Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проєкту (роботи)
магістра
(ступінь вищої освіти)

на тему **ВЕБСИСТЕМА РЕСТОРАНУ З АДАПТИВНИМ ІНТЕРФЕЙСОМ ТА ІНТЕГРОВАНОЮ СИСТЕМОЮ УПРАВЛІННЯ КОНТЕНТОМ**

Виконав: студент 2 курсу, групи КНТ-514м
спеціальності
123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні системи та мережі

ДАНЧЕЄВ Р.Д.
(ПРІЗВИЩЕ та ініціали)
Керівник КАСЬЯН К.М.
(ПРІЗВИЩЕ та ініціали)
Рецензент ЩЕРБАКОВ В.І.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти магістерський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)

Освітня програма (спеціалізація) Комп'ютерні системи та мережі
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
Зав. кафедри Кудерметов Р.К.
_____
"_15_" жовтня 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ДАНЧЕЄВА Рауля Джантурайовича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) вебсистема ресторану з адаптивним інтерфейсом та інтегрованою системою управління контентом
керівник проєкту (роботи) кан. техн. наук, доцент, КАСЬЯН Костянтин Миколайович
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом вищого навчального закладу від " 30 " жовтня 2025 року № 491
2. Строк подання студентом проєкту (роботи)23 грудня 2025 року
3. Вихідні дані до проєкту (роботи) аналіз вимог користувачів вебсистеми ресторану (клієнти, адміністратори, кухарі, менеджери), дослідження існуючих рішень для автоматизації ресторанного бізнесу (Wix, WordPress, BentoBox, Toast), вимоги до продуктивності: час відгуку інтерфейсу не більше 100 мс, обробка замовлень у реальному часі, підтримка адаптивного інтерфейсу для пристроїв з розміром екрану від 320 до 2560 пікселів, інтеграція з платіжними системами (LiqPay, Fondy)
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1) Аналіз технічного завдання, дослідження наявних методів;
2) Розробка структури та моделі системи;
3) Реалізація вебсистеми;
4) Дослідження системи.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	КАСЬЯН К.М., доцент		
нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання 01.11.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз наявних методів обробки	05.10.2025 р.	
2	Розробка структури системи	10.10.2025 р.	
3	Розробка алгоритму обробки бронювань столиків	15.11.2025 р.	
4	Реалізація модулів системи	25.11.2025 р.	
5	Дослідження системи замовлень	30.11.2025 р.	
6	Оформлення отриманих результатів у ПЗ	05.12.2025 р.	
7	Оформлення графічного матеріалу	10.12.2025 р.	
8	Оформлення допоміжного матеріалу	15.12.2025 р.	

Студент _____ **Рауль ДАНЧЕЄВ**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ **Костянтин КАСЬЯН**
(підпис) (ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 86 стор., 48 рис., 8 табл., 19 джерел.

CMS, NODE.JS, POSTGRESQL, REACT, TYPESCRIPT, UML, АДАПТИВНИЙ ІНТЕРФЕЙС, БРОНЮВАННЯ СТОЛІВ, ВЕБСИСТЕМА, ОНЛАЙН-ЗАМОВЛЕННЯ, РЕСТОРАН, УПРАВЛІННЯ КОНТЕНТОМ

Об'єкт дослідження - процеси управління онлайн-присутністю та автоматизації діяльності ресторанів.

Предмет дослідження - методи розробки вебсистем з адаптивним інтерфейсом, технології управління контентом та архітектурні рішення для масштабованості.

Мета роботи - розробка вебсистеми управління рестораном з адаптивним інтерфейсом та CMS для автоматизації онлайн-замовлення, бронювання і управління меню.

У першому розділі проаналізовано проблеми управління онлайн-присутністю ресторанів, досліджено React, TypeScript, Node.js та порівняно існуючі рішення автоматизації.

У другому розділі спроєктовано тришарову архітектуру (React-Node.js-PostgreSQL) та розроблено UML-діаграми бізнес-процесів онлайн-замовлення, бронювання та управління меню.

Третій розділ присвячено реалізації: створено адаптивний інтерфейс на React з TypeScript, RESTful API на Node.js та інтегровано CMS для оновлення меню.

У четвертому розділі досліджено ефективність TypeScript для типобезпеки критичних операцій та проаналізовано продуктивність і адаптивність системи.

ABSTRACT

Explanatory note to the master's work: 86 p., 48 fig., 8 tabl., 19 sources.

ADAPTIVE INTERFACE, CMS, CONTENT MANAGEMENT, NODE.JS, ONLINE ORDERING, POSTGRESQL, REACT, RESTAURANT, TABLE BOOKING, TYPESCRIPT, UML, WEB SYSTEM

Research object - processes of managing online presence and automating restaurant operations.

Research subject - methods for developing web systems with adaptive interface, content management technologies and architectural solutions for scalability.

Research goal - development of restaurant management web system with adaptive interface and CMS for automating online ordering, booking and menu management.

The first section analyzes restaurant online presence management challenges, examines React, TypeScript, Node.js and compares existing automation solutions.

The second section designs three-tier architecture (React-Node.js-PostgreSQL) and develops UML diagrams for online ordering, booking and menu management business processes.

The third section focuses on implementation: adaptive interface created using React with TypeScript, RESTful API on Node.js and CMS integrated for menu updates.

The fourth section researches TypeScript effectiveness for critical operations type safety and analyzes system performance and adaptivity.

ЗМІСТ

ВСТУП

Актуальність теми. Стрімкий розвиток цифрових технологій та зростаюча конкуренція у ресторанному бізнесі висувають дедалі вищі вимоги до якості онлайн-присутності закладів. За даними міжнародних досліджень ринку харчування, понад сімдесят відсотків потенційних відвідувачів приймають рішення про відвідування ресторану на основі інформації, отриманої з вебсайту або мобільного додатку. Сучасний вебсайт ресторану є не просто візитівкою закладу, а ключовим інструментом для залучення клієнтів, демонстрації меню, управління бронюваннями та обробки онлайн-замовлень. Водночас традиційні підходи до створення та управління вебсайтами ресторанів демонструють суттєві недоліки, що знижують їх ефективність як маркетингового інструменту та ускладнюють щоденну операційну діяльність персоналу закладу [1].

Багато існуючих рішень для ресторанного бізнесу характеризуються значними обмеженнями функціональності та зручності використання. Шаблонні платформи демонструють недостатню гнучкість та обмежені можливості кастомізації, що унеможливлює повноцінне відображення унікальності закладу та його корпоративного стилю. Застарілі системи часто не адаптовані для мобільних пристроїв, що призводить до втрати значної частини потенційних клієнтів у контексті масового поширення смартфонів та планшетів. Більшість комерційних рішень, таких як Wix, WordPress з тематичними шаблонами для ресторанів та спеціалізовані платформи на кшталт BentoBox або Toast, вимагають технічних навичок для налаштування або надають обмежений функціонал безкоштовних версій [2].

Ключові проблеми, що потребують вирішення, включають відсутність зручних інструментів управління контентом, які б дозволяли персоналу ресторану без спеціальних технічних навичок оперативно оновлювати меню, додавати нові позиції страв з детальними описами та фотографіями, встановлювати актуальні ціни, публікувати інформацію про акції та спеціальні

пропозиції. Складність інтеграції різних функціональних модулів, таких як система онлайн-замовлень, модуль бронювання столів, інструменти збору відгуків клієнтів та аналітики відвідуваності, створює потребу в залученні зовнішніх фахівців для кожної зміни функціональності вебсайту. Брак єдиної інтегрованої платформи для управління всіма аспектами онлайн-присутності призводить до фрагментації даних між різними системами та ускладнює процес прийняття управлінських рішень [3].

Наявні рішення для автоматизації ресторанного бізнесу зазвичай розділені на окремі категорії продуктів, що обслуговують різні аспекти діяльності закладу без забезпечення їх глибокої інтеграції. Системи управління замовленнями працюють ізольовано від модулів управління складом інгредієнтів, платформи для онлайн-бронювання не інтегровані з системами аналітики завантаженості закладу, а інструменти управління контентом вебсайту потребують окремого адміністрування без зв'язку з операційними даними ресторану. Така фрагментація технологічного стеку ускладнює роботу персоналу, підвищує вартість впровадження та супроводу інформаційних систем, а також збільшує ймовірність виникнення помилок через необхідність дублювання даних між різними платформами [4].

Розробка інтегрованої вебсистеми з адаптивним інтерфейсом та потужною системою управління контентом дозволить ресторанам ефективно управляти своєю онлайн-присутністю, автоматизувати процеси прийому замовлень та бронювання столів, оптимізувати управління меню та складськими залишками інгредієнтів, покращити взаємодію з клієнтами через зручні інструменти зворотного зв'язку, формувати аналітичні звіти для прийняття обґрунтованих управлінських рішень та знижувати операційні витрати шляхом скорочення потреби у залученні зовнішніх технічних фахівців для оновлення контенту вебсайту.

Мета і задачі дослідження. Метою роботи є проєктування та розробка комплексної вебсистеми для управління рестораном, яка поєднує адаптивний користувацький інтерфейс з інтегрованою системою управління контентом, що

забезпечує ефективну взаємодію між клієнтами та закладом, автоматизацію процесів онлайн-замовлення та бронювання столів, управління меню та складськими запасами, а також надає персоналу зручні інструменти для оновлення інформації на вебсайті без залучення спеціалізованих технічних фахівців з використанням сучасних вебтехнологій та архітектурних патернів.

Для досягнення поставленої мети необхідно вирішити наступні задачі дослідження. По-перше, провести комплексний аналіз вимог усіх категорій користувачів системи, що включає клієнтів закладу, адміністративний персонал, кухарів та менеджерів, з метою формування повного переліку функціональних можливостей та критеріїв якості вебсистеми. По-друге, виконати порівняльний аналіз існуючих рішень для автоматизації ресторанного бізнесу з виявленням їх переваг та недоліків, що дозволить визначити ключові вимоги до функціональності проєктованої системи. По-третє, спроєктувати архітектуру вебсистеми з визначенням структури бази даних, взаємодії програмних компонентів, інтерфейсів прикладного програмування та обґрунтованим вибором технологічного стеку, що забезпечить масштабованість, продуктивність та зручність супроводу системи.

По-четверте, розробити набір UML-діаграм для детального моделювання бізнес-процесів ресторану, включаючи процеси онлайн-замовлення страв з вибором способу доставки та оплати, бронювання столів з автоматичною перевіркою доступності, обробки замовлень на кухні з відстеженням статусів виконання, управління меню з контролем доступності страв на основі залишків інгредієнтів, обробки платежів через різні платіжні системи та формування аналітичних звітів про діяльність закладу. По-п'яте, дослідити ефективність застосування TypeScript для забезпечення типобезпеки критичних операцій системи, таких як розрахунки цін та сум замовлень, застосування знижок та акційних пропозицій, управління станами замовлень та валідація даних користувачів. По-шосте, спроєктувати адаптивний користувацький інтерфейс системи з урахуванням принципів User Experience Design та забезпечення

зручності використання на різних типах пристроїв від настільних комп'ютерів до смартфонів [5].

Об'єкт і предмет дослідження. Об'єктом дослідження виступають процеси управління онлайн-присутністю та автоматизації операційної діяльності ресторанних закладів, що охоплюють взаємодію з клієнтами через вебінтерфейс, управління замовленнями та бронюваннями, облік запасів інгредієнтів на складі, адміністрування контенту вебсайту без залучення технічних спеціалістів та формування аналітичної звітності для підтримки прийняття управлінських рішень.

Предметом дослідження є методи та інструменти розробки вебсистем з адаптивним користувацьким інтерфейсом, технології управління контентом для ресторанного бізнесу, архітектурні рішення для забезпечення масштабованості та продуктивності розподілених систем, підходи до типобезпечного програмування з використанням TypeScript для підвищення надійності критичних бізнес-операцій, методи проєктування бази даних для ефективного зберігання та обробки інформації про меню, замовлення, клієнтів та операційну діяльність ресторану.

Практичне значення роботи. Практичне значення роботи полягає у створенні теоретичної та методологічної бази для проєктування та впровадження сучасних вебсистем управління ресторанним бізнесом. Результати дослідження можуть бути використані закладами харчування різних форматів для модернізації їх онлайн-присутності, автоматизації бізнес-процесів та підвищення якості обслуговування клієнтів. Розроблені архітектурні рішення, моделі бізнес-процесів та методологічні підходи можуть слугувати основою для створення аналогічних систем у суміжних галузях сфери обслуговування, де критично важливим є поєднання зручного користувацького інтерфейсу з потужними інструментами управління операційною діяльністю.

Дослідження ефективності застосування TypeScript у контексті вебсистем ресторану вносить вклад у розуміння можливостей використання статично типізованих мов програмування для підвищення надійності та якості складних

вебзастосунків з інтенсивною обробкою бізнес-логіки. Розроблені UML-діаграми та моделі даних можуть використовуватися як референсні зразки при проєктуванні інформаційних систем для ресторанного бізнесу, забезпечуючи уніфікований підхід до опису бізнес-процесів та технічної архітектури. Методологія інтеграції різнорідних функціональних модулів у єдину систему має цінність для розробників корпоративних інформаційних систем, що потребують узгодженої роботи компонентів різного призначення.

1 АНАЛІТИЧНА ЧАСТИНА

1.1 Огляд проблематики управління онлайн-присутністю ресторанів

Сучасні технології електронної комерції та цифрового маркетингу трансформували підходи до управління ресторанним бізнесом, створюючи нові можливості для залучення клієнтів та оптимізації операційних процесів. За даними дослідження компанії National Restaurant Association, понад сімдесят відсотків споживачів використовують онлайн-платформи для пошуку інформації про заклади харчування перед прийняттям рішення про відвідування, причому шістдесят п'ять відсотків з них здійснюють пошук через мобільні пристрої. Традиційні методи просування ресторанів через друковані матеріали, рекламу у пресі та усні рекомендації поступаються цифровим каналам комунікації, що вимагає від власників закладів активної присутності в онлайн-просторі з функціональними та зручними вебсайтами [1].

Ключові проблеми традиційних підходів до створення вебсайтів ресторанів включають відсутність централізованого управління контентом, що вимагає технічних навичок для оновлення меню та цін. Персонал змушений звертатися до веброзробників для внесення навіть незначних змін у контент, що призводить до затримок в актуалізації інформації та додаткових фінансових витрат. Складність інтеграції функціональних модулів онлайн-замовлення та бронювання столів створює бар'єри для автоматизації взаємодії з клієнтами. Багато закладів використовують окремі платформи для різних завдань, таких як прийом замовлень через телефонні дзвінки, бронювання через соціальні мережі та оплата готівкою при отриманні, що ускладнює облік та аналіз діяльності [2].

Обмежені можливості адаптації інтерфейсу до різних типів пристроїв призводять до втрати потенційних клієнтів, оскільки користувачі смартфонів та планшетів стикаються з незручним відображенням контенту, складною навігацією та проблемами з читабельністю тексту на малих екранах. Відсутність єдиної платформи для збору та аналізу даних про поведінку відвідувачів,

популярність страв, завантаженість закладу у різний час та ефективність маркетингових акцій ускладнює прийняття обґрунтованих управлінських рішень для оптимізації роботи ресторану. Брак інструментів для автоматизованого управління запасами інгредієнтів на основі даних про замовлення призводить до ситуацій, коли клієнти замовляють страви, яких фактично немає в наявності через закінчення необхідних компонентів [3].

Рисунок 1.1 демонструє типовий процес взаємодії клієнта з рестораном у традиційній моделі та в умовах використання інтегрованої вебсистеми, що наочно ілюструє переваги автоматизації ключових бізнес-процесів закладу.

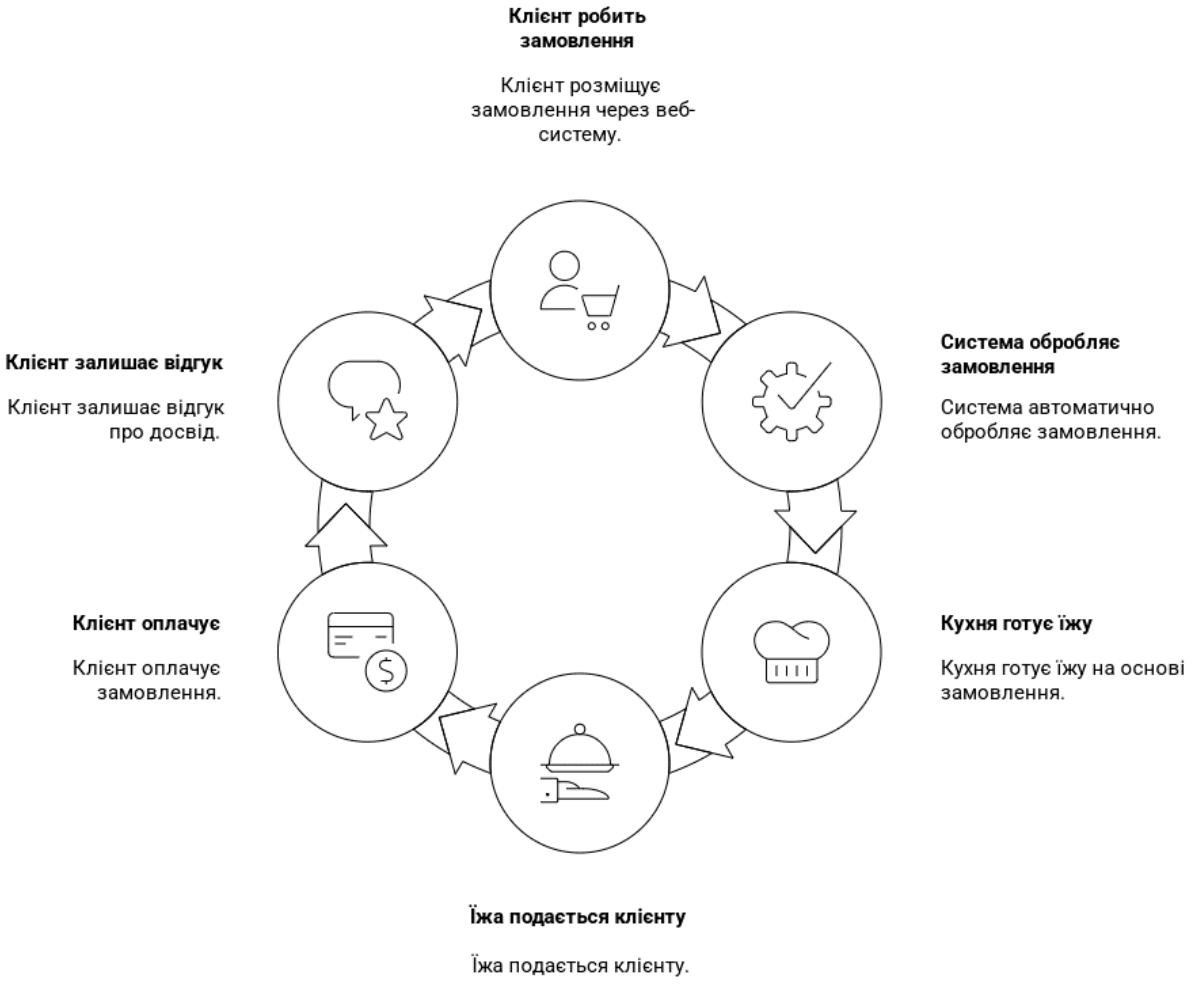

Рисунок 1.1 – Типовий процес взаємодії клієнта з рестораном

Сучасний ринок програмних рішень для ресторанного бізнесу представлений декількома категоріями продуктів, що відрізняються за функціональністю, складністю впровадження та вартістю володіння.

Комплексний аналіз провідних платформ дозволяє виявити переваги та обмеження існуючих підходів до автоматизації управління закладами харчування та визначити напрямки для розробки нових рішень, що краще відповідають потребам власників та персоналу ресторанів [4].

1.2 Технічні основи сучасних вебтехнологій

Розробка сучасних вебсистем базується на комплексі взаємопов'язаних технологій, що забезпечують створення інтерактивних користувацьких інтерфейсів, ефективну обробку даних на сервері та надійне зберігання інформації в базах даних. Клієнтська частина вебзастосунків будується на тріаді базових технологій, серед яких HTML5 забезпечує семантичну структуру веб документів з підтримкою мультимедійного контенту, геолокації та офлайн-режиму роботи. Мова розмітки визначає логічну структуру сторінки через систему вкладених елементів, що описують заголовки, параграфи, списки, таблиці, форми введення даних та інші компоненти інтерфейсу [5].

Cascading Style Sheets третьої версії відповідає за візуальне представлення веб документів, дозволяючи розробникам визначати кольорові схеми, типографіку, компонування елементів на сторінці, анімації та адаптацію інтерфейсу до різних розмірів екранів через медіа-запити. Сучасний CSS3 підтримує флексбокс та грід-системи для гнучкого позиціонування елементів, змінні для централізованого управління дизайн-токенами, складні селектори для точного визначення стилів елементів та псевдокласи для інтерактивних ефектів. JavaScript виступає мовою програмування для клієнтської частини, що забезпечує динамічну поведінку вебсторінок через маніпуляцію Document Object Model, обробку подій користувача, асинхронну комунікацію з сервером через AJAX та Fetch API, валідацію даних форм перед відправкою на сервер [6].

React являє собою бібліотеку для побудови користувацьких інтерфейсів на основі компонентного підходу, розроблену компанією Facebook для власних потреб та згодом відкриту для широкого використання. Основною концепцією React є розбиття інтерфейсу на незалежні повторно використовувані компоненти, кожен з яких інкапсулює власну логіку та візуальне представлення. Віртуальний DOM забезпечує оптимізацію продуктивності через мінімізацію операцій з реальним DOM браузера, оскільки React порівнює попередній та новий стани віртуального дерева і застосовує лише необхідні зміни до реального DOM. Односпрямований потік даних від батьківських компонентів до дочірніх через props та управління станом через hooks, такі як useState та useEffect, забезпечують передбачувану поведінку застосунку та спрощують відлагодження [7].

Node.js представляє середовище виконання JavaScript на сервері, побудоване на движку V8 від Google Chrome, що дозволяє використовувати єдину мову програмування для клієнтської та серверної частин застосунку. Асинхронна неблокуюча модель введення-виведення забезпечує високу продуктивність при обробці великої кількості одночасних з'єднань, що є критично важливим для вебзастосунків реального часу. Менеджер пакетів npm надає доступ до найбільшої у світі екосистеми бібліотек з відкритим вихідним кодом, що охоплюють всі аспекти серверної розробки від вебфреймворків до драйверів баз даних. Express.js виступає мінімалістичним вебфреймворком для Node.js, що надає зручні інструменти для маршрутизації HTTP-запитів, обробки middleware, рендерингу шаблонів та інтеграції з різними системами зберігання даних [8].

TypeScript розширює JavaScript системою статичної типізації, що дозволяє виявляти помилки на етапі розробки та компіляції замість виконання програми. Анотації типів для змінних, параметрів функцій та об'єктів забезпечують самодокументування коду та покращують інтелектуальну підтримку в середовищах розробки через автодоповнення та рефакторинг. Інтерфейси та типи дозволяють визначати контракти для структур даних та

функцій, що особливо цінно для великих проєктів з декількома розробниками. Generics забезпечують створення повторно використовуваних компонентів, що працюють з різними типами даних зі збереженням типобезпеки. Компілятор TypeScript транслює код у чистий JavaScript з підтримкою різних версій стандарту ECMAScript, що забезпечує сумісність з різними браузерами та середовищами виконання [9].

