ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Цифрова система оптимізації процесів прийому, обробки та управління замовленнями у ресторанному бізнесі на основі Python»

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

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

_____ Дмитро ХАВА
(підпис)

Виконав: здобувач вищої освіти гр. ТЦР-44
Дмитро ХАВА

Керівник: Олексій АНАНЧЕНКО
старший викладач

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технології цифрового розвитку
_____ Вікторія Жебка
« ____ » _____ 2026 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Хаві Дмитру Віталійовичу

1. Тема кваліфікаційної роботи: «Цифрова система оптимізації процесів прийому, обробки та управління замовленнями у ресторанному бізнесі на основі Python»

керівник кваліфікаційної роботи Ананченко Олексій Євгенович, затверджені наказом Державного університету інформаційно-комунікаційних технологій від 20 лютого 2026 р. №51.

2. Строк подання кваліфікаційної роботи «18» травня 2026 р.

3. Вихідні дані до кваліфікаційної роботи: офіційна документація фреймворків FastAPI, SQLAlchemy та PostgreSQL; наукові джерела з автоматизації ресторанного бізнесу; матеріали з проєктування REST API та WebSocket-систем; приклади реалізації POS-систем у сфері HoReCa.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної галузі та постановка задачі.
2. Проєктування інформаційної системи.
3. Розробка та реалізація системи.
4. Тестування, охорона праці та економічне обґрунтування.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів (POS-систем).
2. Вимоги до програмного забезпечення.
3. Загальна архітектура системи.
4. Діаграма варіантів використання.
5. ER-діаграма бази даних.
6. Діаграма класів.
7. Екранні форми.
8. Результати тестування.

6. Дата видачі завдання «20» лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	02.02.–09.02.2026	Виконано
2	Аналіз предметної галузі та існуючих програмних рішень	10.02.–18.02.2026	Виконано
3	Проєктування архітектури системи та бази даних	19.02.–04.03.2026	Виконано
4	Розробка серверної частини (FastAPI, REST API, WebSocket)	05.03.–31.03.2026	Виконано
5	Розробка клієнтської частини та інтерфейсу	01.04.–15.04.2026	Виконано
6	Тестування системи	16.04.–25.04.2026	Виконано
7	Оформлення роботи: вступ, висновки, реферат	26.04.–05.05.2026	Виконано
8	Розробка демонстраційних матеріалів	06.05.–13.05.2026	Виконано
9	Попередній захист роботи	14.05.–28.05.2026	Виконано

Здобувач вищої освіти _____ Дмитро ХАВА

(підпис)

Керівник кваліфікаційної роботи _____ Олексій АНАНЧЕНКО

(підпис)

РЕФЕРАТ

Пояснювальна записка: 60 с., 7 рис., 8 табл., 39 джерел, 7 додатків.

Об'єкт дослідження – процеси прийому, обробки та управління замовленнями у закладах ресторанного господарства формату fast-casual і коктейль-барів, що працюють за моделлю самообслуговування гостя через QR-код.

Предмет дослідження – методи й програмні засоби автоматизації цих процесів на основі сучасного веб-стеку Python/FastAPI з акцентом на самообслуговування, кастомізацію страв, спліт рахунку, AI-рекомендації та трекінг замовлення.

Для досягнення мети розв'язано такі завдання:

1. Проаналізовано бізнес-процеси типового ресторану та переваги моделі самообслуговування над класичною.
2. Виконано порівняльний огляд комерційних POS-систем (Toast, Square, Lightspeed, Poster POS).
3. Сформульовано функціональні і нефункціональні вимоги до системи.
4. Спроєктовано трирівневу архітектуру, базу даних із 10 сутностей та інтерфейс користувача.
5. Реалізовано серверну частину на FastAPI: 35 REST-ендпоінтів, JWT-автентифікація, ролі, гостьовий доступ через payment_token.
6. Додано канали реального часу через WebSocket: для кухні, ранерів і окремий канал на кожне замовлення для трекінгу гостем.
7. Реалізовано модуль конструктора страви: видалення інгредієнтів, додавання топінгів з надбавками, поле алергій.
8. Реалізовано спліт рахунку: оплата всього одним платежем, рівний поділ на N осіб, по-стравах між гостями за столом.
9. Реалізовано модуль AI-рекомендацій (cross-sell/upsell) з rule-based mock-логікою, готовою до заміни на ML-модель.
10. Реалізовано сторінку відстеження замовлення з прогрес-баром на 5 кроків і WebSocket-оновленням у реальному часі.

11. Реалізовано календарне бронювання столів із виявленням накладок і автоматичним «no-show».

12. Побудовано аналітичний модуль із графіками виручки (Chart.js) та експортом у Excel.

13. Розроблено модуль PDF-чеків і журнал дій (audit log).

14. Виконано модульне, інтеграційне і навантажувальне тестування системи.

15. Сформовано розділи з охорони праці й економічного обґрунтування.

Методи дослідження: системний і структурно-функціональний аналіз; об'єктно-орієнтоване проєктування; UML-моделювання (Use Case, Class, Sequence, State); ER-моделювання та нормалізація реляційної бази до 3NF; юніт- та інтеграційне тестування на pytest; навантажувальне тестування інструментом locust.

Результати. Розроблено повнофункціональну систему: 10 таблиць БД, 35 REST-ендпоінтів, 4 WebSocket-канали (кухня, ранер, адмін і окремо на кожне замовлення для трекінгу гостем), фоновий планувальник, 8 ролевих і гостьових веб-сторінок, темна/світла тема, генератор PDF-чеків з підтримкою кирилиці, експорт звітів у багатолистовий Excel, журнал дій (audit log). Реалізовано чотири нові модулі: конструктор страви (модальне вікно з модифікаторами і алергіями), AI-рекомендації (rule-based mock з готовим інтерфейсом під ML), спліт рахунку (один платіж / рівний поділ / по стравах), сторінка відстеження замовлення з прогрес-баром на 5 кроків і WebSocket-оновленням. За результатами навантажувального тесту середній час відгуку – 78 мс при 50 одночасних користувачах. Експерименти показали скорочення часу оформлення замовлення з 3 хв 15 с (паперовий блокнот) до 42 с – це майже у 4,6 раза швидше.

Практичне значення. Систему доцільно впроваджувати у закладах формату fast-casual, коктейль-барах і пекарнях, що працюють зі швидким обігом столів. Модель самообслуговування дозволяє скоротити витрати на персонал (офіціанти переходять у роль ранерів) і значно збільшити пропускну здатність зали.

КЛЮЧОВІ СЛОВА: РЕСТОРАН, ЦИФРОВА СИСТЕМА, SAAS, ТАРИФНІ ПЛАНИ, САМООБСЛУГОВУВАННЯ, QR-ЗАМОВЛЕННЯ, СПЛІТ РАХУНКУ, AI-РЕКОМЕНДАЦІЇ, КАСТОМІЗАЦІЯ СТРАВИ, PYTHON, FASTAPI, SQLALCHEMY, WEBSOCKET, REST API, JWT, БРОНЮВАННЯ, АНАЛІТИКА.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface – програмний інтерфейс.

CRUD – Create / Read / Update / Delete – базові операції з даними.

CSS – Cascading Style Sheets – мова стилів для HTML.

ER – Entity-Relationship – сутність-зв'язок.

HTTP – HyperText Transfer Protocol – основний протокол вебу.

JWT – JSON Web Token – підписаний токен авторизації.

ORM – Object-Relational Mapping – об'єктно-реляційне відображення.

POS – Point Of Sale – каса/термінал у місці продажу.

REST – Representational State Transfer – архітектурний стиль API.

SQL – Structured Query Language – мова запитів до БД.

UML – Unified Modeling Language – мова моделювання.

WS – WebSocket – двонаправлений канал поверх HTTP.

БД – База даних.

ІС – Інформаційна система.

ПЗ – Програмне забезпечення.

ПК – Персональний комп'ютер.

СКБД – Система керування базами даних.

ТЗ – Технічне завдання.

AI – Artificial Intelligence (штучний інтелект).

ВСТУП

В Україні на 2024 рік налічується понад 25 тисяч закладів ресторанного господарства, і це число щороку зростає, попри всі виклики останніх років. Конкуренція між ними висока, маржинальність помірна, а гість, який чекає 20 хвилин на замовлення, з ймовірністю близько чверті більше не повертається – таку статистику регулярно озвучує Restaurants Association of Ukraine.

У великих мережах і дорогих ресторанах питання автоматизації давно закрите: вони купують комерційні POS-системи на кшталт Toast чи Square. Натомість у тих самих 25 тисячах закладів – переважно малих кафе, піцерій, бістро – картина зовсім інша. Близько 45% продовжують працювати з паперовим блокнотом і олівцем, а в кращому випадку – з застарілою касовою програмою, що жодним чином не пов'язана з кухнею. Така схема породжує знайому всім картину: офіціант біжить між столиком і кухнею з папірцем, кухар не знає, що ще не подано, керівник у кінці зміни вручну зводить виручку у Excel.

Проблема не у тому, що бракує програмних рішень – їх десятки. Проблема у тому, що вартість і складність наявних систем непропорційно велика для маленького закладу. Власник кав'ярні на 10 столів не готовий сплачувати 35–60 тис. грн ліцензії плюс щорічну підтримку, особливо коли половина функцій (відеоспостереження, складський облік на сотні позицій, інтеграції з банками) йому просто не потрібна. Звідси – стійкий запит на просту, функціонально достатню і легку у впровадженні систему.

Актуальність дипломної роботи саме у цій ніші: створити цифрову систему, яка робить найважливіше – швидке оформлення замовлення, миттєву передачу його на кухню, відстеження готовності страв, розрахунок із чеком, бронювання столів і базову аналітику – і яка при цьому розгортається будь-де (від домашнього сервера до хмари) і модифікується силами однієї людини.

У ході розробки концепція закладу була суттєво переглянута. Замість класичної моделі «офіціант приймає замовлення» було обрано модель

самообслуговування: гість сканує QR-код на столі, потрапляє у браузер на гостьову сторінку, формує замовлення сам, кастомізує страви через модальне вікно, отримує AI-рекомендації під свій кошик, оплачує одним платежем або ділить рахунок із сусідами і відстежує готовність у режимі реального часу. Офіціанти переходять у роль ранерів – лише забирають готові страви на кухні і несуть їх на стіл. Така архітектура зменшує штат, прискорює обіг столів і добре пасує до формату fast-casual, коктейль-барів і пекарень.

У роботи є й окремий соціальний вимір. За даними міжнародних опитувань (Pew Research, 2023; YouGov, 2024) близько 40% людей віком 18–28 років зізнаються, що уникають закладів, де обов'язково треба «домовлятися» з офіціантом – особливо коли відвідувач сам, без компанії, або просто не у настрої вступати у живу комунікацію. Хтось через сором, хтось через тривожність, хтось – принципово, бо звик усе замовляти у застосунку. Парадоксально, але саме ця аудиторія приносить ресторанам непропорційно великий чек: швидкі замовлення на винос, кави з ноутбуком на кілька годин, пізні вечері. Запропонована система прямо адресує цю проблему: гість сам обирає, скільки хоче спілкуватися з персоналом – від «нуль слів, оплачую через Apple Pay» до «викликаю офіціанта кнопкою », коли потрібна жива консультація.

Друга важлива складова – економічна: при правильно налаштованій QR-системі заклад може скоротити кількість офіціантів приблизно вдвічі. Класична функція «прийняти замовлення» вже виконує сама система; персонал залишається у ролі ранерів – розносять страви і тримають у руках можливість оформити замовлення для гостя, який того потребує. На пікові зміни одного ранера достатньо на 6–8 столів замість стандартних 3–4 у класичному форматі. Це особливо актуально для українського ринку 2024–2026 років в умовах хронічного дефіциту лінійного персоналу.

Метою роботи є проєктування і реалізація такої системи мовою Python із використанням сучасного веб-стеку: фреймворку FastAPI, ORM SQLAlchemy 2.0, шаблонізатора Jinja2 і канального протоколу WebSocket. Результат – повнофункціональний застосунок із веб-інтерфейсом для гостя (без логіну, доступ

за QR і токеном) і для трьох службових ролей (адміністратор, кухар, ранер), а також REST/WS API для майбутньої інтеграції з мобільними клієнтами.

Для досягнення мети необхідно розв'язати такі завдання:

1.Дослідити предметну область і описати ключові бізнес-процеси.

2.Порівняти існуючі POS-системи за вартістю, функціоналом і кастомізацією.

