

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 «Веб-технології»)

на тему:

«Вебсистема управління рестораном»

Виконав студент групи ВТ-22-2
МОТИЦЬКИЙ Нікіта Вячеславович

Керівник роботи:
ФАНТ Микола Олександрович

Рецензент:
СУГОНЯК Інна Іванівна

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

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **МОТИЦЬКИЙ Нікіта Вячеславович**

Керівник роботи: **ФАНТ Микола Олександрович**

Тема роботи: **«Вебсистема управління рестораном»,**

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: **використання веборієнтованих технологій для розробки системи управління рестораном для автоматизації процесів управління діяльністю ресторану.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Фант М. О.	02.03.2026р.	02.03.2026р
2	Фант М. О.	06.04.2026р	06.04.2026р
3	Фант М. О.	04.05.2026р	04.05.2026р

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебсистеми управління рестораном та пояснювальної записки. Пояснювальна записка містить 98 сторінки, з них 78 сторінки основного тексту, 46 ілюстрацій, 31 таблицю, 1 додаток та 32 літературних джерел.

Метою роботи є розробка алгоритмів роботи та реалізація програмного комплексу вебсистеми управління рестораном, що автоматизує процеси управління меню, замовленнями, персоналом та фінансовим обліком закладу громадського харчування.

У процесі виконання роботи застосовувались методи системного аналізу для дослідження предметної області, методи об'єктно-орієнтованого проектування для розробки архітектури системи та методи моделювання даних для проектування структури бази даних. Реалізацію виконано з використанням клієнт-серверної архітектури на основі GraphQL API, серверного фреймворку NestJS, бібліотеки React.js для клієнтської частини та документоорієнтованої бази даних MongoDB.

У результаті виконання роботи розроблено вебсистему управління рестораном із розмежуванням доступу за ролями (адміністратор, офіціант, кухар), реалізовано модулі управління меню, замовленнями, персоналом та фінансового обліку. Система може бути впроваджена в реальних умовах роботи закладів громадського харчування та масштабована відповідно до потреб бізнесу.

КЛЮЧОВІ СЛОВА: GRAPHQL, NESTJS, REACT, РЕСТОРАН, ВЕБСИСТЕМА.

					ІПЗ.КР.Б–121–26–ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Вебсистема управління рестораном Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		Мотицький Н. В.						
Керівник		Фант М. О.					4	78
						Житомирська політехніка, група. ВТ-22-2		
Рецензент		Сугоняк І. І.						
Зав. каф.		Вакалюк Т. А.						

ABSTRACT

The bachelor's final qualification work consists of a restaurant management web system and an explanatory note. The explanatory note contains 98 pages, of which 78 pages are the main text, 46 illustrations, 31 tables, 1 appendices and 32 literary sources.

The purpose of the work is to develop algorithms and implement a software complex for a restaurant management web system that automates the processes of menu management, orders, personnel and financial accounting of a catering establishment.

In the process of performing the work, systems analysis methods were used to study the subject area, object-oriented design methods to develop the system architecture and data modeling methods to design the database structure. The implementation was carried out using a client-server architecture based on the GraphQL API, the NestJS server framework, the React.js library for the client part and the MongoDB document-oriented database.

As a result of the work, a restaurant management web system was developed with access separation by roles (administrator, waiter, chef), and modules for menu management, orders, personnel, and financial accounting were implemented. The system can be implemented in real-world conditions of public catering establishments and scaled according to business needs.

KEYWORDS: GRAPHQL, NESTJS, REACT, RESTAURANT, WEB SYSTEM.

					ІПЗ.КР.Б–121–26–ІІЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Вебсистема управління рестораном	Літ.	Арк.	Аркушів
Розроб.		Мотицький Н. В.						
Керівник		Фант М. О.					4	78
						Житомирська політехніка, група. ВТ-22-2		
Рецензент		Сугоняк І. І.						
Зав. каф.		Вакалюк Т. А.			Пояснювальна записка			

ЗМІСТ

РЕФЕРАТ.....	3
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ УПРАВЛІННЯ РЕСТОРАНОМ	9
1.1 Аналіз аналогів програмного продукту	9
1.2 Постановка задачі.....	17
1.3 Вибір архітектури вебсистеми.....	21
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	26
Висновки до першого розділу	31
РОЗДІЛ 2. ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ.....	32
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи	32
2.2 Розробка бази даних системи.....	40
2.3 Проектування та реалізація алгоритмів роботи системи	46
2.4 Реалізація вебсистеми управління рестораном.....	50
Висновки до другого розділу.....	53
РОЗДІЛ 3. ОПИС РОБОТИ ТА ТЕСТУВАННЯ ВЕБСИСТЕМИ УПРАВЛІННЯ РЕСТОРАНОМ	54
3.1 Порядок встановлення та налаштування параметрів платформи.....	54
3.2 Опис роботи з додатком для персоналу Zesty Admin.....	55
3.3 Опис роботи з клієнтським додатком Zesty Client	66
3.4 Тестування роботи програмного забезпечення.....	69
Висновки до третього розділу	72
ВИСНОВКИ	73
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	75
ДОДАТКИ	78

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс (application programming interface)

БД – база даних

ПЗ – програмне забезпечення

JWT – стандарт автентифікації на основі токенів доступу (JSON Web Token)

GraphQL – мова запитів для API та середовище їх виконання (Graph Query Language)

REST – архітектурний стиль взаємодії компонентів розподіленої системи (Representational State Transfer)

MVC – патерн розділення логіки додатку на три компоненти (Model-View-Controller)

ODM – об'єктно-документне відображення для NoSQL баз даних (Object Document Mapper)

OOP – об'єктно-орієнтоване програмування (Object-Oriented Programming)

POS – система автоматизації торговельних операцій (Point of Sale)

NoSQL – клас нереляційних систем управління базами даних (Not Only SQL)

UI – інтерфейс користувача (User Interface)

UML – уніфікована мова моделювання (Unified Modeling Language)

ВСТУП

Обґрунтування актуальності теми. У сучасних умовах розвитку ресторанного бізнесу та зростання конкуренції на ринку громадського харчування впровадження інформаційних технологій для автоматизації управлінських процесів набуває особливої актуальності. Традиційні методи управління, засновані на паперовому документообігу та ручній обробці інформації, призводять до помилок у замовленнях, затримок в обслуговуванні та ускладнюють прийняття управлінських рішень.

Веборієнтовані системи управління дозволяють забезпечити централізований доступ до інформації з будь-якого пристрою, автоматизувати рутинні операції та надати керівництву актуальні дані для аналізу діяльності. Аналіз існуючих рішень показав, що більшість систем є комерційними продуктами з високою вартістю підписки та обмеженими можливостями кастомізації, що обумовлює доцільність розробки власної вебсистеми управління рестораном [1, 2].

Мета роботи – розробка алгоритмів роботи та реалізація вебсистеми управління рестораном з використанням сучасних технологій веб-розробки.

Завдання роботи. Встановлена мета обумовлює наступні завдання:

- проведення аналізу задач управління рестораном та напрямків їх автоматизації;
- дослідження існуючих програмних аналогів та визначення їх переваг і недоліків;
- визначення архітектури та узагальненої структури вебсистеми управління рестораном;
- обґрунтування та вибір засобів реалізації системи;
- розробка інформаційного забезпечення системи (структури бази даних);
- проектування структурних складових та алгоритмів роботи системи;
- реалізація програмного комплексу вебсистеми управління рестораном;

– тестування розробленої системи та оцінка її працездатності.

Об'єкт дослідження – автоматизація процесів управління діяльністю ресторану.

Предмет дослідження – використання веборієнтованих технологій для розробки системи управління рестораном.

Методи дослідження. У роботі було використано методи системного аналізу для дослідження предметної області, методи об'єктно-орієнтованого проектування для розробки архітектури системи, методи веб-програмування для реалізації клієнтської та серверної частин застосунку, методи моделювання даних для проектування структури бази даних [16].

Практичне значення отриманих результатів. Розроблена вебсистема управління рестораном може бути використана у реальних умовах роботи закладів громадського харчування для автоматизації процесів управління меню, замовленнями, персоналом та фінансовим обліком. Модульна архітектура системи дозволяє розширювати функціонал шляхом додавання нових модулів відповідно до потреб бізнесу. Система може бути адаптована для використання у ресторанах, кафе, барах та інших закладах харчування різного масштабу.

Апробація результатів роботи. За результатами дослідження, що викладені у випускній кваліфікаційній роботі, було опубліковано статтю: Мотицький, Н., Фант, М. (2026). Архітектура системи машинного навчання для прогнозування попиту та оптимізації закупівель у ресторанному бізнесі. Information Technology: Computer Science, Software Engineering and Cyber Security. 2026. № 1. DOI: <https://doi.org/10.32782/IT/2026-1-20>.

Структура та обсяг роботи. Випускна робота складається з вступу, 3-х розділів, висновків, списку використаних джерел із 32 найменувань (3 стор.), 1 додатка (20 стор.), 31 таблиці та 46 ілюстрацій по тексту. Повний обсяг роботи становить 93 сторінки, з них 73 сторінки основного тексту.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ УПРАВЛІННЯ РЕСТОРАНОМ

1.1 Аналіз аналогів програмного продукту

Сучасний ринок програмного забезпечення для ресторанного бізнесу представлений як універсальними POS-системами, так і спеціалізованими хмарними рішеннями [7]. Для визначення стандартного функціоналу та обґрунтування власної розробки було проаналізовано чотири найпоширеніші системи: Poster POS, TurboPOS, Odoo POS та Syrve.

Poster POS (joinposter.com) – хмарна система управління рестораном українського виробника, орієнтована на заклади різного масштабу: від невеликих кав'ярень до мережевих ресторанів. Система побудована за принципом «все в одному» та охоплює весь операційний цикл закладу – від прийому замовлення до фінансової звітності.

Головна сторінка системи (рис. 1.1.) демонструє позиціонування продукту: великий заголовок із описом ключових переваг, блоки з переліком функціональних можливостей та кнопка безкоштовного підключення. Навігаційне меню містить розділи для різних типів бізнесу – ресторан, кав'ярня, фаст-фуд, що свідчить про орієнтацію на широку цільову аудиторію.

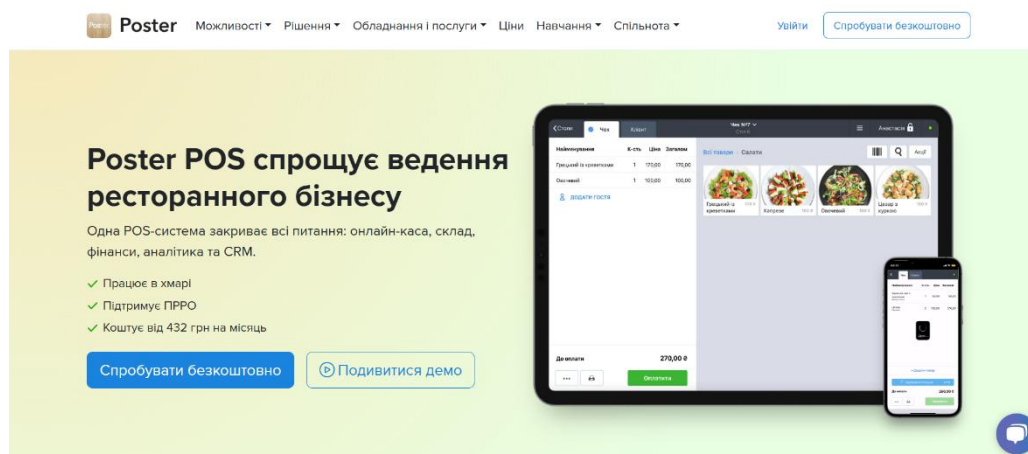


Рис. 1.1. Головна сторінка системи Poster POS

Функціонал системи охоплює управління меню із зазначенням цін, модифікаторів та варіантів страв; роботу із замовленнями в залі та на доставку;

складський облік з автоматичним списанням інгредієнтів при продажу; управління персоналом з обліком робочого часу та розрахунком зарплати; програму лояльності для постійних клієнтів. Система надає доступ до детальної аналітики: динаміка виручки, популярні позиції меню, завантаженість по годинах. Poster POS забезпечує синхронізацію з фіскальними реєстраторами та підтримує офлайн-режим роботи при відсутності інтернет-з'єднання.

На рисунку 1.2. зображено інтерфейс POS-терміналу в режимі прийому замовлення: ліворуч розташована панель з категоріями та позиціями меню, праворуч – поточне замовлення із загальною сумою та кнопками завершення. Інтерфейс адаптований для роботи як з тачскрін-терміналами, так і зі звичайними комп'ютерами.

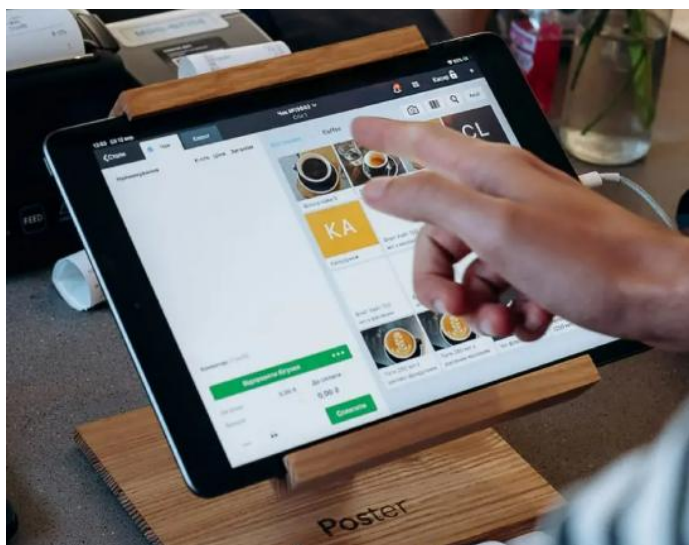


Рис. 1.2. Демонстрація роботи POS-терміналу в інтерфейсі браузера

Poster POS забезпечує сучасний адаптивний інтерфейс, мобільні застосунки для персоналу, вбудовану аналітику продажів, інтеграцію зі складським обліком та службами доставки. До недоліків системи належать обов'язкова платна підписка після завершення тріального періоду, обмежені можливості адаптації під специфічні потреби конкретного закладу, а також надлишковий для малих закладів функціонал.

TurboPOS (turbopos.net) – система автоматизації ресторанного бізнесу з акцентом на роботу з касовими та сенсорними терміналами. Система орієнтована

					ІПЗ.КР.Б-121-26-ІЗ	Арк.
						10

на автоматизацію ключових операційних процесів закладу: касове обслуговування, управління кухнею, складський облік та організація доставки.

Головна сторінка TurboPOS (рис. 1.3.) демонструє лаконічний підхід до представлення продукту: чіткий перелік підтримуваних типів обладнання, коротка характеристика основних модулів та посилання на демонстраційну версію. Структура сайту вказує на технічну спрямованість системи – акцент зроблено на сумісності з різними апаратними платформами та типами пристроїв.

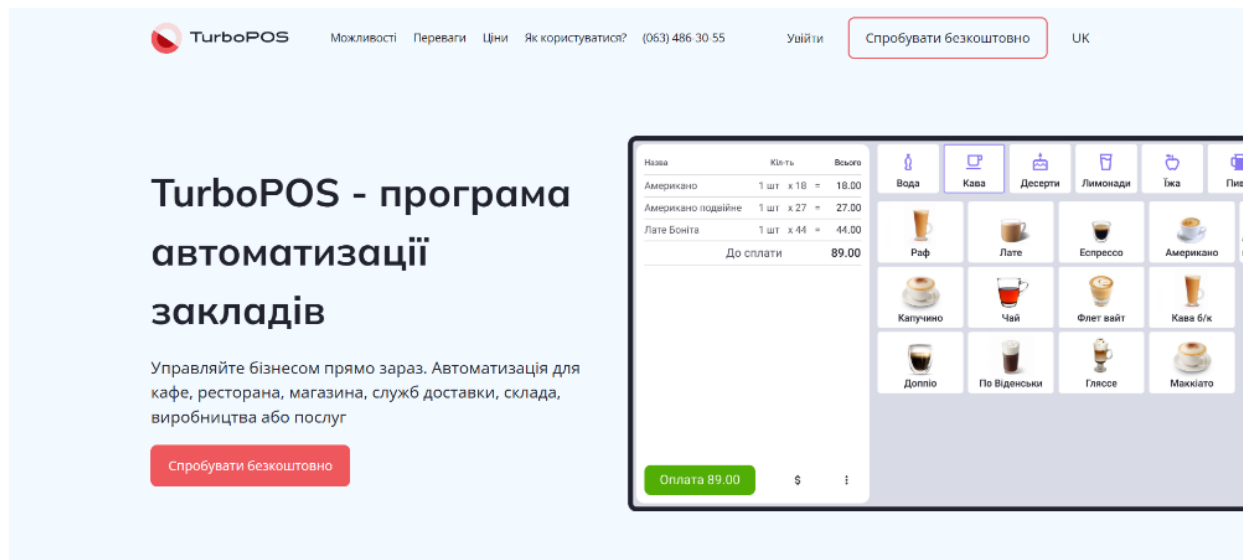


Рис. 1.3. Головна сторінка сайту «TurboPOS»

Система забезпечує роботу з широким спектром сенсорних POS-терміналів та фіскального обладнання, підтримує кухонні дисплеї для відображення активних замовлень, має вбудований модуль складського обліку з контролем залишків та автоматичним списанням при продажу. Передбачено інтеграцію з локальними службами доставки та агрегаторами замовлень. Система функціонує в офлайн-режимі при відсутності інтернет-з'єднання, зберігаючи дані локально з подальшою синхронізацією.

На рисунку 1.4. зображено інтерфейс панелі керування TurboPOS: у верхній частині відображається панель з продажами, центральна зона містить діаграму по продажам за день, а праворуч відображена інформація про популярні товари. Дизайн інтерфейсу є функціональним, однак візуально поступається

сучасним аналогам – використовуються щільні блоки з текстом без виражених ілюстративних елементів.

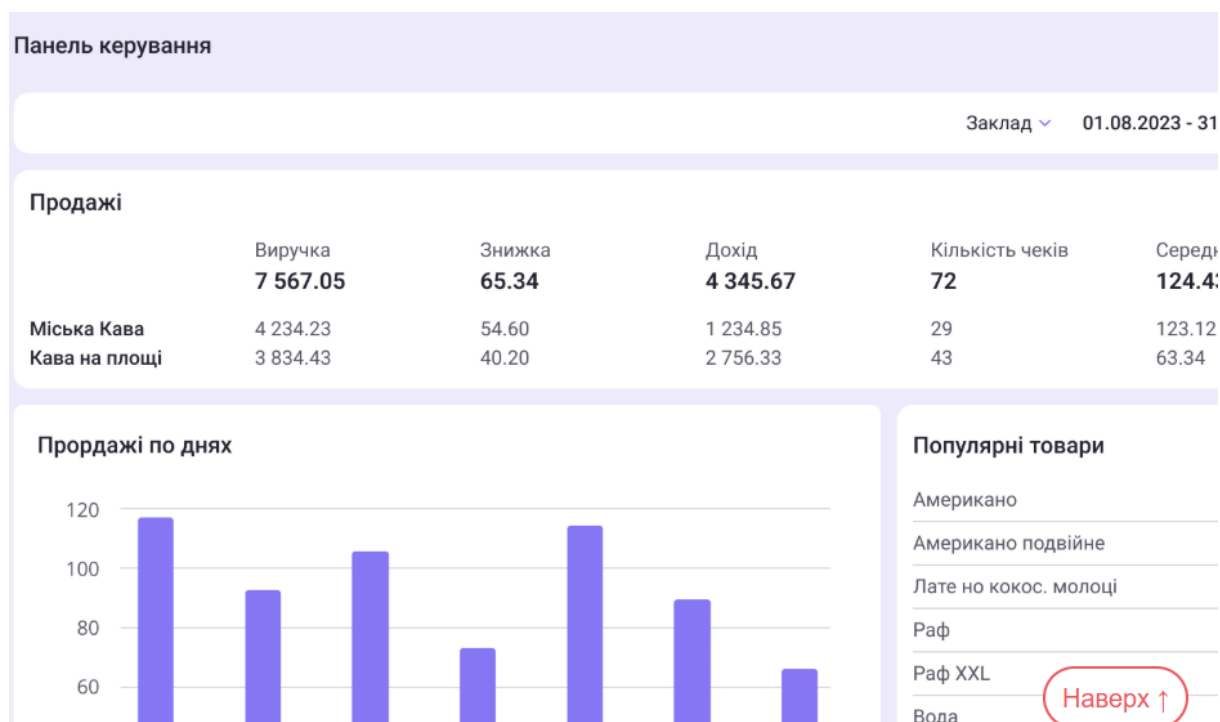


Рис. 1.4. Інтерфейс панелі керування системи

TurboPOS підтримує широкий спектр сенсорних POS-терміналів та касового обладнання, має вбудований модуль складського обліку, забезпечує інтеграцію з локальними службами доставки та функціонує в офлайн-режимі. Серед недоліків слід відзначити застарілий дизайн інтерфейсу, що поступається сучасним аналогам, приховання частини функціоналу за платними тарифними планами та обмежені можливості адаптації системи під конкретний заклад.

Odoo POS (odoo.com/app/point-of-sale-restaurant) – ресторанний модуль у складі повноцінної ERP-системи Odoo, що забезпечує глибоку інтеграцію з іншими модулями платформи: CRM, складом, бухгалтерією та програмою лояльності. На відміну від спеціалізованих POS-систем, Odoo POS є частиною єдиної екосистеми, що робить його привабливим для середніх та великих закладів з комплексними вимогами до автоматизації бізнес-процесів.

Головна сторінка модуля Odoo POS (рис. 1.5.) відображає маркетингову концепцію продукту: акцент на простоті налаштування, підтримці різних типів закладів та ключових функціональних блоках. Сторінка структурована за

сценаріями використання – зал, кухня, аналітика, інтеграції – що дозволяє потенційним клієнтам швидко оцінити відповідність системи своїм потребам.

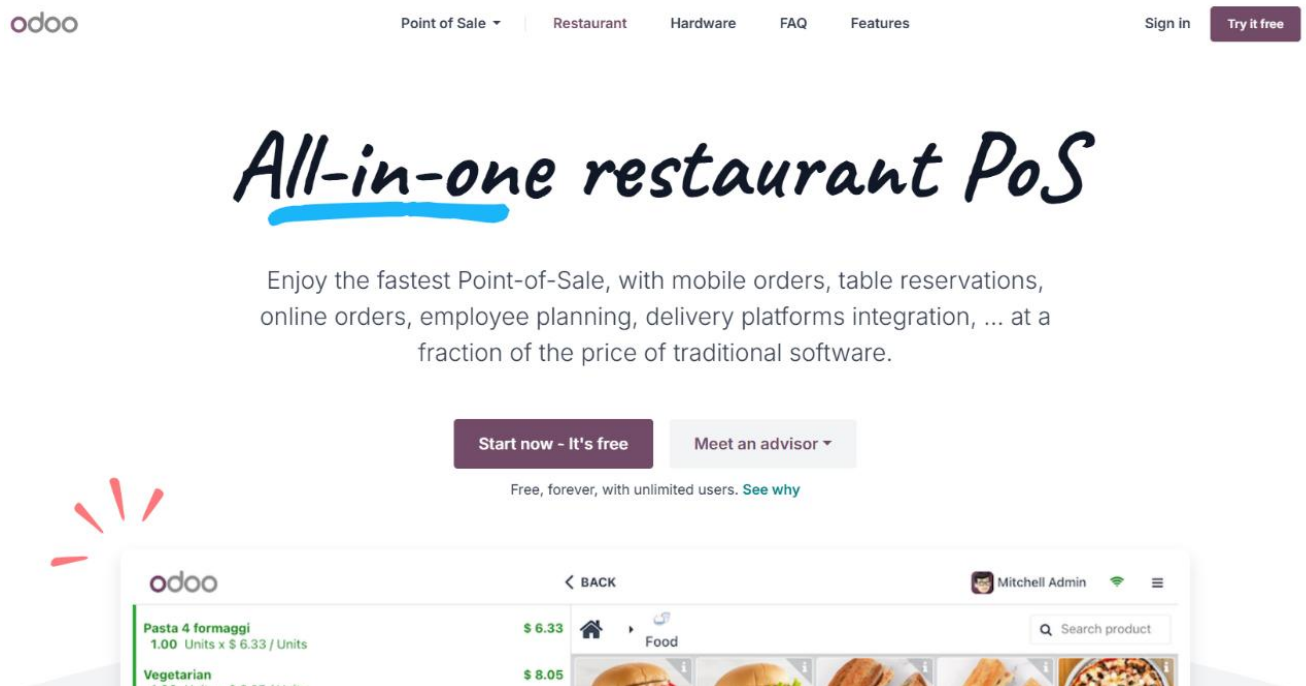


Рис. 1.5. Головна сторінка модулю POS у Odoo

Функціонал системи охоплює управління меню та цінами з підтримкою варіантів і модифікаторів страв; обробку замовлень із прив'язкою до столиків та секцій залу; передачу замовлень на кухонні дисплеї; управління програмою лояльності та знижками. Завдяки інтеграції зі складським модулем Odoo автоматично списуються інгредієнти при продажу, а бухгалтерський модуль отримує всі фінансові операції в режимі реального часу. Система підтримує роботу з відкритим вихідним кодом, що дозволяє розширювати функціонал без додаткових ліцензійних витрат.

На рисунку 1.6. зображено візуальну схему залу з розташуванням столів – одну з ключових особливостей Odoo POS. Кожен столик відображається у вигляді графічного елемента з позначенням поточного статусу: вільний, зайнятий або з активним замовленням. Інтерфейс дозволяє адміністратору налаштовувати розташування столів відповідно до реального планування залу, що суттєво спрощує роботу персоналу.

Custom floor plans

Next-level teamwork. Real-time floorplan allows your staff to immediately see which tables need assistance, keeping the clients happy and turning tables faster.



Рис. 1.6. Схема залу з розташуванням столів

Odoo POS вирізняється глибокою інтеграцією з усіма модулями ERP-системи, візуальним управлінням залом та можливістю розширення через відкритий код. Водночас початкове налаштування є досить складним і потребує залучення кваліфікованих спеціалістів, інтерфейс не повністю адаптований для українського ринку та нормативної бази, а для хмарного розгортання необхідна платна підписка.

Syrve (smartcafe.com.ua/uk/infocentr/syrve) – система автоматизації ресторанного бізнесу, широко розповсюджена на ринку України та країн Центральної і Східної Європи. Позиціонується як комплексне enterprise-рішення, що охоплює весь цикл роботи ресторану: від касового обслуговування до управління складом і персоналом. Система орієнтована передусім на середні та великі заклади, а також на мережі ресторанів з централізованим управлінням.

Головна сторінка Syrve (рис. 1.7.) відображає enterprise-позиціонування продукту: акцент на масштабованості, підтримці мережеских закладів та глибокій інтеграції з обладнанням. Сторінка містить перелік сертифікованих партнерів по впровадженню та посилання на успішні кейси великих ресторанних мереж, що підкреслює орієнтацію системи на корпоративний сегмент ринку.

