки під час створення або передавання замовлення можуть негативно впливати на обслуговування клієнтів (табл. 2.3).

Таблиця 2.3

Вимоги до часових характеристик реакції системи

№	Дія користувача	Очікуваний час реакції
1	Вхід у систему за PIN-кодом	до 2 секунд
2	Завантаження списку категорій або страв	до 2 секунд
3	Додавання позиції до замовлення	до 1 секунди

1.	2.	3.
4	Підтвердження та оплата замовлення	до 3 секунд
5	Передавання замовлення на кухонний дисплей	до 1–2 секунд
6	Зміна статусу замовлення	до 2 секунд
7	Пошук або фільтрація даних у таблицях	до 2 секунд
8	Генерація QR-коду меню	до 2 секунд
9	Формування звіту	до 5 секунд
10	Друк чека або QR-коду	залежить від пристрою друку, але підготовка даних – до 2 секунд

У межах вебсистеми автоматизації закладу харчування основними акторами будуть адміністратор, менеджер, офіціант, працівник кухні та клієнт.

До основних варіантів використання належать: авторизація в системі, керування користувачами, керування меню, керування складом, створення рецептур, створення замовлення, оплата замовлення, передавання замовлення на кухню, перегляд активних замовлень, зміна статусу замовлення, перегляд QR-меню, друк QR-коду та формування звітів. Кожен із цих варіантів використання відповідає окремому бізнес-процесу або його частині (табл. 2.4.)

Таблиця 2.4

Специфікація основних варіантів використання

№	Варіант використання	Актор	Короткий опис
1	Авторизація в системі	Працівник	Користувач вводить PIN-код, після чого система перевіряє дані та надає доступ відповідно до ролі
2	Керування користувачами	Адміністратор	Адміністратор створює, редагує або блокує облікові записи працівників
3	Керування меню	Адміністратор, менеджер	Користувач додає або редагує категорії та страви
4	Керування складом	Адміністратор, менеджер	Користувач додає інгредієнти, змінює залишки, переглядає складські операції
5	Створення рецептури	Адміністратор, менеджер	Користувач задає склад страви через перелік інгредієнтів і їхню кількість
6	Створення замовлення	Офіціант	Користувач обирає страви, додає їх до замовлення та підтверджує продаж
7	Передавання замовлення на кухню	Офіціант, система	Після оплати замовлення система зберігає його та робить доступним для KDS
8	Перегляд активних замовлень	Працівник кухні	Кухар бачить список замовлень, які необхідно приготувати
9	Зміна статусу замовлення	Працівник кухні	Користувач позначає замовлення як готове або завершене

1.	2.	3.	4.
10	Перегляд QR-меню	Клієнт	Клієнт відкриває публічну сторінку меню за QR-кодом
11	Друк QR-коду або чека	Працівник	Система формує друковану форму для QR-коду, чека або звіту
12	Формування звітів	Адміністратор, менеджер	Користувач переглядає звітну інформацію щодо роботи закладу

Основний сценарій роботи системи передбачає, що офіціант входить у систему, відкриває POS-інтерфейс, додає страви до замовлення, підтверджує оплату, після чого замовлення зберігається та передається на кухонний дисплей. Об'єктно-орієнтована структура системи охоплює класи User, Category, Dish, Ingredient, RecipeItem, Order, OrderItem, InventoryTransaction та Shift, які відображають основні сутності предметної області. Саме їх зображено на UML-діаграмі (рис. 2.1) [27].

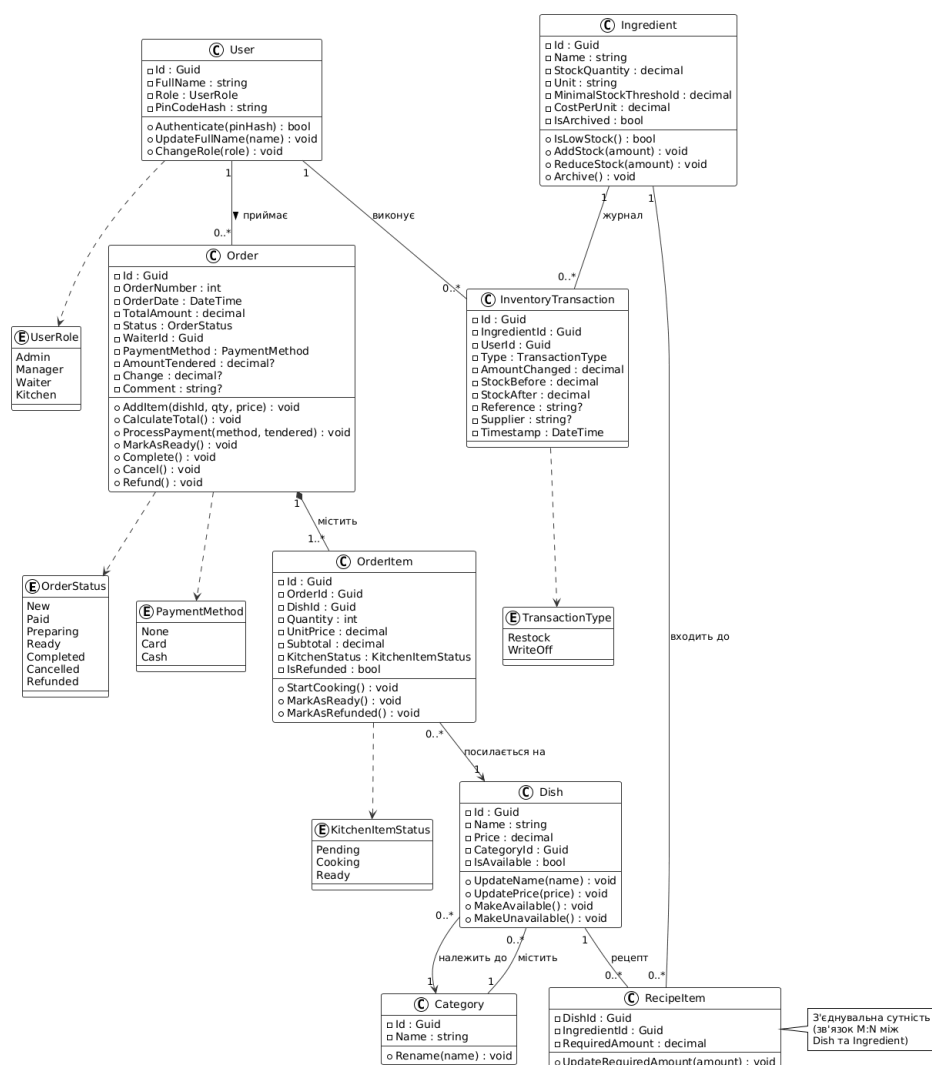


Рисунок 2.2 — UML-діаграма класів основних сутностей вебсистеми

Рис. 2.1. UML-діаграма класів основних сутностей вебсистеми

Основні сутності системи та їхнє призначення наведено в таблиці 2.5.

Таблиця 2.5

Основні сутності об'єктно-орієнтованої структури системи

Сутність	Основні поля	Основні методи / операції	Призначення
User	Id, FullName, PinCodeHash, Role, IsActive	UpdatePinCode(), ChangeRole(), Deactivate()	Описує працівника системи та його роль
Category	Id, Name, Description, IsActive	Rename(), Archive(), Restore()	Визначає категорію меню
Dish	Id, Name, Description, Price, CategoryId, IsAvailable	UpdatePrice(), ChangeAvailability(), Archive()	Описує страву або товар у меню
Ingredient	Id, Name, Unit, StockQuantity, IsArchived	AddStock(), ReduceStock(), Archive(), Restore()	Описує складський інгредієнт і його залишок
RecipeItem	Id, DishId, IngredientId, Quantity	UpdateQuantity()	Визначає кількість інгредієнта, потрібну для страви
Order	Id, OrderNumber, Status, TotalAmount, CreatedAt	ProcessPayment(), MarkAsReady(), Complete(), Cancel()	Описує замовлення та його життєвий цикл
OrderItem	Id, OrderId, DishId, Quantity, UnitPrice	CalculateSubtotal()	Описує окрему позицію в замовленні
InventoryTransaction	Id, IngredientId, Quantity, Type, CreatedAt	CreateIncome(), CreateWriteOff()	Фіксує рух інгредієнтів на складі
Shift	Id, OpenedAt, ClosedAt, OpenedById, ClosedById, CashTotal	Open(), Close(), CalculateTotals()	Описує касову зміну та її підсумки

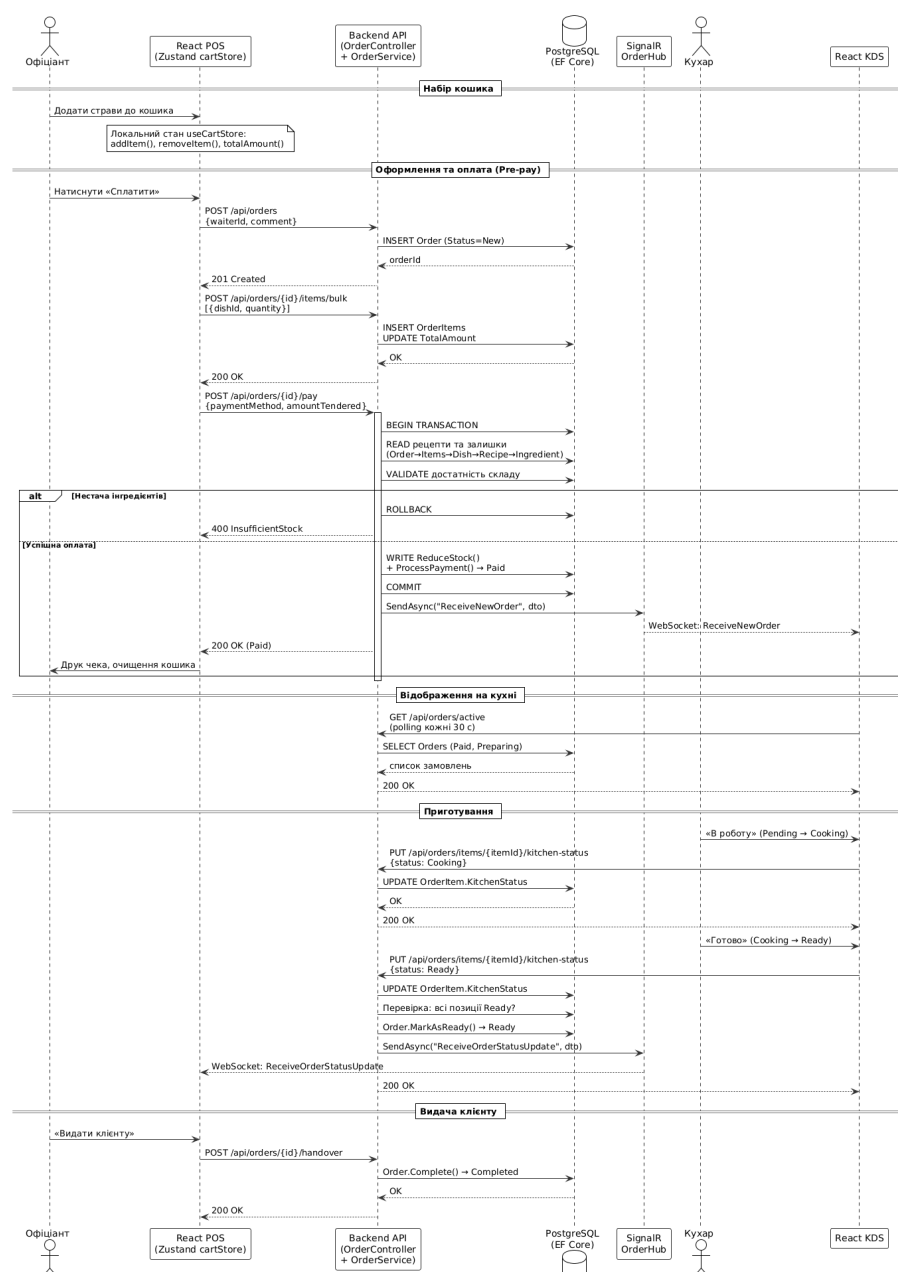
Сутність User зберігає інформацію про працівників, їхні ролі та права доступу. PIN-код користувача зберігається у хешованому вигляді, що підвищує безпеку авторизації. Category використовується для групування страв у меню, а Dish описує конкретну страву або товар із назвою, описом, ціною, доступністю та посиланням на категорію.

Для складського обліку використовуються Ingredient, RecipeItem та InventoryTransaction. Ingredient описує складську позицію, RecipeItem визначає склад страви та кількість інгредієнтів, а InventoryTransaction фіксує надходження, списання й коригування залишків. Це дозволяє зберігати поточні

залишки та історію складських операцій.

Order є центральною сутністю системи й описує замовлення, його номер, статус, суму та дату створення. OrderItem зберігає окремі позиції замовлення, кількість і ціну на момент продажу. Shift використовується для обліку змін.

Основні зв'язки системи: категорія містить багато страв, замовлення містить багато позицій, страва має рецептуру, інгредієнт використовується в рецептурах і складських операціях. Під час створення замовлення дані проходять від POS-інтерфейсу до API, зберігаються в БД і передаються на KDS (рис. 2.2).



Рисунк 2.3 — Схема руху даних під час створення та обробки замовлення

Рис. 2.2. Схема руху даних під час створення та обробки замовлення

Під час створення замовлення користувач POS-інтерфейсу обирає страви, після чого клієнтська частина надсилає запит до серверного API. Сервер перевіряє дані, створює замовлення та його позиції, виконує складські операції, зберігає результат у базі даних і передає оновлення до KDS-модуля. Взаємодія клієнтських модулів із сервером здійснюється через REST API, а оновлення кухонного дисплея – через real-time повідомлення (рис. 2.3).

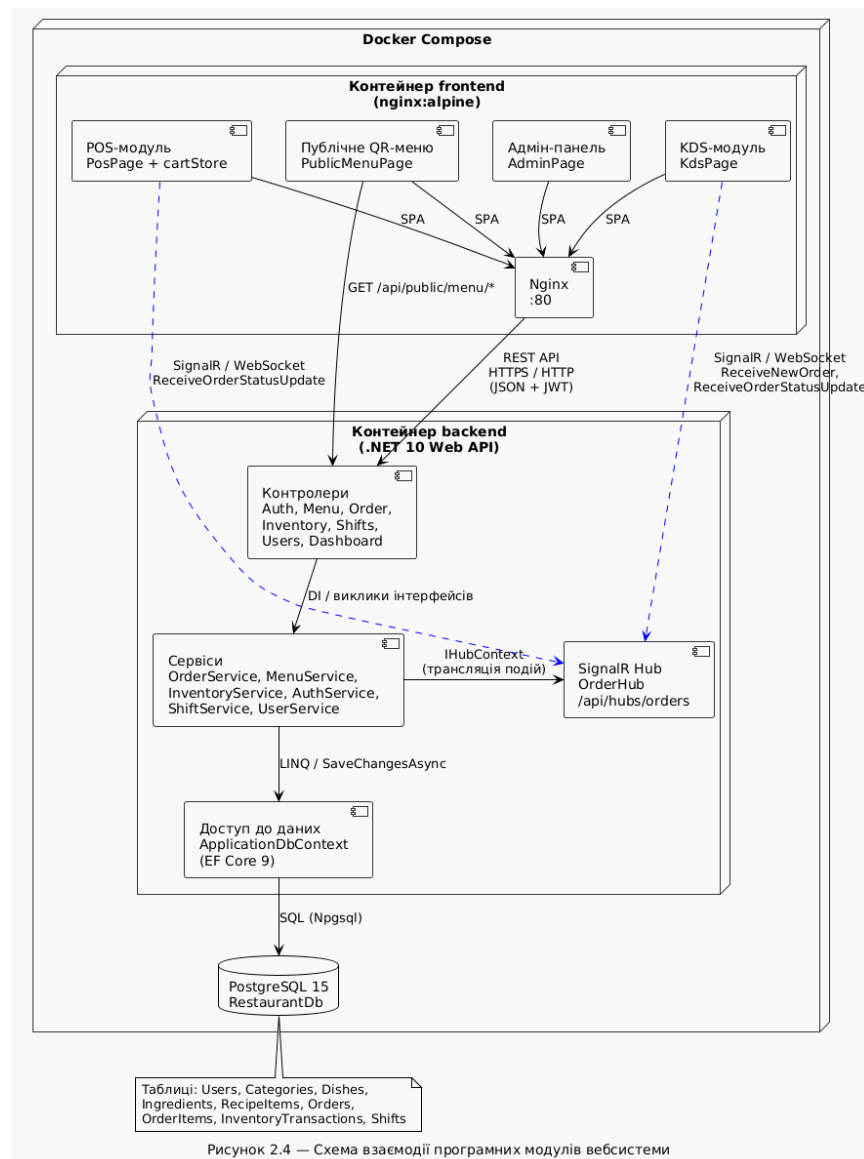


Рис. 2.3. Схема взаємодії програмних модулів вебсистеми

Отже, у підрозділі було визначено основні вимоги до вебсистеми автоматизації роботи закладів харчування, описано типи користувачів. Отримані результати є основою для подальшого проєктування бази даних, алгоритмів роботи системи та реалізації програмних модулів.

2.2. Розробка бази даних вебсистеми

База даних забезпечує централізоване збереження інформації про користувачів, меню, страви, інгредієнти, рецептури, замовлення, складські операції та касові зміни. Для системи використовується реляційна модель, оскільки основні сутності мають чіткі зв'язки: категорія містить страви, замовлення складається з позицій, страва має рецептуру, а інгредієнти беруть участь у складських операціях.

Як СКБД обрано PostgreSQL, що підтримує транзакції, зовнішні ключі, індекси й обмеження цілісності та інтегрується з ASP.NET Core через Npgsql і Entity Framework Core [28]. Схема бази даних формується за підходом Code-First із використанням міграцій (рис. 2.4).



Рис. 2.4. Фізична схема бази даних вебсистеми

Таблиця Users зберігає дані про працівників закладу: ідентифікатор, повне ім'я, хеш PIN-коду, роль, ознаку активності та службові часові поля. Роль визначає доступ користувача до адміністративної панелі, POS-модуля, KDS або управлінських функцій.

Таблиці Categories і Dishes описують структуру меню. Categories використовується для групування страв, а Dishes зберігає назву, опис, ціну, доступність і посилання на категорію. Для грошових значень використовується тип numeric(10,2).

Складський облік забезпечують таблиці Ingredients, RecipeItems і InventoryTransactions. Ingredients містить дані про інгредієнти, одиниці вимірювання, залишки, мінімальну кількість і собівартість. RecipeItems пов'язує страви з інгредієнтами та визначає їхню кількість у рецептурі. InventoryTransactions фіксує надходження, списання, коригування й автоматичне використання інгредієнтів під час продажу.

Таблиця Orders зберігає замовлення, їхні номери, статуси, суми та дати створення, оплати й завершення. OrderItems містить позиції замовлення, кількість, ціну на момент продажу та підсумкову суму. Таблиця Shifts використовується для обліку касових змін, часу відкриття й закриття, відповідальних працівників і підсумкових показників.

Опис основних таблиць бази даних наведено в таблиці 2.6.

