

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для безконтактного оформлення замовлень та керування ними у ресторані»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

*Кваліфікаційна робота містить результати власних досліджень.
Використання ідей, результатів і текстів інших авторів мають посилання
на відповідне джерело*

_____ Ілля БОВКУН
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

_____ Ілля БОВКУН

Керівник: _____ Тимур ДОВЖЕНКО
канд.техн.наук, доц.

Рецензент: _____

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ

« ____ » _____ 2026 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Бовкуну Іллі Віталійовичу

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для безконтактного оформлення замовлень та керування ними у ресторані»
керівник кваліфікаційної роботи канд.техн.наук, доц. Тимур ДОВЖЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.
2. Строк подання кваліфікаційної роботи «29» травня 2026 року.
3. Вихідні дані до кваліфікаційної роботи: технічна документація та наукові публікації з питань автоматизації процесів обслуговування у закладах громадського харчування, огляд існуючих програмних рішень (ChoiceQR, Poster POS, Toast POS, Lightspeed), специфікації протоколів OAuth 2.0/OIDC та Server-Sent Events, технічна документація фреймворку Spring Boot, бібліотеки HTMX та платіжного шлюзу LiqPay API.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
 1. Аналіз предметної галузі безконтактного обслуговування клієнтів у закладах громадського харчування, огляд та порівняння існуючих програмних рішень, формулювання вимог до розроблюваної системи та вибір засобів реалізації.

2. Проєктування системи SmartDine: розробка архітектури, моделювання варіантів використання та діаграм послідовностей, проєктування бази даних і структури інтерфейсу користувача.

3. Програмна реалізація та опис функціонування системи SmartDine: реалізація серверної частини, клієнтського інтерфейсу, інтеграція зовнішніх сервісів та модуля лояльності.

4. Тестування та розгортання системи.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.

2. Вимоги до програмного забезпечення.

3. Програмні засоби реалізації.

4. Діаграма варіантів використання.

5. Діаграма компонентів.

6. Діаграма класів.

7. ER-діаграма бази даних.

8. Діаграма послідовності оформлення замовлення

9. Діаграма послідовності оплати замовлення

10. Екранні форми.

11. Тестування.

12. Розгортання.

13. Апробація результатів дослідження.

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Проєктування архітектури системи автоматизації ресторану	08.03-19.03.2026	
4	Програмна реалізація серверної частини та інтеграція платіжного модуля	20.03-05.04.2026	
5	Розробка адаптивного клієнтського інтерфейсу	06.04-19.04.2026	
6	Комплексне тестування системи та налаштування середовища розгортання	20.04-29.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	30.04-06.05.2026	
8	Розробка демонстраційних матеріалів	07.05-10.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	
10	Подання кваліфікаційної роботи	23.05-30.05.2026	

Здобувач вищої освіти

_____ (підпис)

Ілля БОВКУН

Керівник

кваліфікаційної роботи

_____ (підпис)

Тимур ДОВЖЕНКО

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 69 стор., 3 табл., 18 рис., 23 джерела.

Мета роботи – покращення процесу безконтактного обслуговування клієнтів та замовлення страв у закладах громадського харчування шляхом розробки веб-орієнтованої системи безконтактного самообслуговування SmartDine з підтримкою сповіщень у реальному часі на основі сучасного стеку технологій.

Об'єкт дослідження – процес безконтактного обслуговування клієнтів у закладах громадського харчування.

Предмет дослідження – програмне забезпечення для автоматизації процесу безконтактного обслуговування клієнтів у ресторанах та кафе.

Короткий зміст роботи: у роботі досліджено особливості процесу обслуговування клієнтів у сучасних закладах громадського харчування, визначено основні проблеми традиційної моделі прийому замовлень та проаналізовано існуючі програмні рішення для їх автоматизації. Проаналізовано інструментальні засоби для автоматизації замовлень у ресторані: ChoiceQR, Poster POS, Toast POS, Lightspeed Restaurant. Розроблено архітектуру системи та спроектовано базу даних із 15 таблицями. Програмно реалізовано ключові функціональні можливості, зокрема: перегляд цифрового меню через QR-код, формування замовлення та онлайн-оплата через LiqPay, оновлення статусу замовлення у реальному часі засобами механізму передачі подій від сервера (Server-Sent Events), система лояльності з нарахуванням бонусних балів, система вебсповіщень для персоналу.

Проведено модульне, інтеграційне, наскрізне (E2E) та навантажувальне тестування. У роботі використано Java 25 та Spring Boot для серверної частини, Thymeleaf та HTMX для клієнтського інтерфейсу, PostgreSQL для зберігання даних, Keycloak для управління ідентифікацією та Nginx як зворотний проксі-сервер.

Сферою використання застосунку є автоматизація процесів безконтактного самообслуговування клієнтів у закладах громадського харчування: ресторанах та кафе.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, БЕЗКОНТАКТНЕ ОБСЛУГОВУВАННЯ, PROGRESSIVE WEB APP (PWA), HTMX, SERVER-SENT EVENTS (SSE), WEB PUSH, SPRING BOOT, KEYCLOAK, OIDC, QR-КОД, СИСТЕМА ЛОЯЛЬНОСТІ, ОНЛАЙН-ОПЛАТА.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	11
ВСТУП.....	12
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ВИМОГ ДО СИСТЕМИ БЕЗКОНТАКТНОГО ОБСЛУГОВУВАННЯ.....	14
1.1 Аналіз бізнес-процесів у закладі громадського харчування.....	14
1.1.1 Характеристика традиційної моделі обслуговування.....	15
1.1.2 Цільова модель бізнес-процесу у безконтактній системі.....	16
1.1.3 Організація структури закладу.....	17
1.1.4 Аналіз цільової аудиторії та сегментація ринку.....	18
1.1.5 Огляд наукових досліджень у галузі.....	19
1.2 Аналіз існуючих програмних аналогів.....	20
1.2.1 Poster POS.....	21
1.2.2 ChoiceQR.....	23
1.2.3 Toast POS.....	24
1.2.4 Lightspeed Restaurant.....	25
1.2.5 Зведений порівняльний аналіз.....	27
1.3 Формулювання вимог до вебзастосунку.....	29
1.3.1 Функціональні вимоги.....	29
1.3.2 Нефункціональні вимоги.....	30
1.4 Засоби реалізації застосунку.....	32
1.4.1. Backend-технології.....	32
1.4.2. Засоби керування базами даних.....	33
1.4.3. Frontend-технології.....	33
1.4.4. Засоби автентифікації та зовнішні сервіси.....	34
1.4.5. Засоби тестування, контейнеризації та розгортання.....	34

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА МОДЕЛЮВАННЯ ПРОЦЕСІВ БЕЗКОНТАКТНОГО ОФОРМЛЕННЯ ЗАМОВЛЕНЬ.....	35
2.1 Функціональне моделювання (Use Case).....	35
2.1.1 Діаграма варіантів використання.....	36
2.1.2 Опис логіки розмежування доступу.....	38
2.2 Архітектурне моделювання.....	39
2.2.1 Діаграма компонентів.....	40
2.2.2 Діаграма класів.....	43
2.2.3 Обґрунтування архітектурних рішень та компромісів.....	43
2.3 Проєктування бази даних.....	44
2.3.1 Типи зв'язків між таблицями.....	45
2.3.2 Стратегія індексування та ER-діаграма БД.....	46
2.4 Моделювання динаміки (діаграми послідовності).....	47
2.4.1 Сценарій А: Оформлення замовлення та SSE-сповіщення.....	47
2.4.2 Сценарій Б: Оплата замовлення через LiqPay.....	49
2.5 Проєктування інтерфейсу користувача.....	51
2.5.1 Сценарії виконання задач користувача (User Flows).....	52
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ ДЛЯ АВТОМАТИЗАЦІЇ РЕСТОРАННОГО СЕРВІСУ.....	55
3.1 Розробка серверної частини.....	55
3.1.1 Налаштування Spring Security та інтеграція з Keycloak.....	56
3.1.2 Сервісний шар та транзакційність.....	56
3.1.3 Механізм SSE-сповіщень та транзакційна безпека.....	57
3.1.4 Ідентифікація гостей та сесій та кешування.....	57
3.1.5 Віртуальні потоки, Flyway та розрахунок замовлення.....	58
3.2 Реалізація інтерактивного фронтенду.....	59
3.2.1 Концепція Server-Driven UI та PWA.....	59
3.2.2 Часткове оновлення DOM за допомогою HTMX.....	60
3.2.3 Механізм Server-Sent Events (SSE) для панелі персоналу.....	61

3.2.4 Оптимізація графічних ресурсів.....	63
3.2.5 Адаптивний дизайн та стилізація.....	64
3.2.6 Візуалізація аналітики.....	64
3.3. Інтеграція зовнішніх сервісів.....	65
3.3.1 Платіжний модуль LiqPay.....	65
3.3.2 Web Push сповіщення.....	67
3.3.3 Сервіс автентифікації Keycloak (Identity Provider).....	68
3.4 Персоналізація та система лояльності.....	68
3.4.1 Персоналізація меню та рекомендації страв.....	68
3.4.2 Механізм системи лояльності (бонусні бали).....	69
4 ТЕСТУВАННЯ ТА РОЗГОРТАННЯ СИСТЕМИ В ХМАРНІЙ ІНФРАСТРУКТУРІ.....	71
4.1 Методи тестування.....	71
4.1.2 Оцінка продуктивності клієнтської частини.....	72
4.1.2 Навантажувальне та стрес-тестування.....	73
4.2 Тест-кейси та результати їх виконання.....	75
4.3 Розгортання застосунку.....	77
4.3.1 Контейнеризація (Docker та Docker Compose).....	77
4.3.2 Налаштування вебсервера Nginx (зворотний проксі-сервер).....	78
4.3.3 Розгортання у хмарній інфраструктурі Azure.....	79
ВИСНОВКИ.....	80
ПЕРЕЛІК ПОСИЛАНЬ.....	83
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛ.....	85
ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ.....	94

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface
ACID – Atomicity, Consistency, Isolation, Durability
БД – База даних
DTO – Data Transfer Object
DOM – Document Object Model
E2E – End-to-End
HMAC – Hash-based Message Authentication Code
HSTS – HTTP Strict Transport Security
HTMX – High-power Text Markup eXtensions
HTTP – HyperText Transfer Protocol
HTTPS – HyperText Transfer Protocol Secure
JSON – JavaScript Object Notation
JWT – JSON Web Token
JPA – Java Persistence API
LCP – Largest Contentful Paint
MVC – Model-View-Controller
NoSQL – Not Only SQL
OIDC – OpenID Connect
ORM – Object-Relational Mapping
PWA – Progressive Web App
QR – Quick Response
RBAC – Role-Based Access Control
СКБД – Система керування базами даних
SPA – Single-Page Application
SQL – Structured Query Language
SSE – Server-Sent Events
SSL/TLS – Secure Sockets Layer / Transport Layer Security
UI – User Interface
UML – Unified Modeling Language
VAPID – Voluntary Application Server Identification
VUs – Virtual Users

ВСТУП

Актуальність: Сфера громадського харчування є одним із найбільш трудомістких секторів сервісної економіки, де традиційна модель обслуговування – паперове меню, усний прийом замовлень, ручні розрахунки – породжує системні проблеми: помилки персоналу при передачі замовлень, черги в пікові години та відсутність прозорого зворотного зв'язку між клієнтом і кухнею. Впровадження веб-орієнтованих систем самообслуговування з доступом через QR-код дозволяє автоматизувати процес прийому та обробки замовлень, скоротити час обслуговування та усунути людський фактор як основне джерело операційних помилок. Тема є актуальною в умовах цифровізації сервісного бізнесу та зростання попиту на безконтактні рішення, що набули особливого значення після пандемії COVID-19.

Об'єктом дослідження є процес безконтактного обслуговування клієнтів у закладах громадського харчування.

Предметом дослідження є програмне забезпечення для автоматизації процесу безконтактного обслуговування клієнтів у ресторанах та кафе.

Метою роботи є покращення процесу безконтактного обслуговування клієнтів та замовлення страв у закладах громадського харчування шляхом розробки веб-орієнтованої системи безконтактного самообслуговування SmartDine з підтримкою сповіщень у реальному часі на основі сучасного стеку технологій.

Методи дослідження: Першим кроком було проаналізовано існуючі рішення для автоматизації закладів громадського харчування (ChoiceQR, Poster POS, Toast POS, Lightspeed), виявлено їх функціональні обмеження та сформульовано функціональні й нефункціональні вимоги до розроблюваної системи.

Далі було проведено огляд набору технологій, що забезпечує ефективну реалізацію системи. Для розробки серверної частини обрано фреймворк Spring

Boot і мову програмування Java 25 із підтримкою віртуальних потоків Project Loom. Для зберігання даних використано реляційну СКБД PostgreSQL із версіонуванням схеми через Flyway. Клієнтський інтерфейс реалізовано за принципом інтерфейсу, керованого сервером (Server-Driven UI), із застосуванням шаблонізатора Thymeleaf та бібліотеки HTMX. Для управління ідентифікацією та доступом використано Keycloak (OIDC/OAuth 2.0), а для прийому онлайн-платежів – LiqPay API. Дослідження архітектури та реалізації системи здійснювалося безпосередньо під час етапів проєктування та розробки з використанням UML-нотації для побудови структурних і поведінкових діаграм.

Наукова новизна роботи визначається такими складовими: запропоновано підхід до побудови системи самообслуговування для ресторанів, що поєднує архітектуру інтерфейсу, керованого сервером (HTMX + Thymeleaf), із механізмом сповіщень у реальному часі на основі передачі подій від сервера (Server-Sent Events) без використання клієнтських JavaScript-фреймворків; розроблено метод забезпечення ідемпотентності операцій у системі нарахування бонусів лояльності із захистом від подвійного нарахування балів при мережевих збоях; обґрунтовано доцільність застосування платформи віртуальних потоків Project Loom для обробки масових з'єднань, що підтверджено навантажувальним тестуванням при навантаженні 1000 одночасних користувачів.

Практична значущість результатів полягає у можливості використання розробленої системи SmartDine у реальних закладах громадського харчування для автоматизації процесу прийому замовлень, скорочення часу обслуговування клієнтів, зменшення операційних помилок та надання адміністрації аналітичних даних про роботу закладу в режимі реального часу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ВИМОГ ДО СИСТЕМИ БЕЗКОНТАКТНОГО ОБСЛУГОВУВАННЯ

Створення будь-якого ефективного програмного продукту вимагає глибокого розуміння середовища, у якому він функціонуватиме. Індустрія громадського харчування характеризується високою динамічністю, жорсткою конкуренцією та постійним зростанням вимог до якості клієнтського сервісу. Щоб запропонувати технологічне рішення, яке дійсно оптимізує роботу закладу, необхідно дослідити поточний стан ринку, виявити слабкі місця традиційних бізнес-процесів та проаналізувати існуючі програмні аналоги. Саме ці аналітичні кроки формують фундамент для визначення чітких вимог до майбутньої системи та вибору оптимального стеку технологій.

1.1 Аналіз бізнес-процесів у закладі громадського харчування

Дослідження доцільно розпочати з аналізу безпосереднього середовища розгортання системи – сфери громадського харчування (сегмент HoReCa – Hotel, Restaurant, Cafe). Сьогодні ця індустрія є одним із найбільш активних рушіїв впровадження сучасних вебтехнологій та безконтактних інтерфейсів для автоматизації обслуговування клієнтів. Відповідно до відкритих даних Державної служби статистики України, станом на 2023 рік кількість активних закладів ресторанного господарства в Україні перевищує 25 тисяч одиниць, що підтверджує значний масштаб ринку та гостру конкурентну боротьбу між його учасниками.

Сучасний ресторанний бізнес перебуває під впливом двох ключових тенденцій: зростання операційних витрат (передусім витрат на оплату праці персоналу залу) та зміни моделей поведінки споживачів, які дедалі частіше

очікують від сервісу цифрової взаємодії та мінімального часу очікування. У постпандемічний період (починаючи з 2021 року) значно зріс попит на безконтактні технології обслуговування: за даними Національного ресторанного об'єднання США (NRA), понад 60% відвідувачів надають перевагу закладам із можливістю цифрового замовлення [1]. Зазначена тенденція є характерною і для українського ринку, де цифровізація та впровадження інноваційних безконтактних систем стали ключовим фактором виживання та розвитку підприємств в умовах сучасної економіки [2].

1.1.1 Характеристика традиційної моделі обслуговування

Традиційна (класична) модель обслуговування у закладах громадського харчування передбачає наявність офіціанта як центральної ланки ланцюга комунікації між клієнтом та кухнею. Узагальнений бізнес-процес у форматі «як було» (As-Was) охоплює наступну послідовність операцій:

1. Клієнт займає місце за столом та очікує на офіціанта для отримання меню.
2. Офіціант надає паперове меню, клієнт здійснює вибір.
3. Офіціант приймає замовлення усно або записує вручну та передає на кухню.
4. Кухар готує страви; офіціант подає їх до столу.
5. Клієнт запрошує офіціанта для розрахунку; оплата відбувається через платіжний термінал.

Зазначена модель має ряд системних недоліків, що безпосередньо впливають на ефективність роботи закладу:

– людський фактор та похибки: усне або рукописне фіксування замовлення є схильним до помилок – некоректно зазначені позиції, невірно зрозумілі побажання клієнта щодо модифікацій страв. За даними досліджень якості сервісу

Parasuraman et al., значна частка клієнтських скарг у ресторанній індустрії зумовлена саме помилками на етапі прийому та передачі замовлення [3].

– затримки під час пікового навантаження: показник оборотності столів (кількість обслужених сесій за столом на годину) критично залежить від доступності персоналу. У вечірні години або у вихідні дні, коли всі столи зайняті, кожен офіціант змушений паралельно обслуговувати кілька столів, що призводить до збільшення часу очікування на кожному з етапів. Дослідження Lee S. K. та Kim Y. H. підтверджують, що скорочення часу очікування є одним із ключових чинників підвищення лояльності клієнтів та стійкості бізнесу в умовах криз [4].

– висока вартість утримання персоналу залу: витрати на заробітну плату офіціантів у структурі операційних витрат ресторану становлять значну частку загального фонду оплати праці підприємства. Тому зменшення необхідної чисельності контактного персоналу залу за рахунок впровадження технологій цифровізації та самообслуговування є одним із ключових резервів підвищення рентабельності сучасного закладу, що підтверджується вітчизняними науковцями [2].

– відсутність синхронізації між підрозділами: у закладах без автоматизованих систем управління продажами передача замовлень між залом та кухнею здійснюється фізично (паперові талони), що унеможлиблює оперативний контроль стану виконання замовлення з боку управлінського персоналу.

1.1.2 Цільова модель бізнес-процесу у безконтактній системі

Впровадження безконтактної платформи на базі QR-кодів кардинально трансформує описаний вище процес. Узагальнена цільова модель «як є» безконтактного обслуговування передбачає наступну послідовність взаємодій:

Клієнт займає місце за столом → Сканує унікальний QR-код столу → Система ідентифікує стіл та відкриває сесію замовлення → Клієнт переглядає цифрове меню з фотографіями страв → Формує кошик та підтверджує замовлення

→ Замовлення автоматично надходить на кухонний термінал → Персонал кухні послідовно змінює стан замовлення → Клієнт та офіціант отримують оновлення стану в реальному часі → Клієнт здійснює онлайн-оплату без участі офіціанта.

Ключовою перевагою зазначеної моделі є виключення офіціанта з циклу прийому замовлення та розрахунку, що скорочує операційне навантаження на персонал і дозволяє перерозподілити його ресурси на задачі вищої якості (видача готових страв, контроль залу, вирішення нестандартних ситуацій).

1.1.3 Організація структури закладу

З точки зору функціональних ролей, що беруть участь у бізнес-процесах закладу, безконтактна система обслуговування охоплює чотири основних актори:

Гість (неавторизований клієнт) – відвідувач без облікового запису. Взаємодіє з системою виключно в межах поточної сесії: переглядає меню, формує замовлення та здійснює оплату. Ідентифікується за допомогою тимчасового ідентифікатора сесії, що не вимагає реєстрації.

Авторизований клієнт – зареєстрований користувач. Додатково має доступ до системи накопичувальної лояльності, персоналізованого фільтрування меню за харчовими алергенами та рекомендаційного модуля на основі попередніх замовлень.

Персонал (офіціант / кухар) – співробітник закладу. Офіціант моніторить стан замовлень у своїй зоні відповідальності через планшетний інтерфейс у режимі реального часу. Кухар управляє чергою приготування через окремий термінал, послідовно змінюючи стани замовлення, що автоматично ініціює сповіщення для клієнта та офіціанта.

Адміністратор – управлінський персонал. Здійснює налаштування системи, управління номенклатурою меню (додавання страв, зміна цін, доступність позицій), генерацію QR-кодів для столів та зон ресторану, а також має доступ до інформаційної панелі з ключовими фінансовими показниками.

1.1.4 Аналіз цільової аудиторії та сегментація ринку

Оскільки розроблювана платформа є комплексним рішенням для ресторанного бізнесу, її цільова аудиторія сегментується на два рівні: заклади громадського харчування як замовники системи та кінцеві відвідувачі ресторанів.

Вимоги сегменту «бізнес для бізнесу» (ресторани):

- Відмовостійкість: система має забезпечувати стабільну роботу під час пікових навантажень без деградації часу відповіді.
- Гнучкість конфігурації: можливість оперативної зміни меню, управління доступністю страв та прив'язки QR-кодів до конкретних столів або зон ресторану.
- Апаратна незалежність: функціонування у стандартних браузерях на пристроях будь-яких платформ без необхідності придбання спеціалізованого обладнання або встановлення нативних додатків.
- Інтеграційні можливості: підтримка підключення платіжних шлюзів та систем сповіщення персоналу.

Вимоги сегменту «бізнес для клієнта» (відвідувачі ресторанів):

- Нульовий бар'єр входу: взаємодія з системою розпочинається одразу після сканування QR-коду без необхідності реєстрації або завантаження мобільних додатків.
- Інтуїтивність інтерфейсу: навігація між категоріями меню, формування замовлення та підтвердження оплати мають бути доступними для аудиторії будь-якого вікового діапазону та рівня цифрової грамотності відповідно до стандартів доступності WCAG 2.1 [5].
- Синхронізація стану в реальному часі: клієнт має отримувати актуальний статус свого замовлення без необхідності вручну оновлювати сторінку.

Обмеження предметної галузі:

- Залежність якості роботи системи від надійності локальної Wi-Fi або мобільної мережі закладу.

– Необхідність суворих заходів інформаційної безпеки: захист платіжних та персональних даних клієнтів, запобігання несанкціонованому втручанню у процес замовлення, коректна ізоляція сесій різних клієнтів за одним столом.

– Консерватизм частини аудиторії, що надає перевагу повному циклу традиційного обслуговування; зазначене вимагає збереження можливості прийому замовлень через персонал у паралельному режимі.

1.1.5 Огляд наукових досліджень у галузі

Проблематика автоматизації ресторанного бізнесу та кількісного і якісного впливу Self-Service Technologies (SST) на клієнтський досвід широко представлена у сучасній науковій літературі.

У закордонних дослідженнях значна увага приділяється питанню сприйняття безконтактних технологій з боку споживачів. Зокрема, у дослідженні Lee S. K. та Kim Y. H. встановлено, що впровадження цифрових кіосків самообслуговування та електронних меню на базі QR-кодів підвищує сприйняту клієнтом безпеку та гігієнічний стандарт закладу – особливо в постпандемічних умовах – а також скорочує суб'єктивний час очікування та позитивно впливає на стійкість ресторанного бізнесу в умовах криз [4]. Зростання попиту на цифрові послуги підтверджується і галузевою статистикою: за даними NRA, більшість відвідувачів надають перевагу закладам із можливістю замовлення онлайн [1]. Можливість візуального перегляду страв у цифровій формі у поєднанні з рекомендаційними модулями здатна підвищити середню вартість замовлення завдяки ефекту стимулювання імпульсивних покупок.

Вітчизняні науковці досліджують процес цифровізації ресторанного сектору в контексті сучасних економічних реалій. Русавська В. А. та Кожухівський С. Ю. констатують, що попри активне поширення інноваційних технологій у сфері HoReCa, значна частина вітчизняних закладів стикається з обмеженнями при впровадженні готових комерційних рішень через їхню невідповідність специфіці

малого та середнього бізнесу, зокрема надлишковість функціонального набору та відсутність адаптації до умов місцевого платіжного середовища [2].