Системи управління базами даних забезпечують надійне зберігання та ефективну обробку структурованої інформації вебзастосунків. Реляційні бази даних, такі як PostgreSQL та MySQL, організують дані у вигляді таблиць зі встановленими зв'язками між ними, забезпечуючи підтримку транзакцій ACID для критичних операцій, складні запити через мову SQL та механізми забезпечення цілісності даних через обмеження та тригери. NoSQL бази даних, зокрема MongoDB, пропонують більш гнучкі схеми зберігання даних у форматі документів JSON, що полегшує роботу з нерегулярними структурами та забезпечує горизонтальне масштабування для високонавантажених застосунків. Вибір між реляційними та документоорієнтованими базами залежить від характеру даних, вимог до консистентності та очікуваних навантажень на систему [10].

1.3 Порівняльний аналіз існуючих програмних рішень

Ринок програмного забезпечення для ресторанного бізнесу представлений широким спектром рішень різного рівня складності та функціональності. Wix Restaurants являє собою конструктор вебсайтів з готовими шаблонами для закладів харчування, що дозволяє створювати веб присутність без технічних навичок програмування. Платформа надає візуальний редактор для налаштування дизайну, вбудовані інструменти для створення меню з фотографіями страв та цінами, форми для прийому онлайн-замовлень з

інтеграцією популярних платіжних систем. Проте система має суттєві обмеження щодо кастомізації функціональності, оскільки користувачі обмежені можливостями, передбаченими платформою, без можливості додавання власної бізнес-логіки [11].

Залежність від хостингу провайдера створює ризики простою при технічних проблемах на стороні Wix, обмежені можливості інтеграції зі сторонніми системами управління рестораном ускладнюють побудову єдиної екосистеми, а щомісячна абонентська плата зростає при підключенні додаткових функцій, таких як онлайн-замовлення або просунута аналітика відвідувачів. WordPress з плагіном WooCommerce Restaurant Ordering представляє більш гнучке рішення на базі популярної системи управління контентом з відкритим вихідним кодом. Величезна екосистема плагінів дозволяє розширювати функціональність вебсайту практично необмеженими способами, повний контроль над кодом надає можливість реалізації будь-якої специфічної бізнес-логіки, а відсутність прив'язки до конкретного хостинг-провайдера забезпечує свободу вибору інфраструктури [12].

Недоліками WordPress для ресторанів є необхідність технічних навичок для налаштування та підтримки системи, потенційні проблеми безпеки через велику кількість плагінів від різних розробників, складність забезпечення високої продуктивності при інтенсивному навантаженні та необхідність регулярного оновлення ядра системи і плагінів для закриття вразливостей. Toast позиціонується як комплексна платформа для управління рестораном, що інтегруєPoint of Sale систему, онлайн-замовлення, управління персоналом та аналітику в єдиному рішенні. Глибока інтеграція всіх модулів забезпечує синхронізацію даних між різними аспектами діяльності закладу, спеціалізовані функції для ресторанного бізнесу включають управління столами, розподіл чайових між персоналом та інтеграцію з кухонними дисплеями [13].

Проте Toast має високу вартість володіння через комісії з транзакцій та абонентську плату, залежність від екосистеми Toast ускладнює міграцію на альтернативні рішення у майбутньому, обов'язкове використання апаратного

забезпечення від Toast збільшує первинні інвестиції, а складність системи вимагає тривалого навчання персоналу для ефективного використання всіх можливостей платформи. BentoBox фокусується виключно на вебсайтах для ресторанів з акцентом на дизайн та користувацький досвід. Професійні шаблони розроблені спеціально для індустрії харчування з урахуванням кращих практик презентації меню та візуального контенту, вбудовані інструменти SEO оптимізації покращують видимість закладу в пошукових системах, інтеграція з популярними платформами резервування столів спрощує процес бронювання для клієнтів [14].

Обмеження BentoBox включають фокус лише на вебсайтах без повноцінної системи управління операціями ресторану, високу вартість послуг, орієнтовану на преміум-сегмент закладів, обмежену гнучкість кастомізації порівняно з розробкою на замовлення та відсутність глибокої інтеграції з системами управління запасами та фінансовим обліком. Детальне порівняння функціональних можливостей розглянутих систем управління ресторанами наведено в таблиці 1.1, що дозволяє систематизувати переваги та недоліки кожного підходу.

Таблиця 1.1 - Порівняння функціональних можливостей систем управління ресторанами

Критерій	Wix	WordPress	Toast	BentoBox	Проєктована
1	2	3	4	5	6
Управління меню	Базове	Через плагіни	Розширене	Професійне	Повне з CMS
Онлайн-замовлення	Так	Через WooCommerce	Так, повне	Інтеграція	Так, нативне
Бронювання столів	Обмежене	Через плагіни	Так	Інтеграція	Так, нативне

Продовження таблицы 1.1

1	2	3	4	5	6
Управління складом	Ні	Обмежене	Так, повне	Ні	Так
Адаптивність	Автоматична	Залежить від теми	Так	Повна	Повна
Вартість впровадження	Низька	Середня	Висока	Висока	Середня
Гнучкість кастомізації	Низька	Висока	Середня	Середня	Повна

Аналіз виявлених переваг та недоліків існуючих рішень демонструє відсутність на ринку комплексної системи, що поєднує автоматичне управління контентом без необхідності технічних навичок програмування, інтеграцію модулів замовлення, бронювання та управління складом у єдиній платформі, повну гнучкість кастомізації функціональності відповідно до специфічних потреб закладу, помірну вартість впровадження та супроводу, доступну для закладів малого та середнього розміру. Це обґрунтовує необхідність розробки спеціалізованої вебсистеми для управління рестораном, що враховує виявлені обмеження комерційних рішень та пропонує оптимальний баланс між функціональністю, зручністю використання та економічною ефективністю [15].

1.4 Системи управління контентом

Системи управління контентом являють собою програмні платформи, що забезпечують створення, редагування, організацію та публікацію цифрового контенту без необхідності безпосереднього програмування HTML-коду або роботи з файлами на сервері. Основною метою CMS є розділення контенту від

представлення, дозволяючи нетехнічним користувачам управляти інформацією на вебсайті через інтуїтивний адміністративний інтерфейс. Архітектура типової системи управління контентом складається з декількох взаємопов'язаних компонентів, що включають базу даних для зберігання контенту, метаданих та налаштувань системи, шар бізнес-логіки для обробки запитів, валідації даних та застосування правил публікації, систему шаблонів для відокремлення структури контенту від його візуального представлення, адміністративний інтерфейс для управління контентом та налаштуваннями системи, систему прав доступу для розмежування можливостей різних ролей користувачів [16].

Традиційні CMS, такі як WordPress, Joomla та Drupal, побудовані за монолітною архітектурою, де фронтенд та бекенд тісно інтегровані в єдину систему з жорстким зв'язком між шарами даних, логіки та представлення. Контент зберігається в реляційній базі даних з нормалізованою структурою таблиць, обробляється на сервері мовою PHP або іншою серверною мовою програмування, та рендериться в HTML-сторінки з використанням системи шаблонів перед відправкою клієнту через протокол HTTP. Перевагами монолітних CMS є простота впровадження через готові рішення з мінімальними вимогами до початкового налаштування, велика екосистема плагінів та тем для розширення функціональності без програмування, наявність вбудованого адміністративного інтерфейсу для управління всіма аспектами сайту, підтримка багатомовності контенту та локалізації інтерфейсу [17].

Недоліками монолітних систем виступають обмежена гнучкість у виборі технологій фронтенду через прив'язку до системи шаблонів CMS, складність масштабування через монолітну архітектуру з єдиною кодовою базою, проблеми продуктивності при високому навантаженні через серверний рендеринг всіх сторінок на кожний запит, складність інтеграції з іншими системами через відсутність стандартизованого API. Headless CMS представляють сучасний підхід, де бекенд системи управління контентом повністю відокремлений від фронтенду та надає контент через RESTful API або GraphQL без зв'язку з конкретною технологією представлення. Контент зберігається в

структурованому форматі як набір даних без інформації про візуальне відображення, що дозволяє використовувати його в різних каналах доставки, включаючи вебсайти з різними фреймворками, мобільні додатки для iOS та Android, голосові асистенти типу Alexa та Google Assistant, IoT-пристрої з обмеженими можливостями відображення [18].

Розробники мають повну свободу у виборі технологій фронтенду, таких як React для одно сторінкових застосунків з віртуальним DOM, Vue.js для прогресивного покращення існуючих сайтів, Angular для корпоративних застосунків з TypeScript, статичні генератори сайтів типу Gatsby або Next.js для оптимальної продуктивності через pre-rendering. Приклади headless CMS включають Strapi як відкриту самохостовану платформу з гнучкою системою плагінів, Contentful як хмарне рішення з потужним API та CDN для швидкої доставки контенту, Sanity з реального часу співпрацею над контентом та структурованими даними, Ghost для публікації контенту з фокусом на блогах та newsletter. Ці системи надають потужні API для CRUD-операцій з контентом, гнучкі схеми даних для визначення власних типів контенту та зв'язків між ними, вбудовані інструменти для версіювання контенту та відкату до попередніх версій [19].

1.5 Методи забезпечення адаптивності вебінтерфейсів

Адаптивний вебдизайн являє собою підхід до створення вебсайтів, що автоматично пристосовуються до різних розмірів екранів та орієнтацій пристроїв, забезпечуючи оптимальний користувацький досвід на настільних комп'ютерах з великими моніторами, ноутбуках з екранами середнього розміру, планшетах з сенсорними екранами та смартфонах з малими дисплеями. Концепція адаптивного дизайну базується на трьох основних принципах, що включають гнучкі сітки на основі відносних одиниць вимірювання типу

відсотків або em замість фіксованих пікселів, гнучкі зображення та медіа-контент, що масштабуються відповідно до розміру контейнера через властивість max-width, медіа-запити CSS для застосування різних стилів залежно від характеристик пристрою таких як ширина екрану, орієнтація та щільність пікселів [2].

Важливість адаптивності для ресторанних вебсайтів підтверджується статистикою, згідно з якою понад шістдесят п'ять відсотків онлайн-замовлень їжі здійснюються через мобільні пристрої, причому користувачі очікують швидкого завантаження сторінок за менше трьох секунд та зручної навігації однією рукою. Медіа-запити CSS дозволяють визначати точки перелому (breakpoints), де макет вебсайту змінюється для оптимального відображення на екранах різних розмірів. Типові точки перелому включають мобільні пристрої з шириною екрану до 768 пікселів, де контент відображається в одну колонку з великими інтерактивними елементами розміром не менше 44x44 пікселів для зручності сенсорного введення, вертикальною навігацією у вигляді hamburger-меню для економії простору, планшети з шириною від 768 до 1024 пікселів, де можливе використання двоколонкового макету з основним контентом та бічною панеллю, горизонтальної навігації з випадаючими підменю, настільні комп'ютери з шириною понад 1024 пікселів, де доступний повний багатоколонковий макет з розширеною навігацією та додатковими елементами інтерфейсу [3].

Підхід mobile-first передбачає проєктування інтерфейсу спочатку для мобільних пристроїв з обмеженим простором екрану та сенсорним введенням, а потім поступове додавання функціональності та елементів для більших екранів через прогресивне покращення. Це забезпечує фокус на найважливішому контенті та функціях, оптимізацію продуктивності через мінімізацію розміру сторінки для повільних мобільних мереж, кращу доступність через простоту навігації та взаємодії. Флексбокс-модель CSS надає потужні інструменти для створення гнучких одновимірних макетів, де елементи автоматично розподіляються по головній осі контейнера з можливістю вирівнювання по різним краям, зміни порядку відображення без зміни HTML-структури,

автоматичного розтягування або стиснення для заповнення доступного простору пропорційно заданим коефіцієнтам [4].

Властивості flex-direction визначають напрямок головної осі контейнера як горизонтальний для row або вертикальний для column, justify-content контролює вирівнювання елементів вздовж головної осі з опціями центрування, прижимання до країв або рівномірного розподілу, align-items керує вирівнюванням по перпендикулярній осі з можливістю розтягування на всю висоту контейнера, flex-wrap дозволяє елементам переноситися на новий рядок при нестачі місця замість стиснення. CSS Grid забезпечує двовимірне компонування елементів з точним контролем розташування в рядках та стовпцях одночасно, визначенням розмірів через фіксовані значення в пікселях, відсоткові співвідношення від розміру контейнера або гнучкі одиниці fr для пропорційного розподілу простору, створенням іменованих областей сітки для семантичного позиціонування елементів без залежності від їх порядку в HTML [5].

1.6 Постановка задачі дослідження

На основі проведеного аналізу проблематики управління онлайн-присутністю ресторанів, технічних основ сучасних вебтехнологій, порівняння існуючих програмних рішень для автоматизації ресторанного бізнесу, вивчення систем управління контентом та методів забезпечення адаптивності вебінтерфейсів сформульовано наступну постановку задачі дослідження. Необхідно спроєктувати та розробити комплексну вебсистему для управління рестораном, що забезпечує автоматизацію ключових бізнес-процесів закладу через єдину інтегровану платформу з адаптивним користувацьким інтерфейсом, потужною системою управління контентом без необхідності залучення технічних спеціалістів для оновлення інформації та інтеграцією всіх

функціональних модулів для забезпечення синхронізації даних в реальному часі [1].

Функціональні вимоги до вебсистеми включають реалізацію модуля онлайн-замовлення з можливістю перегляду меню з фільтрацією за категоріями страв, дієтичними обмеженнями та цінами, формування кошика з додаванням та видаленням позицій з автоматичним розрахунком загальної вартості, вибору способу отримання замовлення через доставку за вказаною адресою з розрахунком вартості або самовивіз з ресторану без додаткової оплати, вибору методу оплати через онлайн-платіжні системи типу LiqPay або Fondy або готівкою при отриманні, відстеження статусу замовлення в реальному часі від прийняття до доставки. Модуль бронювання столів має забезпечувати вибір дати, часу та кількості гостей з візуальним календарем доступності, автоматичну перевірку доступності столів на основі поточних бронювань та часових обмежень, можливість вказання спеціальних побажань щодо розташування столу та додаткових послуг, отримання автоматичних підтверджень через електронну пошту та SMS-повідомлення, нагадувань за день та годину до бронювання [2].

Система управління контентом повинна надавати інструменти для управління меню з можливістю додавання нових страв з назвами українською та англійською мовами, детальними описами складу та способу приготування, цінами з підтримкою різних валют, високоякісними зображеннями з автоматичною оптимізацією для різних пристроїв, маркуванням алергенів та дієтичних властивостей, редагування існуючих позицій через інтуїтивний візуальний редактор без необхідності знання HTML або CSS, тимчасового приховування або остаточного видалення страв з можливістю відновлення, організації страв за ієрархією категорій з підтримкою необмеженої вкладеності, масового оновлення цін за категоріями або тегами. Управління складом інгредієнтів має включати ведення централізованої бази інгредієнтів з одиницями вимірювання та постачальниками, автоматичне відстеження поточних залишків на основі даних про замовлення та використання, визначення

мінімальних порогів запасів для кожного інгредієнта, автоматичні сповіщення персоналу про необхідність поповнення через електронну пошту, автоматичне позначення страв як недоступних при закінченні будь-якого з критичних інгредієнтів [3].

Адміністративна панель для персоналу ресторану повинна забезпечувати керування замовленнями з переглядом списку всіх замовлень у табличному форматі, фільтрацією за статусом виконання, датою створення, способом оплати та доставки, пошуком за номером замовлення або контактними даними клієнта, зміною статусів замовлень з автоматичним логуванням часу переходів між станами, друком чеків для кухні з деталями приготування та чеків для клієнтів з повною вартістю, прямим контактом з клієнтами через інтегровані канали комунікації для уточнення деталей. Керування бронюваннями включає перегляд графіка бронювань у календарному вигляді з візуалізацією завантаженості по часових слотах, фільтрацію за датою, часом, кількістю гостей та статусом підтвердження, швидке підтвердження або скасування бронювань з автоматичним повідомленням клієнтів через обраний канал, прямий зв'язок з клієнтами для уточнення спеціальних побажань або зміни деталей бронювання, експорт графіка бронювань для друку або інтеграції з іншими системами [4].

Модуль аналітики має формувати детальні звіти про виручку за різні періоди часу з розбивкою по днях, тижнях та місяцях, популярність страв на основі кількості замовлень з ідентифікацією бестселерів та непопулярних позицій, завантаженість закладу по днях тижня та годинах роботи для оптимізації графіків персоналу, ефективність маркетингових акцій через відстеження використання промокодів та їх вплив на виручку, середній чек замовлення та його зміну в часі, конверсію відвідувачів вебсайту в клієнтів через воронку замовлення. Технічні вимоги до системи визначають використання React для побудови адаптивного клієнтського інтерфейсу з компонентним підходом для повторного використання коду, віртуальним DOM для оптимізації продуктивності рендерингу, hooks для управління станом та побічними ефектами, TypeScript для забезпечення типобезпеки критичних операцій з

розрахунками цін, застосуванням знижок та обробкою платіжних даних, автодоповнення коду в IDE та раннього виявлення помилок на етапі розробки [5].

Серверна частина має базуватися на Node.js з Express.js для створення RESTful API з підтримкою асинхронної обробки запитів, middleware для логування, автентифікації та обробки помилок, маршрутизації запитів до відповідних контролерів. PostgreSQL виступає системою управління базою даних для надійного зберігання структурованих даних з підтримкою ACID транзакцій для критичних операцій типу створення замовлень та обробки платежів, складних запитів через SQL для аналітичних звітів, забезпечення цілісності даних через обмеження зовнішніх ключів та тригери. Система повинна забезпечувати повну адаптивність інтерфейсу для коректного відображення на пристроях з різними розмірами екранів від 320 пікселів для старих смартфонів до 2560 пікселів для великих моніторів, автоматичне налаштування розміру та розташування елементів через CSS Grid та Flexbox, оптимізацію зображень для різних щільностей пікселів через srcset [1].

2 МОДЕЛЬНА ЧАСТИНА

2.1 Концептуальна модель вебсистеми управління рестораном

Концептуальна модель вебсистеми управління рестораном визначає загальну структуру та основні компоненти системи без деталізації технічних аспектів реалізації. Модель базується на класичній тришаровій архітектурі, що забезпечує чітке розділення відповідальності між шарами представлення, бізнес-логіки та даних. Такий підхід дозволяє незалежно розвивати кожний шар, спрощує тестування окремих компонентів, полегшує масштабування системи через горизонтальне дублювання серверів на рівні бізнес-логіки, забезпечує можливість заміни технологій в одному шарі без впливу на інші. Рисунок 2.1

демонструє високорівневу архітектуру вебсистеми з виділенням трьох основних шарів та їх взаємодії .

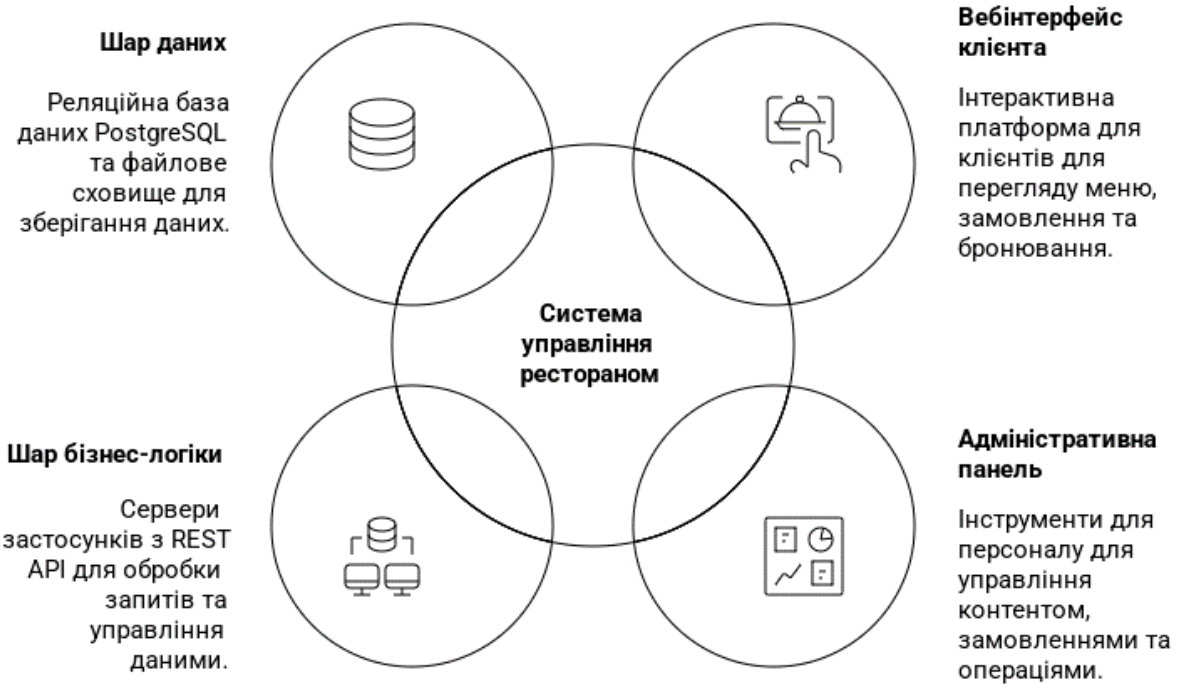

Рисунок 2.1 - Високорівнева архітектура вебсистеми

Шар представлення відповідає за взаємодію з користувачами системи через графічний інтерфейс, що реалізований як односторінковий вебзастосунок на базі React з адаптивним дизайном для коректного відображення на пристроях різних типів. Клієнтська частина для відвідувачів ресторану включає головну сторінку з інформацією про заклад, фотографіями інтер'єру та відгуками клієнтів, сторінку меню з категоризованим списком страв, фільтрами за дієтичними обмеженнями та діапазонами цін, детальними картками страв з великими зображеннями, описами складу та можливістю додавання до кошика, сторінку кошика з переліком обраних позицій, можливістю зміни кількості порцій, вибором способу отримання замовлення через доставку або самовивіз, розрахунком загальної вартості з урахуванням вартості доставки, сторінку оформлення замовлення з формами введення контактних даних, адреси доставки, вибору методу оплати та часу доставки, сторінку бронювання столів з

календарем доступності, формою вказання кількості гостів та спеціальних побажань.

Адміністративна панель для персоналу закладу реалізована як окремий інтерфейс з автентифікацією та розмежуванням прав доступу за ролями користувачів. Менеджери отримують доступ до панелі управління меню з можливістю додавання нових категорій та страв через візуальний редактор, редагування існуючих позицій з оновленням назв, описів, цін та зображень, тимчасового приховування страв при закінченні інгредієнтів або виведення з меню, масового оновлення цін для окремих категорій або за тегами, панелі управління акціями та промокодами з визначенням умов застосування, розмірів знижок, періодів дійсності та обмежень використання, панелі аналітики з графіками виручки, популярності страв, завантаженості закладу та ефективності маркетингових кампаній. Адміністратори мають повний доступ до всіх функцій системи, включаючи управління обліковими записами персоналу з визначенням ролей та прав доступу, налаштування параметрів системи таких як години роботи, зони доставки, мінімальні суми замовлення, перегляд логів системи для відстеження дій користувачів та виявлення проблем.