Таблиця 2.6

Основні таблиці бази даних вебсистеми

Таблиця	Призначення	Основні поля
Users	Збереження даних про працівників системи	Id, FullName, PinCodeHash, Role, IsActive
Categories	Групування страв у меню	Id, Name, Description, IsActive
Dishes	Збереження страв і товарів меню	Id, Name, Description, Price, CategoryId, IsAvailable
Ingredients	Облік складських інгредієнтів	Id, Name, Unit, StockQuantity, MinStock, Cost, IsArchived
RecipeItems	Опис рецептур страв	Id, DishId, IngredientId, Quantity
Orders	Збереження замовлень	Id, OrderNumber, Status, TotalAmount, CreatedAt, PaidAt, CompletedAt

продовж. табл. 2.6

1.	2.	3.
OrderItems	Збереження позицій замовлення	Id, OrderId, DishId, Quantity, UnitPrice, Subtotal
InventoryTransactions	Фіксація руху інгредієнтів на складі	Id, IngredientId, Type, Quantity, CreatedAt, Comment
Shifts	Облік касових змін	Id, OpenedAt, ClosedAt, OpenedById, ClosedById, CashTotal

Основні зв'язки між таблицями наведено в таблиці 2.7.

Таблиця 2.7

Зв'язки між таблицями бази даних

Батьківська таблиця	Дочірня таблиця	Тип зв'язку	Опис
Categories	Dishes	1 : N	Одна категорія містить багато страв
Dishes	RecipeItems	1 : N	Одна страва може мати багато елементів рецептури
Ingredients	RecipeItems	1 : N	Один інгредієнт може використовуватися в багатьох рецептурах
Orders	OrderItems	1 : N	Одне замовлення містить багато позицій
Dishes	OrderItems	1 : N	Одна страва може входити до багатьох позицій замовлень
Ingredients	InventoryTransactions	1 : N	Один інгредієнт може мати багато складських операцій
Users	Orders	1 : N	Один працівник може створити багато замовлень
Users	Shifts	1 : N	Один працівник може відкривати або закривати касові зміни

Структура бази даних забезпечує збереження та коректне оновлення інформації. Зміни меню виконуються через адміністративну панель: клієнтська частина надсилає запит до API, сервер перевіряє дані та оновлює відповідні таблиці. Під час створення замовлення POS-інтерфейс передає список позицій на сервер, де перевіряється доступність страв, визначаються актуальні ціни, створюються записи в Orders і OrderItems, а після оплати замовлення передається до KDS.

Складський облік пов'язаний із рецептурами: система визначає потрібні інгредієнти через RecipeItems, списує залишки з Ingredients і фіксує операції в InventoryTransactions. Для збереження історичної цілісності використовується м'яке видалення через поле IsArchived.

Статуси й типи операцій зберігаються як рядкові значення, наприклад New, Paid, Ready, Completed. Для грошових значень використовується numeric(10,2), для кількості інгредієнтів – numeric(10,3), для дат – timestamp with time zone. Для пришвидшення запитів передбачаються індекси за зовнішніми ключами, статусами та датами. Усі операції з даними виконуються через серверний API, без прямого доступу клієнта до бази даних (рис. 2.5).

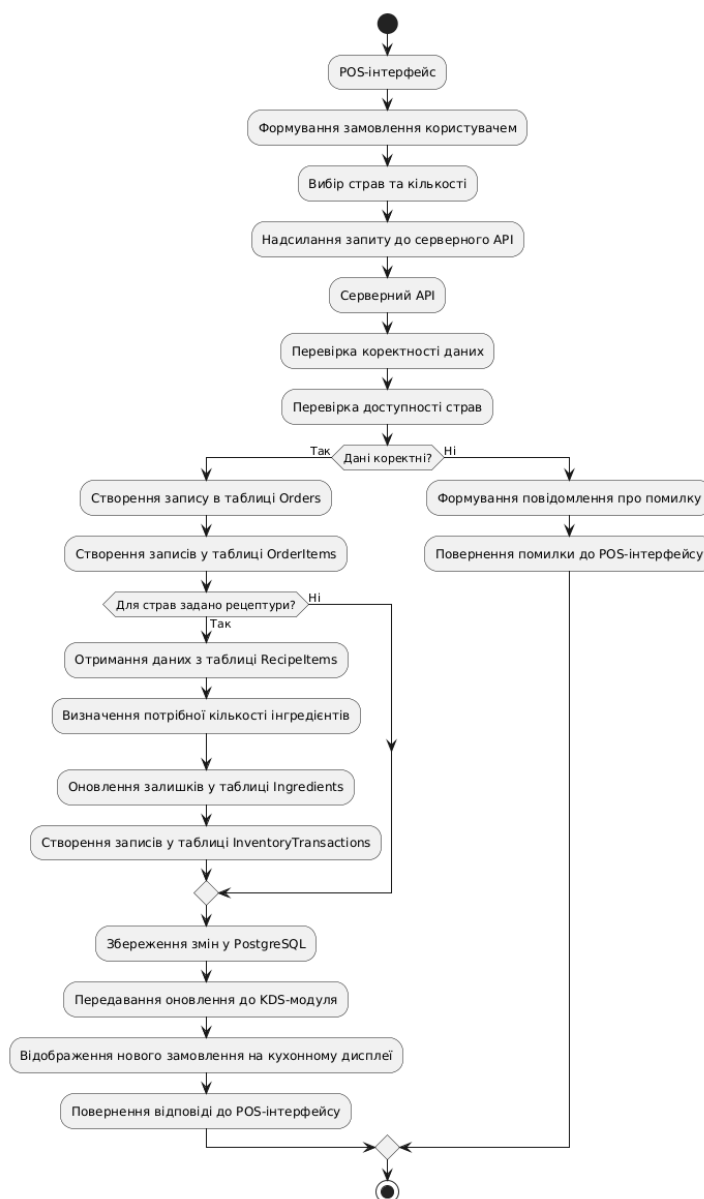


Рис. 2.5. Схема використання даних під час обробки замовлення

Отже, розроблена структура бази даних забезпечує збереження основних даних вебсистеми автоматизації роботи закладів харчування та відображає ключові бізнес-процеси закладу: керування меню, створення замовлень, роботу кухні, складський облік, рецептури та касові зміни. Використання PostgreSQL,

реляційних зв'язків, зовнішніх ключів, точних числових типів, часових міток, міграцій і підходу soft delete дозволить забезпечити цілісність, надійність і придатність бази даних до подальшого розвитку системи.

2.3. Проектування алгоритмів роботи вебсистеми

Алгоритми роботи системи враховують клієнт-серверну архітектуру: користувач взаємодіє з frontend-частиною, яка надсилає запити до серверного API. Сервер перевіряє права доступу, виконує бізнес-логіку, звертається до бази даних і повертає результат клієнту. Для KDS-модуля використовується SignalR, що забезпечує оновлення замовлень у реальному часі без перезавантаження сторінки.

Базовим алгоритмом є авторизація за PIN-кодом. Після введення PIN клієнт надсилає запит до API, сервер перевіряє хеш, визначає роль й формує JWT-токен. Роль визначає доступ до POS, KDS або адмін панелі (рис. 2.6).

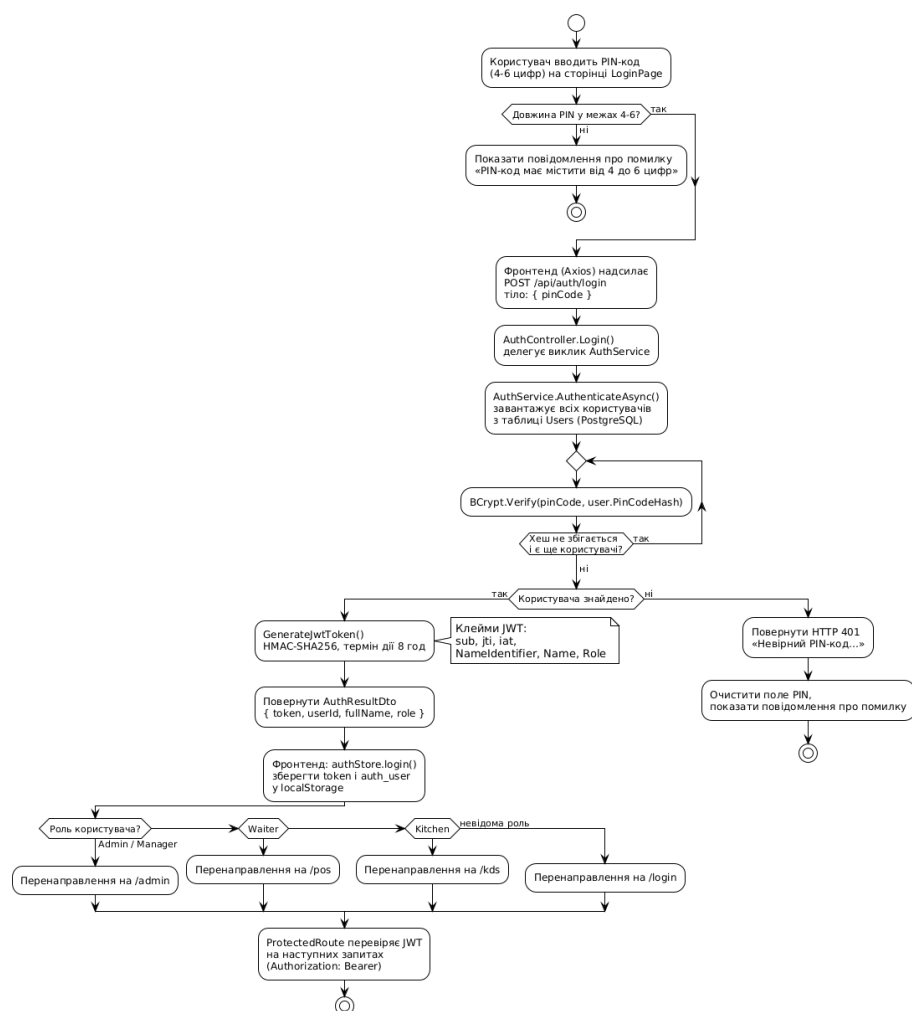


Рис. 2.6. Діаграма активності алгоритму авторизації користувача

Алгоритм створення замовлення починається з відкриття POS-модуля після авторизації. Система завантажує доступні категорії та страви через серверний API. Працівник обирає позиції, додає їх до замовлення, змінює кількість і перевіряє загальну суму. Після підтвердження клієнтська частина надсилає список позицій на сервер.

Сервер перевіряє доступність страв, визначає ціни, створює замовлення та його позиції, розраховує суму й зберігає дані. Після оплати статус змінюється, виконуються складські операції, а оновлення передається до KDS. Життєвий цикл замовлення має послідовність New → Paid → Ready → Completed (рис. 2.7).

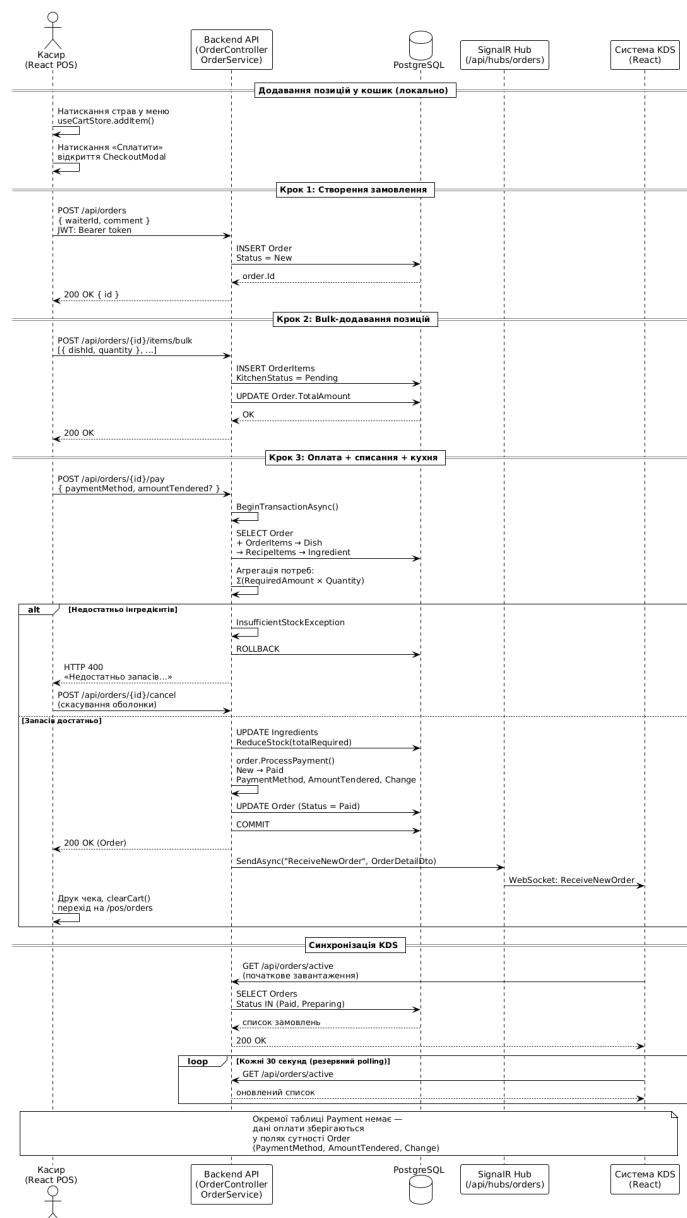


Рис. 2.7. Діаграма послідовності створення, оплати та передавання замовлення на кухню

KDS-модуль відображає активні замовлення для працівників кухні. Після відкриття інтерфейсу клієнтська частина отримує список замовлень через API та встановлює з'єднання із сервером для оновлень у реальному часі. Нове оплачене замовлення автоматично з'являється на екрані кухні. Працівник переглядає склад замовлення й змінює його статус на готове або завершене. Сервер перевіряє допустимість переходу між станами, оновлює базу даних і повертає актуальні дані (рис 2.8).

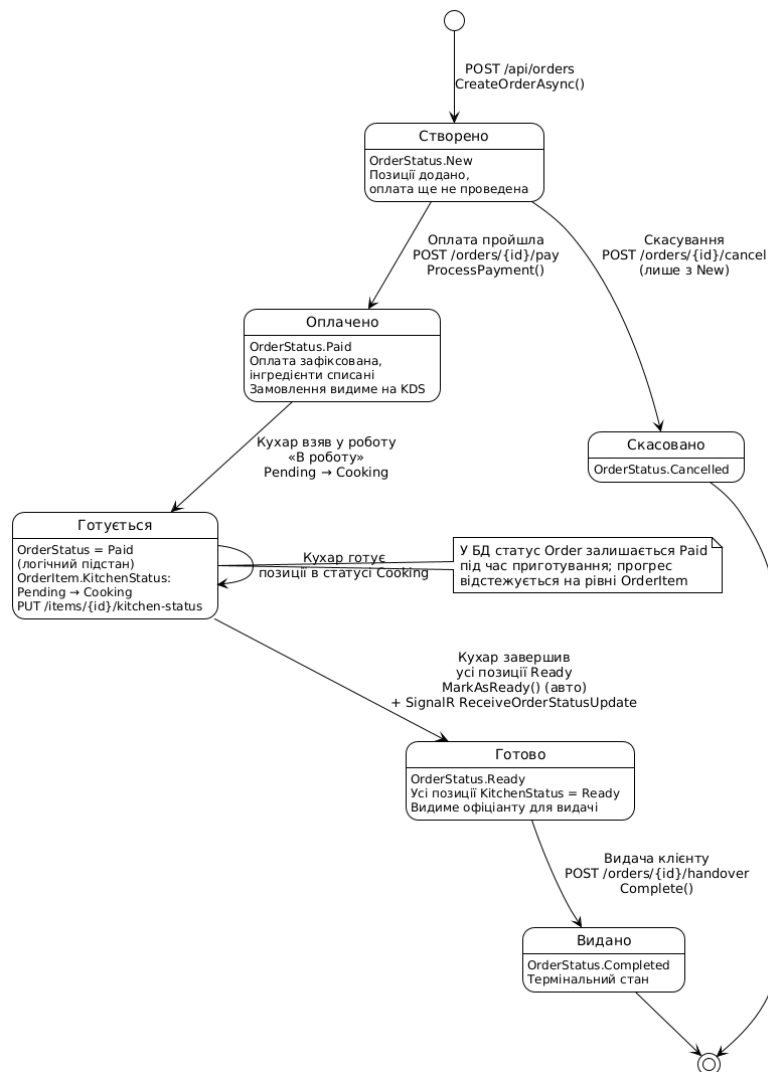


Рис. 2.8. Діаграма станів замовлення у вебсистемі

Алгоритм списання інгредієнтів пов'язує POS-модуль, рецептури та складський облік. Після оплати замовлення сервер визначає страви, отримує їхні рецептури, розраховує потрібну кількість інгредієнтів і перевіряє залишки. Якщо їх достатньо, система зменшує складські кількості та створює транзакції. Оплата, списання й створення складських операцій виконуються узгоджено в межах

транзакції бази даних. У разі помилки зміни скасовуються, а користувач отримує повідомлення (рис. 2.9).

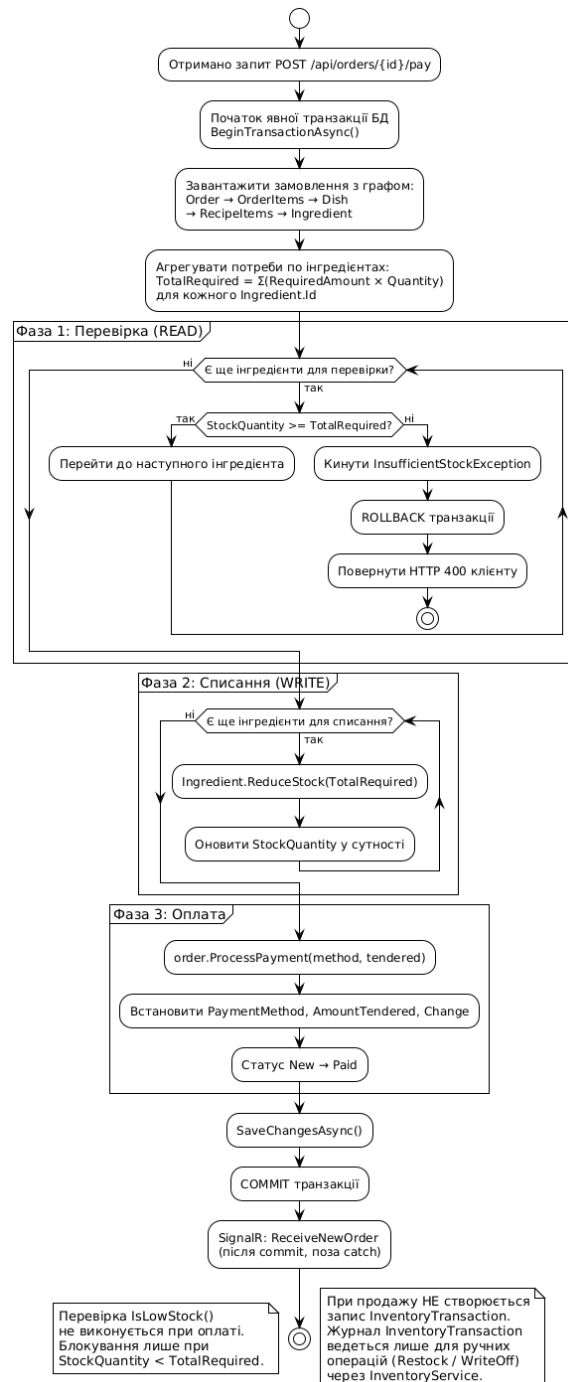


Рис. 2.9. Діаграма активності алгоритму списання інгредієнтів за рецептурою

Система підтримує алгоритми керування довідниковими даними: створення, редагування та архівацію категорій, страв, інгредієнтів і користувачів. Загальна послідовність таких операцій є однаковою: користувач відкриває потрібний розділ адміністративної панелі, вводить або змінює дані,

Також система підтримує формування друкованих документів: чеків, QR-коду меню, звітів касової зміни та службових форм. Для цього клієнтська частина отримує дані через API, формує окремий компонент друку, передає його до області #print-area і викликає стандартний механізм друку браузера. Для QR-меню формується публічне посилання, генерується SVG QR-код і створюється аркуш для друку. Взаємодію основних модулів подано на рисунку 2.10.