В академічному середовищі наголошується на доцільності переходу до архітектурних підходів, керованих подіями (Event-Driven Architecture), що дозволяє забезпечити паралельну та миттєву синхронізацію стану між різними учасниками системи без блокування обчислювальних ресурсів. Дослідження Jose N. демонструє ефективність зазначеного підходу для забезпечення синхронізації даних у реальному часі у сферах, де критично важлива узгодженість стану між багатьма вузлами системи [6]. Актуальним напрямом залишається також питання безпечної ідентифікації неавторизованих користувачів у розподіленому вебсередовищі з метою збереження стану їхнього замовлення та можливості здійснення оплати.

Узагальнення результатів аналізу предметної галузі засвідчує, що розробка комплексної безконтактної платформи для закладів громадського харчування є актуальною науково-практичною задачею. Її вирішення потребує сучасної архітектури, здатної задовольнити підвищені вимоги щодо продуктивності, синхронізації даних у реальному часі та забезпечення позитивного користувацького досвіду для всіх залучених учасників системи. Зазначені висновки визначають вектор проєктних рішень, що розглядаються у наступних розділах роботи.

1.2 Аналіз існуючих програмних аналогів

Програмні засоби для автоматизації процесів у закладах громадського харчування є невід'ємним елементом сучасного ресторанного бізнесу. За функціональним призначенням та архітектурним підходом зазначені рішення поділяються на чотири основні категорії:

- хмарні системи обліку (POS-системи) – призначені для автоматизації касових операцій, складського обліку та формування звітності;
- спеціалізовані сервіси QR-меню та безконтактної оплати – орієнтовані на цифровізацію клієнтського досвіду;
- комплексні ERP-системи – призначені для управління великими мережами закладів із централізованим контролем показників;
- модульні рішення – поєднують POS-систему зі спеціалізованими мобільними інтерфейсами для клієнтів і персоналу.

Кожен із наведених підходів має різну архітектуру та призначений для закриття специфічних бізнес-потреб. Вибір конкретного рішення залежить від масштабу підприємства, бюджету на впровадження та необхідного рівня взаємодії з відвідувачами. У подальшому проведено детальний аналіз чотирьох найбільш поширених на ринку систем: Poster POS, ChoiceQR, Toast POS та Lightspeed Restaurant.

1.2.1 Poster POS

Poster POS [7] – популярна українська хмарна система автоматизації кафе і ресторанів, що функціонує за моделлю програмного забезпечення як послуги (SaaS). Система об'єднує в собі касовий термінал, складський облік, фінансовий модуль, аналітику та систему управління відносинами з клієнтами (CRM). Концептуальною ідеєю Poster є забезпечення можливості розгортання системи без придбання дороговартісного серверного обладнання: для запуску касового терміналу достатньо стандартного планшетного пристрою на базі iOS або Android з підключенням до мережі Інтернет.

Переваги системи Poster POS:

- інтуїтивно зрозумілий інтерфейс касового терміналу, що скорочує час навчання нових співробітників;

- низький поріг входу – відсутня потреба у власній серверній інфраструктурі;
- наявність вбудованого модуля для прийому замовлень від персоналу залу з планшетів;
- можливість генерації QR-кодів для ознайомлення клієнтів із меню у цифровому форматі;
- широкий набір звітів для керівника закладу через веб-панель управління.

Недоліки системи Poster POS:

- базовий функціонал QR-меню обмежується переважно переглядом страв; для повноцінного замовлення та оплати безпосередньо зі смартфона клієнта необхідна інтеграція зі сторонніми сервісами;
- при нестабільному підключенні до мережі Інтернет синхронізація замовлень між касою та кухнею може відбуватися із затримкою;
- система може стикатися з обмеженнями продуктивності при управлінні великими мережами ресторанів зі складною ієрархічною структурою.

Приклади екранних форм системи Poster POS (інтерфейс касового термінала та панель управління) наведено на рисунку 1.1.

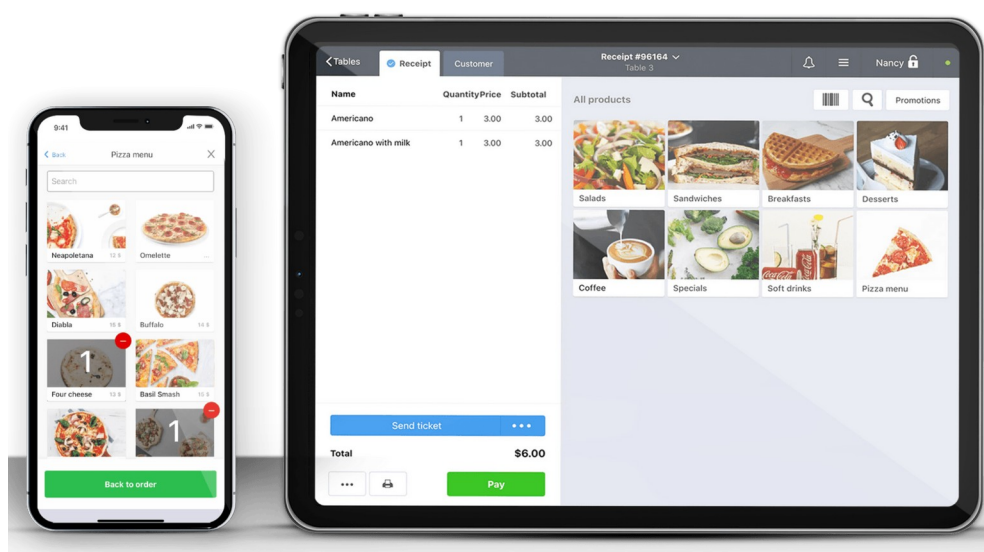


Рис. 1.1 Екранні форми Poster POS

1.2.2 ChoiceQR

ChoiceQR [8] – українська платформа, спеціально розроблена для цифровізації меню та автоматизації безконтактного замовлення. На відміну від класичних POS-систем, ChoiceQR зосереджується на клієнтському інтерфейсі. Після сканування QR-коду на столі відвідувач отримує доступ до вебдодатку без необхідності встановлення нативного застосунку, де може переглянути фотографії страв, сформувати замовлення, здійснити оплату через сучасні мобільні платіжні сервіси, а також залишити чайові.

Ключові функції ChoiceQR:

- глибока інтеграція маркетингових інструментів – рекомендації схожих страв, що безпосередньо впливає на збільшення середнього чека;
- підтримка мультимовності меню та адаптивний дизайн під будь-які мобільні пристрої;
- повний цикл «безконтактного столу»: від вибору страв до миттєвої оплати без участі офіціанта;
- підтримка режимів доставки та самовивозу в рамках єдиного веб-додатку.

Недоліки ChoiceQR:

- платформа не є повноцінною системою обліку – відсутні власний складський та фінансовий модулі у класичному розумінні ERP;
- для забезпечення коректної роботи ресторану ChoiceQR потребує інтеграції з існуючими POS-системами через API, що ускладнює початкове налаштування;
- відсутність вбудованого терміналу персоналу – система орієнтована виключно на клієнтську сторону взаємодії.

Екранні форми сервісу ChoiceQR наведено на рисунку 1.2.

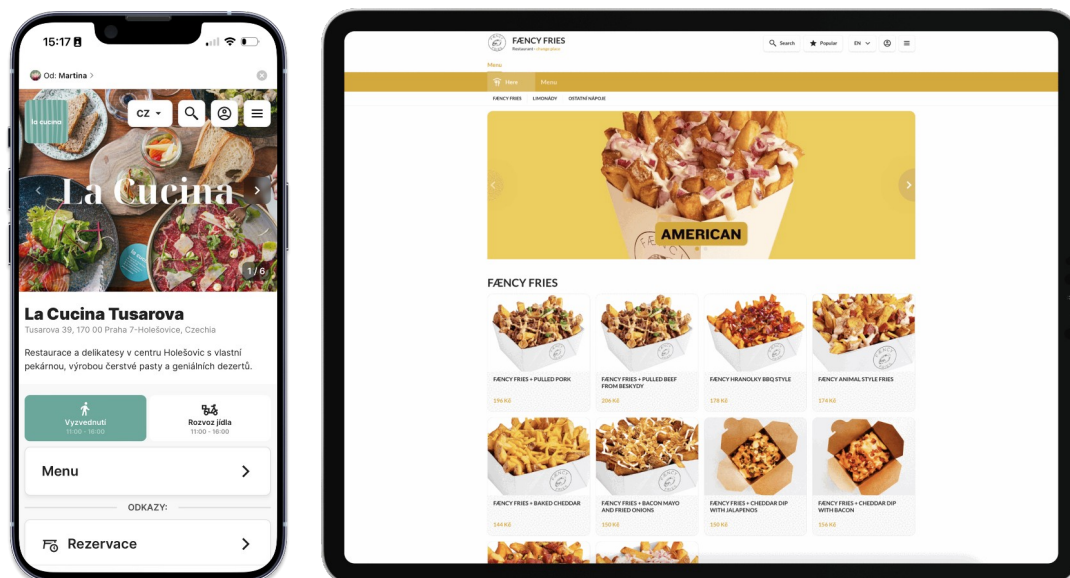


Рис. 1.2 Екранні форми ChoiceQR

1.2.3 Toast POS

Toast POS [9] – американська хмарна платформа автоматизації ресторанів, розроблена компанією Toast, Inc. (Бостон, США). Система позиціонується як одне з найбільш повнофункціональних комплексних рішень на ринку і охоплює весь ланцюг управління закладом: від прийому замовлень та кухонних екранів до детального складського обліку на рівні інгредієнтів та фінансової аналітики для мережевих закладів.

Переваги Toast POS:

- нативна інтеграція між касою та кухонним екраном (KDS) без залучення стороннього програмного забезпечення;
- вбудована підтримка онлайн-замовлень, доставки та QR-оплати рахунку;
- масштабованість: система підходить як для поодиноких закладів, так і для великих мереж із централізованим управлінням меню та звітністю;
- спеціалізоване апаратне забезпечення (термінали, кухонні принтери, сканери), розроблене з урахуванням умов ресторанної кухні.

Недоліки Toast POS:

- компанія налаштована на використання власного платіжного процесора, що обмежує гнучкість у виборі платіжного шлюзу;
- прив'язка до фірмового апаратного забезпечення суттєво підвищує початкові капітальні витрати на розгортання системи;
- обмежена доступність поза ринком США та Ірландії – повний функціонал не підтримується для українського ринку.

Екранні форми та пристрої системи Toast POS наведено на рисунку 1.3



Рис. 1.3 Екранні форми та пристрої Toast POS

1.2.4 Lightspeed Restaurant

Lightspeed Restaurant [10] – канадська хмарна POS-платформа, розроблена компанією Lightspeed Commerce (Монреаль, Канада). Система активно використовується у понад 100 країнах світу та відзначається потужними можливостями управління складом і бізнес-аналітики.

Переваги Lightspeed Restaurant:

- детальний облік інгредієнтів на рівні страв: автоматичне списання сировини зі складу в момент оформлення замовлення;

- потужна аналітична панель з відстеженням ключових показників ефективності (KPI) – популярних страв, часу пікового навантаження, виручки за зонами залу;
- підтримка платежів через Apple Pay, Google Pay та банківські картки безпосередньо у системі;
- широкий ринок інтеграцій з CRM-системами, платформами лояльності та сервісами доставки.

Недоліки Lightspeed Restaurant:

- складніше початкове налаштування порівняно з більш доступними системами;
- вартість підписки на розширені тарифні плани є відносно високою для суб'єктів малого бізнесу;
- функціонал самообслуговування для клієнтів (QR-замовлення зі смартфона) потребує підключення сторонніх сервісів.

Екранні форми системи Lightspeed Restaurant наведено на рисунку 1.4.

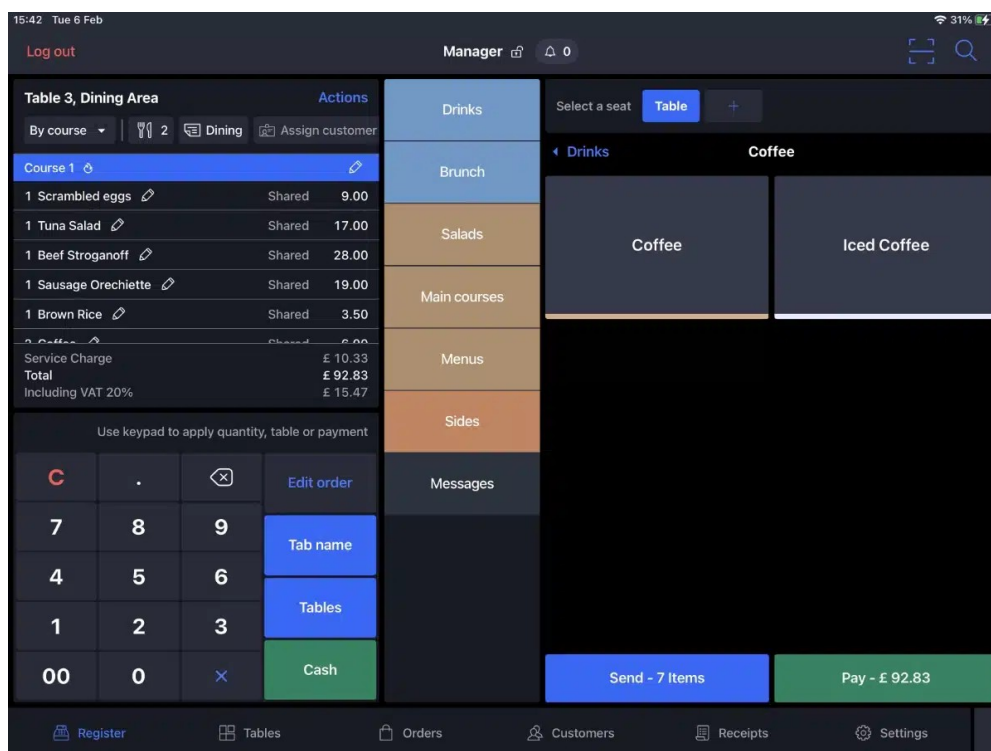


Рис. 1.4 Екранні форми Lightspeed Restaurant

1.2.5 Зведений порівняльний аналіз

За результатами проведеного аналізу складено порівняльну таблицю характеристик розглянутих програмних засобів.

Таблиця 1.1

Зведені результати аналізу характеристик ПЗ для автоматизації ресторанів

Показник	Poster POS	ChoiceQR	Toast POS	Lightspeed Restaurant
Цільовий сегмент	Малий та середній бізнес	Усі типи закладів	Середні та великі мережі	Середні та великі заклади
Архітектура	Хмарна (SaaS)	Хмарна (SaaS)	SaaS + власне обладнання	Хмарна (SaaS)
Складність розгортання	Легка	Легка	Середня	Складна
Наявність POS-терміналу	Наявний	Відсутній (працює як надбудова)	Наявний (фірмове обладнання)	Наявний
Безконтактне замовлення з телефону	Частково (через інтеграції)	Повністю реалізовано	Через вбудований модуль	Через сторонні сервіси
Онлайн-оплата та чайові	Потребує налаштування	Базовий функціонал	Реалізовано (власний процесор)	Реалізовано
Можливості аналітики та складу	Середні	Обмежені (потрібна інтеграція з POS)	Глибокі (мережеві звіти)	Дуже глибокі (KPI, склад)
Вартість впровадження	Низька	Середня/низька	Висока	Висока

Проведений аналіз засвідчує, що на сучасному ринку присутні як потужні, але складні монолітні системи, так і вузькоспеціалізовані рішення, орієнтовані на конкретний аспект обслуговування. Жодна з розглянутих платформ у повній мірі не задовольняє сукупність критеріїв, актуальних для малого та середнього ресторанного бізнесу в умовах українського ринку.

Виявлені системні прогалини охоплюють наступні аспекти:

1. Відсутність наскрізного безконтактного циклу без сторонніх інтеграцій: Poster POS та Lightspeed Restaurant реалізують QR-замовлення лише через підключення зовнішніх сервісів, що ускладнює архітектуру та збільшує сукупну вартість рішення. ChoiceQR, у свою чергу, не має власного модуля обліку та керування персоналом.

2. Залежність від фірмового обладнання: Toast POS вимагає придбання спеціалізованих апаратних засобів виробника, що підвищує поріг входу для малого бізнесу та унеможливлює гнучке масштабування.

3. Обмежена адаптованість до умов українського ринку: американські та канадські системи (Toast POS, Lightspeed) не забезпечують у повному обсязі інтеграцію з вітчизняними платіжними шлюзами.

4. Відсутність персоналізованого клієнтського досвіду: жодна з розглянутих систем не надає вбудованого механізму персоналізації меню на основі харчових алергенів та рекомендаційного модуля на рівні профілю авторизованого клієнта.

5. Надлишкова вартість для середнього бізнесу: системи з розгорнутим функціоналом (Toast POS, Lightspeed) пропонують підписки з надлишковим набором модулів, які важко адаптувати під специфічні потреби конкретного закладу.

Зазначені обмеження обґрунтовують доцільність розробки власного рішення – комплексної вебплатформи безконтактного обслуговування, адаптованої до потреб вітчизняного ринку та позбавленої описаних недоліків.

Формалізація функціональних та нефункціональних вимог до такого рішення є предметом розгляду наступного підрозділу.

1.3 Формулювання вимог до вебзастосунку

Формулювання вимог до розроблюваної системи здійснюється на основі результатів аналізу предметної галузі (див. підрозділ 1.1) та виявлених системних прогалин існуючих програмних аналогів (див. підрозділ 1.2). Відповідно до стандарту IEEE 830, вимоги поділяються на дві категорії: функціональні (що описують поведінку системи та її взаємодію з користувачами) та нефункціональні (що встановлюють якісні характеристики та обмеження системи) [11].

1.3.1 Функціональні вимоги

– FR-01 (Адміністратор, Гість) – Система генерує унікальний QR-код для кожного столу. Сканування відкриває вебінтерфейс меню у браузері без встановлення мобільного додатку.

– FR-02 (Гість, Клієнт) – Система відображає цифрове меню з категоріями, фотографіями, описом, складом та ціною. Меню оновлюється у реальному часі при зміні адміністратором.

– FR-03 (Гість) – Система надає гостьовий режим без реєстрації. Гість ідентифікується через тимчасовий сесійний ідентифікатор, що зберігає стан кошика без облікового запису. При подальшій авторизації гостя забезпечується безшовне перенесення кошика та активних замовлень до його облікового запису.

– FR-04 (Гість, Клієнт) – Система забезпечує повний цикл оформлення замовлення: додавання страв, зміна кількості позицій, підтвердження із зазначенням номера столу.

– FR-05 (Гість, Клієнт) – Система відображає актуальний статус замовлення у реальному часі (Нове → Готується → Готово → Завершено) без необхідності вручну оновлювати сторінку.

– FR-06 (Персонал) – Система надає кухарям термінал для відображення черги замовлень та зміни їх станів. При переведенні у «Готово» клієнт та офіціант отримують сповіщення.

– FR-07 (Гість, Клієнт) – Система забезпечує онлайн-оплату через платіжний шлюз LiqPay. Статус оплати оновлюється автоматично після підтвердження транзакції.

– FR-08 (Клієнт) – Система дозволяє фільтрування меню за алергенами на основі профілю клієнта. Страви з небажаними алергенами позначаються або приховуються.

– FR-09 (Клієнт) – Система реалізує програму лояльності: нарахування балів після кожного оплаченого замовлення та їх списання як знижки на наступне.

– FR-10 (Адміністратор) – Система надає вебпанель управління: додавання/редагування/видалення страв і категорій, керування доступністю позицій, перегляд аналітики продажів.

1.3.2 Нефункціональні вимоги

– NFR-01 (Продуктивність) – Час відповіді на типові запити (перегляд меню, оновлення статусу) не перевищує 200 мс при нормальному навантаженні.

– NFR-02 (Надійність) – Коректна обробка паралельних транзакцій без конфліктів паралельного доступу при одночасному оформленні замовлень з кількох столів; операції збереження відповідають вимогам ACID.

– NFR-03 (Безпека) – Розмежування доступу за рольовою моделлю з чотирма рівнями: гість, клієнт, персонал, адміністратор. Кожна точка доступу захищається відповідно до мінімально необхідних привілеїв.

– NFR-04 (Безпека) – Платіжні запити підписуються криптографічним підписом (HMAC-SHA1) для запобігання підміні параметрів. Система захищена від CSRF-атак.

– NFR-05 (Доступність) – Клієнтський інтерфейс коректно відображається на мобільних пристроях від 320 px та на десктопних платформах [5].

– NFR-06 (Зручність) – Інтерфейс відповідає WCAG 2.1 рівня AA: коректні контрасти, навігація з клавіатури, альтернативні тексти для зображень [5].

– NFR-07 (Масштабованість) – Підтримка не менше 300 одночасних активних сесій замовлень без деградації часу відповіді; архітектура допускає горизонтальне масштабування.

– NFR-08 (Супроводжуваність) – Схема БД версіонується інструментом міграцій для відтворення стану на будь-якому етапі розробки без ручного втручання.

– NFR-09 (Апаратна незалежність) – Система не вимагає спеціалізованого обладнання; всі компоненти (термінал, планшет персоналу, кухонний екран) функціонують у стандартному браузері.

– NFR-10 (Розгортання) – Система контейнеризована та підтримує розгортання через Docker Compose. Секретні параметри передаються через змінні оточення без збереження у кодовій базі.

Сформульована сукупність функціональних та нефункціональних вимог відображає повний спектр потреб усіх залучених учасників системи: гостей ресторану, авторизованих клієнтів, персоналу та адміністраторів. Аналіз вимог дозволяє виокремити ключові архітектурні виклики, що визначають підхід до проєктування:

– необхідність оновлення стану в реальному часі (FR-05, FR-06) вимагає застосування технологій постійного з'єднання між клієнтом та сервером без механізму циклічного опитування;

- вимога нульового бар'єру входу для гостей (FR-03) визначає архітектуру ідентифікації на основі сесійних ідентифікаторів без обов'язкової автентифікації;
- вимоги щодо ACID-відповідності транзакцій (NFR-02) та рольової моделі доступу (NFR-03) обумовлюють вибір реляційної СУБД та централізованого постачальника ідентичності;
- вимога апаратної незалежності (NFR-09) разом із вимогою адаптивного інтерфейсу (NFR-05) визначають вебзастосунок як єдину платформу для всіх ролей без нативних мобільних клієнтів.

Зазначені архітектурні виклики та обмеження є вхідними даними для проектування системи, що розглядається у розділі 2.

1.4 Засоби реалізації застосунку

Вибір програмних засобів та технологічного стеку для реалізації системи SmartDine здійснювався відповідно до сформульованих функціональних і нефункціональних вимог (підрозділ 1.3), а також з урахуванням зрілості технологій, активності їх підтримки спільнотою та наявності якісної документації.

1.4.1. Backend-технології

Основною мовою програмування серверної частини обрано Java (версія 25, LTS). Суворий статичний контроль типів запобігає цілому класу помилок на етапі компіляції, що є особливо важливим при роботі з платіжними даними та станами замовлень. Починаючи з Java 21, платформа підтримує віртуальні потоки (Project Loom), що дозволяє ефективно обробляти тисячі одночасних SSE-з'єднань без блокування системних потоків.

Spring Boot є де-факто індустріальним стандартом для розробки вебсервісів мовою Java. Фреймворк реалізує принцип конвенції замість конфігурації та забезпечує автоматичне налаштування компонентів: маршрутизації (Spring MVC),

безпеки (Spring Security), рівня об'єктно-реляційного відображення (ORM через Hibernate) та кешування даних [12].

1.4.2. Засоби керування базами даних

PostgreSQL обрано як основну СКБД завдяки повній підтримці ACID-транзакцій, механізму блокувань на рівні рядків для захисту від конфліктів паралельного доступу та надійній роботі під високим навантаженням.

Flyway забезпечує версіонування схеми бази даних через SQL-скрипти, що виконуються у суворо визначеному порядку при запуску застосунку, гарантуючи ідентичність схеми у всіх середовищах – розробки, тестування та виробництва (NFR-08).