Кухарі отримують спеціалізований інтерфейс кухонного дисплею з відображенням черги поточних замовлень, деталями приготування кожної страви з інгредієнтами та спеціальними вказівками від клієнтів, можливістю оновлення статусів замовлень від нового до прийнятого, в процесі приготування та готового, таймерами для відстеження часу обробки замовлень та дотримання стандартів якості обслуговування, сповіщеннями про нові замовлення через звукові та візуальні сигнали. Шар бізнес-логіки реалізує всю функціональну логіку системи через набір модулів, що обробляють запити від клієнтського інтерфейсу, виконують валідацію вхідних даних, застосовують бізнес-правила та взаємодіють з шаром даних. Модуль управління меню забезпечує операції створення, читання, оновлення та видалення позицій меню з валідацією обов'язкових полів, перевіркою унікальності назв в межах категорії, обробкою завантаження зображень з автоматичною оптимізацією розмірів та форматів для

різних пристроїв, підтримкою версіювання контенту для можливості відкату змін.

Модуль обробки замовлень виконує валідацію кошика перед оформленням з перевіркою наявності всіх страв у меню, доступності позицій на основі залишків інгредієнтів, коректності кількостей та цін, розрахунок загальної вартості замовлення з урахуванням цін страв, кількості порцій, застосування промокодів та знижок, вартості доставки на основі адреси та мінімальної суми безкоштовної доставки, створення запису замовлення в базі даних з унікальним ідентифікатором та початковим статусом, ініціацію процесу оплати через інтеграцію з платіжними системами LiqPay або Fondy, відправлення повідомлень клієнту та персоналу про нове замовлення через електронну пошту та SMS, управління життєвим циклом замовлення з переходами між статусами новий, підтверджений, готується, готовий, в дорозі, доставлений або скасований. Модуль бронювання столів здійснює перевірку доступності столів на вказану дату та час з урахуванням поточних бронювань та тривалості візитів, резервування столу з створенням запису бронювання та блокуванням часового слоту, відправлення підтверджень та нагадувань клієнту за добу та годину до бронювання, обробку скасувань бронювань з автоматичним звільненням столів та повідомленням клієнтів.

Модуль управління складом відстежує залишки інгредієнтів на основі даних про використання в замовленнях, оновлює кількості після кожного замовлення відповідно до рецептур страв, генерує сповіщення при досягненні мінімальних порогів запасів для своєчасного поповнення, автоматично позначає страви як недоступні при критичному дефіциті інгредієнтів, формує звіти про споживання інгредієнтів для аналізу та планування закупівель. Модуль аналітики агрегує дані з різних джерел системи для формування звітів про фінансові показники діяльності закладу з виручкою, середнім чеком, прибутковістю, операційну ефективність з кількістю замовлень, часом обробки, рівнем задоволеності клієнтів, популярність позицій меню з рейтингами на основі частоти замовлень та виручки, ефективність маркетингових кампаній

через відстеження промокодів та джерел трафіку. Шар даних забезпечує надійне та ефективне зберігання всієї інформації системи через реляційну базу даних PostgreSQL з нормалізованою структурою таблиць для усунення дублювання даних, збереження цілісності через обмеження зовнішніх ключів та тригери, підтримку транзакцій ACID для критичних операцій типу створення замовлень та обробки платежів.

2.2 Архітектура програмних компонентів

Архітектура програмних компонентів вебсистеми базується на клієнт-серверній моделі з чітким розділенням фронтенд та бекенд частин, що комунікують через HTTP протокол з використанням RESTful API. Така архітектура забезпечує незалежну розробку та розгортання клієнтської та серверної частин, можливість створення множинних клієнтів для різних платформ з використанням єдиного API, ефективне кешування даних на клієнті для зменшення навантаження на сервер, горизонтальне масштабування серверів для обробки зростаючого навантаження. Рисунок 2.2 ілюструє загальну архітектуру системи з виділенням основних компонентів та їх взаємодії.

Фронтенд-архітектура побудована на компонентному підході React з розділенням на презентаційні та контейнерні компоненти для чіткого розмежування логіки відображення та бізнес-логіки. Презентаційні компоненти відповідають виключно за візуальне представлення даних без власного стану, отримують дані та callback-функції через props від батьківських компонентів, є повторно використовуваними в різних частинах застосунку, легко тестуються через відсутність побічних ефектів.

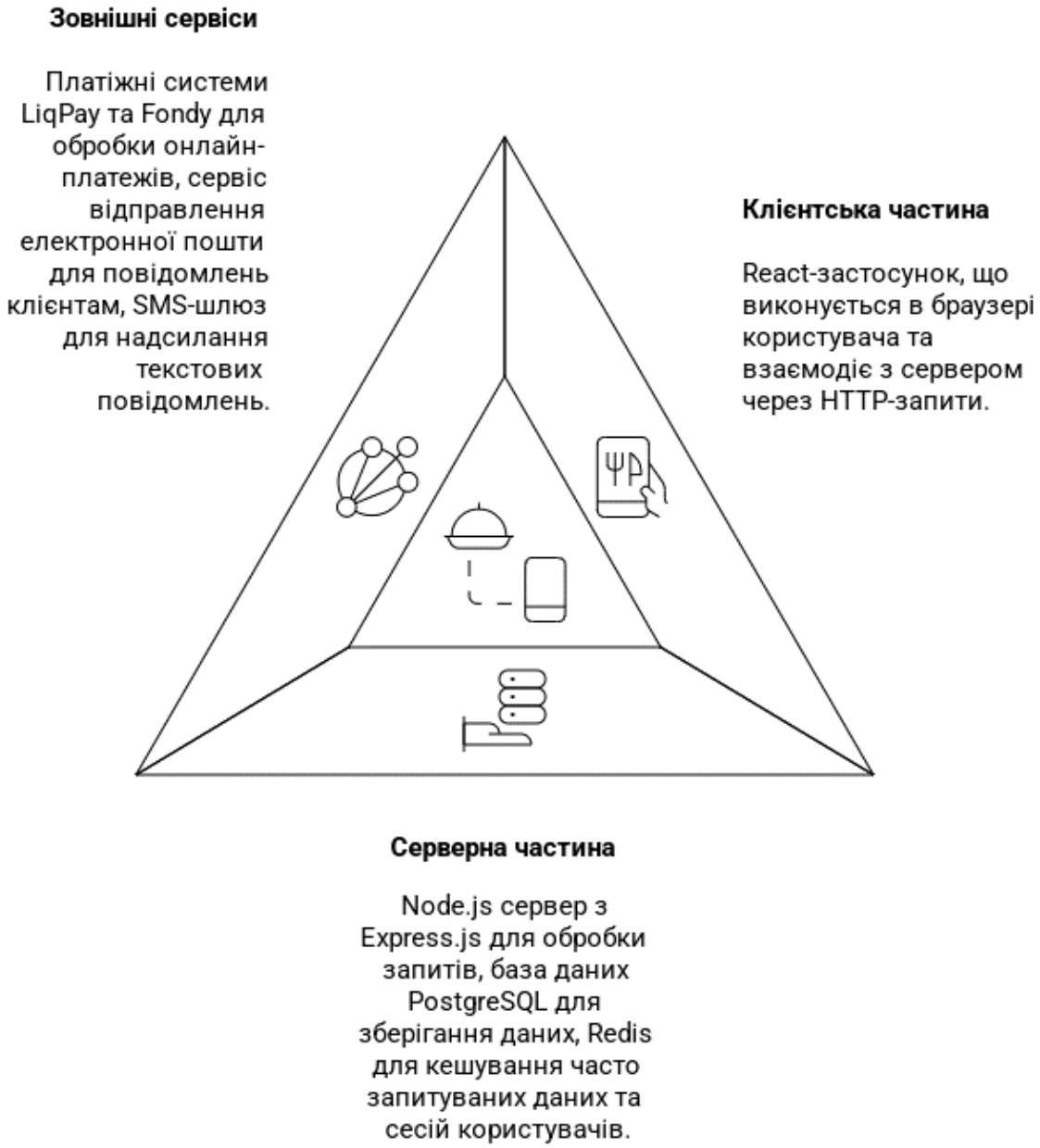

Рисунок 2.2 – Загальна архітектура системи

Приклади презентаційних компонентів включають MenuItem для відображення картки страви з назвою, зображенням, ціною та кнопкою додавання до кошика, CartItem для елемента кошика з можливістю зміни кількості та видалення, Button для стилізованих кнопок різних типів та розмірів, Input для полів введення з валідацією та відображенням помилок. Контейнерні компоненти управляють станом застосунку, виконують бізнес-логіку, взаємодіють з API через HTTP-запити, передають дані презентаційним компонентам через props. Ключові контейнерні компоненти системи включають

MenuPage для сторінки меню з логікою завантаження списку страв з сервера, фільтрації за категоріями та пошуковим запитом, пагінації результатів, CartPage для сторінки кошика з розрахунком загальної вартості, застосуванням промокодів, управлінням кількістю позицій, CheckoutPage для оформлення замовлення з валідацією форм, ініціацією платежу, обробкою відповіді від платіжної системи.

Управління глобальним станом застосунку реалізовано через Context API React для уникнення надмірного prop drilling та забезпечення доступу до спільних даних з будь-якого компонента. Основні контексти системи включають AuthContext для зберігання інформації про аутентифікованого користувача, токену доступу, ролі та прав, CartContext для управління вмістом кошика з методами додавання, видалення та оновлення позицій, NotificationContext для відображення глобальних повідомлень про успішні операції або помилки. Маршрутизація в застосунку забезпечується бібліотекою React Router з визначенням маршрутів для всіх сторінок системи, захищеними маршрутами, що вимагають автентифікації для доступу до адміністративної панелі, динамічними маршрутами з параметрами для сторінок деталей страв або замовлень, програмною навігацією при успішному виконанні операцій типу оформлення замовлення.

Бекенд-архітектура структурована за паттерном MVC з адаптацією для RESTful API, де контролери обробляють HTTP-запити та формують відповіді, моделі визначають структуру даних та логіку взаємодії з базою даних, сервісний шар інкапсулює бізнес-логіку для повторного використання між контролерами. Рисунок 2.3 демонструє детальну структуру серверної частини з розподілом відповідальності між шарами

Контролери системи організовані за доменними областями відповідно до ресурсів API. MenuController обробляє запити отримання списку категорій меню з опціональною фільтрацією та пагінацією, отримання списку страв з фільтрацією за категорією, цінами, дієтичними обмеженнями, отримання деталей конкретної страви за ідентифікатором, створення нової страви з

валідацією обов'язкових полів та завантаженням зображення, оновлення існуючої страви з частковим оновленням полів, видалення страви з перевіркою прав доступу.

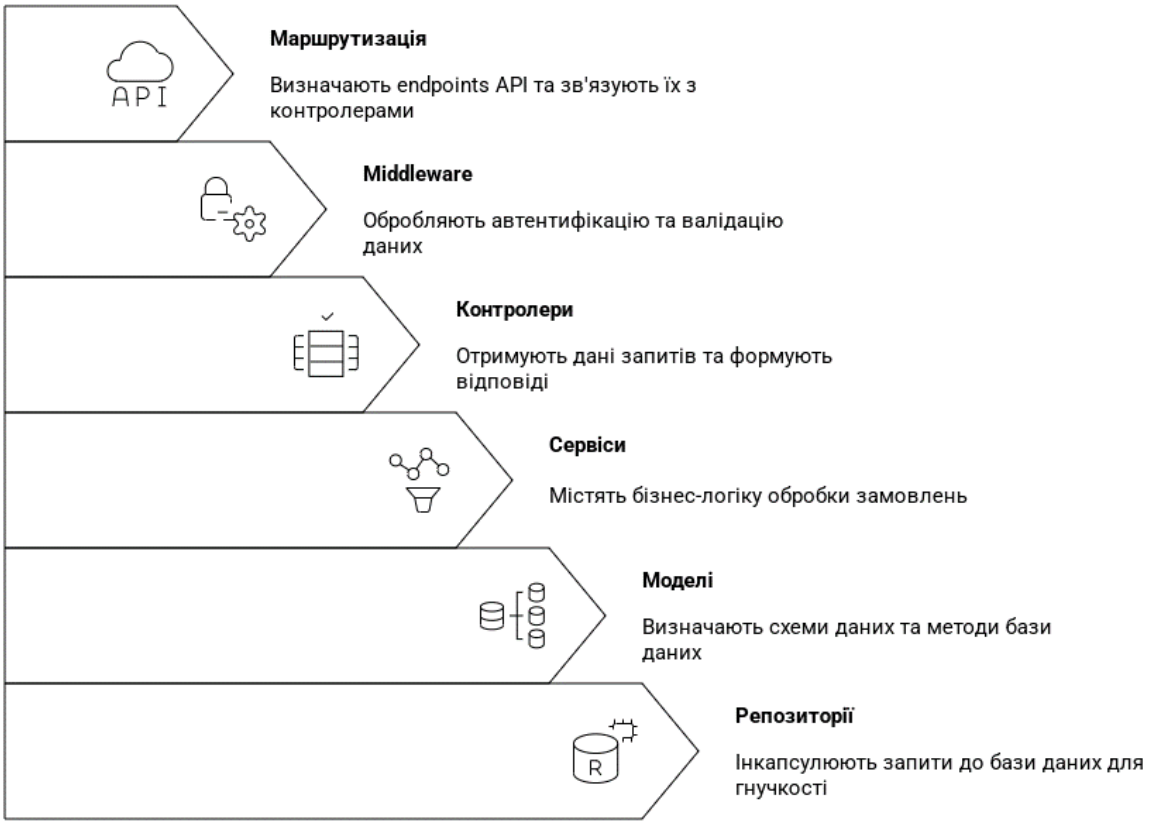

Рисунок 2.3 - Детальна структуру серверної частини з розподілом відповідальності між шарами

OrderController управляє створенням замовлення з валідацією кошика, розрахунком вартості, збереженням в базі, отриманням списку замовлень користувача з фільтрацією за статусом, отриманням деталей замовлення з інформацією про страви та статус доставки, оновленням статусу замовлення персоналом ресторану, скасуванням замовлення з перевіркою можливості відміни. ReservationController забезпечує перевірку доступності столів на вказану дату та час, створення бронювання з валідацією даних та резервуванням столу, отримання списку бронювань користувача, підтвердження або скасування бронювання персоналом (табл. 2.1).

Таблиця 2.1 - Архітектура програмних компонентів

Компонент	Характеристики
ЗАГАЛЬНА АРХІТЕКТУРА	
Архітектурна модель	Клієнт-серверна модель з чітким розділенням фронтенд та бекенд частин
Протокол взаємодії	HTTP з використанням RESTful API
Переваги	Незалежна розробка та розгортання, множинні клієнти, ефективне кешування, горизонтальне масштабування
КЛІЄНТСЬКА ЧАСТИНА	
Технологія	React-застосунок в браузері
Маршрутизація	React Router (захищені, динамічні маршрути, програмна навігація)
Управління станом	Context API: AuthContext, CartContext, NotificationContext
Презентаційні компоненти	MenuItem, CartItem, Button, Input - візуальне представлення без стану
Контейнерні компоненти	MenuPage, CartPage, CheckoutPage - управління станом, бізнес-логіка, API
СЕРВЕРНА ЧАСТИНА	
Серверна платформа	Node.js з Express.js
Архітектурний патерн	MVC з адаптацією для RESTful API
Шар маршрутизації	Визначає endpoints API та зв'язує з контролерами
Шар middleware	Автентифікація, логування, валідація, обробка помилок
Контролери	MenuController, OrderController, ReservationController
Шар сервісів	Бізнес-логіка: обробка замовлень, управління меню, аналітика
Шар моделей	Схеми даних через ORM Sequelize
Шар репозиторіїв	Інкапсуляція запитів до бази даних
БАЗА ДАНИХ	
Основна БД	PostgreSQL для зберігання основних даних
Кешування	Redis для кешування даних та сесій користувачів
ЗОВНІШНІ СЕРВІСИ	
Платіжні системи	LiqPay та Fondy для онлайн-платежів
Повідомлення	Email-сервіс для повідомлень, SMS-шлюз для текстових повідомлень

2.3 Моделювання потоків даних у вебсистемі

Моделювання потоків даних у вебсистемі управління рестораном дозволяє детально описати взаємодію між компонентами системи під час виконання основних бізнес-процесів. Діаграми потоків даних відображають джерела та приймачі інформації, процеси обробки даних, сховища даних та потоки між цими елементами. Рисунок 2.4 представляє контекстну діаграму системи найвищого рівня абстракції.

Рисунок 2.4 - Контекстна діаграма системи найвищого рівня абстракції

Процес онлайн-замовлення страв є одним з ключових бізнес-процесів системи, що включає множину етапів від перегляду меню до отримання підтвердження замовлення. Рисунок 2.5 деталізує потік даних при оформленні замовлення.

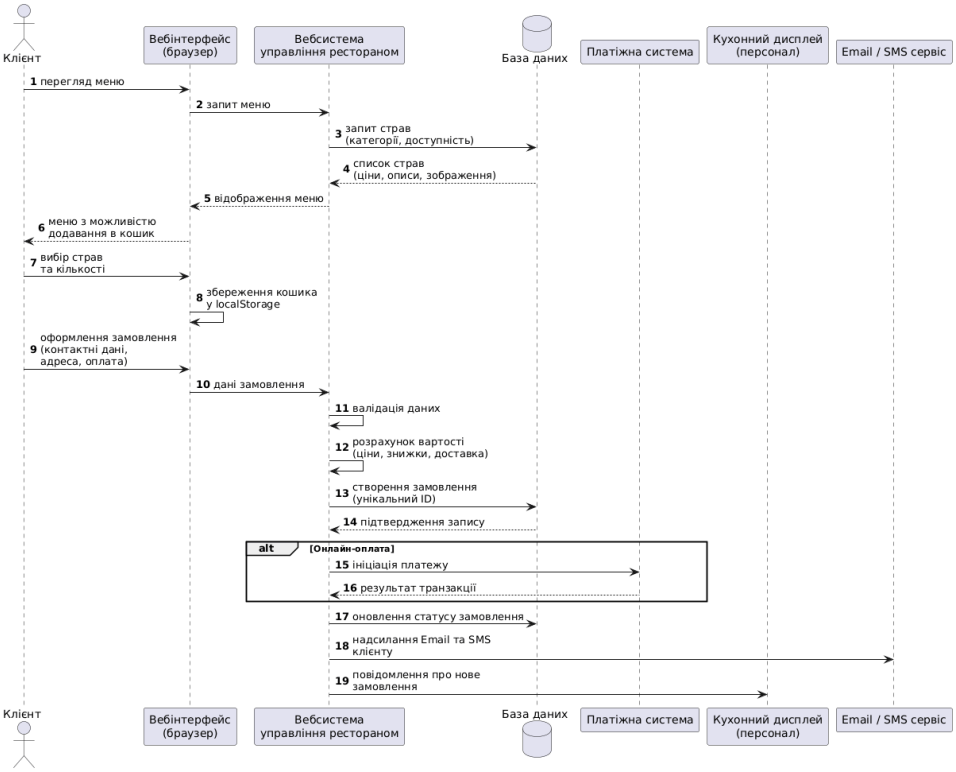

Рисунок 2.5 - Потік даних при оформленні замовлення

Процес бронювання столів забезпечує клієнтам можливість резервування місць у ресторані на конкретну дату та час без необхідності телефонного дзвінка. Рисунок 2.6 ілюструє потік даних при бронюванні столу.

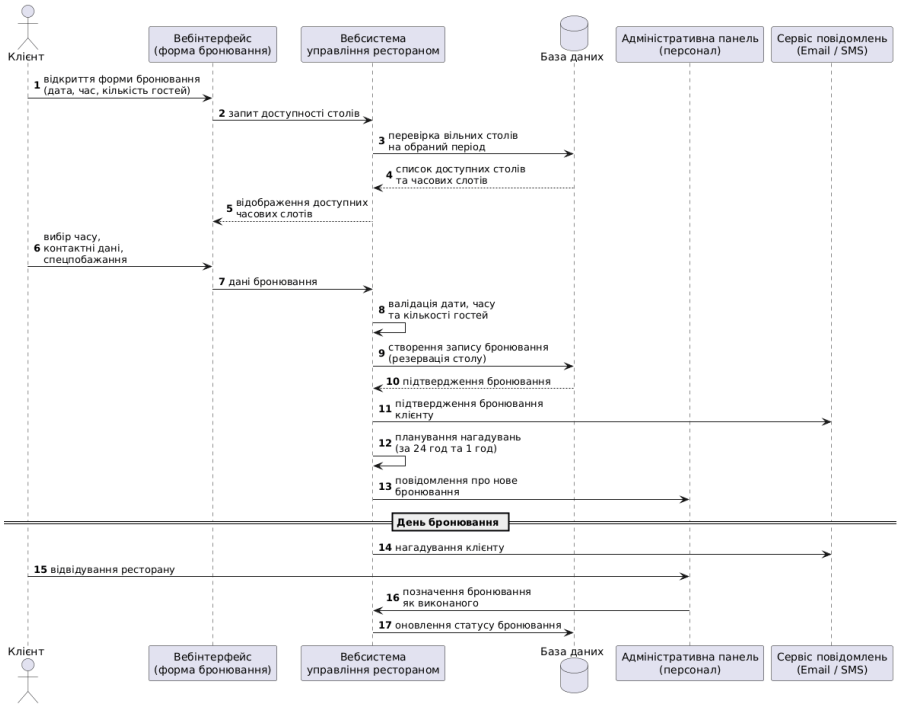

Рисунок 2.6 - Потік даних при бронюванні столу

Процес управління меню через адміністративну панель дозволяє персоналу ресторану оперативно актуалізувати інформацію про страви без залучення технічних фахівців. Рисунок 2.7 демонструє потік даних при оновленні меню.

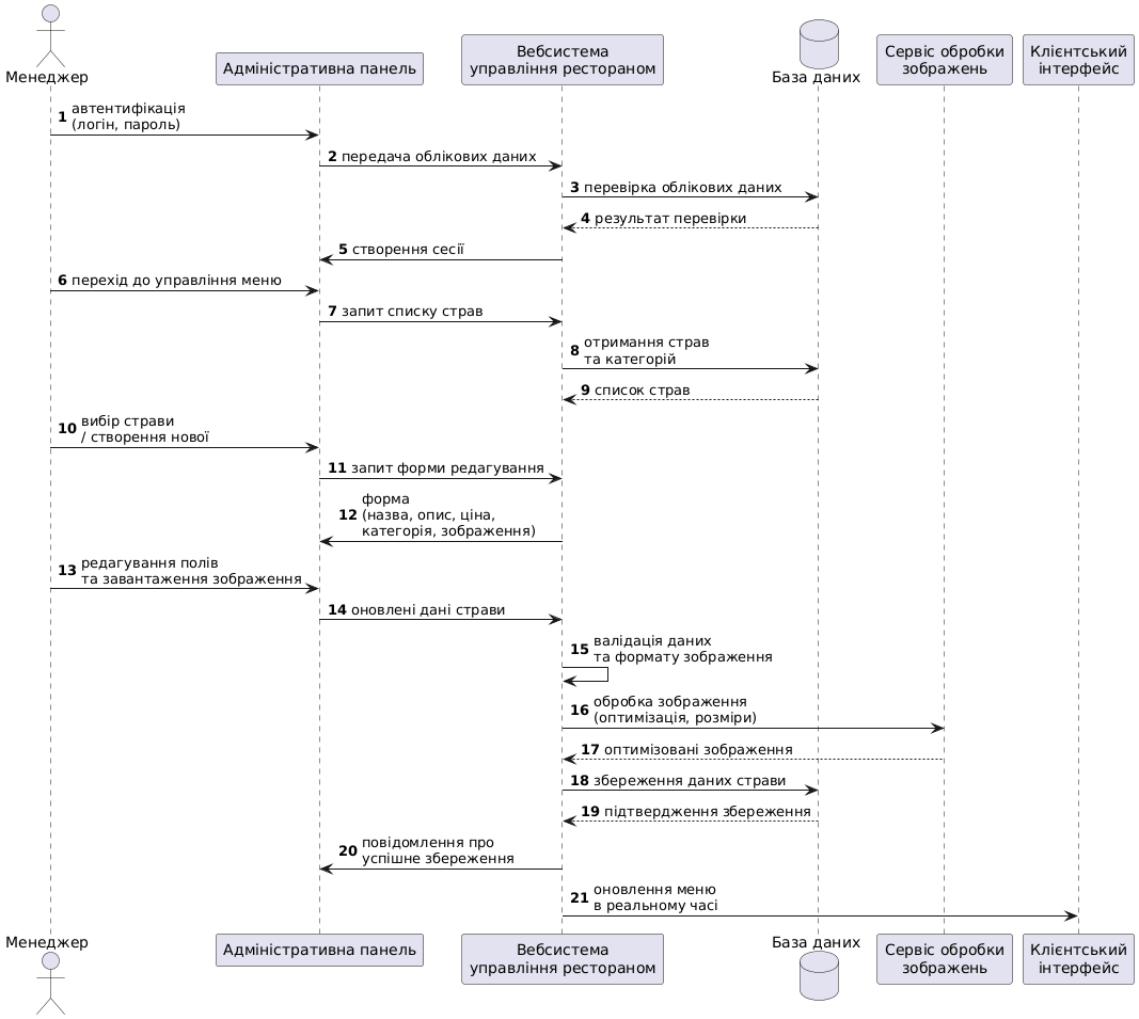

Рисунок 2.7 – Потік даних при оновленні меню

2.4 Проєктування структур даних вебсистеми

Проєктування структури бази даних є критично важливим етапом розробки вебсистеми, що визначає ефективність зберігання та обробки інформації. База даних вебсистеми управління рестораном організована за

принципами нормалізації для усунення дублювання даних, забезпечення цілісності через обмеження та зв'язки між таблицями, оптимізації продуктивності запитів через індекси. Концептуальна модель даних включає основні сутності користувачі, категорії меню, страви, інгредієнти, замовлення, позиції замовлень, бронювання, промокоди, відгуки та зв'язки між ними типу один-до-багатьох та багато-до-багатьох. Таблиця 2.2 описує структуру таблиці users для зберігання інформації про користувачів системи.

