Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

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

на тему СТВОРЕННЯ ПРОГРАМНОЇ СИСТЕМИ ДЛЯ АВТОМАТИЗАЦІЇ ОПРАЦЮВАННЯ ЗАМОВЛЕНЬ У СФЕРІ ПОСЛУГ
DEVELOPMENT AND RESEARCH OF SOFTWARE FOR AUTOMATING ORDER PROCESSING IN THE SERVICE SECTOR

Виконав(ла): студент(ка) 4 курсу, групи КНТ-213сп
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні науки

БІЛАН Б.І.
(ПРІЗВИЩЕ та ініціали)
Керівник ФЕДОРОНЧАК Т.В.
(ПРІЗВИЩЕ та ініціали)
Рецензент БАБЕНКО Н.В.
(ПРІЗВИЩЕ та ініціали)

2026

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)

Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____Сергій СУББОТІН
"____" _____2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

БІЛАНА Богдана Ігоровича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Створення програмної системи для автоматизації опрацювання замовлень у сфері послуг. Development and Research of Software for Automating Order Processing in the Service Sector
керівник проєкту (роботи) к.т.н., доцент, Федорончак Тетяна Василівна,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від " 07 " квітня 2026 року № 139
2. Строк подання студентом проєкту (роботи) 10 червня 2026 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області та математичне моделювання. 2. Проєктування архітектури системи та бази даних. 3. Програмна реалізація системи. 4. Експлуатація, тестування та експериментальне дослідження програми.
5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ФЕДОРОНЧАК Т.В., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки.		
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження	5 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка) _____ Богдан БІЛАН
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) Підпис Тетяна Федорончак
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 91 с., 7 табл., 15 рис., 3 дод., 15 джерел.

АВТОМАТИЗАЦІЯ ЗАМОВЛЕНЬ, ДИНАМІЧНЕ ЦІНОУТВОРЕННЯ, ТЕЛЕГРАМ-БОТ, FASTAPI, POSTGRESQL, RACE CONDITIONS, СФЕРА ПОСЛУГ, АСИНХРОННЕ ПРОГРАМУВАННЯ.

Об’єкт дослідження – моделі, методи, алгоритми та структури даних для побудови розподілених програмних систем автоматизації опрацювання замовлень.

Предмет дослідження – методи і програмні засоби динамічного ціноутворення та управління транзакційною цілісністю у системах бронювання послуг.

Мета роботи – підвищення швидкості опрацювання замовлень та оптимізація завантаженості ресурсів провайдерів послуг шляхом створення програмної системи з алгоритмом динамічного ціноутворення та автоматизованим інтерфейсом бронювання.

Матеріали, методи та технічні засоби: асинхронне та об’єктно-орієнтоване програмування, мова програмування Python, фреймворки FastAPI та aiogram 3, СУБД PostgreSQL (використання діапазонних типів даних та Exclusion Constraints), Redis для управління станами, технологія контейнеризації Docker.

Результати. Створено програмну систему агрегатора послуг, що складається з серверного API та клієнтського Telegram-бота. Реалізовано гібридну модель блокувань на рівні бази даних, яка повністю усуває стани гонитви (race conditions) при паралельному бронюванні слотів. Впроваджено математичну модель динамічного ціноутворення, що адаптує вартість послуги залежно від часових циклів та поточного попиту.

Висновки. Розроблено архітектуру та програмне забезпечення для автоматизації опрацювання замовлень, яке дозволяє масштабувати бізнес у сфері послуг, виключає помилки подвійного бронювання та оптимізує прибутковість платформи.

Галузь використання – автоматизація бізнес-процесів у сфері надання послуг (сервісні центри, оренда техніки, коворкінги) для покращення взаємодії з клієнтами та управління ресурсами.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 91 pages, 7 tables, 15 figures, 3 appendixes, 15 sources.

ORDER AUTOMATION, DYNAMIC PRICING, TELEGRAM BOT, FASTAPI, POSTGRESQL, RACE CONDITIONS, SERVICE SECTOR, ASYNCHRONOUS PROGRAMMING.

Object of study is models, methods, algorithms, and data structures for building distributed software systems for order processing automation.

Subject of study is methods and software tools for dynamic pricing and transactional integrity management in service booking systems.

The purpose of the work is to increase the speed of order processing and optimize the workload of service providers' resources by creating a software system with a dynamic pricing algorithm and an automated booking interface.

Materials, methods, and tools: asynchronous and object-oriented programming, Python programming language, FastAPI and aiogram 3 frameworks, PostgreSQL DBMS (using range data types and Exclusion Constraints), Redis for state management, Docker containerization technology.

Results. A service aggregator software system consisting of a server API and a client Telegram bot has been created. A hybrid locking model at the database level has been implemented, which completely eliminates race conditions during parallel booking of time slots. A mathematical model of dynamic pricing has been introduced, which adapts the cost of the service depending on time cycles and current demand.

Conclusions The architecture and software for order processing automation have been developed. It allows scaling businesses in the service sector, eliminates double booking errors, and optimizes platform profitability.

The field of use is the automation of business processes in the service provision sector (service centers, equipment rental, coworking spaces) to improve interaction with clients and resource management.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

ACID – Atomicity, Consistency, Isolation, Durability;

CRUD – Create, Read, Update, Delete;

DB – Database;

DDL – Data Definition Language;

FSM – Finite State Machine;

HTTP – HyperText Transfer Protocol;

JSON – JavaScript Object Notation;

ORM – Object-Relational Mapping;

REST – Representational State Transfer;

URL – Uniform Resource Locator;

БД – база даних;

ПЗ – програмне забезпечення;

СУБД – система управління базами даних.