1.4.3. Frontend-технології

Thymeleaf є серверним шаблонізатором, що реалізує концепцію інтерфейсу, керованого сервером (Server-Driven UI): готовий HTML формується на сервері та передається клієнту вже з вставленими даними, що прискорює початкове завантаження сторінки та виключає потребу у великих пакунків JavaScript. Thymeleaf інтегрований із засобами безпеки, що дозволяє умовно відображати елементи інтерфейсу залежно від ролі користувача безпосередньо у шаблоні.

HTMX є легковаговою JavaScript-бібліотекою, що надає можливості динамічного оновлення об'єктної моделі документа через спеціальні атрибути без написання власного коду. Сервер повертає готові фрагменти інтерфейсу замість даних у форматі JSON, які бібліотека вбудовує у відповідне місце сторінки без перезавантаження [13].

Server-Sent Events (SSE) – браузерна технологія для отримання потокових подій від сервера через стійке однонаправлене HTTP-з'єднання. Використовується для миттєвого інформування кухаря про нові замовлення та сповіщення клієнта про зміну статусу страви [14].

1.4.4. Засоби автентифікації та зовнішні сервіси

Keycloak (Red Hat) – відкрите рішення для управління ідентифікацією та доступом (IAM). Інтеграція відбувається через сучасні протоколи автентифікації: Keycloak видає підписані маркери доступу (токени), які серверна частина верифікує через відкритий ключ. Рольова модель доступу розмежовує права адміністратора, кухаря та офіціанта [15].

LiqPay API (v3) – платіжний шлюз від ПриватБанку для прийому онлайн-оплат із верифікацією підпису зворотного виклику, що захищає від підробки платіжного статусу (NFR-04).

Web Push забезпечує сповіщення персоналу про нові замовлення навіть при закритій вкладці браузера через зашифровані пакети сповіщень.

1.4.5. Засоби тестування, контейнеризації та розгортання

Для тестування використовується сучасний набір засобів: JUnit 5 (модульні тести), Mockito (мокування залежностей), Playwright (E2E-тестування у реальному браузері Chromium), Testcontainers (інтеграційне тестування з PostgreSQL у Docker-контейнері).

Для контейнеризації та розгортання застосовано Docker (багатоетапна збірка образу) та Docker Compose, що оркеструє основні сервіси застосунку. Nginx виступає зворотним проксі-сервером, забезпечує шифрування трафіку, жорсткі заголовки безпеки та коректну роботу механізму передачі подій у реальному часі.

Додатково використовуються: система контролю версій Git/GitHub, інтегроване середовище розробки IntelliJ IDEA, система збірки Apache Maven, засоби автоматичної кодогенерації Lombok, генерації QR-кодів ZXing та оптимізації зображень страв Thumbnailator/webp-imageio.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА МОДЕЛЮВАННЯ ПРОЦЕСІВ БЕЗКОНТАКТНОГО ОФОРМЛЕННЯ ЗАМОВЛЕНЬ

Перехід від сформульованих вимог до безпосереднього написання коду вимагає етапу ретельного моделювання. Проєктування архітектури є критичним кроком, оскільки закладені на цьому етапі рішення визначають майбутню масштабованість, безпеку та надійність програмного продукту. Для забезпечення цілісності системи необхідно візуалізувати поведінку користувачів, продумати компонентну взаємодію серверної та клієнтської частин, а також розробити оптимальну структуру бази даних. Такий підхід дозволяє виявити потенційні архітектурні конфлікти ще до початку етапу імплементації та гарантує, що застосунок задовольнятиме всі заявлені бізнес-вимоги.

2.1 Функціональне моделювання (Use Case)

Відповідно до сформульованих у підрозділі 1.3 функціональних вимог та результатів аналізу організаційної структури закладу, у системі виокремлено чотири актори. Актор у контексті UML – це роль, яку відіграє зовнішня сутність (людина або система) під час взаємодії зі застосунком [16].

Гість (Guest) – неавторизований відвідувач, що отримує доступ до системи виключно через сканування QR-коду столу. Ідентифікується за допомогою тимчасового сесійного ідентифікатора, що генерується автоматично. Має доступ до повного циклу замовлення та оплати без реєстрації облікового запису. Відповідає ролі GUEST у системі розмежування доступу.

Авторизований клієнт (Customer) – зареєстрований відвідувач, автентифікований через централізований постачальник ідентичності. Наслідує всі можливості Гостя та додатково отримує доступ до системи лояльності,

персоналізованого фільтрування меню за алергенами та перегляду історії замовлень. Відповідає ролі CUSTOMER.

Персонал (Staff) – співробітники закладу двох підвидів: офіціант (роль WAITER) та кухар (роль CHEF). Офіціант має доступ до моніторингу замовлень у своїй зоні відповідальності. Кухар, окрім перегляду черги, має право керувати станами замовлень та редагувати меню. Обидві ролі об'єднуються під спільним доступом до захищеного розділу /staff/**.

Адміністратор (Administrator) – управлінський персонал із роллю ADMINISTRATOR. Має повний доступ до всіх захищених розділів системи: адміністративна панель (/admin/**), генерація QR-кодів, управління меню, перегляд аналітики. Наслідує дозволи ролей CHEF та WAITER.

2.1.1 Діаграма варіантів використання

Діаграма варіантів використання є засобом функціонального моделювання відповідно до стандарту UML 2.5 і відображає взаємодію між акторами та функціональними блоками системи [16]. Побудована діаграма для системи SmartDine охоплює 10 прецедентів, об'єднаних у межах системного контуру, та відображає зв'язки типу <<include>> (обов'язкове включення) та <<extend>> (умовне розширення).

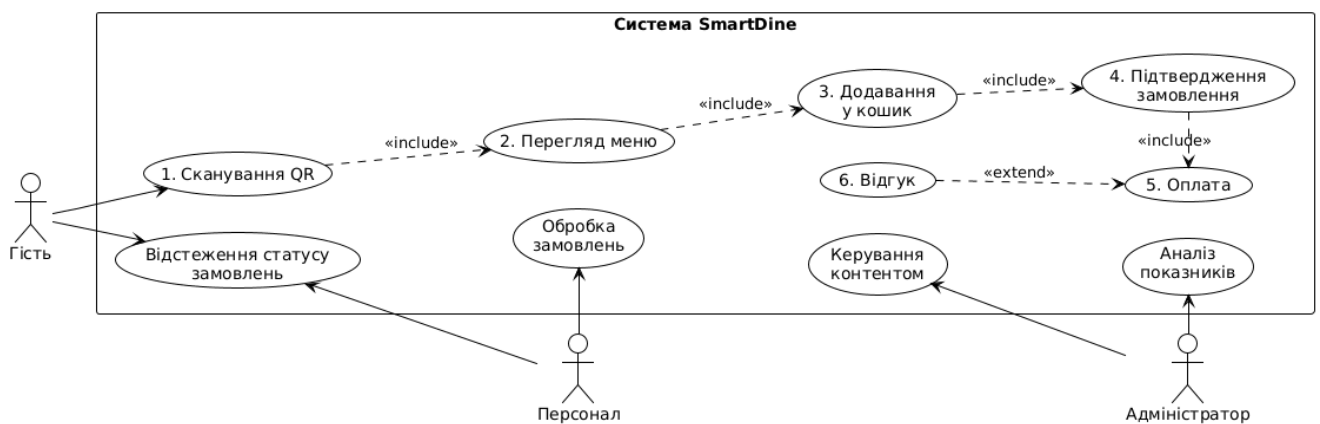


Рис. 2.1 Діаграма варіантів системи SmartDine

Основний потік взаємодії Гостя з системою будується за принципом послідовного ланцюга прецедентів:

- UC-01: Сканування QR-коду ініціює сесію та є відправною точкою для всієї взаємодії клієнта із системою. Містить обов'язковий зв'язок <<include>> до UC-02.

- UC-02: Перегляд меню – перегляд категорій страв із фотографіями, описом та актуальними цінами. Містить обов'язковий зв'язок <<include>> до UC-03.

- UC-03: Додавання страви до кошика – формування переліку обраних позицій. Містить обов'язковий зв'язок <<include>> до UC-04.

- UC-04: Оформлення замовлення – передача кошика до серверної частини, фіксація номера столу та створення запису замовлення. Містить обов'язковий зв'язок <<include>> до UC-05.

- UC-05: Оплата замовлення – перенаправлення клієнта на сторінку платіжного шлюзу. <<include>> UC-05 (оплата можлива лише після оформлення).

- UC-06: Відгук про замовлення – залишення оцінки страв та якості обслуговування після завершення сесії. Зв'язок <<extend>> від UC-05 (умовне розширення: можливість надіслати відгук стає доступною лише після успішного переходу замовлення в статус «Завершено»).

Крім основного ланцюга замовлення, у системі визначено незалежні прецеденти взаємодії:

- UC-07: Відстеження статусу замовлень — перегляд поточного стану замовлення в режимі реального часу. Є незалежною дією, доступною для Гостя та Персоналу у будь-який момент взаємодії із системою, незалежно від поточного кроку основного ланцюга замовлення.

Персонал взаємодіє з двома прецедентами:

- UC-08: Обробка замовлень – перегляд черги вхідних замовлень та послідовна зміна їх станів (Нове → Готується → Готово).

– UC-07: Відстеження статусу – спільний із клієнтом прецедент моніторингу стану замовлень.

Адміністратор взаємодіє з трьома прецедентами адміністративного рівня:

– UC-09: Керування контентом – додавання та редагування страв, категорій меню, генерація QR-кодів для столів.

– UC-10: Аналіз показників – перегляд аналітичної інформаційної панелі з ключовими показниками продажів та оборотності..

2.1.2 Опис логіки розмежування доступу

Розмежування доступу між акторами реалізується за моделлю RBAC (Role-Based Access Control). Кожна точка доступу (HTTP-ендпоінт) системи захищається відповідно до мінімально необхідних привілеїв. Нижче наведено матрицю ключових ресурсів та відповідних ролей.

Таблиця 2.1

Матриця розмежування доступу (RBAC)

Ресурс системи	GUEST	CUSTOMER	WAITER	CHEF	ADMIN
Перегляд меню (/menu/**)	+	+	+	+	+
Кошик та оформлення (/api/cart/**, /api/orders)	+	+	–	–	–
Оплата та виклик офіціанта	+	+	–	–	–
Панель персоналу та зміна станів (/staff/**)	–	–	+	+	+
Редагування меню (/admin/menu)	–	–	–	+	+
Адміністративна панель та QR (/admin/**)	–	–	–	–	+
Аналітика та звітність (/admin/dashboard)	–	–	–	–	+

Ролі 'WAITER', 'CHEF' та 'ADMINISTRATOR' передаються у серверну частину через поле 'realm_access.roles' у складі токена доступу Keycloak. Роль 'GUEST' присвоюється автоматично для неавторизованих запитів через механізм сесійного ідентифікатора.

Додаткова гранулярність захисту на рівні окремих методів сервісного шару забезпечується декларативними анотаціями контролю доступу, що перевіряються при виконанні відповідного методу незалежно від рівня маршрутизації.

Побудована діаграма варіантів використання унаочнює повний спектр функціональних взаємодій системи та підтверджує її відповідність визначеним у підрозділі 1.3 вимогам. Аналіз прецедентів виявляє ключові точки, що визначають архітектурні рішення наступного рівня проєктування:

- наявність спільного прецеденту UC-08 (відстеження статусу) між клієнтом та персоналом вимагає архітектурного механізму синхронізації стану між незалежними браузерними сесіями в реальному часі;
- підтримка гостьового режиму (UC-01...UC-05 без автентифікації) потребує окремого механізму ідентифікації сесії, незалежного від постачальника ідентичності, із можливістю безшовного перенесення кошика при наступній авторизації;
- п'ятиступенева рольова модель (GUEST / CUSTOMER / WAITER / CHEF / ADMINISTRATOR) визначає архітектуру захисту ендпоінтів на основі централізованого постачальника ідентичності.

Зазначені висновки є вхідними даними для архітектурного моделювання, що розглядається у підрозділі 2.2.

2.2 Архітектурне моделювання

Архітектура розроблюваного застосунку побудована за трирівневим принципом розподілу відповідальностей (Separation of Concerns), що є усталеним

підходом для вебзастосунків корпоративного класу [17]. Три підсистеми утворюють ієрархічний стек:

- Client Layer (Рівень клієнта) – браузерна частина, що взаємодіє з серверною частиною через HTTP-запити та підтримує постійне з'єднання для отримання оновлень у реальному часі.
- Application Layer (Рівень застосунку) – серверна частина, що реалізує бізнес-логіку, маршрутизацію, автентифікацію та формування відповідей.
- Infrastructure Layer (Інфраструктурний рівень) – зовнішні сервіси: постачальник ідентичності, платіжний шлюз та реляційна СУБД.

Такий поділ забезпечує незалежність кожного рівня від деталей реалізації суміжних, що спрощує тестування, супровід та масштабування системи.

2.2.1 Діаграма компонентів

Діаграма компонентів згідно з UML 2.5 відображає структурну організацію системи та протоколи взаємодії між її складовими [16]. Побудована діаграма для системи SmartDine охоплює три підсистеми з відповідними компонентами.

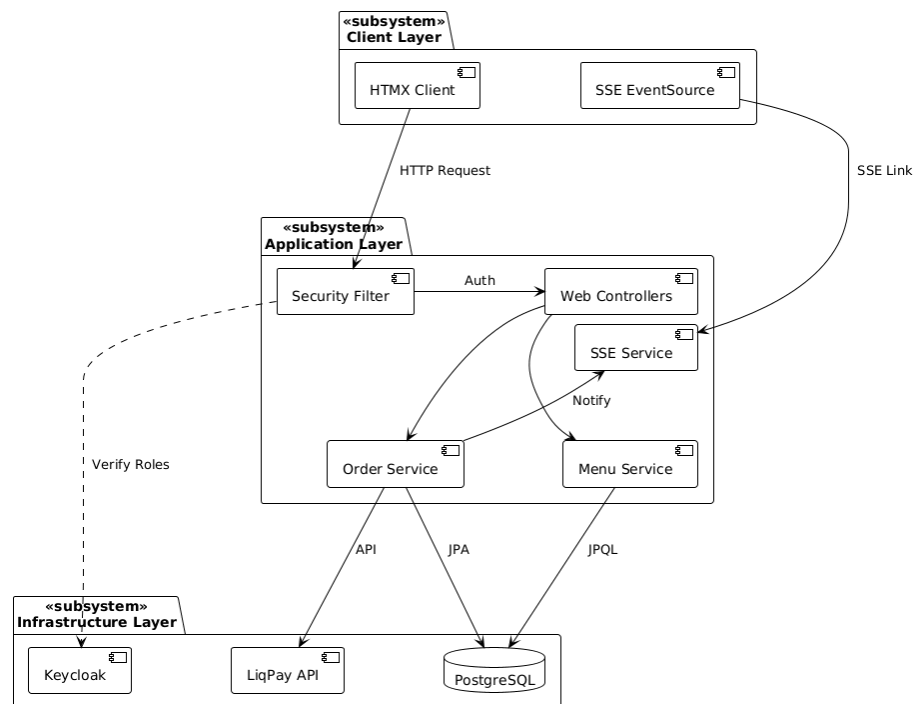


Рис. 2.2 Діаграма компонентів системи

Client Layer містить два компоненти:

- HTMX Client – браузерний клієнт, що надсилає декларативні HTTP-запити до серверної частини та замінює лише відповідні фрагменти об'єктної моделі документа (DOM) без повного перезавантаження сторінки. Взаємодіє з Application Layer через стандартний HTTP-протокол.

- SSE EventSource – браузерний компонент, що встановлює та підтримує постійне однонаправлене з'єднання (SSE Link) з сервером. Отримує потокові події при зміні стану замовлення та автоматично оновлює відповідні блоки сторінки.

Application Layer містить чотири компоненти:

- Security Filter – перший бар'єр обробки вхідних HTTP-запитів. Перехоплює кожен запит, перевіряє наявність та валідність токена автентифікації, після чого звертається до інфраструктурного рівня (Keycloak) для верифікації ролей (Verify Roles). За результатом перевірки формує контекст безпеки та передає запит далі або повертає відповідь з кодом 401/403.

- Web Controllers – рівень маршрутизації, що отримує верифіковані запити від Security Filter (Auth) та делегує їх до відповідних сервісів бізнес-логіки. Формує відповіді у вигляді HTML-фрагментів (для HTMX-клієнта) або JSON (для API-клієнтів). Також підтримує SSE-з'єднання, делегуючи їх до SSE Service.

- Order Service – сервіс бізнес-логіки замовлень. Обробляє операції створення, модифікації та зміни станів замовлень. Після зміни стану замовлення публікує подію до SSE Service (Notify), що ініціює розсилку оновлень усім підписаним клієнтам. Взаємодіє з PostgreSQL через JPA для збереження та зчитування даних.

- Menu Service – сервіс управління меню та стравами. Отримує дані через JPQL-запити безпосередньо до PostgreSQL. Підтримує кешування на рівні застосунку для зменшення навантаження на СУБД при частих запитах до незмінних даних (перегляд меню клієнтами).

- SSE Service – компонент управління SSE-з'єднаннями. Зберігає реєстр активних підписок і при отриманні події від Order Service розсилає її відповідним передплатникам (клієнту конкретного столу та черговому персоналу).

Infrastructure Layer містить три компоненти:

- Keycloak – централізований постачальник ідентичності (Identity Provider). Зберігає облікові дані користувачів, управляє ролями та видає підписані JWT-токени. Security Filter перевіряє цифровий підпис токена за допомогою відкритих ключів Keycloak (JWK Set URI), не розкриваючи облікові дані.

- LiqPay API – зовнішній платіжний шлюз. Сервіс замовлень формує підписаний запит на ініціювання оплати та відправляє його до інтерфейсу шлюзу. Після завершення транзакції LiqPay надсилає підписаний запит зворотного виклику, що обробляється сервісом замовлень та оновлює статус оплати.

- PostgreSQL – реляційна СУБД. Отримує запити від Order Service через JPA (Jakarta Persistence API) та від Menu Service через JPQL-запити. Є єдиним сховищем усіх персистентних даних застосунку.

2.2.2 Діаграма класів

Для безпосередньої реалізації бізнес-логіки бекенду на рівні Application Layer застосовано патерн Service Layer. Внутрішню структуру цього рівня відображає діаграма класів (див. рис. 2.3). Архітектура побудована з використанням інтерфейсів (принцип інверсії залежностей), що забезпечує слабку зв'язність між компонентами (наприклад, сервісами кошика, обслуговування залу та платіжними модулями) та дозволяє легко покривати систему ізольованими модульними тестами.

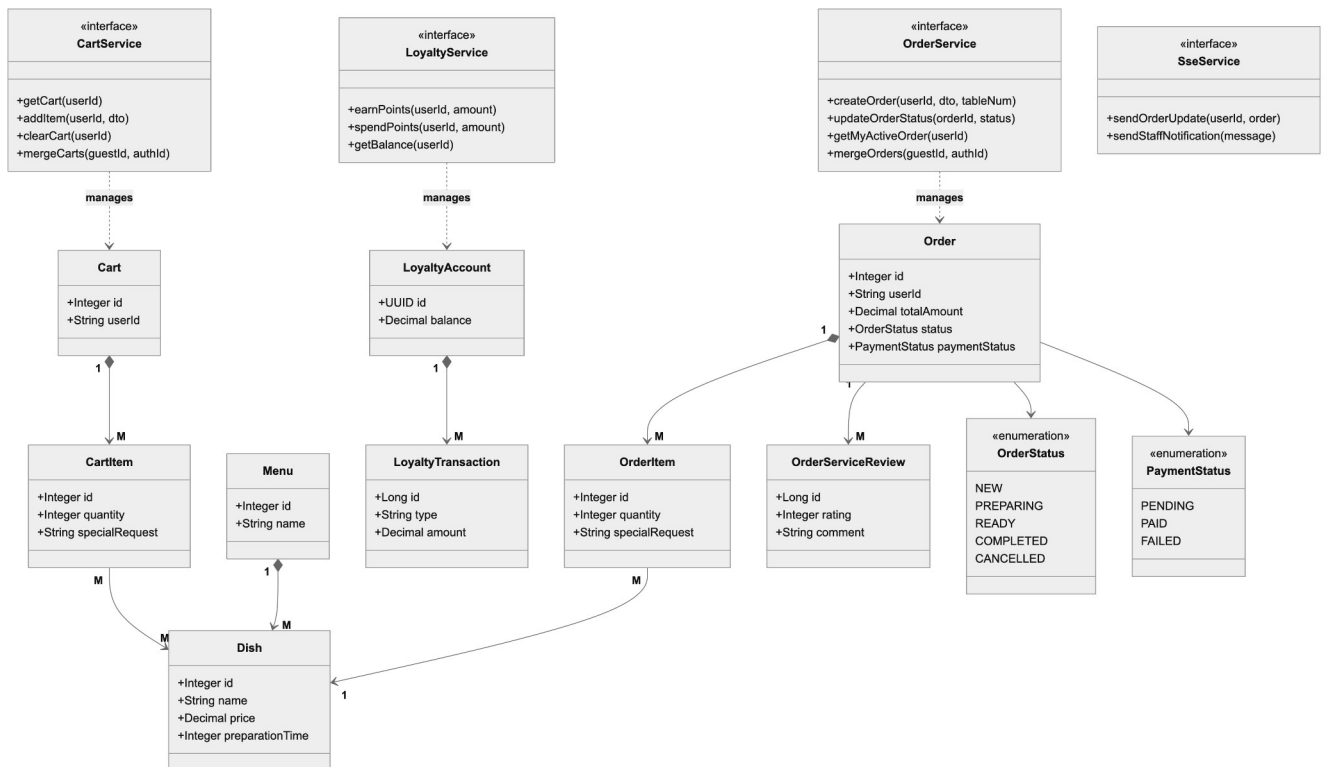


Рис. 2.3 Діаграма класів бізнес-логіки

2.2.3 Обґрунтування архітектурних рішень та компромісів

У процесі проектування системи було прийнято низку критичних інженерних рішень, що базуються на компромісах між продуктивністю, складністю та вимогами. Зважаючи на відсутність потреби у незалежному масштабуванні окремих підсистем ресторану на даному етапі, було обрано монолітну архітектуру (Single-Tenant Deployment), що забезпечило простоту транзакцій без розподілених ACID-проблем та швидкість розгортання без накладних витрат на оркестрацію мікросервісів [17]. Оскільки потік подій системи є односпрямованим від сервера до клієнтів, механізм Server-Sent Events (SSE) став оптимальним вибором замість дуплексного WebSocket, адже він є значно легшим та має вбудовану підтримку автоматичного перепідключення.

Крім того, використання Server-Side Rendering у комбінації з HTMX дозволило відмовитися від складних SPA-фреймворків, уникнувши проблем із клієнтською маршрутизацією, що суттєво зменшило розмір пакунку JavaScript та

гарантувало захист бізнес-логіки виключно на боці сервера. Як основне сховище даних було обрано реляційну СУБД PostgreSQL, оскільки суворі вимоги до цілісності фінансових транзакцій унеможливили використання NoSQL-сховищ, а вбудовані механізми конкурентного доступу (MVCC) вирішили проблеми блокувань при високому навантаженні [18].

2.3 Проєктування бази даних

Схема бази даних налічує 15 таблиць, згрупованих у чотири функціональні кластери:

- Каталог меню: `menu`, `dish`, `dish_menus`, `dish_tags`, `dish_allergens` – зберігають структуру меню, страви з атрибутами (ціна, час приготування, алергени, теги) та зв'язок N:M між стравами і меню.

- Управління замовленнями: `carts`, `cart_item`, `orders`, `order_item` – реалізують повний цикл від формування кошика до фіксації оплаченого замовлення зі статусами, номером столу та часом готовності.

- Система лояльності: `loyalty_account`, `loyalty_transaction`, `user_allergen_exclusions` – зберігають баланс бонусних балів, журнал операцій EARN/REDEEM з механізмом ідемпотентності та персональні виключення алергенів клієнта.

- Відгуки та сповіщення: `order_service_review`, `order_dish_review`, `push_subscriptions` – забезпечують збір оцінок обслуговування та страв, а також зберігають ключі Web Push підписок персоналу.

Повна схема з атрибутами, первинними та зовнішніми ключами наведена на ER-діаграмі (див. рис. 2.4).