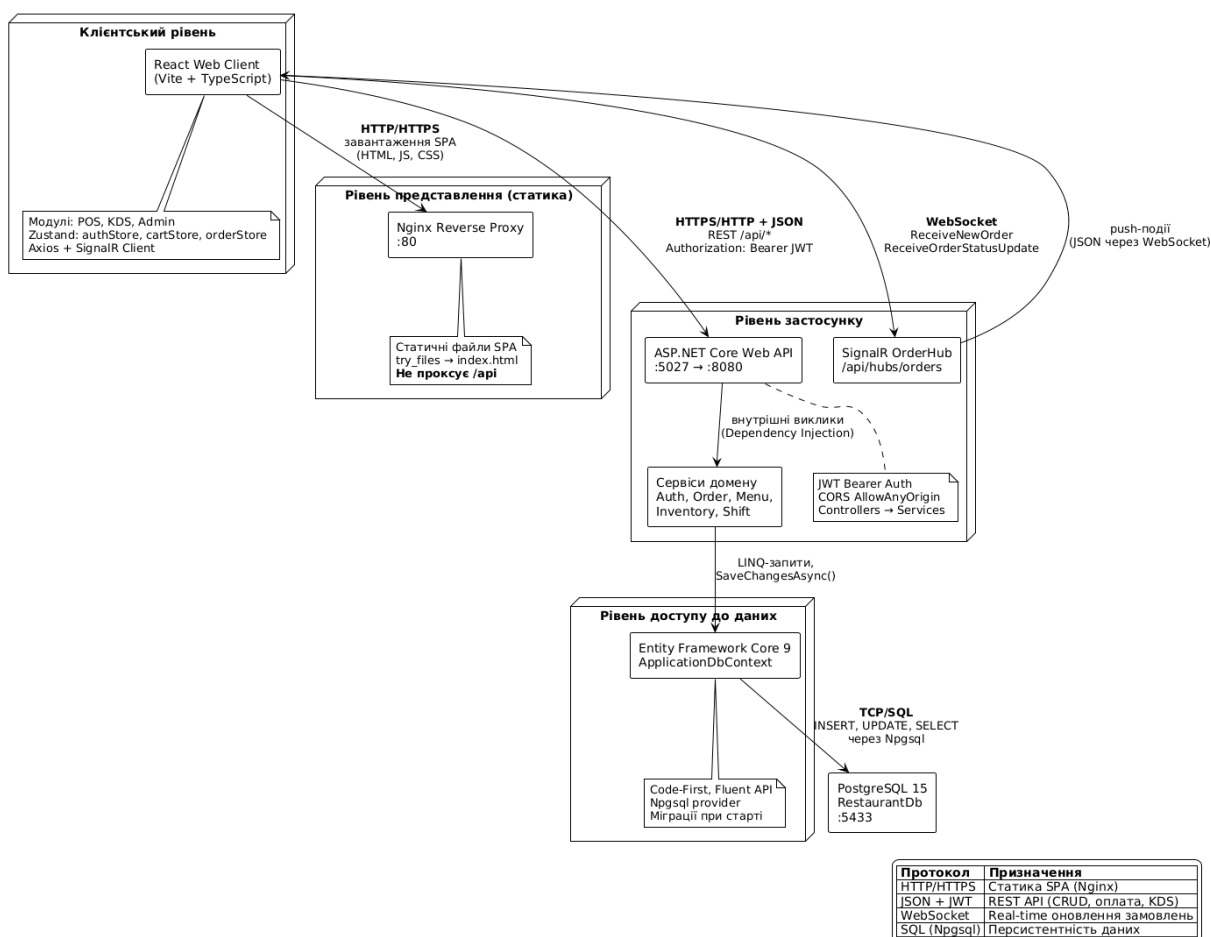


Рис. 2.10. Діаграма взаємодії програмних компонентів вебсистеми

Загалом алгоритми роботи системи можна поділити на три групи. До першої групи належать алгоритми взаємодії користувача з інтерфейсом: авторизація, вибір страв, створення замовлення, перегляд активних замовлень, керування довідниками та друк документів. До другої групи належать серверні алгоритми: перевірка прав доступу, валідація даних, обробка бізнес-операцій, зміна статусів, списання інгредієнтів і формування відповідей API. До третьої групи належать алгоритми роботи з даними: збереження замовлень, оновлення залишків, створення складських транзакцій, архівація записів і формування звітної інформації.

2.4. Реалізація вебсистеми автоматизації роботи закладів харчування

Реалізація вебсистеми автоматизації роботи закладів харчування здійснюється відповідно до обраної клієнт-серверної архітектури, описаної в попередніх підрозділах. Програмний комплекс складається з клієнтської частини, серверної частини, бази даних та засобів розгортання. Такий поділ дозволяє відокремити інтерфейс користувача від бізнес-логіки, централізувати роботу з даними та забезпечити можливість подальшого розширення системи.

Клієнтська частина реалізується як односторінковий застосунок на основі React, TypeScript та Vite. Вона відповідає за відображення інтерфейсу користувача, маршрутизацію між функціональними зонами, взаємодію з REST API, збереження частини клієнтського стану та роботу з механізмами друку. Серверна частина реалізується як ASP.NET Core Web API, що обробляє HTTP-запити, виконує бізнес-логіку, перевіряє права доступу, взаємодіє з базою даних через Entity Framework Core та надсилає оновлення до кухонного дисплея через SignalR. Для збереження даних використовується PostgreSQL.

У межах програмного комплексу реалізуються такі основні функціональні зони: адміністративна панель, POS-модуль, KDS-модуль, публічне QR-меню та модулі складського й касового обліку. Адміністративна панель використовується для керування меню, працівниками, складом, рецептурами та звітами. POS-модуль призначений для створення замовлень і проведення оплати. KDS-модуль використовується працівниками кухні для перегляду активних

замовлень і зміни їхніх статусів. Публічне QR-меню дає змогу клієнтам переглядати актуальне меню закладу без авторизації.

Структура серверної частини побудована за принципом розділення відповідальностей. У проєкті виділяються контролери, сервіси, DTO-об'єкти, доменні моделі, контекст бази даних, міграції та SignalR Hub. Контролери відповідають за приймання HTTP-запитів і повернення відповідей клієнту. Сервіси містять основну бізнес-логіку. DTO-об'єкти використовуються для передавання даних між клієнтом і сервером. Доменні моделі описують основні сутності системи: користувачів, категорії, страви, інгредієнти, рецептури, замовлення, позиції замовлень, складські операції та касові зміни.

Одним із важливих рішень у реалізації серверної частини є використання Service Layer Pattern. Бізнес-логіка не розміщується безпосередньо в контролерах, а виноситься в окремі сервіси. Це дозволяє зробити контролери простішими, зменшити дублювання коду та спростити тестування. Наприклад, логіка роботи із замовленнями зосереджується в сервісі замовлень, логіка складського обліку – у складському сервісі, логіка авторизації – у сервісі автентифікації.

Фрагмент реєстрації сервісів у контейнері залежностей наведено у Додатку А, оскільки він демонструє загальну структуру серверної частини та спосіб підключення основних сервісів.

Особливу увагу в реалізації системи слід приділити механізму авторизації. Для входу працівників використовується PIN-код, що є зручним для умов закладу харчування, де користувачі повинні швидко переходити до роботи з POS або KDS. При цьому PIN-код не зберігається у відкритому вигляді, а перевіряється через хешоване значення. Після успішної авторизації сервер формує JWT-токен, який містить ідентифікатор користувача, ім'я та роль. Надалі цей токен передається клієнтською частиною у заголовок запитів до захищених API-ендпоінтів.

З технічної точки зору складність цього фрагмента полягає в поєднанні швидкого PIN-входу з рольовою моделлю доступу. Система повинна не лише

перевірити правильність PIN-коду, а й визначити роль користувача, сформувати токен, передати його клієнту та забезпечити обмеження доступу до окремих функціональних зон. Код генерації JWT-токена та перевірки PIN-коду наведено у Додатку Б.

На клієнтській стороні рольовий доступ реалізується через захищені маршрути. Після входу в систему користувач отримує доступ лише до тих сторінок, які відповідають його ролі. Наприклад, працівник кухні повинен мати доступ до KDS-модуля, але не повинен мати доступу до адміністративної панелі. Офіціант або касир працює з POS-інтерфейсом, тоді як адміністратор і менеджер можуть мати розширені можливості керування системою. Для цього використовується компонент захищеного маршруту, який перевіряє наявність токена та роль користувача.

Складність цього фрагмента полягає в тому, що захист повинен працювати не лише на рівні API, а й на рівні клієнтської маршрутизації. Клієнтський застосунок має правильно перенаправляти користувача після входу, забороняти перехід до недоступних сторінок і не відображати зайві елементи інтерфейсу. Фрагмент маршрутизації та компонента захищеного маршруту наведено в Додатку В.

Найбільш складною частиною бізнес-логіки є процес створення, оплати та передавання замовлення на кухню. Цей процес поєднує декілька операцій: перевірку позицій замовлення, створення запису замовлення, створення позицій замовлення, зміну статусу, списання інгредієнтів за рецептурою, збереження складських транзакцій, фіксацію змін у базі даних і надсилення оновлення до KDS-модуля. Оскільки ці дії пов'язані між собою, вони повинні виконуватися узгоджено.

У разі помилки система не повинна залишати дані в частково зміненому стані. Наприклад, неприпустимою є ситуація, коли замовлення отримало статус оплаченого, але складські залишки не були списані або складські транзакції не були створені. Тому такі операції мають виконуватися в межах транзакції бази

даних. Якщо будь-який етап завершується помилкою, усі зміни повинні бути скасовані.

Окремо слід врахувати взаємодію з KDS-модулем. Після успішного завершення операції сервер повинен передати інформацію про нове або оновлене замовлення підключеним клієнтам кухонного дисплея. Для цього використовується SignalR, який дозволяє надсилати повідомлення клієнтам без постійного опитування сервера. Саме цей фрагмент є одним із ключових для пояснення, оскільки він демонструє взаємодію бізнес-логіки, бази даних і real-time оновлень. Лістинг коду наведено в Додатку Г.

Складський облік реалізується через сутності інгредієнтів, рецептур і складських транзакцій. Для кожної страви може бути задана рецептура, яка визначає перелік інгредієнтів і їхню кількість. Після продажу страви система визначає необхідні інгредієнти, перевіряє їхню наявність, зменшує залишки та створює записи про рух складу. Такий підхід дозволяє пов'язати POS-продажі з фактичним обліком інгредієнтів.

Важливою особливістю складського модуля є використання м'якого видалення. Інгредієнти, які були використані в рецептурах або складських операціях, не повинні фізично видалятися з бази даних, оскільки це може порушити історичну цілісність даних. Замість цього запис позначається як архівований і приховується з активних списків. Такий підхід дозволяє зберегти історію складських операцій і водночас не відображати застарілі позиції в робочому інтерфейсі. Реалізація м'якого видалення інгредієнтів і фільтрації активних складських позицій наведено в Додатку Г.

Для ефективної роботи з великими списками даних у системі використовується серверна пагінація та фільтрація. Це особливо важливо для адміністративної панелі, де можуть відображатися списки категорій, страв, інгредієнтів, працівників або замовлень. Замість передавання всіх записів на клієнт сервер повертає лише частину даних відповідно до поточної сторінки, розміру сторінки та параметрів пошуку. Такий підхід зменшує навантаження на мережу й покращує швидкість роботи інтерфейсу.

Складність цього фрагмента полягає в правильному поєднанні пошуку, сортування, підрахунку загальної кількості записів і вибірки тільки потрібної сторінки. Для цього на сервері використовується формування запиту до бази даних, підрахунок загальної кількості записів, застосування Skip і Take, а також створення уніфікованої пагінованої відповіді. Реалізація серверної пагінації та фільтрації даних наведено в Додатку Д.

Клієнтська частина системи реалізує декілька функціональних зон. POS-модуль забезпечує перегляд меню, додавання позицій до замовлення, розрахунок підсумкової суми та підтвердження продажу. KDS-модуль відображає активні замовлення для кухні та дозволяє змінювати їхні статуси. Адміністративна панель використовується для керування меню, складом, працівниками, звітністю та QR-меню. Публічне меню доступне клієнтам без авторизації та відображає актуальні позиції меню.

Для взаємодії frontend із backend використовується централізований HTTP-клієнт. У ньому налаштовується базова адреса API, автоматичне додавання JWT-токена до запитів та обробка помилок авторизації. Якщо сервер повертає помилку 401, клієнтська частина повинна очистити стан авторизації та перенаправити користувача на сторінку входу. Це дозволяє уникнути дублювання логіки авторизації в різних компонентах. Налаштування Axios-клієнта з JWT-токеном та обробкою помилки авторизації наведено в Додатку Е.

Окремого пояснення потребує реалізація друку QR-коду, чеків і звітів. У звичайному вебінтерфейсі є багато елементів, які не повинні потрапляти до друкованої форми: навігація, кнопки, панелі керування, фонові блоки. Тому для друку використовується окрема область, у яку передається лише той React-компонент, який потрібно надрукувати. Після цього викликається стандартний механізм друку браузера.

Для QR-меню генерується публічне посилання на сторінку меню, після чого воно перетворюється на QR-код у форматі SVG. Векторний формат забезпечує якісний друк незалежно від масштабу. Ізольований підхід до друку дозволяє однаково формувати QR-код, чеки, refund-документи та X/Z-звіти

касової зміни. Генерація QR-коду меню та друк через ізольовану область `#print-area` наведено в Додатку Є.

Завершальним елементом реалізації є налаштування розгортання системи. Для цього використовується контейнеризований підхід, у якому frontend, backend і база даних запускаються як окремі сервіси. Frontend-застосунок після збирання обслуговується через Nginx, backend працює як окремий API-сервіс, а PostgreSQL зберігає дані системи. Такий підхід спрощує запуск проєкту, забезпечує ізоляцію компонентів і дозволяє відтворити однакове середовище локально та на сервері.

Для автоматизації процесів збирання, тестування та розгортання доцільно використовувати CI/CD-конвеєр. Він дозволяє після внесення змін виконати збірку backend і frontend, запустити тести, а потім розгорнути оновлену версію системи на сервері. У межах дипломної роботи цей фрагмент є важливим, оскільки демонструє не лише розробку програмної логіки, а й підготовку системи до практичного використання. Фрагмент коду налаштування Docker Compose для запуску frontend, backend і PostgreSQL, а також налаштування GitLab CI/CD для збирання, тестування та розгортання системи наведено в Додатку Ж.

Отже, реалізація вебсистеми автоматизації роботи закладів харчування охоплює створення клієнтської частини, серверної API-частини, реляційної бази даних і середовища розгортання. Найбільш складними з технічної точки зору є фрагменти, пов'язані з авторизацією, рольовим доступом, транзакційною обробкою замовлень, списанням інгредієнтів, оновленням KDS у реальному часі, м'яким видаленням, серверною пагінацією, друком QR-коду та контейнеризованим розгортанням. Саме ці частини коду потребують окремого пояснення та будуть наведені в додатках до випускної роботи.

Висновки до другого розділу

У другому розділі було виконано проєктування вебсистеми автоматизації роботи закладів харчування. На основі аналізу предметної області визначено

основні функції системи, типи користувачів, структуру даних, алгоритми роботи та особливості реалізації програмного комплексу.

Було встановлено основні ролі користувачів: адміністратор, менеджер, працівник POS-зони, працівник кухні та клієнт. Для кожної ролі визначено доступні функції: керування меню, складом, працівниками, звітністю, створення замовлень, робота з кухонним дисплеєм і перегляд публічного QR-меню. Також сформульовано функціональні та нефункціональні вимоги до швидкодії, захищеності, зручності використання, цілісності даних і можливості розширення.

Під час проєктування об'єктно-орієнтованої структури визначено основні сутності системи: користувач, категорія, страва, інгредієнт, рецептура, замовлення, позиція замовлення, складська транзакція та касова зміна. На їх основі спроектовано реляційну базу даних PostgreSQL, описано таблиці, зв'язки, зовнішні ключі, типи даних і механізм м'якого видалення.

Окремо розглянуто алгоритми авторизації, створення й оплати замовлення, передавання його на KDS, зміни статусів, списання інгредієнтів, архівації довідників і друку документів. Також описано структуру клієнтської та серверної частин системи й визначено складні фрагменти коду для винесення в додатки.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБСИСТЕМОЮ АВТОМАТИЗАЦІЇ РОБОТИ ЗАКЛАДІВ ХАРЧУВАННЯ

3.1. Порядок встановлення та налаштування параметрів системи

Після проєктування архітектури, бази даних, алгоритмів роботи та програмних модулів необхідно описати порядок встановлення й налаштування вебсистеми автоматизації роботи закладів харчування. Цей підрозділ визначає вимоги до середовища виконання, послідовність підготовки програмного комплексу до запуску, налаштування параметрів підключення, порядок розгортання системи та перевірку її працездатності.

Вебсистема автоматизації роботи закладів харчування є клієнт-серверним програмним комплексом, що складається з трьох основних частин: клієнтського застосунку, серверного API та бази даних. Клієнтська частина відповідає за інтерфейс користувача та надає доступ до адміністративної панелі, POS-модуля, кухонного дисплея й публічного QR-меню. Серверна частина обробляє запити, виконує бізнес-логіку, перевіряє права доступу, взаємодіє з базою даних і забезпечує оновлення замовлень у реальному часі. База даних використовується для збереження інформації про користувачів, меню, замовлення, складські операції, рецептури та касові зміни.

Для встановлення системи використано контейнеризований підхід. У цьому випадку кожен компонент програмного комплексу запускається в окремому контейнері: база даних PostgreSQL, backend-застосунок ASP.NET Core Web API та frontend-застосунок React, який у production-середовищі обслуговується через Nginx. Такий підхід спрощує встановлення, оскільки не потребує ручного налаштування всіх залежностей на сервері, а також забезпечує однакове середовище запуску на локальному комп'ютері та на віддаленому сервері (рис. 3.1.)