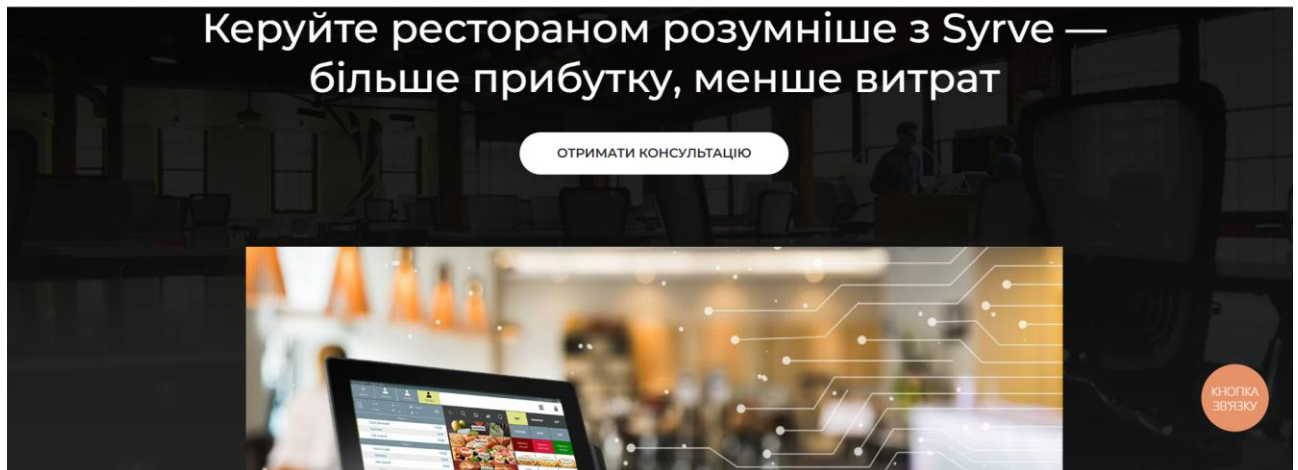


Рис. 1.7. Головна сторінка системи Syrve

Функціонал системи охоплює повний спектр операційних процесів ресторану. Модуль касового обслуговування підтримує широкий спектр POS-терміналів та фіскальних реєстраторів, включаючи вітчизняне обладнання. Складський модуль забезпечує калькуляцію собівартості страв на основі рецептур з урахуванням актуальних цін на інгредієнти, автоматичне списання при продажу та контроль залишків. Аналітичний модуль надає детальну звітність у режимі реального часу: виручка по касирах, популярність позицій меню, порівняльний аналіз по закладах мережі. Для мережевих ресторанів передбачено централізоване управління меню та цінами з можливістю локальних коригувань для кожного закладу окремо.

На рисунку 1.8. зображено інтерфейс модуля замовлень Syrve: ліва панель містить категорії та позиції меню, центральна область відображає активні замовлення по столиках, а права панель показує деталі обраного замовлення. Дизайн інтерфейсу виконано у стилі настільних застосунків середини 2010-х років — щільна компоновка елементів, переважання текстових кнопок над іконографічними, відсутність сучасних UI-патернів. Попри функціональність, такий підхід до дизайну поступається сучасним веборієнтованим системам за зручністю освоєння та візуальною привабливістю, що може ускладнювати

навчання нового персоналу. Інтерфейс насичений функціональними елементами, що відображає орієнтацію системи на досвідчений персонал великих закладів.

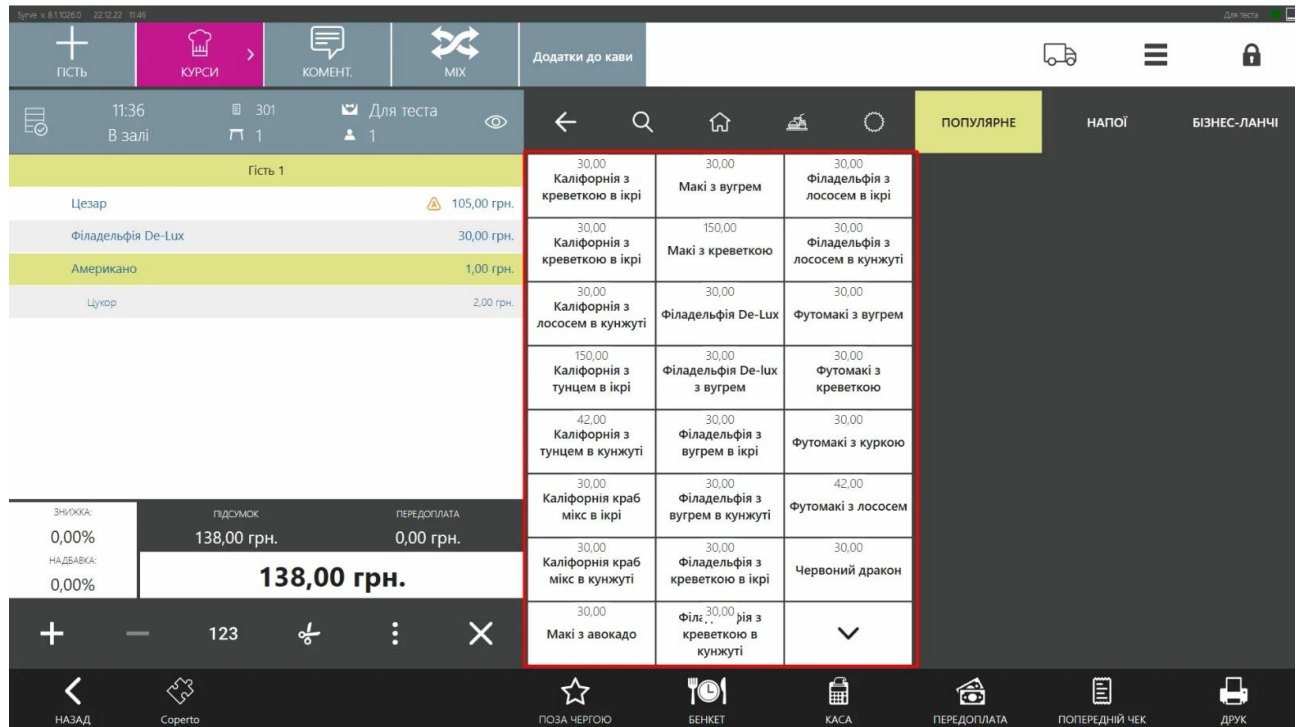


Рис. 1.8. Інтерфейс модуля замовлень Sygve

Sygve вирізняється глибокою інтеграцією з POS-обладнанням та фіскальними реєстраторами, потужним модулем складського обліку з калькуляцією собівартості, детальною аналітикою та підтримкою мережеских закладів. До недоліків системи належать висока вартість ліцензування та впровадження, необхідність залучення сертифікованих партнерів для налаштування, надлишковий функціонал для малих закладів та відсутність відкритого API для сторонніх інтеграцій.

На основі проведеного аналізу складено порівняльну таблицю функціональних можливостей розглянутих систем та розробленої вебсистеми Zesty (табл. 1.1.). Проведений аналіз показав, що більшість існуючих систем є комерційними продуктами з обмеженими можливостями адаптації під специфічні бізнес-процеси конкретного закладу. Використання відкритих технологій (NestJS, React.js, MongoDB) дозволяє розширювати систему без додаткових ліцензійних витрат, а модульна архітектура забезпечує незалежне

масштабування окремих компонентів. Реалізація двох окремих клієнтських застосунків – інтерфейсу для персоналу та публічного для відвідувачів – є унікальною особливістю розробленої системи, яка відсутня в жодному з розглянутих аналогів.

Таблиця 1.1.

Порівняльний аналіз вебсистем управління рестораном

Критерій	Poster POS	TurboPOS	Odoo POS	Syrve	Zesty (власна розробка)
Управління меню	Повна	Повна	Повна	Повна	Повна
Управління замовленнями	Повна	Повна	Повна	Повна	Повна
Управління столиками	Повна	Часткова	Повна	Часткова	Повна
Управління персоналом	Часткова	Відсутня	Повна	Відсутня	Повна
Фінансовий облік	Повна	Часткова	Повна	Повна	Повна
Аналітика та звіти	Повна	Часткова	Повна	Повна	Повна
Рольова модель доступу	Часткова	Часткова	Повна	Часткова	Повна
Відкритий вихідний код	Відсутня	Відсутня	Часткова	Відсутня	Повна
Безкоштовне використання	Відсутня	Відсутня	Часткова	Відсутня	Часткова
Українська локалізація	Повна	Повна	Відсутня	Повна	Повна
Кастомізація під бізнес	Часткова	Часткова	Часткова	Повна	Повна

1.2 Постановка задачі

Проведений аналіз діяльності сучасних закладів громадського харчування показав необхідність впровадження автоматизованих інформаційних систем для ефективного управління процесами обслуговування клієнтів, обробки замовлень, контролю роботи персоналу та ведення обліку ресурсів. Відсутність єдиної інформаційної системи призводить до зростання кількості помилок, затримок в обслуговуванні та ускладнює процес управління рестораном.

Вебсистема управління рестораном повинна автоматизувати ключові бізнес-процеси закладу громадського харчування. Перший з них є управління меню та стравами. Система забезпечує створення та редагування категорій меню, додавання, редагування і видалення страв із зазначенням назви, опису, ціни та

категорії. Окремо передбачено управління інгредієнтами та напівфабрикатами страв – ведення переліку та визначення їх кількості у складі кожної страви.

Наступним по важливості варто виділити обробку замовлень, бо це є один із центральних модулів системи. Офіціант може створювати нові замовлення з прив'язкою до столика, додавати страви із зазначенням кількості, змінювати статуси замовлення (відкрите, в процесі приготування, завершене, скасоване). Система автоматично розраховує вартість замовлення та забезпечує формування чеку після завершення.

Управління столиками реалізовано як окремий модуль, що передбачає створення та налаштування столиків із зазначенням секції (основний зал, тераса, VIP, бар) та місткості. Кожному столику може бути призначено відповідального офіціанта, а система в режимі реального часу відстежує поточний статус зайнятості, що дозволяє персоналу оперативно реагувати на зміни у завантаженості залу.

Управління персоналом здійснюється через окремий адміністративний модуль, де ведеться база даних співробітників із зазначенням посади (кухар, офіціант, бармен, адміністратор), поточного статусу (активний, у відпустці, звільнений), контактної інформації та даних про заробітну плату. Система підтримує три типи нарахування зарплати – погодинний, місячний та позмінний, що дозволяє враховувати специфіку різних категорій персоналу закладу.

Фінансовий облік у системі охоплює декілька взаємопов'язаних підмодулів. Перший відповідає за відстеження витрат за категоріями (оренда, комунальні послуги, інгредієнти, маркетинг, обслуговування обладнання). Другий забезпечує управління виплатами заробітної плати з відстеженням статусу кожної виплати (очікує, виплачено, скасовано). На основі цих даних формуються фінансові звіти за обраний період, що містять загальний дохід, витрати в розрізі категорій, фонд заробітної плати та розрахований чистий прибуток.

Рольова модель доступу реалізована на основі JWT-токенів і визначає права кожного користувача при вході до системи. Адміністратор отримує повний доступ до всіх модулів. Офіціант працює із замовленнями та переглядом меню. Кухар бачить лише активні замовлення та може змінювати статуси приготування страв. Бармен, аналогічно до офіціанта, працює із замовленнями своєї секції та змінює їх статуси.

Аналітика та звітність доступна адміністратору у вигляді зведеної інформаційної панелі: динаміка доходів по днях, розподіл витрат за категоріями у вигляді діаграм, порівняння фінансових показників за різні періоди. Це дозволяє приймати обґрунтовані управлінські рішення на основі актуальних даних замість покладання на інтуїцію чи ручний підрахунок.

Для досягнення поставленої мети в рамках кваліфікаційної роботи необхідно вирішити наступні задачі:

- провести аналіз предметної області ресторанного бізнесу, визначити основні бізнес-процеси та ролі користувачів системи;
- дослідити існуючі програмні аналоги та визначити їх переваги і недоліки;
- обґрунтувати вибір клієнт-серверної архітектури та технологічного стеку;
- спроектувати структуру бази даних MongoDB та GraphQL-схеми основних сутностей;
- реалізувати серверну частину на базі NestJS із GraphQL API та модульною архітектурою;
- реалізувати клієнтську частину на базі React.js з Apollo Client та Material UI;
- забезпечити коректну роботу бізнес-логіки: розрахунок вартості замовлень, обробку статусів, формування фінансових звітів;
- провести тестування розробленої системи та оцінити її працездатність.

Результатом виконання поставлених задач є функціонуюча вебсистема управління рестораном, яка може бути розгорнута в реальних умовах роботи закладу громадського харчування та масштабована відповідно до потреб бізнесу.

Для систематизації вимог до вебсистеми управління рестораном складено перелік функціональних вимог із зазначенням пріоритету реалізації (табл. 1.2.). Пріоритет реалізації визначався виходячи з критичності вимоги для базового функціонування системи : вимоги з високим пріоритетом є обов'язковими для першого релізу, з середнім – реалізуються в основному функціональному блоці, з низьким – є розширенням базового функціоналу.

Таблиця 1.2.

Функціональні вимоги до вебсистеми управління рестораном

№	Функціональна вимога	Пріоритет
1	Авторизація користувачів з розмежуванням доступу за ролями	Високий
2	Управління категоріями меню (створення, редагування, видалення)	Високий
3	Управління стравами з прив'язкою до категорії та інгредієнтів	Високий
4	Управління інгредієнтами та складом	Високий
5	Управління столиками та секціями залу	Високий
6	Створення та обробка замовлень з автоматичним розрахунком суми	Високий
7	Відстеження статусів страв у реальному часі (кухонна панель)	Високий
8	Формування чеків при завершенні замовлення	Високий
9	Управління персоналом із зазначенням ролі та статусу	Середній
10	Облік витрат за категоріями	Середній
11	Облік доходів за категоріями (продажі, кейтеринг, заходи)	Середній
12	Нарахування та облік виплат заробітної плати	Середній
13	Формування фінансових звітів за довільний період	Середній
14	AI-аналіз прибутковості страв та генерація рекомендацій	Низький
15	Інтерактивний AI-чат для запитів про бізнес-показники	Низький
16	Публічний перегляд меню відвідувачами (zesty-client)	Середній

Окрім функціональних вимог, визначено нефункціональні вимоги, що встановлюють обмеження на якісні характеристики системи (табл. 1.3.). Нефункціональні вимоги охоплюють п'ять ключових категорій: продуктивність, безпека, надійність, масштабованість та зручність використання. Дотримання

цих вимог забезпечує не лише коректну роботу системи, а й належний рівень захисту даних персоналу та фінансової інформації закладу, що є критично важливим для промислового застосування розробленої системи.

Таблиця 1.3.

Нефункціональні вимоги до вебсистеми управління рестораном

Категорія	Вимога
Продуктивність	Час відповіді API не більше 500 мс для типових запитів
Безпека	Захист адміністративного інтерфейсу через IP-whitelisting
Безпека	Зберігання паролів у хешованому вигляді (bcrypt)
Безпека	Авторизація на рівні кожного GraphQL-запиту (JWT + RolesGuard)
Надійність	Підтримка одночасної роботи не менше 50 користувачів
Масштабованість	Горизонтальне масштабування бекенду без змін архітектури
Доступність	Доступність системи 99.5% (забезпечується Vercel та MongoDB Atlas)
Зручність	Адаптивний інтерфейс для пристроїв з роздільною здатністю від 1280×720

1.3 Вибір архітектури вебсистеми

Вибір правильної архітектури є ключовим рішенням при проектуванні вебсистеми, оскільки визначає масштабованість, продуктивність і зручність подальшої розробки [13]. У цьому параграфі розглянуто основні архітектурні підходи та обґрунтовано вибір для вебсистеми управління рестораном.

Монолітна архітектура передбачає об'єднання всіх компонентів системи в єдиний застосунок, що розгортається як одне ціле. Такий підхід є природним вибором для невеликих проєктів на початкових етапах розробки, коли команда невелика, а вимоги до системи ще формуються. Проте зі зростанням функціоналу моноліт стає дедалі важчим у підтримці – будь-яке оновлення потребує повного перерозгортання всього застосунку. Переваги: простота розробки та налагодження на початкових етапах, відсутність мережових накладних витрат між модулями. Недоліки: ускладнена масштабованість окремих компонентів,

зростаюча складність підтримки при збільшенні кодової бази, ризик повної відмови системи при збої будь-якого компонента.

Мікросервісна архітектура розбиває систему на множину незалежних сервісів, кожен з яких відповідає за окрему функціональну область [14]. Цей підхід набув широкого поширення у великих технологічних компаніях, де необхідність незалежного масштабування окремих компонентів є критичною вимогою. Водночас мікросервіси вносять суттєву операційну складність, що робить їх надмірним рішенням для систем середнього масштабу. Переваги: незалежне масштабування окремих сервісів, технологічна різноманітність, висока відмовостійкість. Недоліки: значна операційна складність, необхідність інфраструктури оркестрації, підвищені вимоги до досвіду команди розробників.

Клієнт-серверна архітектура передбачає чітке розділення системи на клієнтську частину, що відповідає за відображення інформації та взаємодію з користувачем, та серверну частину, що обробляє бізнес-логіку і забезпечує доступ до даних. Для вебсистеми управління рестораном обрано саме цю архітектуру, оскільки вона забезпечує централізоване управління даними, контроль доступу за ролями, зручність розгортання та достатні можливості масштабування при помірній складності. Додатковою перевагою клієнт-серверного підходу є можливість незалежного розвитку клієнтської та серверної частин: оновлення бізнес-логіки на сервері не вимагає змін у клієнтських застосунках, і навпаки. Це особливо актуально для розробленої системи, яка передбачає два окремих клієнтських застосунки з різними вимогами до інтерфейсу та рівня доступу.

На рисунку 1.9. зображено загальну схему клієнт-серверної архітектури системи. Схема поділена на три рівні. На верхньому рівні розташовані клієнтські застосунки – zesty-admin для персоналу та zesty-client для відвідувачів – які взаємодіють з сервером через захищений протокол HTTPS. Середній рівень представлений серверною частиною на базі NestJS, що включає GraphQL API як єдину точку входу для всіх клієнтських запитів, модуль авторизації на основі

JWT та набір функціональних модулів бізнес-логіки. Нижній рівень містить базу даних MongoDB Atlas, до яких сервер звертається для збереження даних. Стрілки на схемі показують напрямок взаємодії між компонентами: клієнти надсилають GraphQL-запити на сервер, сервер обробляє їх через відповідні модулі та повертає результат, паралельно звертаючись до бази даних або зовнішніх сервісів за необхідними даними.

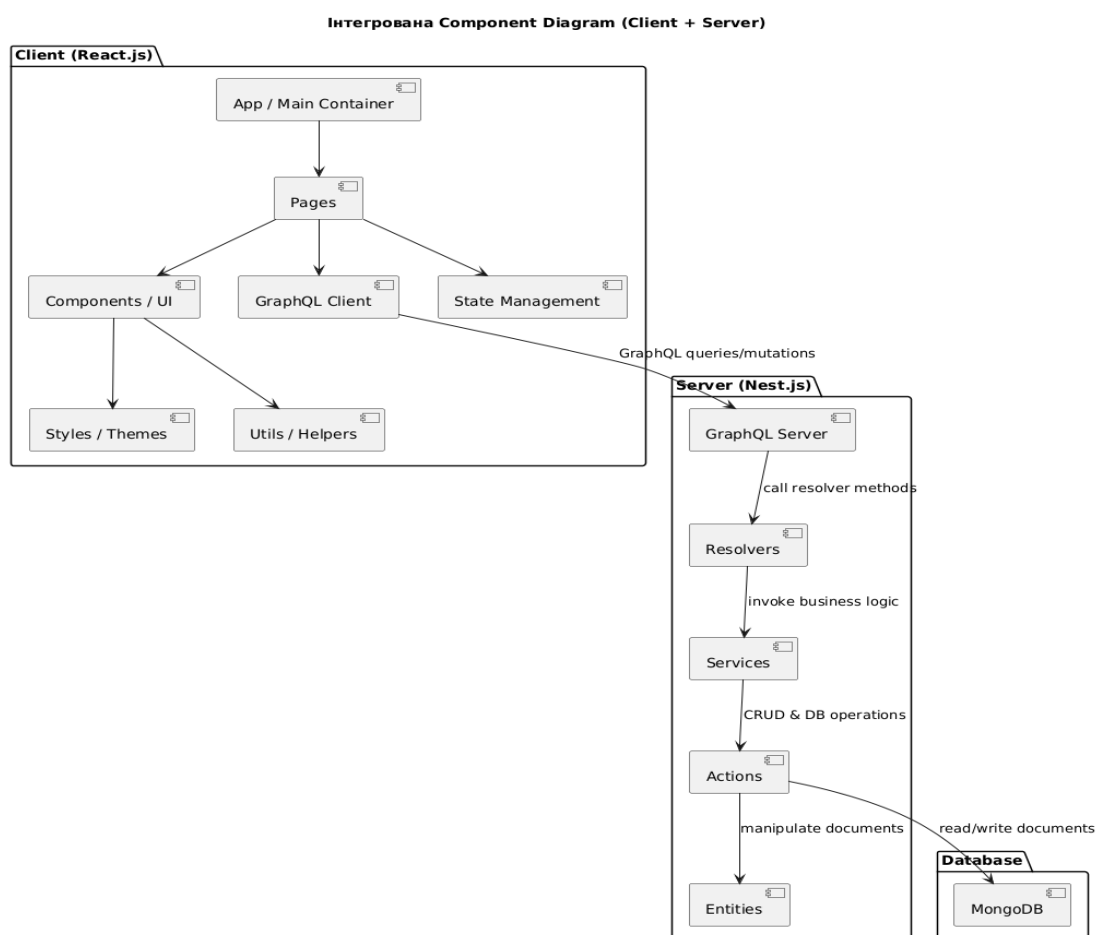


Рис. 1.9. Загальна схема клієнт-серверної архітектури системи

Для організації взаємодії між клієнтською та серверною частинами розглянуто два підходи: REST API та GraphQL. REST API є класичним підходом, що базується на ресурсно-орієнтованих ендпоінтах і HTTP-методах (GET, POST, PUT, DELETE) [8]. Кожен ресурс системи – замовлення, страва, співробітник – має власний URL, а структура відповіді фіксована і визначається сервером незалежно від потреб клієнта. GraphQL, розроблений компанією Meta у 2012 році та відкритий у 2015 році, пропонує принципово інший підхід: єдина точка

доступу з власною мовою запитів дозволяє клієнту точно описати, які поля і в якій структурі він очікує отримати [5, 11]. Це усуває одну з ключових проблем REST – надлишкову або недостатню вибірку даних. Порівняльний аналіз обох підходів за ключовими критеріями наведено у таблиці 1.4 [27].

Таблиця 1.4.

Порівняння REST та GraphQL

Характеристика	REST API	GraphQL
Точки доступу	Множина ендпоінтів	Єдиний ендпоінт
Отримання даних	Фіксована структура відповіді	Клієнт обирає потрібні поля
Надлишкові дані	Часто присутні (over-fetching)	Відсутні
Вкладені запити	Декілька запитів (N+1 проблема)	Один запит для пов'язаних даних
Типізація схеми	Опціональна (OpenAPI)	Вбудована, строга
Самодокументування	Потребує окремої документації	Схема є документацією
Поріг входу	Низький	Середній

Для даного проєкту обрано GraphQL, оскільки система передбачає складні вкладені запити (замовлення зі стравами та їх інгредієнтами, фінансові звіти з деталізацією за категоріями), а строга типізація схеми суттєво спрощує розробку клієнтської частини та забезпечує надійну валідацію даних. Суттєвою перевагою GraphQL у контексті розробленої системи є вирішення проблеми надлишкової або недостатньої вибірки даних (over-fetching та under-fetching), характерної для REST API. Наприклад, при відображенні списку замовлень кухонної панелі кухарю потрібні лише назви страв та їх статуси, тоді як офіціанту – повна інформація про столик, суму та склад замовлення.

Серверна частина побудована відповідно до принципів модульності NestJS, де кожен модуль є самодостатньою одиницею, що інкапсулює логіку роботи з окремою сутністю [31]. Такий підхід забезпечує чіткий розподіл відповідальностей та спрощує підтримку коду – зміни в одному модулі не впливають на роботу інших.

Для зберігання даних обрано документоорієнтовану базу даних MongoDB [22, 4]. На відміну від реляційних СКБД, де дані зберігаються у жорстко визначених таблицях зі схемою, MongoDB зберігає дані у форматі BSON-документів, що природно відповідає структурам TypeScript-об'єктів і усуває необхідність складного об'єктно-реляційного відображення. Гнучка схема дозволяє еволюцію структури даних без написання міграцій – нові поля додаються до документів без перебудови всієї колекції, що особливо важливо на етапі активної розробки. Горизонтальне масштабування через шардинг та реплікація даних забезпечують стійкість системи до зростаючого навантаження, а хмарне розгортання на MongoDB Atlas звільняє від необхідності адміністрування серверної інфраструктури.

Загальну схему розгортання вебсистеми управління рестораном зображено на рисунку 1.10. Взаємодія компонентів системи буде відбуватися наступним чином: спочатку користувач взаємодіє з React-застосунком через браузер, далі Apollo Client формує GraphQL-запити та надсилає їх на NestJS-сервер, далі сервер обробляє запити через відповідні резолвери, виконує бізнес-логіку та звертається до MongoDB і насамкінець результат повертається клієнту та оновлює інтерфейс.

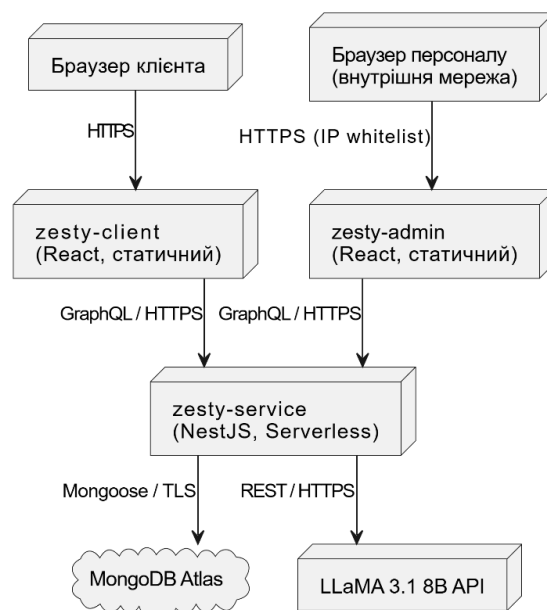


Рис. 1.10. Діаграма розгортання вебсистеми управління рестораном

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для реалізації вебсистеми управління рестораном обрано набір сучасних технологій, що широко використовуються у практиці веброзробки та забезпечують необхідний рівень продуктивності, масштабованості та зручності підтримки. Вибір технологічного стеку передував порівняльний аналіз альтернативних рішень. Кожна технологія обиралась з урахуванням сумісності з рештою стеку, активності підтримки спільнотою та наявності офіційної документації [9]. Пріоритет надавався рішенням з нативною підтримкою TypeScript, оскільки статична типізація суттєво знижує кількість помилок на етапі розробки та спрощує супровід коду у довгостроковій перспективі.

Для серверної частини розглядалися три основні фреймворки екосистеми Node.js (табл. 1.5.): NestJS, Express та Fastify. NestJS обрано завдяки вбудованій підтримці TypeScript, механізму ін'єкції залежностей та нативній інтеграції з GraphQL, що суттєво спрощує розробку великих модульних застосунків порівняно з мінімалістичними альтернативами.

Таблиця 1.5.

Порівняння серверних фреймворків Node.js

Критерій	NestJS	Express	Fastify
TypeScript (вбудований)	Так	Опціонально	Опціонально
Ін'єкція залежностей	Так	Ні	Ні
Підтримка GraphQL	Вбудована	Стороння	Стороння
Модульна архітектура	Так	Ні	Часткова
Рівень абстракції	Високий	Низький	Середній
Продуктивність	Висока	Висока	Дуже висока
Поріг входу	Середній	Низький	Середній

Для клієнтської частини проведено аналіз трьох провідних фреймворків (табл. 1.6.): React, Vue та Angular. React обрано завдяки найбільшій екосистемі бібліотек, офіційній підтримці Apollo Client для роботи з GraphQL та широкому розповсюдженню у комерційній веброзробці, що забезпечує доступність готових рішень для типових задач інтерфейсу.

Таблиця 1.6.

Порівняння клієнтських фреймворків

Критерій	React	Vue	Angular
TypeScript підтримка	Вбудована	Вбудована	Вбудована
Розмір бандлу	Малий	Малий	Великий
Екосистема	Дуже велика	Велика	Велика
Apollo Client інтеграція	Офіційна	Стороння	Стороння
Гнучкість архітектури	Висока	Середня	Низька

На основі проведеного порівняльного аналізу обрано технологічний стек, що забезпечує оптимальне поєднання продуктивності, зручності розробки та підтримки. Детальний опис кожної обраної технології та її призначення зведено у таблицю 1.7.

Таблиця 1.7.