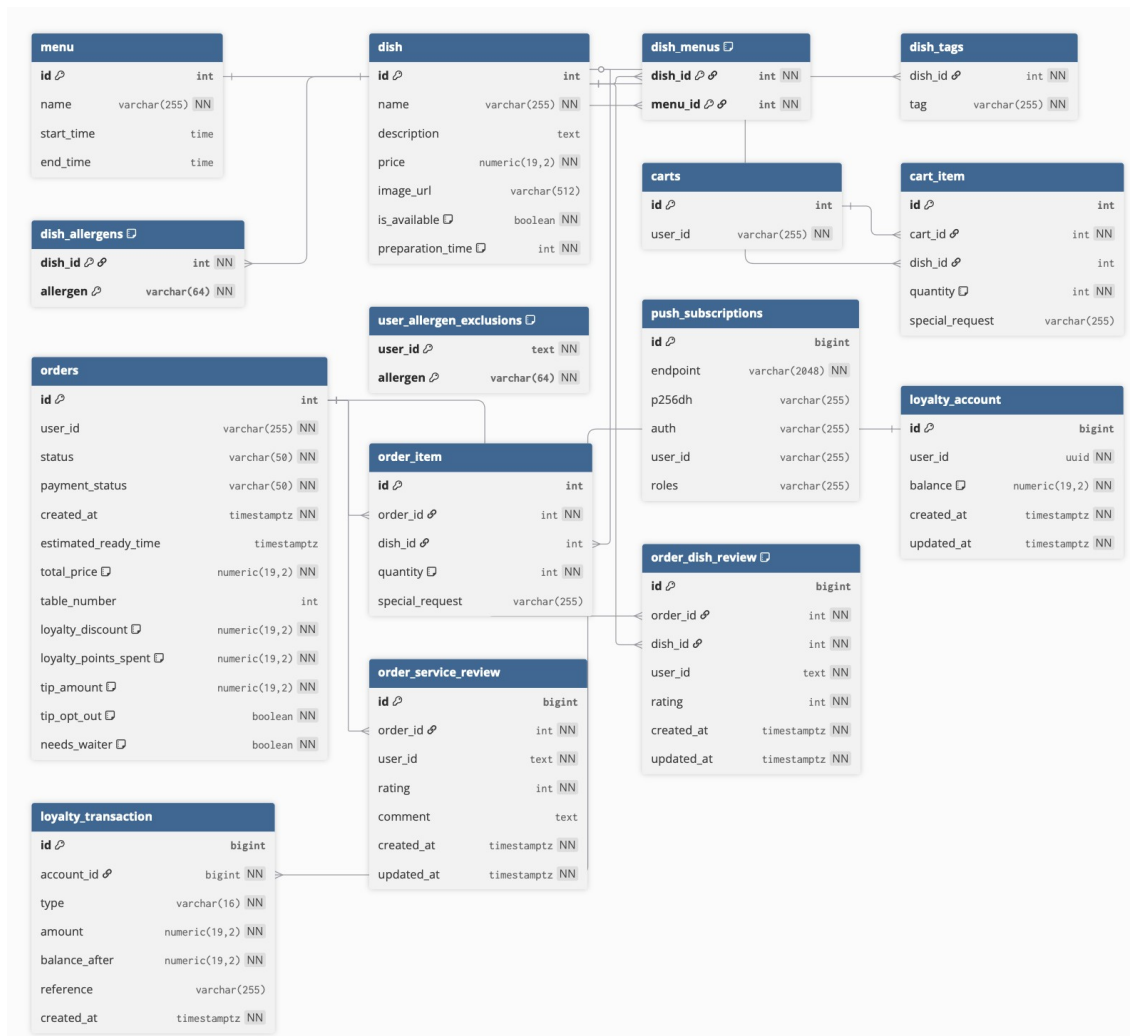


Рис. 2.4 ER-діаграма бази даних системи SmartDine

2.3.1 Типи зв'язків між таблицями

Цілісність даних та логічна узгодженість моделі SmartDine забезпечується через систему зовнішніх ключів та обмежень, що визначають характер взаємодії між таблицями. Нижче наведено основні типи зв'язків, які формують структуру бази даних:

- Багато-до-багатьох (N:M): реалізовано між сутностями 'dish' та 'menu' через сполучну таблицю 'dish_menus' зі складеним первинним ключем та каскадним видаленням записів.

- Один-до-багатьох (1:N): ієрархічні зв'язки між основними сутностями та їх складовими, зокрема: 'carts' → 'cart_item'; 'orders' → 'order_item'; 'loyalty_account' → 'loyalty_transaction'.

- Один-до-одного (1:1): зв'язок між 'orders' та 'order_service_review' (через унікальне обмеження на зовнішній ключ), що гарантує наявність лише одного відгуку про обслуговування для кожного замовлення.

- Колекції елементів: допоміжні таблиці 'dish_tags' та 'dish_allergens', що не мають власного первинного ключа і повністю залежать від життєвого циклу головної сутності страви.

- Незалежні сутності: таблиці 'user_allergen_exclusions' та 'push_subscriptions', які ідентифікуються за рядковим ідентифікатором користувача та не мають жорстких зв'язків на рівні СУБД для забезпечення гнучкості ідентифікації.

2.3.2 Стратегія індексування та ER-діаграма БД

Для забезпечення прийняттого часу відповіді при типових сценаріях навантаження (NFR-01) визначено стратегію індексування, що охоплює найбільш часто запитувані поля таблиць orders, order_item, cart_item та loyalty_transaction.

На таблиці orders створено три B-tree індекси: по полю user_id – для швидкого пошуку активного замовлення конкретного клієнта; складений індекс (status, created_at DESC) – для побудови черги замовлень на панелі персоналу; складений індекс (payment_status, created_at DESC) – для аналітичних запитів адміністратора. На таблицях order_item та cart_item створено індекси по зовнішніх ключах order_id та cart_id відповідно, що прискорює операції JOIN при завантаженні позицій замовлення.

Окремо реалізовано частковий унікальний індекс ux_loyalty_transaction_reference по полю reference WHERE reference IS NOT NULL у таблиці loyalty_transaction. Такий тип індексу гарантує ідемпотентність

нарахування бонусних балів: база даних на рівні обмеження унеможливорює дублювання транзакції з тим самим reference-ключем, навіть при паралельних запитах.

2.4 Моделювання динаміки (діаграми послідовності)

Діаграма послідовності є засобом динамічного моделювання за стандартом UML 2.5 і відображає часовий порядок повідомлень між об'єктами при реалізації конкретного сценарію [16]. На відміну від діаграми компонентів (див. підрозділ 2.2), яка показує структуру системи, діаграма послідовності розкриває поведінку: хто, кому і в якому порядку надсилає повідомлення.

Для системи SmartDine побудовано два ключових сценарії, що охоплюють найкритичніші з точки зору проєктування потоки:

- сценарій А – оформлення замовлення та синхронізація стану через SSE;
- сценарій Б – проведення онлайн-оплати через платіжний шлюз LiqPay із підтвердженням через зворотний виклик.

2.4.1 Сценарій А: Оформлення замовлення та SSE-сповіщення

Сценарій (див. рис. 2.5) охоплює повний цикл від оформлення замовлення клієнтом до отримання персоналом сповіщення у реальному часі. Учасниками взаємодії є: Клієнт (браузер), фільтр безпеки, контролер замовлень, сервіс замовлень, сервіс сповіщень, база даних та Персонал (браузер з активною підпискою на події).

Передумова: клієнт перебуває на сторінці кошика; з'єднання для отримання подій між браузером клієнта та сервером уже встановлено; персонал також підключений до власного каналу сповіщень.

Послідовність дій:

1. Клієнт натискає «Підтвердити замовлення». Браузер надсилає запит на підтвердження через асинхронний механізм бібліотеки взаємодії.
 2. Фільтр безпеки перевіряє токен захисту від підробки запитів та контекст безпеки. Дозвіл надається (точка доступу відкрита для гостей).
 3. Контролер замовлень передає запит до сервісу, що відповідає за підтвердження замовлень із кошика, надаючи ідентифікатор користувача та номер столу з поточної сесії.
 4. Сервіс замовлень формує новий запис замовлення зі статусом «Нове» та зберігає його в базі даних у межах транзакції.
 5. Після успішного збереження сервіс викликає компонент сповіщень для розсилки події оновлення замовлення із відповідними даними.
 6. Сервіс сповіщень розсилає подію всім підключеним пристроям персоналу через відповідні відкриті канали.
 7. Браузер персоналу отримує подію та автоматично оновлює чергу замовлень без перезавантаження сторінки.
 8. Контролер замовлень повертає успішну відповідь з даними замовлення. Клієнтська частина оновлює блок відображення статусу на сторінці клієнта.
- Альтернативний потік (A1): якщо кошик порожній, сервіс генерує виняток про некоректний запит. Контролер повертає помилку, а клієнт бачить спливне повідомлення про неможливість оформлення.

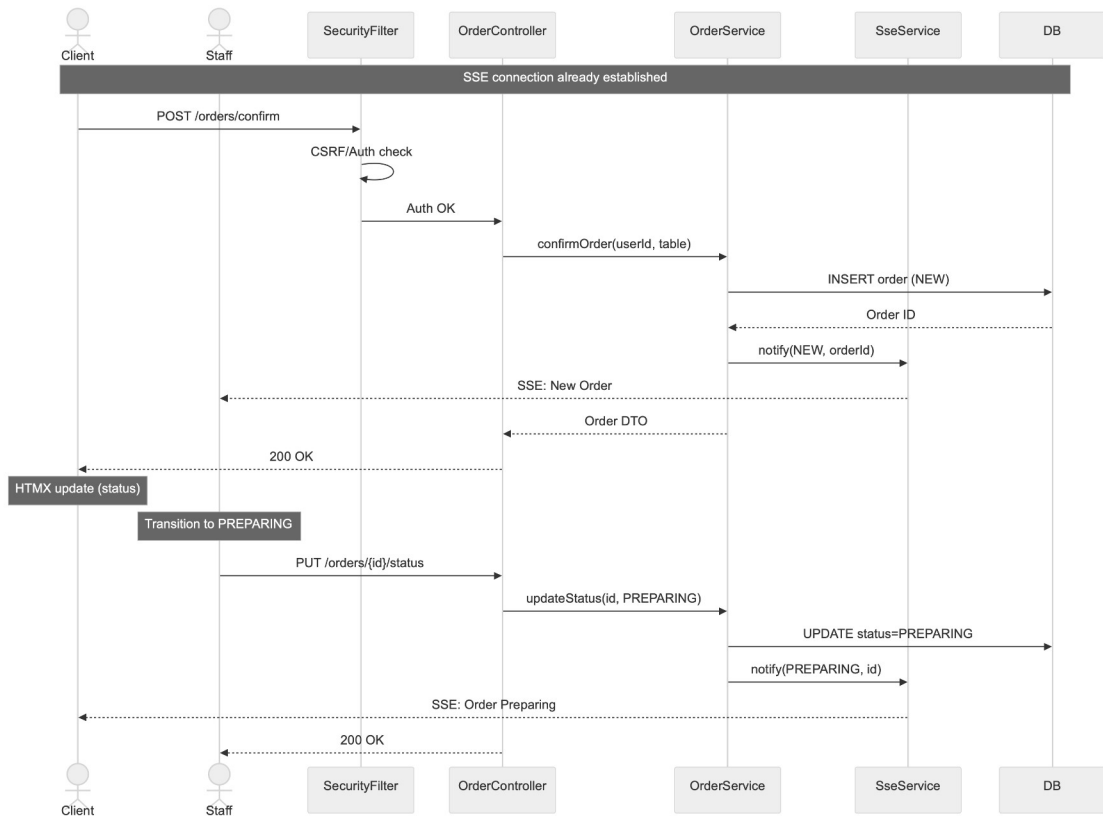


Рис. 2.5 Діаграма послідовності: оформлення замовлення та SSE-сповіщення

2.4.2 Сценарій Б: Оплата замовлення через LiqPay

Сценарій (див. рис. 2.6) описує платіжний потік від ініціювання оплати клієнтом до оновлення статусу замовлення після підтвердження від платіжного шлюзу. Учасниками є: Клієнт (браузер), контролер платежів, сервіс підготовки оплати, сервіс інтеграції з LiqPay, платіжний шлюз (зовнішній сервіс), сервіс обробки зворотних викликів, сервіс сповіщень.

Передумова: замовлення існує в базі даних зі статусом «Очікує оплати»; клієнт перебуває на сторінці підтвердження оплати.

Послідовність дій:

1. Клієнт натискає «Сплатити». Браузер надсилає запит на ініціалізацію платежу до застосунку.
2. Застосунок звертається до модуля безпеки для формування підписаного набору даних (data та signature) за алгоритмом HMAC-SHA1.

3. Клієнту повертається HTML-форма, яка автоматично перенаправляє його на сервер платіжного шлюзу.

4. Платіжний шлюз обробляє реквізити картки. Після успішної транзакції шлюз асинхронно надсилає зворотний виклик (callback) на адресу застосунку.

5. Застосунок передає отримані параметри модулю безпеки для перевірки підпису, що гарантує справжність даних від LiqPay.

6. Після успішної валідації застосунок перевіряє статус платежу від шлюзу. Якщо статус успішний, застосунок оновлює платіжний статус замовлення на «Оплачено». Якщо ж статус транзакції неуспішний, фіксується помилка.

7. Компонент сповіщень розсилає SSE-подію про оновлення статусу, і клієнт миттєво бачить підтвердження оплати на екрані.

Альтернативний потік (B1): якщо підпис зворотного виклику не збігається з очікуваним, система генерує помилку верифікації підпису. Контролер повертає відповідь про некоректний запит. Статус замовлення залишається без змін.

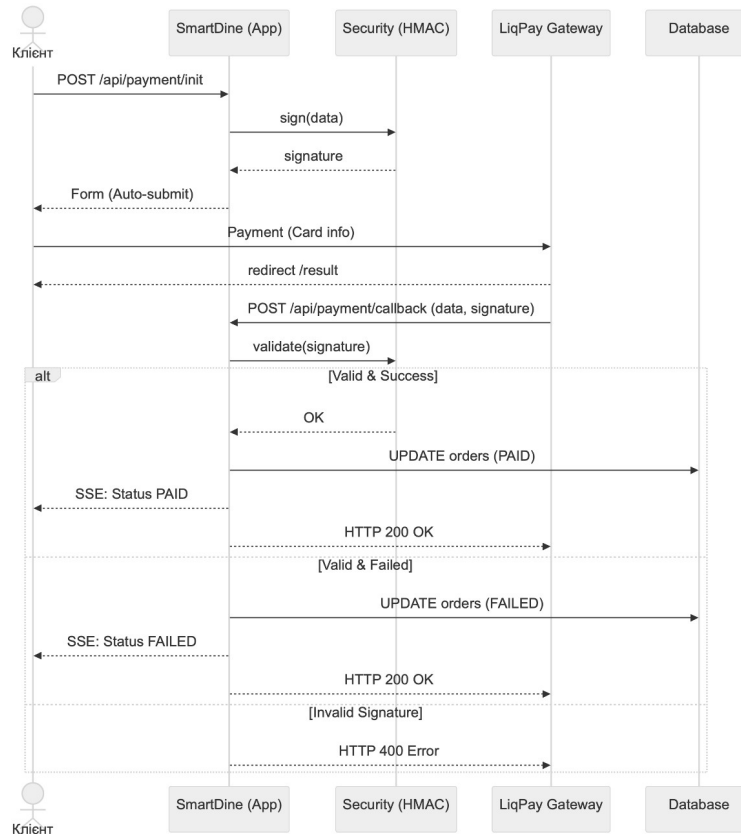


Рис. 2.6 Діаграма послідовності: оплата замовлення через LiqPay

2.5 Проєктування інтерфейсу користувача

Оскільки система SmartDine розроблена насамперед для використання відвідувачами ресторану безпосередньо зі своїх смартфонів, процес проєктування інтерфейсу користувача та користувацького досвіду базувався на парадигмі пріоритету мобільних пристроїв (Mobile-First). Головною метою було створення інтерфейсу, який мінімізує когнітивне навантаження та дозволяє зробити замовлення інтуїтивно зрозумілим навіть для користувачів із низьким рівнем цифрової грамотності.

Основні вимоги до юзабіліті системи:

- Правило трьох кліків: Користувач повинен мати можливість додати страву до кошика та перейти до оплати не більше ніж за три взаємодії з екраном після сканування QR-коду.
- Нульовий бар'єр входу: Відсутність обов'язкових форм реєстрації. Ідентифікація клієнта відбувається автоматично на основі сесії (HTTP-сесії), прив'язаної до сканованого столика.
- Візуальний зворотний зв'язок: Будь-яка дія користувача (додавання до кошика, успішна оплата, виклик офіціанта) повинна супроводжуватися візуальною реакцією інтерфейсу (анімації кнопок, спливні повідомлення).
- Доступність (Accessibility): Інтерфейс спроєктовано з урахуванням стандартів WCAG 2.1: високий контраст тексту та фону, достатній розмір клікабельних зон (мінімум 44x44 CSS-пікселів для зручного натискання пальцем), підтримка читабельних шрифтів.

2.5.1 Сценарії виконання задач користувача (User Flows)

Для забезпечення безперебійної роботи розроблено базові сценарії взаємодії користувачів із системою (User Flow):

Сценарій 1: Ознайомлення з меню та формування замовлення (Клієнт)

1. Клієнт сканує QR-код на столику та миттєво потрапляє на сторінку цифрового меню, де його стіл вже ідентифіковано.
2. Клієнт скролить стрічку страв (категоризовану за типом: закуски, основні страви, напої). Завдяки технології «лінивого завантаження» зображення підвантажуються поступово, економлячи трафік.
3. Натисканням на кнопку "Кошик" біля страви, вона додається до кошика. Анімація підтверджує дію, а лічильник на кнопці Кошика оновлюється.
4. Клієнт відкриває кошик, перевіряє замовлення, залишає коментар (наприклад, "без цибулі") та натискає "Сплатити".
5. Відбувається перехід до віджета LiqPay. Після успішної транзакції клієнт перенаправляється на сторінку статусу замовлення.

Сценарій 2: Відстеження статусу в реальному часі (Клієнт)

1. Після оформлення замовлення клієнт бачить екран статусу ("Нове", "Готується", "Готово").
2. Як тільки кухар на своїй панелі змінює статус страви, завдяки Server-Sent Events (SSE) екран клієнта автоматично, без оновлення сторінки, змінює колір індикатора та статус.

Сценарій 3: Керування чергою замовлень (Кухар)

1. Кухар авторизується в системі та відкриває "Канбан-дошку" замовлень (Kanban Board) на планшеті.
2. При надходженні нового замовлення (через SSE) на дошці миттєво з'являється нова картка із сповіщенням (Web Push).
3. Кухар натискає на картку, щоб перевести замовлення у статус "В роботі", а потім "Готово". Кожен клік асинхронно відправляє HTMX-запит до сервера, який оновлює статус у базі даних і сповіщає клієнта.

2.5.2 Вибір та обґрунтування елементів UI

Для реалізації сценаріїв було обрано набір сучасних елементів інтерфейсу:

- Картковий дизайн: Страви у меню та замовлення на панелі кухаря представлені у вигляді "карток". Це дозволяє чітко розмежувати інформаційні блоки. Кожна картка страви містить високоякісне зображення (з оптимізацією WebP), назву, ціну та кнопку додавання.

- Нижня панель навігації: Використовується замість класичного верхнього меню-гамбургера, оскільки на смартфонах із великою діагоналлю дотягнутися великим пальцем до нижньої частини екрана значно зручніше.

Завдяки інтерфейсу, керованому сервером (через Thymeleaf), основний каркас сторінки формується на сервері, що унеможливлює проблему порожнього екрана при завантаженні важких JavaScript-файлів. Усі інтерактивні елементи (кнопки "Плюс/Мінус", фільтри категорій) працюють через асинхронні запити, забезпечуючи блискавичний відгук інтерфейсу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ ДЛЯ АВТОМАТИЗАЦІЇ РЕСТОРАННОГО СЕРВІСУ

Маючи детальний архітектурний план та спроектовану модель даних, наступним етапом є програмна імплементація системи. Розробка комплексного вебзастосунку вимагає інтеграції різноманітних технологій: від налаштування серверного середовища та розробки бізнес-логіки до створення чуйного клієнтського інтерфейсу. Особливої уваги на цьому етапі потребують питання безпеки, оптимізації продуктивності та надійної взаємодії із зовнішніми сервісами. Етап реалізації перетворює абстрактні UML-моделі на функціонуючий програмний продукт, здатний обробляти реальні клієнтські запити у режимі реального часу.

3.1 Розробка серверної частини

Серверна частина застосунку SmartDine організована відповідно до шарової архітектури «Модель–Представлення–Контролер» (MVC), де кожен шар має чітко визначену відповідальність [12]. Логіку системи розподілено між декількома функціональними рівнями: рівень контролерів забезпечує маршрутизацію та перевірку вхідних даних, тоді як основна бізнес-логіка зосереджена у сервісному шарі. Взаємодія з базою даних здійснюється через репозиторії, що використовують об'єкти передачі даних (DTO) та компоненти перетворення типів для ізоляції внутрішніх моделей від клієнтського інтерфейсу. Обробку виняткових ситуацій та конфігурацію безпеки винесено в окремі модулі, що забезпечує принцип єдиної відповідальності кожного компонента.

3.1.1 Налаштування Spring Security та інтеграція з Keycloak

Центральним елементом захисту є конфігурація безпеки, що визначає єдиний ланцюжок фільтрів для двох режимів роботи: браузерної автентифікації через Keycloak (протокол OAuth 2.0) та перевірки JWT Bearer-токенів для API-запитів.

Оскільки ролі Keycloak передаються у нестандартному полі JWT-токена, реалізовано власний конвертер, що зчитує список ролей та перетворює їх у формат, сумісний зі Spring Security для декларативного контролю доступу:

```
converter.setJwtGrantedAuthoritiesConverter(jwt -> {
    Map<String, Object> realmAccess = jwt.getClaimAsMap("realm_access");
    if (realmAccess == null) return List.of();
    return ((List<String>) realmAccess.get("roles")).stream()
        .map(role -> new SimpleGrantedAuthority("ROLE_" + role.toUpperCase()))
        .collect(Collectors.toList());
});
```

Матриця доступу конфігурується декларативно: маршрути кошика та оплати відкриті для гостей, панелі персоналу та адміністратора захищені відповідними ролями ('WAITER', 'CHEF', 'ADMINISTRATOR') [19]. Захист від міжсайтової підробки запитів (CSRF) реалізовано через спеціалізований репозиторій токенів, сумісний із HTMX-клієнтом. При цьому асинхронні виклики від LiqPay виключені з CSRF-перевірки та захищені верифікацією криптографічного підпису HMAC-SHA1 [20].

3.1.2 Сервісний шар та транзакційність

Сервісний шар реалізований за принципом розділення інтерфейсу та реалізації: для кожної функціональної групи визначено інтерфейс та окремий клас-реалізацію. Логіку роботи із замовленнями розподілено між чотирма спеціалізованими сервісами – читання, обслуговування залу, нарахування лояльності та відгуків. Це запобігає створенню перевантаженого компонента (так званого «сервісу-бога») та спрощує тестування кожного модуля в ізоляції.

Для операцій лише зчитування транзакції позначаються як «тільки для читання», що вимикає механізм відстеження змін Hibernate і знижує споживання пам'яті.

3.1.3 Механізм SSE-сповіщень та транзакційна безпека

Синхронізацію стану в реальному часі централізовано у спеціальному компоненті сповіщень. Критичним архітектурним рішенням є гарантія того, що подія через SSE надходить клієнту лише після успішного збереження транзакції у базі даних. Для цього використовується механізм зворотного виклику після фіксації транзакції:

```
TransactionSynchronizationManager.registerSynchronization(
    new TransactionSynchronization() {
        @Override
        public void afterCommit() {
            orderNotifier.notifyStatusChange(userId, response, newStatus);
        }
    });
```

Цей підхід гарантує, що клієнт ніколи не отримає сповіщення про стан, який було відкочено транзакцією [14]. Окрім SSE, той самий компонент розсилає фонові вебсповіщення: клієнту – при готовності страви, офіціанту – при зміні статусу замовлення.