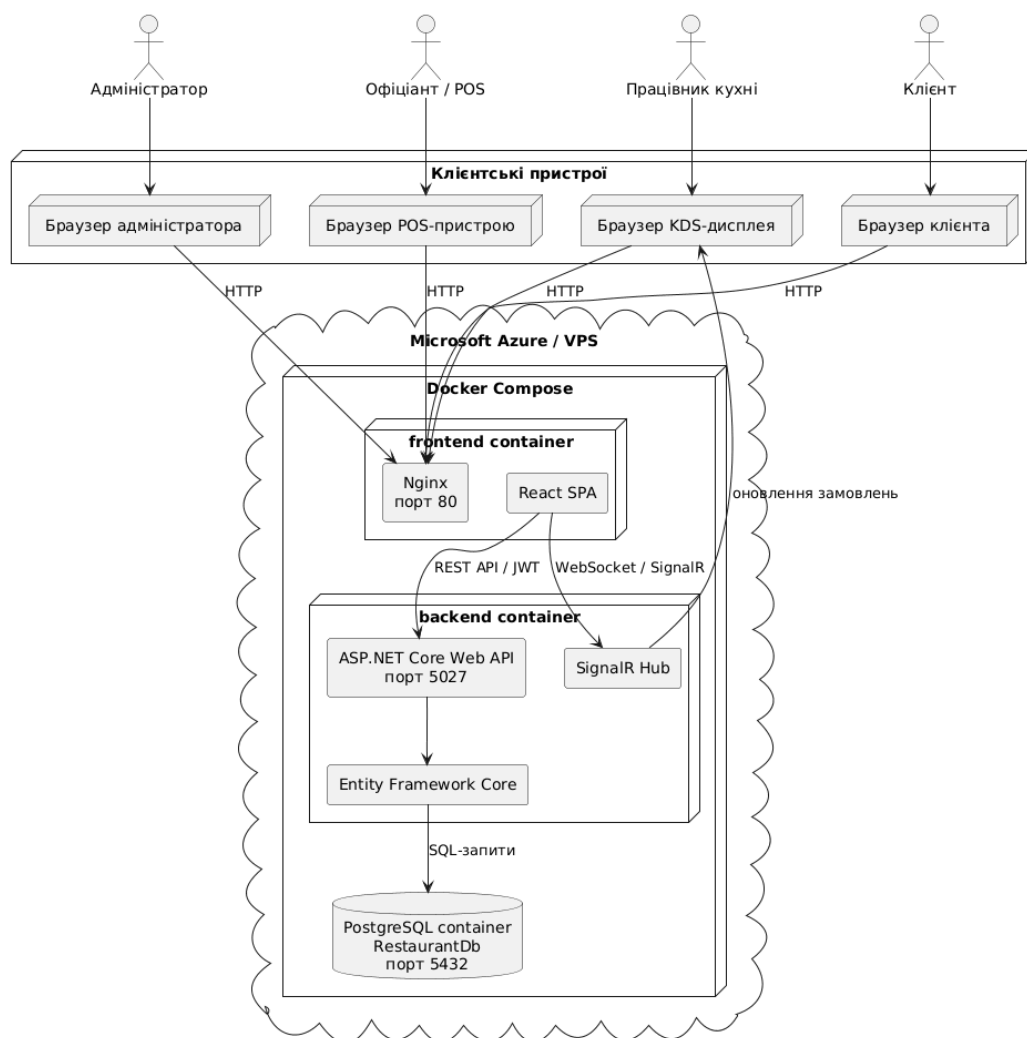


Рис. 3.1. Діаграма розгортання вебсистеми

Перед встановленням системи необхідно підготувати програмне середовище. На комп'ютері розробника або сервері повинні бути встановлені Git, Docker, Docker Compose, а також доступ до репозиторію з вихідним кодом системи. Для локальної розробки додатково можуть використовуватися Node.js, .NET SDK, редактор коду та pgAdmin для перегляду бази даних [29]. Однак для production-запуску через Docker Compose достатньо встановити Docker, Docker Compose і мати доступ до файлів проєкту.

Орієнтовні вимоги до програмного середовища наведено в таблиці 3.1.

Таблиця 3.1

Програмні засоби, необхідні для встановлення системи

Програмний засіб	Призначення
Git	Завантаження вихідного коду з репозиторію
Docker	Запуск окремих компонентів системи в контейнерах
Docker Compose	Одноточасний запуск frontend, backend і бази даних

1.	2.
PostgreSQL / pgAdmin	Перегляд і адміністрування бази даних під час розробки
Node.js	Локальний запуск і збірка frontend-частини без Docker
.NET SDK	Локальний запуск і збірка backend-частини без Docker
Nginx	Обслуговування статичних файлів frontend у production-середовищі
GitLab CI/CD	Автоматизація процесів збирання, тестування та розгортання

Першим етапом встановлення є отримання вихідного коду програмного комплексу. Для цього користувач або адміністратор сервера повинен клонувати репозиторій проєкту за допомогою Git. Після цього необхідно перейти до кореневого каталогу проєкту, у якому розташовані файли конфігурації Docker Compose, каталоги frontend і backend, а також допоміжні файли для розгортання.

Приклад послідовності дій для отримання вихідного коду:

```
git clone <адреса_репозиторію>
cd restaurant-management-system
```

Після завантаження проєкту необхідно налаштувати змінні середовища. Вони визначають параметри підключення до бази даних, адресу серверного API для клієнтського застосунку, параметри JWT-автентифікації та інші службові налаштування. Зберігання таких параметрів у змінних середовища є доцільним, оскільки дозволяє не фіксувати конфіденційні дані безпосередньо у вихідному коді.

До основних параметрів налаштування належать:

- *ConnectionString__DefaultConnection* – рядок підключення backend-частини до PostgreSQL;
- *POSTGRES_DB* – назва бази даних;
- *POSTGRES_USER* – ім'я користувача бази даних;
- *POSTGRES_PASSWORD* – пароль користувача бази даних;
- *Jwt__Key* – секретний ключ для підпису JWT-токенів;
- *Jwt__Issuer* – видавець токена;
- *Jwt__Audience* – цільова аудиторія токена;
- *Jwt__ExpiryHours* – час дії токена;

- *VITE_API_URL* – адреса backend API, яку використовує frontend-застосунок.

Приклад структури конфігураційного файлу .env може мати такий вигляд:

POSTGRES_DB=RestaurantDb

POSTGRES_USER=postgres

POSTGRES_PASSWORD=postgres

ConnectionStrings__DefaultConnection=Host=db;Port=5432;Database=RestaurantDb;Username=postgres;Password=postgres

Jwt__Key=replace_this_secret_key_with_long_random_value

Jwt__Issuer=RestaurantManagementSystem

Jwt__Audience=RestaurantManagementSystemUsers

Jwt__ExpiryHours=8

VITE_API_URL=http://localhost:5027

У production-середовищі значення паролів, секретних ключів і адрес повинні бути змінені. Особливо важливо замінити *Jwt__Key* на довгий випадковий рядок, оскільки він використовується для підпису токенів доступу. Також у production-середовищі для *VITE_API_URL* потрібно вказати фактичну адресу сервера або домен, за яким frontend буде звертатися до backend API.

Наступним етапом є запуск системи за допомогою Docker Compose. У конфігурації розгортання передбачається запуск трьох основних сервісів: db, backend і frontend. Сервіс db використовує образ PostgreSQL і відповідає за збереження даних. Сервіс backend збирає та запускає ASP.NET Core Web API. Сервіс frontend збирає React-застосунок і розміщує його у контейнері з Nginx. У технічному описі проєкту зазначено, що Docker Compose використовується саме для оркестрації трьох сервісів: бази даних, backend і frontend.

Для запуску системи використовується команда:

docker compose up --build -d

Параметр *--build* означає, що Docker має виконати збірку образів перед запуском контейнерів. Параметр *-d* запускає контейнери у фоновому режимі. Після виконання цієї команди Docker Compose створить і запустить усі необхідні

сервіси. У разі першого запуску процес може тривати довше, оскільки необхідно завантажити базові образи, встановити залежності та зібрати frontend і backend.

Перевірити стан контейнерів можна командою:

```
docker compose ps
```

Для перегляду журналів роботи системи використовується команда:

```
docker compose logs -f
```

За потреби можна переглядати журнали окремого сервісу:

```
docker compose logs -f backend
```

```
docker compose logs -f frontend
```

```
docker compose logs -f db
```

Після успішного запуску система має бути доступна через браузер. Frontend-застосунок у production-конфігурації обслуговується через Nginx на порту 80, backend API працює на окремому порту, а PostgreSQL доступний усередині Docker-мережі та, за потреби, може бути відкритий на окремому зовнішньому порту для адміністрування. Frontend-контейнер використовує Nginx, backend публікується через порт 5027, а PostgreSQL у Docker має внутрішній порт 5432 і може бути прокинутий назовні, наприклад як 5433:5432.

Орієнтовні адреси для перевірки роботи системи наведено в таблиці 3.2.

Таблиця 3.2

Орієнтовні адреси доступу до компонентів системи

Компонент	Призначення	Адреса
Frontend	Вебінтерфейс системи	http://localhost
Backend API	Серверний API	http://localhost:5027
Swagger	Документація API в режимі розробки	http://localhost:5027/swagger
PostgreSQL	База даних для підключення через pgAdmin	localhost:5433
Public Menu	Публічне QR-меню	http://localhost/menu
Login	Сторінка авторизації	http://localhost/login

Після запуску backend-сервісу повинні бути застосовані міграції бази даних. У проєкті використовується підхід Entity Framework Core Code-First, тому структура таблиць формується на основі доменних моделей і міграцій. Міграції можуть застосовуватися автоматично під час старту backend-застосунку або вручну через команду Entity Framework Core у середовищі розробки. Міграції

бази даних застосовуються при старті backend-контейнера через програмний код застосунку, а не окремо в CI/CD-конвеєрі.

Для локальної розробки без Docker, backend можна запускати окремо з каталогу серверної частини. У такому випадку необхідно мати встановлений .NET SDK і доступну базу даних PostgreSQL. Команди можуть мати такий вигляд:

```
cd backend/Restaurant.API
```

```
dotnet restore
```

```
dotnet build
```

```
dotnet run
```

Для застосування міграцій вручну може використовуватися команда:

```
dotnet ef database update
```

Frontend-частину під час локальної розробки можна запускати окремо через Node.js. Для цього необхідно перейти до каталогу frontend, встановити залежності та запустити dev-сервер:

```
cd frontend
```

```
npm install
```

```
npm run dev
```

У цьому режимі frontend буде працювати через Vite dev-server, а API-запити можуть проксіюватися на backend або використовувати адресу, задану в змінній VITE_API_URL. У конфігурації Vite для локальної розробки може використовуватися проксі /api до локального backend-сервера, що спрощує взаємодію frontend і API під час розробки.

Після запуску системи необхідно виконати первинну перевірку працездатності. Насамперед перевіряється доступність сторінки входу, коректність авторизації за PIN-кодом, перенаправлення користувача відповідно до його ролі, відкриття POS-модуля, адміністративної панелі, кухонного дисплея та публічного QR-меню. Далі потрібно перевірити взаємодію з API: отримання списку категорій і страв, створення замовлення, передавання його до KDS-модуля, зміну статусу та збереження даних у базі.

Окремо необхідно перевірити підключення до бази даних через pgAdmin. Для цього потрібно створити нове підключення до сервера PostgreSQL, вказавши адресу, порт, назву бази, користувача та пароль відповідно до параметрів .env або docker-compose.yml. Після підключення можна переглянути таблиці бази даних, переконатися в наявності сутностей Users, Categories, Dishes, Ingredients, RecipeItems, Orders, OrderItems, InventoryTransactions і Shifts, а також перевірити появу нових записів після виконання операцій у вебінтерфейсі (рис. 3.2).

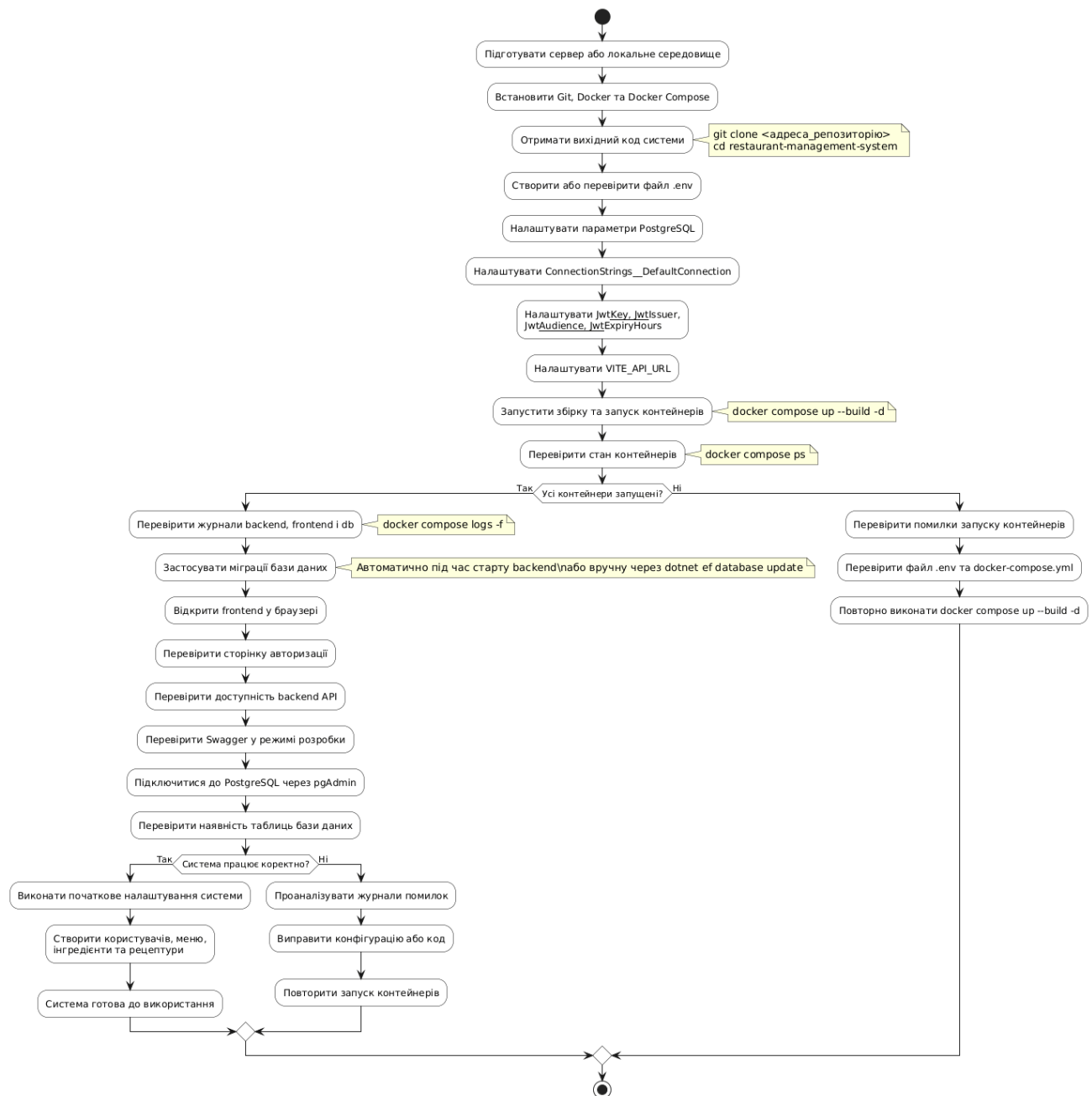


Рис. 3.2. Діаграма активності процесу встановлення та запуску вебсистеми

Після встановлення системи необхідно виконати початкове налаштування параметрів. До таких параметрів належать облікові записи користувачів, ролі, категорії меню, страви, інгредієнти, рецептури, складські залишки та параметри касової зміни. Первинне наповнення даних може виконуватися автоматично через механізм початкового заповнення бази даних або вручну через адміністративну панель.

Початкове налаштування системи включає такі кроки:

1. створення або перевірка облікового запису адміністратора;
2. налаштування ролей користувачів;
3. додавання працівників;
4. створення категорій меню;
5. додавання страв і цін;
6. створення інгредієнтів і встановлення початкових залишків;
7. формування рецептур для страв;
8. перевірка публічного QR-меню;
9. генерація та друк QR-коду;
10. відкриття касової зміни;
11. створення тестового замовлення;
12. перевірка появи замовлення на KDS.

Цю послідовність можна описати як базовий сценарій введення системи в експлуатацію. Вона дозволяє підготувати програмний комплекс до роботи в закладі харчування та перевірити зв'язок між основними модулями (рис. 3.3).

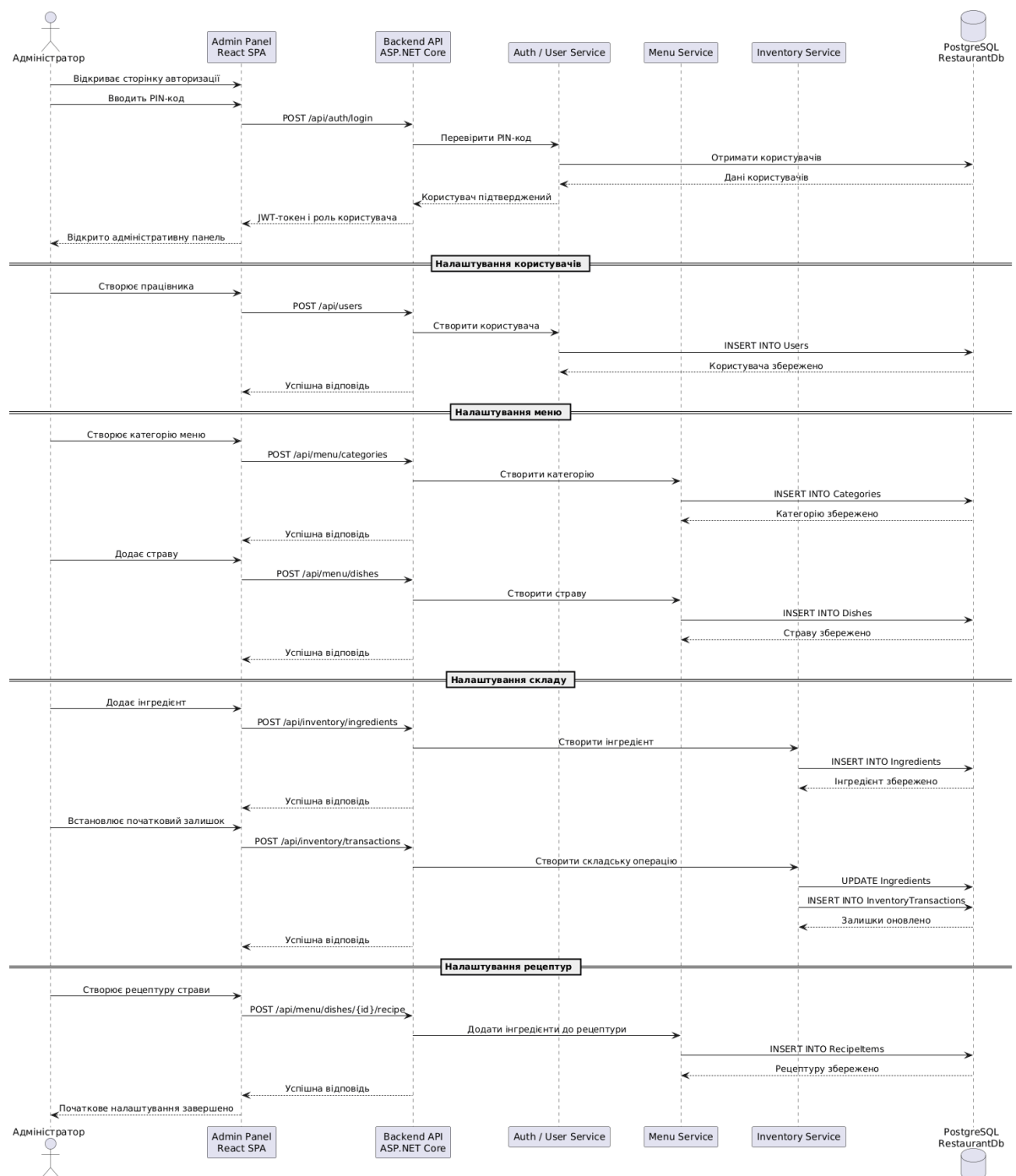


Рис. 3.3. Діаграма послідовності початкового налаштування параметрів системи

У production-середовищі також необхідно врахувати параметри безпеки. Паролі до бази даних, JWT-ключі та інші секрети не повинні зберігатися у відкритому доступі або потрапляти в публічний репозиторій. Для доступу до сервера бажано використовувати SSH-ключі, а не парольну авторизацію. Також потрібно обмежити відкриті порти лише тими, які необхідні для роботи системи: порт frontend, порт backend API та, за потреби, порт бази даних для

адміністрування. Якщо база даних не повинна бути доступною ззовні, її порт не потрібно відкривати за межі Docker-мережі.