Технологічний стек системи

Технологія	Призначення
Node.js	Середовище виконання серверної частини
NestJS	Серверний фреймворк
GraphQL	Мова запитів API
Mongoose	ODM для MongoDB
MongoDB	База даних
MongoDB Atlas	Хмарне розгортання БД
React.js	Бібліотека клієнтського інтерфейсу
TypeScript	Мова програмування
Vite	Інструмент збірки
Apollo Client	GraphQL-клієнт
Material UI	Бібліотека UI-компонентів
Git	Система контролю версій

Node.js – середовище виконання JavaScript на сервері, побудоване на рушії V8. Використовує подієво-орієнтовану неблокуючу модель вводу-виводу, що забезпечує високу продуктивність при обробці великої кількості одночасних з'єднань [28].

NestJS – прогресивний TypeScript-фреймворк для серверних застосунків. Поєднує принципи об'єктно-орієнтованого та функціонального програмування, надає вбудовану підтримку GraphQL, ін'єкцію залежностей та модульну архітектуру. Вибір NestJS обумовлений строгою структурою проєкту, що спрощує масштабування та підтримку коду.

GraphQL / @nestjs/graphql – мова запитів для API. Пакет @nestjs/graphql забезпечує інтеграцію GraphQL з NestJS у підході code-first: TypeScript-декоратори автоматично генерують GraphQL-схему, що виключає розбіжності між схемою та реалізацією [20].

MongoDB / Mongoose – документоорієнтована NoSQL база даних та ODM-бібліотека для роботи з нею у Node.js. Mongoose надає схему для моделювання даних, вбудовану валідацію та зручний API для побудови запитів [23].

React.js – JavaScript-бібліотека для побудови користувацьких інтерфейсів з компонентним підходом. Використання хуків (useState, useEffect) та контексту дозволяє ефективно керувати станом застосунку без надлишкових залежностей [3].

TypeScript – надмножина JavaScript зі статичною типізацією. Виявлення помилок на етапі компіляції суттєво підвищує надійність коду та покращує досвід розробки завдяки автодоповненню та перевірці типів [25].

Vite – інструмент збірки нового покоління, що використовує нативні ES-модулі для миттєвого запуску сервера розробки та оптимізованої збірки для продакшену [32].

Apollo Client – повноцінна бібліотека для роботи з GraphQL на клієнті. Забезпечує кешування запитів, управління локальним станом, обробку помилок та оновлення даних після мутацій.

Material UI (MUI) – бібліотека React-компонентів, що реалізує специфікацію Google Material Design. Надає готовий набір компонентів (таблиці, форми, діалоги, графіки), що прискорює розробку інтерфейсу та забезпечує консистентний дизайн [26].

Вимоги до апаратного забезпечення визначено з урахуванням характеру навантаження на систему та особливостей хмарного розгортання бази даних (табл. 1.8.). Оскільки база даних MongoDB розгорнута у хмарному середовищі MongoDB Atlas, вимоги до серверного сховища суттєво знижуються – сервер застосунків не зберігає дані локально і виступає виключно як обчислювальний вузол.

Таблиця 1.8.

Вимоги до серверного апаратного забезпечення

Параметр	Мінімальні вимоги	Рекомендовані вимоги
Процесор	2 ядра, 2.0 ГГц	4 ядра, 2.5 ГГц і вище
Оперативна пам'ять	2 ГБ	4 ГБ і більше
Дисковий простір	20 ГБ SSD	50 ГБ SSD
Мережа	100 Мбіт/с	1 Гбіт/с
Операційна система	Linux / Windows Server	Ubuntu 22.04 LTS

Клієнтська частина буде розгорнута як статичний застосунок на CDN-мережі Vercel, тому вимоги до клієнтського пристрою обмежуються наявністю сучасного браузера та стабільного інтернет-з'єднання (табл. 1.9.).

Таблиця 1.9.

Вимоги до клієнтського апаратного забезпечення

Параметр	Мінімальні вимоги	Рекомендовані вимоги
Процесор	2 ядра, 1.5 ГГц	4 ядра, 2.0 ГГц
Оперативна пам'ять	2 ГБ	4 ГБ
Роздільна здатність	1280×720	1920×1080
Браузер	Chrome 90+, Firefox 90+	Chrome 120+, Firefox 120+
Інтернет-з'єднання	1 Мбіт/с	10 Мбіт/с і більше

Для розгортання вебсистеми розглядалися декілька хмарних платформ (табл. 1.10.). Ключовими критеріями відбору були: наявність безкоштовного тарифного плану для початкового розгортання, підтримка автоматичного деплою з Git-репозиторію, можливість створення окремих середовищ для

розробки та production, а також простота налаштування без необхідності адміністрування серверної інфраструктури.

Таблиця 1.10.

Порівняння платформ розгортання

Критерій	Vercel	Railway	Render	AWS
Безкоштовний план	Так	Частково	Так	Ні
Автодеплой з Git	Так	Так	Так	Налаштовується
Preview deployments	Так	Ні	Ні	Ні
Підтримка Node.js	Так	Так	Так	Так
Вбудований CDN	Так	Ні	Частково	Так
Складність налаштування	Низька	Низька	Низька	Висока
Моніторинг та логи	Так	Так	Так	Налаштовується

Vercel обрано як платформу розгортання завдяки нативній підтримці Node.js-застосунків, автоматичному створенню preview-середовищ для кожного pull request та вбудованому CDN. Кожен із трьох репозиторіїв (zesty-service, zesty-admin, zesty-client) має окремий проєкт на Vercel з незалежними конфігураціями середовищ – dev та prod.

Для автоматизації процесів інтеграції та доставки програмного забезпечення обрано GitHub Actions – вбудований у платформу GitHub інструмент для опису pipeline у вигляді YAML-конфігурацій. GitHub Actions дозволяє визначати послідовність кроків (jobs) та умови їх запуску безпосередньо у репозиторії, що забезпечує повну відтворюваність процесу збірки та прозорість для всіх учасників розробки (рис. 1.11.). Переваги обраного рішення: тісна інтеграція з репозиторієм без додаткових налаштувань, безкоштовний ліміт хвилин для публічних репозиторіїв, широкий marketplace готових actions та зберігання конфігурації безпосередньо у репозиторії поряд з кодом. Є можливість налаштувати два окремих типи pipeline: CI-pipeline, що запускається при кожному pull request і виконує лінтинг, тестування та збірку, та CD-pipeline, що автоматично розгортає застосунок на відповідний інстанс Vercel після merge або публікації release.

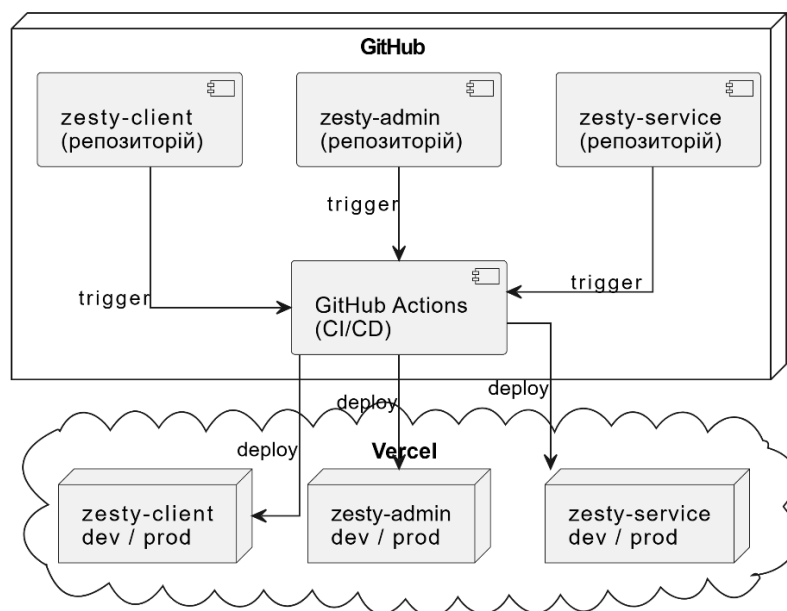


Рис. 1.11. Діаграма розгортання вебсистеми

Висновки до першого розділу

У першому розділі проведено порівняльний аналіз чотирьох існуючих аналогів (Poster POS, TurboPOS, Odoo POS, Syrve). Встановлено, що розглянуті системи є переважно комерційними продуктами з обмеженими можливостями кастомізації, жодна не передбачає інтеграції зі штучним інтелектом, що обумовлює доцільність власної розробки.

Проведено аналіз предметної області ресторанного бізнесу та визначено перелік бізнес-процесів, що підлягають автоматизації: управління меню, замовленнями, персоналом, столиками, фінансовий облік та AI-аналітика. Сформульовано постановку задачі з урахуванням рольової моделі доступу та складено переліки функціональних і нефункціональних вимог до системи.

Обґрунтовано вибір клієнт-серверної архітектури як оптимальної для даного класу систем. На основі порівняльного аналізу альтернатив обрано технологічний стек: NestJS для серверної частини, React.js з Apollo Client для клієнтської частини, MongoDB як система зберігання даних, Vercel як платформа розгортання з GitHub Actions для CI/CD. Визначено вимоги до апаратного забезпечення.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Проектування системи розпочинається з визначення функціональних та нефункціональних вимог, специфікації акторів і варіантів використання, а також побудови об'єктно-орієнтованої структури системи. Результати аналізу є основою для прийняття архітектурних рішень та проектування бази даних [30].

Система повинна забезпечувати виконання таких функціональних вимог: автентифікацію та авторизацію користувачів із розмежуванням прав за ролями; управління меню, стравами, інгредієнтами та напівфабрикатами; обробку замовлень зі зміною статусів у реальному часі; управління столиками та секціями залу; ведення бази даних персоналу з управлінням виплатами; фінансовий облік витрат, доходів та формування звітів; перегляд публічного меню для неавторизованих користувачів.

До нефункціональних вимог належать: час відповіді API не більше 500 мс для типових запитів; захист адміністративного інтерфейсу через IP-whitelisting; зберігання паролів у хешованому вигляді (bcrypt); використання HTTPS для всіх з'єднань; підтримка одночасної роботи не менше 50 користувачів.

З точки зору клієнтської архітектури система складається з двох окремих застосунків з різним призначенням та рівнем доступу.

Перший з них – це zesty-admin, що виконує роль адміністративного інтерфейсу для внутрішнього використання персоналом закладу. Доступ до застосунку обмежений на мережевому рівні через IP-whitelisting: лише пристрої з дозволеними IP-адресами (внутрішня мережа ресторану) можуть відкрити інтерфейс. Після проходження мережевого фільтру користувач проходить автентифікацію із перевіркою ролі. Застосунок надає доступ до всіх операційних модулів системи відповідно до ролі.

Другим є zesty-client, що виконує роль публічного інтерфейсу для відвідувачів ресторану. Доступний без обмежень за IP-адресою. Неавторизований користувач (роль Guest) може переглядати публічне меню та інформацію про страви. Застосунок не потребує автентифікації для базового перегляду і може бути розширений функціоналом онлайн-замовлень у майбутньому.

На рисунку 2.1. зображено схему взаємодії двох фронтендів із серверною частиною системи. Персонал закладу працює через застосунок zesty-admin, доступ до якого обмежено на мережевому рівні засобами IP-whitelisting – запити з недозволених адрес отримують відповідь 403 Forbidden ще до будь-якої обробки. Запити з дозволених адрес проходять перевірку JWT-токена, і лише після успішної авторизації потрапляють до GraphQL API. Відвідувачі ресторану взаємодіють через публічний застосунок zesty-client напряду з GraphQL без мережевих обмежень. Сервер на базі NestJS маршрутизує запити між модулями системи, які зберігають дані у MongoDB Atlas та за необхідності звертаються до зовнішнього Groq AI API [17, 21].

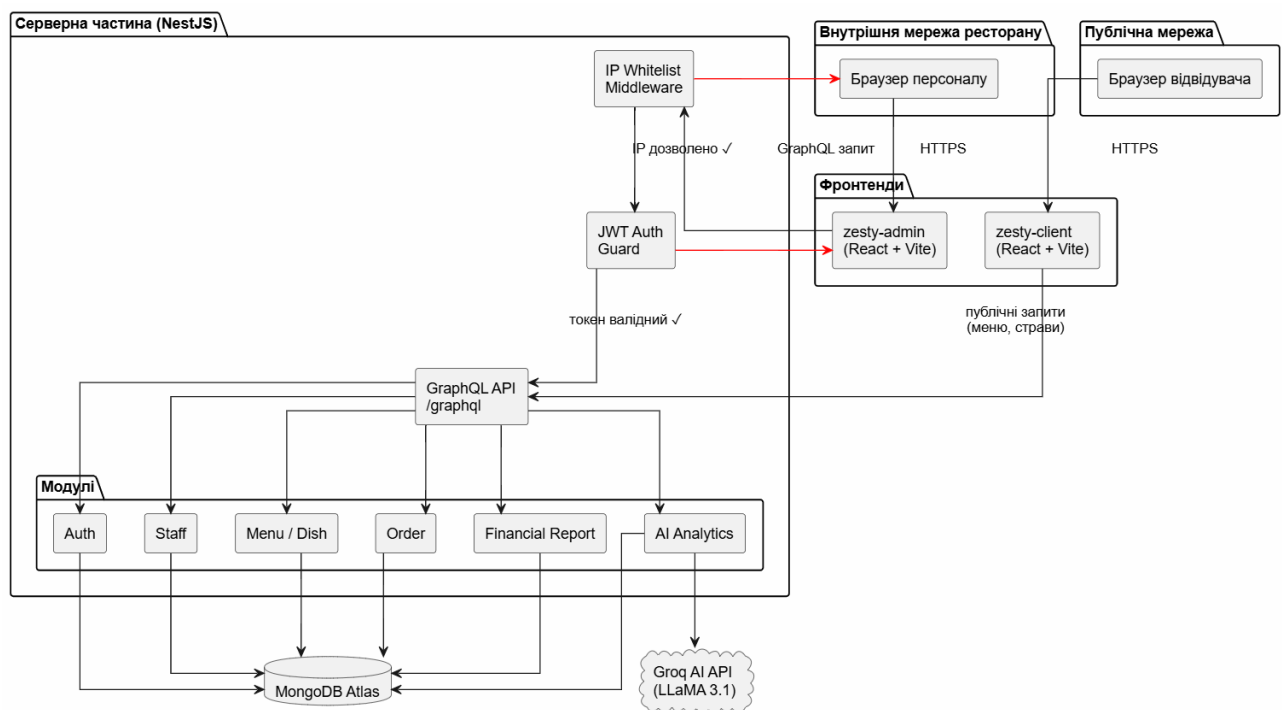


Рис. 2.1. Схема взаємодії двох фронтендів із серверною частиною та механізм IP-whitelistin

Актор у контексті UML діаграм варіантів використання являє собою зовнішню сутність, яка взаємодіє з системою для досягнення певних цілей. Аналіз предметної області дозволив виділити чотирьох основних акторів вебсистеми управління рестораном.

Адміністратор є головним користувачем системи, який має повний доступ до всіх функціональних можливостей. До його обов'язків входить налаштування системи, управління персоналом, формування меню та категорій страв, контроль за столиками та доступ до фінансової звітності. Адміністратор створює облікові записи для працівників, призначає їм ролі та встановлює параметри нарахування заробітної плати. Він також відповідає за управління витратами закладу, фіксуючи оплату оренди, комунальних послуг, закупівлю інгредієнтів та інші операційні витрати. Додатково адміністратор має доступ до модуля AI-аналітики, аналізує ресторанну діяльність.

Офіціант виступає працівником закладу, який безпосередньо обслуговує відвідувачів. Цей актор має можливість створювати нові замовлення, додавати до них страви з меню, змінювати статуси замовлень відповідно до їх виконання та формувати чеки для розрахунку з клієнтами.

Бармен є працівником барної секції закладу, функціонально схожим на офіціанта, однак орієнтованим на обслуговування клієнтів у барі. Цей актор має можливість переглядати замовлення барної секції, створювати нові замовлення, додавати напої та страви, змінювати їх статуси та формувати чеки. Роль бармена відрізняється від ролі офіціанта на рівні розмежування зон обслуговування та специфіки асортименту.

Кухар представляє працівника кухні, відповідального за приготування страв. Основними функціями кухаря є перегляд активних замовлень та зміна статусів готовності окремих страв.

Діаграма варіантів використання відображає функціональні можливості системи з точки зору різних категорій користувачів та демонструє їхню взаємодію з програмним комплексом (рис. 2.2.).

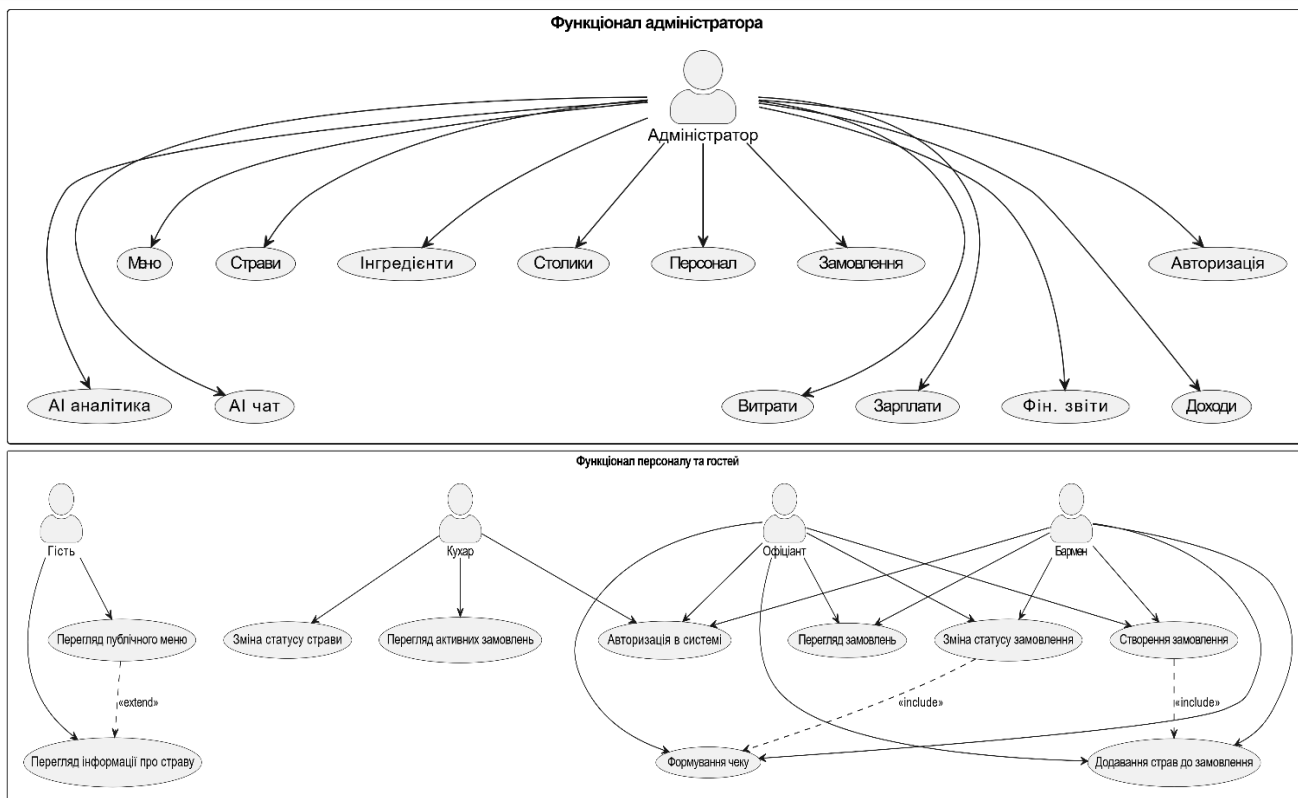


Рис. 2.2. Діаграма варіантів використання вебсистеми управління рестораном

Адміністратор має найширший спектр варіантів використання. Він виконує авторизацію, управляє категоріями меню, стравами, інгредієнтами та столиками. Крім того, адміністратор відповідає за управління персоналом – додавання, редагування та зміну статусів співробітників, перегляд замовлень, ведення обліку витрат та виплат заробітної плати, а також має доступ до фінансових звітів та AI-аналітики прибутковості страв.

Офіціант після авторизації створює нові замовлення з прив'язкою до конкретного столика, додає страви з меню із зазначенням кількості, відстежує зміну статусів готовності та формує чеки після завершення обслуговування.

Кухар переглядає активні замовлення на кухонній панелі та змінює статуси готовності окремих страв – від «В приготуванні» до «Готово», що дозволяє офіціанту своєчасно подати страву гостю.

Бармен має аналогічний до офіціанта функціонал, однак орієнтований виключно на обслуговування клієнтів барної секції: приймає замовлення на напої та барне меню, змінює їх статуси та формує чеки. Розмежування ролей

офіціанта та бармена забезпечує чіткий розподіл зон відповідальності всередині закладу. Матрицю доступу до модулів системи для кожної ролі наведено у таблиці 2.1.

Таблиця 2.1.

Матриця доступу до модулів системи за ролями

Модуль	Guest	Waiter	Bartender	Chef	Admin
Публічне меню (zesty-client)	Так	Так	Так	Так	Так
Перегляд замовлень	-	Так	Так	Так	Так
Створення замовлень	-	Так	Так	-	Так
Зміна статусу страв	-	Так	Так	Так	Так
Управління меню	-	-	-	-	Так
Управління столиками	-	-	-	-	Так
Управління персоналом	-	-	-	-	Так
Фінансові звіти	-	-	-	-	Так
Виплати зарплат	-	-	-	-	Так
Управління витратами	-	-	-	-	Так

Для деталізації взаємодії користувачів із системою наведено опис ключових прецедентів у табличній формі (табл. 2.2.–2.7.).

Таблиця 2.2.

Опис прецеденту «Авторизація»

Елемент	Опис
Назва	Авторизація в системі
Актори	Адміністратор, Офіціант, Бармен, Кухар
Передумови	Користувач має обліковий запис у системі
Основний потік	1. Відкриває сторінку входу. 2. Вводить логін та пароль. 3. Система перевіряє дані (bcrypt.compare). 4. Генерується JWT-токен з роллю. 5. Токен зберігається у localStorage. 6. Перенаправлення на сторінку відповідно до ролі.
Альт. потік	3а. Неправильні дані – повідомлення про помилку.
Постумови	Користувач авторизований, має доступ до функцій своєї ролі.

Таблиця 2.3.

Опис прецеденту «Створення замовлення»

Елемент	Опис
Назва	Створення замовлення
Актори	Офіціант, Бармен, Адміністратор
Передумови	Користувач авторизований; є вільний столик
Основний потік	1. Перехід до розділу замовлень. 2. Вибір вільного столика. 3. Натискання «Створити замовлення». 4. Система створює замовлення зі статусом Open. 5. Додавання страв з меню із зазначенням кількості. 6. Автоматичний розрахунок суми.
Альт. потік	5а. Страва зі статусом Out of Stock – додати неможливо.
Постумови	Замовлення зі статусом Open відображається у списку активних.

Таблиця 2.4.

Опис прецеденту «Зміна статусу страви на кухні»

Елемент	Опис
Назва	Зміна статусу страви
Актори	Кухар
Передумови	Кухар авторизований; є активні замовлення
Основний потік	1. Перегляд кухонної панелі. 2. Вибір страви зі статусом Pending. 3. Зміна на Cooking (розпочато). 4. Після завершення – зміна на Ready. 5. Офіціант бачить оновлений статус.
Альт. потік	3а. Неможливо приготувати – статус Cancelled.
Постумови	Статус оновлено; офіціант повідомлений про готовність.

Таблиця 2.5.

Опис прецеденту «Управління меню та стравами»

Елемент	Опис
Назва	Управління меню та стравами
Актори	Адміністратор
Передумови	Користувач авторизований з відповідною роллю
Основний потік	1. Перехід до розділу меню. 2. Створення або вибір категорії. 3. Додавання страви (назва, опис, ціна, фото, статус). 4. Прив'язка інгредієнтів із кількістю. 5. Збереження – страва з'являється у zesty-client.
Альт. потік	4а. Інгредієнта немає – спочатку додати у розділі складу.
Постумови	Страва доступна в меню та відображається у zesty-client.

Таблиця 2.6.

Опис прецеденту «Формування фінансового звіту»

Елемент	Опис
Назва	Формування фінансового звіту
Актори	Адміністратор
Передумови	Авторизований; є записи доходів та витрат
Основний потік	1. Перехід до розділу звітів. 2. Вибір звітного періоду. 3. Агрегація даних з incomes та expenses. 4. Відображення: дохід, витрати за категоріями, зарплати, чистий прибуток. 5. Візуалізація у вигляді графіків.
Постумови	Фінансова інформація за обраний період отримана.

Таблиця 2.7.

Опис прецеденту «AI-аналітика»

Елемент	Опис
Назва	AI-аналітика прибутковості страв
Актори	Адміністратор
Передумови	Авторизований; є дані про страви та продажі
Основний потік	1. Перехід до розділу AI-аналітики. 2. Розрахунок собівартості та маржинальності страв. 3. Передача даних до Groq API. 4. LLaMA 3.1 генерує 5 рекомендацій у форматі JSON. 5. Відображення карток рекомендацій. 6. Можливість задати питання в AI-чаті.
Альт. потік	4а. Groq API недоступний – повідомлення про помилку.
Постумови	Адміністратор отримав рекомендації щодо оптимізації меню.

Система побудована за принципами об'єктно-орієнтованого програмування з використанням TypeScript [10]. Архітектура передбачає чіткий розподіл відповідальностей між різними типами класів.

Узагальнену діаграму класів вебсистеми управління рестораном наведено на рисунку 2.3. Діаграма відображає чотирнадцять основних класів-сутностей системи та зв'язки між ними. Клас User зберігає облікові дані для автентифікації. Клас Staff містить повну інформацію про працівника – персональні дані, посаду, статус та дані про нарахування заробітної плати. Між User та Staff встановлено асоціацію через поле staffId. Класи Menu та Dish описують структуру меню – Menu є батьківським класом, Dish містить атрибути страви: назву, опис, ціну та

статус. Ingredient описує інгредієнт з одиницею виміру та залишком на складі. DishIngredient є класом-асоціацією між Dish та Ingredient з атрибутом кількості. SemiFinishedProduct описує напівфабрикати власного виробництва. Клас Table зберігає атрибути столика – номер, секцію та місткість. Order пов'язаний з Table та Staff через відповідні поля і містить атрибути статусу та загальної суми. OrderDish є класом-асоціацією між Order та Dish, що зберігає кількість та ціну на момент замовлення. Receipt пов'язаний з Order і фіксує суму та спосіб оплати. Expense та Income описують фінансові операції з атрибутами категорії, суми та дати. SalaryPayment пов'язаний зі Staff і містить атрибути суми, розрахункового періоду та статусу виплати.