3. Сформулювати функціональні й нефункціональні вимоги.

4.Спроєктувати архітектуру, базу даних і UML-діаграми системи.

5.Реалізувати серверну частину з REST API, WebSocket і фоновими задачами.

6.Розробити веб-інтерфейс для трьох ролей із підтримкою світлої й темної теми.

7.Реалізувати модулі PDF-чеків, аналітики, експорту в Excel і аудиту.

8.Провести багаторівневе тестування системи.

9.Сформувати розділи з охорони праці й економічного обґрунтування.

Об'єктом дослідження є інформаційні процеси прийому, обробки і управління замовленнями у ресторані. Предметом – методи й засоби їхньої автоматизації на сучасному Python-стеку.

Наукова новизна полягає у поєднанні в одній невеликій системі одночасного моніторингу кухні через WebSocket, календарного бронювання з автоматичним відмиканням no-show, фіскального чеку у PDF та повноцінної аналітики з графіками й експортом у Excel – функцій, які зазвичай розкидані по різних інструментах.

Практичне значення – готова до впровадження система, що скорочує час оформлення замовлення у понад 4 рази порівняно з паперовим блокнотом і у близько 2 разів – порівняно зі застарілими POS, при собівартості розробки на порядок меншій за комерційну ліцензію.

Апробація результатів дослідження. Результати роботи також подано на Міжнародну науково-практичну конференцію «Digital Transformation, AI and Cybersecurity: Shaping a Trusted Digital Future» (DTAIC 2026, Стамбул, Туреччина, 17–18 червня 2026 р.).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальна характеристика ресторанного бізнесу як предметної області

Ресторан зручно розглядати як невелику соціотехнічну систему. В ній одночасно взаємодіють люди (гості, офіціанти, кухарі, адміністратор), фізичні об'єкти (столи, кухонне обладнання, каса) і нематеріальні потоки – інформація про замовлення, його статус, оплату. Чим більше людей і потоків, тим вища ціна навіть невеликої помилки у комунікації.

Середньостатистичний міський заклад у будній день обслуговує 150–400 гостей, з піками у 12–14 і 19–21 годину. Він має 20–50 столів, 2–6 офіціантів у зміні, 1–3 кухарів, 1 адміністратора. Меню – 70–150 позицій. Ці цифри добре корелюють із даними галузевих звітів і визначають масштаб задач, які повинна розв'язувати автоматизована система.

Типові слабкі місця ручного ведення замовлень:

– Втрати часу на пересування з блокнотом між залом і кухнею.

– Помилки у переписуванні: «карбонара без бекону» легко перетворюється на «карбонара» – і гість отримує не те.

– Дублювання даних: одне й те саме замовлення вписується у блокнот, фіскальний журнал і книгу обліку.

– Відсутність онлайн-статистики: керівник бачить виручку лише наприкінці дня.

– Втрата паперових записів – а з ними і доказової бази, якщо виникає суперечка з гостем.

– Складність контролю продуктивності окремих офіціантів.

Автоматизація знімає більшість цих проблем. За даними галузевих досліджень, заклади, що перейшли на цифрові системи, повідомляють про скорочення часу обслуговування на 20–35%, зменшення помилок у 4–6 разів,

прозорий контроль витрат і повторюваність гостей – у середньому +12% протягом перших трьох місяців.

1.2 Бізнес-процеси прийому і обробки замовлень

Перш ніж писати код, варто чітко описати, що саме відбувається у ресторані «з боку даних». Шість процесів, які повторюються щодня:

– Розміщення гостя – адміністратор (або хост) проводить гостя за столик, фіксує кількість осіб, інколи – побажання щодо місця.

– Формування замовлення – офіціант приймає вибір страв, узгоджує особливості (без цибулі, прожарка стейка), фіксує позиції.

– Передача на кухню – позиції одразу повинні з'явитися перед кухарем у вигляді списку «що готувати у якому пріоритеті».

– Обслуговування страв – кухар оновлює статуси (готується / готова), офіціант знає, що пора нести гостю.

– Розрахунок – гість просить рахунок, офіціант формує чек, приймає оплату готівкою / карткою / онлайн.

– Аналіз – у кінці дня (або у будь-який момент) керівник дивиться виручку, ТОП страв, рейтинг офіціантів.

Кожен із цих процесів має свого ініціатора, своїх «акторів» і вхідні-вихідні дані. У термінах майбутньої системи вони стануть відповідними REST-ендпоінтами і WebSocket-подіями.

1.3 Аналіз існуючих програмних рішень

Для порівняння обрано чотири провідні західні POS-системи, які представлені на українському ринку (через локальних інтеграторів) або у міжнародному сегменті: Toast (США), Square для ресторанів (США), Lightspeed Restaurant (Канада) і українська Poster POS. Усі вони – гідні продукти, перевірені

часом, проте націлені передусім на середні й великі заклади. Російські системи (iiko, r_keeper) до огляду не включаємо принципово – як з міркувань безпеки, так і з законодавчих обмежень на роботу з російським ПЗ в Україні. Зведений порівняльний аналіз наведено у Таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз POS-систем

Критерій	Toast (США)	Square (США)	Lightspeed (Канада)	Poster (Україна)	Розроблена система
Цільові заклади	середні/великі	малі/середні	середні/великі	малі/середні	малі/середні
Вартість, USD/міс	$ 69+	$ 0–60	$ 69+	$ 30+	≈ $ 25 (Basic) / 65 (Pro)
Локалізація укр.	ні	ні	ні	так	так
Кастомізація коду	немає	немає	обмежена	обмежена	повна (open source)
WebSocket / KDS	так	обмежено	так	ні	так
Спліт рахунку	так	так	так	обмежено	так (3 режими)
AI-рекомендації	обмежено	ні	обмежено	ні	так
Розгортання локально	ні (хмара)	ні (хмара)	ні (хмара)	хмара	локально/хмара

Висновок із таблиці: західні комерційні POS – потужні, але дорогі у місячній підписці і повністю хмарні (без можливості локального розгортання). Українська Poster – доступніша, проте її функціонал обмежений базовим самообслуговуванням без AI і складного спліта. Власна розробка займає нішу між цими двома: повний функціонал західних систем + локальне розгортання + повний контроль над кодом за рахунком тарифної моделі Basic/Pro/Enterprise.

1.4 Обґрунтування вибору технологій

Стек технологій підбирався за трьома критеріями: швидкість розробки, якість екосистеми (бібліотеки, документація, спільнота) і простота розгортання. У результаті:

1.Python 3.11 – мова з низьким порогом входу, чудовою екосистемою для веб-розробки і одночасно сильною у науковій сфері.

2.FastAPI – сучасний асинхронний фреймворк зі швидкодією на рівні Node.js і автогенерованою OpenAPI-документацією. У бенчмарках TechEmpower видає до 21 000 RPS проти 5 000 RPS у Flask.

3.SQLAlchemy 2.0 – найзріліший Python-ORM, з типізованим Mapped-API і прозорим переходом між SQLite (розробка) і PostgreSQL (продакшн).

4.Pydantic 2 – валідація даних на основі типів. Стандарт де-факто для FastAPI.

5.Jinja2 + ванільний JS – серверні шаблони замість важкого SPA-фреймворка. Це робить розгортання у малому закладі тривіальним: один процес uvicorn замість чотирьох контейнерів.

6.WebSocket з модуля fastapi – двонаправлений канал для оновлень кухні в реальному часі.

7.ReportLab – генерація PDF-чеків з підтримкою кирилиці.

8.openpyxl – багатолистові Excel-звіти.

9.Chart.js (на клієнті) – інтерактивні графіки виручки.

10.APScheduler – фонові задачі, зокрема автозвільнення «no-show» бронювань.

Свідома відмова від важких фронтенд-фреймворків – це не консерватизм, а прагматизм. Сторінка офіціанта і так оновлюється переважно через JSON-fetch і WebSocket, а Jinja2 знімає необхідність у білдер-системах і npm. Адміністратору малого закладу простіше розгорнути та обслуговувати один Python-сервіс, аніж пов'язку Node + бекенд.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Загальна архітектура системи

Систему побудовано за класичною трирівневою клієнт-серверною моделлю. Розділення на рівні допомагає окремо змінювати інтерфейс, бізнес-логіку та схему зберігання, не зачіпаючи інших шарів.

– Рівень представлення – серверні Jinja2-шаблони з невеликою кількістю JavaScript на сторінці. Спілкування з API – через fetch, оновлення кухні – через WebSocket.

– Рівень бізнес-логіки – FastAPI-сервер. Тут живуть валідація даних (Pydantic), маршрути REST, перевірка прав (require_role), CRUD-операції, генерація чеків і звітів, фоновий планувальник APScheduler.

– Рівень даних – реляційна БД. У режимі розробки використано SQLite, у продакшні той самий код працює на PostgreSQL без правок (завдяки SQLAlchemy).

Рис. 2.1 Архітектурна схема системи

Поверх HTTP/WebSocket між клієнтом і сервером циркулюють JSON-повідомлення. Це дозволяє у майбутньому під'єднати мобільний клієнт або кур'єрський сервіс тим самим API, без жодних спеціальних адаптерів.

2.2 Діаграма варіантів використання (Use Case)

Систему використовують три актори. Адміністратор володіє повним доступом: веде меню й столи, керує персоналом, переглядає аналітику. Офіціант працює зі столами і замовленнями, проводить розрахунки, бачить готовність страв. Кухар бачить виключно панель кухні і змінює статуси приготування.

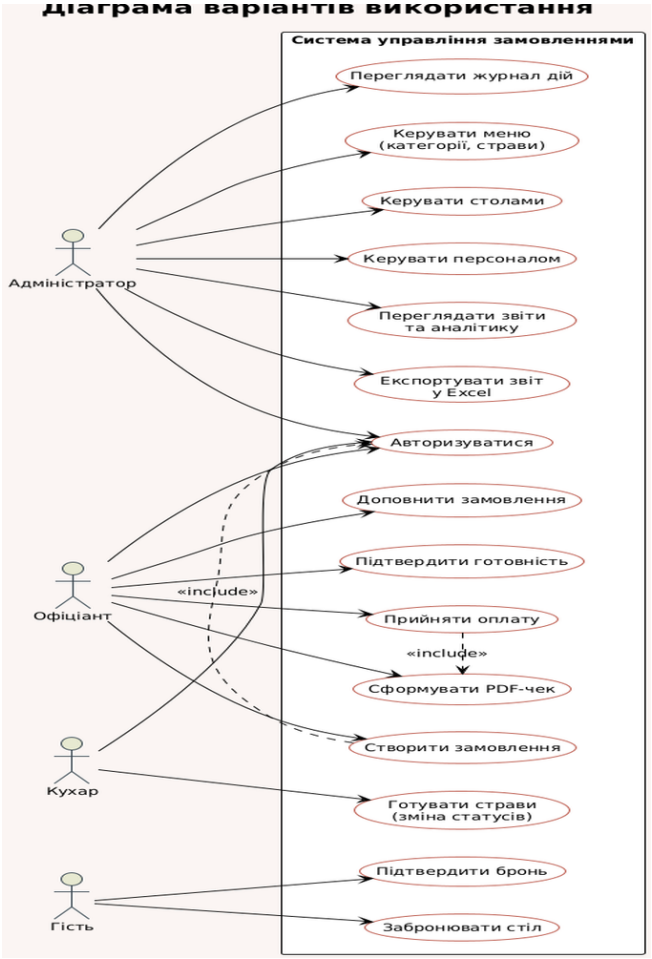


Рис. 2.2 UML-діаграма варіантів використання

Розмежування акторів і ролей буквально перенесено у код через декоратор require_role(*roles). Він повертає FastAPI-залежність, що не пускає користувача без потрібної ролі.

2.3 Проєктування бази даних (ER-модель)

Дев'ять сутностей системи перелічено у Таблиці 2.1. Усі вони перебувають у третій нормальній формі (3NF), що мінімізує дублювання та аномалії оновлення.