Для зупинки системи використовується команда:

```
docker compose down
```

Якщо потрібно зупинити контейнери без видалення ресурсів, можна скористатися командою:

```
docker compose stop
```

Для повторного запуску:

```
docker compose up -d
```

Під час оновлення системи на сервері адміністратор повинен отримати останні зміни з репозиторію, повторно зібрати контейнери та запустити їх. Орієнтовна послідовність оновлення може бути такою:

```
git pull
```

```
docker compose build --no-cache
```

```
docker compose up -d
```

У разі використання GitLab CI/CD ці дії можуть виконуватися автоматично після внесення змін у головну гілку репозиторію. Конвеєр складається зі стадій build, test і deploy, а під час deploy виконується підключення до Azure VPS, оновлення коду, повторна збірка контейнерів і запуск через Docker Compose.

Для забезпечення стабільної роботи системи необхідно передбачити резервне копіювання бази даних. Оскільки PostgreSQL зберігає критичні дані про користувачів, замовлення, склад, рецептури та зміни, втрата бази даних може призвести до втрати важливої інформації. Резервне копіювання може виконуватися засобами PostgreSQL, наприклад через pg_dump, або через автоматизовані сценарії на сервері.

Приклад створення резервної копії:

```
docker exec -t <db_container_name> pg_dump -U postgres RestaurantDb > backup.sql
```

Для відновлення з резервної копії може використовуватися команда:

```
cat backup.sql | docker exec -i <db_container_name> psql -U postgres -d RestaurantDb
```

Отже, встановлення та налаштування вебсистеми автоматизації роботи закладів харчування передбачає підготовку середовища, отримання вихідного коду, налаштування змінних середовища, запуск контейнерів, застосування міграцій, перевірку роботи frontend, backend і бази даних, а також початкове наповнення системи довідниковими даними. Використання Docker Compose, Nginx, PostgreSQL і CI/CD-підходу дозволяє зробити процес розгортання відтворюваним, зменшити кількість ручних операцій і забезпечити стабільну роботу програмного комплексу.

3.2. Структура інтерфейсу вебсистеми автоматизації роботи закладів харчування

Інтерфейс вебсистеми автоматизації роботи закладів харчування розроблявся з урахуванням поділу користувачів на ролі та функціональні зони. Основне завдання інтерфейсу полягає в тому, щоб забезпечити швидке виконання типових операцій: авторизацію працівника, створення замовлення, перегляд активних замовлень на кухні, керування меню, складом, працівниками, касовими змінами та публічним QR-меню. Оскільки система призначена для використання в умовах закладу харчування, інтерфейс повинен бути простим, зрозумілим, швидким у роботі та адаптованим до різних пристроїв.

Клієнтська частина системи реалізується як односторінковий вебзастосунок. Це означає, що користувач взаємодіє з різними сторінками системи без повного перезавантаження вебдодатку. Навігація між функціональними зонами відбувається засобами клієнтської маршрутизації, а дані завантажуються з серверного API. Такий підхід дозволяє швидко перемикатися між розділами, зменшити затримки в роботі інтерфейсу та забезпечити зручну взаємодію з POS-терміналом, кухонним дисплеєм і адміністративною панеллю.

Першим екраном для працівників системи є сторінка авторизації. Вона призначена для входу користувача за PIN-кодом. Такий спосіб авторизації є

зручним для закладу харчування, оскільки працівнику не потрібно щоразу вводити складний логін і пароль. Після введення PIN-коду система перевіряє його, визначає роль користувача та перенаправляє його до відповідної функціональної зони. Наприклад, офіціант або касир переходить до POS-інтерфейсу, працівник кухні – до KDS-модуля, адміністратор або менеджер – до адміністративної панелі або доступних їм розділів.

Сторінка авторизації містить мінімальну кількість елементів: поле для введення PIN-коду, кнопку входу та повідомлення про помилку в разі неправильного введення. Завдяки цьому інтерфейс входу залишається простим і не перевантажує користувача зайвою інформацією (рис. 3.4).

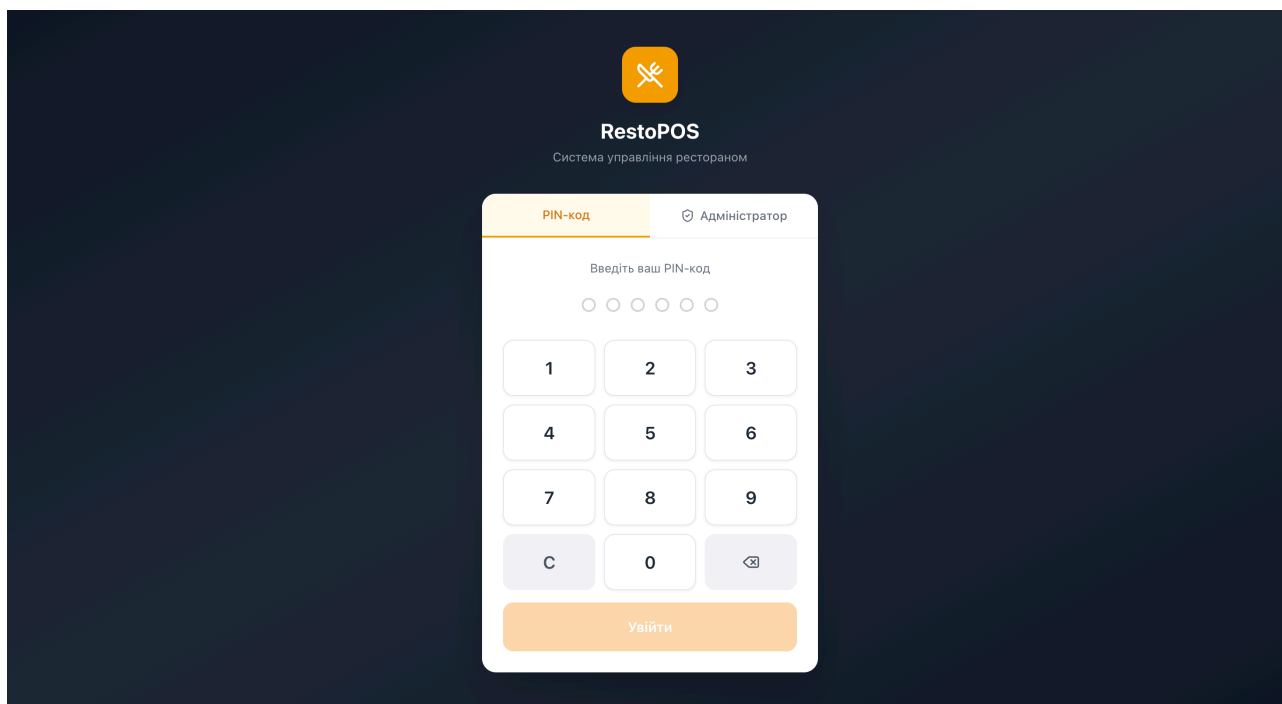


Рис. 3.4. Сторінка авторизації користувача за PIN-кодом

Після успішної авторизації користувач отримує доступ лише до тих сторінок, які відповідають його ролі. У системі передбачається розмежування доступу між адміністраторами, менеджерами, офіціантами та працівниками кухні. Захист маршруту здійснюється на рівні клієнтської частини та дублюється на рівні серверного API. Це означає, що користувач не лише не бачить недоступні йому сторінки в інтерфейсі, а й не може виконати заборонені API-операції. У технічній структурі проєкту рольова маршрутизація є частиною

frontend-рівня, а backend використовує JWT і role-based authorization для захисту ендпоінтів.

Адміністративна панель є центральним інтерфейсом для керування системою. Вона призначена для адміністратора та менеджера й містить розділи для роботи з меню, стравами, категоріями, складом, інгредієнтами, рецептурами, працівниками, касовими зміними, звітами та QR-меню. Структура адміністративної панелі побудована так, щоб користувач міг швидко перейти до потрібного розділу через навігаційне меню.

У бічній частині адміністративної панелі розміщено основні навігаційні елементи. Робоча область сторінки відображає таблиці, форми або аналітичні блоки залежно від вибраного розділу (рис. 3.5). Наприклад, у розділі меню користувач бачить список категорій і страв, у складському розділі – інгредієнти та залишки, у розділі персоналу – список працівників і їхні ролі.

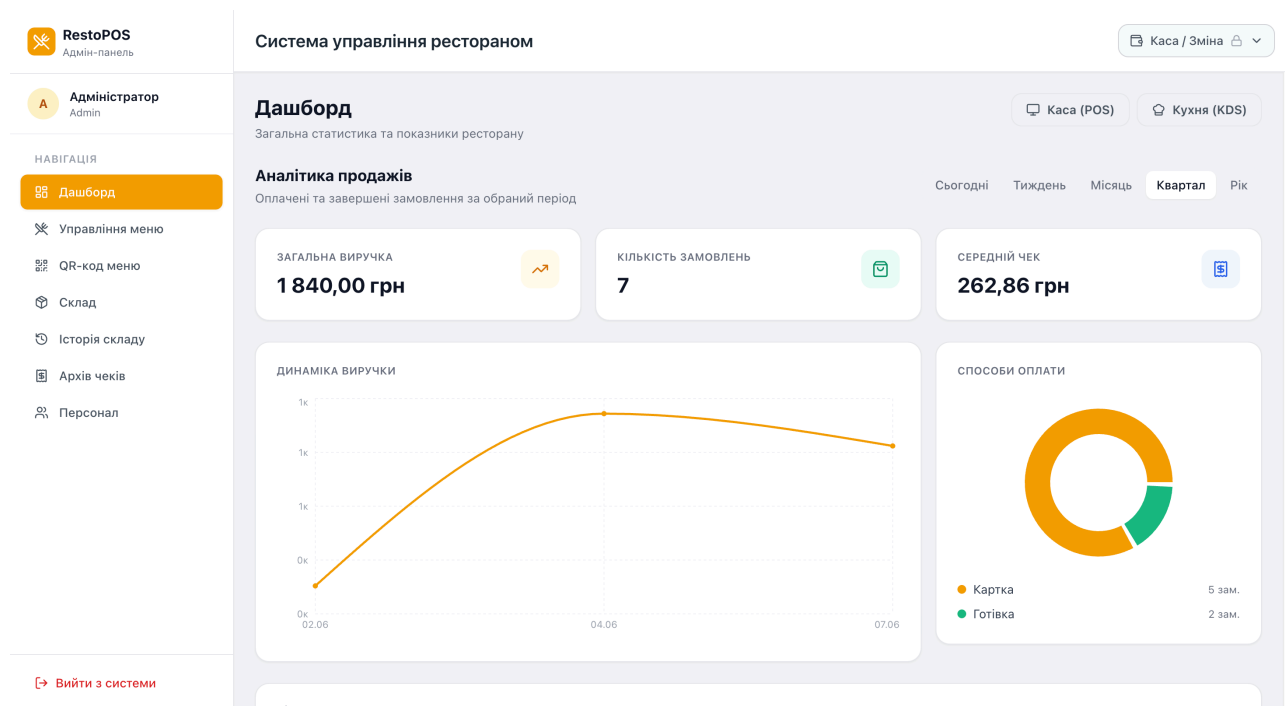


Рис. 3.5. Загальний вигляд адміністративної панелі вебсистеми

Одним із важливих розділів адміністративної панелі є керування меню. У цьому розділі адміністратор або менеджер може створювати категорії, додавати страви, змінювати назви, описи, ціни та доступність позицій. Для зручності роботи з великими списками використано таблиці, пошук, фільтрацію, пагінацію та модальні вікна для створення або редагування записів.

Інтерфейс керування стравами містить список позицій меню, кнопку додавання нової страви, елементи редагування, перемикач доступності та можливість архівації або видалення (рис.3.6). Оскільки страви відображаються не лише в адміністративній панелі, а й у POS-модулі та публічному QR-меню, зміни в цьому розділі повинні впливати на інші частини системи.

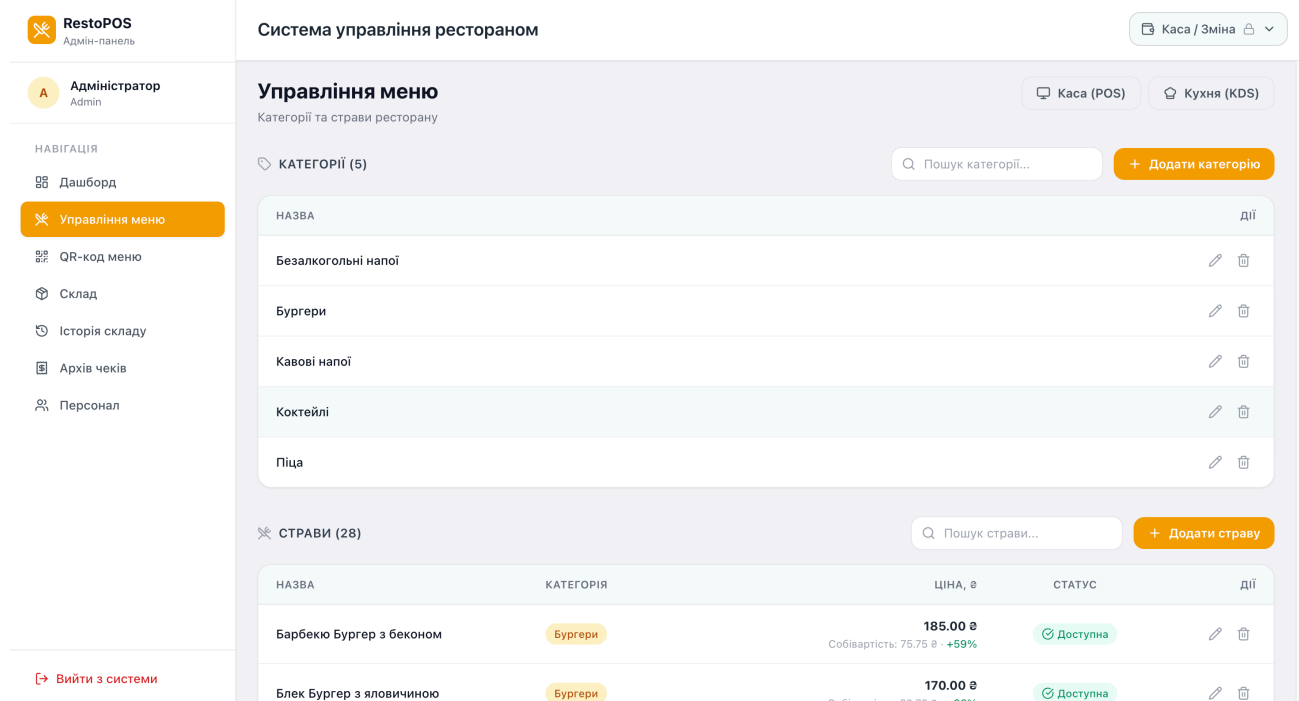


Рис.3.6. Інтерфейс керування категоріями та стравами меню

Окремим елементом адміністративної панелі є складський модуль. Він призначений для роботи з інгредієнтами, залишками, одиницями вимірювання, мінімальними залишками та складськими операціями. У цьому розділі користувач може додавати інгредієнти, переглядати поточні залишки, виконувати надходження, коригування або списання, а також переглядати історію руху складських позицій.

Інтерфейс складського обліку створений максимально зрозумілим, оскільки помилки в залишках можуть впливати на коректність рецептур і автоматичного списання інгредієнтів під час продажу страв. Для цього відображаємо кількість інгредієнтів, одиницю вимірювання, собівартість, мінімальний залишок, статус і доступні дії для кожної позиції (рис. 3.7).

RestoPOS Адмін-панель	Система управління рестораном					Каса / Зміна
Адміністратор Admin	Складські запаси					Каса (POS) Кухня (KDS)
НАВІГАЦІЯ	Інгредієнти та складські запаси ресторану					
Дашборд	37 інгредієнтів					Пошук інгредієнта...
Управління меню						Дивитись Архів Завантажити звіт (Excel) + Додати інгредієнт
QR-код меню						
Склад						
Історія складу						
Архів чеків						
Персонал						
Вийти з системи						
	ІНГРЕДІЄНТ	ЗАЛИШОК	МІНІМУМ	ОДИНИЦЯ	СОБІВАРТІСТЬ/ОД.	СТАН
	Ананас консервований	4.000	1.000	кг	160.00 ₴	Норма
	Бекон	3.760	0.500	кг	514.97 ₴	Норма
	Булочка для бургера	97.000	20.000	шт	12.00 ₴	Норма
	Булочка чорна для бургера	49.000	10.000	шт	15.00 ₴	Норма
	Білий ром	5.000	1.000	л	400.00 ₴	Норма
	Вершки 10%	10.000	2.000	л	80.00 ₴	Норма
	Вода очищена	99.670	20.000	л	2.00 ₴	Норма
	Горілка	9.950	2.000	л	300.00 ₴	Норма
	Джин	5.000	1.000	л	500.00 ₴	Норма

Рис. 3.7. Інтерфейс складського обліку інгредієнтів

Для зв’язку меню зі складом використовується інтерфейс рецептур. Він дозволяє задавати, які інгредієнти входять до складу конкретної страви та в якій кількості. Такий інтерфейс реалізований у вигляді форми редагування страви (рис. 3.8). Користувач обирає страву, додає до неї інгредієнти, вказує кількість і зберігає зміни. Надалі ці дані використовуються під час автоматичного списання інгредієнтів після продажу.

RestoPOS

Адмін-панель

Адміністратор Admin

НАВІГАЦІЯ

Дашборд

Управління меню

QR-код меню

Склад

Історія складу

Архів чеків

Персонал

Вийти з системи

Система управління рестораном

Каса / Зміна

Складські запаси

Інгредієнти та складські запаси ресторану

37 інгредієнтів

Пошук інгредієнта...