ВСТУП

Автоматизація опрацювання замовлень відіграє критичну роль у забезпеченні економічної ефективності, конкурентоспроможності та якості обслуговування у сфері послуг. Такі технології мають велике значення як для малого бізнесу - для оптимізації розкладів окремих майстрів чи приватних сервісів, так і у великих інфраструктурних проєктах, таких як оренда спецтехніки, коворкінги чи логістика. Ручне управління бронюваннями та використання статичних прайс-листів призводять до неефективного розподілу ресурсів, втрати прибутку під час пікових навантажень та виникнення помилок подвійного бронювання через нездатність обробляти паралельні запити.

У сучасному цифровому середовищі розподілені програмні системи та алгоритми динамічного ціноутворення стали ефективним інструментом для розв'язання цих проблем. Асинхронні архітектури дозволяють автоматично адаптувати вартість послуг, базуючись на поточному попиті та завантаженості, і миттєво фіксувати транзакції у базі даних без аномалій конкурентного доступу. Це значно спрощує процес бронювання для клієнтів, роблячи його зручним та доступним завдяки використанню інтерфейсів популярних месенджерів (зокрема, Telegram).

Попит на програмні рішення для автоматичного розподілу послуг постійно зростає. Такі агрегатори дозволяють швидко та якісно опрацьовувати заявки, уникати простоїв ресурсів та підвищувати загальну прибутковість провайдерів. Застосування таких рішень особливо актуальне в умовах масштабування бізнесу, де вимагається безперебійна взаємодія сотень клієнтів із гарантією збереження транзакційної цілісності.

Таким чином, впровадження програмних систем для автоматизації опрацювання замовлень із механізмами динамічного ціноутворення є не лише технологічним кроком уперед, а й необхідною економічною умовою для оптимізації ринку послуг для бізнесу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ

Для того щоб програмна система агрегації послуг була ефективною та економічно вигідною, необхідна наявність надійних алгоритмів обробки транзакцій та гнучких механізмів ціноутворення. Від архітектурної цілісності бази даних та швидкодії серверного програмного забезпечення залежить здатність бізнесу опрацьовувати великі потоки клієнтів без технічних збоїв, подвійних бронювань та втрати прибутку.

Предметною областю є інженерія розподілених програмних систем та управління базами даних, зокрема автоматизація процесів взаємодії між споживачами та провайдерами послуг. Це включає аналіз бізнес-процесів, розв'язання проблеми конкурентного доступу (race conditions) до спільних ресурсів (часових слотів) та застосування математичних моделей для динамічної зміни вартості послуг залежно від поточного попиту і часу.

1.1 Аналіз процесів опрацювання замовлень та їх автоматизація

Сфера послуг, яка охоплює побутове обслуговування, ремонт техніки, оренду обладнання та коворкінги, характеризується високими темпами розвитку. Її головна особливість - неможливість фізичного зберігання послуг (продаж часу фахівця або періоду експлуатації техніки у вигляді фіксованих часових слотів). Отже, нереалізований вчасно часовий ресурс зумовлює безповоротні фінансові втрати для бізнесу.

У практиці малого та середнього бізнесу управління замовленнями переважно здійснюється в ручному режимі (шляхом телефонної комунікації та фіксації даних у журналах або локальних електронних таблицях), що породжує низку критичних проблем:

– людський фактор. Працівники помиляються під час заповнення розкладу, що провокує накладки (overbooking) та відтік клієнтів;

– обмежена комунікація. Клієнти змушені телефонувати лише в робочі години, тоді як сучасні споживачі потребують можливості створення бронювання самостійно у режимі 24/7;

– жорстке ціноутворення. Адміністратори використовують статичну вартість послуг. Бізнес не встигає реагувати на підвищений попит під час «годин пік», внаслідок чого компанії втрачають маржинальний прибуток, а також не можуть швидко знизити ціни для заповнення порожніх часових слотів.

Впровадження засобів автоматизації дозволяє усунути посередників між клієнтом та надавачем послуги шляхом формування єдиного інформаційного простору. Центральним елементом такої системи виступає інтерактивний календар бронювання, логіка функціонування якого зводиться до інженерної задачі формалізації та автоматизованого керування доступом до обмежених часових ресурсів провайдера послуг [часових ресурсів провайдера послуг [1].

Системи автоматизації є необхідною умовою для масштабування бізнесу, оскільки при зростанні обсягів замовлень ручне опрацювання стає критичним «вузьким місцем» (bottleneck), яке суттєво уповільнює розвиток підприємства.

Впровадження програмної системи автоматизації вирішує наведені проблеми завдяки можливостям:

– скорочення часу від першого запиту клієнта до підтвердження замовлення;

– контролю завантаженості ресурсів у реальному часі;

– збирання статистики для аналізу та прогнозування попиту;

– застосування алгоритмів для динамічної зміни параметрів послуги (ціни, тривалості та пріоритету).

Перехід до автоматизованих програмних систем формує інженерний фундамент для ефективної бізнес-моделі. Розроблення системи, яка інтегрує

клієнтський інтерфейс на базі месенджера Telegram та алгоритми динамічного управління ціноутворенням, дозволяє якісно оптимізувати розподіл ресурсів.

1.2 Порівняльний аналіз програмних рішень та агрегаторів

Ефективне функціонування підприємств у сфері надання послуг (салони краси, консалтинг, логістика) безпосередньо залежить від оптимальності управління обмеженими у часі ресурсами. Основною проблемою сучасного малого та середнього бізнесу є використання статичних моделей ціноутворення, за яких вартість послуги залишається незмінною незалежно від коливань поточного попиту, часу доби чи дня тижня.