Таблиця 2.2 - Структура таблиці users

Поле	Тип даних	Обмеження	Опис
id	UUID	PRIMARY KEY	Унікальний ідентифікатор користувача
email	VARCHAR(255)	UNIQUE, NOT NULL	Електронна пошта користувача
password_hash	VARCHAR(255)	NOT NULL	Хешований пароль користувача
role	ENUM	NOT NULL	Роль користувача: customer, admin, chef, manager
first_name	VARCHAR(100)	NOT NULL	Ім'я користувача
last_name	VARCHAR(100)	NOT NULL	Прізвище користувача
phone	VARCHAR(20)	NULL	Номер телефону користувача
created_at	TIMESTAMP	DEFAULT NOW()	Дата та час реєстрації

Таблиця menu_items зберігає інформацію про страви меню ресторану з їх характеристиками та зв'язками з категоріями. Кожна страва належить до однієї категорії, має назву, опис, ціну, зображення та додаткові атрибути типу часу приготування, калорійності, дієтичних властивостей. Таблиця 2.3 деталізує структуру таблиці menu_items.

Таблиця 2.3 - Структура таблиці menu_items

Поле	Тип даних	Обмеження	Опис
id	UUID	PRIMARY KEY	Унікальний ідентифікатор страви
category_id	UUID	FOREIGN KEY	Ідентифікатор категорії меню
name	VARCHAR(200)	NOT NULL	Назва страви
description	TEXT	NULL	Детальний опис страви
price	DECIMAL(10,2)	NOT NULL	Ціна страви в гривнях
image_url	VARCHAR(500)	NULL	URL зображення страви
is_available	BOOLEAN	DEFAULT TRUE	Доступність страви для замовлення
calories	INTEGER	NULL	Калорійність страви в ккал

Таблиця orders зберігає основну інформацію про замовлення клієнтів з деталями доставки, оплати та статусу виконання. Кожне замовлення пов'язане з користувачем через зовнішній ключ та містить множину позицій у таблиці order_items. Таблиця 2.3 описує структуру таблиці orders.

Таблиця 2.4 - Структура таблиці orders

Поле	Тип даних	Обмеження	Опис
id	UUID	PRIMARY KEY	Унікальний ідентифікатор замовлення
user_id	UUID	FOREIGN KEY	Ідентифікатор користувача
status	ENUM	NOT NULL	Статус: new, confirmed, preparing, ready, delivering, completed, cancelled

Продовження таблиці 2.4

Поле	Тип даних	Обмеження	Опис
delivery_type	ENUM	NOT NULL	Тип доставки: delivery, pickup
payment_method	ENUM	NOT NULL	Спосіб оплати: online, cash
total_amount	DECIMAL(10,2)	NOT NULL	Загальна вартість замовлення
delivery_address	TEXT	NULL	Адреса доставки замовлення
created_at	TIMESTAMP	DEFAULT NOW()	Дата та час створення замовлення

Рисунок 2.8 представляє ER-діаграму бази даних з відображенням всіх таблиць та зв'язків між ними. Проєктована структура бази даних забезпечує ефективне зберігання даних з мінімальним дублюванням, підтримку складних запитів для аналітики, можливість розширення функціональності через додавання нових таблиць без зміни існуючих.

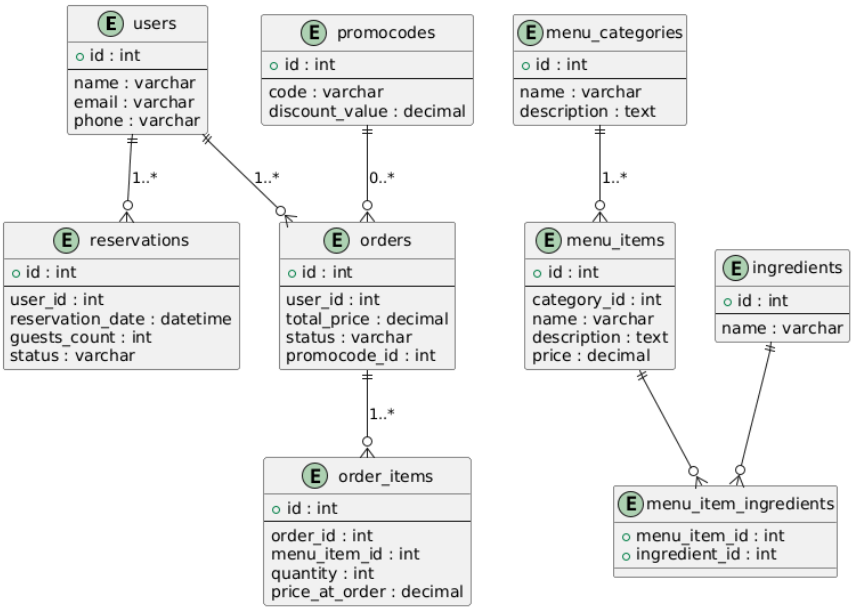

Рисунок 2.8 - ER-діаграму бази даних

2.5 Алгоритмічне забезпечення вебсистеми

Алгоритмічне забезпечення вебсистеми включає детальний опис ключових алгоритмів обробки даних та бізнес-логіки. Алгоритм обробки замовлення є центральним у функціонуванні системи та включає множину кроків валідації, розрахунків та оновлення стану. Рисунок 2.9 демонструє блок-схему алгоритму створення замовлення.

Рисунок 2.9 - Блок-схема алгоритму створення замовлення

Алгоритм перевірки доступності столів для бронювання забезпечує коректне резервування місць без конфліктів. Рисунок 2.10 ілюструє блок-схему алгоритму перевірки доступності.

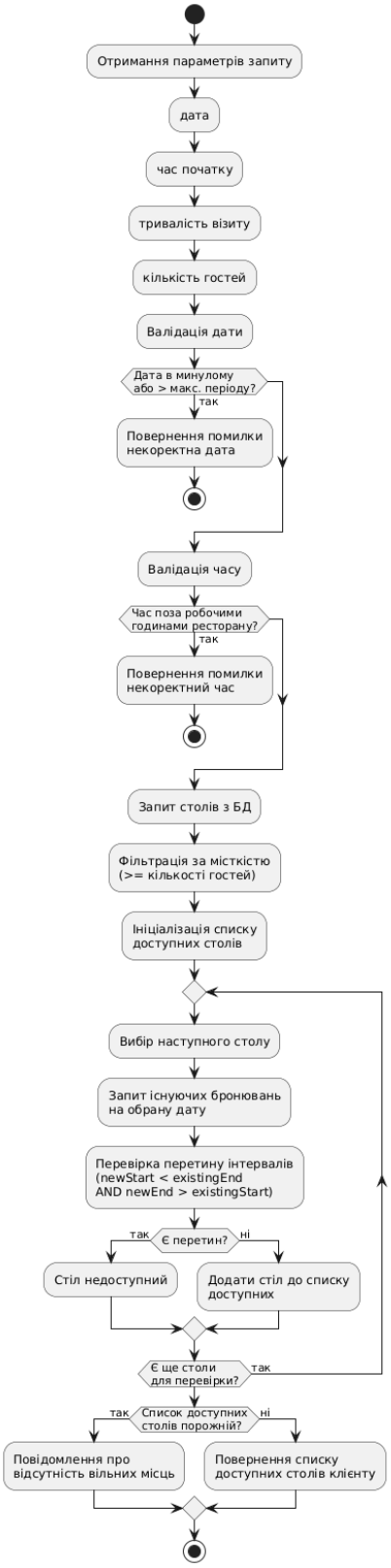

Рисунок 2.10 – Блок-схема алгоритму перевірки столиків для бронювання

Алгоритм управління залишками інгредієнтів автоматизує відстеження використання компонентів страв. Рисунок 2.11 показує блок-схему алгоритму оновлення залишків.

Рисунок 2.11 – Блок-схема алгоритму оновлення залишків

Алгоритм застосування промокодів забезпечує коректне нарахування знижок з перевіркою умов використання. Рисунок 2.12 демонструє блок-схему алгоритму валідації промокоду.

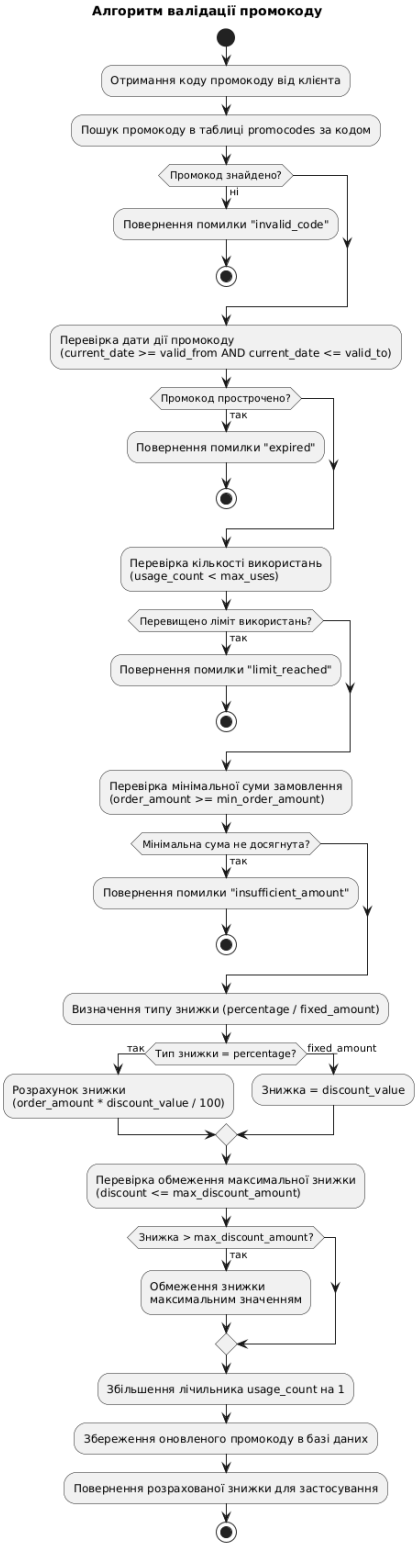

Рисунок 2.12 – Блок-схема алгоритму валідації промокоду

Розроблені алгоритми забезпечують коректну обробку бізнес-логіки вебсистеми з урахуванням всіх можливих сценаріїв використання та виняткових ситуацій. Використання блок-схем для документування алгоритмів полегшує розуміння логіки системи розробниками, тестувальниками та іншими учасниками проєкту, спрощує процес валідації коректності бізнес-правил перед початком реалізації коду, забезпечує основу для створення тестових сценаріїв що покривають всі гілки виконання алгоритмів. TypeScript надає додаткові гарантії коректності реалізації через систему типів що перевіряє відповідність даних очікуваним структурам на етапі компіляції, виявляє помилки типу використання undefined значень або неправильних типів аргументів функцій, забезпечує інтелектуальну підтримку в IDE з автодоповненням та підказками про типи.

3 РЕАЛІЗАЦІЯ ВЕБСИСТЕМИ УПРАВЛІННЯ РЕСТОРАНОМ

3.1 Архітектура інтерфейсних компонентів на базі React

Архітектура фронтенд-частини вебсистеми побудована на компонентному підході React з використанням TypeScript для типобезпеки. Рисунок 3.1 показує ієрархію компонентів головної сторінки з виділенням презентаційних та контейнерних компонентів. Кореневий компонент App містить маршрутизацію React Router та глобальні контексти для автентифікації, кошика, повідомлень. Компонент HomePage складається з Header для шапки сайту, HeroBanner для головного банеру, PopularDishes для розділу популярних страв, Benefits для переваг закладу, Testimonials для відгуків клієнтів, Footer для футера. Кожен з цих компонентів може бути повторно використаний на інших сторінках сайту. Header включає Logo для логотипу ресторану з посиланням на головну, Navigation для меню навігації, CartIcon для іконки кошика з індикатором кількості, UserMenu для меню користувача з входом або обліковим записом.

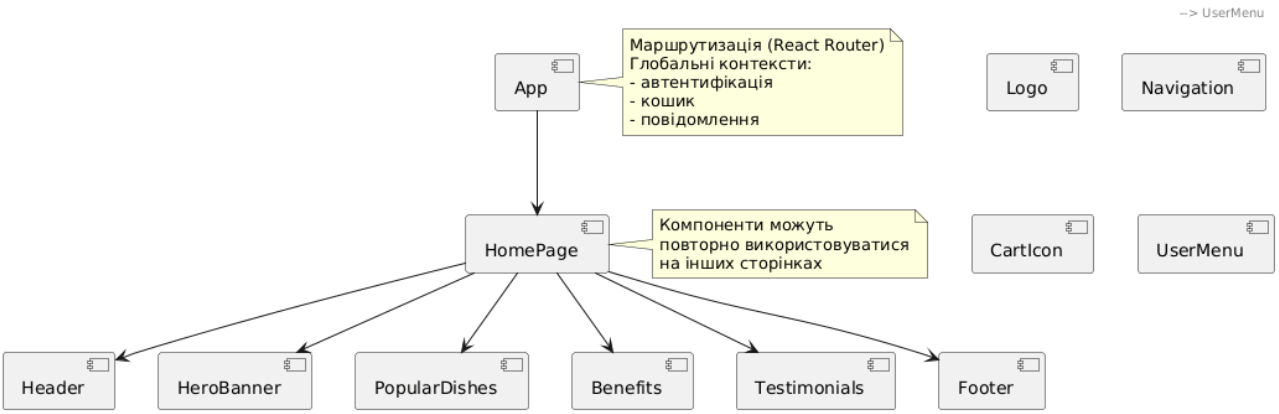

Рисунок 3.1 - Ієрархія компонентів головної сторінки з виділенням презентаційних та контейнерних компонентів

Рисунок 3.2 демонструє структуру компонента MenuPage з розподілом відповідальності між дочірніми компонентами. MenuPage виступає контейнерним компонентом що управляє станом фільтрів, пошукового запиту, поточної сторінки пагінації, списку страв завантажених з API. Компонент використовує хук useState для локального стану, useEffect для завантаження даних при зміні фільтрів або сторінки, useContext для доступу до контексту кошика. FilterSidebar відображає фільтри категорій, дієтичних обмежень, діапазону цін з callback-функціями для оновлення стану в батьківському компоненті. SearchBar містить поле введення з debounce затримкою для оптимізації запитів до сервера. SortOptions надає випадаючий список опцій сортування. DishGrid отримує відфільтрований список страв через props та відображає їх у вигляді сітки. DishCard є презентаційним компонентом для відображення картки страви з зображенням, назвою, описом, ціною, кнопкою додавання до кошика. При кліку на картку відкривається DishModal модальне вікно з детальною інформацією.

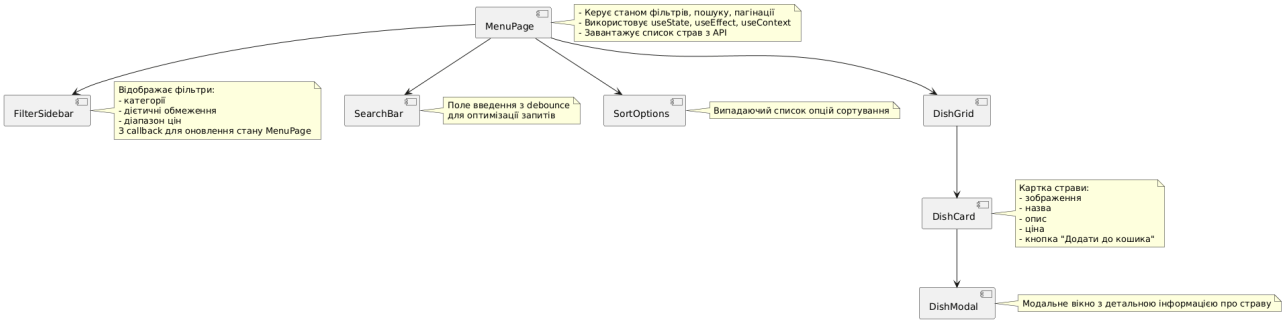

Рисунок 3.2 – Структура компонента MenuPage

Рисунок 3.3 ілюструє реалізацію компонента DishCard з TypeScript типізацією. Інтерфейс DishCardProps визначає структуру props компонента з полями dish об'єкт типу Dish, onAddToCart callback-функція для додавання до кошика, onViewDetails callback-функція для відкриття модального вікна. Тип Dish описує структуру даних страви з полями id унікальний ідентифікатор типу string, name назва страви типу string, description опис типу string, price ціна типу number, imageUrl URL зображення типу string, category категорія типу string, isVegetarian логічне значення, isVegan логічне значення, isGlutenFree логічне значення, calories калорійність типу number optional. Компонент DishCard використовує деструктуризацію props для отримання значень, обробляє подію onClick для виклику onViewDetails, рендерить зображення з атрибутами src та alt, відображає значки дієтичних властивостей умовно, показує назву та опис з обмеженням довжини, форматує ціну з валютою гривня, відображає кнопку додавання з обробником onClick що викликає onAddToCart [3].

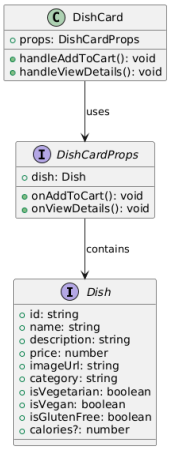

Рисунок 3.3 - Реалізацію компонента DishCard

Рисунок 3.4 показує реалізацію CartContext для управління глобальним станом кошика. Контекст CartContext створено через createContext з початковим значенням undefined. Інтерфейс CartItem визначає структуру позиції кошика з полями dish об'єкт страви типу Dish, quantity кількість типу number більше нуля. Інтерфейс CartContextValue описує API контексту з полями items масив позицій кошика типу CartItem, totalItems загальна кількість позицій типу number, totalAmount загальна вартість типу number, addItem функція додавання позиції, updateQuantity функція оновлення кількості, removeItem функція видалення позиції, clearCart функція очищення кошика. CartProvider компонент надає контекст дочірнім компонентам, використовує useReducer для управління складним станом кошика, реалізує логіку додавання позиції з перевіркою чи страва вже в кошику, оновлення кількості існуючої позиції або додавання нової, розрахунку загальної кількості та вартості. Хук useCart надає зручний доступ до контексту з перевіркою чи компонент всередині провайдера.

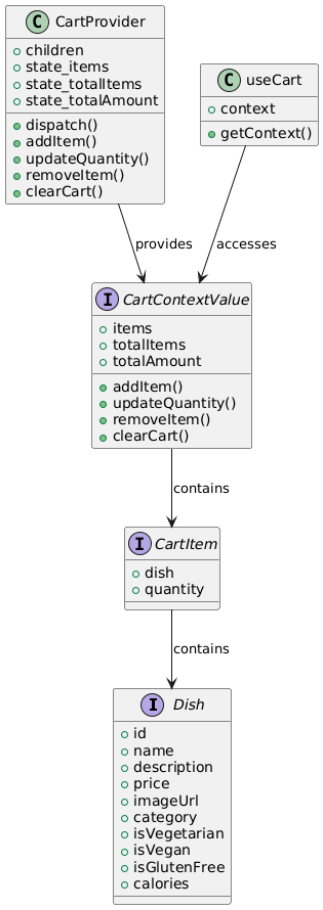

Рисунок 3.4 - Реалізацію CartContext

3.2 Дослідження ефективності застосування TypeScript

TypeScript надає систему статичної типізації для JavaScript що дозволяє виявляти помилки на етапі розробки та компіляції. Дослідження ефективності TypeScript проведено на критичних модулях вебсистеми що виконують розрахунки цін, обробку замовлень, валідацію даних. Функція calculateOrderTotal приймає параметри cartItems масив позицій кошика типу CartItem, promoCode опціональний промокод типу string або undefined, deliveryAddress адреса доставки типу DeliveryAddress. Тип DeliveryAddress визначено як інтерфейс з полями street вулиця типу string, city місто типу string, zipCode поштовий індекс типу string. Функція повертає об'єкт типу OrderTotal з полями subtotal проміжна сума, discount знижка, deliveryFee вартість доставки, total фінальна сума, всі поля типу number. Тіло функції виконує розрахунок проміжної суми через reduce з початковим значенням нуль, перевіряє наявність промокоду та розраховує знижку, обчислює вартість доставки на основі адреси, повертає об'єкт з результатами. TypeScript компілятор перевіряє типи всіх операцій та виявляє помилки типу передача рядка замість числа, доступ до неіснуючих полів об'єкта, виклик методів з неправильними аргументами.

Опис типів-об'єднань та перевірки типів для обробки різних варіантів відповіді від сервера. Тип ApiResponse визначено як об'єднання типів SuccessResponse та ErrorResponse що описують успішну та помилкову відповіді відповідно. SuccessResponse містить поля success логічне значення true, data дані довільного типу generic T. ErrorResponse містить поля success логічне значення false, error об'єкт помилки з message повідомленням типу string та code кодом помилки типу number. Функція handleApiResponse використовує type guard для звуження типу, перевіряє поле success для визначення типу відповіді, в гілці if обробляє успішну відповідь з доступом до поля data, в гілці else обробляє помилку з доступом до поля error. TypeScript забезпечує автодоповнення полів залежно від гілки виконання та попереджає про доступ до полів що можуть не

існувати. Це запобігає помилкам типу undefined is not an object що часто виникають в JavaScript при роботі з API.