Додатковим елементом безпеки системи є захист точок входу клієнтів. URL-посилання, за кодовані у QR-кодах столиків, захищено унікальною криптографічною HMAC-сигнатурою. Це повністю унеможливорює несанкціоновану підміну номера столика злоумисником шляхом ручної зміни параметра '?table=N' у браузері, захищаючи замовлення інших гостей.

3.1.4 Ідентифікація гостьових сесій та кешування

Для підтримки повного циклу замовлення неавторизованого відвідувача система генерує тимчасовий рядковий ідентифікатор на основі ідентифікатора HTTP-сесії, який використовується нарівні з постійними ідентифікаторами

авторизованих користувачів. Глобальний перехоплювач контролерів автоматично додає до кожної сторінки необхідні атрибути – ключ Push-сповіщень, баланс лояльності – без дублювання коду. Окремим архітектурним рішенням є механізм безшовного злиття сесій: якщо гість авторизується після наповнення кошика або створення замовлення, система автоматично переносить усі його дані до облікового запису.

Для зменшення навантаження на базу даних при масовому перегляді меню реалізовано кешування у пам'яті через бібліотеку Caffeine. Дані завантажуються з бази лише при першому запиті, а при редагуванні страви адміністратором кеш анулюється автоматично завдяки декларативним анотаціям.

3.1.5 Віртуальні потоки, Flyway та розрахунок замовлення

Для ефективної обробки тисяч тривалих SSE-з'єднань увімкнено підтримку віртуальних потоків (Project Loom), що усуває обмеження класичного пулу потоків операційної системи без збільшення споживання пам'яті [12].

Версіонування схеми бази даних забезпечено через Flyway – усього застосовано 15 послідовних міграцій. Режим перевірки схеми Hibernate налаштовано так, щоб запобігати випадковим змінам структури бази та гарантувати її відповідність коду на всіх середовищах.

Бізнес-логіку розрахунку підсумкової вартості та орієнтовного часу приготування замовлення інкапсульовано в окремому компоненті-калькуляторі. Для грошових розрахунків використовується тип з фіксованою точністю та заокругленням «половина вгору», що виключає накопичення похибки, характерної для типів з рухомою комою.

Реалізована серверна частина відповідає всім архітектурним вимогам: рольовий контроль доступу через Keycloak, транзакційно-безпечна відправка SSE-сповіщень, кешування меню, версіонування схеми бази даних та підтримка гостей сесій. Принцип розділення інтерфейсу та реалізації забезпечує повну

тестованість сервісного шару в ізоляції, що підтверджується набором автоматизованих тестів (Розділ 4).

3.2 Реалізація інтерактивного фронтенду

3.2.1 Концепція Server-Driven UI та PWA

Замість використання важких клієнтських JavaScript-фреймворків (таких як React або Angular), що переносять логіку маршрутизації та рендерингу на пристрій користувача, клієнтська частина застосунку SmartDine побудована за архітектурою Server-Driven UI [13] (інтерфейс, керований сервером). Цей підхід реалізовано на базі серверного шаблонізатора Thymeleaf.

Суть підходу полягає в тому, що браузер отримує від сервера не «сирі» дані у форматі JSON, які потрібно обробляти та перетворювати на HTML засобами клієнтського коду, а вже готові, згенеровані сервером фрагменти розмітки. Це значно зменшує обсяг клієнтського JavaScript-коду, прискорює час першого відмальовування контенту та знижує енергоспоживання мобільних пристроїв.

Для перевикористання коду інтерфейс розбито на модульні компоненти – фрагменти шаблонів для картки страви, панелі персоналу та кошика. Контролери повертають назву конкретного фрагмента замість повної сторінки, коли цього вимагає логіка часткового оновлення.

Використання PWA-технології спрощує доступ до системи, дозволяючи персоналу працювати з інтерфейсом як з мобільним застосунком без необхідності використання сторонніх магазинів програмного забезпечення. Завдяки Service Worker та файлу маніфесту, технологія забезпечує локальне кешування статичних ресурсів, що прискорює завантаження інтерфейсу та робить роботу застосунку стабільною навіть за умов слабого інтернет-з'єднання.

3.2.2 Часткове оновлення DOM за допомогою HTMX

Для надання статичному HTML-коду динаміки сучасних Single-Page Applications (SPA) без написання ручного JavaScript-коду використано легковагову бібліотеку HTMX. Вона розширює можливості HTML, дозволяючи виконувати асинхронні запити безпосередньо з елементів сторінки через спеціальні атрибути.

Приклад додавання страви до кошика без перезавантаження сторінки (лістинг із шаблону картки страви):

```
<form hx-post="/htmx/cart/items" hx-swap="none">
  <input type="hidden" name="dishId" th:value="${dish.id}">
  <input type="hidden" name="quantity" value="1">
  <button class="btn btn-primary add-to-cart-btn" type="submit">
     У кошик
  </button>
</form>
```

У цьому прикладі спеціальний атрибут перехоплює стандартну подію відправки форми та виконує асинхронний запит до сервера. Додаткові інструкції вказують, що відповідь не повинна замінювати поточний елемент – натомість сервер повертає дані для оновлення кошика або сигнал для відображення спливного повідомлення.

Аналогічний підхід застосовується на панелі персоналу для швидкої зміни статусу замовлення:

```
<button type="button" class="btn btn-warning btn-sm"
  th:attr="hx-put=@{/htmx/staff/orders/{id}/status(id=${
order.id},newStatus=PREPARING)},
  hx-target='#staffBoardRoot',
  hx-swap='outerHTML'"
  data-toast-pending="Оновлюємо статус...">Готувати</button>
```

При натисканні кухарем кнопки «Готувати», бібліотека відправляє запит на зміну стану, а сервер повертає оновлений фрагмент дошки замовлень, який автоматично замінює старий контейнер на сторінці.

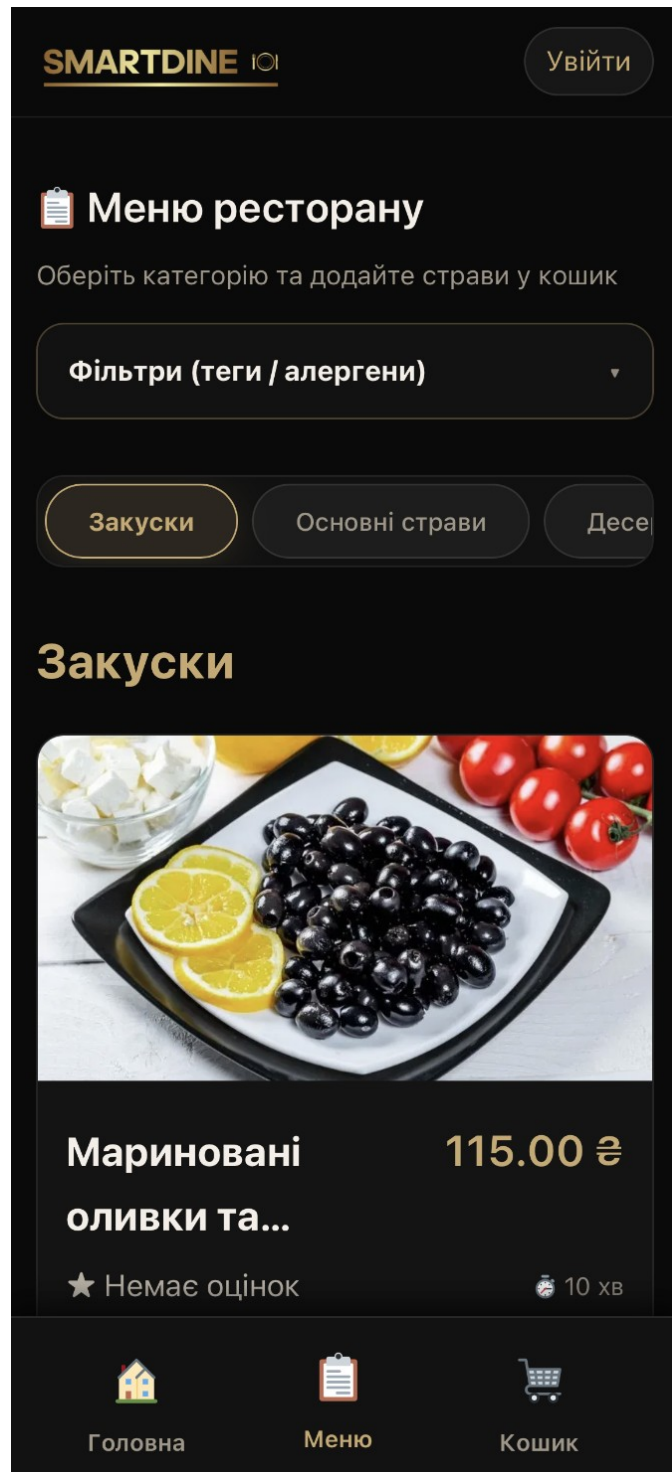


Рис. 3.1 Інтерфейс клієнтського меню

3.2.3 Механізм Server-Sent Events (SSE) для панелі персоналу

Ресторанна система вимагає миттєвої реакції на події: надходження нового замовлення, виклик офіціанта або зміна статусу страви. Замість неефективного

циклічного опитування сервера (коли браузер регулярно надсилає HTTP-запити для перевірки змін) реалізовано технологію Server-Sent Events (SSE).

SSE забезпечує стійке односпрямоване з'єднання від сервера до браузера поверх стандартного протоколу HTTP. При авторизації персоналу клієнтський код автоматично відкриває постійне з'єднання через спеціальну адресу підписки. Сервер утримує це з'єднання відкритим (використовуючи віртуальні потоки, описані у підрозділі 3.1) і надсилає події у стандартному форматі потоку подій.

Коли клієнт робить замовлення, сервер генерує подію оновлення замовлення. На стороні клієнта вона або обробляється атрибутом HTMX (який автоматично оновлює відповідний фрагмент сторінки), або власним JavaScript-кодом для відображення системних сповіщень:

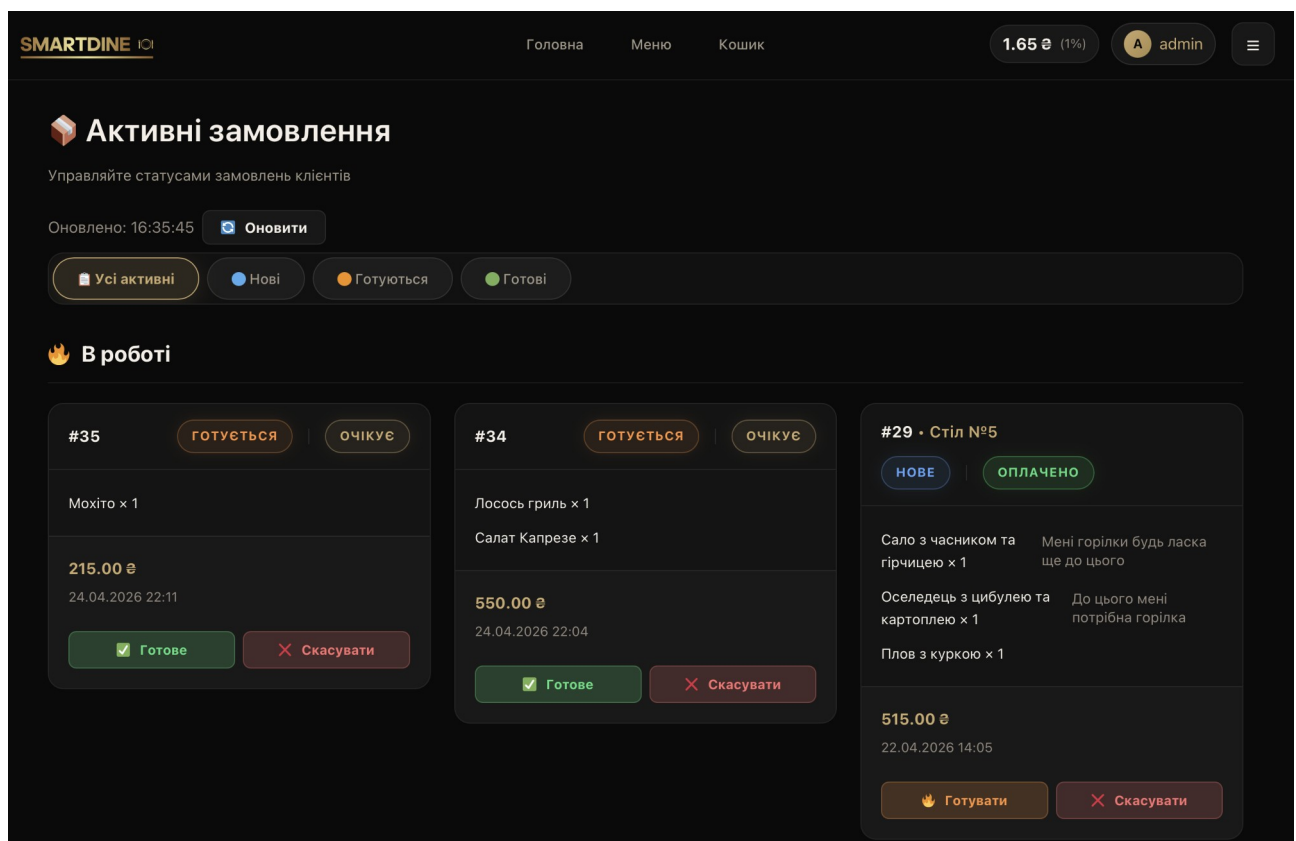


Рис. 3.2 Панель персоналу з динамічною дошкою замовлень

На відміну від WebSockets, SSE є простішим у налаштуванні, не вимагає спеціальних протоколів балансування навантаження, не блокується

корпоративними фаєрволами і має вбудований у браузер механізм автоматичного перепідключення у разі обриву мережі.

3.2.4 Оптимізація графічних ресурсів

Оскільки меню ресторану містить велику кількість фотографій страв, оптимізація завантаження медіафайлів є критичною для продуктивності (зокрема метрики Largest Contentful Paint – LCP).

У системі реалізовано механізм автоматичної конвертації завантажених адміністратором зображень у сучасний формат WebP за допомогою бібліотек Thumbnailator та webp-imageio. WebP забезпечує на 25-35% краще стиснення порівняно з JPEG при ідентичній візуальній якості. Додатково сервер генерує набір мініатюр різних розмірів (шириною 320px, 480px, 640px, 800px).

Для відображення оптимізованих зображень використовується атрибут 'srcset', який дозволяє браузеру самостійно обрати найменший файл, достатній для екрану поточного пристрою:

```

```

Додатково застосовуються техніки пріоритизації завантаження: головне зображення сторінки завантажується з найвищим пріоритетом одразу, тоді як решта зображень завантажуються відкладено – лише коли вони наближаються до видимої частини екрана.

3.2.5 Адаптивний дизайн та стилізація

Оскільки основною цільовою аудиторією застосунку є гості ресторану, які сканують QR-код за допомогою власних смартфонів, інтерфейс спроектовано за принципом пріоритету мобільних пристроїв (Mobile-First).

Для стилізації використано виключно чистий CSS без залучення зовнішніх CSS-фреймворків (Bootstrap, TailwindCSS). Це рішення дозволило мінімізувати розмір таблиць стилів та уникнути примусового перемальовування сторінки, що критично впливає на швидкість завантаження.

Основою дизайн-системи є CSS-змінні (Custom Properties), які визначають кольорову палітру (включаючи підтримку темної теми), типографіку та відступи. Гнучкість адаптивної верстки забезпечується використанням сучасних модулів Flexbox та CSS Grid, які дозволяють автоматично перебудовувати сітку карток меню від 1 колонки на мобільних пристроях до 3 колонок на десктопних моніторах.

3.2.6 Візуалізація аналітики

Для задоволення потреб адміністратора закладу реалізовано інтерактивну інформаційну панель із ключовими показниками ефективності ресторану. Візуалізація даних забезпечується легкою JavaScript-бібліотекою Chart.js.

Для уникнення зайвих мережевих запитів при завантаженні панелі дані агрегуються на сервері та передаються безпосередньо у JavaScript-контекст під час генерації сторінки:

```
<!-- dashboard.html -->
<script th:inline="javascript">
    // Thymeleaf автоматично серіалізує Java-колекції у JSON-масиви
    window.topDishesData = /*[[${topDishes}]]*/ [];
    window.hourlyOrdersData = /*[[${hourlyOrdersToday}]]*/ [];
</script>
<script th:src="@{/js/dashboard-charts.js}"></script>
```

Цей підхід дозволяє графікам відмальовуватися миттєво після завантаження сторінки, без додаткових запитів до сервера.

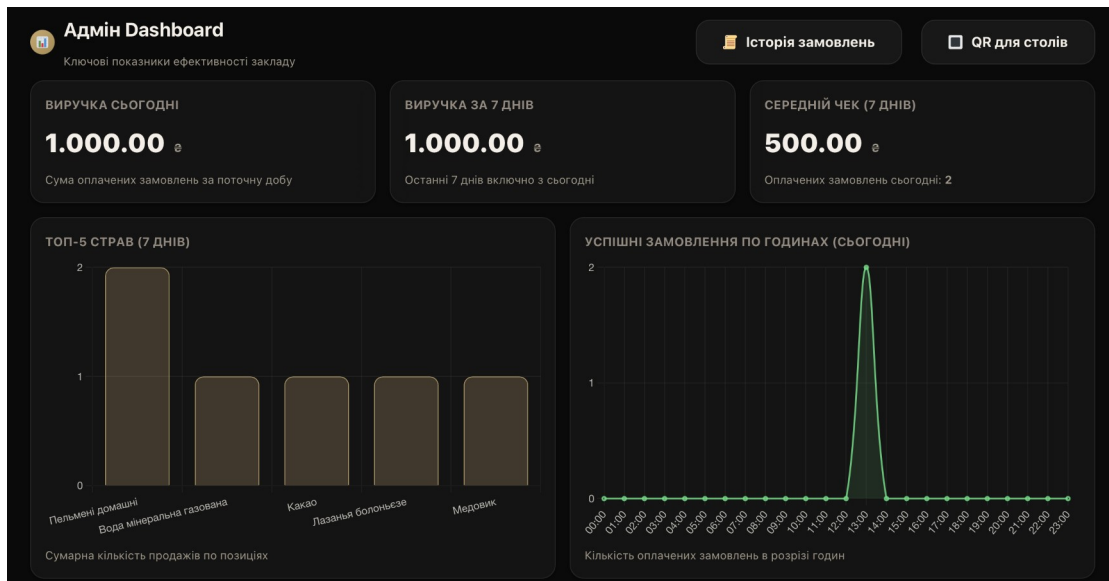


Рис. 3.3 Інформаційна панель адміністратора з графіками аналітики

Обраний підхід на базі серверного генерування інтерфейсу, бібліотеки HTMX та SSE дозволив створити реактивний, швидкий та енергоефективний вебзастосунок без ускладнення архітектури важкими клієнтськими фреймворками. Генерація розмітки на боці сервера гарантує миттєве відображення контенту, HTMX забезпечує плавність взаємодії з кошиком, відмова від сторонніх CSS-бібліотек мінімізує розмір ресурсів, а комбінація SSE з оптимізованими зображеннями у форматі WebP задовольняє всі нефункціональні вимоги щодо швидкодії мобільних пристроїв та своєчасного інформування персоналу.

3.3. Інтеграція зовнішніх сервісів

3.3.1 Платіжний модуль LiqPay

Для забезпечення можливості безконтактної оплати замовлень онлайн до системи інтегровано український платіжний шлюз LiqPay [21]. Архітектура інтеграції побудована за принципом Checkout-сторінки: клієнт не вводить дані

банківської картки на сервері ресторану, а перенаправляється на захищену сторінку LiqPay, що гарантує відповідність стандарту безпеки PCI DSS.

Для ініціалізації платежу сервер ресторану повинен сформувати JSON-об'єкт із параметрами замовлення (сума, валюта, ідентифікатор) та підписати його за допомогою алгоритму HMAC-SHA1.

Логіка створення підпису реалізована у сервісі інтеграції платіжного шлюзу:

```
// LiqPayServiceImpl.java - генерація підпису HMAC-SHA1
private String buildSignature(String data) {
    try {
        String input = liqPayConfig.getPrivateKey() + data +
            liqPayConfig.getPrivateKey();
        MessageDigest sha1 = MessageDigest.getInstance("SHA-1");
        byte[] hash = sha1.digest(input.getBytes(StandardCharsets.UTF_8));
        return Base64.getEncoder().encodeToString(hash);
    } catch (Exception e) {
        throw new IllegalStateException("Failed to build LiqPay signature", e);
    }
}
```

У цьому алгоритмі конкатенується приватний ключ закладу (`private_key`), Base64-закодований JSON (`data`) та знову приватний ключ. Отриманий рядок хешується SHA-1 і кодується у Base64. Цей підпис (`signature`) гарантує незмінність суми замовлення під час передачі клієнту.

Після успішного або невдалого платежу сервери LiqPay відправляють асинхронний POST-запит (зворотний виклик) на визначену адресу сервера.

```
// PaymentCallbackServiceImpl.java - обробка підтвердження платежу
@Override
@Transactional
public void handleCallbackSuccess(LiqPayCallbackDto callback) {
    Integer dbOrderId = LiqPayUtil.extractDbOrderId(callback.getOrderId());
    Order order = orderRepository.findById(dbOrderId)
        .orElseThrow(() -> new
            NotFoundException(ErrorMessages.ORDER_NOT_FOUND));
    if (PaymentStatus.SUCCESS.equals(order.getPaymentStatus())) {
        return; // Ідемпотентність: платіж вже оброблено
    }
    order.setPaymentStatus(PaymentStatus.SUCCESS);
    if (OrderStatus.READY.equals(order.getStatus())) {
        order.setStatus(OrderStatus.COMPLETED);
    }
    orderRepository.save(order);
    earnCashbackIfEligible(order); // Нарахування лояльності
}
```

Система перевіряє валідність підпису зворотного запиту, щоб унеможливити підробку успішного статусу зловмисником. Важливою архітектурною деталлю є перевірка ідемпотентності: якщо зворотний виклик надійде кілька разів (наприклад, через збій мережі), система не нараховуватиме бонусні бали двічі.

3.3.2 Web Push сповіщення

Окрім з'єднання, що працює лише коли вкладка ресторану відкрита у браузері, система підтримує фонові повідомлення через технологію Web Push API. Це дозволяє офіціантам отримувати сповіщення на заблокований екран смартфона або розумний годинник, коли клієнт робить нове замовлення або натискає кнопку виклику персоналу. Безпека передачі push-повідомлень забезпечується стандартом VAPID (Voluntary Application Server Identification). Сервер має пару криптографічних ключів (публічний і приватний стандарту P-256).

Процес взаємодії розпочинається з етапу підписки, під час якого клієнтський браузер використовує отриманий від сервера публічний ключ для реєстрації на push-сервері відповідного вендора (наприклад, FCM для Chrome або APNs для Safari). Отриманий унікальний об'єкт підписки, що містить кінцеву точку (endpoint) та ключі шифрування, передається на серверну частину SmartDine для тривалого зберігання у базі даних. У момент виникнення події спеціалізований сервіс повідомлень формує інформаційне навантаження, шифрує його за допомогою приватного VAPID-ключа та надсилає до відповідного вузла доставки. Така інтеграція значно підвищує швидкість реакції персоналу, що є критично важливим для безперебійної роботи закладу у періоди високого навантаження.

3.3.3 Сервіс автентифікації Keycloak (Identity Provider)

Хоча деталі обробки ролей (JWT) описані у підрозділі 3.1, сам по собі Keycloak виступає як зовнішній незалежний сервіс (Identity Provider / IdP) [15]. Застосунок SmartDine не зберігає паролі користувачів і не займається логікою скидання чи реєстрації облікових записів, делегуючи ці завдання Keycloak за протоколом OAuth 2.0 / OpenID Connect (OIDC).

Keycloak розгорнуто як окремий контейнер. Усі HTTP-запити до сторінок входу (Login) або профілю користувача перенаправляються сервером закладу безпосередньо до Keycloak. Після введення облікових даних Keycloak повертає серверу спеціальний код, який серверна частина обмінює на токени доступу. Це забезпечує Single Sign-On (SSO) і гарантує, що персональні дані користувачів захищені сертифікованим індустріальним рішенням, стійким до методу «грубої сили» (перебору паролів) та витоків даних.