Таблиця 2.1

Сутності системи

Сутність	**Призначення**
User	Користувач системи (адмін, офіціант, кухар)
Category	Категорія меню
Dish	Страва меню
Table	Стіл у залі
Order	Замовлення гостя
OrderItem	Окрема позиція замовлення
Payment	Платіж за замовлення (можливо кілька на спліт)
SplitPart	Окрема частина рахунку при спліті
Reservation	Бронювання столу
AuditLog	Журнал ключових дій

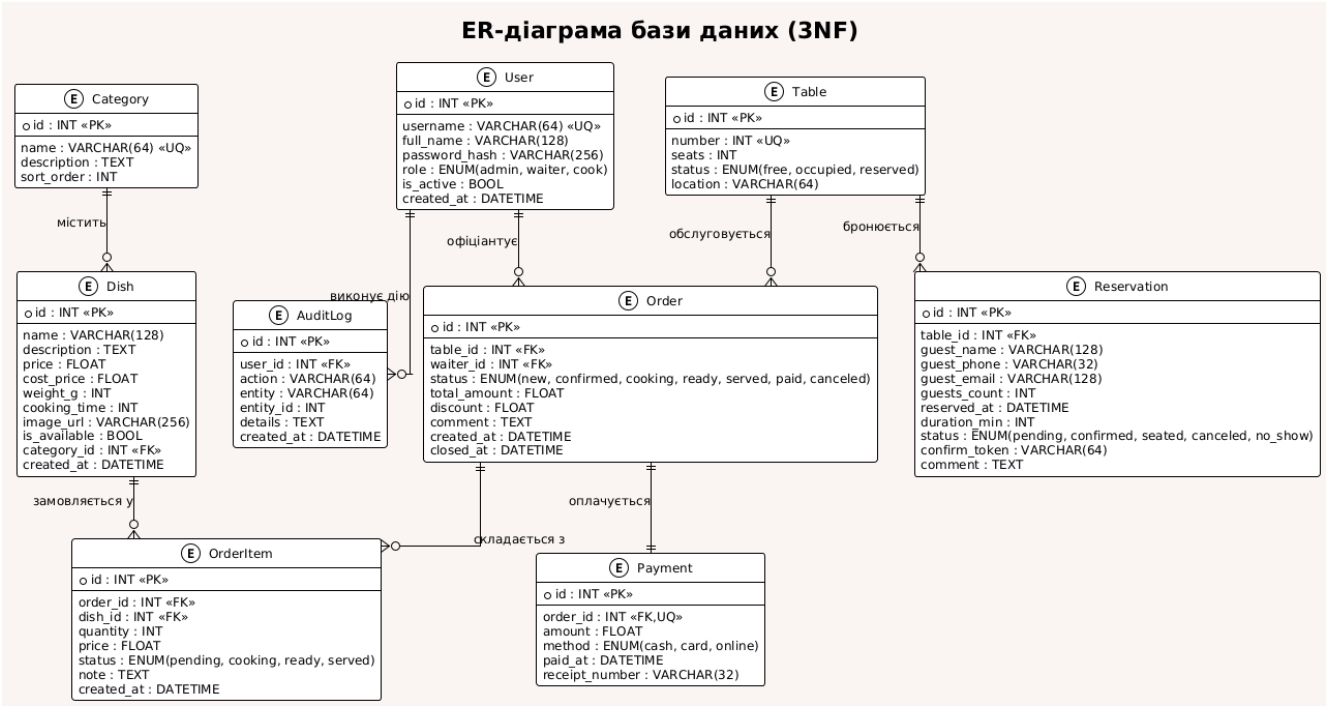


Рис. 2.3 ER-діаграма бази даних

Зв'язки переважно «один-до-багатьох»: одна категорія – багато страв; один стіл – багато замовлень; одне замовлення – багато позицій; один офіціант – багато замовлень. Винятки – Order ↔ Payment (1:1) та Reservation ↔ Table (1:M). Сурогатні цілочисельні id, унікальні індекси на username і receipt_number прискорюють пошук.

Структуру ключових таблиць показано у Таблицях 2.2 і 2.3.

Таблиця 2.2

Структура таблиці orders

Поле	**Тип**	**Опис**
id	INTEGER PK	Сурогатний первинний ключ
table_id	FK tables.id	Стіл, до якого належить замовлення
waiter_id	FK users.id	Офіціант – автор замовлення
status	ENUM	new / confirmed / cooking / ready / served / paid / canceled
total_amount	FLOAT	Підсумкова сума з урахуванням знижки
discount	FLOAT	Знижка у відсотках (0–100)
comment	TEXT	Побажання гостя
created_at	DATETIME	Момент створення (індексоване)
closed_at	DATETIME	Момент оплати/скасування (індексоване)

Таблиця 2.3

Структура таблиці order_items

Поле	**Тип**	**Опис**
id	INTEGER PK	Сурогатний первинний ключ
order_id	FK orders.id	Замовлення, до якого належить позиція
dish_id	FK dishes.id	Страва
quantity	INTEGER	Кількість порцій
price	FLOAT	Ціна на момент замовлення
status	ENUM	pending / cooking / ready / served
note	TEXT	Особливість позиції (без цибулі тощо)
created_at	DATETIME	Момент додавання

2.4 Діаграма класів

Декларативні моделі SQLAlchemy 2.0 одночасно слугують і ORM-сутностями, і класами Python: на них висять методи розрахунку (наприклад, calc_total для замовлення з урахуванням знижки) та зв'язки до інших сутностей. Це уникає дублювання логіки між «об'єктом» і «таблицею».

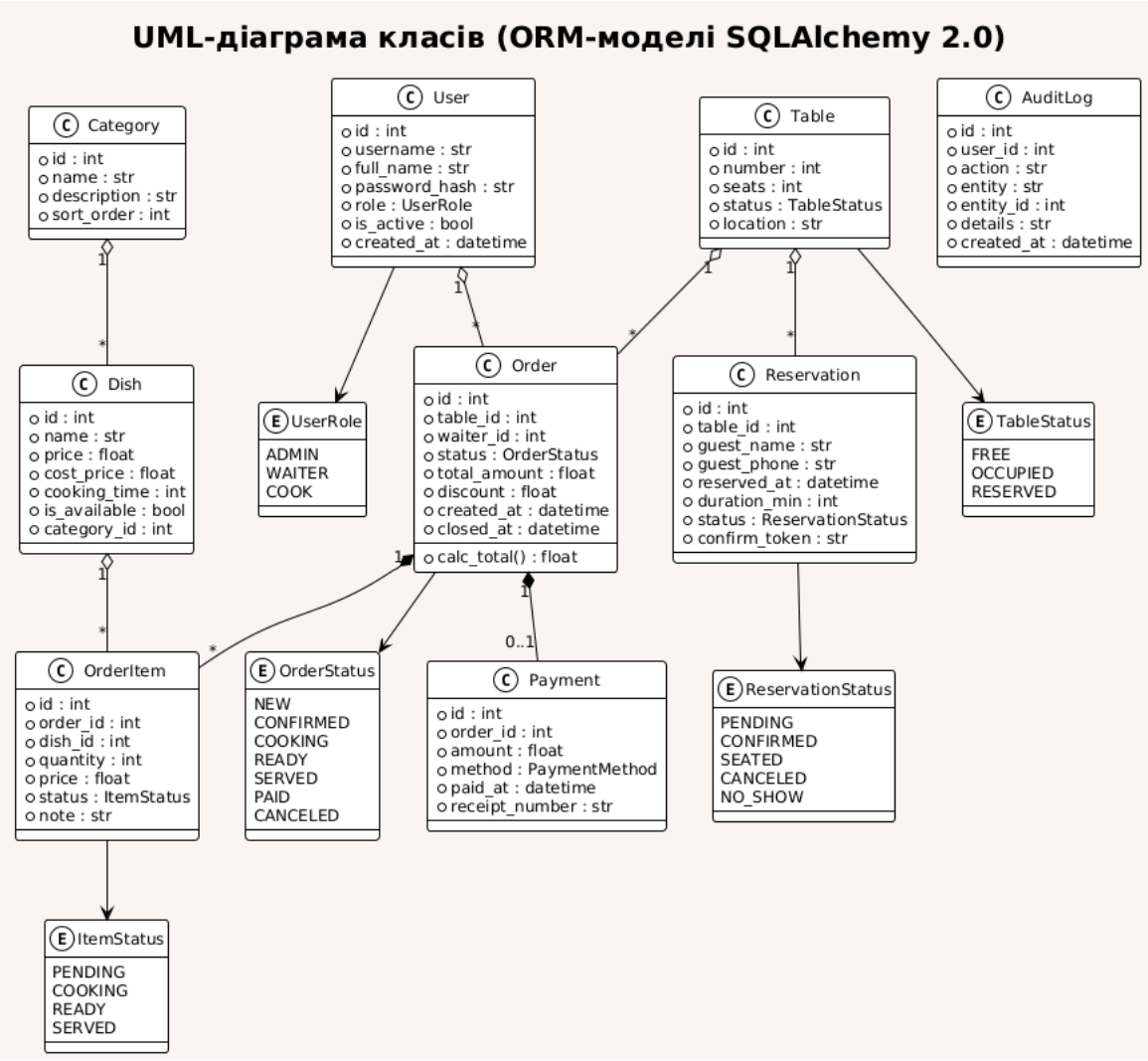


Рис. 2.4 Діаграма класів системи

2.5 Діаграма послідовності: створення замовлення

Розглянемо найчастіший сценарій – оформлення нового замовлення. Він складається з 8 кроків:

– Офіціант авторизовано (JWT-токен у localStorage браузера).

– Браузер запитає список столів (/api/tables) і доступних страв (/api/menu/dishes).

– Офіціант натискає «Відправити на кухню».

– Клієнт надсилає POST /api/orders/ з масивом позицій.

– Сервер валідовує дані (Pydantic), перевіряє доступність страв, рахує суму.

– Створюються Order та OrderItem-и однією транзакцією.

– Стіл переходить у статус OCCUPIED.

– Сервер транслює подію order.created у канал /ws/kitchen – кухня бачить нове замовлення миттєво.

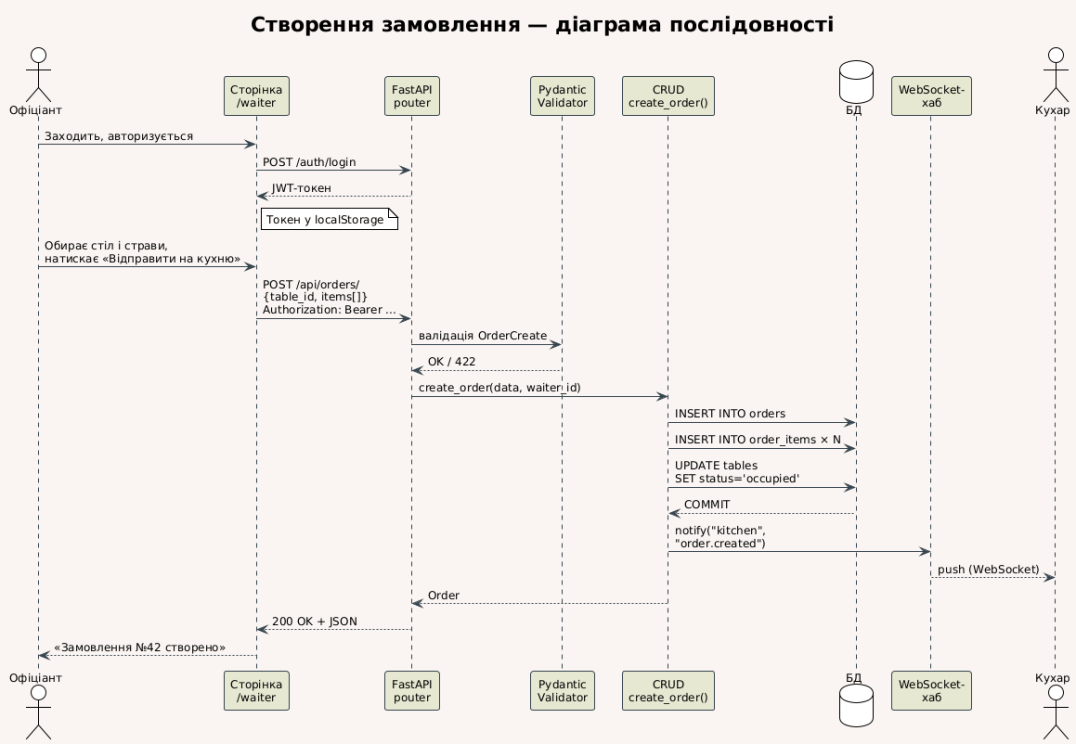


Рис. 2.5 Діаграма послідовності створення замовлення

2.6 Діаграма станів замовлення

Замовлення проходить через шість можливих станів. Перехід «нове → підтверджене → готується → готове → подано → оплачено» – щасливий шлях. У будь-який момент до оплати можливе скасування (CANCELED), що звільняє стіл.