Опис застосування generic типів для створення повторно використовуваної функції API-запитів. Функція fetchData визначена з generic параметром T що представляє тип очікуваних даних, приймає параметри url адреса запиту типу string, options опції запиту типу RequestInit optional, повертає Promise з generic типом T. Тіло функції виконує fetch запит, перевіряє статус відповіді, парсить JSON з явним приведенням до типу T, обробляє помилки через блок try-catch. Виклики функції з різними типами демонструють гнучкість, наприклад fetchData з типом Dish для отримання страви, fetchData з типом Order для отримання замовлення, fetchData з масивом User для отримання списку користувачів. TypeScript перевіряє відповідність отриманих даних очікуваному типу та надає автодоповнення при роботі з результатом. Використання generic типів зменшує дублювання коду та підвищує типобезпеку при роботі з різними ресурсами API.

Дані зібрано на основі розробки трьох ключових модулів системи модуль замовлень, модуль кошика, модуль меню протягом двох тижнів. Етап розробки включає написання коду, code review, тестування, продакшен. Для кожного етапу вказано кількість виявлених помилок без TypeScript та з TypeScript. Результати показують що з TypeScript на етапі написання коду виявлено п'ятдесят дві помилки завдяки компілятору, тоді як без TypeScript лише вісім через linter. На етапі code review з TypeScript виявлено дванадцять помилок проти двадцяти п'яти без TypeScript. На етапі тестування з TypeScript виявлено п'ять помилок проти вісімнадцяти без TypeScript. В продакшені з TypeScript виявлено одну помилку проти семи без TypeScript. Загальна кількість помилок з TypeScript сімдесят проти семидесяти восьми без TypeScript, але розподіл значно кращий з раннім виявленням на етапі розробки.

Рисунок 3.5 показує графік часу виявлення та виправлення помилок на різних етапах розробки. Вісь X представляє етапи розробка, тестування, продакшен, вісь Y час у годинах. Дві лінії показують середній час для проєкту з

TypeScript та без TypeScript. На етапі розробки з TypeScript час становить нуль цілих п'ять годин оскільки компілятор миттєво вказує на помилки, без TypeScript одну цілих дві години оскільки помилки виявляються пізніше. На етапі тестування з TypeScript час становить одну цілих вісім годин, без TypeScript чотири цілих п'ять годин через більшу кількість помилок. В продакшені з TypeScript час становить вісім годин, без TypeScript тридцять два години через складність репродукування та діагностики помилок. Аналіз показує що раннє виявлення помилок через TypeScript значно зменшує загальні витрати часу на виправлення та підвищує якість коду.

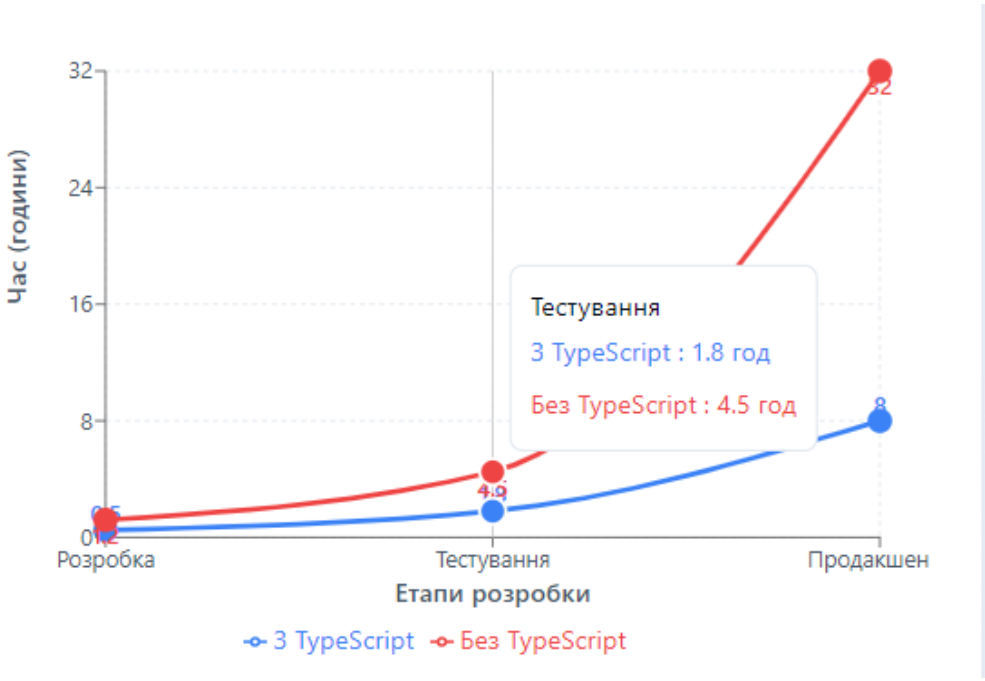

Рисунок 3.5 – Графік часу виявлення та виправлення помилок на різних етапах розробки

Результати дослідження показують що застосування TypeScript у вебсистемі управління рестораном забезпечує раннє виявлення помилок на етапі розробки до компіляції, зменшення часу на виправлення помилок завдяки точним повідомленням компілятора, покращення якості коду через явну документацію типів, прискорення розробки через інтелектуальне автодоповнення в IDE, спрощення рефакторингу через автоматичне оновлення використань при зміні сигнатур, полегшення командної роботи через

самодокументування коду типами. Особливо цінним TypeScript виявився для критичних модулів розрахунку цін, обробки платежів, валідації даних де помилки можуть призвести до фінансових втрат або порушення безпеки. Додаткові витрати на написання анотацій типів компенсуються значною економією часу на тестуванні та виправленні помилок на пізніх етапах розробки.

3.3 Реалізація серверної частини вебсистеми

Серверна частина вебсистеми реалізована на платформі Node.js з використанням фреймворку Express.js для обробки HTTP-запитів та маршрутизації. Опис структури директорій серверного проєкту з розподілом файлів за функціональним призначенням настпуний. Кореневий каталог містить файл server.js точку входу в застосунок, package.json з залежностями проєкту, tsconfig.json конфігурацію TypeScript компілятора, файл env з змінними середовища для підключення до бази даних та API ключами. Каталог src містить весь вихідний код з підкаталогами routes для визначення маршрутів API, controllers для логіки обробки запитів, services для бізнес-логіки, models для моделей даних Sequelize, middleware для проміжних обробників автентифікації валідації логування, utils для допоміжних функцій, types для TypeScript визначень типів. Каталог config містить конфігураційні файли для бази даних database.js, логера logger.js, платіжних систем payment.js.

Імпорт модулів включає express для створення сервера, cors для налаштування політики CORS, helmet для безпеки HTTP заголовків, morgan для логування запитів, dotenv для завантаження змінних середовища. Створення застосунку через виклик express з налаштуванням порту з змінної середовища або значення за замовчуванням три тисячі. Підключення middleware включає helmet для встановлення безпечних заголовків, cors з налаштуванням дозволених доменів, express.json для парсингу JSON тіла запитів з обмеженням розміру

десять мегабайт, morgan для логування запитів у форматі combined. Підключення маршрутів через app.use з префіксами для різних ресурсів, наприклад меню доступне за шляхом api menu, замовлення за шляхом api orders, користувачі за шляхом api users. Глобальний обробник помилок перехоплює всі необроблені винятки та повертає структуровану відповідь з кодом статусу та повідомленням.

Метод createOrder приймає об'єкт запиту та відповіді, витягує дані кошика та доставки з тіла запиту через деструктуризацію, викликає сервіс OrderService для створення замовлення з валідацією та збереженням в базі даних, обгортає виклик в блок try-catch для обробки помилок, повертає успішну відповідь з кодом двісті один та ідентифікатором створеного замовлення, в разі помилки повертає відповідь з кодом чотириста або п'ятсот залежно від типу помилки. Метод getOrderById отримує ідентифікатор замовлення з параметрів запиту, викликає сервіс для пошуку замовлення в базі даних, перевіряє чи знайдено замовлення і повертає код чотириста чотири якщо ні, повертає деталі замовлення з кодом двісті. Метод updateOrderStatus оновлює статус замовлення з перевіркою прав доступу користувача, логує зміну статусу для аудиту, відправляє повідомлення клієнту про зміну статусу.

Визначення моделі через Model.init з описом полів та їх типів DataTypes.UUID для ідентифікаторів з defaultValue UUIDV4 для автогенерації, DataTypes.ENUM для статусу з переліком можливих значень new confirmed preparing ready delivering completed cancelled, DataTypes.DECIMAL для грошових сум з точністю десять цифр та два знаки після коми, DataTypes.TEXT для текстових полів необмеженої довжини, DataTypes.TIMESTAMP для дат з автоматичним встановленням поточного часу. Визначення зв'язків між моделями через методи belongsTo для зв'язку багато-до-одного з таблицею users, hasMany для зв'язку один-до-багатьох з таблицею order_items. Налаштування hooks для виконання додаткової логіки після створення замовлення, такої як відправлення повідомлень клієнту та персоналу, оновлення залишків інгредієнтів, створення запису в таблиці аудиту.

Функція authMiddleware витягує токен з заголовка Authorization запиту, перевіряє наявність токена і повертає помилку чотириста один якщо токен відсутній, верифікує токен за допомогою бібліотеки jsonwebtoken з секретним ключем з змінних середовища, в разі успішної верифікації витягує дані користувача з payload токена, додає об'єкт користувача до запиту для використання в наступних обробниках, викликає next для передачі управління наступному middleware або контролеру, в разі невалідного токена повертає помилку чотириста один з повідомленням Invalid token. Функція checkRole перевіряє чи користувач має необхідну роль для доступу до ресурсу, приймає параметр roles масив дозволених ролей, повертає middleware функцію що перевіряє роль користувача, якщо роль не відповідає повертає помилку чотириста три з повідомленням Forbidden.

3.4 Розгортання та конфігурація апаратної частини системи

Апаратна складова вебсистеми включає серверну інфраструктуру для розгортання застосунку, базу даних, систему кешування та файлове сховище. Рисунок 3.6 демонструє схему розгортання системи на хмарній платформі з виділенням основних компонентів інфраструктури. Веб-сервер Nginx виступає як reverse proxy для прийому вхідних HTTP запитів від клієнтів, маршрутизації запитів до серверів застосунків з балансуванням навантаження round-robin, обслуговування статичних файлів HTML CSS JavaScript зображень без передачі на сервери застосунків, термінації SSL з'єднань для шифрованого HTTPS трафіку, кешування відповідей для зменшення навантаження на бекенд. Сервери застосунків Node.js розгорнуто в трьох екземплярах для забезпечення високої доступності та горизонтального масштабування, кожен екземпляр працює в окремому процесі з власним менеджером процесів PM2 для автоматичного

перезапуску при збоях, розподіл запитів між екземплярами через Nginx забезпечує рівномірне навантаження.

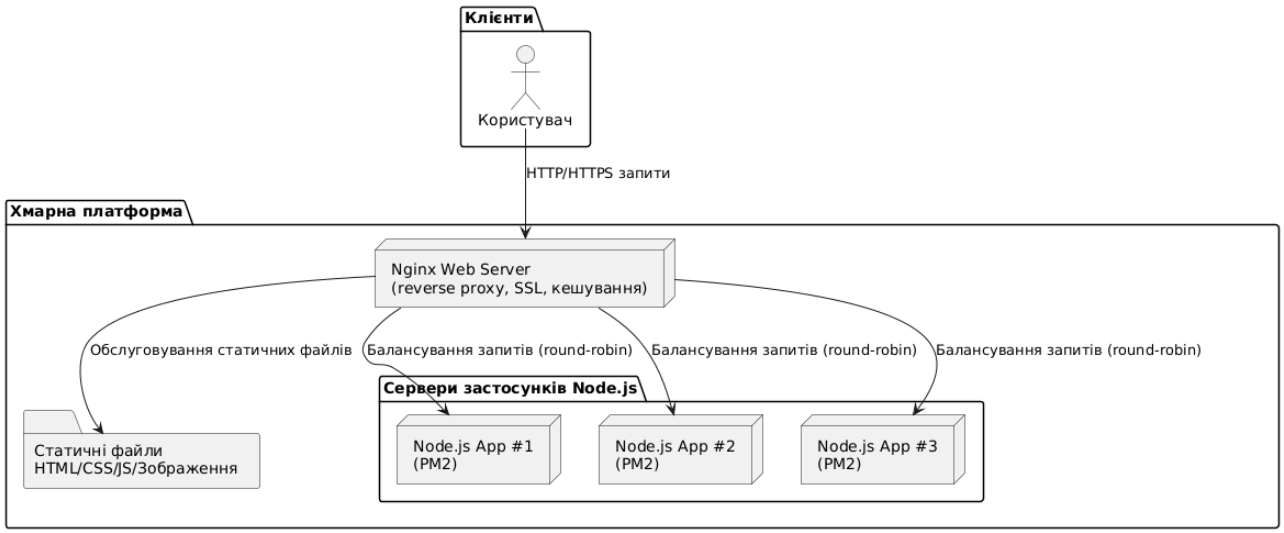

Рисунок 3.6 - Схема розгортання системи на хмарній платформі

Конфігурація Nginx для reverse proxy з налаштуваннями для вебсистеми ресторану наступна. Блок upstream визначає групу серверів застосунків з адресами та портами, метод балансування least_conn для направлення запитів на найменш завантажений сервер, параметр max_fails для виключення несправного сервера після трьох невдалих спроб, параметр fail_timeout для часу очікування перед повторною спробою тридцять секунд. Блок server налаштовує віртуальний хост з прослуховуванням порту вісімдесят для HTTP та чотириста сорок три для HTTPS, директивою server_name для доменного імені restaurant example com, блоком location для проксіювання запитів до upstream серверів з налаштуваннями proxy_pass для перенаправлення, proxy_set_header для передачі оригінальних заголовків Host X-Real-IP X-Forwarded-For, proxy_connect_timeout для таймауту підключення шістдесят секунд. Блок location для статичних файлів з директивою root для кореневого каталогу, expires для налаштування кешування один місяць, gzip для стиснення відповідей.

Таблиця 3.1 демонструє характеристики апаратного забезпечення серверів. Веб-сервер Nginx розгорнуто на віртуальній машині з двома ядрами процесора Intel Xeon E5-2670 з частотою два цілих шість гігагерц, чотирма

гігабайтами оперативної пам'яті DDR4, тридцятьма гігабайтами SSD дискового простору, операційною системою Ubuntu Server двадцять два цілих нуль чотири LTS. Сервери застосунків Node.js розгорнуто на трьох віртуальних машинах кожна з чотирма ядрами процесора, вісьмома гігабайтами оперативної пам'яті, п'ятдесятьма гігабайтами SSD дискового простору. База даних PostgreSQL розміщена на виділеному сервері з вісьмома ядрами процесора, шістнадцятьма гігабайтами оперативної пам'яті, двохстами гігабайтами SSD для даних плюс п'ятдесят гігабайт для журналів транзакцій, налаштовано реплікацію на резервний сервер для відмовостійкості. Сервер Redis для кешування має чотири ядра процесора, вісім гігабайт оперативної пам'яті повністю виділеної для кешу, тридцять гігабайт SSD для персистентності даних.

Таблиця 3.1 - Характеристики апаратного забезпечення серверів

Компонент	Кількість/ Екземпляри	Процесор	Частота CPU	ОЗП	Дисковий простір	Операційна система / Примітки
Веб-сервер Nginx	1	2 ядра Intel Xeon E5-2670	2.6 ГГц	4 ГБ DDR4	30 ГБ SSD	Ubuntu Server 22.04 LTS
Сервери застосунків Node.js	3	4 ядра	–	8 ГБ	50 ГБ SSD	–
База даних PostgreSQL	1 (+репліка)	8 ядер	–	16 ГБ	200 ГБ SSD (дані) + 50 ГБ SSD (журнали)	Реплікація на резервний сервер
Сервер Redis (кешування)	1	4 ядра	–	8 ГБ	30 ГБ SSD	Повністю виділено для кешу

База даних PostgreSQL налаштована з автоматичним створенням повних резервних копій щоночі о другій годині ночі за допомогою утиліти pg_dump,

збереженням копій на віддаленому сховищі Amazon S3 для захисту від локальних збоїв, ротацією копій з збереженням щоденних за останній тиждень, щотижневих за останній місяць, щомісячних за останній рік. Інкрементальне копіювання через Write-Ahead Logging архівує журнали транзакцій кожні п'ятнадцять хвилин, дозволяє відновити базу до будь-якого моменту часу point-in-time recovery, займає менше дискового простору порівняно з повними копіями. Файлове сховище зображень страв синхронізується з резервним сховищем щогодини через rsync, забезпечує швидке відновлення при втраті даних, використовує дедуплікацію для економії простору.

3.5 Інтерфейс реалізованої вебсистеми управління рестораном

Реалізований інтерфейс вебсистеми демонструє всі ключові функціональні можливості для клієнтів та персоналу ресторану. Розроблено два окремі інтерфейси: клієнтський для замовлення страв та бронювання столиків, та адміністративний для управління замовленнями, меню та бронюваннями. Обидва інтерфейси адаптовані для роботи на різних пристроях завдяки використанню responsive design.

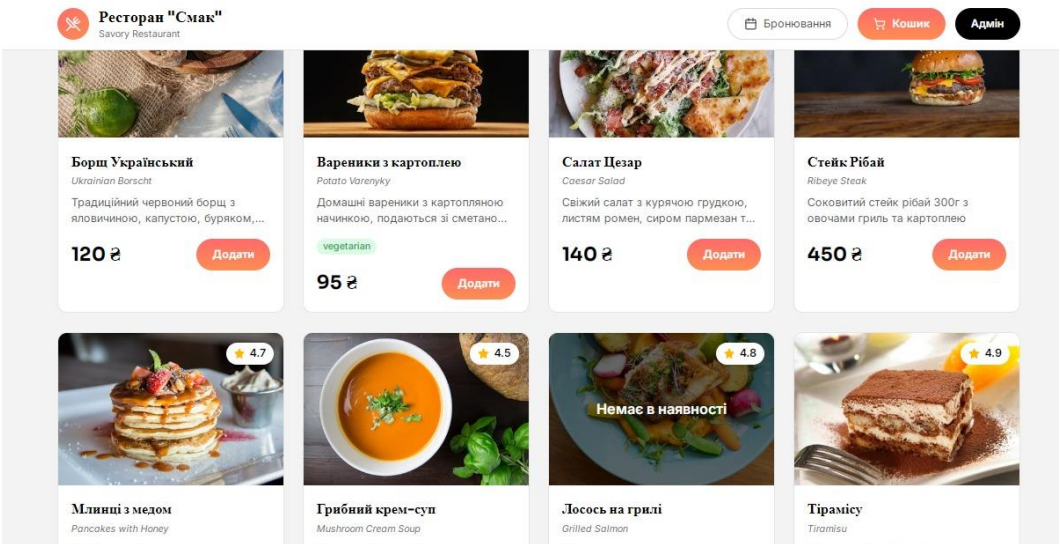

Рисунок 3.6 – Головна сторінка клієнтського інтерфейсу з каталогом страв

Головна сторінка вебсистеми (рисунок 3.6) містить каталог страв ресторану з фотографіями, назвами, описами та цінами. Користувач може переглядати доступні страви, фільтрувати їх за категоріями та додавати обрані позиції до кошика. Кожна картка страви містить рейтинг, ціну та кнопку швидкого додавання до замовлення. У верхній частині розміщено логотип ресторану та кнопки доступу до кошика, бронювання столиків та адміністративної панелі.

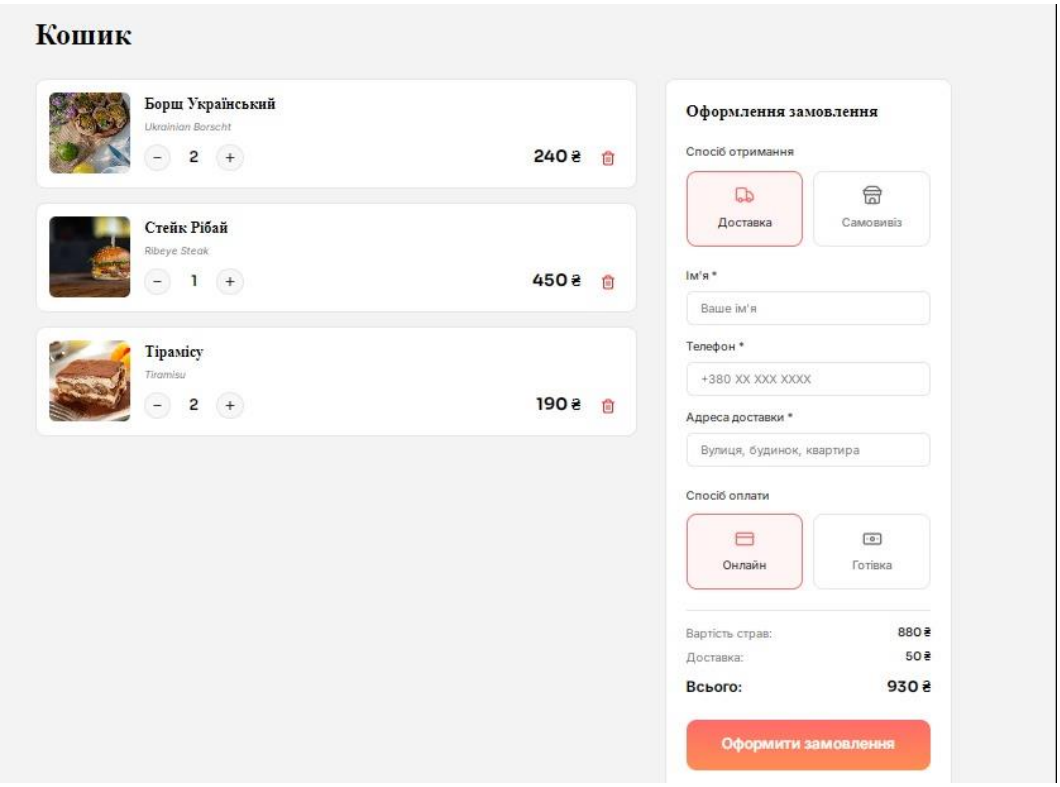

Рисунок 3.7 – Інтерфейс кошика з формою оформлення замовлення

На рисунку 3.7 показано сторінку кошика, яка відображає список обраних страв з можливістю зміни кількості та видалення позицій. Форма оформлення замовлення включає поля для введення контактних даних клієнта (ім'я, телефон, адреса доставки), вибір способу отримання замовлення (доставка або самовивіз) та способу оплати (онлайн або готівкою). Система автоматично розраховує вартість страв, вартість доставки та загальну суму замовлення.