Інтеграція зовнішніх сервісів розширює функціональні можливості застосунку SmartDine до рівня повноцінної системи самообслуговування. Делегування управління ідентичністю до Keycloak знімає ризики зберігання паролів. Реалізація криптографічних алгоритмів HMAC-SHA1 для LiqPay гарантує фінансову безпеку як закладу, так і його клієнтів. Своєю чергою, впровадження стандарту VAPID для Web Push повідомлень забезпечує автономність персоналу, дозволяючи їм миттєво реагувати на запити без необхідності постійно тримати екран термінала увімкненим.

3.4 Персоналізація та система лояльності

3.4.1 Персоналізація меню та рекомендації страв

Сучасні заклади харчування повинні враховувати індивідуальні харчові обмеження клієнтів (алергії, непереносимість лактози, вегетаріанство). У застосунку SmartDine розроблено механізм персоналізації, який дозволяє авторизованим користувачам обирати алергени для виключення з меню.

При формуванні рекомендацій (популярних страв або супутніх пропозицій), система автоматично фільтрує позиції з небажаними інгредієнтами. Алгоритм реалізовано у сервісі рекомендацій:

```
@Override
@Transactional(readOnly = true)
public List<DishResponseDto> getRecommendations(String userId) {
    Set<String> excluded = userPreferenceService.getExcludedAllergens(userId);
    List<Dish> base = dishRepository.findRecommendedDishes(userId);
    List<DishResponseDto> dishes = base.stream()
        .map(dishMapper::toResponseDto)
        .peek(dto -> { /* завантаження алергенів та тегів */ })
        .filter(dto -> excluded.isEmpty()
            || dto.getAllergens() == null
            || dto.getAllergens().stream().noneMatch(excluded::contains))
        .collect(Collectors.toList());
    dishRatingService.enrichWithRatings(dishes);
    return dishes.size() > 6 ? dishes.subList(0, 6) : dishes;
}
```

Також реалізовано алгоритм супутніх пропозицій: при додаванні страви до кошика сервер аналізує історичні чеки інших клієнтів і пропонує страву, яку найчастіше купують разом із поточною. Алгоритм застосовує зважений випадковий вибір, що запобігає одноманітності пропозицій.

3.4.2 Механізм системи лояльності (бонусні бали)

Для стимулювання повторних візитів до ресторану реалізовано накопичувальну бонусну систему, яка веде облік балансу та повну історію транзакцій кожного користувача.

Після успішної оплати замовлення через LiqPay спрацьовує нарахування бонусних балів. Відсоток нарахування залежить від історії клієнта:

- Базовий рівень – 1% (менше 10 замовлень);
- Срібний рівень – 3% (від 10 до 99 замовлень);
- Золотий рівень – 5% (від 100 замовлень).

```
// LoyaltyServiceImpl.java - визначення ставки та нарахування
@Override
@Transactional(readOnly = true)
public BigDecimal resolveCashbackRate(UUID userId) {
    Long count = orderRepository.countSuccessfulOrdersByUserId(userId.toString());
    if (count != null && count >= 100) return RATE_100; // 0.05
}
```

```

        if (count != null && count >= 10) return RATE_10;    // 0.03
        return RATE_NEW;                                     // 0.01
    }
    @Override
    @Transactional
    public void earnPointsInternal(UUID userId, BigDecimal orderAmount, String
reference) {
        // Перевірка на ідемпотентність
        if (reference != null &&
loyaltyTransactionRepository.existsByReference(reference)) {
            return;
        }
        BigDecimal rate = resolveCashbackRate(userId);
        BigDecimal points = normalizeMoney(orderAmount.multiply(rate));
        creditPointsInternal(userId, points, reference);
    }
}

```

Ключовою архітектурною деталлю є принцип ідемпотентності: кожне нарахування отримує унікальний ідентифікатор операції. Оскільки зворотний виклик від LiqPay може надходити кількаразово через нестабільність мережі, сервіс лояльності перевіряє наявність запису з таким ідентифікатором у базі даних. Якщо він вже існує – нарахування не відбувається, що унеможливорює подвійне нарахування балів.

При наступному візиті користувач може використати накопичені бали для часткової або повної оплати замовлення. Списання відбувається в рамках тієї ж транзакції, що й оновлення статусу замовлення.

Система лояльності та персоналізації меню додають застосунку відчутну комерційну цінність. Механізм виключення алергенів забезпечує безпеку клієнтів і покращує користувацький досвід. Накопичувальна багаторівнева бонусна система стимулює клієнтів здійснювати більше замовлень. Архітектура фінансових транзакцій системи лояльності розроблена з дотриманням принципів ACID [18] та ідемпотентності, що гарантує цілісність балансів користувачів навіть за умов нестабільної роботи зовнішніх платіжних шлюзів.

4 ТЕСТУВАННЯ ТА РОЗГОРТАННЯ СИСТЕМИ В ХМАРНІЙ ІНФРАСТРУКТУРІ

Верифікація розробленої архітектури SmartDine передбачає проведення серії тестів для оцінки її стійкості до навантажень, безпеки обробки даних та коректності розгортання в хмарному середовищі. У цьому розділі описано методологію автоматизованого та ручного тестування, оцінку продуктивності системи, а також процес публікації готового продукту.

4.1 Методи тестування

Для забезпечення надійності, безпеки та продуктивності системи SmartDine застосовано комплексний підхід, що охоплює перевірку бізнес-логіки на рівні коду, оцінку клієнтського інтерфейсу та стрес-тестування сервера.

4.1.1 Автоматизоване тестування

Реалізовано багаторівневу стратегію тестування за принципом піраміди. Загалом розроблено 187 автоматизованих тестів, серед яких 139 модульних (Unit) тестів, 43 інтеграційних та 5 наскрізних (E2E) тестів, що охоплюють майже всю бізнес-логіку системи. Перевірка бізнес-логіки в ізоляції здійснюється за допомогою модульних тестів (JUnit 5 + Mockito), де зовнішні залежності замінюються імітаційними об'єктами. Інтеграційний рівень перевірки охоплює шар доступу до даних (із використанням контейнеризованих СУБД) та тестування інтерфейсу програмування (API), що симулює повний цикл HTTP-запитів без запуску сервера. Фінальний етап перевірки реалізовано за допомогою наскрізних тестів (Playwright), які автоматизують роботу реального браузерa, перевіряючи динамічне оновлення інтерфейсу та SSE-з'єднання.

```
Results:

Tests run: 187, Failures: 0, Errors: 0, Skipped: 0

-----
BUILD SUCCESS
-----
Total time: 17.772 s
```

Рис. 4.1 Результати автоматизованого тестування

4.1.2 Оцінка продуктивності клієнтської частини

Оскільки застосунок орієнтований на мобільні пристрої, для аудиту клієнтської частини використовувався Google Lighthouse. Завдяки архітектурі інтерфейсу, керованого сервером, відмові від важких пакунків JavaScript та конвертації зображень у WebP отримано такі результати:

Результати аудиту для десктопної версії:

Швидкість: 97 / 100; Доступність: 100; Найкращі практики: 100; SEO: 100

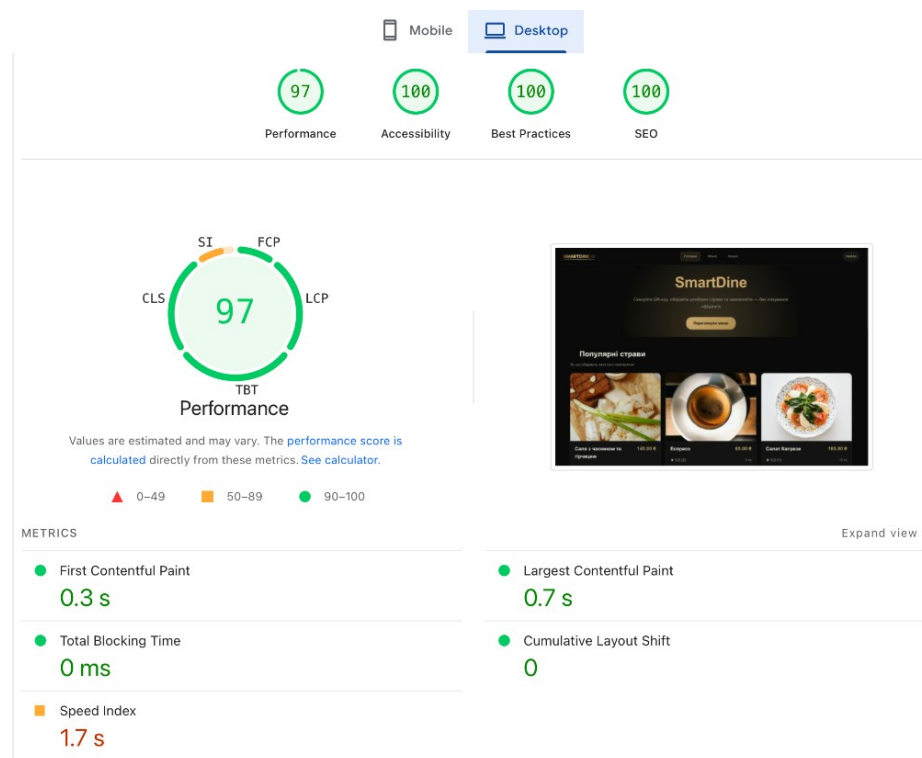


Рис. 4.2 Результати аудиту Google Lighthouse (Desktop-версія)

Результати аудиту для мобільної версії:

Швидкість: 92 / 100 (FCP – 0.9с, LCP – 3.0с); Доступність: 100; Найкращі практики: 100; SEO: 100

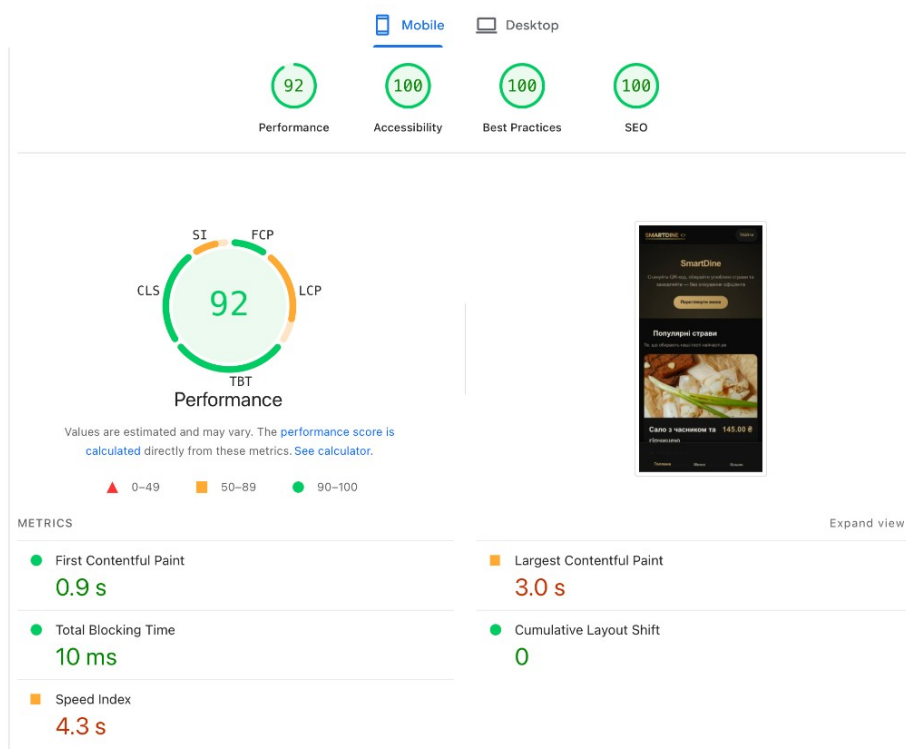


Рис. 4.3 Результати аудиту Google Lighthouse (Mobile-версія)

4.1.2 Навантажувальне та стрес-тестування

Для перевірки стійкості серверної частини під піковими навантаженнями проведено стрес-тестування за допомогою інструменту k6 [22]. Тестування проводилося на розгорнутому екземплярі системи у хмарі Azure VM (конфігурація Standard B2ls v2: 2 vCPUs, 4 GiB memory), що дозволило оцінити реальну продуктивність у продуктивному середовищі. Було реалізовано два сценарії: віртуальні користувачі через короткі проміжки часу (кожні 5 секунд) надсилали запити до точки доступу меню.

Сценарій 1: Робоче навантаження (300 віртуальних користувачів, 7.5 хв.):

– Виконано запитів: 10 748; Помилки: 0.00%, Медіана: 459 мс; 95-й перцентиль: 674 мс.

Цей тест імітує ситуацію повного завантаження великого ресторану. Система відпрацювала бездоганно: всі HTTP-відповіді мали статус 200, а час обробки запитів повністю відповідав нормативам (менше 1 секунди).

```

scenarios: (100.00%) 1 scenario, 300 max VUs, 7m30s max duration (incl. graceful stop):
  * default: Up to 300 looping VUs for 7m0s over 4 stages (gracefulRampDown: 30s, gracefulStop: 30s)

■ THRESHOLDS

http_req_duration
✓ 'p(95)<5000' p(95)=674.75ms

http_req_failed
✓ 'rate<0.01' rate=0.00%

■ TOTAL RESULTS

checks_total.....: 10748 25.42375/s
checks_succeeded...: 100.00% 10748 out of 10748
checks_failed.....: 0.00% 0 out of 10748

✓ status 200

HTTP
http_req_duration.....: avg=505.82ms min=408.84ms med=459.56ms max=3.91s p(90)=593.37ms p(95)=674.75ms
{ expected_response:true }...: avg=505.82ms min=408.84ms med=459.56ms max=3.91s p(90)=593.37ms p(95)=674.75ms
http_req_failed.....: 0.00% 0 out of 10748
http_reqs.....: 10748 25.42375/s

EXECUTION
iteration_duration.....: avg=5.51s min=5.4s med=5.46s max=8.91s p(90)=5.62s p(95)=5.75s
iterations.....: 10748 25.42375/s
vus.....: 1 min=1 max=300
vus_max.....: 300 min=300 max=300

NETWORK
data_received.....: 991 MB 2.3 MB/s
data_sent.....: 4.0 MB 9.4 kB/s

```

Рис. 4.4 Результати тестування k6 (до 300 користувачів)

Сценарій 2: Стрес-тест (1000 віртуальних користувачів, 12.5 хв.):

– Виконано запитів: 47 259; Помилки: 0.00%; 95-й перцентиль: 15.76 с.

Цей сценарій імітує екстремальне навантаження (наприклад, цілеспрямовану DDoS-атаку або масове одночасне сканування меню сотнями користувачів). Незважаючи на деградацію часу відповіді (через вичерпання пулу з'єднань з базою даних), використання віртуальних потоків на сервері запобігло критичному збою — жоден запит не був відхилений, що підтверджує високу відмовостійкість системи.

```

scenarios: (100.00%) 1 scenario, 1000 max VUs, 12m30s max duration (incl. graceful stop):
  * default: Up to 1000 looping VUs for 12m0s over 5 stages (gracefulRampDown: 30s, gracefulStop: 30s)

■ THRESHOLDS

http_req_duration
✗ 'p(95)<8000' p(95)=15.76s

http_req_failed
✓ 'rate<0.01' rate=0.00%

■ TOTAL RESULTS

checks_total.....: 47259 65.209844/s
checks_succeeded...: 100.00% 47259 out of 47259
checks_failed.....: 0.00% 0 out of 47259

✓ status 200

HTTP
http_req_duration.....: avg=5.1s min=408.95ms med=3.23s max=36.78s p(90)=12.68s p(95)=15.76s
{ expected_response:true }...: avg=5.1s min=408.95ms med=3.23s max=36.78s p(90)=12.68s p(95)=15.76s
http_req_failed.....: 0.00% 0 out of 47259
http_reqs.....: 47259 65.209844/s

EXECUTION
iteration_duration.....: avg=10.11s min=5.41s med=8.24s max=41.78s p(90)=17.69s p(95)=20.77s
iterations.....: 47259 65.209844/s
vus.....: 5 min=2 max=1000
vus_max.....: 1000 min=1000 max=1000

NETWORK
data_received.....: 4.4 GB 6.0 MB/s
data_sent.....: 17 MB 24 kB/s

```

Рис. 4.5 Результати стрес-тестування k6 (до 1000 користувачів)

Проведене тестування підтвердило високу якість розробленого програмного забезпечення. Завдяки Server-Driven UI клієнтська частина досягла показників Performance у 92-97 балів, що забезпечує миттєве завантаження меню навіть за умов слабкого мобільного зв'язку. Бекенд продемонстрував здатність безперебійно обробляти тисячі одночасних запитів без падінь, що гарантує надійність системи SmartDine в умовах інтенсивної експлуатації.

4.2 Тест-кейси та результати їх виконання

Для підтвердження коректності роботи системи SmartDine проведено мануальне тестування критичних сценаріїв за методом «чорного ящика» (Black-Box Testing).

Таблиця 4.1

Результати виконання тест-кейсів

ID	Опис тест-кейсу	Передумова	Кроки виконання	Очікуваний результат	Фактичний результат	Статус
ТС-01	Оформлення замовлення та оплата через LiqPay	У кошику клієнта є обрані страви.	1. Натиснути "Сплатити". 2. Ввести тестові реквізити у віджеті LiqPay. 3. Підтвердити оплату.	Сервер отримує callback, верифікує підпис (HMAC-SHA1) та переводить статус на PAID.	Замовлення сплачено, статус змінено на PAID, клієнт бачить підтвердження.	Passed
ТС-02	Миттєве SSE-сповіщення кухаря при новому замовленні	Кухар авторизований і тримає відкритою Канбан-дошку.	Клієнт з іншого пристрою оформлює нове оплачене замовлення.	На планшеті кухаря без перезавантаження сторінки з'являється нова картка замовлення.	Картка з'явилася миттєво через Server-Sent Events зі звуковим сповіщенням.	Passed
ТС-03	Захист доступу до адміністративної панелі (RBAC)	Користувач має роль CUS TOMER.	Перейти за прямим посиланням /htmx/admin/orders.	Сервер повертає HTTP 403 (Forbidden), сторінку не відображено.	Доступ заблоковано фільтром Spring Security, дані не скомпрометовано.	Passed

Результати тестування підтверджують, що розроблена система SmartDine відповідає заявленим функціональним та нефункціональним вимогам.

Автоматизоване Unit- та інтеграційне тестування довело надійність розрахункової бізнес-логіки та захист бази даних від неконсистентних станів (race conditions). Стрес-тестування (k6) підтвердило високу відмовостійкість архітектури на базі віртуальних потоків Project Loom. Мануальні тест-кейси продемонстрували, що усі критичні сценарії – формування кошика, інтеграція зовнішніх оплат LiqPay, оновлення у реальному часі через Server-Sent Events та механізми Role-Based Access Control (RBAC) – функціонують безперебійно.

Програмний продукт має високий ступінь готовності та може бути переданий до етапу дослідної експлуатації у реальних закладах громадського харчування.

4.3 Розгортання застосунку

4.3.1 Контейнеризація (Docker та Docker Compose)

Для забезпечення портативності, ізоляції та передбачуваності середовища виконання, розгортання застосунку SmartDine базується на технології контейнеризації Docker [23]. Архітектура розгортання на робочому сервері оркеструється за допомогою інструменту Docker Compose, який декларативно описує взаємодію трьох ключових сервісів:

1. База даних: Контейнер на базі офіційного образу PostgreSQL. Дані зберігаються у спеціальних томах Docker, що гарантує збереження та цілісність інформації навіть після перестворення контейнера.

2. Сервіс автентифікації: Контейнер на базі Keycloak. Запускається у робочому режимі з підключенням до тієї ж бази даних PostgreSQL (у сервісі автентифікації є власна схема в межах тієї ж БД для збереження користувачів та клієнтів).

3. Основний застосунок: Контейнер зі Spring Boot застосунком. Збирається за допомогою багатоетапної збірки: спочатку інструмент збірки компілює код, а

потім готовий файл переноситься у легкий образ середовища виконання. Це значно зменшує розмір кінцевого образу та підвищує безпеку (оскільки у робочому контейнері відсутні вихідні коди та інструменти розробки).

Усі три контейнери об'єднані у віртуальну мережу (Docker network), що дозволяє їм взаємодіяти один з одним за внутрішніми DNS-іменами (наприклад, 'jdbc:postgresql://db:5432/smartdine'), залишаючись при цьому ізольованими від зовнішньої мережі. Назовні перенаправляються лише порти для вебтрафіку (8081 для основного застосунку та 8080 для сервісу автентифікації).

4.3.2 Налаштування вебсервера Nginx (зворотний проксі-сервер)

Для маршрутизації вхідного трафіку, шифрування з'єднань та захисту від базових мережесих атак перед контейнерами встановлюється вебсервер Nginx, який виконує роль зворотного проксі-сервера.

Конфігурація вебсервера забезпечує декілька критично важливих функцій. По-перше, здійснюється маршрутизація та шифрування трафіку: Nginx приймає запити на порту 443 (HTTPS), розшифровує їх за допомогою сертифікатів Let's Encrypt та перенаправляє до відповідних контейнерів (маршрут '/auth/' — до Keycloak, усі інші — до основного застосунку). По-друге, реалізовано спеціальну підтримку протоколу Server-Sent Events (SSE) через вимкнення буферизації та кешування відповідей для відповідних точок доступу, що гарантує миттєву доставку подій клієнту.

Для підвищення безпеки до кожної відповіді додаються жорсткі заголовки, зокрема механізм HSTS для примусового використання HTTPS та захист від атак типу Clickjacking. Крім того, на рівні вебсервера налаштовано ліміти завантаження медіафайлів високої роздільної здатності та делеговано кешування статичних ресурсів (зображень та маніфестів PWA) на бік браузерів, що суттєво розвантажує обчислювальні потужності сервера.

Обрана стратегія розгортання забезпечує високий рівень безпеки та стійкості. Контейнеризація гарантує ідентичність роботи застосунку на різних середовищах, а використання Nginx як зворотного проксі-сервера дозволило реалізувати єдину точку входу з тонким налаштуванням параметрів передачі даних у реальному часі.

4.3.3 Розгортання у хмарній інфраструктурі Azure

Для продуктивної експлуатації системи SmartDine обрано хмарну платформу Microsoft Azure. Розгортання реалізовано на базі віртуальної машини конфігурації Standard B2ls v2 (2 vCPUs, 4 GiB memory) під управлінням ОС Ubuntu Server, що забезпечує оптимальний баланс між обчислювальною потужністю та вартістю володіння інфраструктурою.

Процес підготовки середовища передбачає конфігурування мережесхем правил безпеки в Azure Network Security Group для обмеження зовнішнього доступу лише необхідними портами вебтрафіку (HTTP/HTTPS) та адміністрування. Безпосередній запуск стека сервісів здійснюється засобами оркестрації Docker Compose, що дозволяє автоматизувати створення ізольованого мережевого сегмента та підключення дискових томів для гарантованого збереження бази даних. Для забезпечення безперебійної роботи та оперативного реагування на зміну навантаження налаштовано моніторинг ключових показників продуктивності (процесора та пам'яті) за допомогою інтегрованих інструментів Azure Monitor. Вибір конфігурації B2ls v2 зумовлений її здатністю підтримувати стабільну роботу системи за умов пікових навантажень, що підтверджено результатами тестування.

ВИСНОВКИ

Результатом виконаної кваліфікаційної роботи стала веб-орієнтована система безконтактного самообслуговування SmartDine, призначена для автоматизації процесу прийому та обробки замовлень у закладах громадського харчування. Система дозволяє клієнтам переглядати цифрове меню, формувати замовлення та здійснювати оплату безпосередньо з мобільного пристрою після сканування QR-коду столу, без встановлення будь-яких застосунків. Персонал закладу отримує замовлення у реальному часі через механізм передачі подій від сервера (Server-Sent Events), що усуває затримки та помилки, характерні для традиційного усного прийому замовлень.

У ході виконання кваліфікаційної роботи було досягнуто таких результатів:

1. Проведено аналіз предметної галузі та існуючих програмних рішень для автоматизації закладів громадського харчування (ChoiceQR, Poster POS, Toast POS, Lightspeed Restaurant). Виявлено їх ключові обмеження: надмірна складність для малого бізнесу, відсутність сповіщень у реальному часі та необхідність встановлення спеціалізованих застосунків. На основі аналізу визначено актуальність розробки власного рішення та сформульовано функціональні й нефункціональні вимоги до системи SmartDine.