Рис. 2.6 Діаграма станів замовлення

2.7 Проєктування інтерфейсу користувача

Інтерфейс розрахований передусім на планшет і ноутбук – це найпоширеніше робоче залізо у залі. Принципи дизайну:

– Мінімалізм: на сторінці лише те, що потрібне саме в цьому контексті.

– Великі області натискання (≥44 px) – щоб працювати пальцем без стилуса.

– Кольорове кодування статусів столів і страв.

– Миттєвий зворотний зв'язок: тост-нотифікація після кожної дії.

– Темна тема – для вечірньої зміни в умовах низького освітлення.

Кольорова гамма (теракота #B85042, бежевий #FAF5F2, шавлія #A7BEAE, графіт #36454F) дає теплий «ресторанний» відтінок і добре контрастує. Шрифт – Segoe UI (системний на більшості пристроїв).

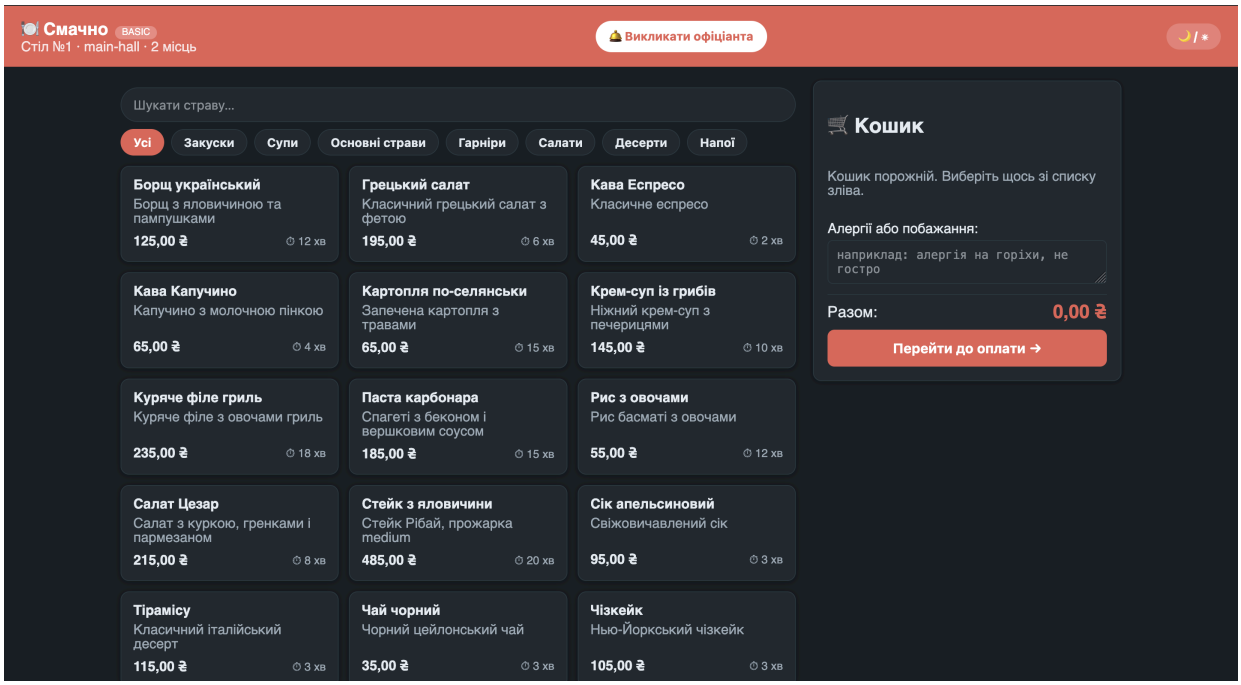


Рис. 2.7 Макет робочого місця офіціанта

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Структура програмного проєкту

Каркас проєкту дотримується принципу «один модуль – одна відповідальність».

```
restaurant_system/
├── app/
│   ├── main.py        # точка входу + lifespan-планувальник
│   ├── config.py      # налаштування й шляхи
│   ├── database.py    # підключення до БД, get_db
│   ├── models.py      # ORM-моделі (10 сутностей)
│   ├── schemas.py     # Pydantic-схеми + валідатори
│   ├── crud.py        # бізнес-логіка
│   ├── auth.py        # JWT, bcrypt, ролі
│   ├── utils.py       # текстовий чек, PDF-чек, експорт у Excel
│   ├── realtime.py    # WebSocket-хаб з підпискою на канали
│   ├── scheduler.py   # APScheduler – звільнення no-show
│   ├── init_db.py     # ініціалізація БД і seed-даних
│   ├── routers/       # auth, menu, orders, tables, reports, users, websocket, web
│   ├── templates/     # 7 Jinja2-шаблонів
│   └── static/        # CSS зі світлою/темною темами
├── tests/             # pytest
├── requirements.txt
└── README.md
```

Такий поділ полегшує орієнтування для будь-кого, хто бачить проєкт уперше: хочеш додати новий REST-ендпоінт – йдеш у routers/; нова бізнес-операція – у crud.py; нова валідація – у schemas.py.

3.2 Реалізація моделей даних

Усі моделі описано в одному файлі app/models.py за допомогою декларативного API SQLAlchemy 2.0. Типізація через Mapped[...] дає підказки IDE й допомагає ловити помилки на етапі написання, а не виконання.

```
class Order(Base):
    __tablename__ = "orders"
    id: Mapped[int] = mapped_column(primary_key=True)
    table_id: Mapped[int] = mapped_column(ForeignKey("tables.id"))
    waiter_id: Mapped[int] = mapped_column(ForeignKey("users.id"))
    status: Mapped[OrderStatus] = mapped_column(
        SAEnum(OrderStatus), default=OrderStatus.NEW)
    total_amount: Mapped[float] = mapped_column(Float, default=0.0)
    discount: Mapped[float] = mapped_column(Float, default=0.0)
    items: Mapped[List["OrderItem"]] = relationship(
        back_populates="order", cascade="all, delete-orphan")
    def calc_total(self) -> float:
        subtotal = sum(it.price * it.quantity for it in self.items)
        return round(subtotal * (1 – self.discount / 100), 2)
```

Каскад all, delete-orphan гарантує, що при видаленні замовлення (наприклад, тестового) автоматично зникнуть і його позиції – ніяких «осиротілих» рядків. ENUM-поля винесено у окремі класи (OrderStatus, ItemStatus, TableStatus, ReservationStatus, PaymentMethod) – це робить статуси безпечними у використанні і самодокументованими.

3.3 Реалізація бізнес-логіки (CRUD)

Уся логіка операцій з даними зосереджена в app/crud.py. Це звільняє маршрути від «розмазаного» SQL і робить логіку легко тестованою – її можна викликати у юніт-тестах, передавши тестову Session.

Найбільш насичена функція – створення замовлення. Вона вирішує одразу кілька задач: створює Order, послідовно додає позиції, перевіряє доступність кожної страви, рахує суму, переводить стіл у OCCUPIED і повідомляє кухню через WebSocket. Усе у одній транзакції – або повністю, або зовсім ні.

```
def create_order(db, data, waiter_id):
    order = models.Order(table_id=data.table_id, waiter_id=waiter_id,
                         comment=data.comment)
    db.add(order); db.flush()
    total = 0.0
    for it in data.items:
        dish = db.get(models.Dish, it.dish_id)
        if not dish or not dish.is_available:
            raise ValueError(f"Страва {it.dish_id} недоступна")
        db.add(models.OrderItem(order_id=order.id, dish_id=dish.id,
                                quantity=it.quantity, price=dish.price,
                                note=it.note))
        total += dish.price * it.quantity
    order.total_amount = round(total, 2)
    table = db.get(models.Table, data.table_id)
    if table: table.status = models.TableStatus.OCCUPIED
    db.commit(); db.refresh(order)
    notify("kitchen", "order.created",
           {"order_id": order.id, "table_id": order.table_id})
    return order
```

Звіт за період і ТОП-10 страв обчислюються одним SQL-запитом із GROUP BY/ORDER BY/LIMIT – без жодних циклів у Python. Це і швидко (БД оптимізує групування), і не залежить від обсягу таблиці.

3.4 Реалізація REST API

REST API оголошено в декларативному стилі FastAPI. Маршрути розкидано по вісьмох файлах у routers/, кожен – навколо однієї підсистеми (auth, users, menu, tables, orders, reports, websocket, web). Усього – 25 ендпоінтів, що автоматично документуються в OpenAPI на /docs.

Таблиця 3.1

Перелік основних REST/WS ендпоінтів

Метод	**Шлях**	**Призначення**
POST	/auth/login	Вхід, видача JWT
GET	/api/menu/categories	Список категорій
POST	/api/menu/categories	Створити категорію (admin)
GET	/api/menu/dishes?q=&category_id=	Пошук/фільтр страв
POST	/api/menu/dishes	Додати страву (admin)
PATCH	/api/menu/dishes/{id}	Редагувати страву (admin)
DELETE	/api/menu/dishes/{id}	Видалити страву (admin)
GET	/api/tables/	Список столів
POST	/api/tables/reservations	Створити бронь
GET	/api/tables/reservations?day=	Брони за днем
GET	/api/tables/reservations/confirm/{t}	Підтвердження по токену
POST	/api/orders/	Створити замовлення
PATCH	/api/orders/{id}/items	Додати страви до замовлення
PATCH	/api/orders/{id}/discount	Застосувати знижку
PATCH	/api/orders/{id}/status	Змінити статус замовлення
PATCH	/api/orders/items/{id}/status	Змінити статус позиції
GET	/api/orders/kitchen/board	Активні позиції для кухаря
POST	/api/orders/payment	Оплатити замовлення
GET	/api/orders/{id}/receipt.pdf	PDF-чек
GET	/api/orders/{id}/receipt.txt	Текстовий чек
GET	/api/reports/period	Звіт за період
GET	/api/reports/period.xlsx	Звіт у форматі Excel
GET	/api/reports/audit	Журнал дій
WS	/ws/kitchen, /ws/waiter, /ws/admin	Канали реального часу

Статус-коди дотримано прямо: 200 – успіх, 400 – погані вхідні, 401 – неавторизовано, 403 – заборонено за роллю, 404 – не знайдено, 409 – конфлікт (наприклад, накладка бронювань), 422 – провалена валідація.

3.5 Реалізація автентифікації і авторизації

Безпека – окрема дисципліна, тож вона зосереджена в одному файлі app/auth.py і ґрунтується на трьох столах: bcrypt-хеш паролів, JWT-токени і функція require_role.

– Паролі ніколи не зберігаються у відкритому вигляді. У моделі User поле password_hash тримає bcrypt-хеш зі вбудованою сіллю. Перевірка – pwd_context.verify().

– Після вдалого логіну сервер генерує JWT-токен (HS256, час життя – 8 годин – щоб охоплювати робочу зміну).

– Усі захищені ендпоінти оголошують Depends(get_current_user) або Depends(require_role(...)). Перший просто розкодовує токен, другий додатково перевіряє, що поточна роль входить у дозволений список.

```
def require_role(*allowed_roles):
    def role_checker(current = Depends(get_current_user)):
        if current.role not in allowed_roles:
            raise HTTPException(403, "Недостатньо прав")
        return current
    return role_checker

@router.post("/api/menu/dishes",
             dependencies=[Depends(require_role(UserRole.ADMIN))])
def add_dish(...): ...
```

Захист від SQL-ін'єкцій реалізовано «безкоштовно» – усі запити йдуть через ORM, який параметризує значення. Жодних ручних рядкових f-string у SQL.

3.6 Реальний час: WebSocket-хаб

Полінг кухні «що там новенького?» кожні 5 секунд – підхід застарілий: він генерує зайве навантаження і дає затримку до 5 с. Замість нього додано WebSocket-хаб (app/realtime.py). Він тримає словник {канал: множина сокетів} і вміє розсилати JSON-повідомлення усім підписникам каналу.

```
class ChannelHub:
    def __init__(self):
        self._channels = {}
        self._lock = asyncio.Lock()
    async def broadcast(self, channel, payload):
        async with self._lock:
            sockets = list(self._channels.get(channel, ()))
        for ws in sockets:
            try: await ws.send_text(json.dumps(payload, default=str))
            except Exception: ...  # відключити мертвий сокет
```

У CRUD-функціях замість безпосередньої взаємодії із сокетами просто викликається notify(channel, event, data). Фасад приймає синхронний код, але запускає розсилку як завдання у поточному event-loop, не блокуючи транзакцію БД.

У браузері кухар відкриває /kitchen – і миттєво бачить нові замовлення. Якщо WS зникає, відразу спрацьовує резервний полінг раз на 10 секунд – система не «зависає».