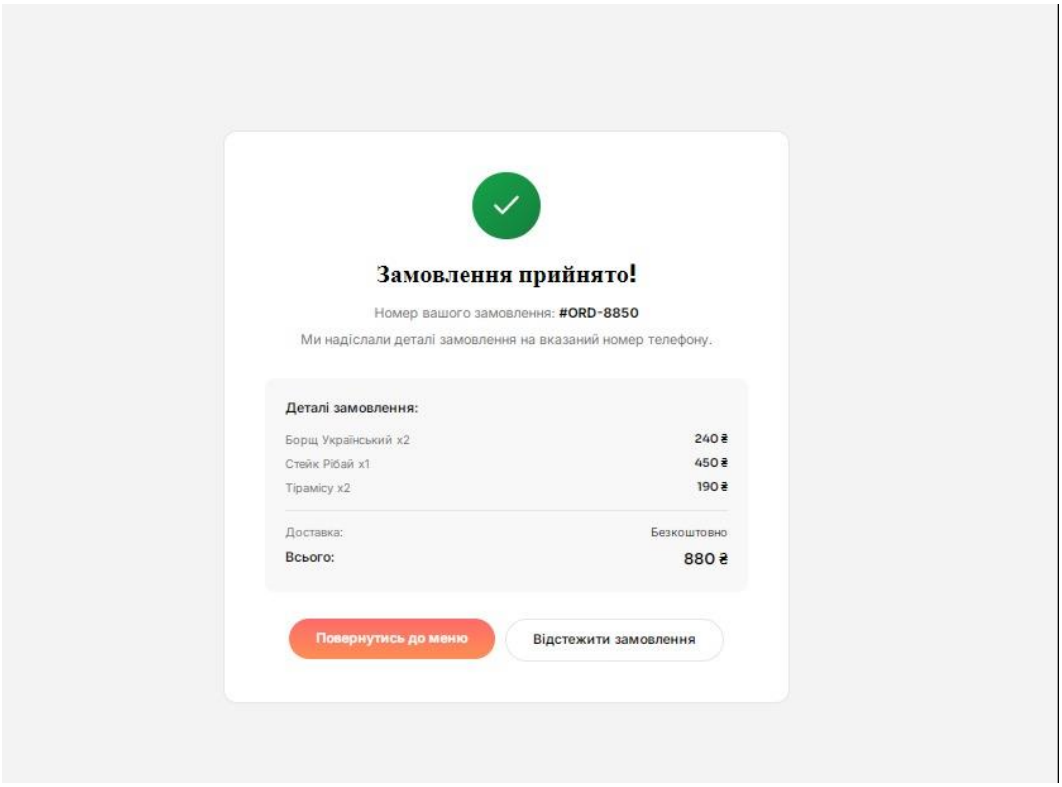

Рисунок 3.8 – Сторінка підтвердження прийнятого замовлення

Після успішного оформлення замовлення користувач отримує підтвердження з унікальним номером замовлення (рисунок 3.8). На сторінці відображаються деталі замовлення: список замовлених страв з кількістю та цінами, спосіб доставки та загальна сума. Користувач має можливість повернутися до меню для нового замовлення або відстежити статус поточного замовлення за вказаним номером телефону.

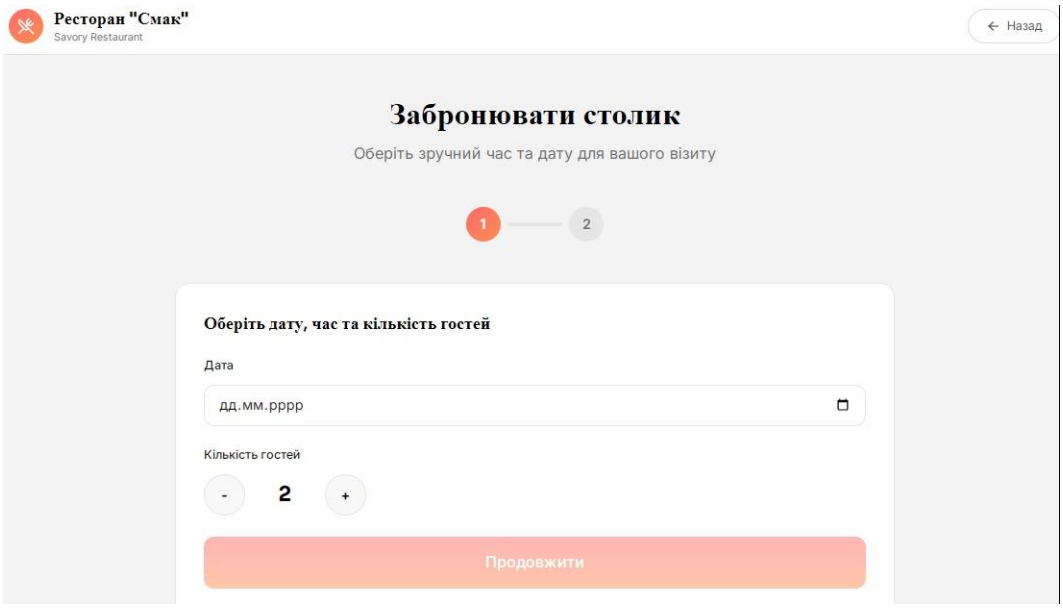

Рисунок 3.9 – Інтерфейс бронювання столика (крок 1 – вибір дати та кількості гостей)

Перший крок процесу бронювання столика (рисунок 3.9) дозволяє клієнту обрати дату візиту та кількість гостей. Інтерфейс використовує покроковий wizard з індикатором прогресу для зручної навігації. Користувач може вказати бажану дату через інтерактивний календар та налаштувати кількість гостей за допомогою кнопок збільшення/зменшення значення.

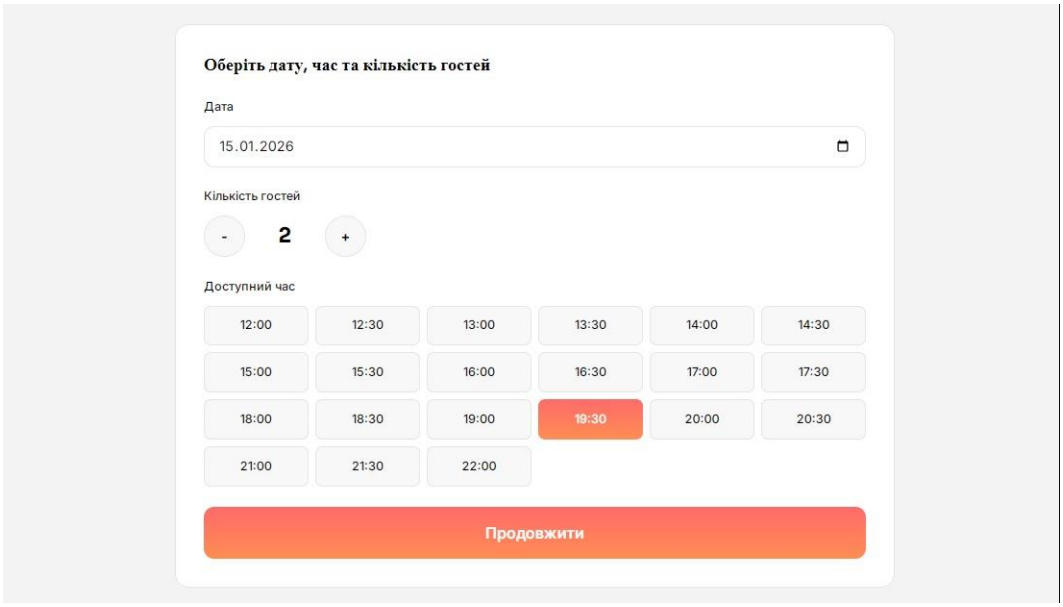

Рисунок 3.10 – Інтерфейс бронювання столика (вибір часу)

Після вибору дати система відображає доступні часові слоти для бронювання (рисунок 3.10). Часові інтервали представлені у вигляді кнопок з кроком у 30 хвилин. Доступні слоти виділені білим кольором, обраний час підсвічується помаранчевим, а недоступні слоти можуть бути деактивовані. Така візуалізація дозволяє клієнту швидко знайти зручний час для візиту та уникнути конфліктів з іншими бронюваннями.

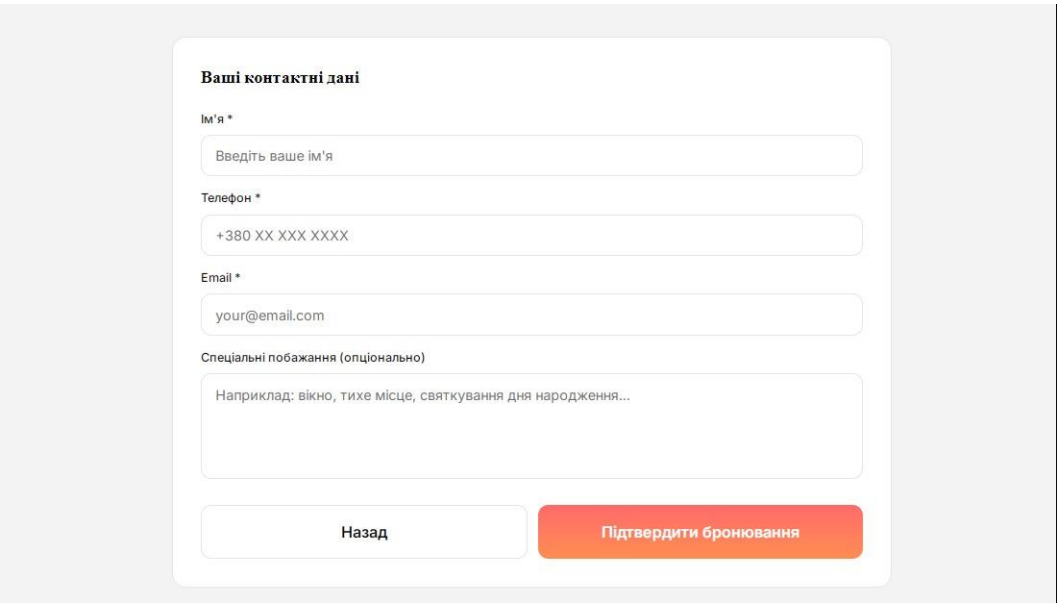

Рисунок 3.11 – Інтерфейс бронювання столика (крок 2 – введення контактних даних)

На другому кроці бронювання (рисунок 3.11) клієнт заповнює контактну інформацію: ім'я, номер телефону, електронну пошту та може додати спеціальні побажання щодо бронювання. Форма містить валідацію введених даних для забезпечення коректності інформації. Після заповнення всіх обов'язкових полів користувач може підтвердити бронювання або повернутися на попередній крок для внесення змін.

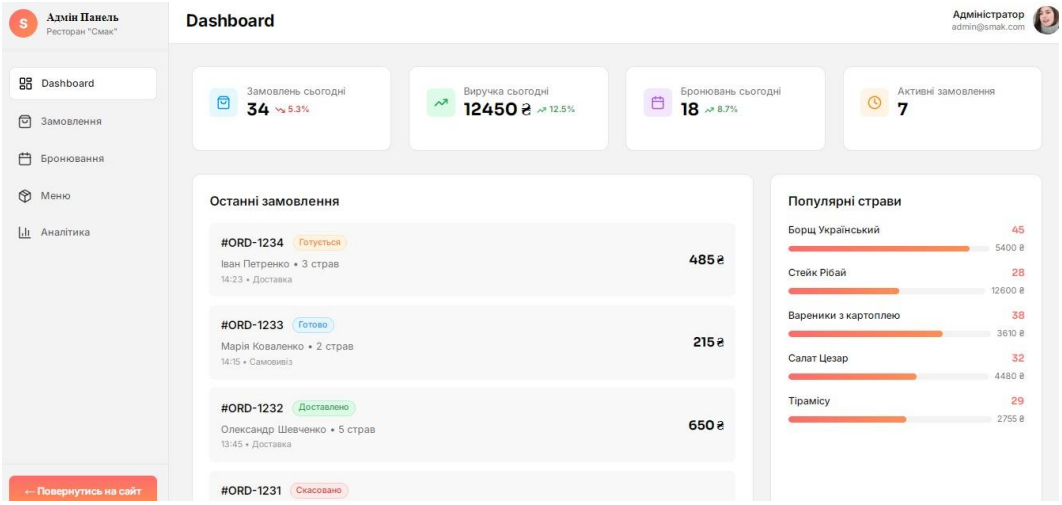

Рисунок 3.12 – Головна панель адміністратора (Dashboard)

Dashboard адміністративної панелі (рисунок 3.12) надає загальний огляд діяльності ресторану. Верхня частина містить ключові метрики: кількість замовлень за день, загальну виручку, кількість бронювань та активні замовлення з динамікою відносно попереднього періоду. Нижня частина відображає список останніх замовлень зі статусами виконання (готується, готово, доставлено, скасовано) та діаграму популярних страв з кількістю замовлень та загальною виручкою по кожній позиції.

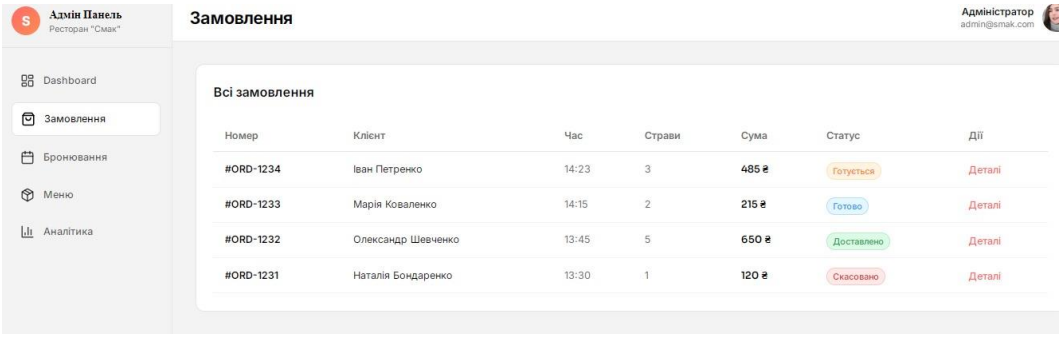

Рисунок 3.13 – Інтерфейс управління замовленнями

Як показано на рисунку 3.13, сторінка управління замовленнями містить повний список всіх замовлень у вигляді таблиці. Для кожного замовлення відображається унікальний номер, ім'я клієнта, час створення, кількість страв, загальна сума та поточний статус. Кольорові індикатори статусів (помаранчевий – готується, синій – готово, зелений – доставлено, червоний – скасовано)

дозволяють адміністратору швидко оцінити стан замовлень. Кнопка "Деталі" надає доступ до детальної інформації про кожне замовлення.

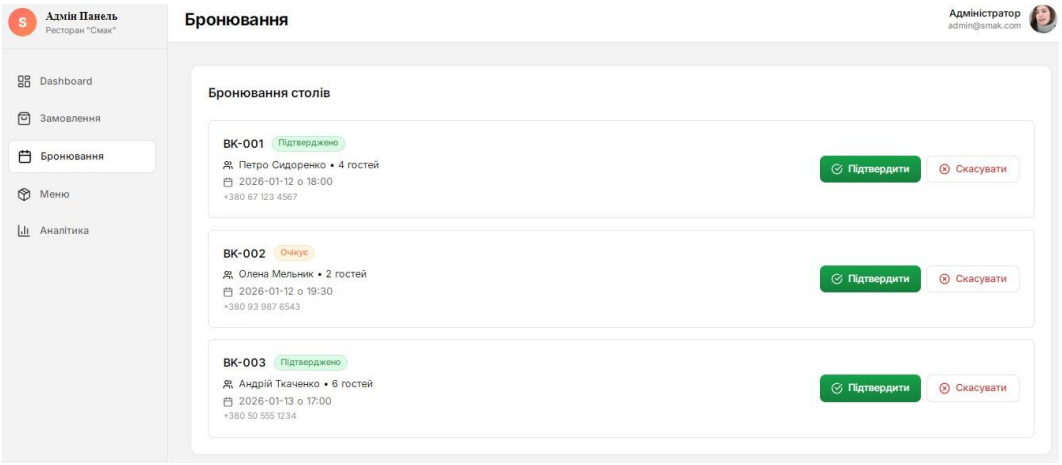

Рисунок 3.14 – Інтерфейс управління бронюваннями столиків

Розділ управління бронюваннями (рисунок 3.14) відображає список всіх бронювань з ключовою інформацією: унікальний номер бронювання, ім'я клієнта, кількість гостей, дата та час візиту, контактний телефон та статус бронювання (підтверджено, очікує). Адміністратор може підтвердити або скасувати бронювання за допомогою відповідних кнопок. Зелений колір статусу "Підтверджено" та помаранчевий "Очікує" забезпечують швидку візуальну ідентифікацію стану кожного бронювання.

Мобільна версія вебсистеми адаптована для використання на смартфонах та планшетах. Інтерфейс оптимізовано для сенсорного управління з використанням великих кнопок та елементів управління. Всі функціональні можливості десктопної версії доступні у мобільному інтерфейсі з урахуванням особливостей невеликого екрану.

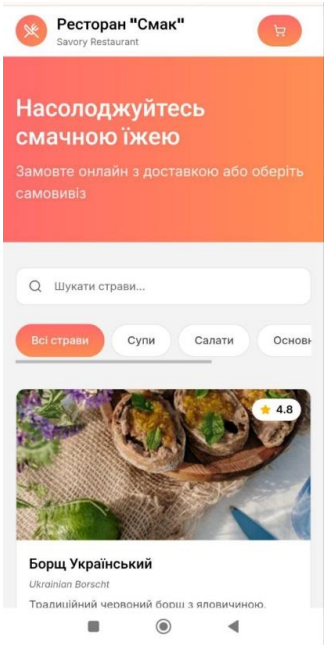

Рисунок 3.15 – Головна сторінка мобільного інтерфейсу

Мобільна версія головної сторінки (рисунок 3.15) оптимізована для вертикального прокручування. Верхня частина містить банер з гаслом та кнопку кошика. Користувач може шукати страви за допомогою поля пошуку та фільтрувати їх за категоріями (всі страви, супи, салати, основні страви). Картки страв відображаються у одну колонку з великими зображеннями, рейтингом, назвою, описом та ціною.

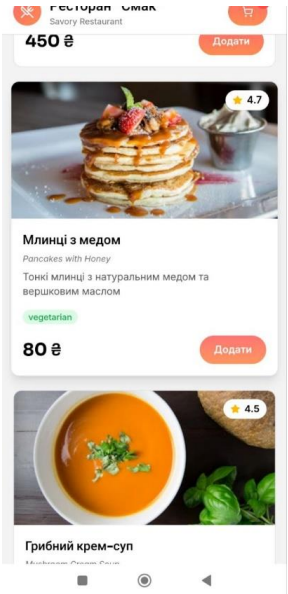

Рисунок 3.16 – Детальна інформація про страву (мобільна версія)

При натисканні на картку страви відкривається детальна сторінка з повною інформацією (рисунок 3.16). Великий розмір зображення дозволяє краще оцінити вигляд страви. Відображається рейтинг, назва англійською та українською мовами, детальний опис, позначка вегетаріанської страви (якщо застосовується), ціна та кнопка додавання до кошика. Інтерфейс оптимізовано для сенсорного управління з великими кнопками.

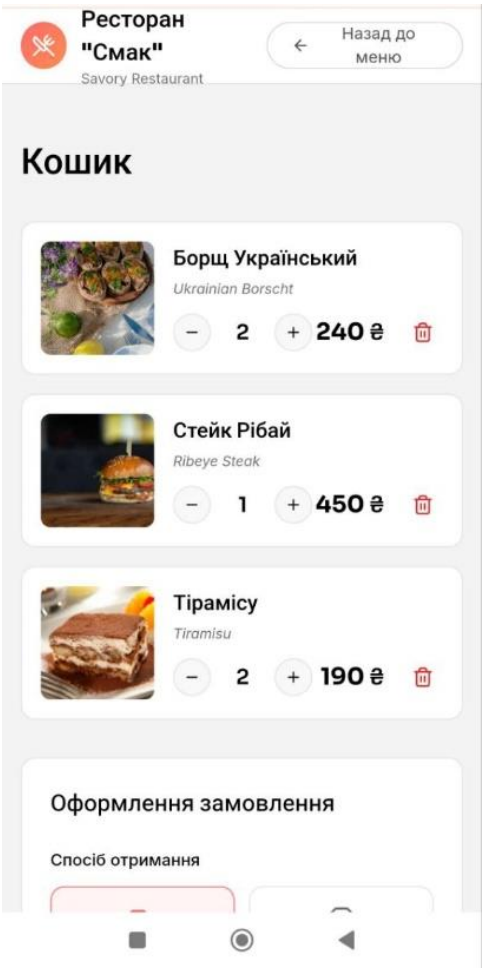

Рисунок 3.17 – Кошик з формою оформлення замовлення (мобільна версія)

На рисунку 3.17 представлено мобільну версію кошика, яка відображає обрані страви з мініатюрами фотографій, назвами та цінами. Користувач може змінювати кількість кожної позиції за допомогою кнопок +/- або видаляти страви з кошика. Нижня частина екрану містить форму оформлення замовлення з вибором способу отримання (доставка або самовивіз) та полями для введення

контактних даних. Інтерфейс адаптований для введення даних на мобільних пристроях.

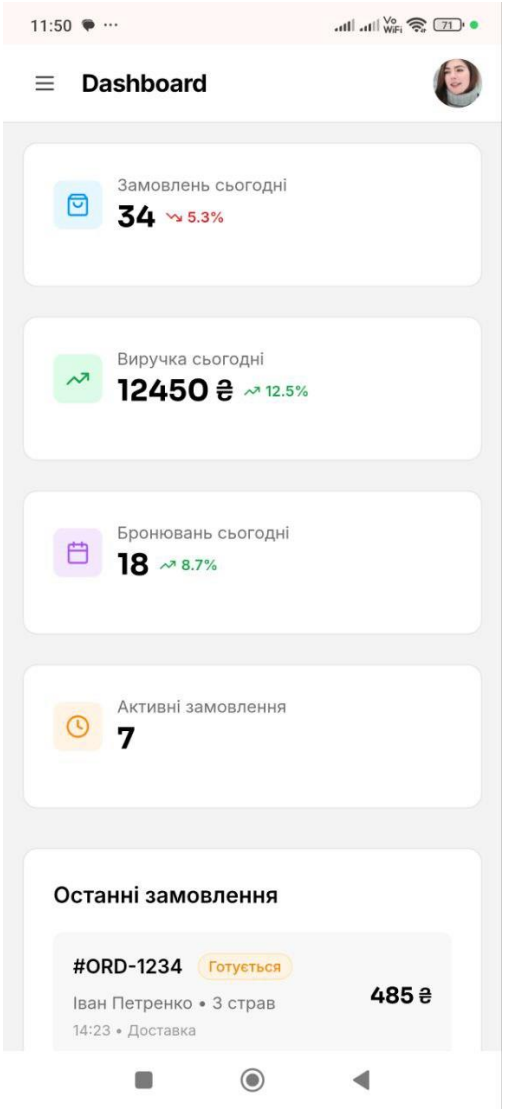

Рисунок 3.18 – Мобільна версія адміністративної панелі (Dashboard)

Dashboard адмін-панелі у мобільній версії (рисунок 3.18) відображає ключові метрики у вигляді окремих карток, розташованих вертикально. Кожна картка містить іконку, назву метрики, числове значення та динаміку зміни у відсотках. Нижня частина екрану показує список останніх замовлень з основною інформацією: номером, іменем клієнта, кількістю страв, сумою та статусом. Меню навігації адаптовано для сенсорного управління.