Відсутність математичних моделей для автоматичної корекції тарифів в існуючих засобах автоматизації є системним недоліком, що призводить до трьох критичних проблем:

– недоотримання маржинального прибутку в пікові години**.** Коли попит на конкретні часові слоти (наприклад, вечірній час або вихідні дні) різко зростає, фіксована вартість штучно обмежує дохідність. Клієнти, які виявляють високу готовність платити за дефіцитний час, купують послугу за базовим тарифом, що знижує потенційну рентабельність бізнесу;

– операційні збитки через простій інфраструктури в непікові періоди. У ранкові години або будні дні обладнання, орендовані площі та персонал функціонують вхолосту, генеруючи постійні витрати за відсутності доходу. Без інструменту автоматичного зниження ціни неможливо оперативно залучити еластичний за ціною сегмент споживачів для згладжування графіку завантаженості;

– запізнення та суб'єктивність ручного керування. Спроби менеджерів змінювати прайс-листи вручну залежно від суб'єктивної оцінки завантаженості є трудовитратними, неефективними та призводять до запізнення реакції на реальні ринкові зміни.

Розроблення та інтеграція математичної моделі автоматичного коригування вартості орієнтована на задоволення потреб трьох ключових суб'єктів розподіленої системи:

– для власників сервісного бізнесу модель є інструментом максимізації показника RevPAH (дохід на одну доступну годину). Вона дозволяє реалізувати принципи управління дохідністю (Yield Management), аналогічні тим, що використовуються в авіаційній та готельній індустріях, забезпечуючи рівномірне завантаження персоналу;

– для кінцевих споживачів (клієнтів) гнучка тарифна сітка забезпечує справедливий розподіл ресурсів: користувачі з високою еластичністю попиту отримують можливість економити кошти (обираючи дешевші непікові слоти), а клієнти з обмеженим часом - гарантований доступ до послуги в пікові години за рахунок відсікання надлишкового попиту ціною;

– для розробників розподілених архітектур математична модель з обчислювальною складністю $O_{(1)}$ є необхідною умовою для інтеграції в асинхронне середовище вебсервера (фреймворк FastAPI). Розрахунок ціни має відбуватися миттєво в момент обробки Webhook-запиту від Telegram API без блокування циклу подій (event loop) та виникнення затримок в інтерфейсі користувача.

Таким чином, автоматична корекція вартості є не просто додатковою функцією, а необхідним механізмом ліквідації економічних та технічних аномалій статичного розподілу ресурсів. Далі розглянемо критичний аналіз існуючих класів програмних рішень під кутом впровадження зазначеної логіки.

Для системного аналізу та порівняння функціональних можливостей наявного програмного забезпечення, основні параметри та критерії оцінки існуючих рішень і систем-агрегаторів зведено в таблиці 1.1. Це дозволить детально оцінити рівень інтеграції алгоритмів автоматичної корекції вартості та виявити технічні обмеження сучасних систем автоматизації під кутом досліджуваної логіки розподілу ресурсів.

З метою обґрунтування створення власної архітектури було проаналізовано три класи продуктів: системи керування розкладами (Core scheduling), комплексні CRM-платформи та спеціалізовані інтерфейси (Telegram-first). Проведений ретроспективний аналіз дозволив чітко розмежувати технічні межі застосування зазначених категорій програмного забезпечення та детермінувати ключові фактори їхньої неконкурентоспроможності в розрізі динамічного оперування вартісними показниками.

Таблиця 1.1 – Порівняльний аналіз існуючих систем бронювання та розроблюваного рішення

Критерій порівняння	Yclients	Calendly	Розроблювана система
Автоматичне динамічне ціноутворення	Відсутнє (тільки ручна зміна прайсів)	Відсутнє	Наявне (адитивно-мультиплікативна модель)
Захист від стану гонитви (Race conditions)	На рівні прикладного коду (іноді виникають збої)	Обмежений (вебінтерфейс)	Гібридний дворівневий (Redis Soft Lock + PostgreSQL Exclude Constraint)
Основний інтерфейс взаємодії	Вебпанель / Мобільний додаток	Вебвіджети	Асинхронний Telegram-бот (aiogram 3.x)
Архітектура отримання оновлень	Синхронні HTTP запити / Webhooks	REST API / Webhooks	Подієво-орієнтована через Webhook на FastAPI
Кастомізація під бізнес-логіку майстра	Низька (жорстка структура CRM)	Середня (налаштування шаблонів)	Висока (індивідуальні параметри сигмоїди та піків)

1.2.1 Системи керування розкладами (Core scheduling)

Ці платформи фокусуються на логіці розподілу часових слотів. Типовими прикладами є Cal.com та Acuity Scheduling. Сервіс Cal.com пропонує API-first підхід, що дозволяє інтегрувати його як бекенд-рушій у власні проєкти. Acuity Scheduling містить інструменти для управління чергами.

Незважаючи на гнучкість, зазначені системи працюють переважно через вебінтерфейс. Їхня інтеграція з месенджерами вимагає розробки додаткових прошарків (middleware). Основний недолік таких систем - відсутність вбудованих математичних моделей для автоматичної корекції вартості залежно від поточного попиту [2].

1.2.2 Бізнес-орієнтовані CRM-системи

Цей клас охоплює платформи Altegio (Yclients), Fresha та продукти з елементами штучного інтелекту типу Starta.one. Вони формують екосистеми для управління фінансами, персоналом та клієнтською базою. Платформа Starta.one надає AI-асистентів для Telegram.