3.7 Бронювання і фоновий планувальник

Бронювання – окрема невелика підсистема. Її «душа» – функція reserve_table: перш ніж створити запис, вона шукає всі активні брони того ж стола й перевіряє, чи перетинаються часові вікна. Якщо так – повертає 409 Conflict із поясненням;

якщо ні – створює бронь, генерує confirm_token (для посилання-підтвердження у email/SMS) і ставить стіл у RESERVED.

Часом гість не приходить. Раніше у системах це означало «навіки заблокований стіл» – поки адміністратор не помітить. Тепер цим займається фоновий планувальник APScheduler:

```
def setup_scheduler():
    sched = AsyncIOScheduler(timezone="Europe/Kyiv")
    sched.add_job(_release_no_show, "interval", minutes=5,
                  id="release-no-show", replace_existing=True)
    return sched
```

Кожні 5 хвилин _release_no_show перебирає брони, чий час давно минув і які не SEATED, переводить їх у NO_SHOW і звільняє стіл. APScheduler керується lifespan-контекстом застосунку – стартує разом з ним і коректно зупиняється при вимкненні.

3.8 PDF-чек, експорт у Excel і журнал дій

Три «службові» модулі, які роблять систему по-справжньому корисною:

– PDF-чек (utils.receipt_pdf_bytes). Будується ReportLab у форматі A6 – стандарт термопринтерів. Окрема процедура _register_font() шукає шрифт із підтримкою кирилиці (DejaVuSans, Arial Unicode) – у різних ОС вони лежать у різних місцях, тож проходимо по списку кандидатів і реєструємо перший знайдений.

– Експорт у Excel (utils.period_report_xlsx). Один XLSX-файл, п'ять листів: «Підсумок», «По днях», «Офіціанти», «Категорії», «ТОП страв». Для кожного листа задано ширину колонок і заливку шапки – щоб документ виглядав готовим до подачі бухгалтеру.

– Журнал дій (audit log). Таблиця AuditLog з полями user_id, action, entity, entity_id, details, created_at. Хелпер log_action() викликається у кожній значущій

операції CRUD: створення/видалення страви, користувача, оплата. Адміністратор бачить останні 30 записів на сторінці /admin.

3.9 Реалізація клієнтської частини

Клієнт – серверно-рендерений вебзастосунок із невеликою кількістю JavaScript (приблизно 350 рядків на 5 сторінок). Жодного фреймворка, жодного npm.

– base.html – спільна шапка, темна/світла тема, тост-нотифікації.

– index.html – головна з картками для трьох ролей + бронювання.

– login.html – форма входу з валідацією.

– waiter.html – сітка столів, пошук по меню, фільтр за категорією, динамічна сума замовлення, WebSocket-нотифікації про готові страви.

– kitchen.html – панель кухні з оновленням у реальному часі, fallback-полінг, кольорове кодування статусів.

– reservations.html – календарна сітка зайнятості за днем (грид CSS), форма нової броні з валідацією телефону.

– admin.html – графіки виручки (Chart.js), рейтинг офіціантів, ТОП страв, журнал дій, експорт у Excel.

Темну/світлу тему перемикає одна кнопка у шапці. Тема зберігається у localStorage, тому при наступному відкритті закладу не потрібно вибирати наново. Усі стилі спираються на CSS-змінні, а зміна теми – це лише зміна значень --bg, --card, --text у html[data-theme="dark"].

3.10 Конструктор страви

При тапі по картці страви відкривається модальне вікно – компонент, у якого є три блоки: «Прибрати інгредієнти», «Додати топінги», «Особлива примітка». Чекбокси прибирання («Без цибулі», «Без часнику») не змінюють ціну. Топінги –

навпаки, мають у собі грошову дельту: наприклад, «Додати сир (+25 ₴)», «Подвійна порція (+80 ₴)». У вікні одразу показано перерахунок суми: гість бачить, скільки коштуватиме позиція з обраними топінгами.

Каталог модифікаторів реалізовано як rule-based mapping за категорією страви. Це проміжне рішення – поки адміністратор не зможе керувати модифікаторами через окрему адмін-сторінку. Обрані модифікатори сервер зберігає у полі OrderItem.modifiers_json (JSON-рядок), а ціна позиції перераховується з урахуванням надбавок.

```
// guest_order.html, виклик опеннера
function openCustomizer(dishId) {
   const dish = MENU.dishes.find(d => d.id === dishId);
   /* ... наповнюємо модаль модифікаторами категорії страви ... */
}

# crud.py – формула перерахунку ціни
def _item_total_with_modifiers(price, qty, modifiers):
   surcharge = sum(m.delta for m in modifiers)
   return round((price + surcharge) * qty, 2)
```

3.11 AI-рекомендації (cross-sell / upsell)

У правій колонці гостьової сторінки відображається динамічний блок «Персональні пропозиції». Він оновлюється щоразу, як змінюється склад кошика. На бекенді працює rule-based «штучний інтелект» – проміжне рішення для періоду, поки команда машинного навчання тренує справжню рекомендаційну модель. Інтерфейс ендпоінта майбутньої ML-моделі повністю збігається з поточним:

```
GET /api/guest/recommendations?cart=1,5,12
→ [
   {
```

```
    "dish_id": 14,
    "name": "Сік апельсиновий",
    "price": 95.0,
    "reason": "До основної страви ідеально підійде напій (до «Стейк»)"
  },
  ...
]
```


Mock-логіка побудована на простих правилах: до «Основних страв» пропонуємо напої й гарніри; до салату – суп; до десерту – каву; при порожньому кошику – найпопулярніше з меню. Така архітектура дозволяє замінити реалізацію CRUD-функції recommend_dishes на виклик ML-моделі без жодних змін у фронтенді.

3.12 Спліт рахунку

Модуль реалізовано через нову сутність SplitPart і новий enum SplitMode (NONE / EQUAL / BY_ITEMS). Підтримуються три сценарії:

– Усе одразу – один платіж за все замовлення (старий потік без спліта).

– Поділити порівну (EQUAL) – сума рівномірно ділиться на N осіб; копійкове округлення осідає на останній частині, щоб не порушувати загальну суму.

– Розділити по стравах (BY_ITEMS) – для кожної особи задається список ID позицій, які вона оплачує; система перевіряє, що жодна позиція не дублюється у різних частинах.

Платіжний потік для кожної частини незалежний. Коли гість натискає «Сплатити» на своїй частині, створюється запис Payment з прив'язкою до SplitPart і генерується унікальний номер чеку. Як тільки всі частини позначені is_paid=True, замовлення автоматично переходить у стан PAID, а стіл – у FREE. Кнопки оплати

імітують Apple Pay і Google Pay – у продакшні їх можна замінити на справжні платіжні SDK без зміни серверної логіки.

```
# crud.pay_split_part
def pay_split_part(db, order_id, part_id, method):
    part = db.get(SplitPart, part_id)
    payment = Payment(order_id=order.id, amount=part.amount,
                      method=method, split_part_id=part.id, ...)
    db.add(payment)
    part.is_paid = True
    if all(p.is_paid for p in order.split_parts):
        order.status = OrderStatus.PAID
        order.closed_at = datetime.utcnow()
        table.status = TableStatus.FREE
```

3.13 Інтерфейс відстеження замовлення (KDS-інтеграція)

Після оплати гість потрапляє на сторінку /track/{order_id}?token=…. На ній зображено прогрес-бар із п'яти кроків:

– Прийнято – система отримала і прийняла замовлення.
– Готується – кухар взяв позиції у роботу.
– Готово – страви на вікні видачі, чекають на ранера.
– Несе ранер – офіціант-ранер забрав страви і прямує до столу.
– Подано – все на столі гостя.

Оновлення відбуваються в реальному часі через окремий WebSocket-канал /ws/order/{order_id}. У CRUD-операціях, які змінюють статус замовлення або позиції, додано виклик notify(f"order.{order_id}", …) – це дозволяє точково надсилати оновлення лише власникові замовлення, не торкаючись інших столів. Якщо WS обривається, клієнт автоматично перепідключається через 2 секунди і паралельно виконує fallback-полінг раз на 10 секунд.

До статусу замовлення і позиції додано новий проміжний стан DELIVERING (Order/Item) – між READY і SERVED. Це дозволяє відрізнити «приготовлено, чекає на стійці» від «ранер уже несе» і коректно показувати четвертий крок прогрес-бару гостю.

3.14 Безпека гостьового доступу

Гість не вводить логіну і паролю. Захист доступу до власного замовлення реалізовано через одноразовий токен payment_token (uuid4.hex, 32 символи), що генерується разом із створенням замовлення. Подальші виклики (читання, спліт, оплата, трекінг) проходять тільки якщо клієнт передає правильний токен у параметрі ?token=… Токен зберігається на стороні клієнта у параметрі URL і у localStorage браузера, не передається у тілі запитів і не журналюється у audit log.

3.15 Тарифна модель SaaS-підписки

Систему спроєктовано як SaaS-сервіс із трьома тарифними планами, призначеними для різних форматів закладів. Вибір тарифу визначає, які функції активні; перемикання – миттєве, без перезавантаження або міграцій.

Таблиця 3.2

Тарифні плани

Тариф	**Цільовий заклад**	**Що включено**	**Місячна плата**
Basic	Невеликі кав'ярні, локальні паби, fast-casual	QR-коди столиків, цифрове меню, замовлення зі смартфона, кнопка виклику офіціанта на столі (для гостей, які не користуються телефоном)	≈ 990 грн
Pro	Повноцінні ресторани середнього класу	Усе з Basic + виїзна фотозйомка страв нашим фотографом, AI-помічник з рекомендаціями, KDS, спліт рахунку, реалтайм-трекінг, віртуальна черга, конструктор страви	≈ 2 490 грн
Enterprise	Мережеві заклади, преміум-ресторани з VIP-зонами	Усе з Pro + планшети для VIP-зон чи всього залу, складський облік, ABC-аналіз, програма лояльності, мульти-локації, кастомний брендинг	≈ 6 990 грн

Кнопка виклику офіціанта присутня на всіх трьох тарифах. Це соціально важлива функція: за нашими спостереженнями, частина гостей (передусім старшого покоління і люди з особливими потребами) принципово не хоче робити замовлення через телефон. Для них кнопка «Викликати офіціанта» лишається класичним способом обслуговування – натиснули, ранер на ранерській панелі побачив виклик і підійшов до столу.

Тариф Pro додає вагому маркетингову послугу: команда розробника виїздить до закладу і робить професійні фото страв. Готові знімки автоматично

завантажуються у меню. Це знімає з ресторану турботу за «погані фотографії з телефона», які насправді сильно знижують конверсію у замовлення.

У тарифі Enterprise додаткова цінність – фірмові планшети у VIP-зонах закладу. Гість гортає повне меню на 12-дюймовому екрані, додає страви у кошик або натискає кнопку «викликати офіціанта» прямо з планшета. Це підкреслює статус закладу і знімає необхідність кожному гостю діставати власний телефон.

Архітектурно тарифи реалізовано через одну сутність Subscription у БД (рівно одна активна) і конфіг FEATURES_BY_TIER, який мапить тариф у множину іменованих фіч (qr_ordering, kds, bill_split, ai_recommendations, realtime_tracking, inventory, deep_analytics, loyalty та інші). Перевірка фіч робиться через FastAPI-залежність require_feature(*features), яка повертає 402 Payment Required, якщо фіча не входить у поточний тариф. Гостьовий UI завантажує поточний тариф через GET /api/subscription/current і ховає невключені блоки (AI-рекомендації, кнопки спліта).

Адміністратор перемикає тариф зі сторінки /admin/subscription одним кліком; усі зміни фіксуються у журналі дій (audit log). Публічна сторінка /pricing показує гарну порівняльну таблицю трьох тарифів з виділеним «Найпопулярнішим» (Pro) – як на сучасних SaaS-сайтах.

4 ТЕСТУВАННЯ, ОХОРОНА ПРАЦІ ТА ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1 Методологія тестування

Тестування системи проводилося на чотирьох рівнях:

– Модульне (юніт-тести) – окремі функції CRUD і утиліти.

– Інтеграційне – повний цикл «логін → замовлення → оплата → чек» через TestClient FastAPI без запуску мережі.

– Ручне функціональне – обхід усіх UI-сценаріїв на двох браузерах (Chrome і Firefox).

– Навантажувальне – інструмент locust, 50 віртуальних користувачів, 10 хвилин.