2. Розроблено архітектуру системи за тривірневою моделлю з чітким поділом на рівні представлення, бізнес-логіки та доступу до даних. Спроектовано базу даних із 15 таблицями (замовлення, страви, кошик, лояльність, алергени тощо) з урахуванням вимог до цілісності та продуктивності. Побудовано UML-діаграми варіантів використання, послідовностей та компонентів. Додатково спроектовано інтерфейс користувача з пріоритетом для мобільних пристроїв (Mobile-First) із визначенням ключових сценаріїв взаємодії клієнта та персоналу із системою.

3. Реалізовано серверну частину системи на основі Java 25 та Spring Boot із підтримкою віртуальних потоків Project Loom для ефективної обробки масових з'єднань. Впроваджено механізм автентифікації та рольову модель доступу (RBAC) через Keycloak за протоколом OIDC/OAuth 2.0, інтеграцію з платіжним шлюзом LiqPay (з верифікацією підписів зворотних викликів), систему сповіщень для персоналу та модуль лояльності з ідемпотентним нарахуванням бонусних балів. Версіонування схеми бази даних PostgreSQL забезпечено через Flyway-міграції.

4. Реалізовано клієнтський інтерфейс за принципом інтерфейсу, керованого сервером (Server-Driven UI), що забезпечило динамічне оновлення сторінок без використання важких клієнтських JavaScript-фреймворків. Усі компоненти системи взаємодіють із сервером через асинхронні запити та механізм передачі подій від сервера. Додатково клієнтську частину реалізовано у форматі прогресивного вебзастосунку (Progressive Web App, PWA), що дозволяє користувачам встановлювати систему на головний екран смартфона, забезпечує кешування ресурсів та покращує користувацький досвід.

5. Проведено комплексне тестування системи. Модульне тестування перевірило коректність бізнес-логіки сервісного шару. Інтеграційне тестування з реальною базою даних у Docker-контейнері підтвердило цілісність транзакцій і коректність роботи за стандартами ACID. Наскрізне тестування (E2E) автоматизовано перевірило ключові сценарії клієнта та персоналу. Аналіз продуктивності зафіксував високі результати за показниками швидкодії, доступності та SEO. Навантажувальне тестування при навантаженні у 300 одночасних користувачів підтвердило стабільну роботу системи без втрати функціональної працездатності.

Разом з тим, успішна реалізація базового функціоналу відкриває широкі перспективи для подальшого розвитку системи та її масштабування на ринку. Зокрема, оскільки поточна архітектура розрахована на один заклад, пріоритетною

ціллю для переходу до моделі «програмне забезпечення як послуга» (SaaS) є впровадження багатокористувацької архітектури (Multi-tenancy) з відповідною перебудовою бази даних. Для забезпечення надійного горизонтального масштабування потоків передачі подій при зростанні навантаження планується інтеграція зовнішніх брокерів повідомлень. Додатково передбачається розробка повноцінної інтеграції зі спеціалізованими системами управління складом (Inventory Management) для автоматизації процесів списання інгредієнтів.

Поставлені в межах кваліфікаційної роботи завдання виконано в повному обсязі. Розроблена система SmartDine може слугувати основою для подальшого комерційного розвитку та впровадження у реальних закладах громадського харчування

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Бовкун І.В., Шахматов І.О. Визначення вимог до веб-системи для управління безконтактними замовленнями у ресторані. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.432-434.

2. Бовкун І.В. Захист інформації у web-застосунку для безконтактного оформлення замовлень та керування ними у ресторані. Матеріали Всеукраїнської науково-практичної конференції «Цифрова трансформація кібербезпеки», 29 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. К.: ДУІКТ, 2026. С.13-14.

ПЕРЕЛІК ПОСИЛАНЬ

1. National Restaurant Association. State of the Restaurant Industry 2023. Washington: NRA, 2023. URL: <https://restaurant.org/research-and-media/research/research-reports/state-of-the-industry/> (дата звернення: 20.04.2026).
2. Русавська В. А., Кожухівський С. Ю. Цифровізація та інноваційні технології в ресторанному бізнесі: сучасний стан та перспективи розвитку. *Економіка та суспільство*. 2025. Вип. 81. DOI: <https://doi.org/10.32782/2524-0072/2025-81-60>.
3. Parasuraman A., Zeithaml V. A., Berry L. L. SERVQUAL: A multiple-item scale for measuring consumer perceptions of service quality. *Journal of Retailing*. 1988. Vol. 64, No. 1. P. 12–40.
4. Lee S. K., Kim Y. H. How Do Self-Service Kiosks Improve COVID-19 Pandemic Resilience in the Restaurant Industry? *Sustainability*. 2023. Vol. 15, iss. 13. No. 10168. DOI: <https://doi.org/10.3390/su151310168>.
5. W3C Web Accessibility Initiative. Introduction to Web Accessibility. URL: <https://www.w3.org/WAI/fundamentals/accessibility-intro/> (дата звернення: 21.04.2026).
6. Jose N. Event-Driven Architecture in Retail: Real-Time Inventory Synchronization for Omnichannel Retail. *International Journal of Computing and Engineering*. 2025. Vol. 7, no. 16. P. 13–23. DOI: <https://doi.org/10.47941/ijce.3014>.
7. Poster POS: офіційний вебсайт. URL: <https://joinposter.com> (дата звернення: 22.04.2026).
8. ChoiceQR: офіційний вебсайт. URL: <https://choiceqr.com> (дата звернення: 22.04.2026).
9. Toast POS: офіційний вебсайт. URL: <https://pos.toasttab.com> (дата звернення: 22.04.2026).
10. Lightspeed Restaurant: офіційний вебсайт. URL: <https://www.lightspeedhq.com/pos/restaurant> (дата звернення: 22.04.2026).

11. IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications. New York : IEEE, 1998. 37 p.
12. Walls C. Spring in Action. 6th ed. Shelter Island : Manning Publications, 2022. 536 p.
13. Gross D., Stephan C. Hypermedia Systems. Leanpub, 2023. 285 p. URL: <https://hypermedia.systems/> (дата звернення: 23.04.2026).
14. Fette I., Melnikov A. The WebSocket Protocol. RFC 6455. IETF, 2011. URL: <https://www.rfc-editor.org/rfc/rfc6455> (дата звернення: 23.04.2026).
15. Thorgersen S., Silva P. I. Keycloak – Identity and Access Management for Modern Applications. Birmingham : Packt Publishing, 2021. 445 p.
16. Rumbaugh J., Jacobson I., Booch G. The Unified Modeling Language Reference Manual. 2nd ed. Boston : Addison-Wesley, 2004. 721 p.
17. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 533 p.
18. DeBarros A. Practical SQL: A Beginner's Guide to Storytelling with Data. 2nd ed. San Francisco : No Starch Press, 2022. 464 p.
19. Spilca L. Spring Security in Action. 2nd ed. Shelter Island : Manning Publications, 2024. 416 p.
20. OWASP Cross-Site Request Forgery Prevention Cheat Sheet / OWASP Foundation. URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html (дата звернення: 24.04.2026).
21. LiqPay API Documentation / PrivatBank. URL: <https://www.liqpay.ua/documentation> (дата звернення: 24.04.2026).
22. k6 Load Testing Documentation / Grafana Labs. URL: <https://grafana.com/docs/k6/latest/> (дата звернення: 25.04.2026).
23. Matthias K., Kane S. Docker: Up and Running. 3rd ed. Sebastopol : O'Reilly Media, 2023. 394 p.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



Розробка вебзастосунку для безконтактного оформлення замовлень та керування ними у ресторані

Виконав студент 4 курсу
групи ПД-43
Бовкун Ілля Віталійович
Керівник роботи

канд.техн.наук., доц. Довженко Тимур Павлович

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: покращення процесу безконтактного обслуговування клієнтів та замовлення страв у закладах громадського харчування шляхом розробки веб-орієнтованої системи SmartDine з підтримкою сповіщень у реальному часі на основі сучасного стеку технологій.

Об'єкт дослідження: процес безконтактного обслуговування клієнтів у закладах громадського харчування.

Предмет дослідження: програмне забезпечення для автоматизації процесу безконтактного обслуговування клієнтів у ресторанах та кафе.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати стан ресторанної галузі та існуючі програмні аналоги.
2. Сформулювати вимоги до розроблюваного ПЗ та обґрунтувати вибір стеку технологій.
3. Спроектиувати архітектуру, базу даних та інтерфейс користувача.
4. Розробити серверну частину системи та інтегрувати модуль безготівкових онлайн-оплат.
5. Реалізувати адаптивний інтерфейс користувача з динамічним відображенням даних.
6. Провести комплексне тестування розробленого програмного забезпечення та оцінити його стабільність і швидкодію під навантаженням.
7. Здійснити контейнеризацію та розгорнути розроблену систему на хмарній платформі.

3

АНАЛІЗ АНАЛОГІВ

Параметр	Poster POS	ChoiceQR	Toast POS	Lightspeed	SmartDine
Цикл QR -> Оплата	Тільки перегляд	Окремі інтеграції	Вантажний цикл; власні шлюзи	QR відсутній; замовлення на терміналі	Замкнений цикл: QR -> KDS -> LiqPay
Гостьові сесії	Виклик офіціанта або вхід	Частково (залежність від POS-системи)	Обов'язкова авторизація користувача	Тільки через персонал або акаунт	Замовлення та оплата без реєстрації
Оновлення (Real-time)	Polling (затримка 3-5 сек)	Polling (затримка 3-5 сек)	WebSockets (тільки для кухні)	Ручне оновлення сторінки	SSE: миттєве оновлення
Рекомендації	Ручне налаштування «хітів»	Ручні маркетингові ваги	Тільки звіти в адмін-панелі	KPI звіти без авто-рекомендацій	Автоматичні рекомендації за тегами
Безпека (RBAC)	Базові ролі касира/офіціанта	Ролі не передбачені	Вбудовані пропріетарні ролі	Адмін-панель без IAM	Ролі через Keycloak
Впровадження	Від \$50/міс; запуск за 1 день	Комісія %; вручну з POS	Від \$110/міс + залізо (\$800+)	Від \$69/міс; складне налаштування	Низький TCO; швидке Docker-розгортання

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Забезпечення гостьового доступу без реєстрації з можливістю безшовного злиття сесій.
2. Динамічне оновлення статусів замовлень у реальному часі за допомогою SSE.
3. Здійснення безготівкової онлайн-оплати через платіжний шлюз LiqPay.
4. Накопичення бонусів у межах вбудованої системи лояльності (кешбеку).
5. Фільтрування меню й автоматичне рекомендування страв за тегами та алергенами.
6. Автоматизація робочого місця персоналу за допомогою мобільних пристроїв.
7. Керування контентом меню та перегляд аналітики у вебпанелі адміністратора.

Нефункціональні вимоги:

1. Швидке завантаження на мобільних пристроях (Lighthouse > 90).
2. Гарантія цілісності даних відповідно до принципів ACID та ідемпотентність операцій.
3. Рольова модель доступу (RBAC) на базі Keycloak.
4. Висока відмовостійкість (підтримка 300+ одночасних сесій).

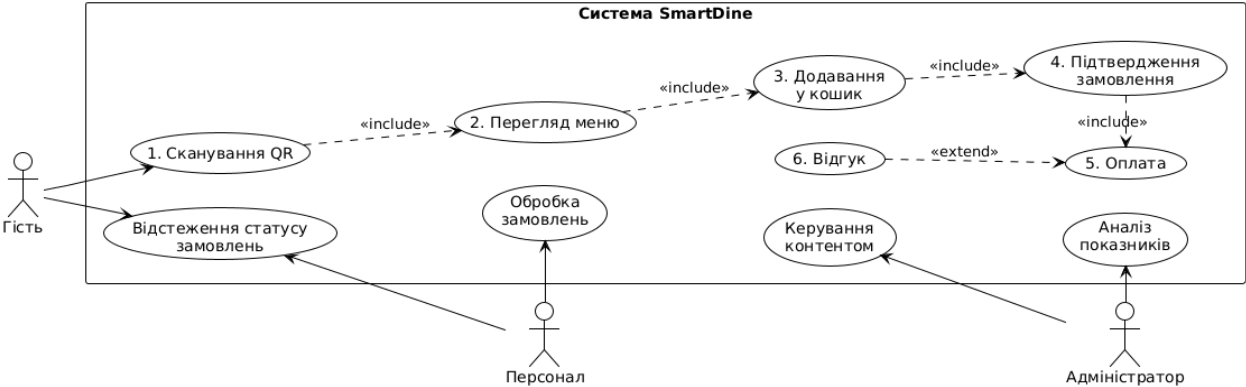
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



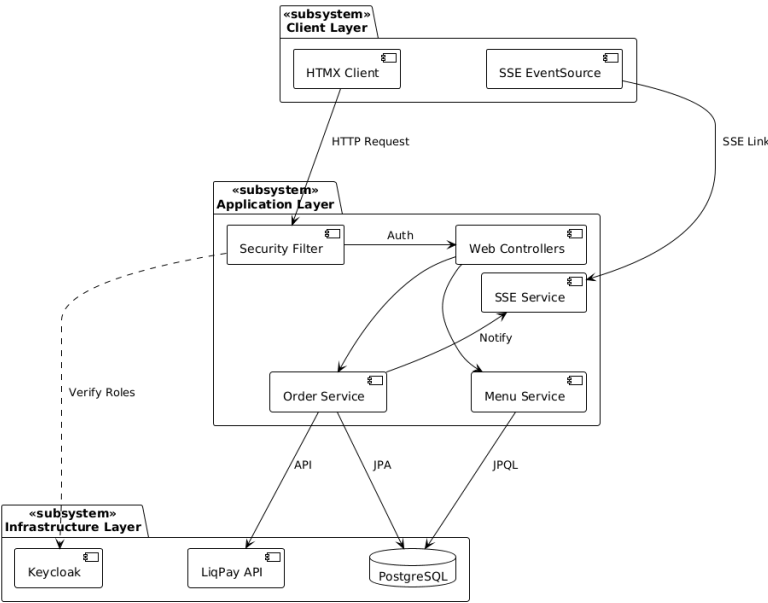
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ СИСТЕМИ



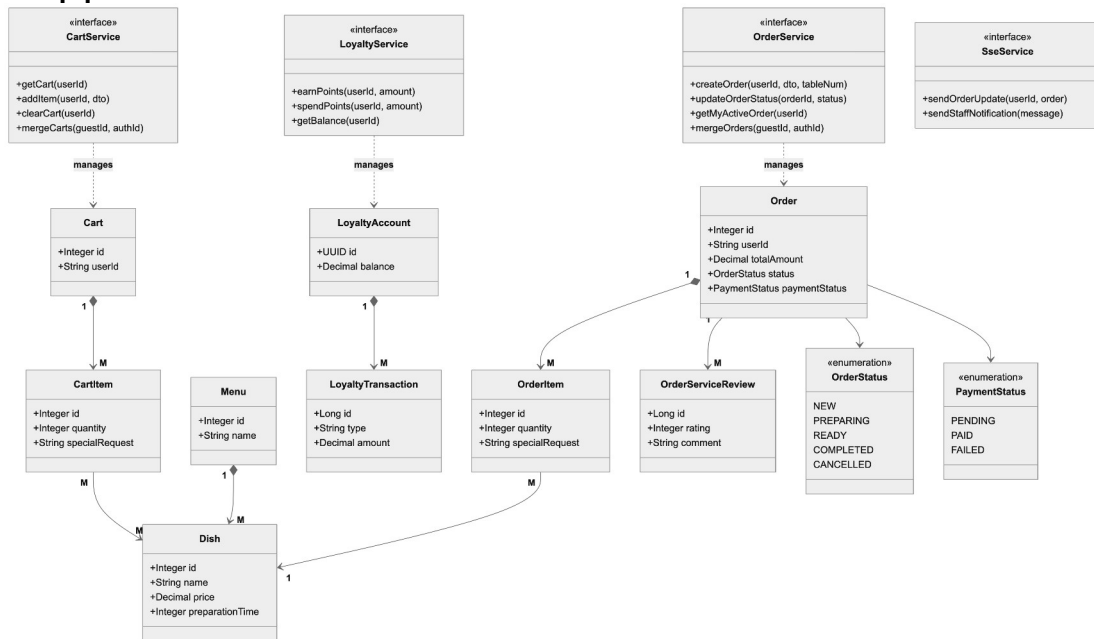
7

ДІАГРАМА КОМПОНЕНТІВ СИСТЕМИ



8

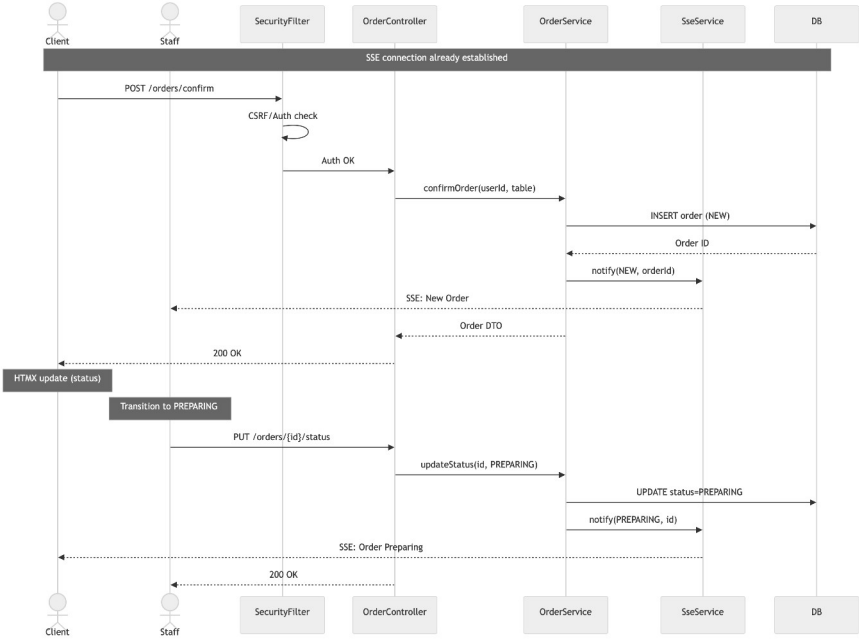
ДІАГРАМА КЛАСІВ БІЗНЕС-ЛОГІКИ СИСТЕМИ



ER-ДІАГРАМА БАЗИ ДАНИХ СИСТЕМИ

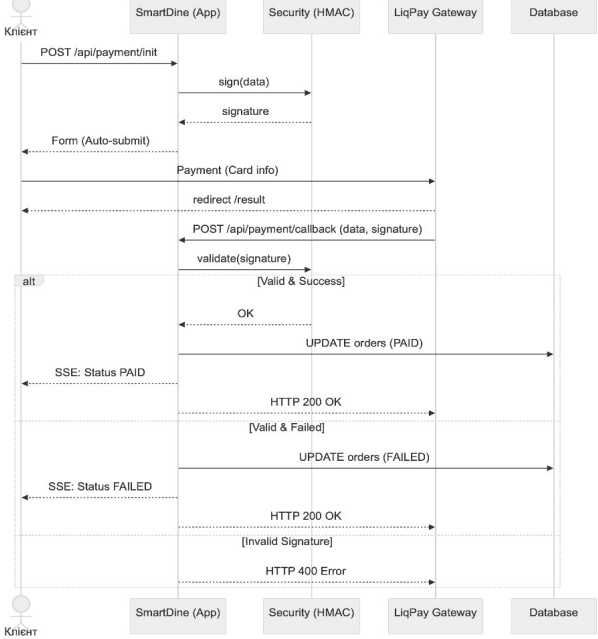


ДІАГРАМА ПОСЛІДОВНОСТІ ОФОРМЛЕННЯ ЗАМОВЛЕННЯ



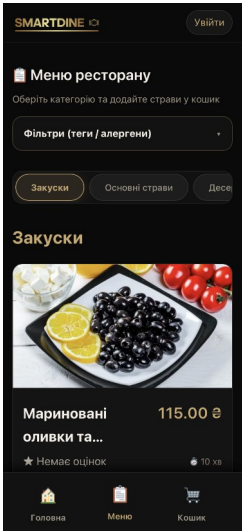
11

ДІАГРАМА ПОСЛІДОВНОСТІ ОПЛАТИ ЗАМОВЛЕННЯ

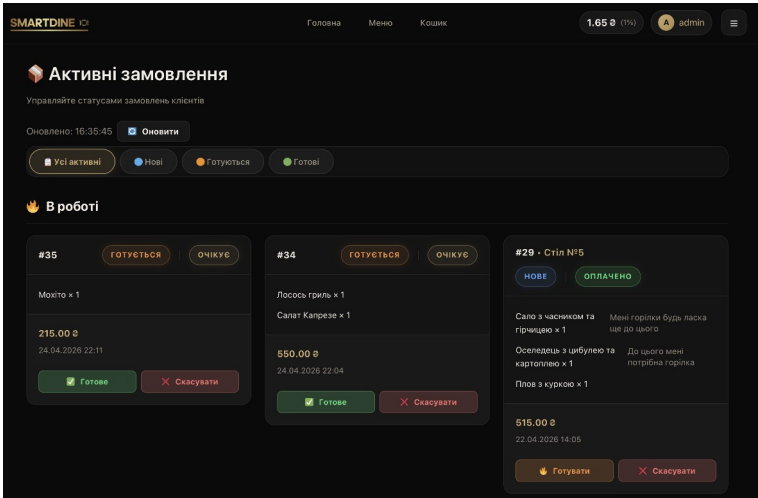


12

ЕКРАННІ ФОРМИ



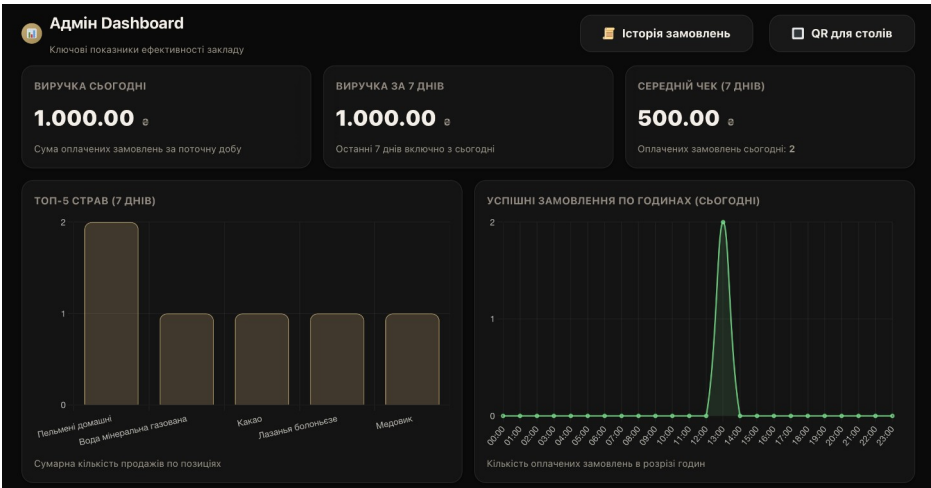
Клієнтське меню



Панель персоналу з динамічною дошкою замовлень

13

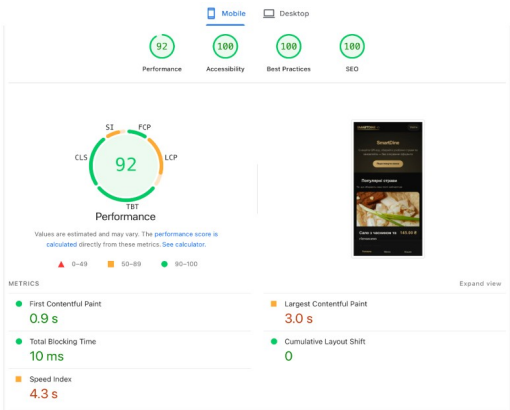
ЕКРАННІ ФОРМИ



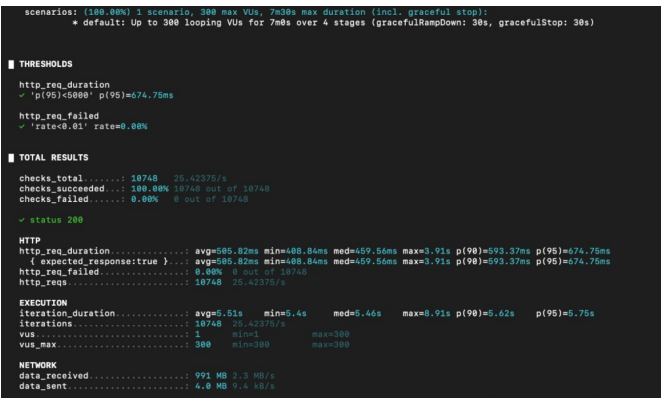
Інформаційна панель адміністратора з графіками аналітики

14

ТЕСТУВАННЯ

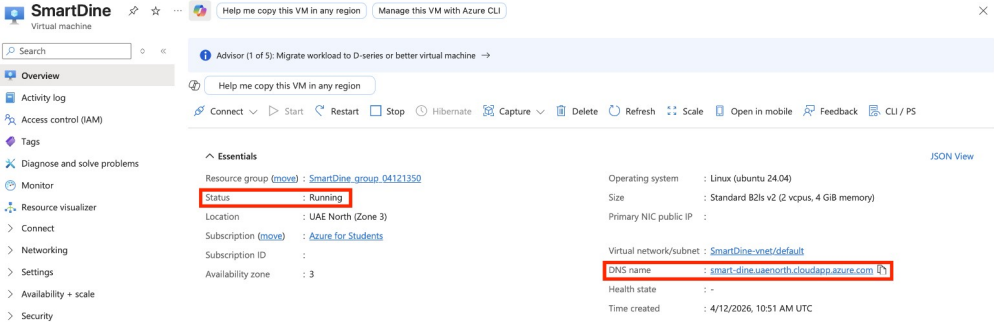


Результати Lighthouse (Mobile)

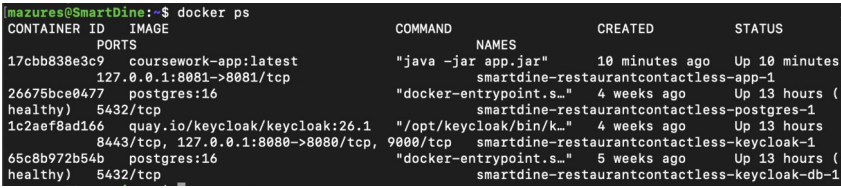


Результати k6 (300 VUs)

РОЗГОРТАННЯ



Оглядове вікно Microsoft Azure VM системи SmartDine



Запущені Docker-контейнери системи на Azure VM

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Бовкун І.В., Шахматов І.О. Визначення вимог до веб-системи для управління безконтактними замовленнями у ресторані. Матеріали ІІ Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.432-434.
2. Бовкун І.В. Захист інформації у web-застосунку для безконтактного оформлення замовлень та керування ними у ресторані. Матеріали Всеукраїнської науково-практичної конференції «Цифрова трансформація кібербезпеки», 29 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. К.: ДУІКТ, 2026. С.13-14.