Існують також вузькоспеціалізовані надбудови (наприклад, AI Beauty Bot), що розгортають інтелектуальний інтерфейс поверх базових CRM. Проте такі комплекси є надто складними та дорогими для малого бізнесу. Головним недоліком залишається жорстка прив'язка до статичних тарифних сіток: будь-яка зміна базової вартості послуги вимагає ручного втручання адміністратора, а автоматизована система не може миттєво відреагувати на динамічні коливання ринкового попиту [1].

Внаслідок такої технологічної обмеженості інформаційні контури подібних платформ функціонують у відриві від реальних мікроекономічних трендів, що призводить до виникнення касових розривів у періоди низької

споживчої активності та до упущення потенційного маржинального прибутку під час аномальних сплесків клієнтського попиту.

1.2.3 Спеціалізовані інтерфейсні рішення (Telegram)

Ці сервіси використовують Telegram як головне середовище взаємодії (UX-layer). Сюди належать Appointexo, HeyBookMeBot та AppointBot. Платформа Appointexo працює як SaaS-конструктор ботів. HeyBookMeBot синхронізує записи з Google Calendar. AppointBot аналізує текстові запити через AI-інтерфейс.

Такі продукти забезпечують високу якість клієнтського досвіду (Native UX). Однак їхня закрита комерційна архітектура блокує глибоку кастомізацію, зокрема унеможливлює інтеграцію алгоритмів динамічного ціноутворення на основі математичних функцій попиту [3].

1.2.4 Глобальні маркетплейси та платформи-агрегатори

До цієї категорії входять великі B2C-платформи, такі як Treatwell, Kabanchik та сервіси типу Uber. Вони працюють як централізовані посередники, самостійно забезпечуючи потік клієнтів. Вони інтегрують складні алгоритми динамічного ціноутворення (surge pricing), що дозволяють автоматично підвищувати ціну під час пікового попиту або дефіциту фахівців [2].

Ця модель спричиняє суттєві фінансові ризики для малого бізнесу, оскільки маркетплейси монополізують клієнтську базу та стягують комісію до 30% за транзакцію. Провайдер послуги втрачає контроль над алгоритмами, тоді як платформа змінює ціни виключно у власних інтересах, що призводить до розмивання ідентичності бренду.

Ринок засобів автоматизації не має ідеального продукту. Сучасні сервіси не поєднують низький поріг входу, транзакційну надійність та гнучке

ціноутворення. Комплексні CRM-системи обтяжують малий бізнес, а глобальні маркетплейси позбавляють контролю над прибутком. Існуючі Telegram-боти пропонують шаблонні функції та не мають архітектурного захисту від аномалій конкурентного доступу до спільних даних [4].

Для подолання наведених обмежень доцільним є розроблення спеціалізованої системи на базі асинхронного фреймворку FastAPI [5]. Впровадження власної математичної моделі ціноутворення створює конкурентну перевагу та повною мірою задовольняє потреби сучасного ринку послуг.

1.3 Обґрунтування моделей динамічного ціноутворення

З метою цифровізації бізнесу здійснюється відмова від статичних прайс-листів та впровадження алгоритмів гнучкого тарифоутворення. Динамічне ціноутворення (Dynamic Pricing) є стратегією, яка адаптує вартість послуги в реальному часі на основі аналізу поточного попиту, завантаженості фахівців та часових чинників [3].

Інтеграція даної моделі в архітектуру агрегатора спирається на три економічні фактори:

– еластичність попиту за часом. Споживачі формують чіткі цикли активності, створюючи «години пік» та спади у вихідні дні. Статична ціна порушує баланс: у пікові періоди попит перевищує пропозицію, зумовлюючи черги, а в періоди спаду ресурси фахівця простоюють. Динамічна ціна дозволяє згладжувати ці коливання;

– обмеженість ресурсів (Perishable Inventory). Адміністратор не може зберегти часовий слот на складі з метою його подальшої реалізації. Порожній інтервал у розкладі спричиняє прямі збитки. Алгоритм автоматично знижує ціну в моменти низького попиту, що мотивує клієнтів створювати бронювання;

– максимізація доходу (Revenue Management). Математичні моделі оптимізують фінансові показники, забезпечуючи зростання загального обсягу замовлень та утримання високої маржинальності кожного запису [2].

В основу системи покладено адитивно-мультиплікативну модель, оскільки прості лінійні алгоритми не забезпечують необхідної точності коригування. Розроблена математична модель обчислює три незалежні параметри одночасно:

– базову вартість конкретної послуги;

– коефіцієнт циклічного часу, який використовує гармонічні функції;

– коефіцієнт терміновості та попиту, який спирається на властивості логістичної кривої.

Ця формула плавно змінює підсумковий тариф. Відсутність різких цінових стрибків зберігає лояльність користувачів, які взаємодіють із системою через Telegram-бот. Клієнти отримують економічно обґрунтовані пропозиції, а підприємство повністю автоматизує фінансову політику, усуваючи необхідність ручного моніторингу ринкової ситуації адміністратором.

1.4 Розробка математичної моделі формування вартості послуг

Алгоритм динамічного ціноутворення спирається на математичну модель, яка обчислює вартість бронювання в режимі реального часу залежно від двох незалежних змінних: часу запису (розподіл значень часового коефіцієнта наведено на рис. 1.1) та поточного завантаження провайдера.

Застосування адитивно-мультиплікативного підходу зумовлене тим, що суто мультиплікативні моделі, які перемножують усі коефіцієнти, провокують експоненційне та економічно необґрунтоване зростання ціни у разі збігу кількох пікових факторів.