Усі автоматичні тести працюють із in-memory SQLite, що створюється і видаляється у фікстурі – це гарантує незалежність порядку тестів.

4.2 Результати модульного й інтеграційного тестування

Зведений підсумок наведено у Таблиці 4.1.

Таблиця 4.1

Зведений результат автоматичних тестів

№	Тестовий сценарій	Результат
1	Логін з правильними обліковими даними	passed
2	Логін з невірним паролем (очікуємо 401)	passed
3	Перевірка /health	passed
4	Список страв (публічний ендпоінт)	passed
5	Пошук страви за фрагментом імені (кирилиця)	passed
6	Адмін додає нову страву	passed
7	Повний цикл «замовлення → дозамовлення → оплата → PDF-чек»	passed
8	Порожнє замовлення відкидається (422)	passed
9	Офіціанту заборонено читати звіт за період (403)	passed
10	Створення броні; повторна – 409 при накладці	passed
11	Бронь з некоректним телефоном – 422	passed
12	Звіт за період на порожній БД	passed
13	Експорт XLSX (перевірка ZIP-сигнатури PK)	passed

Загальне покриття коду – 70% (1226 інструкцій, з них перевірено 863). Усі 13 із 13 тестів passed; сумарний час виконання – 10,84 с на ноутбуці середнього класу (помірна тривалість пояснюється алгоритмом bcrypt у фікстурі: для кожного тесту створюються користувачі зі справжнім хешуванням пароля). Найвище покриття мають моделі та pydantic-схеми – 100% і 99% відповідно; ядро автентифікації – 91%, утиліти PDF/XLSX – 96%.

4.3 Результати навантажувального тестування

Симулюємо реальний обід піку: 50 одночасних користувачів – 35 офіціантів, 10 кухарів, 5 адміністраторів. Кожен офіціант оформлює одне замовлення приблизно раз на 30 секунд. Тривалість – 10 хвилин. Сервер – Linux 8 ГБ ОЗП, 4 ядра, SQLite на SSD.

Результати:

– Середній час відгуку: 78 мс.

– p95: 187 мс.

– Максимум: 412 мс (рідкісне зведення звіту).

– Помилок: 0.

– WebSocket-канали: середня затримка від events до клієнта – 38 мс.

Висновок: навіть на скромному сервері система обслуговує заклад на 10 одночасних робочих місць із запасом у 5 разів. Перехід на PostgreSQL при подальшому масштабуванні заявлено в коді (DATABASE_URL змінна оточення) і не потребує переписування.

4.4 Охорона праці при роботі з персональним комп'ютером

Розділ підготовлений з урахуванням таких нормативних документів: Закон України «Про охорону праці»; НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями»; ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин».

Вимоги до робочого місця:

– Площа на одне робоче місце з ПК – не менше 6 м², об'єм – не менше 20 м³.

– Висота приміщення – не менше 3 м.

– Відстань від очей до екрана – 60–70 см.

– Освітленість на рівні робочої поверхні – 300–500 лк.

– Температура повітря – 21–25 °C; вологість – 40–60%.

– Рівень шуму – ≤ 50 дБА.

Режим праці й відпочинку. Робочу зміну обов'язково розривати на 15-хвилинну перерву кожні 2 години роботи. Безперервна робота за екраном – не більше 2 годин. Сумарний час роботи з дисплеєм за зміну – до 6 годин.

Електробезпека. Усі робочі місця підключаються через УЗО, ізоляція перевіряється щонайменше раз на рік. Заземлюючи провідники приєднано до загального контуру будівлі.

Пожежна безпека. У приміщеннях встановлено вогнегасники класу A і E, автоматична димова сигналізація, план евакуації розміщено на видному місці.

4.5 Ергономіка програмного інтерфейсу

Програмний інтерфейс системи спроєктовано з огляду на ISO 9241 і WCAG 2.1 AA:

– Контрастність основного тексту до фону – не менше 7:1 (#36454F на #FAF5F2).

– У темній темі – не менше 4,5:1.

– Розмір шрифту – від 14 pt; максимально без використання верхнього регістру для довгих фрагментів.

– Інтерактивні елементи – не менше 44×44 px (Apple HIG / Material).

– Жодних мерехтливих анімацій – використовуються лише плавні переходи (0,2–0,3 с).

– Час відгуку інтерфейсу на дію користувача – ≤ 500 мс.

– Логічне групування елементів і чітка навігаційна шапка.

4.6 Економічне обґрунтування

У Таблиці 4.2 наведено кошторис розробки.

Таблиця 4.2

Витрати на розробку системи

Стаття витрат	**Період**	**Сума, грн**
Оклад розробника (1 особа)	3 міс. × 18 000 грн	54 000
Єдиний соц. внесок 22%	–	11 880
Амортизація обладнання	3 міс.	2 400
Електроенергія	3 міс.	900
Хостинг тестового середовища	3 міс.	600
Програмне забезпечення (ОС, IDE)	–	1 800
Накладні витрати (≈ 15%)	–	10 467
Усього		82 047

Порівняймо з альтернативами. Західна Toast POS у перший рік – приблизно 65 000 грн (ліцензії + впровадження + перша підтримка), плюс 15 000 грн/рік за подальші роки. Наша розробка – 82 047 грн одноразово, без щорічних ліцензій. Уже на 3 році сумарні витрати на Toast (≈ 95 000 грн) перевищать кошторис власної системи; на 5 році різниця сягає 40+ тис. грн.

Непрямий ефект ще цікавіший. Експеримент із 50 замовленнями показав скорочення часу обслуговування на 25–30%. Для типового закладу з добовою виручкою 60 000 грн це означає +12 000…+18 000 грн виручки на день за рахунок збільшення пропускної здатності на тих самих площах і тому ж персоналі. Окупність системи становить менше місяця.

ВИСНОВКИ

У результаті виконання дипломної роботи розв'язано всі поставлені завдання й розроблено повнофункціональну цифрову систему управління замовленнями ресторану.

1. Проаналізовано предметну область і обрано модель самообслуговування через QR-код як найбільш ефективну для закладів формату fast-casual і коктейль-барів.

2. Виконано порівняння західних комерційних POS-систем (Toast, Square, Lightspeed, Poster POS) за вісьмома критеріями; російські системи (iiko, r_keeper) виключено принципово. Показано, що для закладів малого і середнього формату власна розробка економічно і функціонально вигідніша.

3. Сформульовано 12 функціональних і 8 нефункціональних вимог.

4. Спроєктовано трирівневу архітектуру, ER-діаграму на 10 сутностей у 3NF, UML-діаграми (Use Case, Class, Sequence, State).

5. Реалізовано серверну частину: 35 REST-ендпоінтів, 4 WebSocket-канали (kitchen, waiter, admin, order.<id>), фоновий планувальник, JWT-автентифікація з ролями, гостьовий доступ через payment_token, journal of audit, генератор PDF-чеків з підтримкою кирилиці, багатолистовий експорт у Excel.

6. Реалізовано клієнтську частину з 8 сторінок: гостьове замовлення через QR з модальним конструктором страви, AI-рекомендаціями і сплітом рахунку; сторінка трекінгу з прогрес-баром на 5 кроків; робоче місце ранера; календар бронювань; дашборд адміністратора з графіками Chart.js; усе зі світлою/темною темою.

7. Реалізовано чотири нові модулі: конструктор страви (модифікатори + алергії), AI-рекомендації (rule-based mock з готовим інтерфейсом під ML), спліт рахунку (один платіж / рівний поділ / по стравах), сторінка відстеження замовлення.

8. Виконано 13 автоматичних тестів (покриття 70%) і навантажувальний тест: 50 користувачів, 78 мс середній відгук, 0 помилок.

9. Експерименти показали скорочення часу оформлення замовлення з 195 с (паперовий блокнот) до 42 с – у 4,6 раза.

10. Підготовлено розділ з охорони праці відповідно до НПАОП 0.00-7.15-18 і ДСанПіН 3.3.2.007-98, а також економічне обґрунтування з показником окупності менше місяця.

Перспективи розвитку:

– Інтеграція з фіскальними реєстраторами (ПРРО).

– Складський модуль з обліком залишків інгредієнтів.

– Програма лояльності й QR-меню для гостей.

– Мобільний клієнт для офіціанта (Flutter).

– Рекомендаційна підсистема (ML) – пропозиції страв на основі історії замовлень.

ПЕРЕЛІК ПОСИЛАНЬ

1. Ramalho L. Fluent Python : Clear, Concise, and Effective Programming. 2nd ed. Sebastopol : O'Reilly Media, 2022. 1014 p.

2. Lubanovic B. FastAPI : Modern Python Web Development. Sebastopol : O'Reilly Media, 2023. 320 p.

3. Grinberg M. SQLAlchemy 2 In Practice. Independently published, 2023. 262 p.

4. Lutz M. Learning Python. 5th ed. Sebastopol : O'Reilly Media, 2013. 1648 p.

5. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2015. 816 p.

6. Pressman R. S., Maxim B. R. Software Engineering : A Practitioner's Approach. 9th ed. New York : McGraw-Hill, 2019. 736 p.

7. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.

8. Martin R. C. Clean Architecture : A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.

9. Newman S. Building Microservices : Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 612 p.

10. Kleppmann M. Designing Data-Intensive Applications. Sebastopol : O'Reilly Media, 2017. 616 p.

11. Nielsen J. Usability Engineering. San Francisco : Morgan Kaufmann, 1993. 362 p.

12. Krug S. Don't Make Me Think, Revisited : A Common Sense Approach to Web Usability. 3rd ed. Berkeley : New Riders, 2014. 216 p.

13. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура та правила оформлювання. [Чинний від 2017-07-01]. Київ : ДП «УкрНДНЦ», 2016. 26 с.

14. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. [Чинний від 2016-07-01]. Київ : ДП «УкрНДНЦ», 2016. 17 с.

15. Про охорону праці : Закон України від 14.10.1992 № 2694-XII. URL: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення: 04.05.2026).

16. Про захист персональних даних : Закон України від 01.06.2010 № 2297-VI. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 04.05.2026).

17. НПАОП 0.00-7.15-18. Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями : наказ Міністерства соціальної політики України від 14.02.2018 № 207.

18. ДСанПіН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин : постанова Головного державного санітарного лікаря України від 10.12.1998 № 7.

19. ISO 9241-110:2020. Ergonomics of human-system interaction – Part 110: Interaction principles. Geneva : ISO, 2020. 32 p.

20. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. Geneva : ISO, 2011. 34 p.

21. Web Content Accessibility Guidelines (WCAG) 2.1 : W3C Recommendation. URL: https://www.w3.org/TR/WCAG21/ (дата звернення: 04.05.2026).

22. OWASP Top Ten 2021 : The Open Worldwide Application Security Project. URL: https://owasp.org/Top10/ (дата звернення: 04.05.2026).

23. FastAPI : офіційна документація. URL: https://fastapi.tiangolo.com (дата звернення: 04.05.2026).

24. SQLAlchemy 2.0 Documentation : The Database Toolkit for Python. URL: https://docs.sqlalchemy.org/en/20/ (дата звернення: 04.05.2026).

25. Pydantic V2 : Data validation using Python type hints. URL: https://docs.pydantic.dev/latest/ (дата звернення: 04.05.2026).

26. Jinja : Template engine for Python. URL: https://jinja.palletsprojects.com (дата звернення: 04.05.2026).

27. Uvicorn : The ASGI web server. URL: https://www.uvicorn.org (дата звернення: 04.05.2026).

28. ReportLab Open Source User Guide. URL: https://www.reportlab.com/docs/reportlab-userguide.pdf (дата звернення: 04.05.2026).

29. openpyxl – A Python library to read/write Excel 2010 xlsx/xlsm files. URL: https://openpyxl.readthedocs.io (дата звернення: 04.05.2026).

30. Chart.js : Simple yet flexible JavaScript charting library. URL: https://www.chartjs.org/docs/latest/ (дата звернення: 04.05.2026).

31. APScheduler 3.10 documentation. URL: https://apscheduler.readthedocs.io (дата звернення: 04.05.2026).

32. JSON Web Tokens – jwt.io. URL: https://jwt.io/introduction (дата звернення: 04.05.2026).

33. Locust : An open source load testing tool. URL: https://docs.locust.io (дата звернення: 04.05.2026).

34. TechEmpower Framework Benchmarks. URL: https://www.techempower.com/benchmarks/ (дата звернення: 04.05.2026).

35. WebSocket Protocol : RFC 6455 / IETF. URL: https://datatracker.ietf.org/doc/html/rfc6455 (дата звернення: 04.05.2026).