Дивитись Архів

Завантажити звіт (Excel)

Додати інгредієнт

ІНГРЕДІЄНТ	ЗАЛИШОК	МІНІМУМ	ОДИНИЦЯ	СОБІВАРТІСТЬ/ОД.	СТАН	ДІЇ
Ананас консервований	4.000	1.000	кг	160.00 ₴	Норма	
Бекон	3.760	0.500	кг	514.97 ₴	Норма	
Булочка для бургера	97.000	20.000	шт	12.00 ₴	Норма	
Булочка чорна для бургера	49.000	10.000	шт	15.00 ₴	Норма	
Білий ром	5.000	1.000	л	400.00 ₴	Норма	
Вершки 10%	10.000	2.000	л	80.00 ₴	Норма	
Вода очищена	99.670	20.000	л	2.00 ₴	Норма	
Горілка	9.950	2.000	л	300.00 ₴	Норма	
Джин	5.000	1.000	л	500.00 ₴	Норма	

Редагувати страву

Назва страви

Барбекю Бургер з беконом

Ціна, ₴

185

Категорія

Бургери

Страва доступна для замовлення

Технологічна карта (Склад страви)

Додати інгредієнт

ІНГРЕДІЄНТ	КІЛЬКІСТЬ	
Булочка для бургера (шт)	1	шт
Котлета яловича (напівфабрикат) (шт)	1	шт
Бекон (кг)	0,03	кг
Соус Барбекю (кг)	0,03	кг

Скасувати

Зберегти

Рис. 3.8. Інтерфейс налаштування рецептури страви

POS-інтерфейс призначений для швидкого створення замовлень працівником залу, касиром або офіціантом. Його структура має бути максимально простою, оскільки цей модуль використовується під час безпосереднього обслуговування клієнтів. Зазвичай POS-екран зручно поділити на дві основні частини: область вибору страв і область поточного замовлення.

В області вибору страв відображаються категорії меню та доступні позиції. Користувач обирає категорію, після чого бачить відповідні страви. Натискання на страву додає її до замовлення. В області поточного замовлення відображається список доданих позицій, їхня кількість, ціна, підсумкова сума та кнопки для підтвердження або очищення замовлення. Така структура дозволяє швидко формувати замовлення та зменшує кількість дій, необхідних для продажу (рис. 3.9).

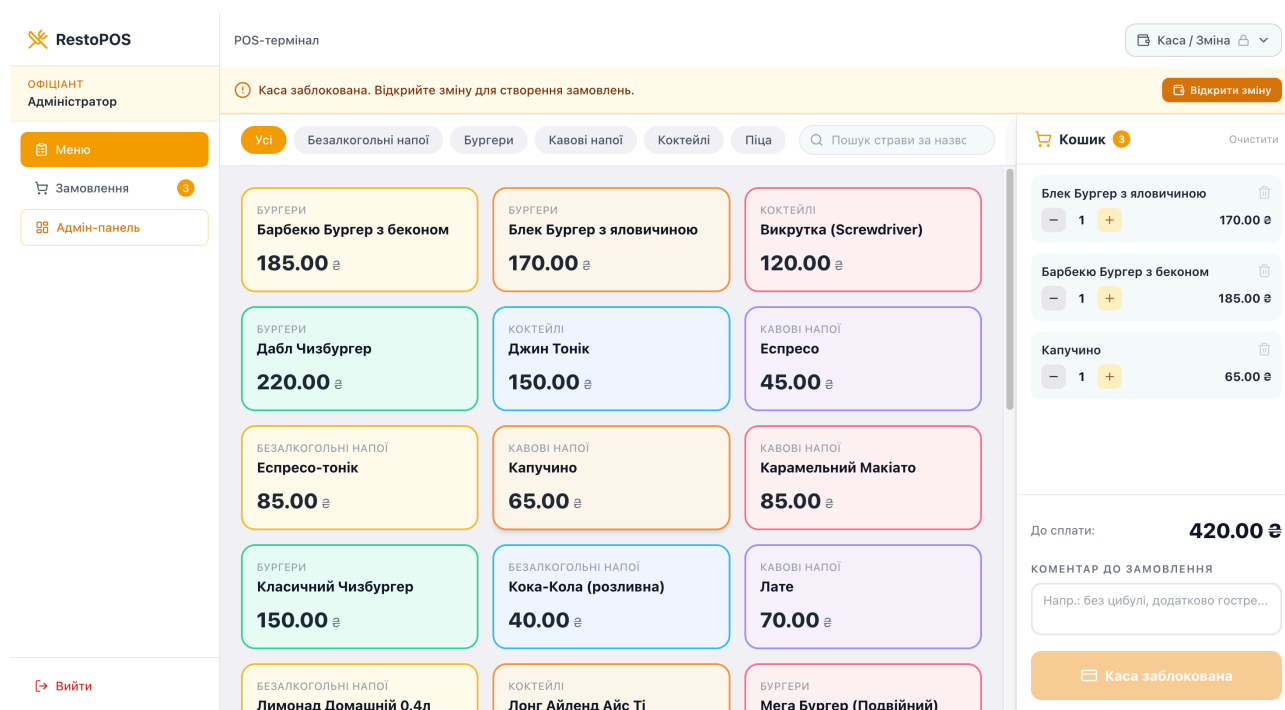


Рис. 3.9. Загальний вигляд POS-інтерфейсу створення замовлення

Після створення замовлення користувач переходить до підтвердження оплати. У моделі pre-рау замовлення передається на кухню після оплати. Тому інтерфейс POS повинен чітко показувати суму замовлення, список позицій, кількість кожної страви та результат виконання операції (рис. 3.10). Після успішного підтвердження замовлення система повідомляє користувача про його створення та відправлення на кухню.

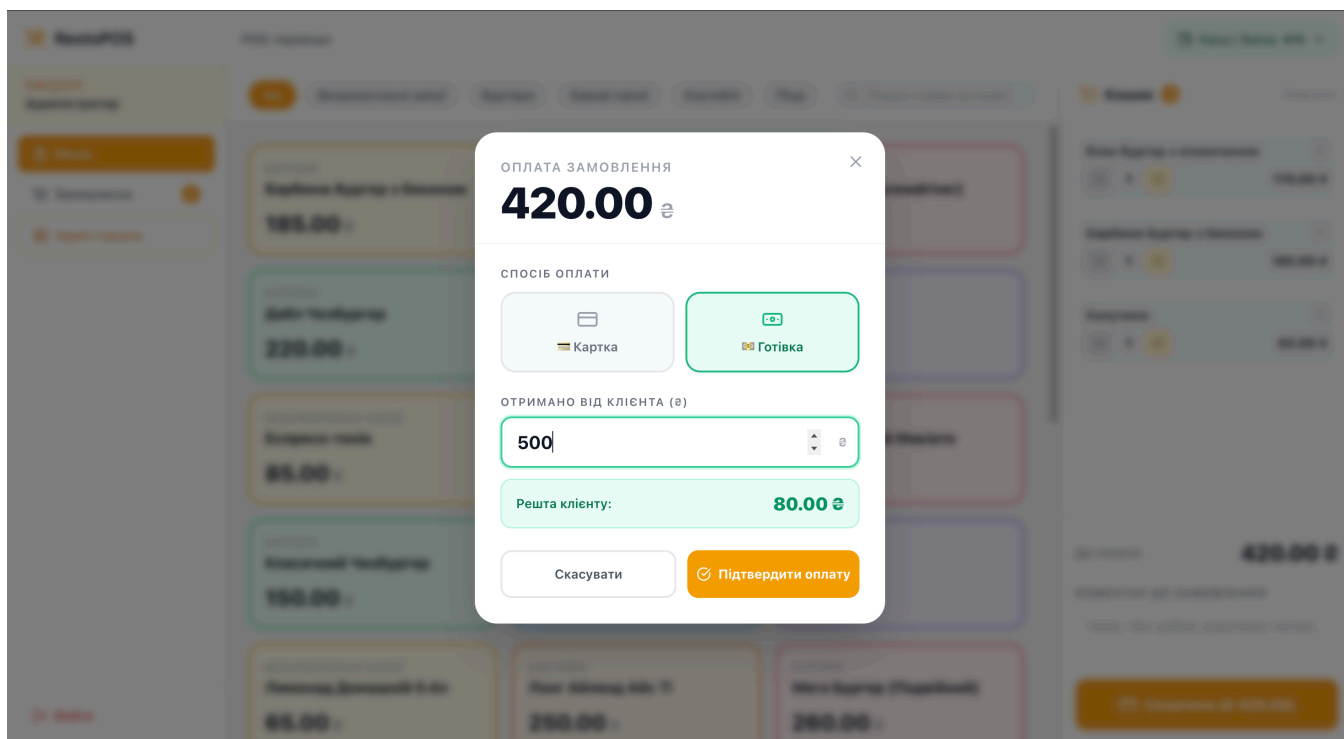


Рис. 3.10. Інтерфейс підтвердження та оплати замовлення

KDS-інтерфейс призначений для працівників кухні. Його головна функція – відображення активних замовлень, які необхідно приготувати. На відміну від адміністративної панелі, кухонний дисплей повинен мати мінімум зайвих елементів і бути добре читабельним на відстані. Замовлення відображено у вигляді карток, де зазначено номер замовлення, час створення, перелік страв, кількість позицій і поточний статус.

Працівник кухні повинен мати можливість швидко змінити статус замовлення, наприклад позначити його як готове або завершене. Оскільки KDS використовується в умовах кухні, кнопки мають бути великими, а інформація – структурованою та легкою для сприйняття. Оновлення замовлень у цьому модулі відбувається автоматично через механізм real-time повідомлень. Для KDS використовується SignalR, який дозволяє передавати оновлення замовлень підключеним клієнтам (рис. 3.11).

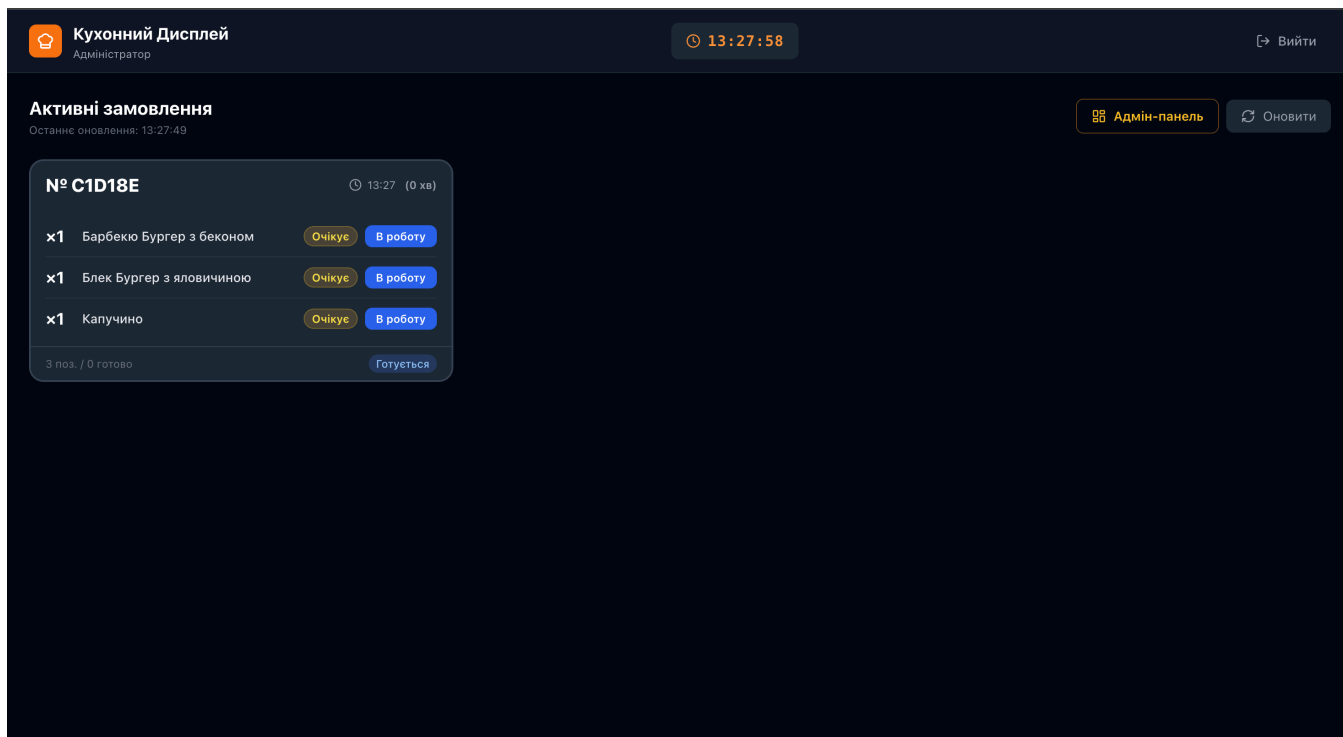


Рис. 3.11. Інтерфейс кухонного дисплея з активними замовленнями

Публічне QR-меню призначене для клієнтів закладу [30]. Воно відкривається без авторизації після сканування QR-коду. На цій сторінці клієнт може переглянути категорії, страви, описи, ціни та доступність позицій. Такий інтерфейс адаптований для мобільних пристроїв, оскільки більшість користувачів відкриватиме меню зі смартфона.

Публічне меню не містить адміністративних функцій або службової інформації. Його основне завдання – надати клієнту актуальний і зручний перелік страв (рис. 3.12). Якщо адміністратор змінює ціну, назву або доступність страви, ці зміни відображаються і в публічному меню.

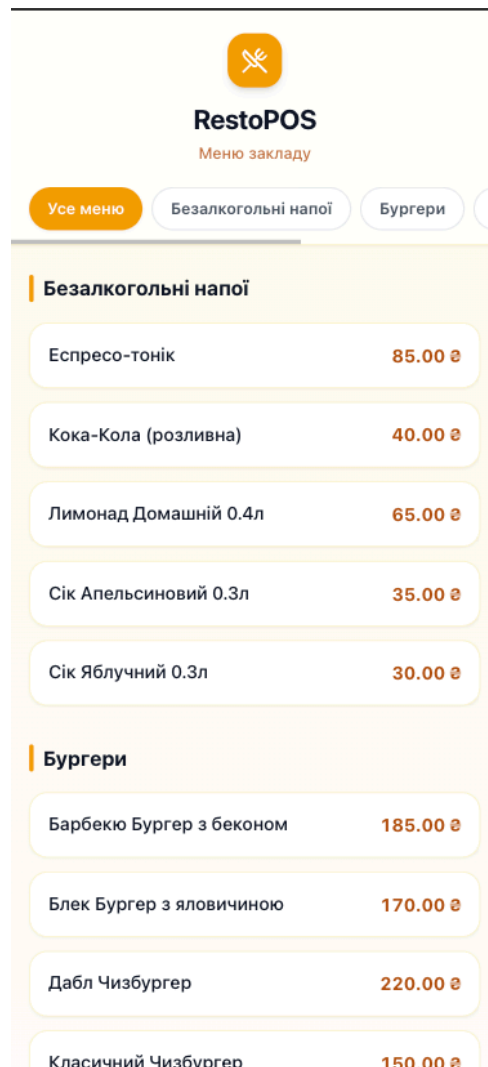


Рис. 3.12. Публічне QR-меню для клієнта

Для використання QR-меню в закладі передбачається інтерфейс генерації та друку QR-коду (рис. 3.13). Адміністратор або менеджер відкриває відповідний розділ, бачить QR-код із посиланням на публічне меню та може надрукувати його. QR-код формується у векторному форматі SVG, оскільки такий формат забезпечує якісний друк. У системі також передбачається ізольований механізм друку через окрему область, що дозволяє друкувати лише потрібний документ без зайвих елементів інтерфейсу. Для друку QR-коду, чеків і звітів використовується `qrcode.react`, `QRCodeSVG` і окреме `print-pipeline` через `#print-area`.

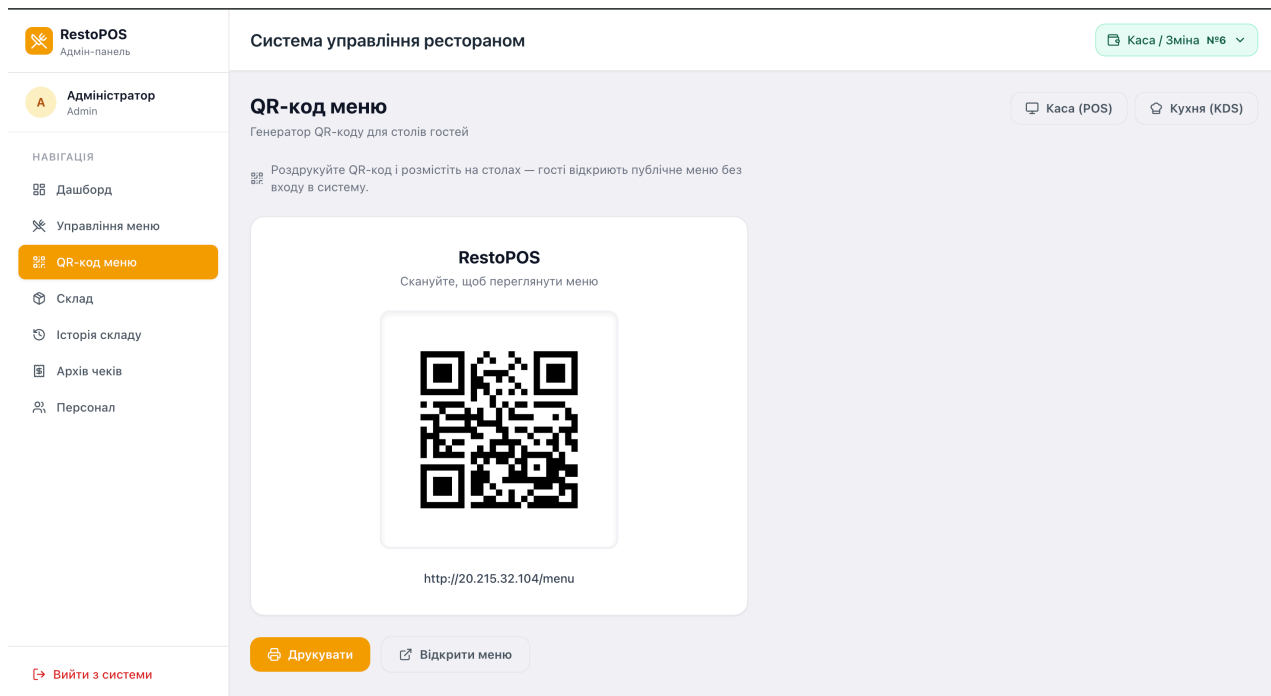


Рис. 3.13. Інтерфейс генерації та друку QR-коду меню

Окрім QR-коду, система підтримує друк чеків, refund-документів і X/Z-звітів касової зміни. Друковане представлення має відрізнятися від екранного інтерфейсу: воно містить тільки службову інформацію, необхідну для документа, без навігації та кнопок. Для цього використовується окрема друкована форма, яка передається в спеціальну область друку (рис. 3.14).