На основі класичних принципів управління дохідністю (Yield Management) [1] та концепції маржинального ціноутворення [6], у роботі запропоновано авторську адитивно-мультиплікативну модель.

Базова функція ціноутворення виглядає так (1.1):

$$P_{dynamic} = P_{base} \cdot \left(1 + K_{(time)(t)} + K_{(demand)(d)}\right), \quad (1.1)$$

де P_{dynamic} - фінальна динамічна вартість послуги;

P_{base} - номінальна базова вартість послуги, яку встановлює адміністратор;

$K_{\text{time}}(t)$ - функція часового коефіцієнта для конкретної години t;

$K_{\text{demand}}(d)$ - функція коефіцієнта попиту для поточного рівня завантаженості.

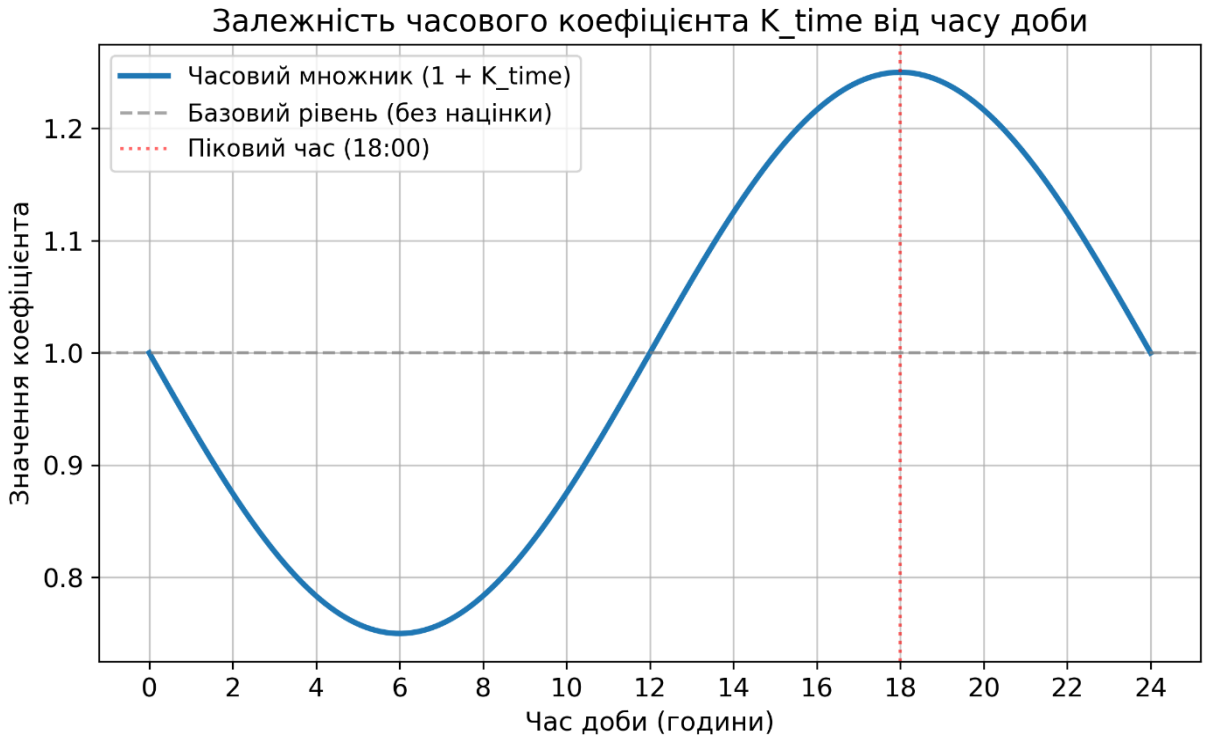

Рисунок 1.1 – Залежність часового коефіцієнта від години запису

Клієнтський попит протягом доби генерує виражені цикли. Щоб математично описати цей процес, ми застосували гармонічну функцію на базі косинуса. Цей інструмент плавно коригує ціну Графік залежності коефіцієнта попиту від рівня завантаженості майстра наведено на рис. 1.2.

Формула часового коефіцієнта (1.2):

$$C_{(time)(t)} = 1 + A \cdot \cos\left(\frac{2\pi \cdot (t - t_{peak})}{T}\right), \quad (1.2)$$

де A - амплітуда коливання ціни (наприклад, $A = 0.15$);

t - поточна година бронювання;

t_{peak} - година максимального пікового попиту (наприклад, 18:00);

T - період циклу (для добового циклу $T = 24$).