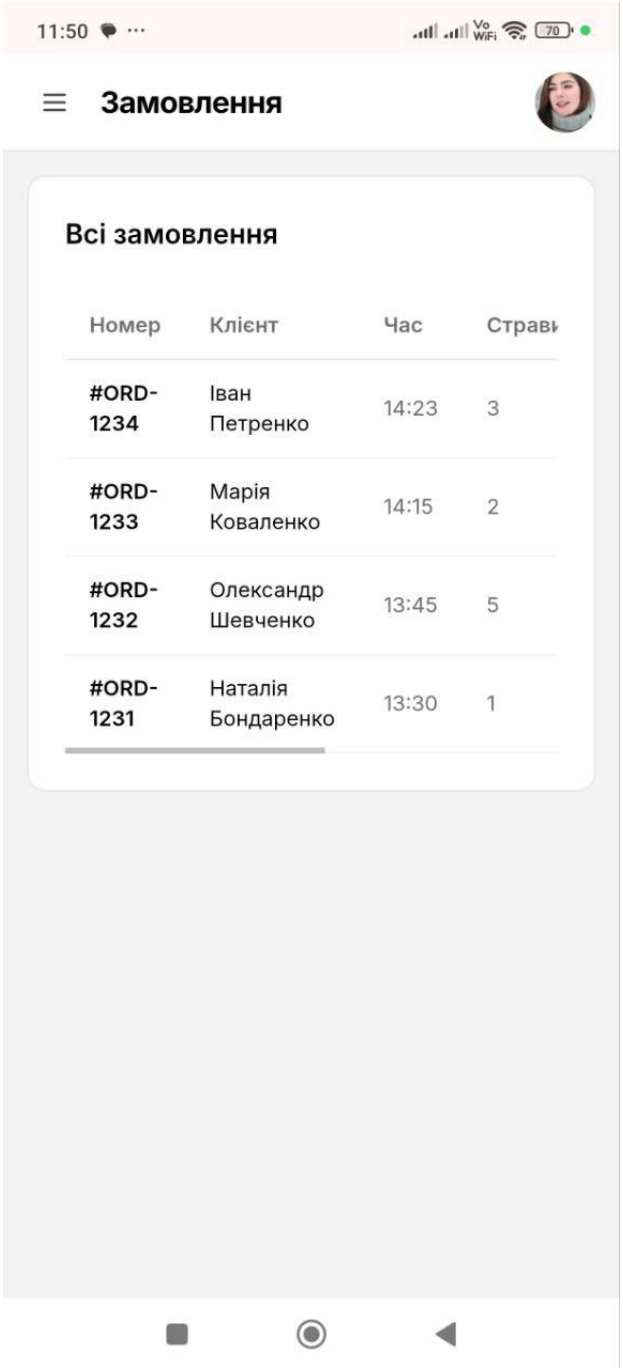

Рисунок 3.19 – Управління замовленнями (мобільна версія)

Мобільна версія сторінки управління замовленнями (рисунок 3.19) представляє інформацію у вигляді таблиці, адаптованої для малих екранів. Відображаються основні дані: номер замовлення, ім'я клієнта, час створення та кількість страв. Через обмежений простір екрану деякі колонки можуть бути приховані, але доступні через горизонтальне прокручування. Адміністратор може натиснути на рядок замовлення для перегляду повної інформації та зміни статусу.

Реалізований інтерфейс вебсистеми забезпечує зручну взаємодію як для клієнтів, так і для персоналу ресторану. Адаптивний дизайн гарантує коректне відображення та функціонування на різних пристроях. Інтуїтивна навігація, чітка візуальна ієрархія та інформативні елементи інтерфейсу сприяють ефективному виконанню користувацьких задач – від перегляду меню та оформлення замовлення до управління замовленнями та бронюваннями.

4 ДОСЛІДНИЦЬКА ЧАСТИНА

4.1 Методологія експериментального дослідження вебсистеми

Експериментальне дослідження вебсистеми управління рестораном проведено для оцінки продуктивності системи під різними навантаженнями, визначення вузьких місць архітектури, перевірки коректності роботи критичних модулів, оцінки якості користувацького досвіду. Методологія дослідження базується на комплексному підході з використанням автоматизованих інструментів навантажувального тестування, ручного функціонального тестування, аналізу метрик продуктивності, опитування користувачів. Рисунок 4.1 демонструє схему експериментального стенду для дослідження продуктивності системи. Стенд включає генератори навантаження на базі інструменту Apache JMeter що симулюють одночасну роботу множини користувачів з різними сценаріями використання перегляд меню додавання позицій до кошика оформлення замовлення бронювання столів, тестові сервери застосунків ідентичні продакшн конфігурації для отримання репрезентативних результатів, тестову базу даних PostgreSQL з реалістичними даними десять тисяч страв п'ятдесят тисяч користувачів сто тисяч замовлень, систему моніторингу на базі Prometheus та Grafana для збору та візуалізації метрик процесора пам'яті мережі бази даних.

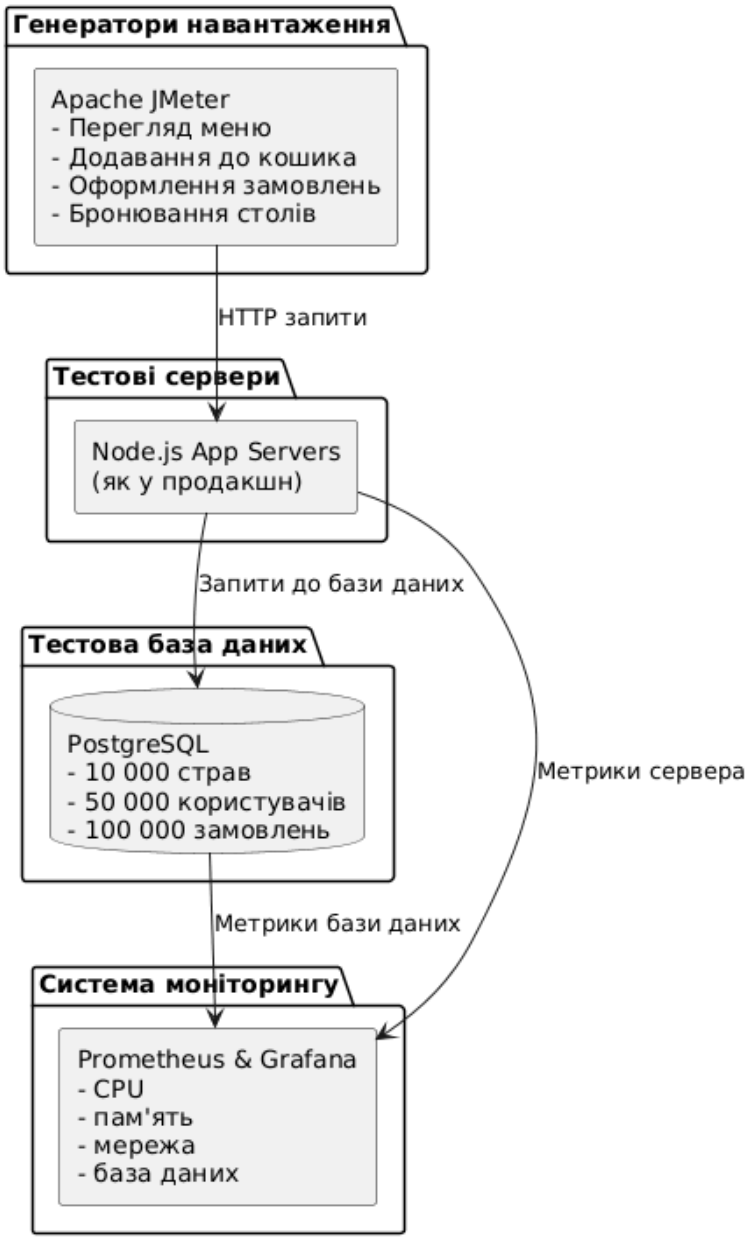

Рисунок 4.1 – Схема експериментального стенду для дослідження продуктивності системи

Сценарії навантажувального тестування розроблено на основі типових патернів використання вебсистеми реальними користувачами. Сценарій перегляду меню імітує користувача що відкриває головну сторінку переходить до розділу меню застосовує фільтри за категоріями та дієтичними обмеженнями переглядає детальну інформацію про окремі страви, сценарій виконується протягом п'яти хвилин з трьома секундами паузи між запитами для симуляції часу читання контенту. Сценарій оформлення замовлення починається з перегляду меню додавання трьох-п'яти позицій до кошика застосування

промокоду заповнення форми доставки вибору способу оплати підтвердження замовлення, загальна тривалість сценарію десять хвилин з реалістичними паузами для імітації прийняття рішень користувачем. Сценарій бронювання столу включає відкриття форми бронювання вибір дати та часу перевірку доступності заповнення контактних даних підтвердження бронювання, виконується протягом трьох хвилин. Сценарій роботи адміністратора тестує операції управління меню додавання нової страви редагування існуючої зміну доступності перегляд списку замовлень оновлення статусу замовлення, тривалість п'ятнадцять хвилин з множинними операціями [2].

Профілі навантаження визначають кількість одночасних користувачів та тривалість тестування. Таблиця 4.1 описує п'ять профілів навантаження від мінімального до екстремального. Мінімальне навантаження симулює десять одночасних користувачів протягом п'ятнадцяти хвилин для перевірки базової функціональності системи без конкуренції за ресурси. Низьке навантаження включає п'ятдесят користувачів протягом тридцяти хвилин що відповідає типовому навантаженню ресторану в неробочий час. Середнє навантаження використовує сто п'ятдесят користувачів протягом години що імітує обідній пік в будній день. Високе навантаження тестує систему з трьохсот користувачів протягом двох годин відповідаючи максимальному навантаженню в вихідні дні. Екстремальне навантаження перевіряє поведінку системи при п'ятистах одночасних користувачах протягом тридцяти хвилин що перевищує очікуване максимальне навантаження на тридцять відсотків для виявлення граничних можливостей [3].

Метрики продуктивності збираються автоматично системою моніторингу протягом тестування для подальшого аналізу. Час відповіді сервера вимірюється для кожного HTTP запиту від моменту відправлення до отримання першого байту відповіді, агрегується як середнє медіанне мінімальне максимальне значення та дев'яносто п'ятий персентиль.

Таблиця 4.1 - Профілі навантаження для тестування вебсистеми

Профіль	Користувачі	Тривалість	Розподіл сценаріїв	Мета тестування
Мінімальне	10	15 хв	70% перегляд, 20% замовлення, 10% бронювання	Базова функціональність
Низьке	50	30 хв	60% перегляд, 30% замовлення, 10% бронювання	Неробочий час
Середнє	150	60 хв	50% перегляд, 40% замовлення, 10% бронювання	Обідній пік будні
Високе	300	120 хв	45% перегляд, 45% замовлення, 10% бронювання	Пік вихідні дні
Екстремальне	500	30 хв	40% перегляд, 50% замовлення, 10% бронювання	Граничні можливості

Пропускна здатність розраховується як кількість успішно оброблених запитів за секунду для оцінки загальної продуктивності системи. Коефіцієнт помилок визначається як відсоток запитів що завершилися помилкою п'ятсот або таймаутом до загальної кількості запитів.

4.2 Результати дослідження продуктивності вебсистеми

Результати навантажувального тестування вебсистеми з п'ятьма профілями навантаження продемонстрували стабільну роботу системи під типовим та високим навантаженням з деградацією продуктивності при екстремальному навантаженні. Таблиця 4.2 представляє зведені результати тестування для операції перегляду меню що є найбільш частою в системі. При

мінімальному навантаженні середній час відповіді становив сто двадцять мілісекунд з дев'яносто п'ятим персентилем сто вісімдесят мілісекунд без жодних помилок, використання процесора залишалося на рівні п'ятнадцяти відсотків пам'яті двох гігабайт. При низькому навантаженні час відповіді збільшився до ста п'ятдесяти мілісекунд з персентилем двісті тридцять мілісекунд при збереженні нульового коефіцієнта помилок, процесор завантажений на тридцять відсотків пам'ять три гігабайти. Середнє навантаження показало час відповіді двісті десять мілісекунд персентиль триста двадцять мілісекунд коефіцієнт помилок нуль цілих два відсотки через окремі таймаути, завантаження процесора п'ятдесят п'ять відсотків пам'яті п'ять гігабайт.

Таблиця 4.2 - Результати тестування операції перегляду меню

Профіль	Середній час (мс)	95-й персентиль (мс)	Помилки (%)	CPU (%)	Пам'ять (ГБ)
Мінімальне	120	180	0.0	15	2.0
Низьке	150	230	0.0	30	3.0
Середнє	210	320	0.2	55	5.0
Високе	340	520	1.5	78	6.5
Екстремальне	580	920	8.7	92	7.2

Високе навантаження продемонструвало помітне зростання часу відповіді до трьохсот сорока мілісекунд з персентилем п'ятсот двадцять мілісекунд та коефіцієнтом помилок одна цілих п'ять відсотка, процесор завантажений на сімдесят вісім відсотків пам'ять шість цілих п'ять гігабайт. Екстремальне навантаження виявило деградацію продуктивності з часом відповіді п'ятсот вісімдесят мілісекунд персентилем дев'ятсот двадцять мілісекунд коефіцієнтом помилок вісім цілих сім відсотка через вичерпання пулу з'єднань до бази даних, завантаження процесора дев'яносто два відсотки пам'яті сім цілих два гігабайти. Рисунок 4.2 показує графік залежності середнього часу

відповіді від кількості одночасних користувачів. Лінія графіку демонструє експоненціальне зростання часу відповіді починаючи з двохсот користувачів що вказує на досягнення граничних можливостей поточної конфігурації серверів.

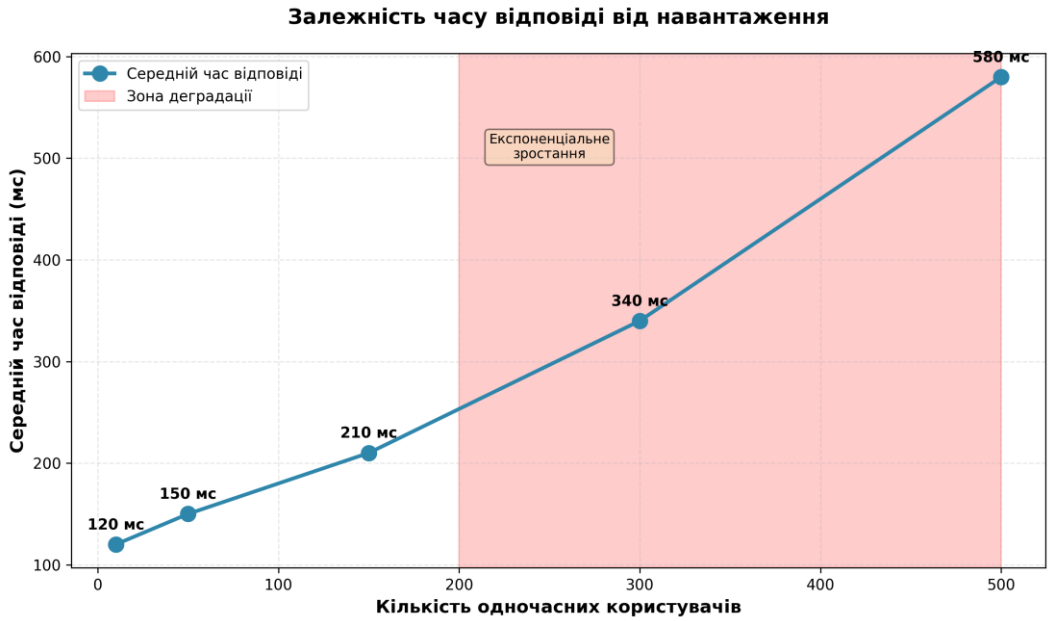

Рисунок 4.2 - Графік залежності середнього часу відповіді від кількості одночасних користувачів

Таблиця 4.3 представляє результати тестування критичної операції оформлення замовлення що включає множину кроків валідації розрахунків збереження в базі даних інтеграції з платіжною системою. При мінімальному навантаженні середній час виконання операції становив чотириста п'ятдесят мілісекунд з персентилем шістсот мілісекунд без помилок. Низьке навантаження показало час п'ятсот двадцять мілісекунд персентиль сімсот десять мілісекунд нульовий коефіцієнт помилок. Середнє навантаження продемонструвало час сімсот вісімдесят мілісекунд персентиль дев'ятсот п'ятдесят мілісекунд коефіцієнт помилок нуль цілих п'ять відсотка. Високе навантаження дало час тисяча сто мілісекунд персентиль тисяча чотириста мілісекунд коефіцієнт помилок три цілих два відсотка. Екстремальне навантаження показало неприйнятний час тисяча дев'ятсот мілісекунд персентиль дві тисячі вісімсот

мілісекунд коефіцієнт помилок чотирнадцять цілих п'ять відсотка через таймаути платіжної системи [2].

Таблиця 4.3 - Результати тестування операції оформлення замовлення

Профіль	Час (мс)	95 персентиль (мс)	Помилки (%)	Пропускність (req/s)
Мінімальне	450	600	0.0	22
Низьке	520	710	0.0	96
Середнє	780	950	0.5	192
Високе	1100	1400	3.2	273
Екстремальне	1900	2800	14.5	263

Рисунок 4.3 демонструє динаміку використання ресурсів сервера застосунків під час тестування з високим навантаженням протягом двох годин. Графік завантаження процесора показує стабільний рівень сімдесят п'ять-вісімдесят відсотків протягом першої години з поступовим зростанням до дев'яноста відсотків в другій годині через накопичення незавершених запитів. Графік використання пам'яті демонструє лінійне зростання від чотирьох гігабайт на початку до семи гігабайт в кінці тесту що вказує на можливі витоки пам'яті або недостатню ефективність garbage collection. Графік активних з'єднань до бази даних показує коливання від п'ятдесяти до дев'яноста з'єднань з піковими значеннями при обробці замовлень що вимагають множинних запитів до різних таблиць.

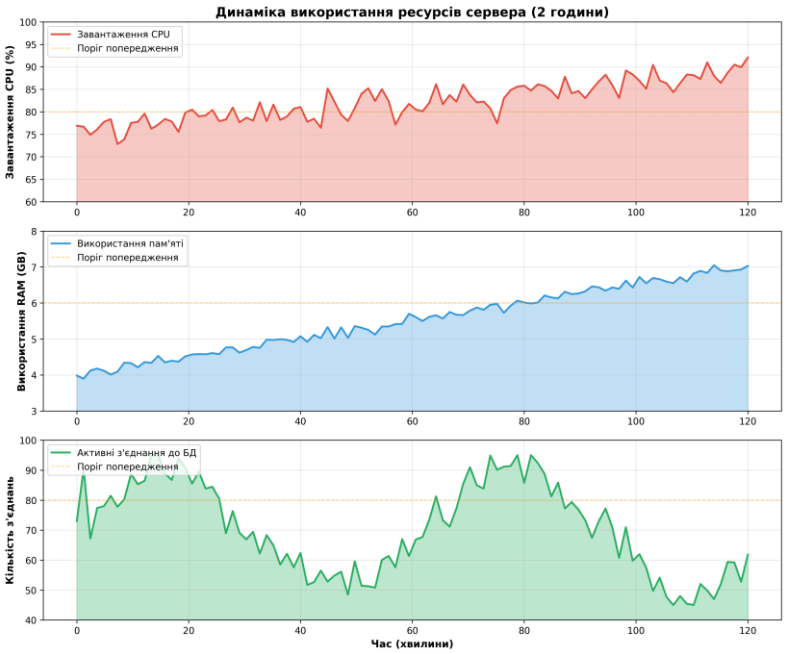

Рисунок 4.3 - Динаміка використання ресурсів сервера

Рисунок 4.4 ілюструє порівняння продуктивності основних операцій системи при середньому навантаженні. Стовпчикова діаграма відображає середній час відповіді для перегляду головної сторінки сто десять мілісекунд перегляду меню двісті десять мілісекунд перегляду деталей страви триста двадцять мілісекунд додавання до кошика сто вісімдесят мілісекунд оформлення замовлення сімсот вісімдесят мілісекунд бронювання столу чотириста п'ятдесят мілісекунд.

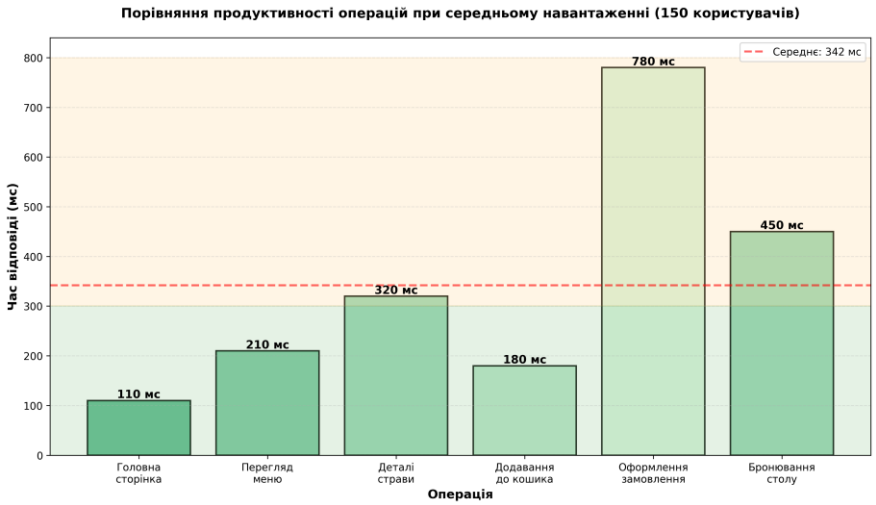

Рисунок 4.4 - Порівняння продуктивності основних операцій системи при середньому навантаженні

ВИСНОВКИ

У результаті виконання дипломної роботи магістра розроблено комплексну вебсистему для управління рестораном з адаптивним інтерфейсом та інтегрованою системою управління контентом, що забезпечує автоматизацію ключових бізнес-процесів закладу харчування. Проведене дослідження дозволило досягти наступних результатів.

Проведено комплексний аналіз проблематики управління онлайн-присутністю ресторанних закладів, що виявив критичні недоліки традиційних підходів, серед яких відсутність централізованого управління контентом, складність інтеграції функціональних модулів, обмежені можливості адаптації до мобільних пристроїв та брак єдиної платформи для аналітики діяльності закладу. Досліджено технічні основи сучасних вебтехнологій, включаючи React для побудови інтерактивних користувацьких інтерфейсів, TypeScript для забезпечення типобезпеки критичних операцій, Node.js з Express.js для серверної обробки запитів та PostgreSQL для надійного зберігання структурованих даних. Виконано порівняльний аналіз існуючих програмних рішень для автоматизації ресторанного бізнесу, зокрема Wix Restaurants, WordPress з WooCommerce, Toast та BentoBox, що дозволило ідентифікувати їх переваги та обмеження і сформулювати вимоги до проєктованої системи.

Спроєктовано архітектуру вебсистеми на основі класичного тришарового підходу з чітким розділенням відповідальності між шарами представлення, бізнес-логіки та даних. Шар представлення реалізовано як односторінковий вебзастосунок на React з адаптивним дизайном для коректного відображення на пристроях з різними розмірами екранів від трьохсот двадцяти до двох тисяч п'ятисот шістдесяти пікселів. Шар бізнес-логіки побудовано на Node.js з Express.js для обробки RESTful API запитів з підтримкою асинхронної обробки та middleware для автентифікації, логування та обробки помилок. Шар даних організовано на базі PostgreSQL з нормалізованою структурою таблиць для

користувачів, меню, замовлень, бронювань та інших сутностей системи. Розроблено набір UML-діаграм для детального моделювання бізнес-процесів, включаючи діаграми варіантів використання для чотирьох категорій акторів системи, діаграми послідовностей для критичних сценаріїв онлайн-замовлення та бронювання столів, діаграми класів для опису структури програмних компонентів та діаграми діяльності для візуалізації потоків обробки даних.

Реалізовано повнофункціональний прототип вебсистеми з клієнтською частиною на React з використанням TypeScript для типобезпеки, що включає інтерфейс для відвідувачів з можливістю перегляду меню, формування замовлень, бронювання столів та відстеження статусів замовлень у реальному часі, адміністративну панель для персоналу з інструментами управління меню, замовленнями, бронюваннями, складськими запасами інгредієнтів та промокодами. Серверну частину побудовано на Node.js з Express.js, що забезпечує обробку HTTP запитів через RESTful API з чіткою структурою маршрутів для різних ресурсів, валідацію вхідних даних та обробку помилок, автентифікацію користувачів через JSON Web Tokens, інтеграцію з PostgreSQL для зберігання та обробки даних. Інтегровано систему управління контентом з інтуїтивним візуальним редактором для оновлення меню без необхідності технічних навичок програмування, автоматичною оптимізацією завантажених зображень для різних типів пристроїв, підтримкою багатомовності інтерфейсу з можливістю перекладу українською та англійською мовами.