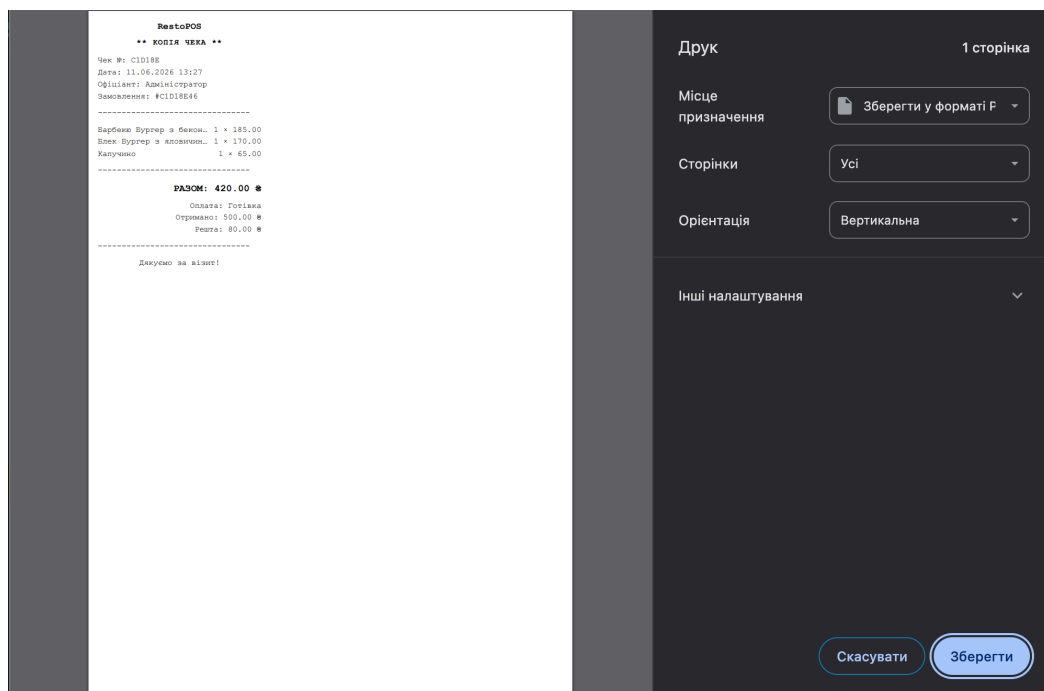


Рис. 3.14. Приклад друкованої форми чека

Важливою частиною інтерфейсу є повідомлення про результати операцій. Після створення замовлення, редагування страви, додавання інгредієнта або зміни статусу користувач отримує зрозуміле повідомлення про успішне виконання дії або про помилку. Повідомлення мають бути короткими, інформативними та не заважати основній роботі. Для підтвердження критичних дій, наприклад архівації або видалення запису, використовуються модальні вікна підтвердження.

Структуру основних інтерфейсних модулів та їх призначення наведено в таблиці 3.3.

Таблиця 3.3

Основні інтерфейсні модулі вебсистеми

Інтерфейсний модуль	Користувачі	Основне призначення
Сторінка авторизації	Працівники системи	Вхід за PIN-кодом і перенаправлення відповідно до ролі
Адміністративна панель	Адміністратор, менеджер	Керування меню, складом, працівниками, звітами та QR-меню
Модуль керування меню	Адміністратор, менеджер	Створення категорій, страв, редагування цін і доступності
Складський модуль	Адміністратор, менеджер	Облік інгредієнтів, залишків і складських операцій
Модуль рецептур	Адміністратор, менеджер	Налаштування складу страв і кількості інгредієнтів
POS-модуль	Офіціант, менеджер, адміністратор	Створення та оплата замовлень
KDS-модуль	Працівник кухні	Перегляд активних замовлень і зміна статусів
Публічне QR-меню	Клієнт	Перегляд меню без авторизації
Модуль друку	Працівники відповідних ролей	Друк QR-коду, чеків і звітів
Модуль повідомлень	Усі користувачі	Відображення результатів операцій і помилок

Логіка переходів між основними інтерфейсами залежить від ролі користувача. Після авторизації система визначає роль і відкриває відповідний розділ. Якщо користувач має декілька доступних зон, наприклад адміністратор або менеджер, в інтерфейсі можуть відображатися кнопки переходу між адміністративною панеллю, POS-модулем і KDS-модулем. Якщо користувач не має прав для перегляду певної сторінки, система перенаправляє його до

доступної функціональної зони або показати повідомлення про відсутність доступу.

З погляду зручності використання, інтерфейс відповідає кільком вимогам. По-перше, основні дії доступні з мінімальною кількістю кліків. Це особливо важливо для POS-модуля, де швидкість створення замовлення безпосередньо впливає на обслуговування клієнтів. По-друге, інтерфейс є візуально зрозумілим: кнопки мають очевидне призначення, таблиці – зрозумілі заголовки, а форми – підказки або валідаційні повідомлення. По-третє, система коректно відображається на різних типах пристроїв: ноутбуках, планшетах, POS-терміналах.

Окремо слід зазначити адаптивність публічного QR-меню. Оскільки клієнти переважно відкриватимуть його зі смартфонів, структура сторінки є зручною для вертикального перегляду. Категорії, картки страв, ціни й описи є читабельними на невеликому екрані. Водночас адміністративна панель і складський модуль зручніше використовувати на ширших екранах, оскільки вони містять таблиці та форми з великою кількістю інформації.

3.3. Тестування роботи програмного комплексу

У процесі розробки було використано автоматизовані засоби перевірки. Для backend-частини застосовано тестування за допомогою xUnit і WebApplicationFactory, що дозволяє перевіряти роботу ASP.NET Core Web API в умовах, наближених до реального запуску застосунку. Для frontend-частини виконувалася перевірка збірки React-застосунку та перевірка TypeScript-коду. Для тестування backend використовується Restaurant.Tests, а в CI/CD-процесі передбачено запуск dotnet test Restaurant.Tests і перевірку frontend-частини.

Окрему роль у тестуванні відіграє CI/CD-конвеєр GitLab. У проєкті реалізовано pipeline, який складається з трьох основних стадій: build, test і deploy. На стадії build виконується збірка backend-частини командою dotnet build і збірка frontend-частини командою npm run build. На стадії test запускаються backend-тести через dotnet test Restaurant.Tests, а також виконується перевірка TypeScript-коду. На стадії deploy, яка виконується для головної гілки, система

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						72

підключається до Azure VPS, отримує останню версію коду, повторно збирає контейнери та запускає їх через Docker Compose.

Автоматизована перевірка backend-частини дозволила переконатися, що серверний застосунок успішно компілюється, має коректні залежності та може запускатися в тестовому середовищі. Це особливо важливо для системи, у якій серверна частина відповідає за авторизацію, рольовий доступ, роботу із замовленнями, складські операції, рецептури, касові зміни та взаємодію з базою даних. Якщо під час збірки або тестування виникає помилка, CI/CD-конвеєр зупиняється, що дозволяє не допустити розгортання некоректної версії системи.

Для frontend-частини перевірялося, чи може React-застосунок бути успішно зібраний у production-режимі. Це дозволяє виявити помилки типізації, некоректні імпорти, відсутні залежності, помилки маршрутизації або проблеми з конфігурацією змінних середовища. Оскільки frontend взаємодіє з backend API через змінну VITE_API_URL, під час збірки важливо переконатися, що адреса API задається правильно. У production-середовищі frontend-застосунок після збірки обслуговується через Nginx, тому успішна frontend-збірка є обов'язковою умовою розгортання системи.

Контейнеризований запуск також був частиною перевірки програмного комплексу. Система запускається за допомогою Docker Compose, який оркеструє три сервіси: db, backend і frontend. Для бази даних використовується PostgreSQL-контейнер із healthcheck-перевіркою, backend залежить від успішного запуску бази даних, а frontend-контейнер містить зібраний React-застосунок і Nginx. Така структура дозволила перевірити, що всі компоненти системи можуть працювати разом в одному ізольованому середовищі.

Під час ручного функціонального тестування було перевірено основні сценарії роботи користувачів різних ролей. Насамперед тестувався вхід у систему за PIN-кодом. Було перевірено успішну авторизацію користувачів із різними ролями, обробку неправильного PIN-коду, перенаправлення після входу та доступність відповідних інтерфейсних модулів. Окремо перевірялося, що

користувач без необхідної ролі не може відкрити заборонену сторінку або виконати заборонену API-операцію.

Тестування POS-модуля охоплювало перевірку завантаження категорій і страв, додавання позицій до замовлення, зміну кількості, розрахунок підсумкової суми, підтвердження оплати та створення замовлення в базі даних. Також перевірялося, чи зберігається ціна страви в позиції замовлення на момент продажу та чи коректно формується склад замовлення. Це важливо для збереження історичної достовірності даних, оскільки ціни в меню можуть змінюватися після виконання продажу.

Окремо перевірялася взаємодія POS-модуля з кухонним дисплеєм. Після створення й оплати замовлення воно повинно з'явитися в KDS-інтерфейсі. Для цього у системі використовується SignalR, який забезпечує оновлення замовлень у реальному часі. Під час тестування було перевірено, що нове замовлення відображається на кухонному дисплеї без ручного оновлення сторінки, а зміна статусу замовлення коректно зберігається та відображається в інтерфейсі.

Тестування адміністративної панелі включало перевірку створення, редагування та архівації категорій, страв, інгредієнтів і користувачів. Було перевірено, що зміни в меню відображаються в POS-модулі та публічному QR-меню. Також перевірялося, що архівовані або недоступні позиції не відображаються серед активних елементів, але їхні історичні зв'язки не порушуються. Це важливо для складського обліку, оскільки фізичне видалення інгредієнта могло б призвести до втрати зв'язків із рецептурами або складськими транзакціями.

Складський модуль перевірявся через сценарії додавання інгредієнтів, зміни залишків, створення рецептур і автоматичного списання інгредієнтів після продажу страви. Під час тестування перевірялося, що система правильно визначає склад страви за рецептурою, обчислює необхідну кількість інгредієнтів, зменшує залишки та створює записи складських транзакцій. Окремо перевірялася ситуація недостатнього залишку, оскільки система повинна

запобігати некоректному списанню або повідомляти користувача про неможливість виконання операції.

Було також протестовано роботу серверної пагінації та фільтрації. У розділах із таблицями система повинна повертати не всі записи одразу, а лише необхідну сторінку даних. Це перевірялося для списків категорій, страв та інших довідників. Такий підхід дозволяє зменшити навантаження на клієнтську частину й забезпечити швидшу роботу інтерфейсу при збільшенні кількості записів. Пагінація реалізується на сервері через CountAsync, Skip, Take, AsNoTracking() і уніфіковану відповідь PagedResponse<T>.

Результати ручного тестування основних сценаріїв наведено в таблиці 3.4.

Таблиця 3.4

Результати функціонального тестування вебсистеми

№	Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус
1	Вхід користувача за правильним PIN-кодом	Користувач авторизується і переходить до доступного модуля	Авторизація виконана, роль визначена коректно	+
2	Вхід за неправильним PIN-кодом	Система відображає повідомлення про помилку	Повідомлення про помилку відображено	+
3	Перехід користувача до забороненого маршруту	Доступ блокується або виконується перенаправлення	Доступ обмежено відповідно до ролі	+
4	Створення категорії меню	Категорія зберігається в базі даних і відображається в інтерфейсі	Категорія створена та відображена	+
5	Додавання страви до меню	Страва зберігається, доступна в POS і QR-меню	Страва доступна у відповідних модулях	+
6	Створення замовлення через POS	Замовлення створюється з правильними позиціями та сумою	Замовлення створене коректно	+
7	Оплата замовлення	Статус замовлення змінюється, дані зберігаються	Статус оновлено	+
8	Передавання замовлення на KDS	Замовлення з'являється на кухонному дисплеї	Замовлення відображено без ручного оновлення	+
9	Зміна статусу замовлення на кухні	Новий статус зберігається і відображається в системі	Статус змінено коректно	+
10	Списання інгредієнтів за рецептурою	Залишки інгредієнтів зменшуються, створюються транзакції	Списання виконано	+

1.	2.	3.	4.	5.
11	Архівація інгредієнта	Інгредієнт приховується з активного списку, історія зберігається	Архівація працює коректно	+
12	Відкриття QR-меню без авторизації	Клієнт бачить публічне меню	Доступне за публічним маршрутом	+

Отже, тестування підтвердило працездатність основних модулів системи, коректну взаємодію frontend, backend і бази даних, а також стабільність процесу збирання, перевірки та розгортання через CI/CD.

Висновки до третього розділу

У третьому розділі було розглянуто порядок встановлення, налаштування, використання та тестування вебсистеми автоматизації роботи закладів харчування. Описано підготовку середовища, запуск системи за допомогою Docker Compose, налаштування змінних середовища, підключення до PostgreSQL, застосування міграцій і перевірку роботи контейнерів frontend, backend та db.

Було проаналізовано структуру інтерфейсу системи, яка охоплює сторінку авторизації, адміністративну панель, POS-модуль, KDS-модуль, публічне QR-меню та засоби друку документів. Інтерфейс побудовано з урахуванням рольового доступу, що дозволяє адміністраторам, менеджерам, працівникам POS-зони, кухні та клієнтам працювати лише з потрібними функціональними модулями. Особливу увагу приділено POS-інтерфейсу для створення й оплати замовлень та KDS-модулю, який забезпечує відображення активних замовлень у реальному часі.

Також було проведено тестування програмного комплексу. У межах CI/CD-конвеєра GitLab перевірено збирання frontend і backend, запуск backend-тестів та розгортання системи через Docker Compose.

Отже, результати третього розділу підтвердили практичну готовність системи до використання, її цілісну структуру, працездатність основних модулів і можливість відтворюваного розгортання в серверному середовищі.

ВИСНОВКИ

У результаті виконання роботи було спроектовано, реалізовано та протестовано вебсистему автоматизації роботи закладів харчування. Розроблений програмний комплекс призначений для оптимізації основних процесів закладу: керування меню, створення та обробки замовлень, передавання замовлень на кухню, контролю складських залишків, роботи з рецептурами, формування QR-меню та адміністрування користувачів. Отриманий результат відповідає поставленій меті, оскільки система забезпечує централізовану взаємодію між адміністративною панеллю, POS-модулем, кухонним дисплеєм, публічним меню та базою даних.

У роботі було проведено аналіз предметної області та розглянуто існуючі системи автоматизації закладів харчування, зокрема Poster POS, Skyservice POS і SERVIO POS. На основі аналізу було визначено основні вимоги до майбутньої системи та обґрунтовано доцільність розробки власного програмного комплексу, орієнтованого на гнучке налаштування бізнес-логіки, підтримку ключових процесів закладу та можливість подальшого розширення.

Під час проєктування було визначено основні ролі користувачів: адміністратор, менеджер, офіціант або працівник POS-зони, працівник кухні та клієнт. Для них було сформовано функціональні сценарії роботи та визначено структуру системи. Було спроектовано базу даних на основі PostgreSQL, яка зберігає інформацію про користувачів, категорії, страви, інгредієнти, рецептури, замовлення, позиції замовлень, складські операції та касові зміни. Також було описано алгоритми авторизації, створення й оплати замовлення, передавання замовлення на KDS, зміни статусів, списання інгредієнтів і друку службових документів.

У процесі реалізації було створено клієнтську частину на основі React, TypeScript і Vite та серверну частину на основі ASP.NET Core Web API. Для роботи з базою даних використано Entity Framework Core, для оновлення замовлень у реальному часі – SignalR, а для розгортання – Docker Compose. У системі реалізовано авторизацію за PIN-кодом, рольовий доступ,

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						77

адміністративну панель, POS-інтерфейс, KDS-модуль, публічне QR-меню, облік складу, рецептури, автоматичне списання інгредієнтів, друк QR-коду, чеків і звітів.

Тестування підтвердило працездатність основних модулів програмного комплексу. Було перевірено авторизацію, рольовий доступ, створення замовлень, роботу кухонного дисплея, керування меню, складський облік, QR-меню, друк документів і контейнеризоване розгортання. Також реалізовано CI/CD-процес для автоматизованої збірки, тестування та розгортання системи.

Практична цінність роботи полягає в можливості використання розробленої системи як основи для автоматизації невеликого або середнього закладу харчування. Подальше вдосконалення може бути спрямоване на інтеграцію з платіжними терміналами та ПРРО, реалізацію модуля доставки, програм лояльності, підтримку кількох торгових точок, розширення аналітики та покращення механізмів безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sommerville I. Software Engineering. Boston : Pearson Education Limited, 2016. 816 p.
2. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.
3. What Is POS?. Oracle. Industries. URL: <https://www.oracle.com/ua/food-beverage/what-is-pos/> (дата звернення: 7.03.2026).
4. Entity Framework Core documentation. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/ef/core/> (дата звернення: 20.03.2026).
5. ISO/IEC 25010:2023. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality model. Geneva : International Organization for Standardization, 2023.
6. Poster POS – програма автоматизації закладів громадського харчування на планшеті. URL: <https://joinposter.com/ua> (дата звернення: 04.04.2026).
7. Інтеграція Poster QR з системою автоматизації Poster POS. URL: <https://joinposter.com/ua/applications/poster-qr> (дата звернення: 15.04.2026).
8. Poster Kitchen Kit. Google Play. URL: https://play.google.com/store/apps/details?id=com.poster_kitchen.app&hl=uk (дата звернення: 16.04.2026).
9. Хмарна каса Skyservice POS – автоматизація бізнесу. URL: <https://skyservice.pro/uk/main/> (дата звернення: 27.04.2026).
10. Повний контроль бізнесу з будь-якої точки світу. Skyservice POS. URL: <https://skyservice.pro/uk/about/> (дата звернення: 28.04.2026).
11. Програма для обліку товару. Skyservice POS. URL: <https://skyservice.pro/uk/solutions/warehouse-accounting/> (дата звернення: 29.04.2026).
12. Фіскалізація та програмний РРО. Skyservice POS. URL: <https://skyservice.pro/uk/solutions/fiscalization/> (дата звернення: 29.04.2026).

13. Автоматизація ресторанів. SERVIO Software. URL: <https://serviosoft.com/uk/rishennya/hotel-automation-copy> (дата звернення: 03.05.2026).
14. SERVIO POS – функціональна касова система для автоматизації ресторанного бізнесу. Expert Solution. URL: <https://expertsolution.com.ua/produkty/pz/servio-pos-dlya-avtomatyzatsiyi-restorannogo-biznesu/> (дата звернення: 03.05.2026).
15. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 616 p.
16. Richardson L., Ruby S. RESTful Web Services. Sebastopol : O'Reilly Media, 2007. 448 p.
17. Microsoft. ASP.NET Core SignalR documentation. URL: <https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction> (дата звернення: 10.04.2026).
18. Docker Documentation. Docker Compose overview. URL: <https://docs.docker.com/compose/> (дата звернення: 25.05.2026).
19. React Documentation. Describing the UI. URL: <https://react.dev/learn/describing-the-ui> (дата звернення: 12.04.2026).
20. Vite Documentation. Why Vite. URL: <https://vite.dev/guide/why> (дата звернення: 13.04.2026).
21. Tailwind CSS Documentation. Core concepts. URL: <https://tailwindcss.com/docs/styling-with-utility-classes> (дата звернення: 29.04.2026).
22. Microsoft. ASP.NET Core documentation. URL: <https://learn.microsoft.com/en-us/aspnet/core/> (дата звернення: 11.05.2026).
23. PostgreSQL. The world's most advanced open source database. URL: <https://www.postgresql.org/> (дата звернення: 11.05.2026).
24. Nginx Documentation. Beginner's Guide. URL: https://nginx.org/en/docs/beginners_guide.html (дата звернення: 26.05.2026).