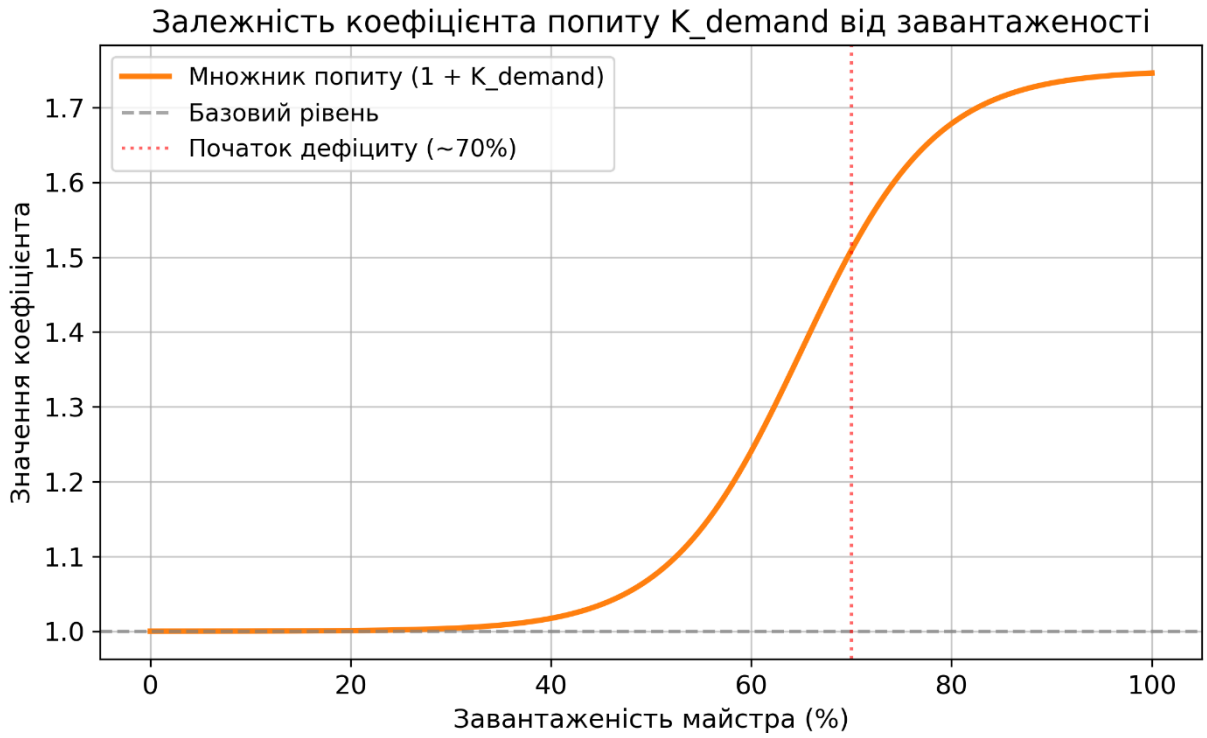

Рисунок 1.2 – Залежність коефіцієнта попиту від завантаженості майстра

Рівень завантаженості впливає на ціну нелінійно. Коли вільний час фахівця стає дефіцитом, система повинна обмежити попит. Дане завдання вирішується за допомогою логістичної функції (сигмоїди), математичні властивості якої детально описані у [7]. Вона забезпечує стрімке збільшення націнки після проходження точки перегину.

Формула коефіцієнта попиту (1.3):

$$C_{(demand)(d)} = 1 + \frac{L}{1 + e^{-k \cdot (d - d_0)}}, \quad (1.3)$$

де L - максимальний ліміт націнки за попит;

k - коефіцієнт крутизни кривої;

d - поточний показник завантаженості (від 0 до 1);

d_0 - точка перегину функції (рівень завантаженості, після якого алгоритм активно піднімає ціну).

Поділ алгоритму на дві незалежні функції є архітектурним рішенням, що забезпечує гнучке налаштування бізнес-логіки розподіленої системи під специфіку будь-якої сфери послуг.

1.5 Формалізація алгоритму розрахунку ціни

Алгоритм роботи програмного модуля ціноутворення формалізовано на основі адитивно-мультиплікативної моделі, а його загальну структуру у вигляді блок-схеми представлено на рис. 1.3.

Цей алгоритм функціонує як окремий сервісний компонент бізнес-логіки. Система викликає його під час запиту користувачем переліку доступних слотів або створення нового бронювання.

Функціонування зазначеного алгоритмічного модуля базується на обробці вхідного масиву параметрів у реальному часі (on-the-fly), що висуває жорсткі вимоги до обчислювальної ефективності процедури. Основними системними тригерами для ініціалізації розрахунку виступають дві події: генерація інтерактивного календаря зайнятості в інтерфейсі клієнтського бота та верифікація фінальної вартості транзакції безпосередньо перед записом транзакційних даних у реляційне сховище. Таке дублювання викликів дозволяє нівелювати ризики актуалізації застарілих тарифів, якщо користувач здійснював вибір часового інтервалу протягом тривалого проміжку часу, за який загальний попит на послуги провайдера встиг суттєво змінитися.