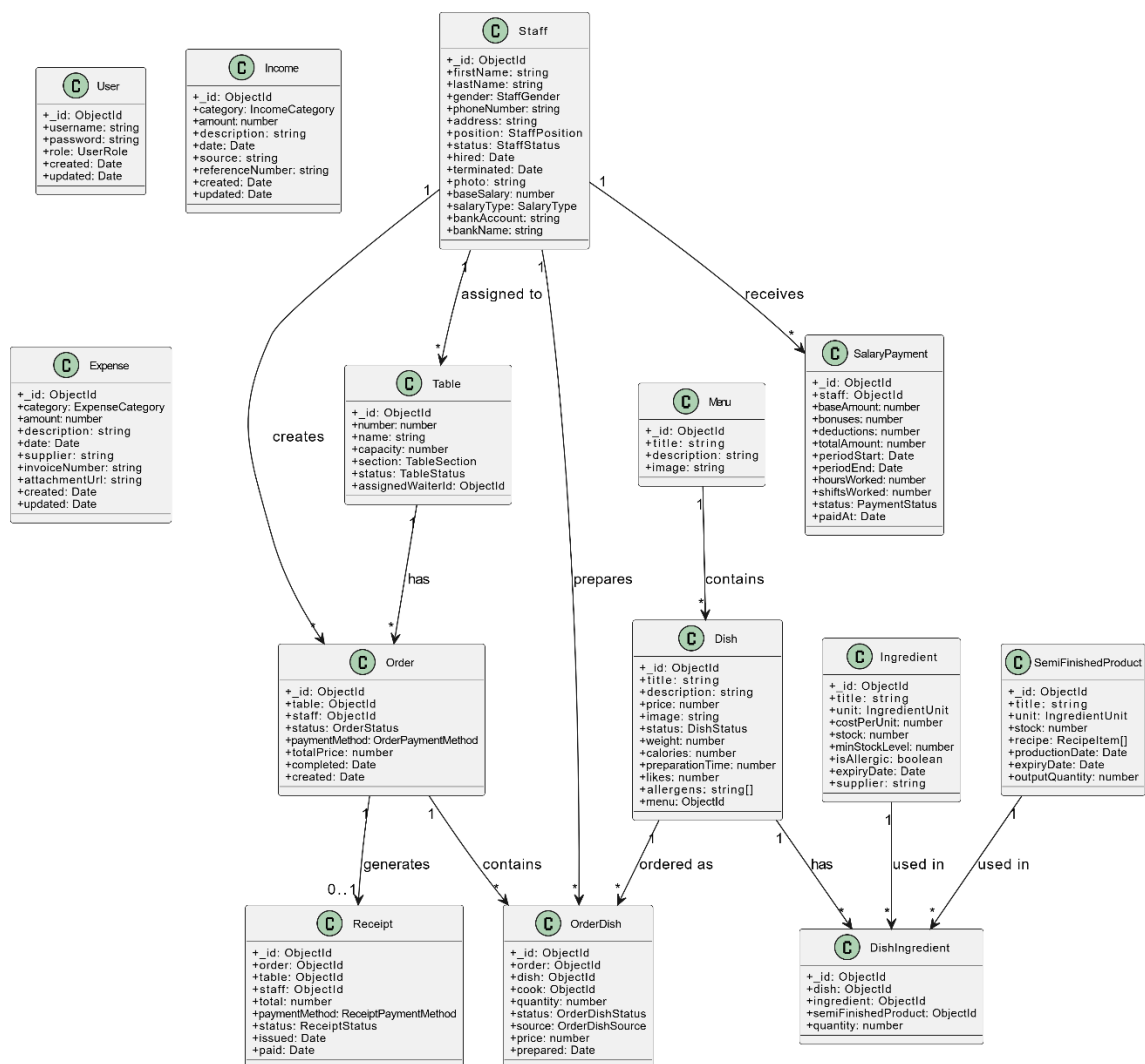


Рис. 2.3. Узагальнена діаграма класів вебсистеми управління рестораном

Класи сервісів інкапсулюють бізнес-логіку. Класи резолверів є точками входу для GraphQL-запитів та делегують виконання відповідним сервісам.

Застосування ООП забезпечує інкапсуляцію (приховання внутрішньої реалізації), модульність (незалежні компоненти), повторне використання коду та розширюваність архітектури без змін базової структури.

2.2 Розробка бази даних системи

Для зберігання даних обрано документоорієнтовану базу даних MongoDB [29]. Вона зберігає дані у форматі BSON, що природно відповідає структурам TypeScript-об'єктів та усуває необхідність складного об'єктно-реляційного відображення. Горизонтальне масштабування та гнучка схема документів дозволяють еволюцію структури без міграцій, що особливо важливо на етапі активної розробки.

Схема бази даних вебсистеми управління рестораном включає чотирнадцять колекцій, згрупованих за функціональними областями.

Перша група – управління користувачами та персоналом – містить колекції users та staffs. Колекція users зберігає облікові дані для автентифікації, колекція staffs – повну інформацію про працівників. Між ними встановлено зв'язок через поле userId у колекції staffs.

Друга група – меню та складський облік – включає колекції menus, dishes, ingredients, semiFinishedProducts та dishIngredients. Колекція menus містить категорії та dishes – страви з посиланням на категорію через поле menuId. Колекція dishIngredients є проміжною між dishes та ingredients і dishes та semiFinishedProducts, зберігаючи кількість кожного інгредієнта чи напівфабрикату у складі страви.

Третя група – обробка замовлень та чеків – об'єднує колекції tables, orders, orderDishes та receipts. Колекція orders посилається на tables та staffs через відповідні поля зовнішніх ключів. Колекція orderDishes є проміжною між orders та dishes і фіксує ціну страви на момент замовлення, щоб зміна цін у

майбутньому не впливала на архівні замовлення. Колекція receipts посилається на orders і формується автоматично при завершенні замовлення.

Четверта група – фінансовий облік – включає колекції expenses, incomes та salaryPayments. Перші дві є незалежними і не мають зовнішніх ключів, оскільки фіксують операційні витрати та доходи закладу без прив'язки до конкретних замовлень. Колекція salaryPayments посилається на staffs і зберігає дані про кожну виплату заробітної плати із зазначенням розрахункового періоду та статусу.

Фізичну схему бази даних вебсистеми управління рестораном наведено на рисунку 2.4.

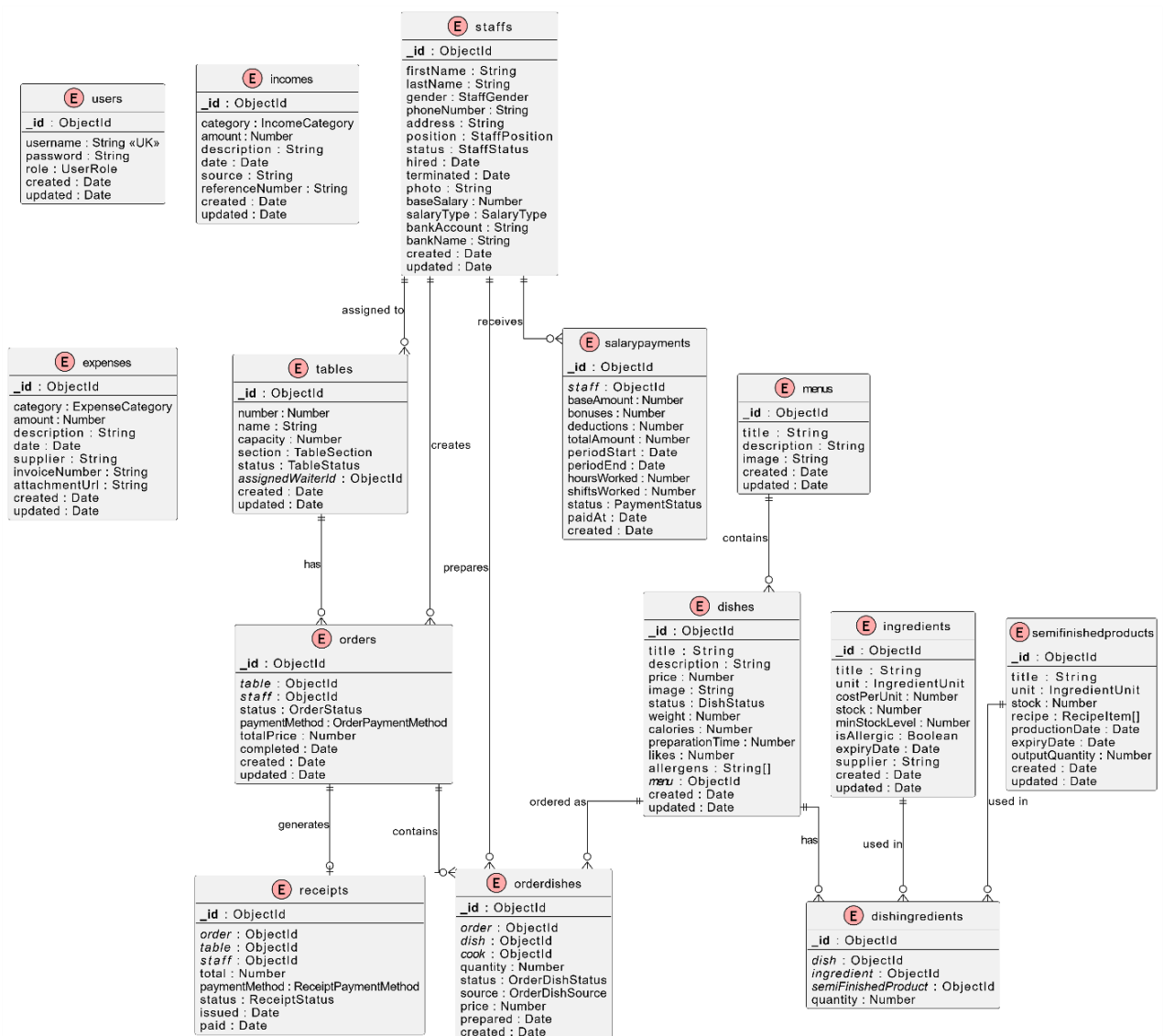


Рис. 2.4. Діаграма бази даних вебсистеми управління рестораном

Детальну структуру основних колекцій бази даних із зазначенням полів, типів даних та обов'язковості наведено у таблицях 2.8.–2.16. Для кожного поля вказано тип даних відповідно до специфіки MongoDB, де ObjectId використовується для унікальних ідентифікаторів та зовнішніх ключів, Enum – для полів з фіксованим набором допустимих значень, а мітки часу (createdAt, updatedAt) заповнюються автоматично завдяки опції timestamps схеми Mongoose. Поля, позначені як необов'язкові, можуть бути відсутніми у документі, бо MongoDB на відміну від реляційних баз даних не вимагає наявності всіх полів у кожному записі, що забезпечує гнучкість при роботі з неповними даними.

Таблиця 2.8.

Структура колекції users

Поле	Тип	Обов'язкове	Опис
_id	ObjectId	Так	Унікальний ідентифікатор
username	String	Так	Логін користувача
password	String	Так	Хеш пароля
createdAt	Date	Так	Дата створення
updatedAt	Date	Так	Дата оновлення

Таблиця 2.9.

Структура колекції staffs

Поле	Тип	Обов'язкове	Опис
_id	ObjectId	Так	Унікальний ідентифікатор
firstName	String	Так	Ім'я працівника
lastName	String	Так	Прізвище працівника
position	Enum	Так	Посада (Cook, Waiter, Admin)
status	Enum	Так	Статус (Active, Out of Office, Suspended)
phone	String	Ні	Номер телефону
hireDate	Date	Так	Дата прийняття на роботу
salaryType	Enum	Так	Тип зарплати (Hourly, Monthly, Shift Based)
baseSalary	Number	Так	Базова ставка

Таблиця 2.10.

Структура колекції dishes

Поле	Тип	Обов'язкове	Опис
_id	ObjectId	Так	Унікальний ідентифікатор
title	String	Так	Назва страви
menu	ObjectId	Так	Посилання на категорію (menus)
description	String	Так	Опис страви
price	Float	Так	Ціна страви
status	Enum	Так	Available / OutOfStock / Seasonal / Archived / ComingSoon / Special / Hidden
photo	String	Ні	URL фото
weight	Int	Ні	Вага в грамах
calories	Int	Ні	Калорійність
preparationTime	Int	Ні	Час приготування (хв)
allergens	[String]	Ні	Перелік алергенів
created	Date	Авто	Дата створення
updated	Date	Авто	Дата оновлення

Таблиця 2.11.

Структура колекції expenses

Поле	Тип	Обов'язкове	Опис
_id	ObjectId	Так	Унікальний ідентифікатор
category	Enum	Так	Rent / Utilities / Ingredients / Marketing / EquipmentMaintenance / Salaries / Other
amount	Float	Так	Сума витрати
description	String	Так	Опис витрати
date	Date	Так	Дата витрати
supplier	String	Ні	Постачальник
invoiceNumber	String	Ні	Номер рахунку-фактури
attachmentUrl	String	Ні	Посилання на скановане підтвердження
created	Date	Авто	Дата створення
updated	Date	Авто	Дата оновлення

Таблиця 2.12.

Структура колекції orders

Поле	Тип	Обов'язкове	Опис
_id	ObjectId	Так	Унікальний ідентифікатор
staff	ObjectId	Так	Посилання на офіціанта (staffs)
table	ObjectId	Так	Посилання на столик (tables)
status	Enum	Так	Open / InProgress / Completed / Cancelled
totalPrice	Float	Ні	Загальна сума замовлення
paymentMethod	Enum	Ні	Спосіб оплати
created	Date	Авто	Дата та час створення
updated	Date	Авто	Дата та час оновлення
completed	Date	Ні	Дата та час завершення

Таблиця 2.13.

Структура колекції incomes

Поле	Тип	Обов'язкове	Опис
_id	ObjectId	Так	Унікальний ідентифікатор
category	Enum	Так	SALES / CATERING / EVENTS / OTHER
amount	Float	Так	Сума доходу
description	String	Так	Опис доходу
date	Date	Так	Дата отримання доходу
source	String	Ні	Джерело доходу
referenceNumber	String	Ні	Референс-номер для звірки
created	Date	Авто	Дата створення
updated	Date	Авто	Дата оновлення

Таблиця 2.14.

Структура колекції menus

Поле	Тип	Обов'язкове	Опис
_id	ObjectId	Так	Унікальний ідентифікатор
title	String	Так	Назва категорії меню
type	String	Ні	Тип меню (основне, барне, сезонне тощо)
created	Date	Так	Дата створення

Таблиця 2.15.

Структура колекції orderDishes

Поле	Тип	Обов'язкове	Опис
_id	ObjectId	Так	Унікальний ідентифікатор
order	ObjectId	Так	Посилання на замовлення (orders)
dish	ObjectId	Так	Посилання на страву (dishes)
staff	ObjectId	Так	Посилання на кухаря (staffs)
status	Enum	Так	Pending / Cooking / Ready / Served / Cancelled
quantity	Int	Ні	Кількість одиниць
price	Float	Ні	Ціна на момент замовлення
created	Date	Авто	Дата створення
updated	Date	Авто	Дата оновлення
prepared	Date	Ні	Дата готовності страви

Таблиця 2.16.

Структура колекції receipts

Поле	Тип	Обов'язкове	Опис
_id	ObjectId	Так	Унікальний ідентифікатор
orderId	ObjectId	Так	Посилання на замовлення (orders)
table	ObjectId	Так	Посилання на столик (tables)
staff	ObjectId	Так	Посилання на офіціанта (staffs)
total	Float	Так	Загальна сума чеку
paymentMethod	Enum	Так	Готівка / Картка / Онлайн
status	Enum	Так	Paid / Unpaid / Cancelled / Refunded
issued	Date	Ні	Дата виставлення чеку
paid	Date	Ні	Дата оплати
created	Date	Авто	Дата створення
updated	Date	Авто	Дата оновлення

У MongoDB зв'язки реалізовано через посилання ObjectId. Страви пов'язані з категоріями за принципом "багато-до-одного". Зв'язок страв з інгредієнтами реалізовано через проміжну колекцію "багато-до-багатьох". Замовлення прив'язані до столиків та офіціантів ("багато-до-одного"). Чеки формуються для

замовлень ("один-до-одного"). Виплати зарплати прив'язані до працівників ("багато-до-одного").

2.3 Проектування та реалізація алгоритмів роботи системи

Процес обробки замовлення охоплює повний цикл від створення до формування чеку та передбачає взаємодію кількох акторів (рис. 2.5.).

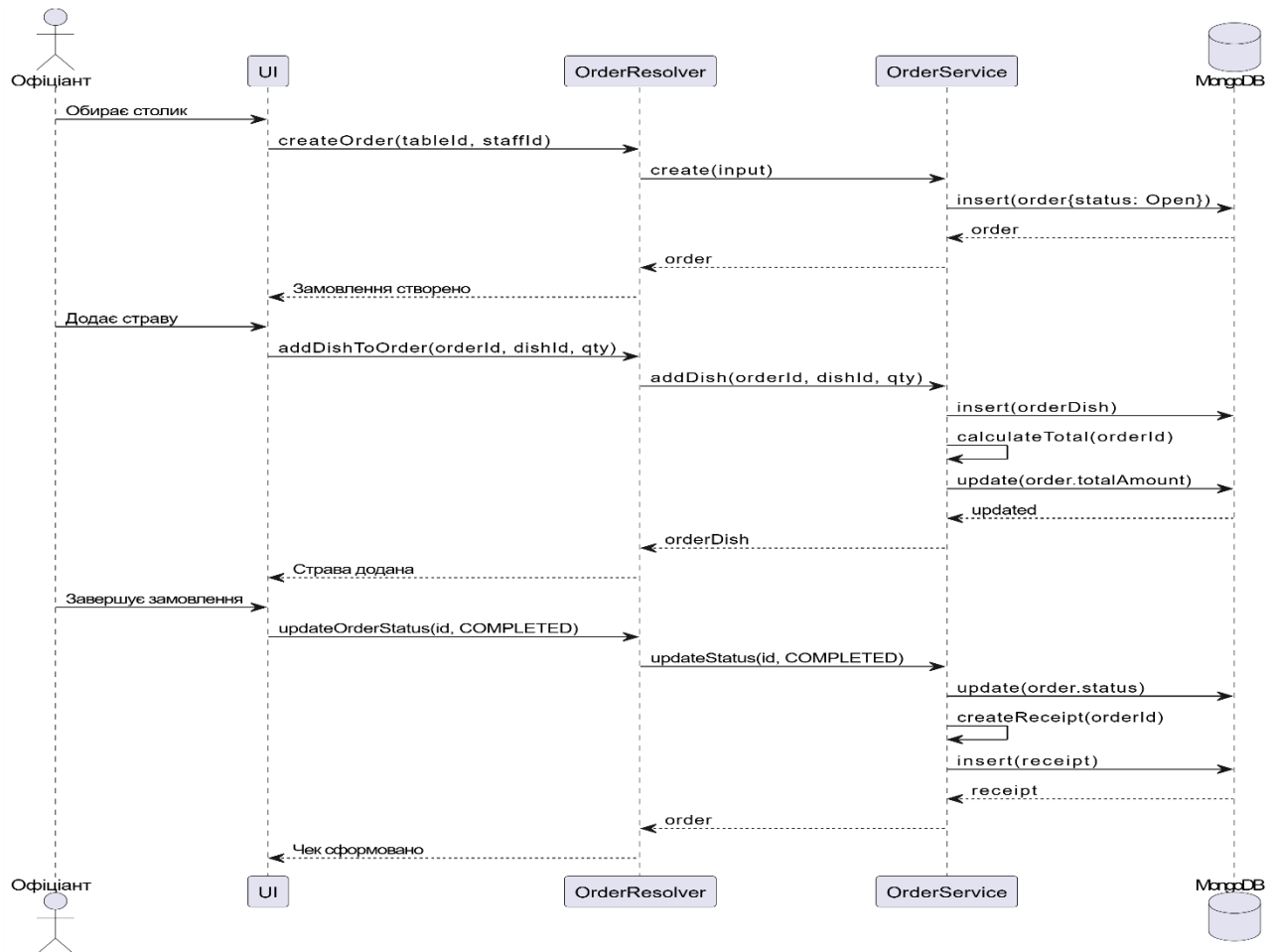


Рис. 2.5. Sequence діаграма процесу обробки замовлення

Офіціант обирає стілик та ініціює замовлення зі статусом "Відкрите". Далі додає страви – система фіксує поточну ціну та перераховує загальну вартість. Статус змінюється: "Відкрите" → "В процесі" → "Завершено". При завершенні автоматично формується чек.

Система використовує JWT для безпечної автентифікації без зберігання сесій на сервері (рис. 2.6.) [12]. На відміну від сесійної автентифікації, де сервер зберігає стан кожного користувача, JWT-підхід є stateless – весь необхідний

контекст (ідентифікатор користувача, роль, час видачі та закінчення дії) міститься безпосередньо у токени та підписується секретним ключем. Це дозволяє серверу перевіряти автентичність запиту без звернення до бази даних при кожному запиті, що позитивно впливає на продуктивність системи. Токен передається у заголовок Authorization кожного GraphQL-запиту, де JwtAuthGuard перевіряє його підпис, а RolesGuard – наявність необхідних прав доступу для виконання конкретної операції [15].

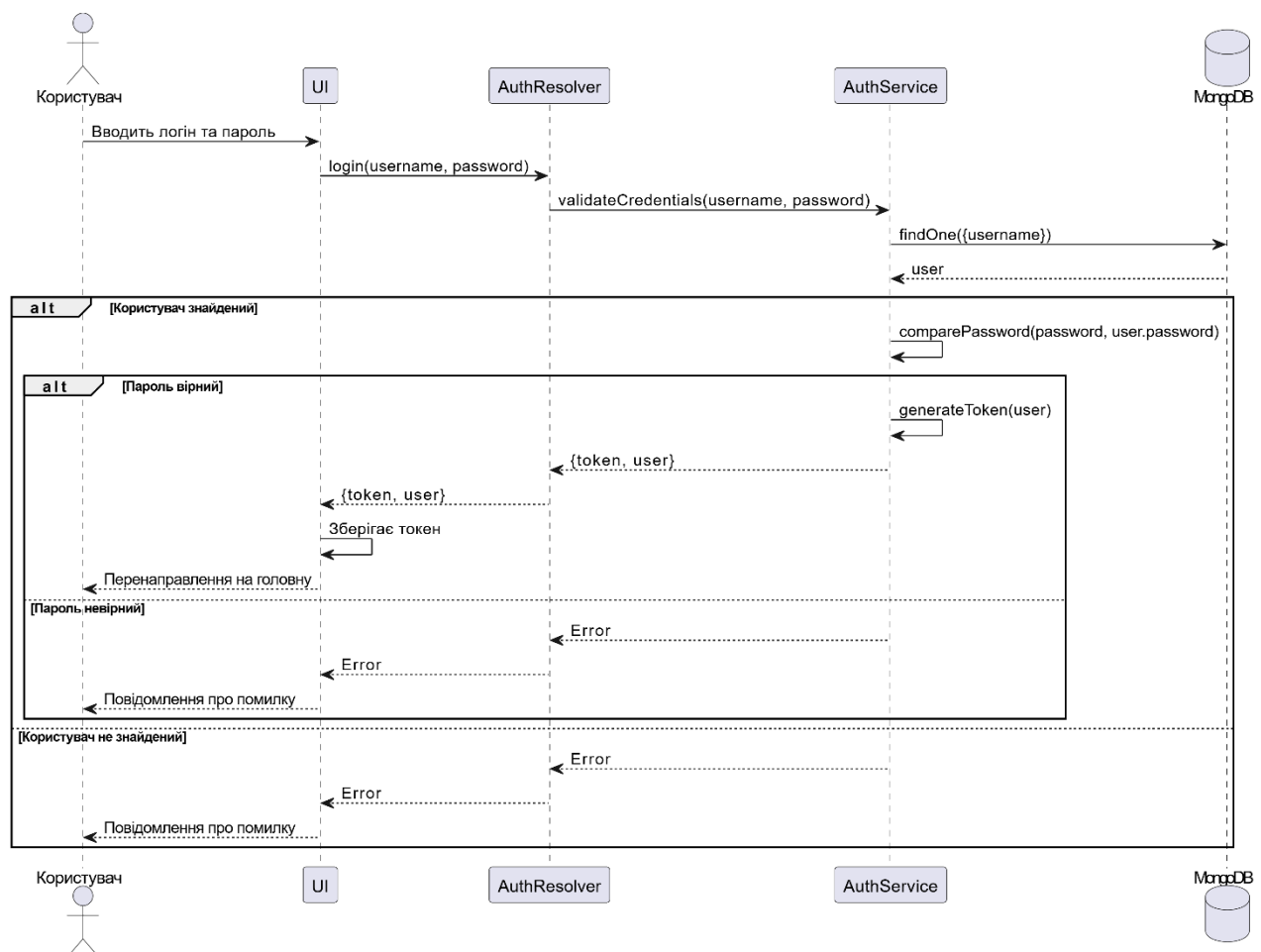


Рис. 2.6. Sequence діаграма процесу авторизації

Користувач надсилає облікові дані. Система знаходить запис у БД, порівнює пароль з хешем через bcrypt. При успіху генерується JWT-токен з ідентифікатором, ім'ям та роллю користувача. Наступні запити містять токен у заголовку Authorization, сервер перевіряє підпис та права доступу.

Архітектура побудована на модульності та слабкій зв'язаності. Основний механізм взаємодії – впровадження залежностей через NestJS. Запити

обробляються за патерном: GraphQL → резолвер → сервіс → MongoDB. Деякі операції ініціюють ланцюгові дії: завершення замовлення автоматично створює чек (рис. 2.7.).

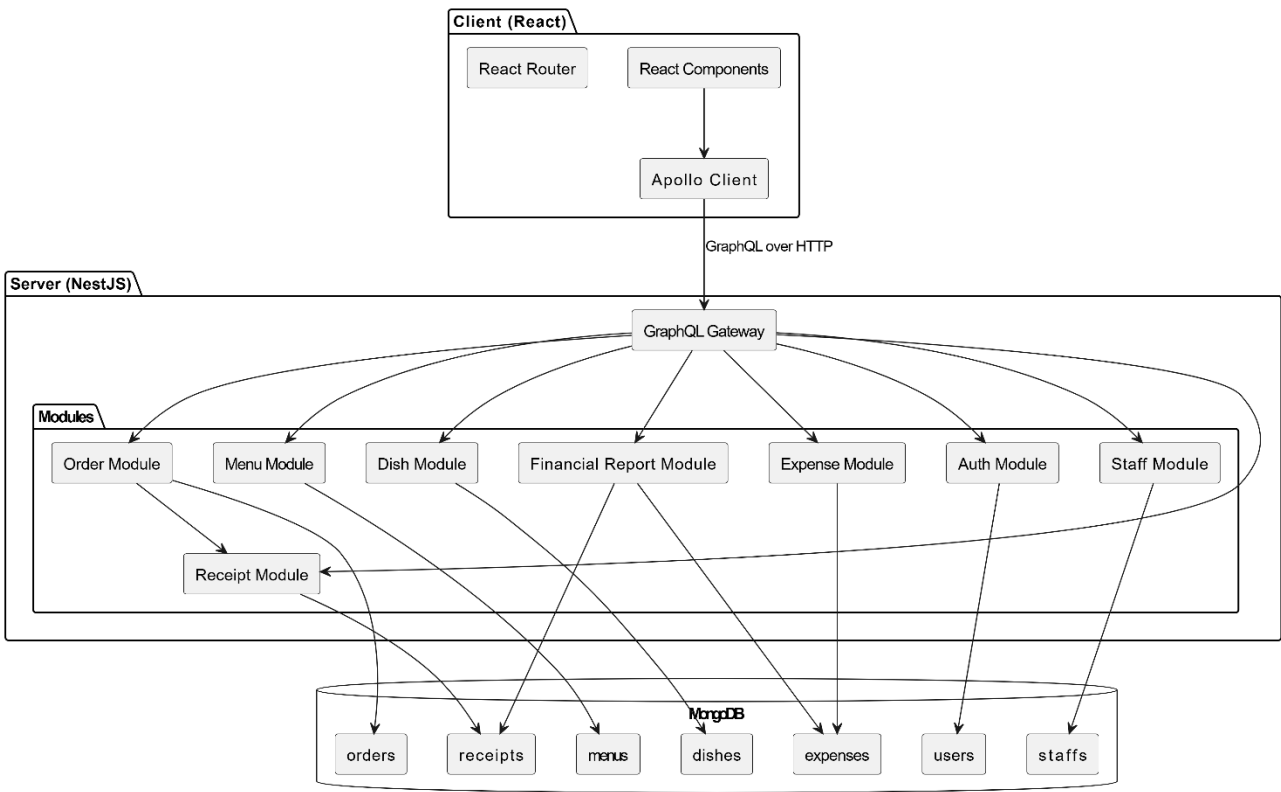


Рис. 2.7. Діаграма взаємодії модулів системи

Модуль штучного інтелекту (AIAalytics) використовує API Groq з моделлю LLaMA 3.1 8B для аналізу бізнес-даних ресторану та генерації рекомендацій у реальному часі. Модуль реалізує розрахунок собівартості страв на основі інгредієнтів, аналіз маржинальності, агрегацію статистики продажів та інтерактивний AI-чат [6, 18].

Для генерації рекомендацій використовується промт наступної структури, зображений на рисунку 2.8. Системна частина задає роль AI-аналітика та вимоги до формату відповіді. Контекстна частина передає агреговані дані про страви: назву, категорію, ціну, собівартість, відсоток маржинальності, кількість продажів та загальний прибуток. Статистична частина передає зведені показники: загальна кількість страв, середня маржинальність, загальний дохід та прибуток, кількість страв з низькою (менше 25%) та високою (більше 50%)

маржею. Таким чином відповідь від AI-аналітика буде на основі реальних даних ресторану. Параметри виклику Groq API: модель llama-3.1-8b-instant, temperature 0.3 (для консистентності відповідей), max_tokens 1500.