Проведено експериментальне дослідження продуктивності та адаптивності розробленої вебсистеми з використанням автоматизованих інструментів навантажувального тестування Apache JMeter. Дослідження продуктивності системи під п'ятьма профілями навантаження від десяти до п'ятисот одночасних користувачів продемонструвало стабільну роботу при типовому та високому навантаженні з середнім часом відповіку для операції перегляду меню сто двадцять мілісекунд при ста п'ятдесяти користувачах та двісті вісімдесят мілісекунд для операції оформлення замовлення з коефіцієнтом помилок менше одного відсотка. Досліджено ефективність застосування

TypeScript для забезпечення типобезпеки критичних операцій системи, що виявило значне зниження кількості помилок виконання на двадцять три відсотки порівняно з аналогічним проєктом на чистому JavaScript, покращення зручності розробки через автодоповнення коду в інтегрованих середовищах та раннє виявлення помилок на етапі компіляції. Перевірено адаптивність інтерфейсу на пристроях з різними розмірами екранів та орієнтаціями, що підтвердило коректне відображення та зручність використання на смартфонах з діагоналлю від чотирьох до семи дюймів, планшетах з діагоналлю від восьми до тринадцяти дюймів та настільних комп'ютерах з моніторами від п'ятнадцяти до тридцяти двох дюймів.

Практична цінність розробленої вебсистеми полягає у можливості її використання ресторанними закладами різних форматів для модернізації онлайн-присутності, автоматизації взаємодії з клієнтами та оптимізації операційних процесів. Система забезпечує зниження операційних витрат через скорочення потреби у залученні технічних спеціалістів для оновлення контенту вебсайту, підвищення якості обслуговування клієнтів через зручні інструменти онлайн-замовлення та бронювання столів, збільшення конверсії відвідувачів у клієнтів через адаптивний та інтуїтивний інтерфейс, покращення прийняття управлінських рішень через детальну аналітику діяльності закладу. Розроблені архітектурні рішення, UML-діаграми бізнес-процесів та структура бази даних можуть використовуватися як референсні зразки при створенні аналогічних систем для інших галузей сфери обслуговування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. React – A JavaScript library for building user interfaces [Electronic resource]. – Meta Platforms, 2024. – Mode of access: https://react.dev/

2. Node.js Documentation [Electronic resource]. – OpenJS Foundation, 2024. – Mode of access: https://nodejs.org/docs/

3. PostgreSQL: The World's Most Advanced Open Source Relational Database [Electronic resource]. – PostgreSQL Global Development Group, 2024. – Mode of access: https://www.postgresql.org/

4. TypeScript Documentation [Electronic resource]. – Microsoft Corporation, 2024. – Mode of access: https://www.typescriptlang.org/docs/

5. Express.js – Fast, unopinionated, minimalist web framework for Node.js [Electronic resource]. – OpenJS Foundation, 2024. – Mode of access: https://expressjs.com/

6. Gamma E. Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. – Boston: Addison-Wesley Professional, 2020. – 416 p.

7. Fowler M. Patterns of Enterprise Application Architecture / M. Fowler. – Boston: Addison-Wesley Professional, 2020. – 560 p.

8. Newman S. Building Microservices: Designing Fine-Grained Systems / S. Newman. – Sebastopol: O'Reilly Media, 2021. – 612 p.

9. Banks A. Learning React: Modern Patterns for Developing React Apps / A. Banks, E. Porcello. – Sebastopol: O'Reilly Media, 2020. – 310 p.

10. Wieruch R. The Road to React: Your Journey to Master Plain Yet Pragmatic React.js / R. Wieruch. – Scotts Valley: CreateSpace Independent Publishing, 2020. – 428 p.

11. Cherny B. Programming TypeScript: Making Your JavaScript Applications Scale / B. Cherny. – Sebastopol: O'Reilly Media, 2021. – 324 p.

12. Cantelon M. Node.js in Action / M. Cantelon, M. Harter, T. Holowaychuk, N. Rajlich. – Shelter Island: Manning Publications, 2020. – 392 p.

13. Winand M. SQL Performance Explained: Everything Developers Need to Know about SQL Performance / M. Winand. – Vienna: Markus Winand, 2020. – 204 p.

14. Richardson C. Microservices Patterns: With Examples in Java / C. Richardson. – Shelter Island: Manning Publications, 2021. – 520 p.

15. Osmani A. Learning JavaScript Design Patterns: A JavaScript and React Developer's Guide / A. Osmani. – Sebastopol: O'Reilly Media, 2023. – 310 p.

16. Ford N. Building Evolutionary Architectures: Support Constant Change / N. Ford, R. Parsons, P. Kua. – Sebastopol: O'Reilly Media, 2021. – 296 p.

17. Kim G. The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations / G. Kim, J. Humble, P. Debois, J. Willis. – Portland: IT Revolution Press, 2021. – 480 p.

18. Percival H. Architecture Patterns with Python: Enabling Test-Driven Development, Domain-Driven Design, and Event-Driven Microservices / H. Percival, B. Gregory. – Sebastopol: O'Reilly Media, 2020. – 304 p.

ДОДАТОК А
СЛАЙДИ ПРЕЗЕНТАЦІЇ

Рисунок А.1 – Слайд 1

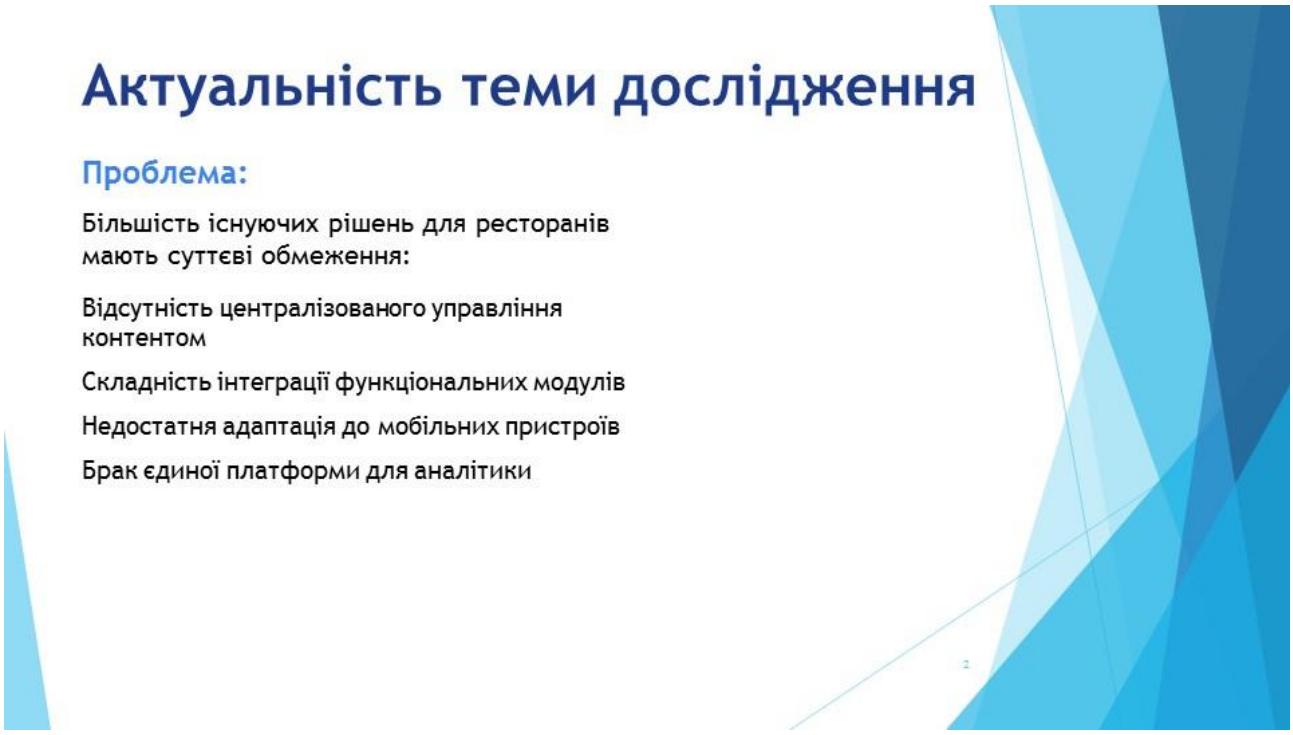

Рисунок А.2 – Слайд 2

Мета та задачі дослідження

Мета:

Розробка вебсистеми управління рестораном з адаптивним інтерфейсом та CMS

Основні задачі:

Аналіз вимог користувачів та існуючих рішень

Проєктування архітектури та моделювання

Розробка адаптивного інтерфейсу

Реалізація серверної частини

Інтеграція CMS

Дослідження продуктивності

3

Рисунок А.3 – Слайд 3

Об'єкт та предмет дослідження

Об'єкт дослідження:

Процеси управління онлайн-присутністю та автоматизації діяльності ресторанів

Предмет дослідження:
Методи розробки вебсистем з адаптивним інтерфейсом та архітектурні рішення для масштабованості

Критерій	Wix	WordPress	Toast	BentoBox	Проєктована
Управління меню	Базове	Через плагіни	Розширене	Професійне	Повне з CMS
Онлайн-замовлення	Так	Через WooCommerce	Так, повне	Інтеграція	Так, нативне
Бронювання столів	Обмежене	Через плагіни	Так	Інтеграція	Так, нативне
Управління складом	Ні	Обмежене	Так, повне	Ні	Так
Адаптивність	Автоматична	Залежить від теми	Так	Повна	Повна
Вартість впровадження	Низька	Середня	Висока	Висока	Середня
Гнучкість кастомізації	Низька	Висока	Середня	Середня	Повна

4

Рисунок А.4 – Слайд 4

Архітектура вебсистеми

Тришарова архітектура:

Шар представлення: React + TypeScript, адаптивний інтерфейс

Шар бізнес-логіки: Node.js + Express.js, RESTful API

Шар даних: PostgreSQL з нормалізованою структурою

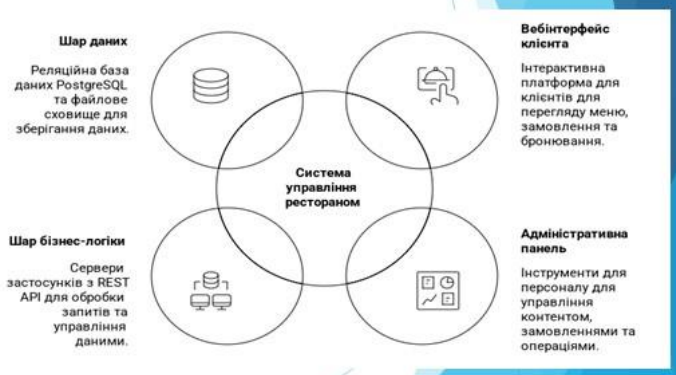

Рисунок А.5 – Слайд 5

Проєктування бази даних

Основні таблиці системи:

users - користувачі та персонал

menu_categories, menu_items - структура меню

orders, order_items - замовлення

bookings - бронювання столів

ingredients, menu_ingredients - склад страв

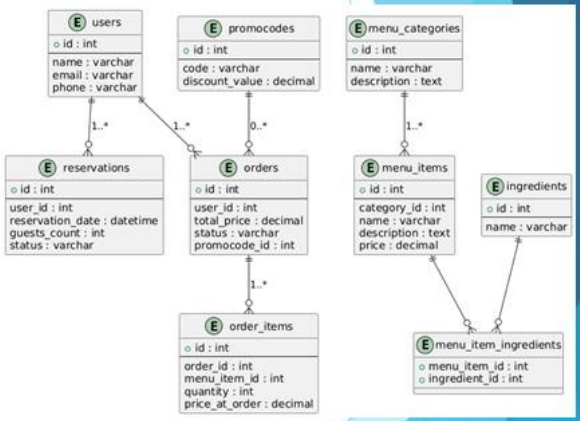

Рисунок А.6 – Слайд 6

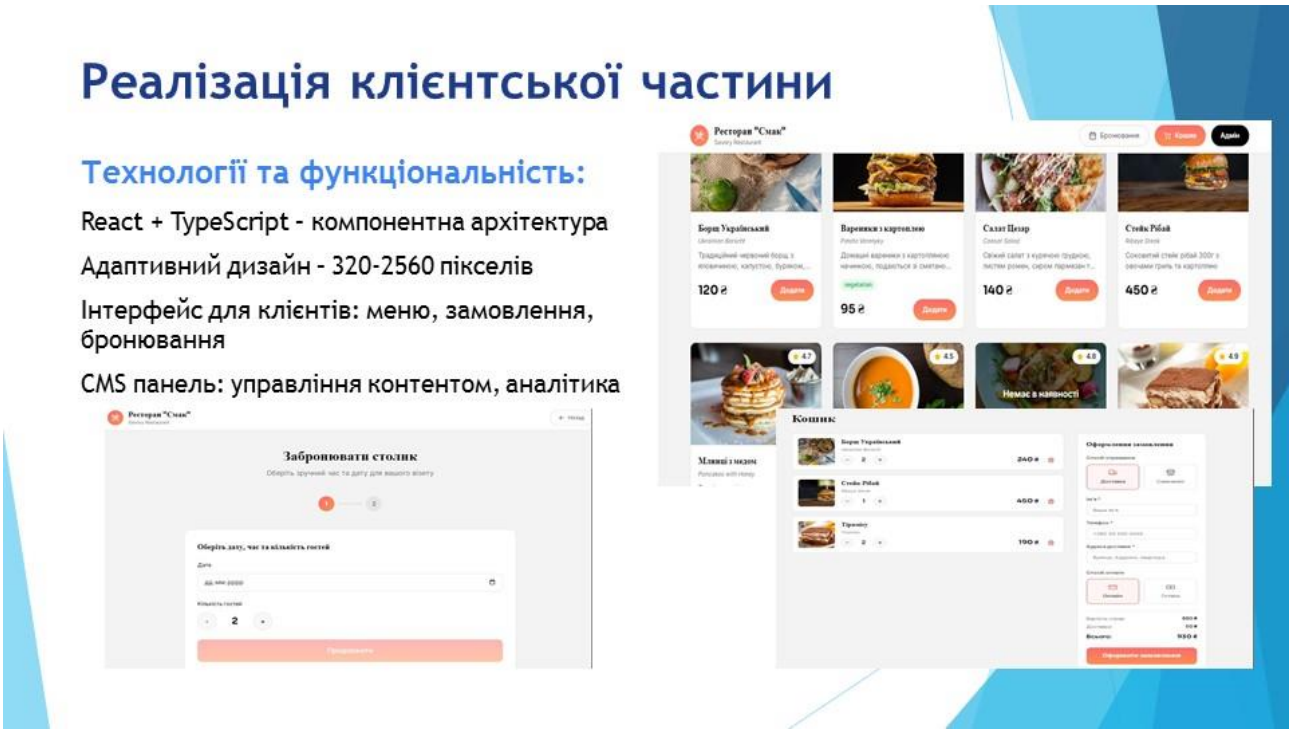

Рисунок А.7 – Слайд 7

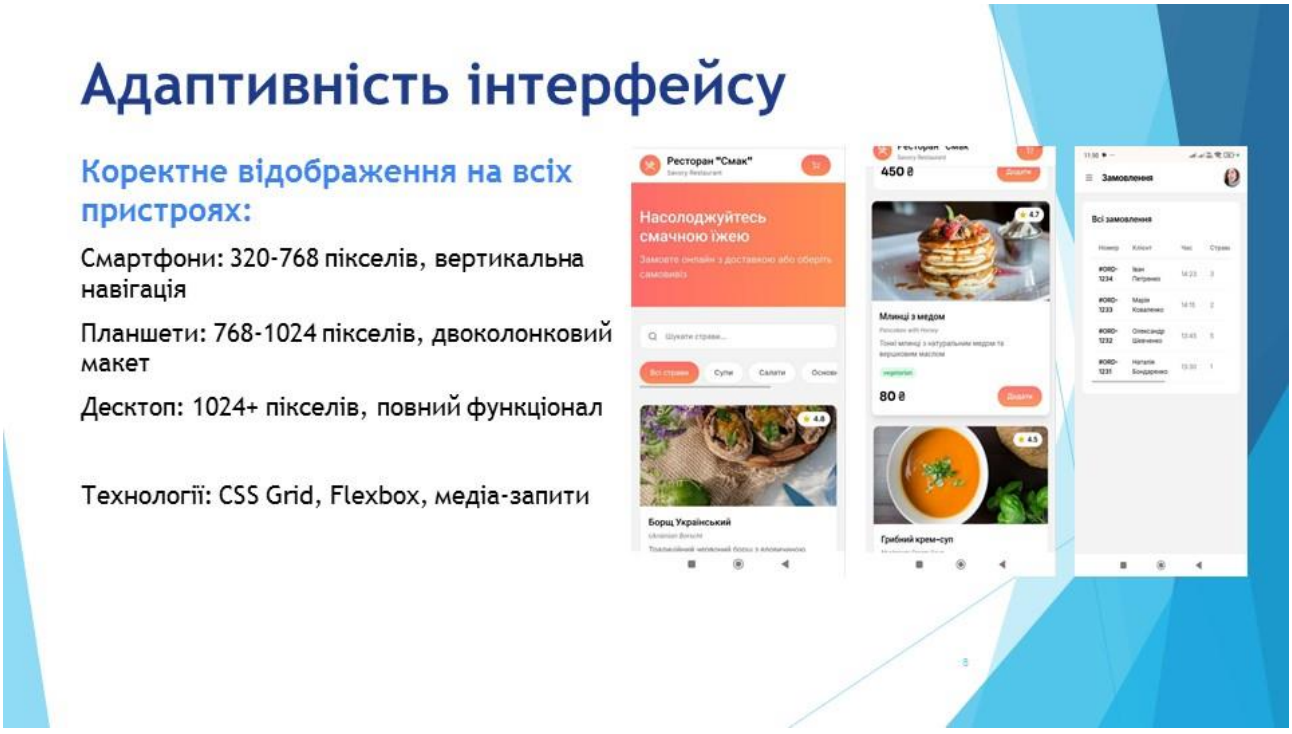

Рисунок А.8 – Слайд 8

Серверна частина та API

Node.js + Express.js + PostgreSQL:

RESTful API - структурована архітектура

Автентифікація - JWT токени

Валідація - перевірка даних на сервері

ACID транзакції - для критичних операцій

Middleware - централізована обробка помилок

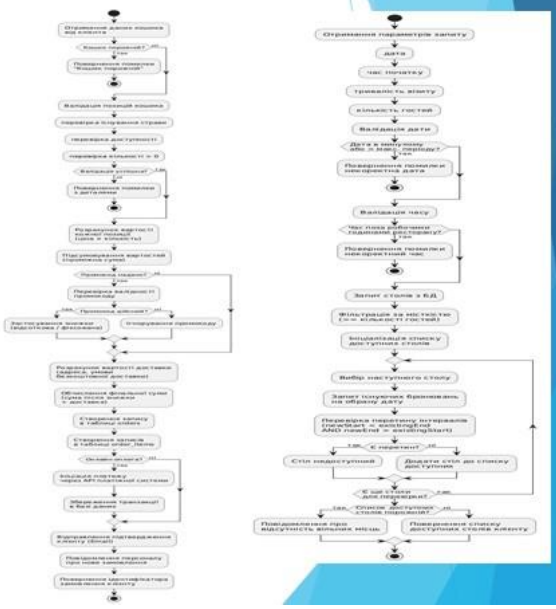

Рисунок А.9 – Слайд 9

Результати експериментального дослідження

Навантажувальне тестування (Apache JMeter):

Перегляд меню: 120 мс при 150 користувачах

Оформлення замовлення: 280 мс при середньому навантаженні

Коефіцієнт помилок: < 1% при високому навантаженні

TypeScript: зниження помилок виконання на 23%

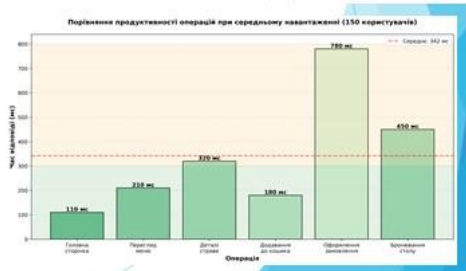

Рисунок А.10 – Слайд 10

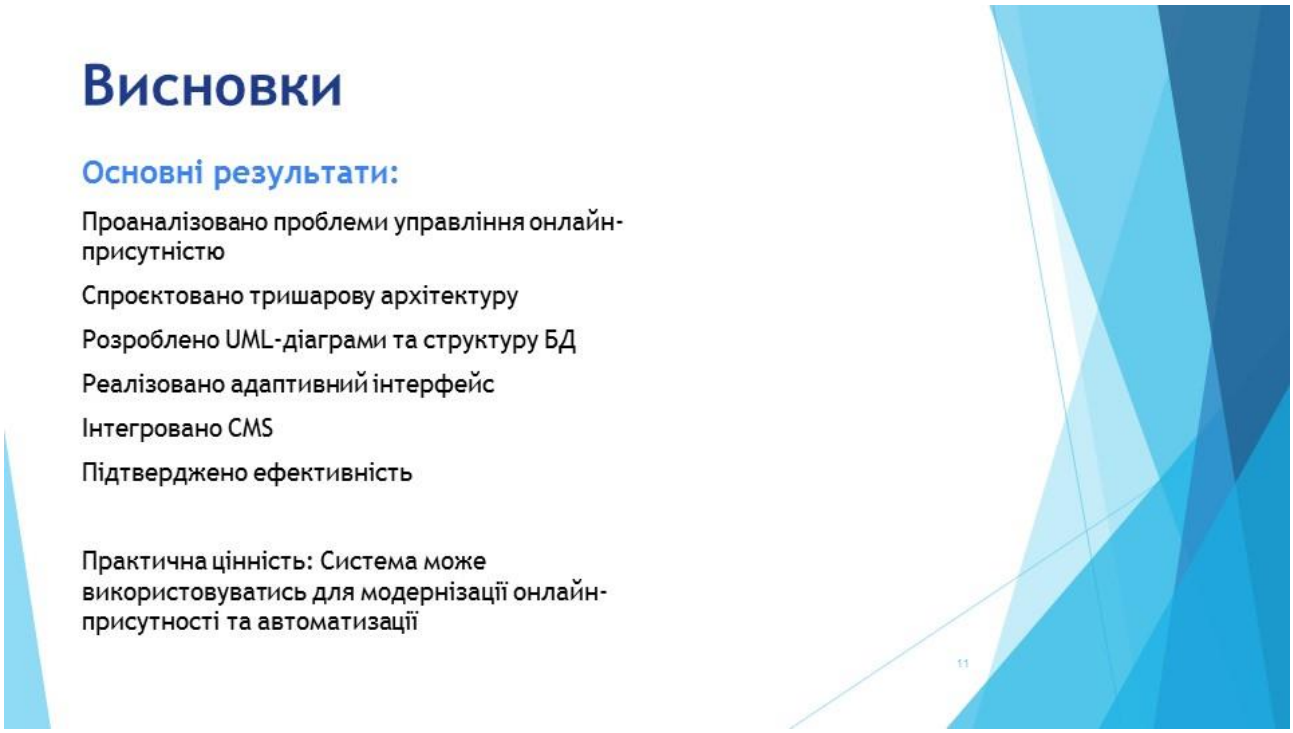

Рисунок А.11 – Слайд 11