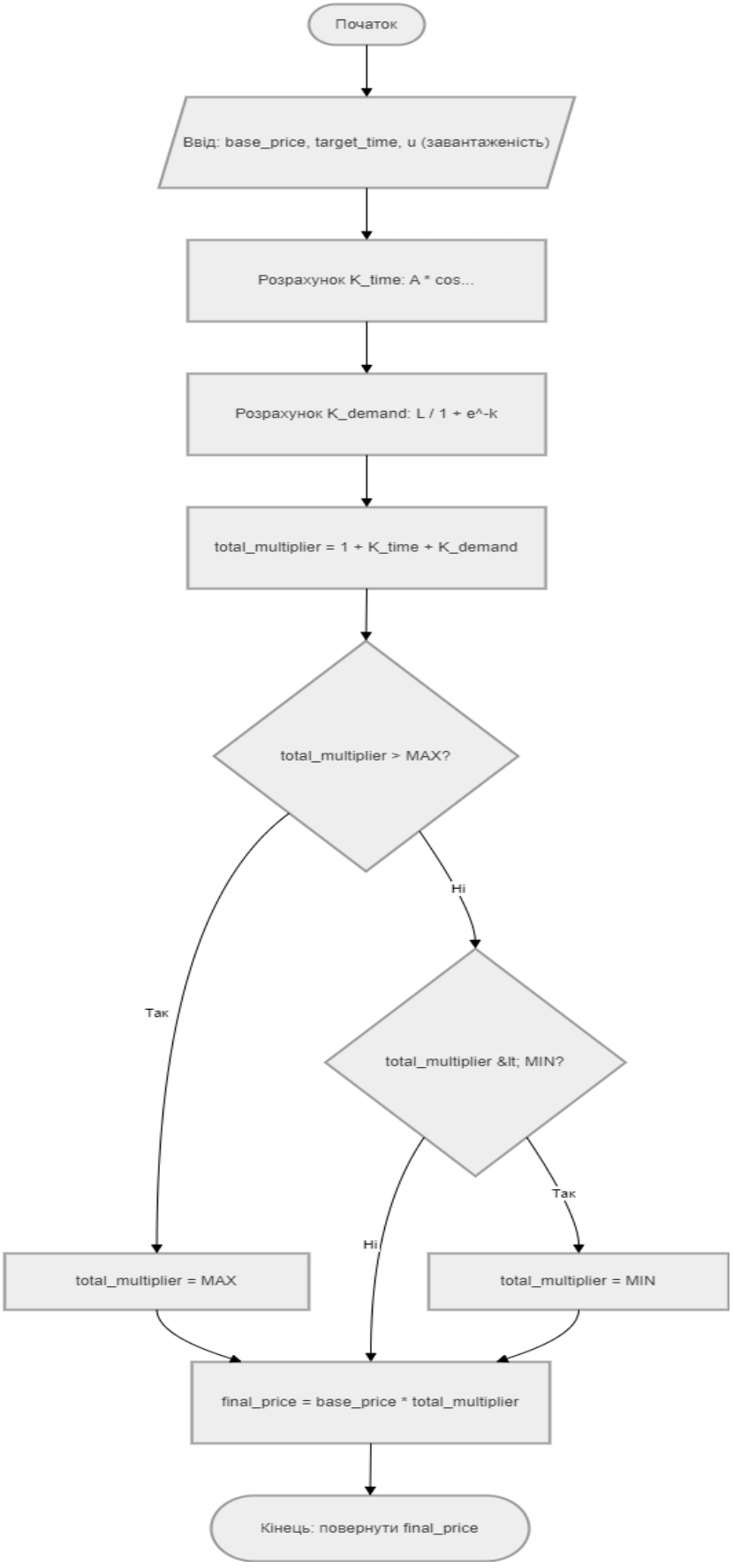

Рисунок 1.3 – Блок-схема алгоритму розрахунку динамічної ціни

Процес обчислення вартості замовлення складається з п'яти етапів:

– збір вхідних даних. Система отримує з бази даних PostgreSQL базову вартість послуги P_base, завантажує конфігураційні параметри алгоритму (A, L, k, d_0), фіксує показники стану: час обраного слоту t та кількість уже заброньованих місць для розрахунку завантаженості d;

– обчислення часової складової. Модуль вираховує відхилення обраного часу від точки піку, застосовує гармонічну функцію та визначає коефіцієнт K_time для врахування добової циклічності активності споживачів;

– обчислення складової попиту. Алгоритм ділить кількість зайнятих слотів на загальну місткість ресурсу для отримання показника d, після чого обчислює коефіцієнт K_demand за допомогою логістичної функції;

– агрегація коефіцієнтів та розрахунок ціни. Модуль підсумовує коригувальний множник і формує фінальну динамічну ціну;

– валідація результату. Алгоритм перевіряє отриману ціну на відповідність жорстким лімітам, що блокує генерацію некоректних значень (надто низьких або завищених цін).

Формалізований алгоритм працює з обчислювальною складністю O(1). Висока швидкість виконання є критично важливою для асинхронного API, оскільки такий розрахунок ціни не блокує подієвий цикл та гарантує паралельну обробку вхідних запитів від Telegram-бота [8].

1.6 Висновки до розділу 1

У першому розділі проаналізовано предметну область, що підтверджує нагальну необхідність автоматизації опрацювання замовлень у сфері послуг. За результатами дослідження зроблено такі висновки:

– по-перше, ручне управління та статичне ціноутворення зумовлюють значні економічні втрати, тому для масштабування сервісного бізнесу підприємства повинні впроваджувати автоматизацію;

– по-друге, існуючі програмні рішення не задовольняють повною мірою потреби ринку: комплексні продукти виявляються занадто складними для малого бізнесу, а прості сервіси не пропонують інструментів для інтелектуального управління доходами;

– по-третє, обґрунтовано доцільність використання адитивно-мультиплікативної математичної моделі ціноутворення, яка плавно та об’єктивно коригує вартість послуг, адаптуючись до часу обраного слоту та поточного попиту;

– також формалізовано алгоритм розрахунку ціни, який формує міцний математичний фундамент для програмної реалізації відповідного модуля, гарантує автономну роботу системи та мінімізує втручання адміністратора.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ ТА БАЗИ ДАНИХ

У цьому розділі здійснено проєктування загальної архітектури програмного комплексу. Насамперед визначено оптимальний стек технологій для серверної та клієнтської частин. Наступним етапом є розроблення реляційної моделі бази даних та інтеграція механізмів захисту від конкурентного доступу. Також сформовано подієво-орієнтовану модель взаємодії всіх компонентів, загальну структурну схему якої представлено на рис. 2.1.