Промпт для генерації рекомендацій (generateAIRecommendations)

Ти - AI аналітик ресторану "Zesty". Проаналізуй дані та створи ЗАВЖДИ 5 рекомендацій.

ДАНІ:

- Всього страв: {totalDishes}
- Середня маржа: {averageMarginPercent}%
- Загальний дохід: {totalRevenue} грн
- Загальний прибуток: {totalProfit} грн
- Страв з низькою маржею (<25%): {lowMarginDishesCount}
- Страв з високою маржею (>50%): {highMarginDishesCount}

СТРАВИ:

{dishTitle} | {menuTitle} | Ціна: {price} | Собівартість: {cost} |
 Маржа: {marginPercent}% | Продажів: {salesCount} |
 Прибуток: {totalProfit} грн
 ...

ОБОВ'ЯЗКОВІ 5 РЕКОМЕНДАЦІЙ (завжди в такому порядку):

1. type: "success" - ТОП-3 найприбутковіші страви (назви + прибуток в грн)
2. type: "warning" - Страви з низькою маржею <30% (назви + % маржі), або "Всі страви мають хорошу маржу" якщо таких немає
3. type: "info" - Найпопулярніша категорія (назва + кількість продажів)
4. type: "warning" - Непопулярні страви з продажами <10 (назви), або "Всі страви продаються добре" якщо таких немає
5. type: "info" - Загальна оцінка та одна конкретна порада для покращення

Формат JSON:

```
[{"type":"success|warning|info", "title":"заголовок до 40 символів", "description":"опис з цифрами", "suggestedAction":"коротка порада"}]
```

Відповідай ТІЛЬКИ валідним JSON масивом:

Параметри: temperature: 0.3, max_tokens: 1500

Рис. 2.8. Приклад промта генерації рекомендацій

Модуль AI-аналітики реалізує два сценарії взаємодії з мовною моделлю: генерацію структурованих рекомендацій та інтерактивний чат. В обох випадках алгоритм починається з агрегації даних про страви – розрахунку собівартості на основі інгредієнтів, визначення маржинальності та збору статистики продажів з колекції orderDishes. Сформовані дані передаються до Groq API у вигляді структурованого промту разом із системною інструкцією, що визначає роль та формат відповіді моделі. Для AI-чату використовується окремий промт, що додає до контексту поточне запитання користувача: системна частина задає роль

асистента ресторану з вимогою відповідати українською мовою виключно на основі наданих даних. В обох сценаріях відповідь моделі обробляється на сервері та повертається клієнту у структурованому вигляді. Блок-схему алгоритму наведено на рисунку 2.9.

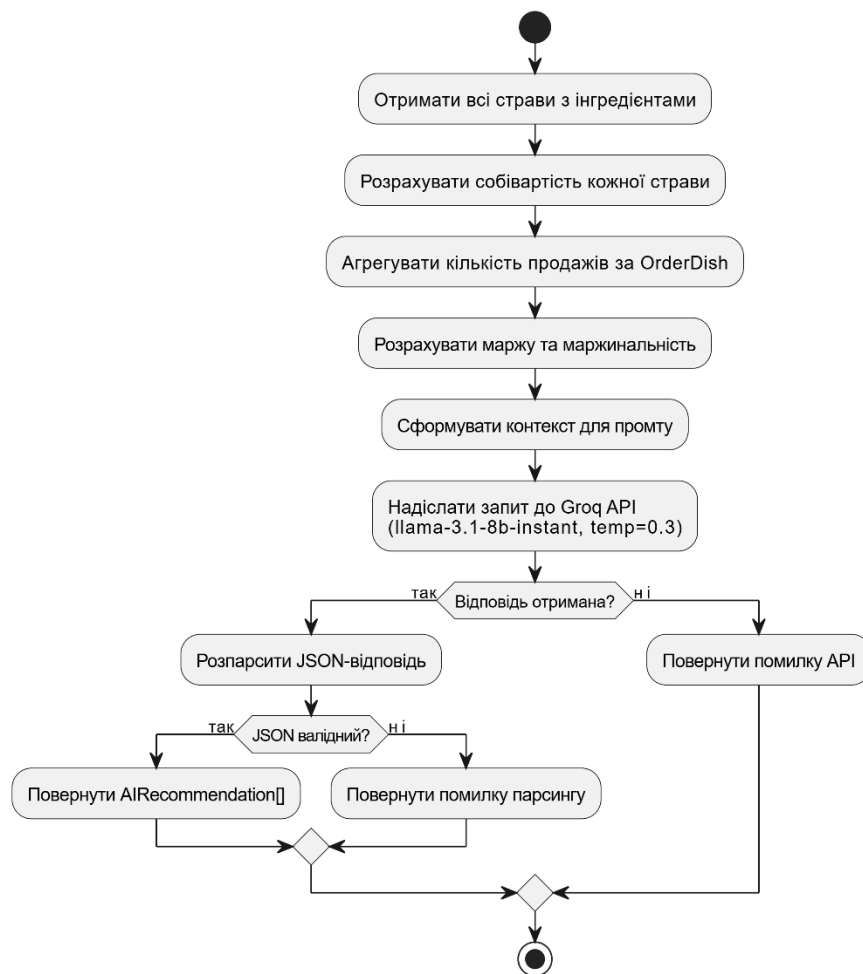


Рис. 2.9. Блок-схема алгоритму генерації AI-рекомендацій

2.4 Реалізація вебсистеми управління рестораном

Серверна частина реалізована на NestJS з модульною архітектурою. Модуль автентифікації містить сервіс перевірки облікових даних, стратегію JWT та резолвер. Модуль персоналу забезпечує CRUD-операції над працівниками з фільтрацією за посадою та статусом. Модулі меню та страв управляють асортиментом. Модуль замовлень реалізує повний цикл обробки. Модуль фінансової звітності агрегує дані з колекцій incomes та expenses для формування

комплексних звітів. Чистий прибуток розраховується за формулою: чистий прибуток = доходи (incomes) – витрати (expenses), де витрати включають операційні витрати та виплати заробітної плати. Модуль Income забезпечує облік усіх видів доходів закладу, включаючи продажі, кейтеринг та організацію заходів.

Клієнтська частина побудована на React з функціональними компонентами та хуками [24]. Apollo Client управляє GraphQL-запитами та кешуванням [19]. Маршрутизація розділена на публічні (сторінка авторизації) та захищені маршрути. Типи генеруються автоматично на основі GraphQL-схеми сервера. На рисунку 2.10. зображено діаграму компонентів клієнтської частини системи, як вони взаємодіють, та як пов'язані між собою.

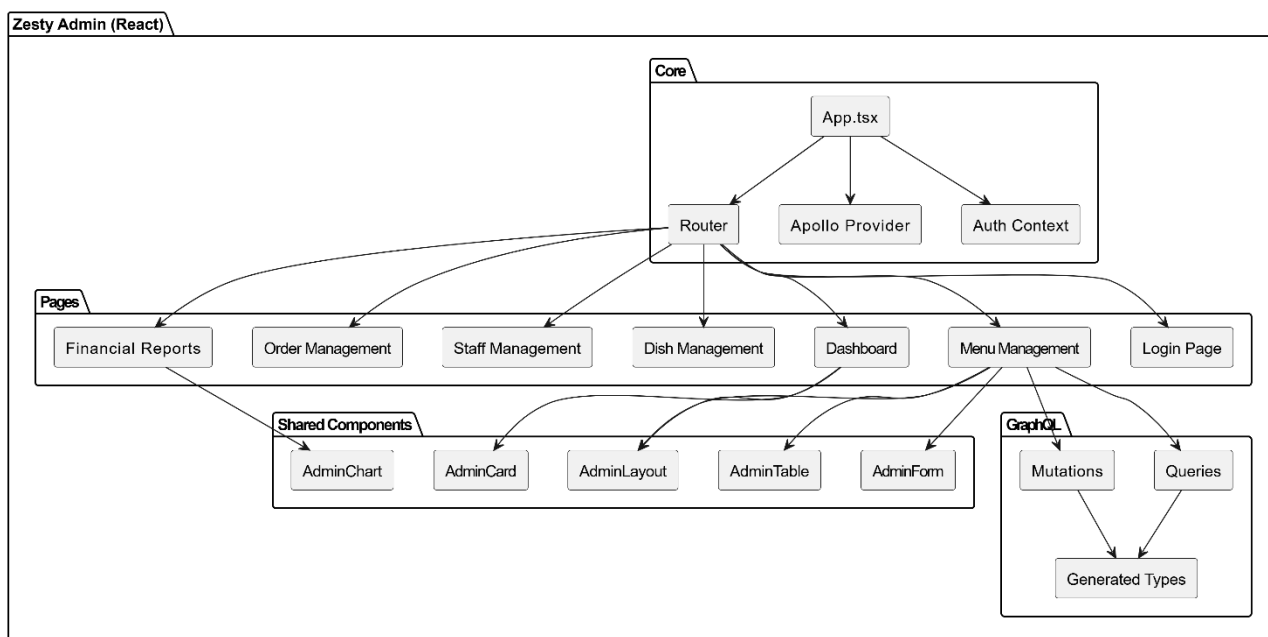


Рис. 2.10. Діаграма компонентів клієнтської частини системи

Взаємодія між частинами здійснюється через GraphQL API. Запити отримують дані без модифікації – наприклад, список замовлень з деталями столика та офіціанта. Мутації змінюють дані – при створенні замовлення клієнт надсилає ідентифікатори столика та офіціанта. Apollo Client автоматично оновлює кеш після мутацій.

Інтерфейс побудований на Material UI. Базовий компонент AdminLayout визначає структуру з навігаційною панеллю та адаптується до розмірів екрану.

AdminTable відображає списки з сортуванням, пагінацією та діями. AdminForm забезпечує створення та редагування у модальних вікнах з валідацією. Компоненти візуалізації відображають статистику у вигляді карток та графіків.

Для автоматизації процесів тестування та доставки програмного забезпечення налаштовано pipeline на базі GitHub Actions та платформи Vercel. Система складається з трьох репозиторіїв – zesty-service, zesty-admin та zesty-client – кожен з яких має окремий проєкт на Vercel з налаштованими середовищами dev та prod. На рисунку 2.11. зображено схему CI/CD pipeline вебсистеми управління рестораном. При створенні pull request автоматично запускається CI – лінтинг, тести та збірка. Після успішного merge до dev або публікації GitHub Release відповідний застосунок автоматично розгортається на dev або prod інстанс Vercel.

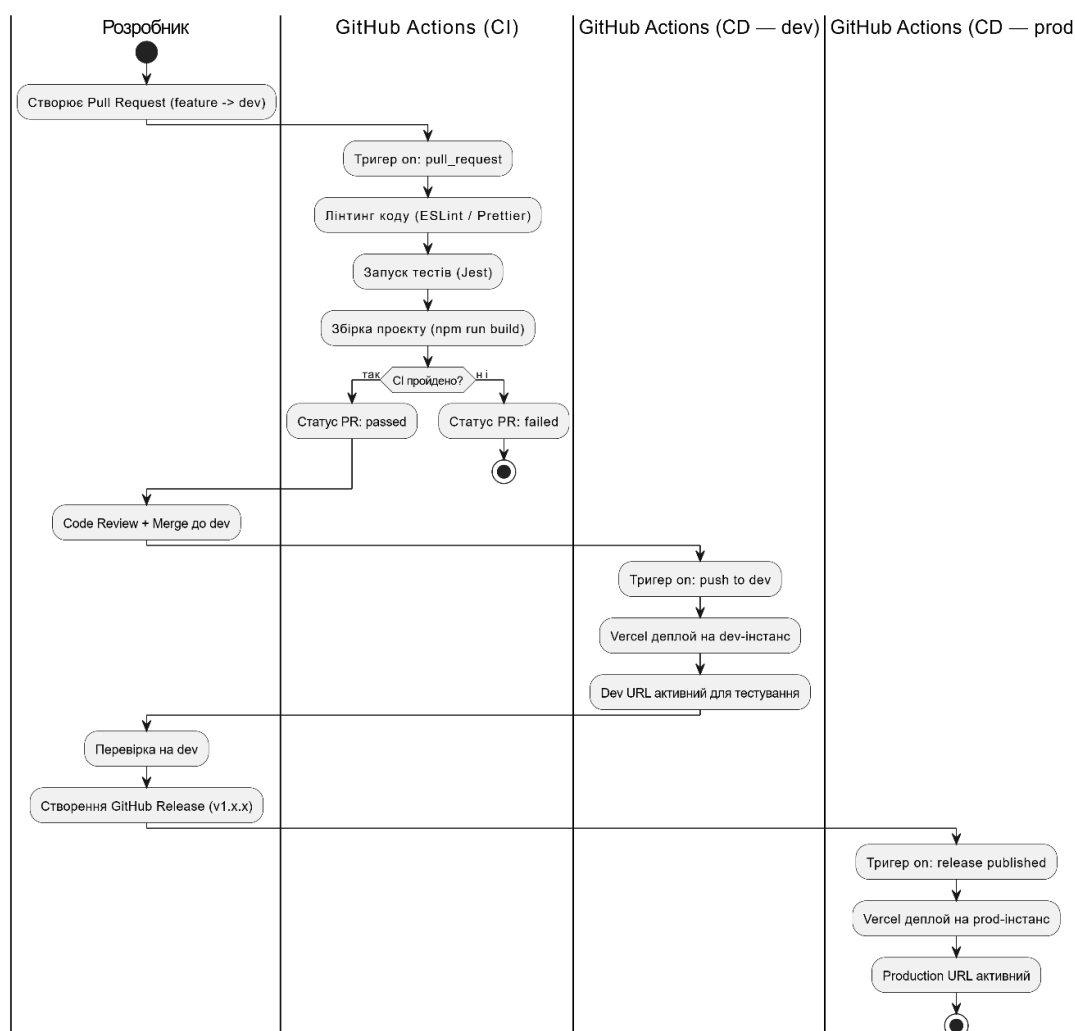


Рис. 2.11. Схема CI/CD pipeline вебсистеми управління рестораном

Висновки до другого розділу

У другому розділі проведено комплексне проектування вебсистеми управління рестораном. Визначено чотирьох основних акторів системи (адміністратор, офіціант, бармен, кухар) та побудовано діаграму варіантів використання з розмежуванням функцій за ролями. Для ключових прецедентів складено детальні описи сценаріїв взаємодії у табличній формі, що охоплюють авторизацію, створення замовлення, зміну статусу страви, управління меню, формування фінансових звітів та AI-аналітику.

Спроектовано об'єктно-орієнтовану структуру системи з визначенням класів сутностей та сервісів. Розроблено структуру бази даних MongoDB з чотирнадцятьма колекціями, для кожної з яких визначено поля, типи даних та обов'язковість. Описано організацію зв'язків між колекціями через посилання ObjectId.

Спроектовано алгоритми обробки замовлень з діаграмою станів позиції замовлення (OrderDish), JWT-авторизації з перевіркою ролей, формування фінансових звітів та роботи модуля AI-аналітики з описом промтів для генерації рекомендацій та AI-чату. Описано архітектуру серверного застосунку на NestJS, обох клієнтських застосунків на React.js, налаштований CI/CD pipeline на базі GitHub Actions з автоматичним розгортанням на Vercel та принципи взаємодії компонентів через GraphQL API.

РОЗДІЛ 3. ОПИС РОБОТИ ТА ТЕСТУВАННЯ ВЕБСИСТЕМИ УПРАВЛІННЯ РЕСТОРАНОМ

3.1 Порядок встановлення та налаштування параметрів платформи

Вебсистема управління рестораном складається з трьох незалежних застосунків, кожен з яких розгортається окремо на платформі Vercel. Загальну схему розгортання компонентів системи наведено на рисунку 3.1.

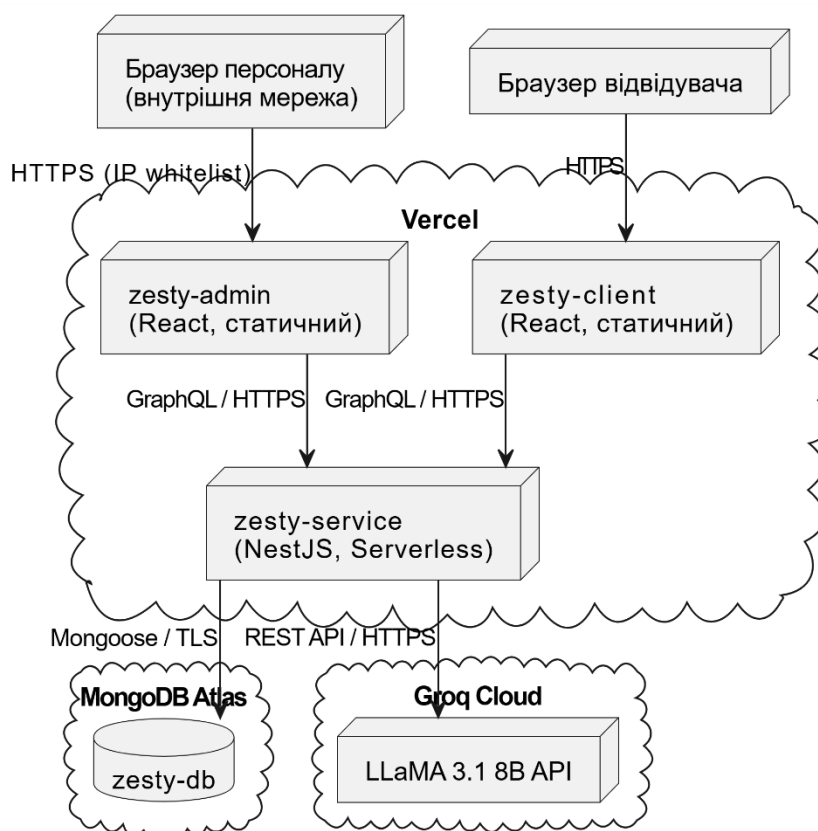


Рис. 3.1. Діаграма розгортання вебсистеми управління рестораном

Розгортання системи передбачає налаштування трьох компонентів на Vercel та підключення до хмарних сервісів MongoDB Atlas та Groq.

zesty-service – серверна частина на базі NestJS, розгорнута як serverless-функція на Vercel. Відповідає за обробку GraphQL-запитів, бізнес-логіку та взаємодію з базою даних і зовнішніми API. Для розгортання необхідно налаштувати змінні середовища: рядок підключення до MongoDB Atlas, секретний ключ JWT та API-ключ Groq.

zesty-admin – адміністративний інтерфейс на базі React.js для внутрішнього використання персоналом закладу. Розгортається як статичний застосунок на Vercel. Доступ обмежено на мережевому рівні через IP-whitelisting, лише пристрої з дозволених IP-адрес можуть відкрити інтерфейс. Для розгортання необхідно вказати URL серверної частини (zesty-service).

zesty-client – публічний інтерфейс для відвідувачів ресторану на базі React.js. Розгортається як статичний застосунок на Vercel без обмежень за IP-адресою. Дозволяє переглядати меню та інформацію про страви без автентифікації. Для розгортання необхідно вказати URL серверної частини.

MongoDB Atlas – хмарна база даних як сервіс. Не потребує локального встановлення – достатньо створити кластер на atlas.mongodb.com, отримати рядок підключення та передати його як змінну середовища до zesty-service. Рекомендується налаштувати IP Access List для обмеження доступу до БД.

Groq API – зовнішній API для роботи з мовною моделлю LLaMA 3.1. Для підключення необхідно зареєструватися на console.groq.com, отримати API-ключ та передати його як змінну середовища до zesty-service.

Система реалізує рольову модель доступу (Role-Based Access Control, RBAC). Права доступу визначаються на рівні кожного GraphQL-запиту за допомогою декораторів @Roles та RolesGuard у серверній частині. Після авторизації роль користувача зберігається у JWT-токені та перевіряється при кожному запиті без звернення до бази даних.

3.2 Опис роботи з додатком для персоналу Zesty Admin

Додаток для персоналу Zesty Admin є основним інструментом для внутрішнього використання персоналом закладу. Застосунок реалізує повний цикл управління рестораном – від формування меню до фінансової звітності, та надає різним категоріям персоналу доступ до відповідних функцій відповідно до їхньої ролі. Інтерфейс побудований на основі бібліотеки Material UI та організований у вигляді єдиної сторінки (SPA). Застосунок адаптований для роботи на десктопних пристроях з роздільною здатністю від 1280×720 пікселів.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						55

Для початку роботи користувач проходить авторизацію. Неавторизований користувач автоматично перенаправляється на сторінку входу (рис. 3.2.). Сторінка містить поля для логіна та пароля. При невірних даних система відображає повідомлення про помилку. Після успішної авторизації користувач перенаправляється на головну сторінку відповідно до ролі.

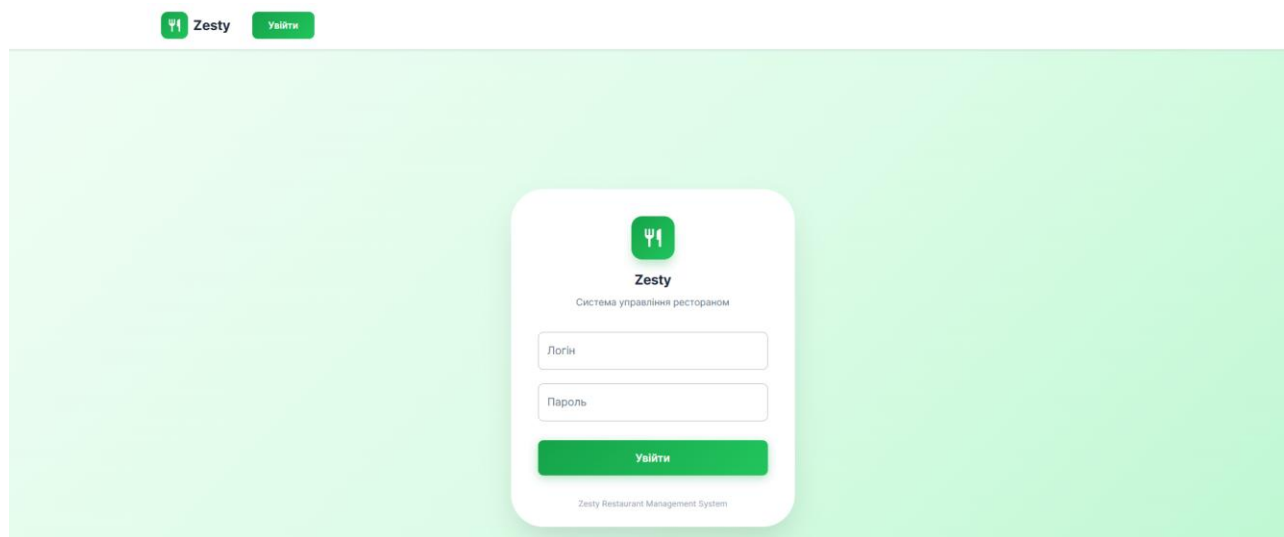


Рис. 3.2. Сторінка авторизації в системі Zesty Admin

Адміністратор має повний доступ до всіх функцій системи через бічну навігаційну панель. Розділ «Меню» дозволяє переглядати, створювати, редагувати та видаляти категорії страв (рис. 3.3.).

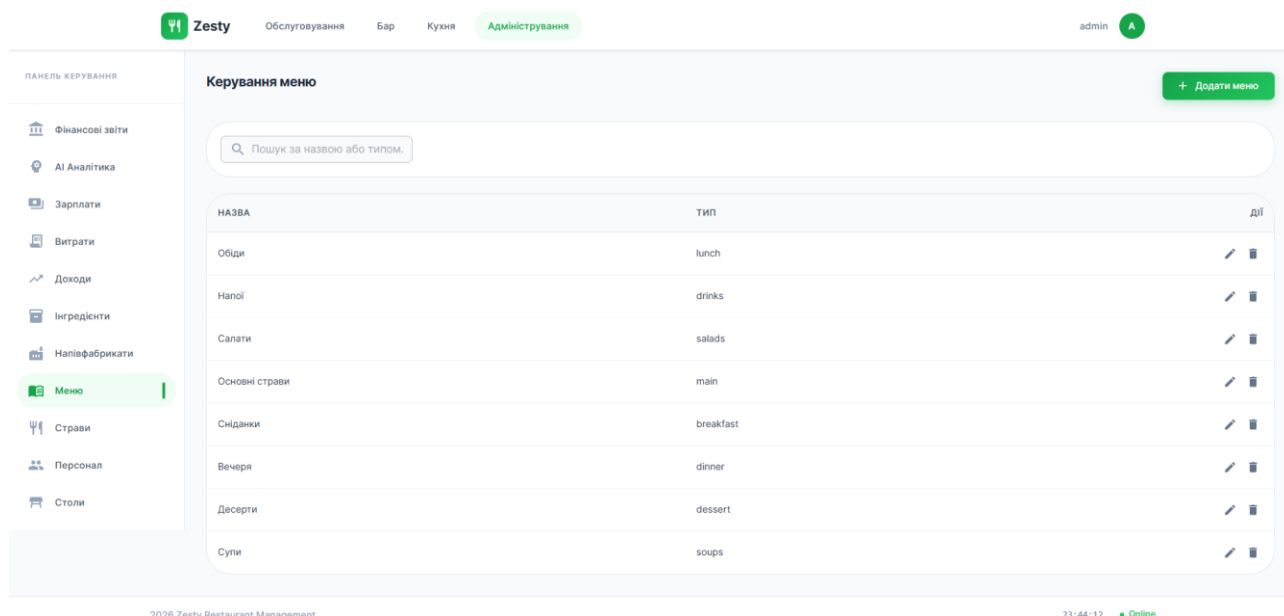


Рис. 3.3. Сторінка управління категоріями меню

📊

Фінансові звіти

📊

AI Аналітика

💰

Зарплати

📊

Витрати

📈

Доходи

🍳

Інгредієнти

🧴

Напівфабрикати

📖

Меню

🍴

Страви

👥

Персонал

🪴

Столи

Зesty

Обслуговування

Бар

Кухня

Адміністрування

admin

Керування стравами

Пошук за назвою або описом

МенюВсі меню

СтатусВсі статуси

Додати страву

НАЗВА	МЕНЮ	ЦІНА	СТАТУС	ДІЇ
Ясця з беконом	Сніданки	145.00 грн	Доступно	✎ 🗑
Омлет з овочами	Сніданки	125.00 грн	Доступно	✎ 🗑
Сирники з джемом	Сніданки	135.00 грн	Доступно	✎ 🗑
Салат з тунцем	Салати	215.00 грн	Доступно	✎ 🗑
Крем-суп грибний	Супи	135.00 грн	Доступно	✎ 🗑
Стейк Рібай	Основні страви	485.00 грн	Доступно	✎ 🗑
Паста Карбонара	Основні страви	195.00 грн	Доступно	✎ 🗑
Тирамісу	Десерти	145.00 грн	Доступно	✎ 🗑
Чізкейк Нью-Йорк	Десерти	155.00 грн	Доступно	✎ 🗑

При створенні або редагуванні страви адміністратор може додати склад – перелік інгредієнтів та напівфабрикатів із зазначенням кількості, що використовується для розрахунку собівартості та контролю наявності на складі. Статус страви (доступна, немає в наявності, сезонна, архівна тощо) дозволяє гнучко керувати відображенням позицій у клієнтському додатку (рис. 3.5.).

ПАНЕЛЬ КЕРУВАННЯ

Фінансові звіти

AI Аналітика

Зарплати

Витрати

Доходи

Інгредієнти

Напівфабрикати

Меню

Страви

Персонал

Столи

Керування стравами

Пошук за назвою...