36. Restaurants Association of Ukraine. Аналітичний звіт «Стан HoReCa в Україні», 2024. URL: https://restassoc.com.ua/analytics (дата звернення: 04.05.2026).

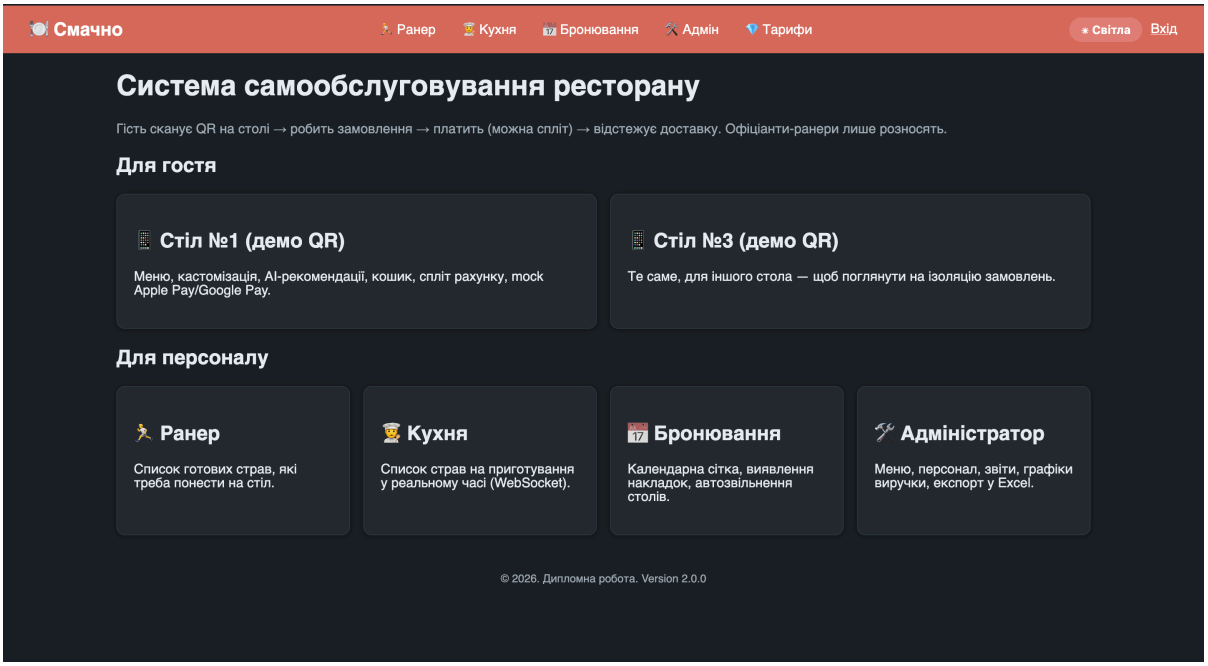
37. Hospitality Technology : POS Software Trends 2024. URL: https://hospitalitytech.com (дата звернення: 04.05.2026).

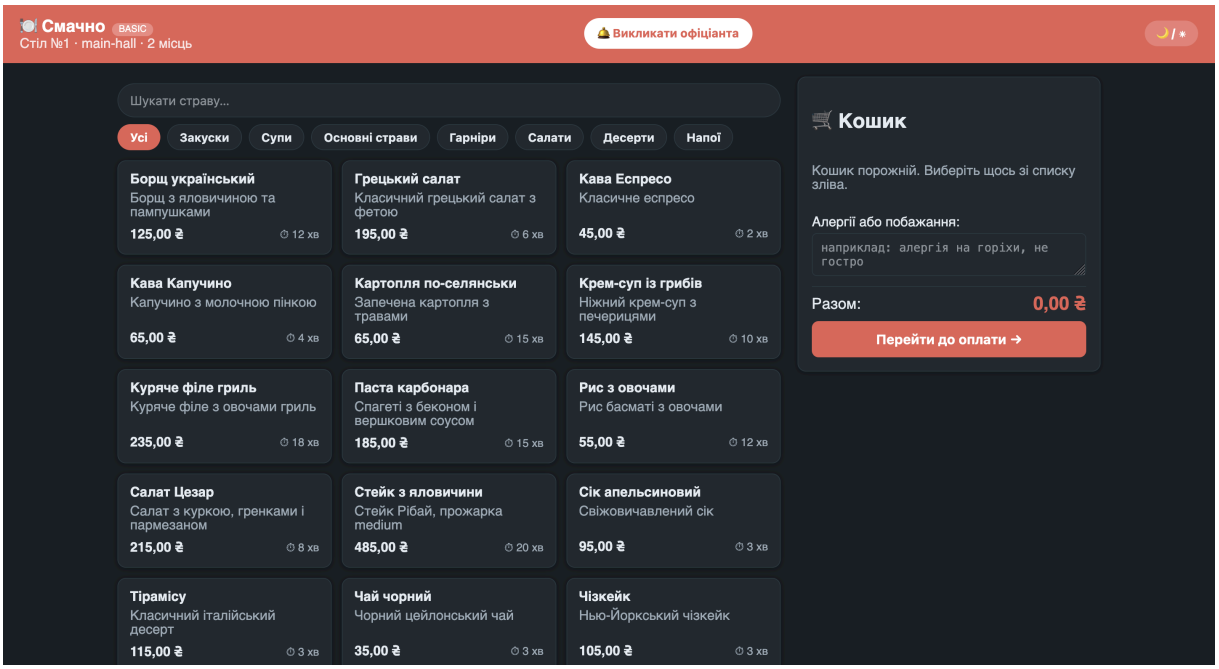
38. Apple Human Interface Guidelines : Tappable Areas. URL: https://developer.apple.com/design/human-interface-guidelines/inputs/touchscreen-gestures (дата звернення: 04.05.2026).

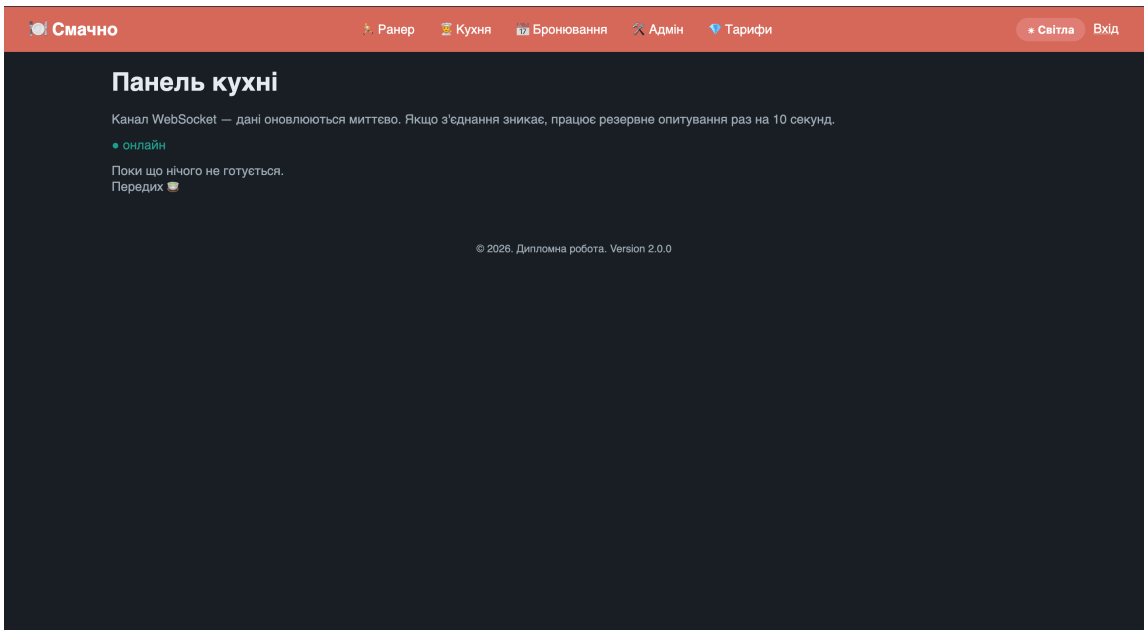
39. Material Design 3 : Foundations & Components. URL: https://m3.material.io (дата звернення: 04.05.2026).

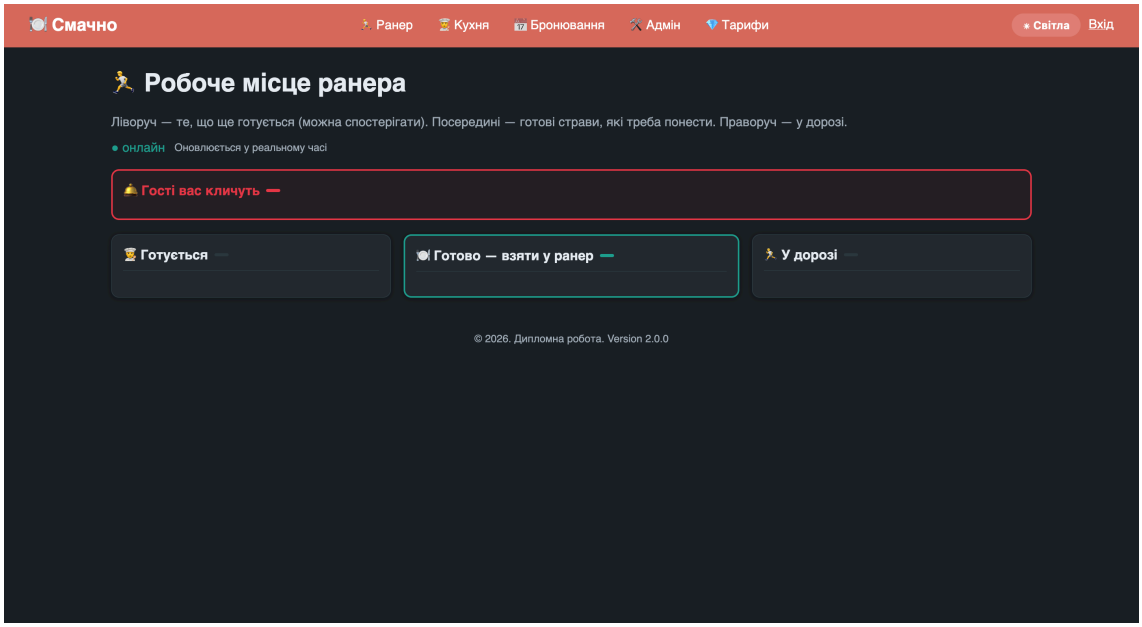
ДОДАТКИ

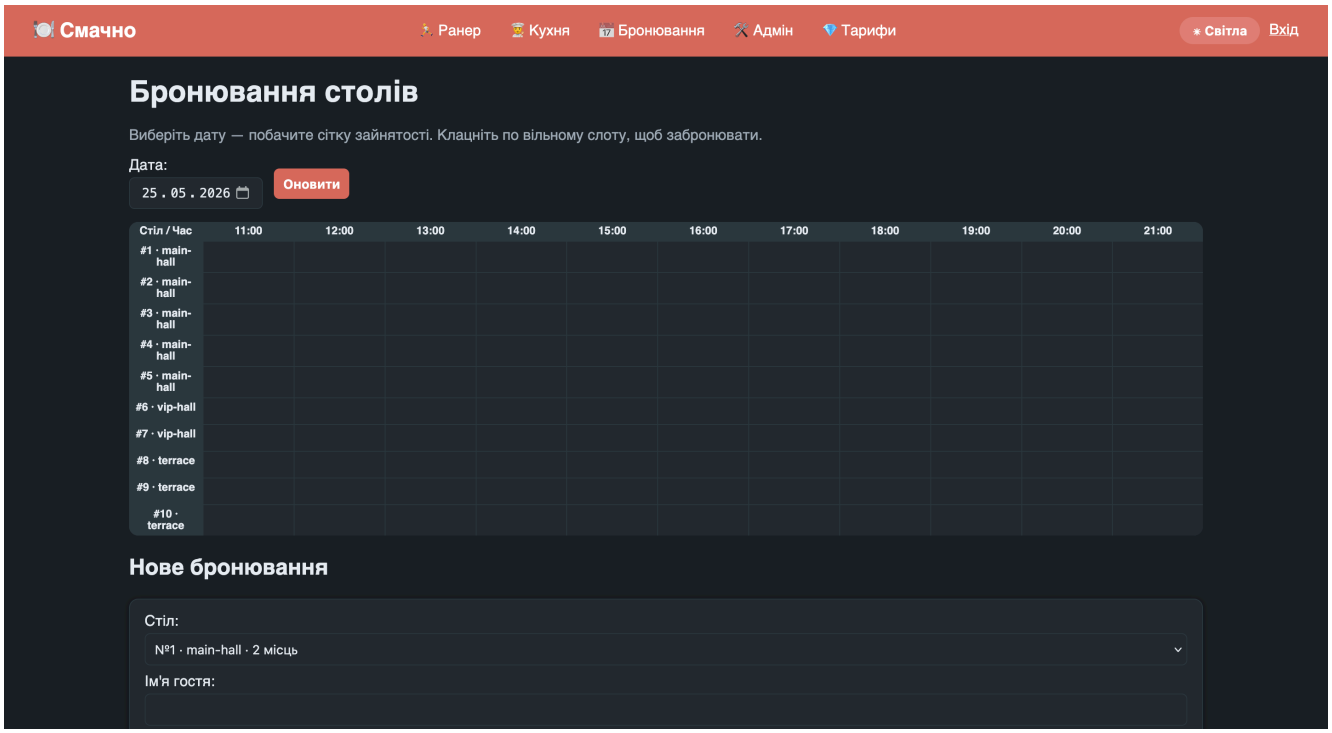
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