25. GitLab Docs. GitLab CI/CD. URL: <https://docs.gitlab.com/ci/> (дата звернення: 24.05.2026).
26. Microsoft Azure Documentation. Azure virtual machines documentation. URL: <https://learn.microsoft.com/en-us/azure/virtual-machines/> (дата звернення: 27.05.2026).
27. Object Management Group. OMG Unified Modeling Language Specification. Version 2.5.1. URL: <https://www.omg.org/spec/UML/2.5.1/> (дата звернення: 01.06.2026).
28. Microsoft. Npgsql Entity Framework Core Provider. URL: <https://learn.microsoft.com/en-us/ef/core/providers/npgsql/> (дата звернення: 06.05.2026).
29. pgAdmin 4 Documentation. ERD Tool. URL: https://www.pgadmin.org/docs/pgadmin4/latest/erd_tool.html (дата звернення: 20.05.2026).
30. Resaturant Management System. Menu. URL: <http://20.215.32.104/menu> (дата звернення: 01.06.2026).

ДОДАТКИ

					ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
						82

Реєстрація сервісів у контейнері залежностей ASP.NET Core

```

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseNpgsql(builder.Configuration.GetConnectionString("DefaultConnection")));

var jwtSecret = builder.Configuration["Jwt:Secret"]
?? throw new InvalidOperationException(
    "Відсутній параметр 'Jwt:Secret' у конфігурації.");

builder.Services
    .AddAuthentication(options =>
    {
        options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = builder.Configuration["Jwt:Issuer"],
            ValidAudience = builder.Configuration["Jwt:Audience"],
            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSecret)),
            ClockSkew = TimeSpan.Zero,
        };
    });

builder.Services.AddAuthorization();
builder.Services.AddHealthChecks();

builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowFrontend", policy =>
    {
        policy
            .AllowAnyOrigin()
            .AllowAnyMethod()
            .AllowAnyHeader();
    });
});

```

```

    });
});

builder.Services.AddScoped<IAuthService, AuthService>();
builder.Services.AddScoped<IMenuService, MenuService>();
builder.Services.AddScoped<IInventoryService, InventoryService>();
builder.Services.AddScoped<IOrderService, OrderService>();
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<IDashboardService, DashboardService>();
builder.Services.AddScoped<IShiftService, ShiftService>();

builder.Services.AddControllers()
    .AddJsonOptions(options =>
        options.JsonSerializerOptions.Converters.Add(
            new System.Text.Json.Serialization.JsonStringEnumConverter()));

builder.Services.AddSignalR()
    .AddJsonProtocol(options =>
        options.PayloadSerializerOptions.Converters.Add(
            new System.Text.Json.Serialization.JsonStringEnumConverter()));
builder.Services.AddSwaggerGen(options =>
{
    options.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "Restaurant Management API",
        Version = "v1",
        Description = "REST API для POS-системи управління рестораном."
    });

    options.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
    {
        Description = "JWT авторизація. Введіть токен у форматі: Bearer {ваш_токен}",
        Name = "Authorization",
        In = ParameterLocation.Header,
        Type = SecuritySchemeType.Http,
        Scheme = "bearer",
        BearerFormat = "JWT"
    });

    options.AddSecurityRequirement(new OpenApiSecurityRequirement
    {
        {
            new OpenApiSecurityScheme
            {
                Reference = new OpenApiReference

```

```

        {
            Type = ReferenceType.SecurityScheme,
            Id = "Bearer"
        }
    },
    Array.Empty<string>()
}
});
});

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(options =>
    {
        options.SwaggerEndpoint("/swagger/v1/swagger.json", "Restaurant Management API v1");
        options.DocumentTitle = "Restaurant POS API";
    });
}

if (!app.Environment.IsDevelopment())
{
    app.UseHttpsRedirection();
}

app.UseCors("AllowFrontend");
app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();
app.MapHub<OrderHub>("/api/hubs/orders");
app.MapHealthChecks("/api/health");

app.Run();

```

Генерація JWT-токена та авторизація користувача за PIN-кодом

```

public async Task<AuthResultDto> AuthenticateAsync(string pinCode, CancellationToken ct =
default)
{
    if (string.IsNullOrEmpty(pinCode))
        throw new ArgumentException("PIN-код не може бути порожнім.", nameof(pinCode));

    var users = await _db.Users
        .AsNoTracking()
        .ToListAsync(ct);

    User? authenticatedUser = null;

    foreach (var user in users)
    {
        if (BCrypt.Net.BCrypt.Verify(pinCode, user.PinCodeHash))
        {
            authenticatedUser = user;
            break;
        }
    }

    if (authenticatedUser is null)
        throw new UnauthorizedAccessException(
            "Невірний PIN-код. Перевірте введені дані та спробуйте ще раз.");

    var token = GenerateJwtToken(authenticatedUser);

    return new AuthResultDto(
        token,
        authenticatedUser.Id,
        authenticatedUser.FullName,
        authenticatedUser.Role);
}

private string GenerateJwtToken(User user)
{
    {
        var secret = _config["Jwt:Secret"]
            ?? throw new InvalidOperationException(
                "Секретний ключ JWT не знайдено у конфігурації.");

        if (secret.Length < 32)
            throw new InvalidOperationException(
                "Секретний ключ JWT занадто короткий. Мінімальна довжина – 32 символи для
                HMAC-SHA256.");

        var signingKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secret));
        var credentials = new SigningCredentials(signingKey, SecurityAlgorithms.HmacSha256);
    }
}

```

```

var claims = new[]
{
    new Claim(JwtRegisteredClaimNames.Sub, user.Id.ToString()),
    new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
    new Claim(JwtRegisteredClaimNames.Iat,
        DateTimeOffset.UtcNow.ToUnixTimeSeconds().ToString(),
        ClaimValueTypes.Integer64),
    new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
    new Claim(ClaimTypes.Name, user.FullName),
    new Claim(ClaimTypes.Role, user.Role.ToString()),
};

var expiryHours = int.TryParse(_config["Jwt:ExpiryHours"], out var h) && h > 0 ? h : 8;

var token = new JwtSecurityToken(
    issuer: _config["Jwt:Issuer"],
    audience: _config["Jwt:Audience"],
    claims: claims,
    notBefore: DateTime.UtcNow,
    expires: DateTime.UtcNow.AddHours(expiryHours),
    signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);
}

```

Реалізація ролівої маршрутизації та захищених маршрутів у React

App.tsx – конфігурація React Router:

```

const App = () => (
  <BrowserRouter>
    <PrintAreaHost />
    <Routes>
      <Route path="/login" element={<LoginPage />} />
      <Route path="/menu" element={<PublicMenuPage />} />

      <Route
        path="/pos/*"
        element={
          <ProtectedRoute allowedRoles={['Waiter', 'Admin', 'Manager']}>
            <PosLayout />
          </ProtectedRoute>
        }
      />
      <Route index element={<Navigate to="orders" replace />} />
      <Route path="orders" element={<PosPage />} />
      <Route path="menu" element={<PosPage />} />
    </Route>

    <Route
      path="/kds/*"
      element={
        <ProtectedRoute allowedRoles={['Kitchen', 'Admin', 'Manager']}>
          <KdsLayout />
        </ProtectedRoute>
      }
    />
    <Route index element={<KdsPage />} />
  </Route>

  <Route
    path="/admin/*"

    element={
      <ProtectedRoute allowedRoles={['Admin', 'Manager']}>

        <AdminLayout />
      </ProtectedRoute>
    }
  />
  <Route index element={<Navigate to="menu" replace />} />
  <Route path="dashboard" element={<AdminPage />} />
  <Route path="menu" element={<AdminPage />} />
</Route>

  <Route path="/" element={<Navigate to="/login" replace />} />
  <Route path="*" element={<Navigate to="/login" replace />} />

```

```

    </Routes>
  </BrowserRouter>
);

```

ProtectedRoute.tsx – RBAC-захист маршрутів:

```

const ProtectedRoute = ({ children, allowedRoles }: ProtectedRouteProps) => {
  const user = useAuthStore((state) => state.user);
  const location = useLocation();

  if (!user) {
    return <Navigate to="/login" state={{ from: location }} replace />;
  }

  const hasAccess =
    !allowedRoles || allowedRoles.length === 0 || allowedRoles.includes(user.role);

  if (!hasAccess) {
    const homePath = getHomeRoute(user);

    if (location.pathname.startsWith(homePath)) {
      return <AccessDenied homePath={homePath} userRole={user.role} />;
    }

    return <Navigate to={homePath} replace />;
  }

  return <>{children}</>;
};

```

Транзакційна обробка оплати замовлення, списання інгредієнтів і надсилання оновлення до KDS

```

public async Task<Order> PayAndSendToKitchenAsync(
    Guid orderId,
    PaymentMethod method,
    decimal? tendered,
    CancellationToken ct = default)
{
    await using var transaction = await _db.Database.BeginTransactionAsync(ct);

    Order order;
    try
    {
        order = await _db.Orders
            .Include(o => o.OrderItems)
            .ThenInclude(oi => oi.Dish)
            .ThenInclude(d => d.RecipeItems)
            .ThenInclude(ri => ri.Ingredient)
            .FirstOrDefaultAsync(o => o.Id == orderId, ct)
            ?? throw new NotFoundException(nameof(Order), orderId);

        // Агрегація потреб по інгредієнтах
        var ingredientRequirements = order.OrderItems
            .SelectMany(oi => oi.Dish.RecipeItems.Select(ri => new
            {
                ri.Ingredient,
                RequiredForItem = ri.RequiredAmount * oi.Quantity
            }))
            .GroupBy(x => x.Ingredient.Id)
            .Select(g => new
            {
                Ingredient = g.First().Ingredient,
                TotalRequired = g.Sum(x => x.RequiredForItem)
            })
            .ToList();

        foreach (var req in ingredientRequirements)
        {
            if (req.Ingredient.StockQuantity < req.TotalRequired)
                throw new InsufficientStockException(
                    req.Ingredient.Name,
                    req.TotalRequired,
                    req.Ingredient.StockQuantity,
                    req.Ingredient.Unit);
        }
        foreach (var req in ingredientRequirements)
        {
            req.Ingredient.ReduceStock(req.TotalRequired);
        }
        order.ProcessPayment(method, tendered);
    }
}

```

					ІПЗ.КР.Б – 121 – 26 – ІПЗ	Арк.
						90

```

        await _db.SaveChangesAsync(ct);
        await transaction.CommitAsync(ct);
    }
    catch
    {
        await transaction.RollbackAsync(ct);
        throw;
    }

    var dto = OrderDetailDto.FromEntity(order);
    await _hubContext.Clients.All.SendAsync("ReceiveNewOrder", dto, ct);

    return order;
}

```

Реалізація м'якого видалення інгредієнтів і фільтрації активних складських позицій

Ingredient.cs – властивість **IsArchived** та доменний метод архівування:

```
public bool IsArchived { get; set; } = false;
// ... інші властивості та методи ...
public void Archive() => IsArchived = true;
public void Restore() => IsArchived = false;
```

InventoryService.cs – м'яке видалення:

```
public async Task DeleteIngredientAsync(Guid id, CancellationToken ct = default)
{
    var ingredient = await _db.Ingredients
        .FirstOrDefaultAsync(i => i.Id == id && !i.IsArchived, ct)
        ?? throw new NotFoundException(nameof(Ingredient), id);

    ingredient.Archive();
    await _db.SaveChangesAsync(ct);
}
```

Реалізація серверної пагінації та фільтрації даних**PagedResponse.cs:**

```
public sealed record PagedResponse<T>(
    IEnumerable<T> Items,
    int TotalCount,
    int TotalPages,
    int CurrentPage)
{
    public static PagedResponse<T> Create(
        IReadOnlyList<T> items,
        int totalCount,
        int page,
        int pageSize) =>
        new(
            items,
            totalCount,
            totalCount == 0 ? 0 : (int)Math.Ceiling(totalCount / (double)pageSize),
            page);
}
```

MenuService.cs – серверна пагінація та фільтрація страв:

```
public async Task<PagedResponse<Dish>> GetDishesPagedAsync(
    bool onlyAvailable,
    string? searchTerm,
    int page,

    int pageSize,
    CancellationToken ct = default)
{
    page = Math.Max(1, page);
    pageSize = Math.Clamp(pageSize, 1, 500);

    var query = _db.Dishes
        .AsNoTracking()
        .Include(d => d.Category)
        .Include(d => d.RecipeItems)
        .ThenInclude(ri => ri.Ingredient)
        .AsQueryable();

    if (onlyAvailable)
        query = query.Where(d => d.IsAvailable);

    if (!string.IsNullOrEmpty(searchTerm))
    {
        var pattern = $"%{searchTerm.Trim()}%";
        query = query.Where(d => EF.Functions.ILike(d.Name, pattern));
    }

    var totalCount = await query.CountAsync(ct);
```

```

var items = await query
    .OrderBy(d => d.Name)
    .Skip((page - 1) * pageSize)
    .Take(pageSize)
    .ToListAsync(ct);

return PagedResponse<Dish>.Create(items, totalCount, page, pageSize);
}

```

Налаштування Axios-клієнта з JWT-токеном та обробкою помилки авторизації

```

export const api = axios.create({
  baseURL: BASE_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});
api.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => Promise.reject(error),
);
api.interceptors.response.use(
  (response) => response,
  (error) => {
    const requestUrl = error.config?.url ?? '';
    if (
      error.response?.status === 401 &&
      !requestUrl.includes('/auth/login') &&
      !requestUrl.includes('/public/menu')
    ) {
      useAuthStore.getState().logout();
      window.location.replace('/login');
    }
    return Promise.reject(error);
  },
);

```

Генерація QR-коду меню та друк через ізольовану область #print-area

QrCodeGeneratorTab.tsx – генерація векторного QR-коду (SVG):

```
import { QRCodeSVG } from 'qrcode.react';
import { printReactContent } from '../utils/printReceipt';
import QrCodePrintSheet from './QrCodePrintSheet';

const QrCodeGeneratorTab = () => {
  const menuUrl = useMemo(() => `${window.location.origin}/menu`, []);

  const handlePrint = async () => {
    await printReactContent(<QrCodePrintSheet menuUrl={menuUrl} />);
  };

  return (
    // ...
    <QRCodeSVG
      value={menuUrl}
      size={240}
      level="M"
      includeMargin
    />
    // ...
  );
};
```

printReceipt.ts – рендеринг у #print-area та виклик друку:

```
export async function printReactContent(
  content: ReactNode,

  delayMs = 150,
): Promise<void> {
  const root = getPrintRoot();

  flushSync(() => {
    root.render(content);
  });

  await new Promise<void>((resolve) => {
    requestAnimationFrame(() => {
      requestAnimationFrame(() => resolve());
    });
  });

  await new Promise<void>((resolve) => {
    setTimeout(() => {
      window.print();
      resolve();
    }, delayMs);
  });
};
```

```
const cleanup = () => {
  clearPrintArea();
  window.removeEventListener('afterprint', cleanup);
};
window.addEventListener('afterprint', cleanup);
setTimeout(cleanup, 3_000);
}
```

index.css – ізоляція області друку:

```
@media print {
  @page {
    margin: 0;
  }
  body > *:not(#print-area) {
    display: none !important;
  }
  #print-area {
    display: block !important;
    position: absolute !important;
    left: 0 !important;
    top: 0 !important;
    width: 100% !important;
    height: auto !important;
    overflow: visible !important;
    opacity: 1 !important;
    visibility: visible !important;
    margin: 0 !important;
    padding: 0 !important;
    background: #fff !important;
  }
  #print-area,
  #print-area * {
    visibility: visible !important;
    color: #000 !important;
  }
  #print-area .qr-print-sheet {

    display: flex !important;
    flex-direction: column !important;
    align-items: center !important;
    justify-content: center !important;
    min-height: 100vh !important;
    background: #fff !important;
    text-align: center !important;
  }

  #print-area .qr-print-sheet__code svg {
    display: block !important;
    width: 280px !important;
    height: 280px !important;
  }
}
```

Налаштування Docker Compose для запуску frontend, backend і PostgreSQL, а також, налаштування GitLab CI/CD для збирання, тестування та розгортання системи

docker-compose.yml:

services:

db:

image: postgres:15-alpine

container_name: restaurant_db

environment:

POSTGRES_USER: postgres

POSTGRES_PASSWORD: rootpassword

POSTGRES_DB: RestaurantDb

ports:

- "5433:5432"

volumes:

- postgres_data:/var/lib/postgresql/data

healthcheck:

test: ["CMD-SHELL", "pg_isready -U postgres -d RestaurantDb"]

interval: 5s

timeout: 5s

retries: 10

backend:

build:

context: ./backend/Restaurant.API

dockerfile: Dockerfile

container_name: restaurant_backend

environment:

- ASPNETCORE_ENVIRONMENT=Development

- ASPNETCORE_URLS=http://+:8080

-

ConnectionString__DefaultConnection=Host=db;Port=5432;Database=RestaurantDb;Username=postgres;Password=rootpassword

ports:

- "5027:8080"

depends_on:

db:

condition: service_healthy

frontend:

build:

context: ./frontend

dockerfile: Dockerfile

args:

VITE_API_URL: \${VITE_API_URL:-http://20.215.32.104:5027}

container_name: restaurant_frontend

ports:

- "80:80"

depends_on:

- backend

					ІПЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
						98

volumes:
 postgres_data:

.gitlab-ci.yml

stages:
 - build
 - test
 - deploy

variables:
 DOTNET_CLI_TELEMETRY_OPTOUT: "1"

=== BACKEND ===

build-backend:
 stage: build
 image: mcr.microsoft.com/dotnet/sdk:10.0
 script:
 - echo "Building Backend..."
 - cd backend/Restaurant.API
 - dotnet restore
 - dotnet build --no-restore -c Release

test-backend:
 stage: test
 image: mcr.microsoft.com/dotnet/sdk:10.0
 needs: ["build-backend"]
 script:
 - echo "Running Backend Tests (if any)..."
 - dotnet test backend/Restaurant.Tests/Restaurant.Tests.csproj -c Release

=== FRONTEND ===

build-frontend:
 stage: build
 image: node:20-alpine
 variables:

 VITE_API_URL: "http://20.215.32.104:5027"
 script:
 - echo "Building Frontend..."
 - cd frontend
 - npm ci
 - npm run build

lint-frontend:
 stage: test
 image: node:20-alpine
 needs: ["build-frontend"]
 script:
 - echo "Checking TypeScript types..."
 - cd frontend

- npm ci
- npm run tsc || echo "TypeScript has warnings, but proceeding..."

=== DEPLOY ===

deploy_to_azure:

stage: deploy

image: alpine:latest

only:

- main

before_script:

- apk add --no-cache openssh-client sshpass
- mkdir -p ~/.ssh
- ssh-keyscan -H \$SERVER_IP >> ~/.ssh/known_hosts

script:

- sshpass -p "\$SERVER_PASSWORD" ssh \$SERVER_USER@\$SERVER_IP "cd restaurant-management-system && git pull origin main && docker compose build --no-cache && docker compose up -d"

					ИП3.КР.Б – 121 – 26 – П3	Арк.
						100