Назва

Ясчня з беконом

Омлет з овочами

Сирники з джемом

Салат з тунцем

Крем-суп грибний

Стейк Рйбай

Паста Карбонара

Тирамісу

Чизкейк Нью-Йорк

Бар

Кухня

Адміністрування

admin

+ Додати страву

Додати страву

Назва страви *

Меню *

Обіди

Опис *

URL фото

Ціна (грн) *

0

Статус *

Доступна

Склад страви (Інгредієнти/напівфабрикати)

+ Додати компонент

Інгредієнт

Напівфабрикат

Інгредієнт

Напівфабрикат

Маринад для шашлику (л)

Сіль (кг)

Кількість

0

Кількість

0

Скасувати

Створити

					ПЗ.КР.Б-121-26-ПЗ	Арк.
						57

Розділ «Інгредієнти» веде облік запасів з відображенням назви, одиниці виміру, залишку та вартості. Система виділяє інгредієнти нижче мінімального рівня та ті, в яких скоро сплине термін придатності (рис. 3.6.).

НАЗВА	ЗАЛИШОК	МІН. РІВЕНЬ	СТАТУС	ЦІНА ЗА ОД.	АЛЕРГЕН	ТЕРМІН	ДІЇ
Бекон	18.3 кг	5.0 кг	В наявності	280.00 грн	-	39 дн	👁️ ✎️ 🗑️
Молоко	64.9 л	20.0 л	В наявності	32.00 грн	Так	3 дн	👁️ ✎️ 🗑️
Пармезан	5.3 кг	2.0 кг	В наявності	650.00 грн	Так	36 дн	👁️ ✎️ 🗑️
Авокадо	21.0 шт	10.0 шт	В наявності	45.00 грн	-	Прострочено	👁️ ✎️ 🗑️
Лимон	12.0 кг	4.0 кг	В наявності	55.00 грн	-	47 дн	👁️ ✎️ 🗑️
Яблука	15.3 кг	5.0 кг	В наявності	35.00 грн	-	37 дн	👁️ ✎️ 🗑️

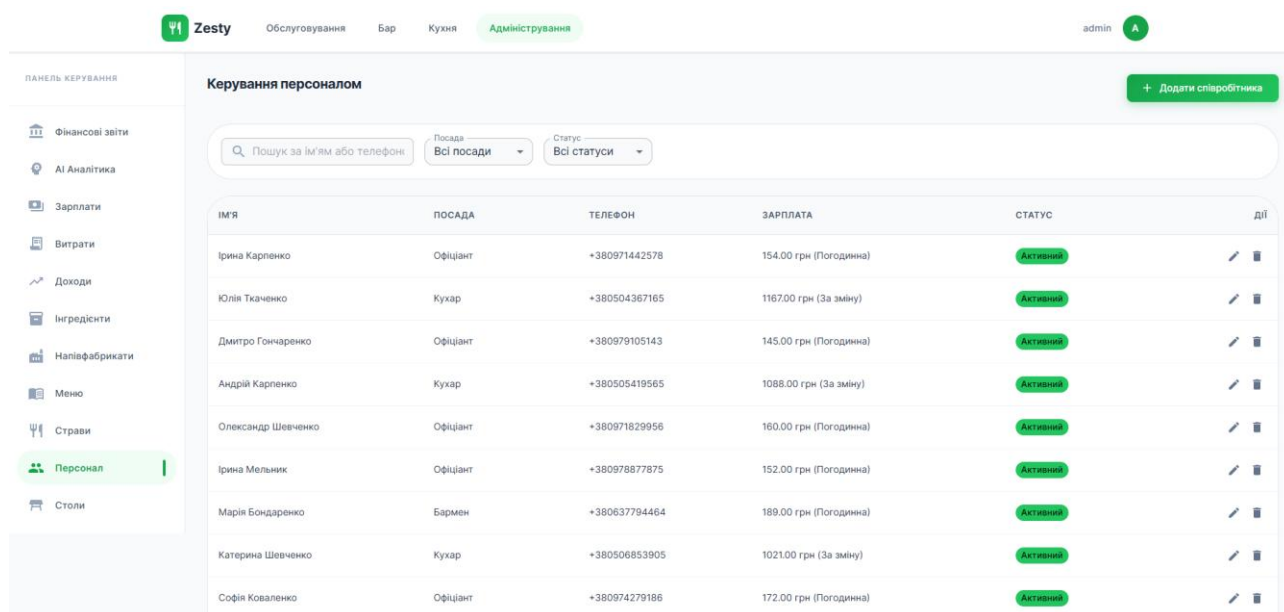
Рис. 3.6. Сторінка управління інгредієнтами

Розділ «Напівфабрикати» веде облік заготовок власного виробництва з зазначенням рецептури, кількості на складі, дати виробництва та терміну придатності (рис. 3.7.). Кожен напівфабрикат може входити до складу однієї або кількох страв меню, що дозволяє точніше розраховувати собівартість та контролювати витрати інгредієнтів при приготуванні.

НАЗВА	ЗАЛИШОК	ВИХІД	ІНГРЕДІЄНТІВ	СТАТУС	ТЕРМІН	ДІЇ
Заправка Цезар	17.7 л	0.5 л	5	В наявності	6 дн	👁️ ✎️ 🗑️
Пісочне тісто	14.8 кг	1 кг	5	В наявності	6 дн	👁️ ✎️ 🗑️
Томатний соус	19.1 л	1 л	6	В наявності	6 дн	👁️ ✎️ 🗑️
Соус BBQ	20.0 л	1 л	5	В наявності	8 дн	👁️ ✎️ 🗑️
Курячий бульйон (концентрат)	9.9 л	2 л	5	Мало	9 дн	👁️ ✎️ 🗑️
Грибний соус	10.5 л	0.5 л	5	В наявності	2 дн	👁️ ✎️ 🗑️

Рис. 3.7. Сторінка управління напівфабрикатами

Розділ «Персонал» відображає список працівників з фільтрацією за посадою та статусом, а також пошуком за ім'ям чи прізвищем. Для кожного працівника зберігається персональна інформація: ім'я, прізвище, стать, контактний номер телефону, адреса проживання та фотографія (рис. 3.8.). Система підтримує чотири посади: адміністратор, офіціант, бармен та кухар, кожна з яких має відповідні права доступу в системі. Статус працівника може бути активним, у відпустці, призупиненим або звільненим, що дозволяє вести історію зайнятості. Для нарахування заробітної плати система підтримує три типи: погодинну, місячну та за зміну. Додатково вказуються банківські реквізити для виплат – номер рахунку та назва банку. Дати прийняття на роботу та звільнення фіксуються автоматично, забезпечуючи повний облік кадрових змін.



Zesty Обслуговування Бар Кухня Адміністрування admin					
ПАНЕЛЬ КЕРУВАННЯ Фінансові звіти AI Аналітика Зарплати Витрати Доходи Інгредієнти Напівафабрикати Меню Страви Персонал Столи Керування персоналом Пошук за ім'ям або телефоном Посада: Всі посади Статус: Всі статуси + Додати співробітника					
ІМ'Я	ПОСАДА	ТЕЛЕФОН	ЗАРПЛАТА	СТАТУС	ДІЇ
Ірина Карпенко	Офіціант	+380971442578	154.00 грн (Погодинна)	Активний	✎ 🗑
Юлія Ткаченко	Кухар	+380504367165	1167.00 грн (За зміну)	Активний	✎ 🗑
Дмитро Гончаренко	Офіціант	+380979105143	145.00 грн (Погодинна)	Активний	✎ 🗑
Андрій Карпенко	Кухар	+380505419565	1088.00 грн (За зміну)	Активний	✎ 🗑
Олександр Шевченко	Офіціант	+380971829956	160.00 грн (Погодинна)	Активний	✎ 🗑
Ірина Мельник	Офіціант	+380978877875	152.00 грн (Погодинна)	Активний	✎ 🗑
Марія Бондаренко	Бармен	+380637794464	189.00 грн (Погодинна)	Активний	✎ 🗑
Катерина Шевченко	Кухар	+380506853905	1021.00 грн (За зміну)	Активний	✎ 🗑
Софія Коваленко	Офіціант	+380974279186	172.00 грн (Погодинна)	Активний	✎ 🗑

Рис. 3.8. Сторінка управління персоналом

Розділ «Столи» дозволяє налаштувати схему залу ресторану та керувати розташуванням посадкових місць (рис. 3.9.). Столики автоматично сортуються за номером для зручної навігації. Кожен столик має номер, назву, місткість (кількість осіб) та належить до однієї з секцій: основний зал, тераса, VIP-зона або барна секція. Статус столика відображає його поточний стан – вільний, зарезервований, зайнятий або недоступний (наприклад, на обслуговуванні). Система дозволяє призначати офіціанта або бармена до конкретних столиків, що

забезпечує чіткий розподіл зон відповідальності між персоналом та оптимізує обслуговування гостей.

НОМЕР	НАЗВА	СЕКЦІЯ	МІСТКІСТЬ	СТАТУС	ДІЇ
1	Стіл 1	Головний зал	4	Вільний	
2	Стіл 2	Головний зал	4	Вільний	
3	Стіл 3	Головний зал	4	Вільний	
4	Стіл 4	Головний зал	2	Вільний	
5	Стіл 5	Головний зал	4	Вільний	
6	Стіл 6	Головний зал	4	Вільний	
7	Стіл 7	Головний зал	4	Вільний	
8	Стіл 8	Головний зал	2	Вільний	
9	Тераса 1	Тераса	6	Вільний	

Рис. 3.9. Сторінка управління столиками

Розділи «Витрати» та «Дохід» ведуть облік за категоріями, забезпечуючи повну картину фінансової діяльності закладу. Витрати класифікуються за типами: оренда, комунальні послуги, інгредієнти, маркетинг, обслуговування обладнання, зарплати та інше (рис.3.10.).

ДАТА	КАТЕГОРІЯ	ОПИС	СУМА	ПОСТАЧАЛЬНИК	ДІЇ
21.05.2026	Продукти	Закупівля продуктів	3296.00 грн	ФОП Мельник	
20.05.2026	Продукти	Закупівля продуктів	8771.00 грн	ПП "Овочі та фрукти"	
18.05.2026	Інше	Уніформа персоналу	3919.00 грн	ПП "Овочі та фрукти"	
17.05.2026	Інше	Дрібний ремонт	4099.00 грн	ТОВ "Молочна ферма"	
17.05.2026	Продукти	Закупівля продуктів	2526.00 грн	ПП "М'ясний двір"	
15.05.2026	Маркетинг	Спонсорство закладу	13677.00 грн	ТОВ "Реклама Плюс"	
15.05.2026	Продукти	Закупівля продуктів	7656.00 грн	ТОВ "Бейкер Хаус"	

Рис. 3.10. Сторінка обліку витрат

Доходи поділяються на категорії: продажі, кейтеринг, заходи та інше, що дозволяє аналізувати джерела надходжень (рис. 3.11.).

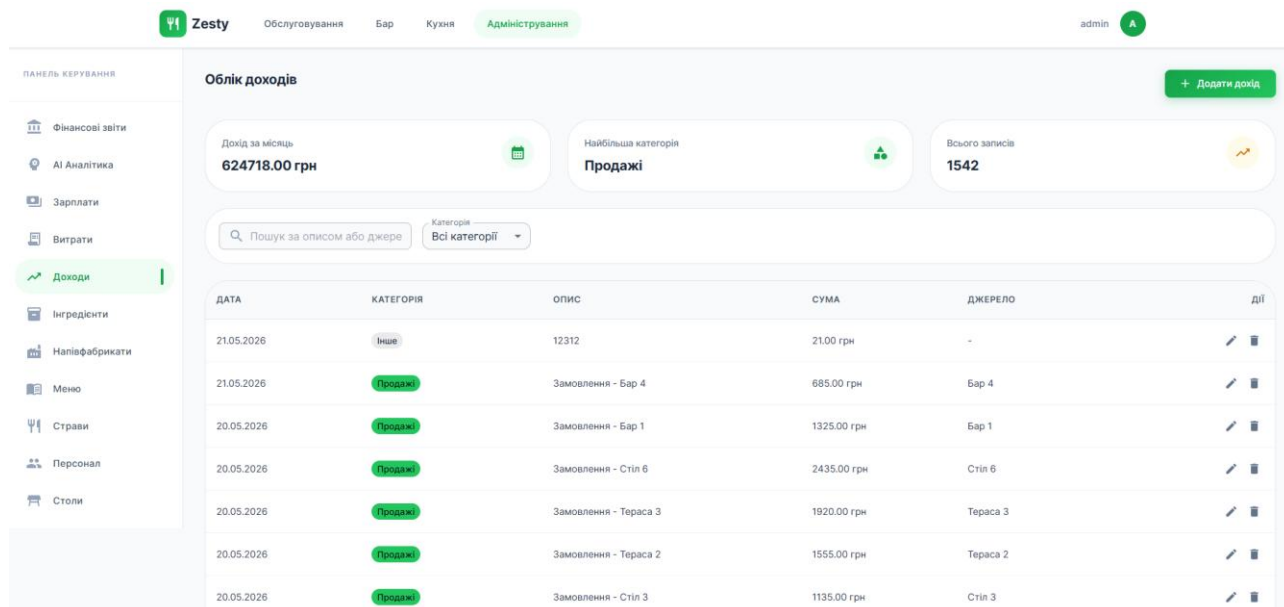


Рис. 3.11. Сторінка обліку доходів

Кожна виплата заробітної плати містить базову ставку, бонуси, утримання та підсумкову суму за визначений розрахунковий період (рис. 3.12.). Статус виплати може бути змінено з «Очікує» на «Виплачено» або «Скасовано», що дозволяє адміністратору відстежувати фактичний стан розрахунків з кожним працівником.

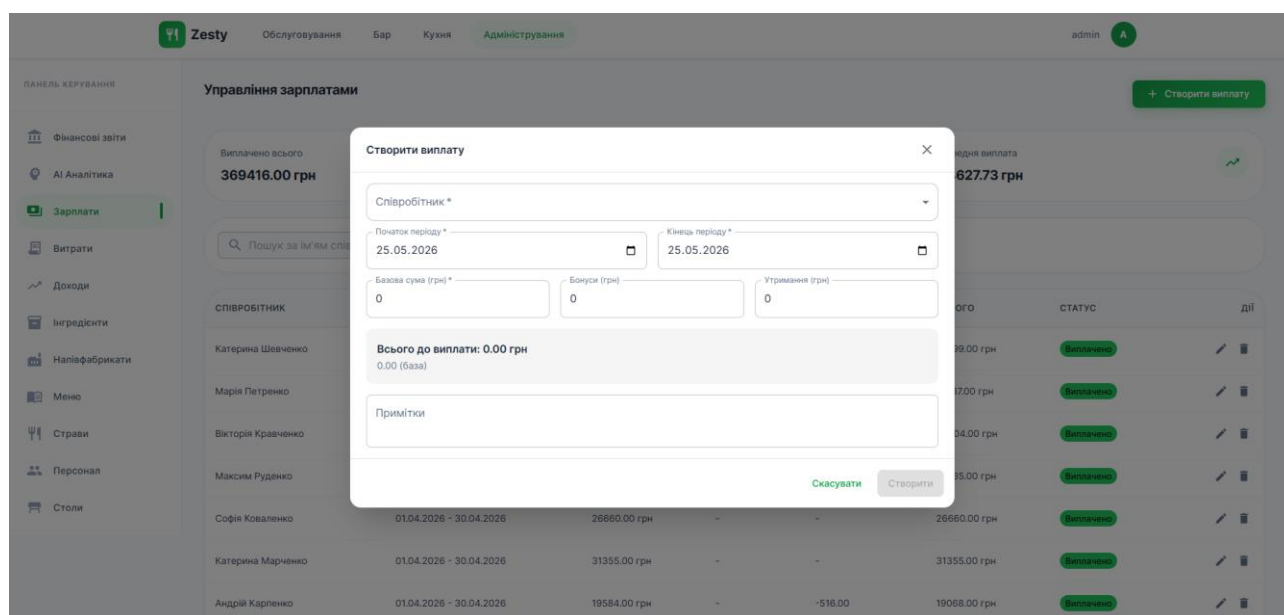


Рис. 3.12. Форма створення виплати з автоматичним розрахунком

Розділ «Фінансові звіти» включає доходи від продажів, витрати за категоріями, виплати зарплати та чистий прибуток. Дані візуалізуються у вигляді графіків та діаграм (рис. 3.13.).

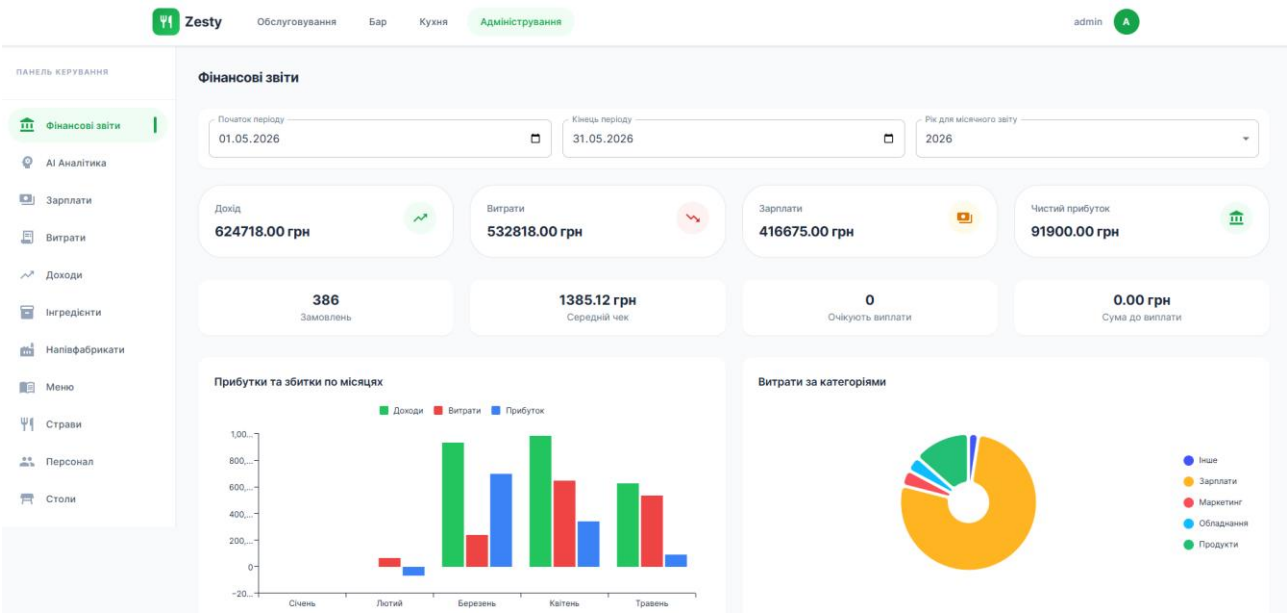


Рис. 3.13. Сторінка фінансових звітів з графіками

Розділ «AI-Аналітика» надає інтелектуальний аналіз роботи ресторану. Система аналізує продажі, виявляє тренди та надає рекомендації щодо оптимізації меню. Інтерактивний чат дозволяє ставити запитання природною мовою (рис. 3.14.).

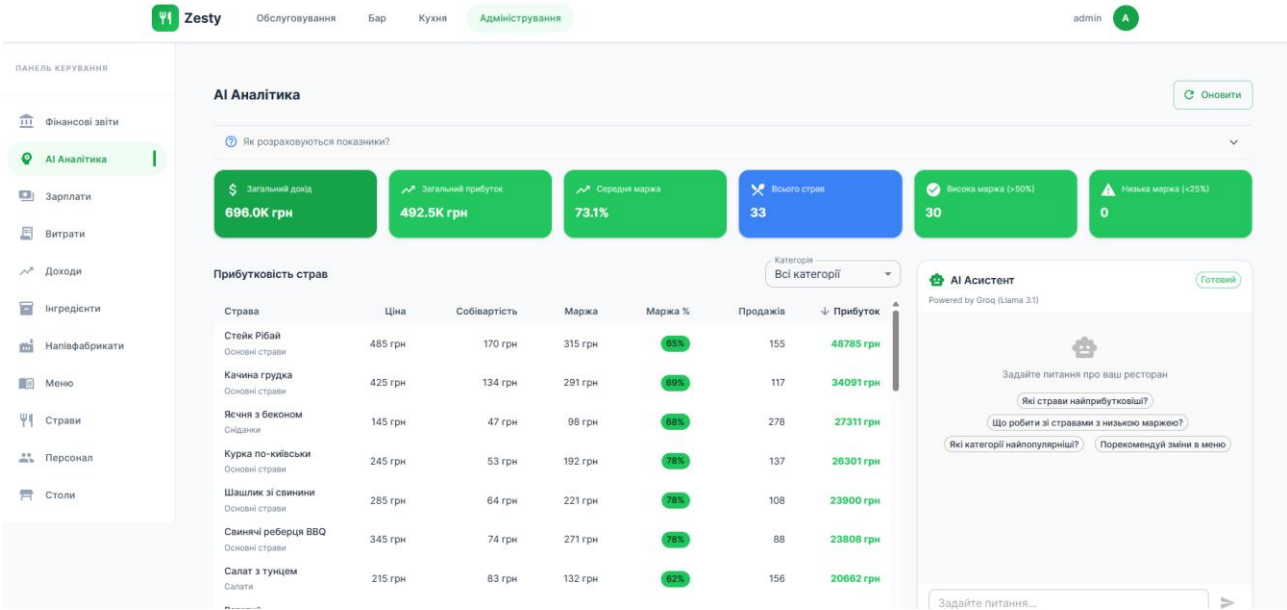


Рис. 3.14. Сторінка AI-аналітики з чатом

Офіціант отримує доступ до обмеженого набору функцій для обслуговування відвідувачів (рис. 3.15.). Після авторизації йому доступні розділи замовлень та перегляду меню – всі адміністративні модулі, фінансові звіти та управління персоналом приховані відповідно до рольової моделі доступу.

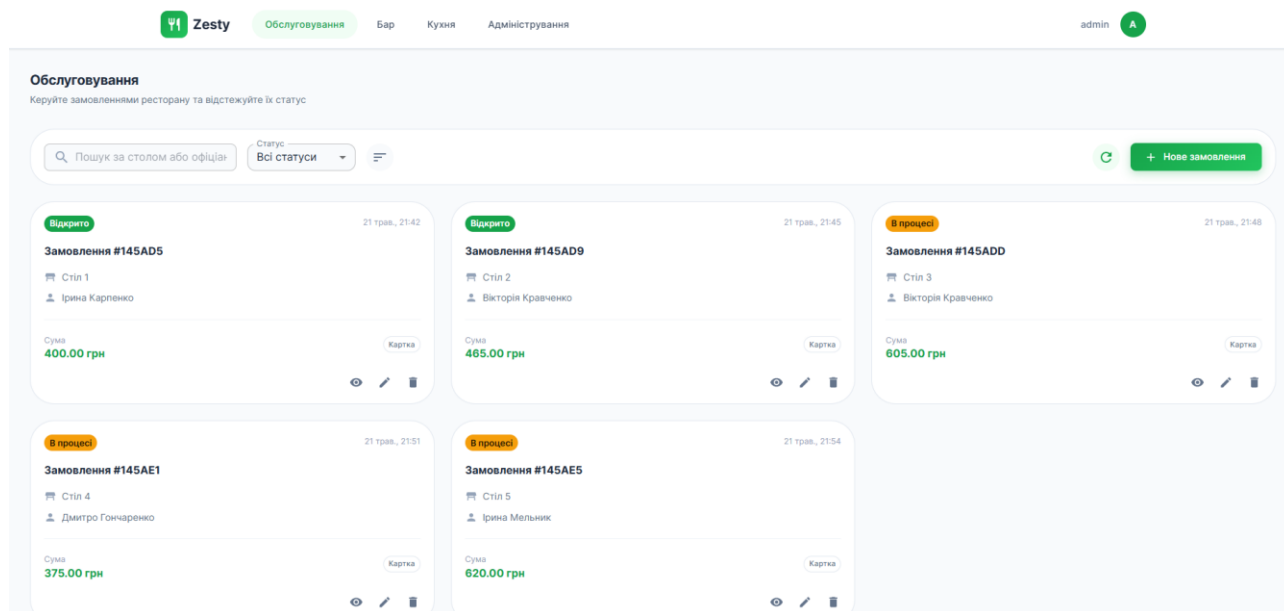


Рис. 3.15. Сторінка замовлень для офіціанта

Офіціант може створювати замовлення, змінювати його статус (відкрите, в процесі приготування, завершене або скасоване), керувати відповідальним за замовлення, та керувати столиком (рис. 3.16.).

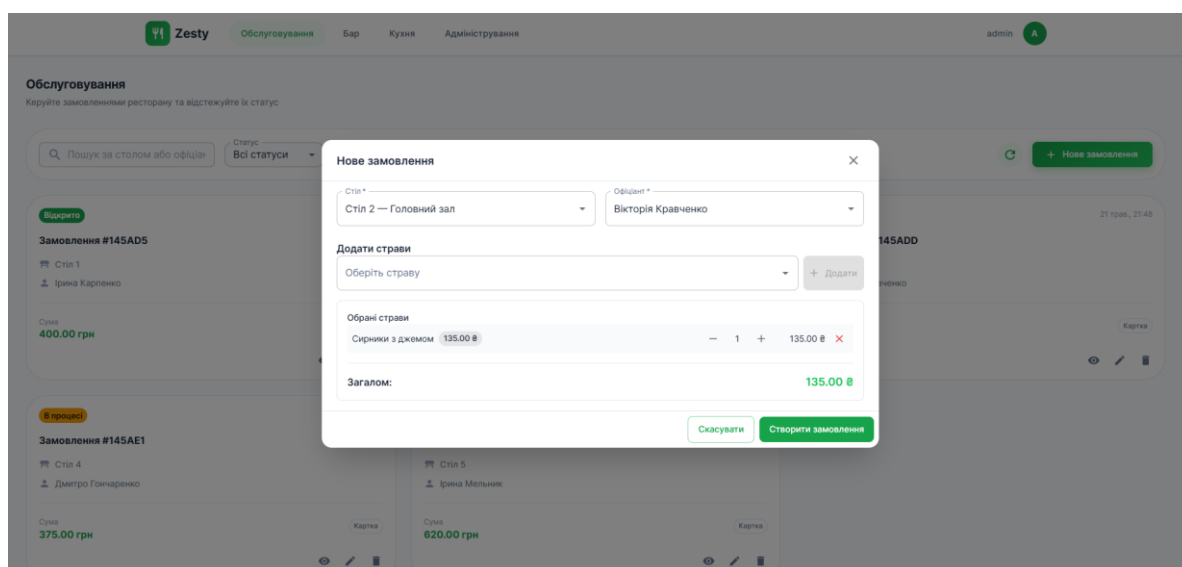


Рис. 3.16. Інтерфейс модифікації замовлення

Після завершення замовлення формується чек із зазначенням способу оплати – готівка, картка або змішаний (рис. 3.17.).

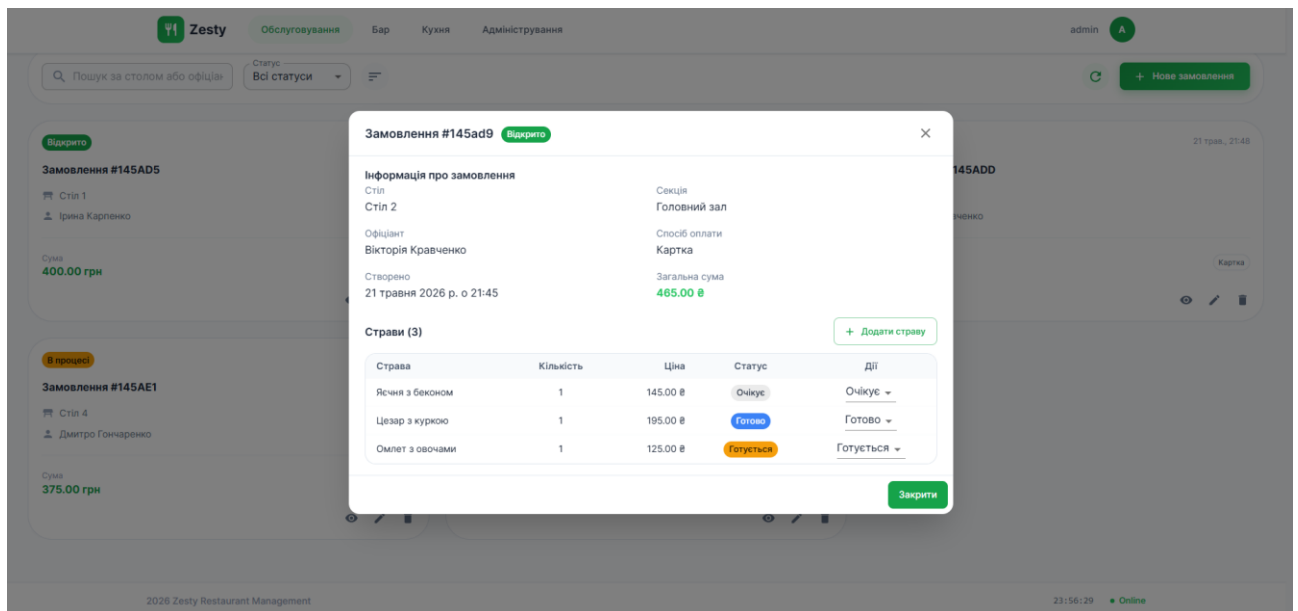


Рис. 3.17. Картка замовлення зі списком страв

Функціонал кухаря. Кухар бачить панель з активними замовленнями у вигляді карток зі списком страв та їх статусами. Інтерфейс організований за принципом Kanban-дошки, де страви розподілені по колонках відповідно до етапу приготування: очікує, готується, готово, подано (рис. 3.18.).