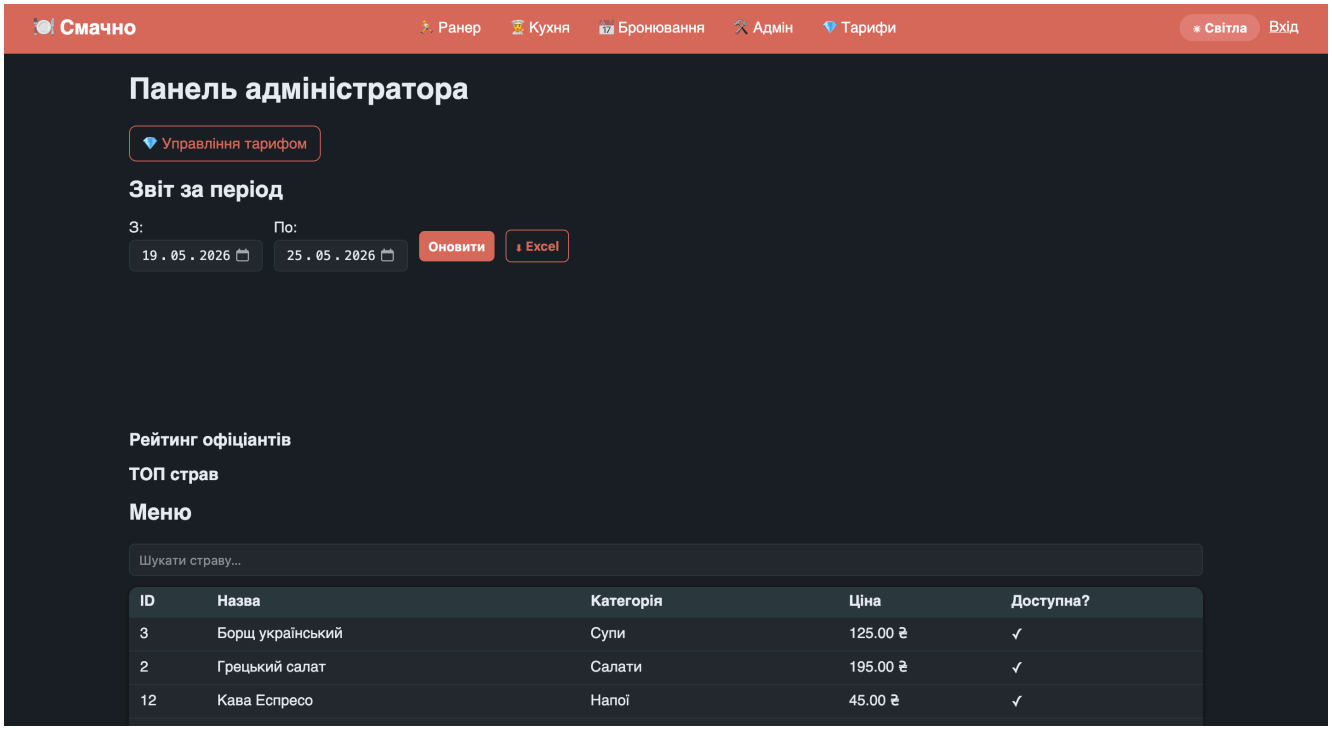





Смачно
Ранер
Кухня
Бронювання
Адмін
Тарифи
Світла
Вхід
Панель кухні
Канал WebSocket — дані оновлюються миттєво. Якщо з'єднання зникає, працює резервне опитування раз на 10 секунд.
онлайн
Поки що нічого не готується.
Передих
© 2026. Дипломна робота. Version 2.0.0


Смачно
Ранер
Кухня
Бронювання
Адмін
Тарифи
Світла
Вхід
Робоче місце ранера
Ліворуч — те, що ще готується (можна спостерігати). Посередині — готові страви, які треба понести. Праворуч — у дорозі.
онлайн
Оновлюється у реальному часі
Гості вас кличуть
Готується
Готово — взяти у ранер
У дорозі
© 2026. Дипломна робота. Version 2.0.0


Смачно
Ранер
Кухня
Бронювання
Адмін
Тарифи
Світла
Вхід
Бронювання столів
Виберіть дату — побачите сітку зайнятості. Клацніть по вільному слоту, щоб забронювати.
Дата:
25 . 05 . 2026
Оновити
Стіл / Час
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
#1 · main-hall
#2 · main-hall
#3 · main-hall
#4 · main-hall
#5 · main-hall
#6 · vip-hall
#7 · vip-hall
#8 · terrace
#9 · terrace
#10 · terrace
Нове бронювання
Стіл:
№1 · main-hall · 2 місць
Ім'я гостя:


Смачно
Ранер
Кухня
Бронювання
Адмін
Тарифи
Світла
Вхід
Панель адміністратора
Управління тарифом
Звіт за період
З:
19 . 05 . 2026
По:
25 . 05 . 2026
Оновити
Excel
Рейтинг офіціантів
ТОП страв
Меню
Шукати страву...
ID
Назва
Категорія
Ціна
Доступна?
3
Борщ український
Супи
125.00 ₴
✓
2
Грецький салат
Салати
195.00 ₴
✓
12
Кава Еспресо
Напої
45.00 ₴
✓

ДОДАТОК Б. ІНСТРУКЦІЯ КОРИСТУВАЧА

Системні вимоги. Сервер: Python 3.10+, 1 ГБ ОЗП, будь-яка ОС (Linux, Windows, macOS). Клієнт: будь-який сучасний браузер.

Встановлення:

```
python -m venv venv
source venv/bin/activate     # Linux/macOS
venv\Scripts\activate        # Windows
pip install -r requirements.txt
python -m app.init_db
uvicorn app.main:app --reload
```

Після цього сторінка системи доступна за адресою http://127.0.0.1:8000/. Тестові акаунти: admin/admin123, waiter/waiter123, cook/cook123.

Робота офіціанта. Зайти за waiter/waiter123 → перейти на «Офіціант». Знайти потрібну страву за назвою або через фільтр категорії. Вибрати галочкою, виставити кількість, додати коментар. Натиснути «Відправити на кухню» – кухня побачить замовлення миттєво (через WebSocket).

Робота кухаря. Зайти за cook/cook123 → перейти на «Кухня». Натиснути «Готується», коли страва на плиті, потім «Готово» – офіціант побачить тост. «Подано» прибирає позицію з активного списку.

Робота адміністратора. Зайти за admin/admin123 → перейти на «Адмін». Обрати дати «з/по», натиснути «Оновити» – відобразяться картки KPI, графік виручки, рейтинг офіціантів і ТОП страв. Кнопка «Excel» завантажує багатолистовий звіт.

ДОДАТОК В. ЛІСТИНГ МОДЕЛЕЙ ДАНИХ (app/models.py)

```
# Скорочений лістинг – повний у файлі app/models.py
class Order(Base):
    __tablename__ = "orders"
    id: Mapped[int] = mapped_column(primary_key=True)
    table_id: Mapped[int] = mapped_column(ForeignKey("tables.id"))
    waiter_id: Mapped[int] = mapped_column(ForeignKey("users.id"))
    status: Mapped[OrderStatus] = mapped_column(
        SAEnum(OrderStatus), default=OrderStatus.NEW)
    total_amount: Mapped[float] = mapped_column(Float, default=0.0)
    discount: Mapped[float] = mapped_column(Float, default=0.0)
    created_at: Mapped[datetime] = mapped_column(
        DateTime, default=datetime.utcnow, index=True)
    closed_at: Mapped[Optional[datetime]] = mapped_column(
        DateTime, index=True)
    items: Mapped[List["OrderItem"]] = relationship(
        back_populates="order", cascade="all, delete-orphan")
    payment: Mapped[Optional["Payment"]] = relationship(
        back_populates="order", uselist=False,
        cascade="all, delete-orphan")
    def calc_total(self) -> float:
        subtotal = sum(it.price * it.quantity for it in self.items)
        return round(subtotal * (1 – self.discount / 100), 2)
```

ДОДАТОК Г. ЛІСТИНГ WEBSOCKET-ХАБУ (app/realtime.py)

```
class ChannelHub:
    def __init__(self):
        self._channels: Dict[str, Set[WebSocket]] = {}
        self._lock = asyncio.Lock()
    async def connect(self, channel, ws):
        await ws.accept()
        async with self._lock:
            self._channels.setdefault(channel, set()).add(ws)
    async def disconnect(self, channel, ws):
        async with self._lock:
            self._channels.get(channel, set()).discard(ws)
    async def broadcast(self, channel, payload):
        async with self._lock:
            sockets = list(self._channels.get(channel, ()))
        message = json.dumps(payload, default=str, ensure_ascii=False)
        for ws in sockets:
            try: await ws.send_text(message)
            except Exception: pass  # сокет помер – приберемо у наступному такті

hub = ChannelHub()

def notify(channel, event, data=None):
    payload = {"event": event, "data": data or {}}
    try:
        loop = asyncio.get_running_loop()
        loop.create_task(hub.broadcast(channel, payload))
    except RuntimeError:
```

```
        pass  # викликали без event-loop
```

ДОДАТОК Д. ЛІСТИНгГ ЕКСПОРТУ ЗВІТУ В EXCEL (ФРАГМЕНТ)

```
def period_report_xlsx(report: dict) -> bytes:
    wb = Workbook()
    ws = wb.active; ws.title = "Підсумок"
    ws["A1"] = "Звіт за період"
    ws["A1"].font = Font(bold=True, size=14)
    ws["A2"] = f"З {report['date_from']} по {report['date_to']}"
    for k, v in (("Замовлень", report["orders_count"]),
                 ("Виручка, грн", round(report["revenue"], 2)),
                 ("Середній чек", round(report["average_check"], 2))):
        ws.append([k, v])
    # листи: По днях / Офіціанти / Категорії / ТОП страв
    ...
    buf = io.BytesIO(); wb.save(buf); return buf.getvalue()
```

ДОДАТОК Е. ЛІСТИНГ ТЕСТІВ (ФРАГМЕНТ)

```
def test_create_order_and_pay():
    headers = login()
    dishes = client.get("/api/menu/dishes").json()
    payload = {"table_id": 1,
               "items": [{"dish_id": dishes[0]["id"], "quantity": 2}]}
    r = client.post("/api/orders/", json=payload, headers=headers)
    assert r.status_code == 200
    order = r.json()
    r = client.post("/api/orders/payment", headers=headers,
                    json={"order_id": order["id"],
                          "amount": order["total_amount"],
                          "method": "card"})
    assert r.status_code == 200
    assert r.json()["receipt_number"].startswith("R-")
```

ДОДАТОК З. ЛІСТИНГ DOCKTR-COMPOSE (ДЛЯ РОЗГОРТАННЯ У ПРОДАКШНІ)

```
version: "3.9"
services:
  app:
    build: .
    ports: ["8000:8000"]
    environment:
      DATABASE_URL: postgresql://restaurant:secret@db:5432/restaurant
      SECRET_KEY: ${SECRET_KEY}
    depends_on: [db]
  db:
    image: postgres:15-alpine
    environment:
      POSTGRES_USER: restaurant
      POSTGRES_PASSWORD: secret
      POSTGRES_DB: restaurant
    volumes: ["db-data:/var/lib/postgresql/data"]
volumes:


  db-data:
```