17

ВИСНОВКИ

1. Проведено аналіз стану ресторанного бізнесу та існуючих аналогів, що виявив їхні критичні обмеження: високу вартість впровадження, затримки оновлення даних та відсутність повної автономії гостя без реєстрації.
2. Сформульовано вимоги до ПЗ та обґрунтовано вибір технологічного стеку на базі Java 25 та Project Loom для стабільної обробки 300+ одночасних сесій на бюджетній хмарній інфраструктурі.
3. Спроектовано архітектуру системи з «нульовим бар'єром входу», структуру реляційної бази даних та концепцію інтерфейсу користувача, що забезпечує миттєвий доступ до сервісу через QR-код без встановлення стороннього ПЗ.
4. Розроблено систему SmartDine із замкненим циклом (QR→KDS→LiqPay) та інтегровано модуль безготівкових онлайн-оплат з ідемпотентністю транзакцій, а також підсистеми персоналізації меню й лояльності.
5. Реалізовано адаптивний інтерфейс користувача з динамічним відображенням даних на базі підходу Server-Driven UI та технології Server-Sent Events, що дозволило досягти швидкодії інтерфейсу на рівні нативних додатків.
6. Проведено комплексне тестування розробленого програмного забезпечення та оцінено його стабільність і швидкість під навантаженням: успішно пройдено 187 тестів, підтверджено роботу під навантаженням (300 VUs) з показником помилок, який наближається до 0% та результатом Lighthouse 92/100.
7. Здійснено контейнеризацію розробленої системи за допомогою Docker та розгорнуто її на хмарній платформі Microsoft Azure, що підтвердило її практичну готовність до експлуатації.

18

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

HmacSignatureServiceImpl.java

```
package com.example.CourseWork.service.security.impl;

import
com.example.CourseWork.service.security.HmacSignature
Service;
import lombok.extern.slf4j.Slf4j;
import
org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.security.InvalidKeyException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;

@Service
@Slf4j
public class HmacSignatureServiceImpl implements
HmacSignatureService {

    private static final String HMAC_ALGO =
"HmacSHA256";
    private final String secretKey;

    public HmacSignatureServiceImpl(@Value("$
{app.security.qr.hmac-secret}") String secretKey) {
        if (secretKey == null || secretKey.isBlank()) {
            log.warn("HMAC secret key is missing. QR codes
will not be properly signed.");
            this.secretKey = "default-unsafe-secret";
        } else {
            this.secretKey = secretKey;
        }
    }

    @Override
    public String signTableNumber(int tableNumber) {
```

```
        return calculateHmac("table=" + tableNumber);
    }

    @Override
    public boolean verifyTableNumber(int tableNumber,
String signature) {
        if (signature == null || signature.isBlank()) {
            return false;
        }
        String expectedSignature = calculateHmac("table=" +
tableNumber);

        return MessageDigest.isEqual(
            expectedSignature.getBytes(StandardCharsets.UT
F_8),
            signature.getBytes(StandardCharsets.UTF_8)
        );
    }

    private String calculateHmac(String data) {
        try {
            Mac mac = Mac.getInstance(HMAC_ALGO);
            SecretKeySpec secretKeySpec = new
SecretKeySpec(
                secretKey.getBytes(StandardCharsets.UTF_8),
                HMAC_ALGO);
            mac.init(secretKeySpec);

            byte[] hmacBytes =
mac.doFinal(data.getBytes(StandardCharsets.UTF_8));

            return
Base64.getUrlEncoder().withoutPadding().encodeToString(
hmacBytes);
        } catch (NoSuchAlgorithmException |
InvalidKeyException e) {
            log.error("Failed to calculate HMAC signature", e);
            throw new IllegalStateException("Failed to
calculate HMAC signature", e);
        }
    }
}
```

LoginSuccessMergeHandler.java

```
package com.example.CourseWork.security;

import com.example.CourseWork.service.cart.CartService;
import
com.example.CourseWork.service.order.OrderService;
import jakarta.servlet.ServletException;
```

```
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.Authentication;
import
org.springframework.security.oauth2.core.oidc.user.OidcUs
er;
```

```
import
org.springframework.security.web.authentication.SavedReq
uestAwareAuthenticationSuccessHandler;
import org.springframework.stereotype.Component;
import java.io.IOException;
```

```
@Component
@RequiredArgsConstructor
public class LoginSuccessMergeHandler extends
SavedRequestAwareAuthenticationSuccessHandler {

    private final CartService cartService;
    private final OrderService orderService;

    @Override
    public void onAuthenticationSuccess(HttpServletRequest
request, HttpServletResponse response,
Authentication authentication) throws
ServletException, IOException {
        HttpSession session = request.getSession(false);
        if (session != null && authentication.getPrincipal()
instanceof OidcUser oidcUser) {
            String guestId = (String)
session.getAttribute(GuestSessionFilter.PREVIOUS_GUE
ST_ID);
            String authId = oidcUser.getSubject();
```

OrderNotifier.java

```
package
com.example.CourseWork.service.order.component;

import com.example.CourseWork.model.OrderStatus;
import
com.example.CourseWork.model.NotificationMessages;
import
com.example.CourseWork.dto.order.OrderResponseDto;
import
com.example.CourseWork.service.push.PushNotificationSe
rvice;
import com.example.CourseWork.service.sse.SseService;
import org.springframework.stereotype.Component;
```

```
@Component
public class OrderNotifier {

    private final SseService sseService;
    private final PushNotificationService
pushNotificationService;

    public OrderNotifier(SseService sseService,
PushNotificationService pushNotificationService) {
        this.sseService = sseService;
        this.pushNotificationService =
pushNotificationService;
    }
```

```
if (guestId != null && authId != null) {
    boolean guestHasActiveOrder =
orderService.getMyActiveOrder(guestId).isPresent();
    boolean authHasActiveOrder =
orderService.getMyActiveOrder(authId).isPresent();
```

```
        if (!guestHasActiveOrder && !
authHasActiveOrder) {
            cartService.mergeCarts(guestId, authId);
        } else {
            cartService.clearCart(authId);
            cartService.clearCart(guestId);
        }
```

```
        orderService.mergeOrders(guestId, authId);
        session.removeAttribute(GuestSessionFilter.PREVIOUS_G
UEST_ID);
    }
}

this.setDefaultTargetUrl("/");
this.setAlwaysUseDefaultTargetUrl(true);
super.onAuthenticationSuccess(request, response,
authentication);
}
```

```
public void notifyUserOfUpdate(String userId,
OrderResponseDto order) {
    if (userId == null) return;
    sseService.sendOrderUpdate(userId, order);

    if (order.getStatus() == OrderStatus.COMPLETED ||
order.getStatus() == OrderStatus.CANCELLED) {
        sseService.sendUserNotification(userId,
NotificationMessages.userOrderFinalized(order.getId(),
order.getStatus()));
    }
}
```

```
public void notifyWaitersAboutNewOrder(Integer
tableNumber, Integer orderId) {

    pushNotificationService.sendNotificationToRole(Notification
Messages.ROLE_WAITER,
NotificationMessages.pushWaiterNewOrder(tableNumber))
;

    pushNotificationService.sendNotificationToRole(Notification
Messages.ROLE_CHEF,
NotificationMessages.pushChefNewOrder(tableNumber));

    sseService.sendStaffNotification(NotificationMessages.staf
fNewOrderCreated(orderId));
```

```

    }

    public void notifyStatusChange(String userId,
    OrderResponseDto order, OrderStatus newStatus) {
        notifyUserOfUpdate(userId, order);
        String statusText =
        NotificationMessages.getLocalizedStatus(newStatus);
        String url = (userId != null &&
        userId.startsWith("GUEST_")) ? "/" : "/orders";

        pushNotificationService.sendNotificationToUser(userId,

        NotificationMessages.pushUserOrderStatus(statusText,
        url));

        if (newStatus == OrderStatus.READY) {

```

LoyaltyServiceImpl.java

```

package com.example.CourseWork.service.loyalty.impl;

import
com.example.CourseWork.model.LoyaltyTransactionType;
import
com.example.CourseWork.exception.BadRequestException
;
import
com.example.CourseWork.exception.ErrorMessages;
import
com.example.CourseWork.exception.InsufficientPointsExc
eption;
import com.example.CourseWork.model.LoyaltyAccount;
import
com.example.CourseWork.model.LoyaltyTransaction;
import
com.example.CourseWork.repository.LoyaltyAccountRepo
sitory;
import
com.example.CourseWork.repository.LoyaltyTransactionR
epository;
import
com.example.CourseWork.repository.OrderRepository;
import
com.example.CourseWork.service.loyalty.LoyaltyService;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Transactional;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.UUID;

@Service
@RequiredArgsConstructor

```

```

pushNotificationService.sendNotificationToRole(NotificationMessages.ROLE_WAITER,

NotificationMessages.pushWaiterOrderReady(order.getTableNumber()));
    }

    boolean reload = order.getStatus() ==
    OrderStatus.COMPLETED || order.getStatus() ==
    OrderStatus.CANCELLED;

    sseService.sendStaffNotification(NotificationMessages.staf
fOrderStatusUpdated(order.getId(), reload));
    }
}

```

public class LoyaltyServiceImpl implements
LoyaltyService {

```

    private static final int MONEY_SCALE = 2;
    private static final BigDecimal RATE_NEW = new
    BigDecimal("0.01");
    private static final BigDecimal RATE_10 = new
    BigDecimal("0.03");
    private static final BigDecimal RATE_100 = new
    BigDecimal("0.05");

    private final LoyaltyAccountRepository
    loyaltyAccountRepository;
    private final LoyaltyTransactionRepository
    loyaltyTransactionRepository;
    private final OrderRepository orderRepository;

    @Override
    @Transactional
    public void earnPoints(UUID userId, BigDecimal
    orderAmount) {
        earnPointsInternal(userId, orderAmount, null);
    }

    @Override
    @Transactional
    public void earnPointsInternal(UUID userId, BigDecimal
    orderAmount, String reference) {
        if (userId == null) {
            throw new
            BadRequestException(ErrorMessages.USER_ID_REQUIRE
            D);
        }
        if (orderAmount == null) {
            throw new
            BadRequestException(ErrorMessages.ORDER_AMOUNT
            _REQUIRED);
        }
    }
}

```

```

        if (orderAmount.compareTo(BigDecimal.ZERO) <= 0) {
            return;
        }

        if (reference != null && !reference.isBlank() &&
            loyaltyTransactionRepository.existsByReference(reference)
        ) {
            return;
        }

        BigDecimal rate = resolveCashbackRate(userId);
        BigDecimal points =
            normalizeMoney(orderAmount.multiply(rate));

        if (points.compareTo(BigDecimal.ZERO) <= 0) {
            return;
        }

        creditPointsInternal(userId, points, reference);
    }

    @Override
    @Transactional
    public void creditPointsInternal(UUID userId,
        BigDecimal points, String reference) {
        if (userId == null) {
            throw new
                BadRequestException(ErrorMessages.USER_ID_REQUIRE
                    D);
        }
        if (points == null) {
            throw new
                BadRequestException(ErrorMessages.POINTS_REQUIRE
                    D);
        }
        BigDecimal toCredit = normalizeMoney(points);
        if (toCredit.compareTo(BigDecimal.ZERO) <= 0) {
            return;
        }
        if (reference != null && !reference.isBlank() &&
            loyaltyTransactionRepository.existsByReference(reference)
        ) {
            return;
        }

        // Pessimistic locking to prevent race conditions on
        // balance update
        LoyaltyAccount account =
            loyaltyAccountRepository.findByIdForUpdate(userId)
                .orElseGet() -> {
                LoyaltyAccount created = new
                    LoyaltyAccount();
                created.setUserId(userId);

                created.setBalance(BigDecimal.ZERO.setScale(MONEY_S
                    CALE, RoundingMode.HALF_UP));
                return loyaltyAccountRepository.save(created);
            };
    }

```

```

    });

    BigDecimal current =
        normalizeMoney(account.getBalance());
    BigDecimal updated =
        normalizeMoney(current.add(toCredit));
    account.setBalance(updated);
    loyaltyAccountRepository.save(account);

    LoyaltyTransaction tx = new LoyaltyTransaction();
    tx.setAccount(account);
    tx.setType(LoyaltyTransactionType.EARN);
    tx.setAmount(toCredit);
    tx.setBalanceAfter(updated);
    tx.setReference(reference != null && !
        reference.isBlank() ? reference : null);
    loyaltyTransactionRepository.save(tx);
}

@Override
@Transactional
public void spendPoints(UUID userId, BigDecimal
    amount) {
    spendPointsInternal(userId, amount, null);
}

@Override
@Transactional
public void spendPointsInternal(UUID userId,
    BigDecimal amount, String reference) {
    if (userId == null) {
        throw new
            BadRequestException(ErrorMessages.USER_ID_REQUIRE
                D);
    }
    if (amount == null) {
        throw new
            BadRequestException(ErrorMessages.AMOUNT_REQUIRE
                D);
    }

    BigDecimal toSpend = normalizeMoney(amount);
    if (toSpend.compareTo(BigDecimal.ZERO) <= 0) {
        return;
    }

    if (reference != null && !reference.isBlank() &&
        loyaltyTransactionRepository.existsByReference(reference)
    ) {
        return;
    }

    LoyaltyAccount account =
        loyaltyAccountRepository.findByIdForUpdate(userId)
            .orElseGet() -> {
            LoyaltyAccount created = new
                LoyaltyAccount();
            created.setUserId(userId);

```

```

created.setBalance(BigDecimal.ZERO.setScale(MONEY_SCALE, RoundingMode.HALF_UP));
return loyaltyAccountRepository.save(created);
});

BigDecimal current =
normalizeMoney(account.getBalance());
if (current.compareTo(toSpend) < 0) {
    throw new
InsufficientPointsException("Недостатньо балів.
Поточний баланс: " + current);
}

BigDecimal updated =
normalizeMoney(current.subtract(toSpend));
account.setBalance(updated);
loyaltyAccountRepository.save(account);

LoyaltyTransaction tx = new LoyaltyTransaction();
tx.setAccount(account);
tx.setType(LoyaltyTransactionType.SPEND);
tx.setAmount(toSpend);
tx.setBalanceAfter(updated);
tx.setReference(reference != null && !
reference.isBlank() ? reference : null);
loyaltyTransactionRepository.save(tx);
}

@Override
@Transactional(readOnly = true)
public BigDecimal getBalance(UUID userId) {
    if (userId == null) {
        throw new
BadRequestException(ErrorMessages.USER_ID_REQUIRED);
    }
    return
loyaltyAccountRepository.findById(userId)

```

OrderCreateServiceImpl.java

```

package com.example.CourseWork.service.order.impl;

import com.example.CourseWork.model.OrderStatus;
import com.example.CourseWork.model.PaymentStatus;
import com.example.CourseWork.dto.order.OrderItemDto;
import
com.example.CourseWork.dto.order.OrderRequestDto;
import
com.example.CourseWork.dto.order.OrderResponseDto;
import
com.example.CourseWork.exception.NotFoundException;
import com.example.CourseWork.model.Dish;
import com.example.CourseWork.model.Order;
import com.example.CourseWork.model.OrderItem;

```

```

        .map(a -> normalizeMoney(a.getBalance()))
        .orElse(BigDecimal.ZERO.setScale(MONEY_SCALE, RoundingMode.HALF_UP));
    }

    private BigDecimal normalizeMoney(BigDecimal value)
    {
        BigDecimal v = value == null ? BigDecimal.ZERO :
value;
        return v.setScale(MONEY_SCALE,
RoundingMode.HALF_UP);
    }

    @Override
    @Transactional(readOnly = true)
    public Long getSuccessfulOrdersCount(UUID userId) {
        if (userId == null) {
            throw new
BadRequestException(ErrorMessages.USER_ID_REQUIRED);
        }
        return
orderRepository.countSuccessfulOrdersByUserId(userId.toString());
    }

    @Override
    @Transactional(readOnly = true)
    public BigDecimal resolveCashbackRate(UUID userId) {
        Long count = getSuccessfulOrdersCount(userId);
        if (count != null && count >= 100) {
            return RATE_100;
        }
        if (count != null && count >= 10) {
            return RATE_10;
        }
        return RATE_NEW;
    }
}

```

```

import
com.example.CourseWork.repository.DishRepository;
import
com.example.CourseWork.repository.OrderRepository;
import
com.example.CourseWork.service.order.OrderCreateService;
import
com.example.CourseWork.service.order.component.Order
Notifier;
import
com.example.CourseWork.service.order.component.OrderTotalCalculator;
import com.example.CourseWork.mapper.OrderMapper;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

```

```
import
org.springframework.transaction.annotation.Transactional;
```

```
import java.time.OffsetDateTime;
import java.util.ArrayList;
```

```
@Service
@RequiredArgsConstructor
public class OrderCreateServiceImpl implements
OrderCreateService {

    private final OrderRepository orderRepository;
    private final DishRepository dishRepository;
    private final OrderTotalCalculator orderTotalCalculator;
    private final OrderNotifier orderNotifier;
    private final OrderMapper orderMapper;
```

```
@Override
@Transactional
    public OrderResponseDto createOrder(String userId,
OrderRequestDto dto, Integer tableNumber) {
        Order order = new Order();
        order.setUserId(userId);
        order.setTableNumber(tableNumber);
        order.setCreatedAt(OffsetDateTime.now());
        order.setStatus(OrderStatus.NEW);
        order.setPaymentStatus(PaymentStatus.PENDING);
```

CartController.java

```
package com.example.CourseWork.controller;
```

```
import
com.example.CourseWork.dto.cart.CartItemRequestDto;
import com.example.CourseWork.service.cart.CartService;
import
com.example.CourseWork.service.security.CurrentUserIde
ntity;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
```

```
@Controller
@RequestMapping("/cart")
@RequiredArgsConstructor
public class CartController {
```

```
    private final CartService cartService;
    private final CurrentUserIdentity identity;
```

SseServiceImpl.java

```
package com.example.CourseWork.service.sse.impl;
```

```
order.setItems(new ArrayList<>());
```

```
for (OrderItemDto itemDto : dto.getItems()) {
    Dish dish =
dishRepository.findById(itemDto.getDishId())
        .orElseThrow(() -> new
NotFoundException("Dish not found"));

    OrderItem item = new OrderItem();
    item.setDish(dish);
    item.setQuantity(itemDto.getQuantity());

    item.setSpecialRequest(itemDto.getSpecialRequest());
    item.setOrder(order);
    order.getItems().add(item);
}
```

```
orderTotalCalculator.recalculateAndSetTotal(order);
Order saved = orderRepository.save(order);
```

```
orderNotifier.notifyWaitersAboutNewOrder(tableNumber,
saved.getId());
```

```
        return orderMapper.toResponseDto(saved);
    }
}
```

```
@PostMapping("/add")
public String addItem(@ModelAttribute
CartItemRequestDto request, Model model) {
    String userId = identity.getIdOrCreateUserId();
    cartService.addItemToCart(userId, request);
```

```
    model.addAttribute("cart",
cartService.getCart(userId));
    return "fragments/cart-sidebar :: cart-content";
}
```

```
@DeleteMapping("/remove/{itemId}")
public String removeItem(@PathVariable Integer itemId,
Model model) {
    String userId = identity.getIdOrCreateUserId();
    cartService.removeItem(userId, itemId);

    model.addAttribute("cart",
cartService.getCart(userId));
    return "fragments/cart-sidebar :: cart-content";
}
}
```

```
import com.example.CourseWork.service.sse.SseService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;
```

```

import
org.springframework.web.servlet.mvc.method.annotation.S
seEmitter;
import java.io.IOException;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

@Service
@Slf4j
public class SseServiceImpl implements SseService {

    private final Map<String, SseEmitter> emitters = new
ConcurrentHashMap<>();

    @Override
    public SseEmitter createEmitter(String userId) {
        SseEmitter emitter = new SseEmitter(1800_000L);

        emitter.onCompletion(() -> emitters.remove(userId));
        emitter.onTimeout(() -> emitters.remove(userId));
        emitter.onError((ex) -> emitters.remove(userId));
    }

```

sw.js

```

const CACHE_NAME = 'smartdine-v1.0';
const ASSETS_TO_CACHE = [
    '/css/base.css', '/js/ui.js', '/manifest.json',
    '/icons/icon-192.png'
];

self.addEventListener('install', event => {
    event.waitUntil(
        caches.open(CACHE_NAME).then(cache =>
            cache.addAll(ASSETS_TO_CACHE))
    );
});

self.addEventListener('push', event => {
    const data = event.data ? event.data.json() : { title:
'SmartDine', body: 'Нове сповіщення' };

```

```

        emitters.put(userId, emitter);
        return emitter;
    }

    @Override
    public void sendOrderUpdate(String userId, Object data)
    {
        SseEmitter emitter = emitters.get(userId);
        if (emitter != null) {
            try {
                emitter.send(SseEmitter.event()
                    .name("order-update")
                    .data(data));
            } catch (IOException e) {
                emitters.remove(userId);
            }
        }
    }
}

```

```

const options = {
    body: data.body,
    icon: '/icons/android-chrome-192x192.png',
    data: { url: data.url || '/' }
};

event.waitUntil(
    self.registration.showNotification(data.title, options)
);

self.addEventListener('notificationclick', event => {
    event.notification.close();
    event.waitUntil(
        clients.openWindow(event.notification.data.url)
    );
});

```