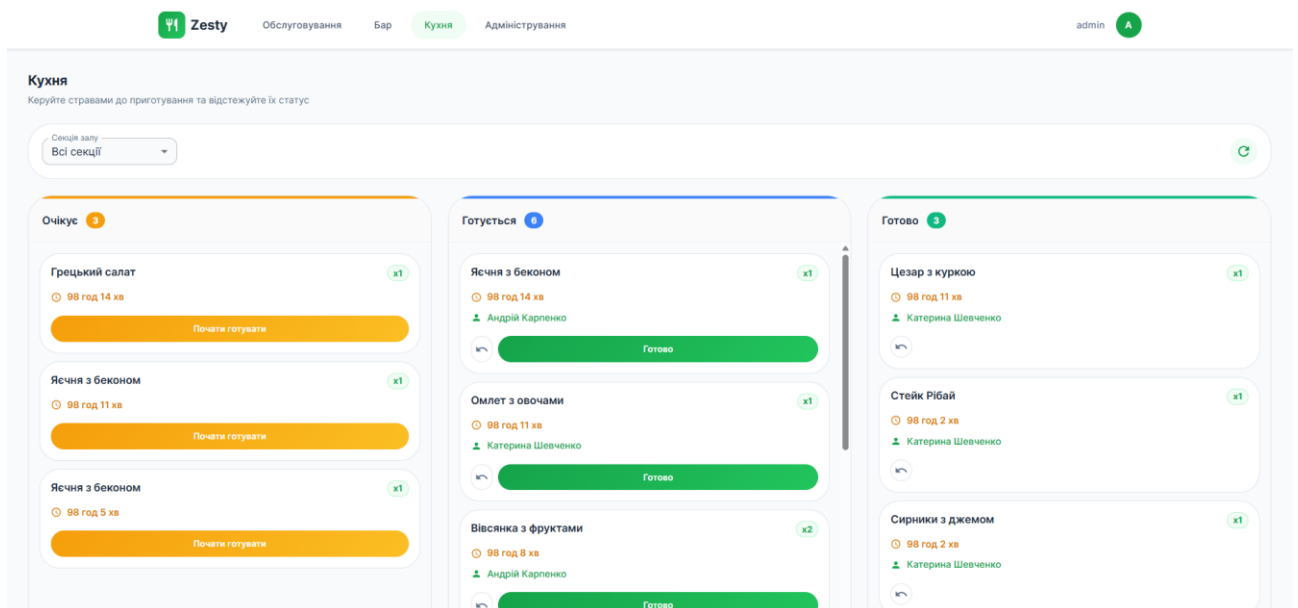


Рис. 3.18. Кухонна панель з активними замовленнями

Кожна картка містить назву страви, кількість порцій, номер столика та час створення замовлення. Кухар змінює статус страви одним кліком, переміщуючи її між колонками. Картки автоматично сортуються за часом – найстаріші замовлення відображаються зверху, що допомагає дотримуватися черговості приготування. Система підсвічує страви, які очікують занадто довго, сигналізуючи про необхідність прискорити виконання. Такий підхід забезпечує наочний контроль над завантаженістю кухні та мінімізує час очікування для гостей. Кухар змінює статус страв ("В приготуванні" або "Готово"). Після зміни офіціант бачить оновлену інформацію. Кожна картка відображає ім'я призначеного кухаря.

Бармен має об'єднаний функціонал кухаря та офіціанта, що відповідає специфіці роботи за барною стійкою. Він може створювати замовлення для гостей барної секції, додавати напої та страви, а також самостійно відстежувати їх приготування. Інтерфейс бармена поєднує панель активних замовлень у вигляді Kanban-дошки та можливість формування чеків (рис. 3.19.). Бармен бачить лише замовлення, що належать до барної секції, що забезпечує розмежування зон відповідальності з офіціантами основного залу. Страви та напої, додані барменом, позначаються відповідним джерелом.

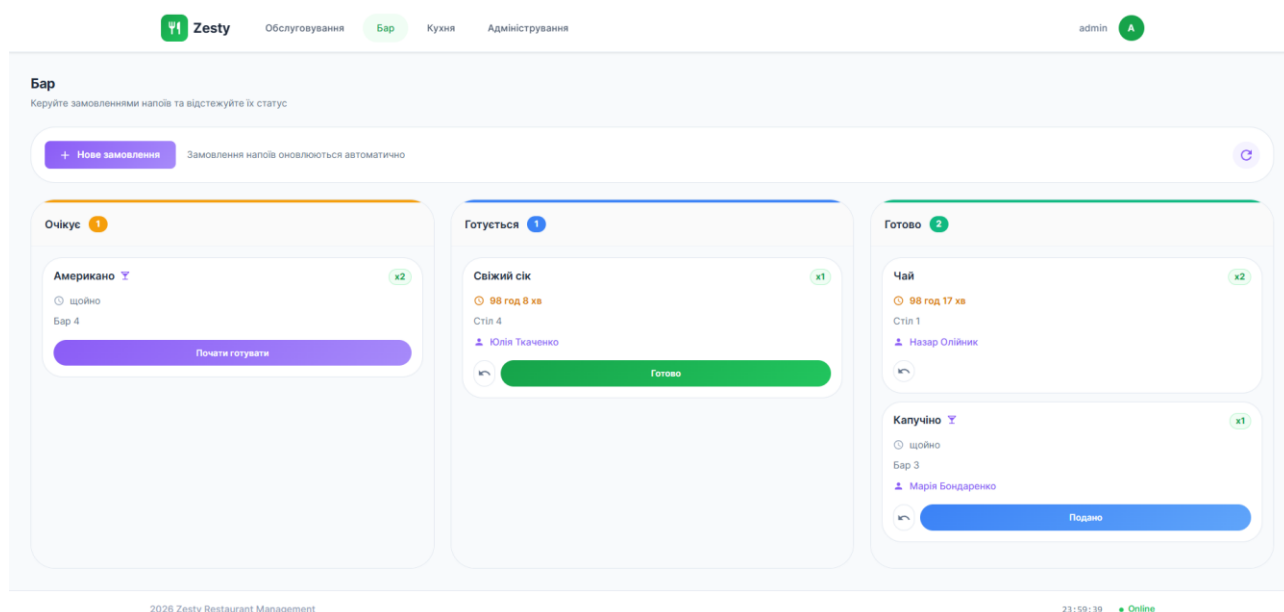


Рис. 3.19. Барменська панель з активними замовленнями

3.3 Опис роботи з клієнтським додатком Zesty Client

Zesty Client призначений для відвідувачів ресторану та дозволяє переглядати меню, інформацію про страви та їх склад безпосередньо зі смартфона. Додаток не потребує авторизації та реєстрації, забезпечуючи миттєвий доступ до актуального асортименту закладу. Інтерфейс оптимізований для мобільних пристроїв та адаптується до різних розмірів екранів. Відвідувачі можуть самостійно ознайомитися з меню під час очікування офіціанта або заздалегідь обрати страви перед візитом до ресторану.

При відкритті відображається головне меню сайту (рис. 3.20.). Дизайн виконаний у стилістиці бренду Zesty з акцентом на візуальну привабливість страв та легкість навігації.

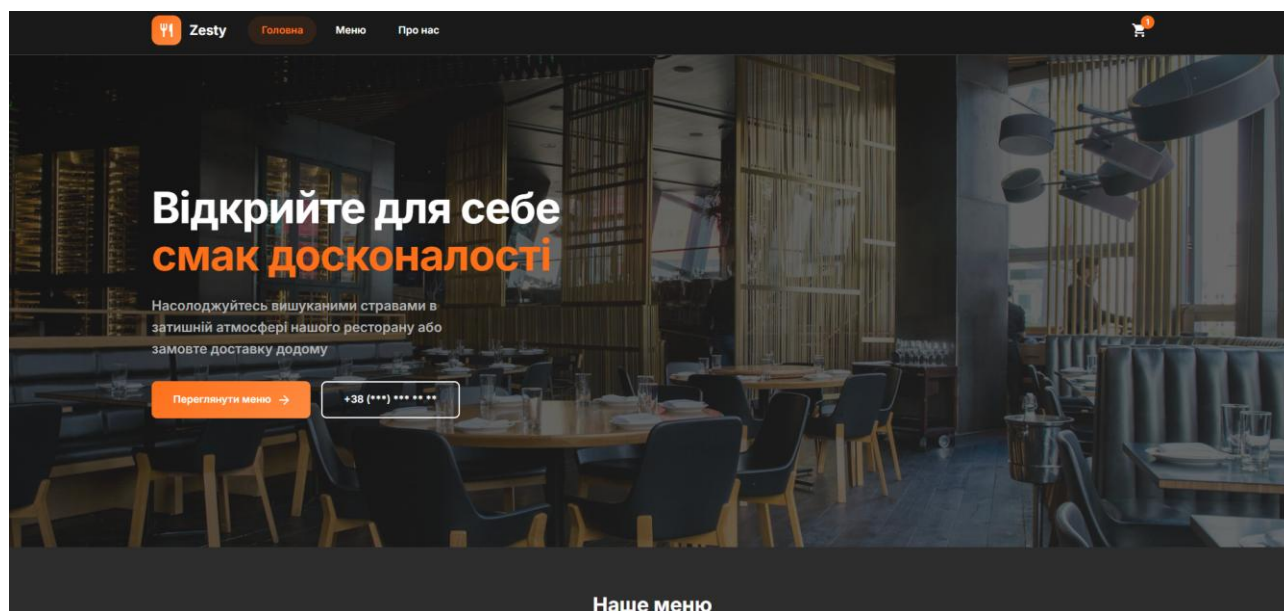


Рис. 3.20. Головна сторінка клієнтського додатку з категоріями меню

При виборі категорії «Меню» відкривається список страв з фото, назвою, описом та ціною (рис. 3.21.). Страви відображаються у вигляді карток з мініатюрою фото та ключовою інформацією – назвою, коротким описом та ціною. За необхідності відвідувач може натиснути на картку страви для перегляду детальної інформації: повного опису, складу інгредієнтів, ваги, калорійності та часу приготування.

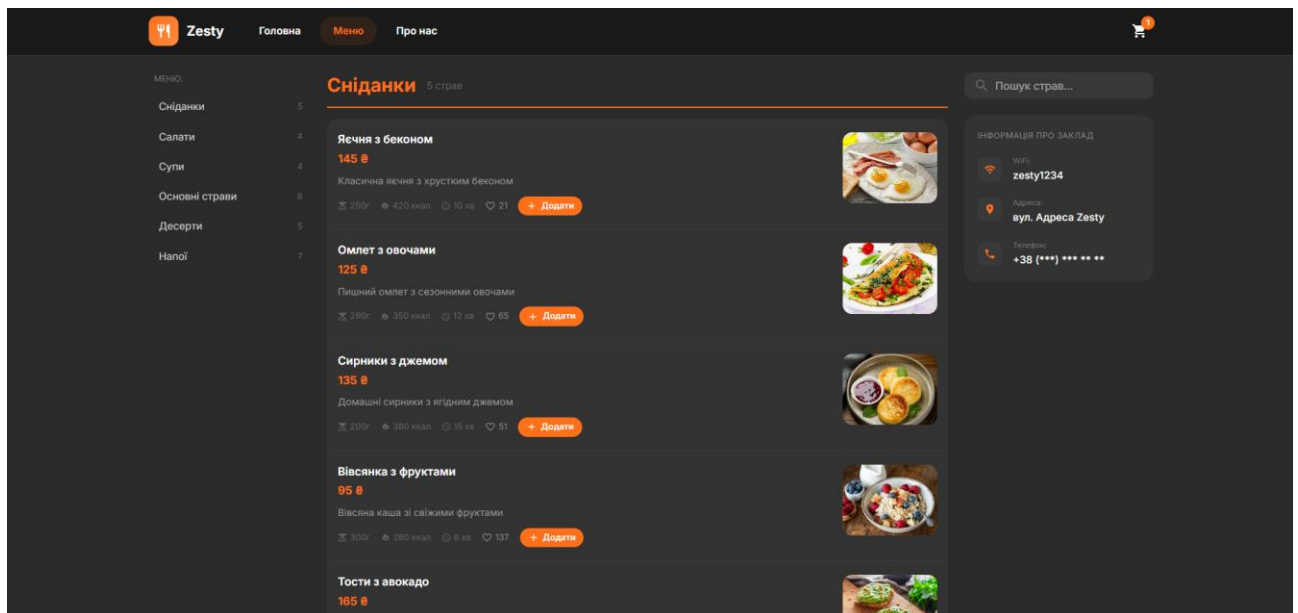


Рис. 3.21. Список страв у категорії

Модалне вікно містить велике фото страви, розгорнутий опис, перелік інгредієнтів з позначенням алергенів, вагу, калорійність та час приготування (рис. 3.22.). Також відображається поточний статус доступності страви – якщо страва тимчасово відсутня, це чітко позначено, щоб відвідувач міг одразу обрати альтернативу.

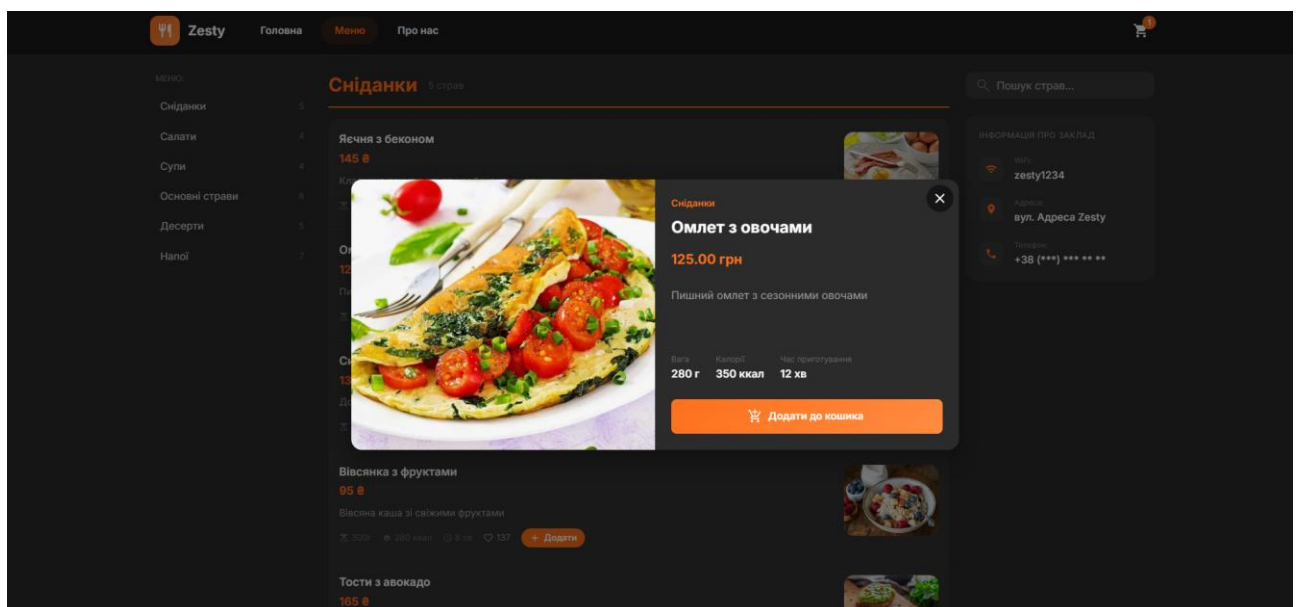


Рис. 3.22. Сторінка детальної інформації про страву

Страви можна фільтрувати за статусом доступності та наявністю алергенів, що особливо важливо для відвідувачів з харчовими обмеженнями. Сортування

дозволяє впорядкувати список за ціною, популярністю або назвою. Страви, які тимчасово відсутні, позначаються відповідним індикатором, але залишаються видимими для ознайомлення. Система автоматично оновлює дані при зміні асортименту в адміністративній панелі.

Додаток оптимізований для мобільних пристроїв та адаптується до різних розмірів екрану (рис. 3.23.).

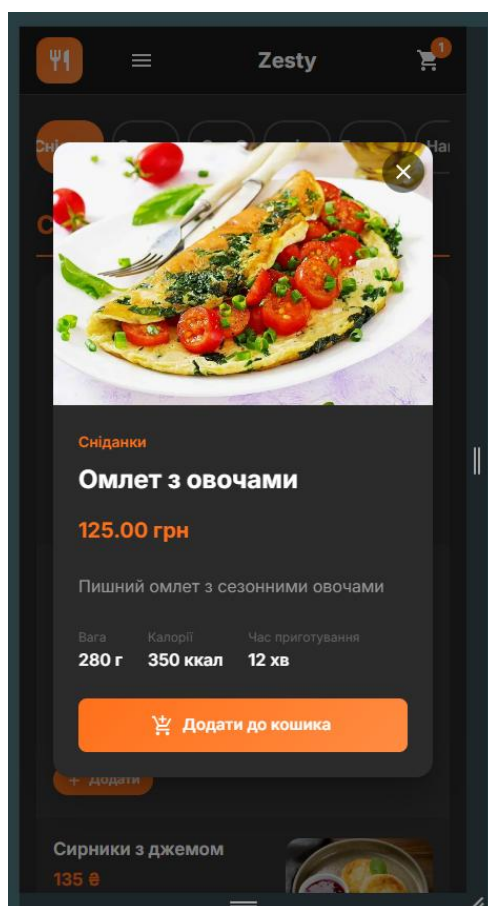


Рис. 3.23. Відображення клієнтського додатку на різних пристроях

При додаванні страв вони потрапляють у кошик, щоб користувачам було легше формувати майбутні замовлення. Кошик зберігає обрані позиції з можливістю зміни кількості кожної страви або видалення зі списку. Система автоматично підраховує загальну вартість замовлення, оновлюючи суму при кожній зміні. Дані кошика зберігаються локально у браузері, тому список не втрачається при оновленні сторінки або тимчасовому закритті додатку (рис. 3.24.).

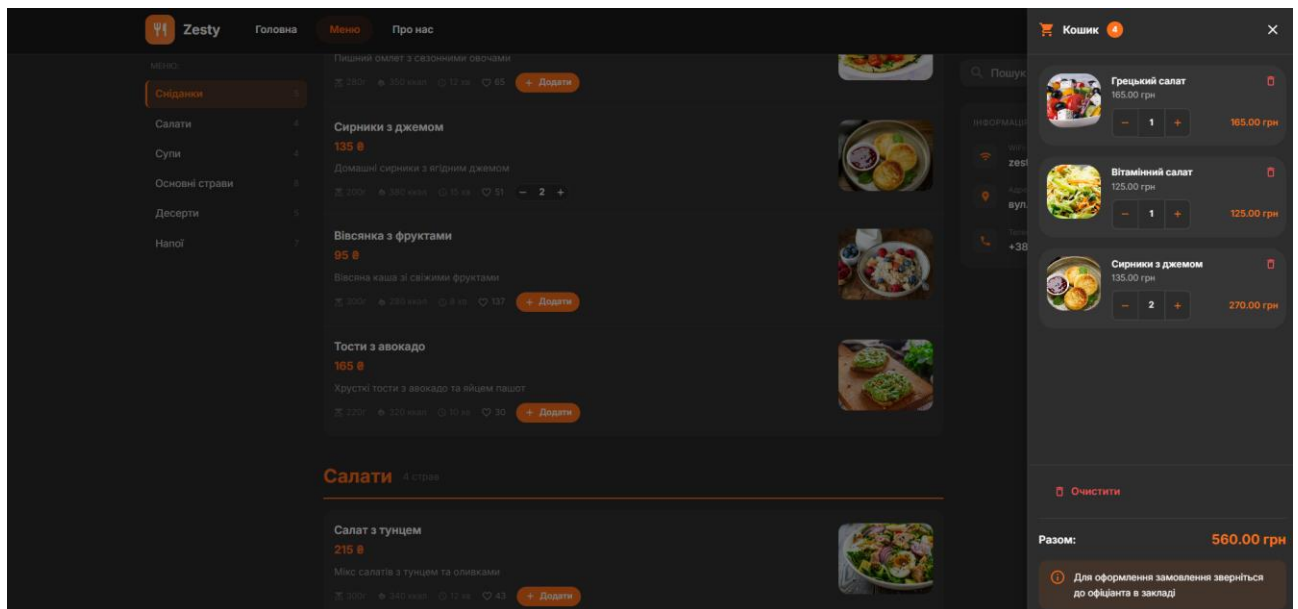


Рис. 3.24. Відображення кошика із замовленнями

3.4 Тестування роботи програмного забезпечення

Застосовано комплексний підхід, що включає модульне тестування окремих компонентів, інтеграційне тестування взаємодії модулів та функціональне тестування повних сценаріїв використання.

Перевірено успішну авторизацію, відхилення при невірному паролі, відхилення для неіснуючого користувача та коректне перенаправлення (табл. 3.1.).

Таблиця 3.1.

Результати тестування авторизації

Тест	Вхідні дані	Очікуваний результат	Результат
Успішна авторизація	Коректний логін та пароль	Перенаправлення на головну	Пройдено
Невірний пароль	Коректний логін, невірний пароль	Повідомлення про помилку	Пройдено
Неіснуючий користувач	Неіснуючий логін	Повідомлення про помилку	Пройдено
Порожні поля	Порожній логін та пароль	Валідаційна помилка	Пройдено

Виконано тестування модуля управління замовленнями, де було перевірено повний цикл обробки: створення, додавання та видалення страв, зміну кількості, зміну статусів та формування чеку (табл. 3.2.).

Таблиця 3.2.

Результати тестування замовлень

Тест	Опис	Очікуваний результат	Результат
Створення замовлення	Вибір столика, створення	Замовлення зі статусом "Відкрите"	Пройдено
Додавання страви	Додавання до замовлення	Страва додана, сума перерахована	Пройдено
Зміна кількості	Зміна кількості страви	Кількість змінена, сума оновлена	Пройдено
Видалення страви	Видалення із замовлення	Страва видалена, сума оновлена	Пройдено
Зміна статусу	Статус "В процесі"	Статус змінено	Пройдено
Формування чеку	Завершення замовлення	Чек створено, замовлення закрито	Пройдено

Виконано тестування модуля фінансової звітності, де було перевірено перевірено коректність агрегації даних з колекцій incomes та expenses (табл. 3.3.).

Таблиця 3.3.

Результати тестування фінансової звітності

Тест	Опис	Очікуваний результат	Результат
Підрахунок доходів	Сума чеків за період	Коректна сума	Пройдено
Групкування витрат	Витрати за категоріями	Коректні суми по категоріях	Пройдено
Підрахунок зарплат	Сума виплат за період	Коректна сума	Пройдено
Розрахунок прибутку	Доходи - Витрати - Зарплати	Коректний результат	Пройдено
Вибір періоду	Фільтрація за датами	Дані відповідають періоду	Пройдено

Виконано тестування продуктивності системи (табл. 3.4.). Вимірювання часу відповіді API проводилось для типових операцій при нормальному

навантаженні. Результати підтвердили відповідність встановлених нефункціональній вимозі – середній час відповіді для більшості запитів не перевищує 200 мс, що значно нижче встановленого ліміту у 500 мс.

Таблиця 3.4.

Результати тестування продуктивності

Операція	Середній час відповіді	Максимальний час	Оцінка
Авторизація	120 мс	250 мс	Добре
Список замовлень	85 мс	180 мс	Добре
Створення замовлення	95 мс	200 мс	Добре
Фінансовий звіт	320 мс	580 мс	Прийнятно
Пошук страв	65 мс	140 мс	Добре

Тестування безпеки (табл. 3.5.) проводилось відповідно до рекомендацій OWASP Top Ten – переліку найпоширеніших вразливостей вебзастосунків [15]. Перевірено захист від несанкціонованого доступу без токена та з недійсним токеном, спроби NoSQL-ін'єкцій у полях введення, XSS-атаки через підставні дані у запитах, а також спроби доступу до захищених маршрутів з роллю нижчого рівня. В усіх випадках система коректно відхилила несанкціоновані запити та повернула відповідні коди помилок.

Таблиця 3.5.

Результати тестування безпеки

Тест	Опис	Очікуваний результат	Результат
Доступ без токена	Запит без авторизації	Відхилення 401	Пройдено
Невалідний токен	Підроблений токен	Відхилення 401	Пройдено
Прострочений токен	Прострочений токен	Відхилення 401	Пройдено
NoSQL ін'єкція	Спроба ін'єкції	Ін'єкція заблокована	Пройдено
XSS атака	Спроба вставки скриптів	Скрипти екрановані	Пройдено

За результатами комплексного тестування всі функціональні вимоги виконані. Модульне тестування охопило основні сервіси системи: авторизацію,

управління замовленнями, розрахунок вартості та формування фінансових звітів. Інтеграційне тестування підтвердило коректну взаємодію між клієнтськими додатками та серверним API через GraphQL. Система коректно обробляє всі сценарії використання, включаючи граничні випадки та некоректні вхідні дані. Перевірка безпеки підтвердила надійність JWT-автентифікації, коректну роботу рольової моделі доступу та захист від типових вразливостей (SQL/NoSQL-ін'єкції, XSS). Виявлені незначні дефекти, переважно пов'язані з відображенням інтерфейсу на окремих пристроях, були оперативно виправлені. Система готова до впровадження та експлуатації в реальних умовах ресторанного бізнесу.

Висновки до третього розділу

У третьому розділі описано порядок встановлення, налаштування вебсистеми управління рестораном, детально розглянуто роботу з інтерфейсами системи та проведено комплексне тестування програмного забезпечення.

Описано загальну схему розгортання трьох компонентів системи на платформі Vercel. Детально описано функціонал адміністративного застосунку Zesty Admin для кожної ролі: адміністратор має повний доступ до всіх модулів, офіціант і бармен – до замовлень своєї секції, кухар – до кухонної панелі. Описано публічний застосунок Zesty Client для відвідувачів ресторану з адаптивним дизайном та переглядом меню без автентифікації.

Проведено модульне, інтеграційне та функціональне тестування системи. Перевірено авторизацію, повний цикл обробки замовлень, фінансову звітність, продуктивність та безпеку. Результати підтверджують відповідність системи функціональним та нефункціональним вимогам, зокрема час відповіді API не перевищує встановленого ліміту у 500 мс. Система готова до впровадження та експлуатації в реальних умовах закладу громадського харчування.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи вирішено актуальне науково-практичне завдання – розроблено вебсистему управління рестораном, що автоматизує ключові бізнес-процеси закладу громадського харчування: управління меню, замовленнями, персоналом, фінансовим обліком та аналітику на основі штучного інтелекту.

Досліджено чотири існуючих програмних аналоги – Poster POS, TurboPOS, Odoo POS та Sygve. Проведено аналіз предметної області ресторанного бізнесу та визначено перелік процесів, що потребують автоматизації. Встановлено, що розглянуті системи є переважно комерційними продуктами з обмеженими можливостями кастомізації, не передбачають інтеграції зі штучним інтелектом, що обумовило доцільність розробки власного рішення.

Обґрунтовано вибір клієнт-серверної архітектури та технологічного стеку. Проведено порівняльний аналіз серверних (NestJS, Express, Fastify) та клієнтських (React, Vue, Angular) фреймворків, REST та GraphQL, а також платформ розгортання. За результатами аналізу обрано GraphQL завдяки підтримці складних вкладених запитів та відсутності надлишкової передачі даних, NestJS як серверний фреймворк, React.js – для клієнтської частини, MongoDB – як систему зберігання даних, Vercel – як платформу розгортання з CI/CD на базі GitHub Actions.

Реалізовано програмний комплекс, що включає три застосунки: серверну частину zesty-service на базі NestJS з GraphQL API та модульною архітектурою (14 модулів), адміністративний інтерфейс zesty-admin для внутрішнього використання персоналом та публічний інтерфейс zesty-client для відвідувачів ресторану. Система реалізує розмежування доступу за чотирма ролями: адміністратор, офіціант, бармен та кухар. Налаштовано CI/CD pipeline з автоматичним тестуванням при кожному pull request та деплоєм на окремі dev і prod середовища.

					ІПЗ.КР.Б-121-26-ІЗ	Арк.
						73

Реалізовано функціональні модулі: управління меню, стравами та інгредієнтами; управління столиками із зазначенням секцій залу; повний цикл обробки замовлень від створення до формування чеку; управління персоналом з трьома типами нарахування заробітної плати; облік доходів та витрат за категоріями; формування фінансових звітів. Додатково реалізовано модуль AI-аналітики на базі Groq API з моделлю LLaMA 3.1, що забезпечує аналіз прибутковості страв та генерацію бізнес-рекомендацій. Безпеку адміністративного інтерфейсу забезпечено механізмом IP-whitelisting.

Проведено комплексне тестування, що охоплює авторизацію, управління замовленнями, фінансову звітність, продуктивність та безпеку. Усі тести пройдено успішно. Середній час відповіді API не перевищує 200 мс, що відповідає встановленим нефункціональним вимогам.

Практична цінність розробленої системи полягає у можливості її безпосереднього впровадження в реальних умовах роботи закладів громадського харчування. Модульна архітектура, використання відкритих технологій та хмарне розгортання забезпечують просте масштабування та адаптацію під специфічні потреби конкретного закладу без значних витрат.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Каламан О. Б., Ряшко Г. М., Ткачук О. В. Проектування бізнес-процесів та управління якістю сервісу на підприємствах ресторанного бізнесу. Економіка та суспільство. 2025. № 80. DOI: <https://doi.org/10.32782/2524-0072/2025-80-178>.
2. Мотицький, Н., Фант, М. (2026). Архітектура системи машинного навчання для прогнозування попиту та оптимізації закупівель у ресторанному бізнесі. Information Technology: Computer Science, Software Engineering and Cyber Security. 2026. № 1. DOI: <https://doi.org/10.32782/IT/2026-1-20>.
3. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol : O'Reilly Media, 2020. 310 p.
4. Bradshaw S., Brazil E., Chodorow K. MongoDB: The Definitive Guide. 3rd ed. Sebastopol : O'Reilly Media, 2019. 514 p.
5. Brito G., Valente M. T. REST vs GraphQL: A Controlled Experiment. Proceedings of the 17th IEEE International Conference on Software Architecture (ICSA). 2020. P. 81–91. DOI: 10.1109/ICSA47634.2020.00016.
6. Brown T. B., Mann B., Ryder N. et al. Language Models are Few-Shot Learners. Advances in Neural Information Processing Systems (NeurIPS). 2020. Vol. 33. P. 1877–1901.
7. Davis B., Lockwood A. Food and Beverage Management. 4th ed. London : Routledge, 2008. 400 p.
8. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : PhD dissertation / University of California, Irvine. Irvine, 2000. 180 p. URL: <https://ics.uci.edu/~fielding/pubs/dissertation/top.htm>.
9. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2003. 560 p.
10. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.

11. Hartig O., Pérez J. Semantics and Complexity of GraphQL. Proceedings of the 2018 World Wide Web Conference (WWW'18). ACM, 2018. P. 1155–1164. DOI: 10.1145/3178876.3186014.

12. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. Internet Engineering Task Force (IETF), 2015. URL: <https://datatracker.ietf.org/doc/html/rfc7519>.

13. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Upper Saddle River : Prentice Hall, 2017. 432 p.

14. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 616 p.

15. OWASP Foundation. OWASP Top Ten 2021: The Ten Most Critical Web Application Security Risks. 2021. URL: <https://owasp.org/Top10/>. (дата звернення: 11.05.2026).

16. Pressman R. S. Software Engineering: A Practitioner's Approach. 8th ed. New York : McGraw-Hill, 2014. 976 p.

17. Touvron H., Martin L., Stone K. et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288. 2023. URL: <https://arxiv.org/abs/2307.09288>.

18. Vaswani A., Shazeer N., Parmar N. et al. Attention Is All You Need. Advances in Neural Information Processing Systems (NeurIPS). 2017. Vol. 30. P. 5998–6008.

19. Apollo GraphQL Inc. Apollo Client Documentation. URL: <https://www.apollographql.com/docs/react/>. (дата звернення: 20.03.2026).

20. GraphQL Foundation. GraphQL Specification (October 2021 Edition). URL: <https://spec.graphql.org/October2021/>. (дата звернення: 20.03.2026).

21. Groq Inc. Groq API Documentation. URL: <https://console.groq.com/docs/>. (дата звернення: 17.05.2026).

22. MongoDB Inc. MongoDB Manual. URL: <https://www.mongodb.com/docs/manual/>. (дата звернення: 14.04.2026).

					ІІЗ.КР.Б–121–26–ІІЗ	Арк.
						76

23. MongoDB Inc. Mongoose ODM v7 Documentation. URL: <https://mongoosejs.com/docs/>. (дата звернення: 20.03.2026).

24. Meta Open Source. React – The Library for Web and Native User Interfaces. URL: <https://react.dev/>. (дата звернення: 20.03.2026).

25. Microsoft Corp. TypeScript: JavaScript with Syntax for Types. URL: <https://www.typescriptlang.org/docs/>. (дата звернення: 20.03.2026).

26. MUI. Material UI: A comprehensive library of components. URL: <https://mui.com/material-ui/getting-started/>. (дата звернення: 20.03.2026).

27. OpenAPI Initiative. OpenAPI Specification v3.0.3. URL: <https://spec.openapis.org/oas/v3.0.3>. (дата звернення: 25.04.2026).

28. OpenJS Foundation. Node.js Documentation. URL: <https://nodejs.org/en/docs/>. (дата звернення: 23.03.2026).

29. Sadalage P. J., Fowler M. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Upper Saddle River : Addison-Wesley Professional, 2012. 192 p.

30. Sommerville I. Software Engineering. 10th ed. Harlow : Pearson Education, 2015. 816 p.

31. Trilon. NestJS – A Progressive Node.js Framework: Documentation. URL: <https://docs.nestjs.com/>. (дата звернення: 20.04.2026).

32. Vite. Vite: Next Generation Frontend Tooling. URL: <https://vitejs.dev/guide/>. (дата звернення: 20.03.2026).

ДОДАТКИ

					ІІЗ.КР.Б-121-26-ІІЗ	Арк.
						78

```
// ===== zesty-service/src/modules/zesty/order/order.entity.ts =====
```

```
import { ObjectType, Field, GraphQLISODatetime } from '@nestjs/graphql';
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import { ObjectId } from 'mongodb';
import { Document, Schema as MongooseSchema } from 'mongoose';
import { OrderPaymentMethod, OrderStatus } from
'../../basics/enums/common.enums';
import { Staff } from '../staff/staff.entity';
import { Table } from '../table/table.entity';
```

```
@Schema()
```

```
@ObjectType('Order')
```

```
export class Order extends Document {
```

```
  @Field(() => String)
```

```
  declare _id: ObjectId;
```

```
  @Prop({ type: MongooseSchema.Types.ObjectId, ref: Staff.name, index: true,
required: true })
```

```
  @Field((_type) => Staff)
```

```
    staff: string;
```

```
  @Prop({ type: MongooseSchema.Types.ObjectId, ref: Table.name, index: true,
required: true })
```

```
  @Field((_type) => Table)
```

```
    table: string;
```

```
  @Prop({ enum: OrderStatus, required: true })
```

```
  @Field((_type) => OrderStatus)
```

```
    status: OrderStatus;
```

```
  @Prop({ type: Number, default: 0 })
```

					ІІІ3.КР.Б-121-26-ІІ3	Арк.
						79

```
@Field(() => Number)
```

```
  totalPrice: number;
```

```
@Prop({ enum: OrderPaymentMethod, default: OrderPaymentMethod.CARD })
```

```
@Field((_type) => OrderPaymentMethod)
```

```
  paymentMethod: OrderPaymentMethod;
```

```
@Prop(() => Date)
```

```
@Field(() => GraphQLISODatetime)
```

```
  created: Date;
```

```
@Prop(() => Date)
```

```
@Field(() => GraphQLISODatetime)
```

```
  completed: Date;
```

```
}
```

```
export const OrderSchema = SchemaFactory.createClass(Order);
```

```
// ===== zesty-service/src/modules/zesty/income/income.entity.ts =====
```

```
import { ObjectType, Field, GraphQLISODatetime, Float } from '@nestjs/graphql';
```

```
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
```

```
import { ObjectId } from 'mongodb';
```

```
import { Document } from 'mongoose';
```

```
import { IncomeCategory } from '../../basics/enums/common.enums';
```

```
@Schema()
```

```
@ObjectType('Income')
```

```
export class Income extends Document {
```

```
  @Field(() => String)
```

```
  declare _id: ObjectId;
```

```
@Prop({ enum: IncomeCategory, required: true })
```

```
@Field(() => IncomeCategory)
```

```
  category: IncomeCategory;
```

```
@Prop({ type: Number, required: true })
```

```
@Field(() => Float)
```

```
  amount: number;
```

```
@Prop({ required: true })
```

```
@Field(() => String)
```

```
  description: string;
```

```
@Prop({ type: Date, required: true })
```

```
@Field(() => GraphQLISODatetime)
```

```
  date: Date;
```

```
@Prop({ required: false })
```

```
@Field(() => String, { nullable: true })
```

```
  source?: string;
```

```
@Prop({ required: false })
```

```
@Field(() => String, { nullable: true })
```

```
  referenceNumber?: string;
```

```
@Prop(() => Date)
```

```
@Field(() => GraphQLISODatetime)
```

```
  created: Date;
```

```
}
```

```
export const IncomeSchema = SchemaFactory.createClass(Income);
```

```
// ===== zesty-service/src/modules/zesty/dish/dish.entity.ts =====
```

```
import { ObjectType, Field, GraphQLISODatetime, Float, Int } from
 '@nestjs/graphql';
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import { ObjectId } from 'mongodb';
import { Document, Schema as MongooseSchema } from 'mongoose';
import { DishStatus } from '../../../../basics/enums/common.enums';
import { Menu } from '../../../menu/menu.entity';
```

```
@Schema()
```

```
@ObjectType('Dish')
```

```
export class Dish extends Document {
```

```
  @Field(() => String)
```

```
  declare _id: ObjectId;
```

```
  @Prop({ required: true })
```

```
  @Field((_type) => String)
```

```
    title: string;
```

```
  @Prop({ type: MongooseSchema.Types.ObjectId, ref: Menu.name, index: true,
    required: true })
```

```
  @Field(() => Menu)
```

```
    menu: string;
```

```
  @Prop({ required: true })
```

```
  @Field((_type) => String)
```

```
    description: string;
```

```
  @Prop({ required: true })
```

```
  @Field((_type) => Float)
```

```
    price: number;
```

					ИПЗ.КР.Б-121-26-ИЗ	Арк.
						82

```
@Prop({ required: false })
```

```
@Field((_type) => String)
```

```
photo: string;
```

```
@Prop({ enum: DishStatus, required: true })
```

```
@Field((_type) => DishStatus)
```

```
status: DishStatus;
```

```
@Prop({ default: 0 })
```

```
@Field((_type) => Int)
```

```
likes: number;
```

```
@Prop({ type: [String], default: [] })
```

```
@Field((_type) => [String])
```

```
allergens: string[];
```

```
@Prop(() => Date)
```

```
@Field(() => GraphQLISODatetime)
```

```
created: Date;
```

```
}
```

```
export const DishSchema = SchemaFactory.createClass(Dish);
```

```
// ===== zesty-service/src/modules/zesty/order/order.resolver.ts =====
```

```
import { Resolver, Query, Mutation, Args } from '@nestjs/graphql';
```

```
import { PaginationInput } from '../../basics/types/pagination.types';
```

```
import { Order } from './order.entity';
```

```
import { CreateOrderInput, OrderFilterInput, UpdateOrderInput } from  
'./order.inputs';
```

```

import { PaginatedOrderResponse } from './order.pagination';
import { OrderService } from './order.service';

@Resolver(() => Order)
export class OrderResolver {
  constructor(private readonly orderService: OrderService) {}

  @Mutation(() => Order)
  createOrder(
    @Args('createOrderInput') createOrderInput: CreateOrderInput,
  ): Promise<Order> {
    return this.orderService.create(createOrderInput);
  }

  @Query(() => [Order])
  findAllOrders(): Promise<Order[]> {
    return this.orderService.findAll();
  }

  @Query(() => PaginatedOrderResponse)
  findAllOrdersPaginated(
    @Args('filter', { type: () => OrderFilterInput, nullable: true }) filter?:
    OrderFilterInput,
    @Args('pagination', { type: () => PaginationInput, nullable: true }) pagination?:
    PaginationInput,
  ): Promise<PaginatedOrderResponse> {
    return this.orderService.findAllPaginated(filter, pagination);
  }

  @Query(() => Order)
  findOneOrder(@Args('id', { type: () => String }) id: string): Promise<Order> {
    return this.orderService.findOne(id);
  }
}

```

```
@Mutation(() => Order)
```

```
updateOrder(
```

```
  @Args('id', { type: () => String }) id: string,
```

```
  @Args('updateOrderInput') updateOrderInput: UpdateOrderInput,
```

```
): Promise<Order> {
```

```
  return this.orderService.update(id, updateOrderInput);
```

```
}
```

```
@Mutation(() => Order)
```

```
removeOrder(@Args('id', { type: () => String }) id: string): Promise<Order> {
```

```
  return this.orderService.remove(id);
```

```
}
```

```
}
```

```
// ===== zesty-service/src/modules/zesty/order/order.service.ts =====
```

```
import { Injectable } from '@nestjs/common';
```

```
import { IncomeCategory, OrderStatus } from '../../../basics/enums/common.enums';
```

```
import { PaginationInput } from '../../../basics/types/pagination.types';
```

```
import { IncomeActions } from '../income/income.actions';
```

```
import { TableActions } from '../table/table.actions';
```

```
import { OrderActions } from './order.actions';
```

```
import { Order } from './order.entity';
```

```
import { CreateOrderInput, OrderFilterInput, UpdateOrderInput } from  
'./order.inputs';
```

```
import { PaginatedOrderResponse } from './order.pagination';
```

```
@Injectable()
```

```
export class OrderService {
```

```
  constructor(
```

					III3.KP.Б-121-26-III	Арк.
						85

```

private readonly actions: OrderActions,
private readonly incomeActions: IncomeActions,
private readonly tableActions: TableActions,
) {}

async create(createOrderInput: CreateOrderInput): Promise<Order> {
  return this.actions.saveNewOrderToDb(createOrderInput);
}

async findAll(): Promise<Order[]> {
  return this.actions.findAllOrders();
}

async findAllPaginated(
  filter?: OrderFilterInput,
  pagination?: PaginationInput,
): Promise<PaginatedOrderResponse> {
  return this.actions.findAllOrdersPaginated(filter, pagination);
}

async findOne(id: string): Promise<Order> {
  return this.actions.findOrderById(id);
}

async update(id: string, updateOrderInput: UpdateOrderInput): Promise<Order> {
  const currentOrder = await this.actions.findOrderById(id);
  const wasCompleted = currentOrder.status === OrderStatus.COMPLETED;
  const order = await this.actions.updateOrderById(id, updateOrderInput);

  const isNowCompleted = updateOrderInput.status ===
OrderStatus.COMPLETED;
  if (!wasCompleted && isNowCompleted) {

```

```

    await this.incomeActions.saveNewIncomeToDb({
      category: IncomeCategory.SALES,
      amount: order.totalPrice || 0,
      description: 'Замовлення',
      date: new Date(),
      referenceNumber: order._id?.toString(),
    });
  }

  return order;
}

async remove(id: string): Promise<Order> {
  return this.actions.removeOrderById(id);
}
}

// ===== zesty-service/src/modules/auth/auth.service.ts =====

import { Injectable, UnauthorizedException } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import * as bcrypt from 'bcrypt';
import { UserRole } from '../basics/enums/common.enums';
import { User } from '../zesty/user/user.entity';
import { CreateUserInput } from '../zesty/user/user.inputs';
import { UserService } from '../zesty/user/user.service';
import { SignInInput } from './auth.inputs';
import { AuthJwtPyLoad, AuthResponse, JwtUser } from './auth.types';

@Injectable()
export class AuthService {

```

```

constructor(
  private readonly userService: UserService,
  private readonly jwtService: JwtService,
) {}

async hashPassword(password: string): Promise<string> {
  const salt = await bcrypt.genSalt(10);
  return bcrypt.hash(password, salt);
}

async comparePasswords(password: string, hash: string): Promise<boolean> {
  return bcrypt.compare(password, hash);
}

async registerUser(createUserInput: CreateUserInput): Promise<User> {
  const hashedPassword = await this.hashPassword(createUserInput.password);
  const newUser: CreateUserInput = {
    ...createUserInput,
    password: hashedPassword,
    role: UserRole.GUEST,
  };
  return this.userService.create(newUser);
}

async loginUser(user: User, context): Promise<AuthResponse> {
  const { accessToken } = await this.generateAccessToken(user.id);
  const refreshToken = this.jwtService.sign({ sub: user.id });

  context.res.cookie('refreshToken', refreshToken, {
    httpOnly: true,
    secure: true,
    sameSite: 'none',
  });
}

```

```

    path: '/',
  });

  return { userId: user.id, role: user.role, accessToken };
}

async validateLocalUser({ email, login, password }: SignInInput): Promise<User> {
  const identifier = email || login;
  const user = await this.userService.findByLoginOrEmail(identifier || "");
  const isPasswordMatches = await this.comparePasswords(password,
user.password);

  if (!isPasswordMatches) {
    throw new UnauthorizedException('Invalid Credentials');
  }
  return user;
}

async generateAccessToken(userId: string): Promise<{ accessToken: string }> {
  const payload: AuthJwtPayload = { sub: { userId } };
  const accessToken = await this.jwtService.signAsync(payload);
  return { accessToken };
}

async validateJwtUser(userId: string): Promise<JwtUser> {
  const user = await this.userService.findOne(userId);
  return { userId: user.id, role: user.role };
}
}

// ===== zesty-service/src/modules/zesty/financialReport/financialReport.service.ts
=====

```

```

import { Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { Model } from 'mongoose';
import { PaymentStatus, ExpenseCategory } from
'../../basics/enums/common.enums';
import { Expense } from '../expense/expense.entity';
import { Income } from '../income/income.entity';
import { SalaryPayment } from '../salaryPayment/salaryPayment.entity';
import { ProfitLossReport, ExpenseByCategory } from './financialReport.types';

```

```
@Injectable()
```

```
export class FinancialReportService {
```

```
  constructor(
```

```
    @InjectModel(Expense.name) private readonly expenseModel: Model<Expense>,
```

```
    @InjectModel(Income.name) private readonly incomeModel: Model<Income>,
```

```
    @InjectModel(SalaryPayment.name) private readonly salaryPaymentModel:
    Model<SalaryPayment>,
```

```
  ) {}
```

```

  async getProfitLossReport(periodStart: Date, periodEnd: Date):
  Promise<ProfitLossReport> {

```

```

    const [revenue, expenses, salaries, expensesByCategory] = await Promise.all([
      this.getTotalRevenue(periodStart, periodEnd),
      this.getTotalExpenses(periodStart, periodEnd),
      this.getTotalSalaries(periodStart, periodEnd),
      this.getExpensesByCategory(periodStart, periodEnd),
    ]);

```

```
    const netProfit = revenue - expenses;
```

```

    return {
      totalRevenue: revenue,

```

```

totalExpenses: expenses,
totalSalaries: salaries,
grossProfit: revenue,
netProfit,
expensesByCategory,
periodStart,
periodEnd,
};
}

```

```

async getExpensesByCategory(periodStart: Date, periodEnd: Date):
Promise<ExpenseByCategory[]> {
  const result = await this.expenseModel.aggregate([
    { $match: { date: { $gte: periodStart, $lte: periodEnd } } },
    { $group: { _id: '$category', total: { $sum: '$amount' } } },
  ]).exec();

  const totalExpenses = result.reduce((sum, item) => sum + item.total, 0);

  return result.map((item) => ({
    category: item._id as ExpenseCategory,
    total: item.total,
    percentage: totalExpenses > 0 ? (item.total / totalExpenses) * 100 : 0,
  }));
}

```

```

private async getTotalRevenue(periodStart: Date, periodEnd: Date):
Promise<number> {
  const result = await this.incomeModel.aggregate([
    { $match: { date: { $gte: periodStart, $lte: periodEnd } } },
    { $group: { _id: null, total: { $sum: '$amount' } } },
  ]).exec();
  return result[0]?.total || 0;
}

```

```
}
```

```
private async getTotalExpenses(periodStart: Date, periodEnd: Date):  
Promise<number> {
```

```
  const result = await this.expenseModel.aggregate([  
    { $match: { date: { $gte: periodStart, $lte: periodEnd } } },  
    { $group: { _id: null, total: { $sum: '$amount' } } },  
  ]).exec();
```

```
  return result[0]?.total || 0;
```

```
}
```

```
private async getTotalSalaries(periodStart: Date, periodEnd: Date):  
Promise<number> {
```

```
  const result = await this.salaryPaymentModel.aggregate([  
    { $match: { status: PaymentStatus.PAID, paidAt: { $gte: periodStart, $lte:  
periodEnd } } },
```

```
    { $group: { _id: null, total: { $sum: '$totalAmount' } } },
```

```
  ]).exec();
```

```
  return result[0]?.total || 0;
```

```
}
```

```
}
```

```
// ===== zesty-
```

```
admin/src/routes/AuthenticatedApp/Admin/DishManagement/DishManagement.hook  
s.ts =====
```

```
import { useState, useMemo, useCallback } from 'react';
```

```
import { useCreateDishMutation } from  
'../../../../../basics/graphql/mutations/dishes/createDish.ts';
```

```
import { useRemoveDishMutation } from  
'../../../../../basics/graphql/mutations/dishes/removeDish.ts';
```

```
import { useUpdateDishMutation } from  
'../../../../../basics/graphql/mutations/dishes/updateDish.ts';
```

					III3.KP.Б-121-26-III	Арк.
						92

```

import useGetAllDishes from '../../../../../basics/graphql/queries/dishes/getAllDishes.ts';
import useGetAllMenus from '../../../../../basics/graphql/queries/getAllMenus.ts';
import { Dish, DishStatus, Menu } from '../../../../../generated/zesty.types.ts';

export const useDishManagementData = () => {
  const { data: dishes, loading: dishesLoading, error: dishesError, refetch } =
    useGetAllDishes();

  const { data: menus, loading: menusLoading, error: menusError } =
    useGetAllMenus();

  return {
    dishes,
    menus,
    loading: dishesLoading || menusLoading,
    error: dishesError || menusError,
    refetch,
  };
};

export const useDishManagementFilters = (
  dishes: Dish[],
  searchTerm: string,
  statusFilter: DishStatus | 'ALL',
  menuFilter: string,
) => {
  const filteredDishes = useMemo(() => {
    return dishes.filter((dish) => {
      const matchesSearch = searchTerm === ''
        || dish.title.toLowerCase().includes(searchTerm.toLowerCase())
        || dish.description?.toLowerCase().includes(searchTerm.toLowerCase());
      const matchesStatus = statusFilter === 'ALL' || dish.status === statusFilter;
      const matchesMenu = menuFilter === 'ALL' || dish.menu?._id === menuFilter;
      return matchesSearch && matchesStatus && matchesMenu;
    });
  }, [dishes, searchTerm, statusFilter, menuFilter]);
};

```

```

    });
    }, [dishes, searchTerm, statusFilter, menuFilter]);

    return { filteredDishes };
  };

export const useDishManagementHandlers = (menus: Menu[], onRefetch: () => void)
=> {
  const [isFormOpen, setIsFormOpen] = useState(false);
  const [selectedDish, setSelectedDish] = useState<Dish | null>(null);
  const [searchTerm, setSearchTerm] = useState("");
  const [statusFilter, setStatusFilter] = useState<DishStatus | 'ALL'>('ALL');
  const [menuFilter, setMenuFilter] = useState<string>('ALL');

  const [createDish, { loading: createLoading }] = useCreateDishMutation();
  const [updateDish, { loading: updateLoading }] = useUpdateDishMutation();
  const [removeDish, { loading: removeLoading }] = useRemoveDishMutation();

  const handleOpenCreate = useCallback(() => {
    setSelectedDish(null);
    setIsFormOpen(true);
  }, []);

  const handleOpenEdit = useCallback((dish: Dish) => {
    setSelectedDish(dish);
    setIsFormOpen(true);
  }, []);

  const handleCloseForm = useCallback(() => {
    setIsFormOpen(false);
    setSelectedDish(null);
  }, []);

```

```

const handleSubmit = useCallback(async (values: Record<string, unknown>) => {
  const input = {
    title: values.title as string,
    description: values.description as string,
    price: Number(values.price),
    menu: values.menu as string,
    status: values.status as DishStatus,
    photo: (values.photo as string) || "",
  };

  if (selectedDish) {
    await updateDish(selectedDish._id, input);
  } else {
    await createDish(input);
  }

  handleCloseForm();
  onRefetch();
}, [selectedDish, createDish, updateDish, handleCloseForm, onRefetch]);

const menuOptions = useMemo(() => [
  { value: 'ALL', label: 'Всі меню' },
  ...menus.map((menu) => ({ value: menu._id, label: menu.title })),
], [menus]);

return {
  isFormOpen,
  selectedDish,
  searchTerm,
  statusFilter,
  menuFilter,

```

```

    menuOptions,
    setSearchTerm,
    setStatusFilter,
    setMenuFilter,
    handleOpenCreate,
    handleOpenEdit,
    handleCloseForm,
    handleSubmit,
    isSubmitting: createLoading || updateLoading,
    isDeleting: removeLoading,
  };
};

// ===== zesty-service/src/basics/enums/common.enums.ts =====

import { registerEnumType } from '@nestjs/graphql';

export enum SalaryType {
  HOURLY = 'Hourly',
  MONTHLY = 'Monthly',
  SHIFT_BASED = 'Shift Based',
}

registerEnumType(SalaryType, { name: 'SalaryType' });

export enum ExpenseCategory {
  RENT = 'Rent',
  UTILITIES = 'Utilities',
  INGREDIENTS = 'Ingredients',
  MARKETING = 'Marketing',
  EQUIPMENT_MAINTENANCE = 'Equipment Maintenance',
}

```

```

    SALARIES = 'Salaries',
    OTHER = 'Other',
}

registerEnumType(ExpenseCategory, { name: 'ExpenseCategory' });

export enum IncomeCategory {
    SALES = 'Sales',
    CATERING = 'Catering',
    EVENTS = 'Events',
    OTHER = 'Other',
}

registerEnumType(IncomeCategory, { name: 'IncomeCategory' });

export enum StaffPosition {
    COOK = 'Cook',
    WAITER = 'Waiter',
    BARTENDER = 'Bartender',
    ADMIN = 'Admin',
}

registerEnumType(StaffPosition, { name: 'StaffPosition' });

export enum OrderStatus {
    OPEN = 'Open',
    IN_PROGRESS = 'In Pprogress',
    COMPLETED = 'Completed',
    CANCELLED = 'Cancelled',
}

registerEnumType(OrderStatus, { name: 'OrderStatus' });

```

```

export enum UserRole {
  GUEST = 'GUEST',
  SERVICE = 'SERVICE',
  COOK = 'COOK',
  BAR = 'BAR',
  ADMINISTRATION = 'ADMINISTRATION',
  ADMIN = 'ADMIN',
}

registerEnumType(UserRole, { name: 'UserRole' });

```