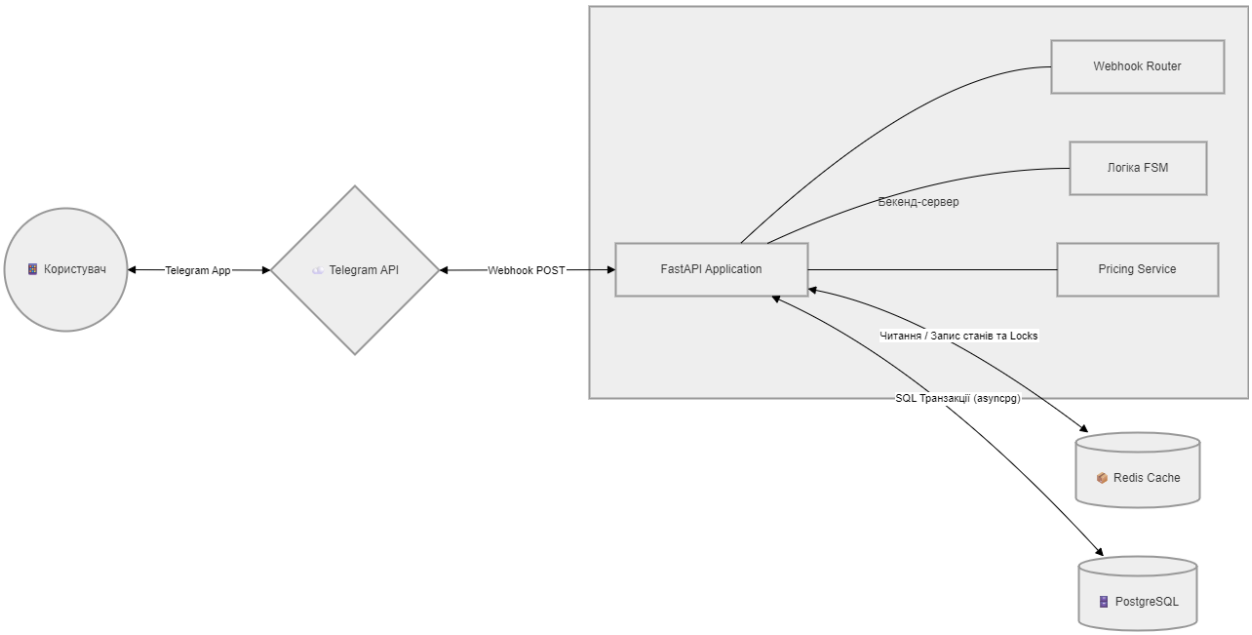

Рисунок 2.1 – Загальна архітектура системи

2.1 Вибір та обґрунтування технологічного стека

З метою вибору інструментів для бекенду і клієнтського інтерфейсу проведено порівняння сучасних мов програмування та спеціалізованих фреймворків. Головними критеріями стали загальна продуктивність, швидкість розробки (Time-to-Market) та наявність готових бібліотек для взаємодії з Telegram API.

2.1.1 Вибір мови програмування:Python та його аналоги

Розглянуто три індустріальні платформи для веброзробки: Node.js (JavaScript/TypeScript), Go (Golang) та Python.

Node.js (JavaScript/TypeScript). Платформа використовує асинхронну подієво-орієнтовану архітектуру (Event Loop), що забезпечує ефективне опрацювання задач типу I/O-bound. Неблокуючий ввід-вивід гарантує високу пропускну здатність. Проте екосистема для розроблення Telegram-ботів (наприклад, telegraf.js) функціонально поступається рішенням на Python, а специфіка роботи JavaScript із числами з плаваючою комою ускладнює точні математичні розрахунки в модулі ціноутворення.

Go (Golang). Компільована мова зі статичною типізацією від компанії Google, яка має вбудовані механізми багатопотоковості (goroutines). Go демонструє видатну швидкодію та споживає мінімум оперативної пам'яті, що є ідеальним для високонавантажених мікросервісів. Однак високий поріг входу уповільнює прототипування та загальний час розробки, що для агрегатора середнього масштабу спричиняє надлишкове інженерне ускладнення (overengineering).

Python. Інтерпретована мова загального призначення. Версії Python 3.7+ підтримують потужну асинхронну модель через модуль asyncio, що вирішує історичні проблеми продуктивності (GIL). Наявна екосистема для Telegram-ботів нативно підтримує машини станів (FSM), а вбудована статична типізація (Type Hinting) зменшує кількість помилок. Математичні бібліотеки дозволяють швидко програмно реалізувати алгоритм ціноутворення. Хоча швидкість виконання (CPU-bound) є нижчою, ніж у Go, розроблювана система належить до класу I/O-bound (сервер очікує відповіді від бази даних або Telegram API), що нівелює цей недолік.

Враховуючи вищезазначене, як найоптимальнішу мову для проєкту обрано Python, що дозволяє застосувати сучасні архітектурні патерни, чітко

розділити бізнес-логіку та транспортний рівень, а також гарантує високу швидкість розробки [9].

2.1.2 Вибір вебфреймворку: FastAPI, Express.js та Flask

Для створення серверної частини (API) агрегатора проведено порівняння трьох популярних інструментів: Express.js (Node.js), Flask та FastAPI (Python).

Express.js є мінімалістичним фреймворком зі швидким опрацюванням запитів, однак відсутність вбудованої валідації даних та автоматичної документації вимагає підключення сторонніх бібліотек, що ускладнює розробку.

Flask є класичним мікрофреймворком, що відрізняється простотою, проте його синхронна природа (один потік опрацьовує лише один запит) є критичним недоліком для систем із частими зверненнями до зовнішніх сервісів.

FastAPI - це сучасний асинхронний фреймворк, швидкодія якого завдяки інструментам Starlette та Pydantic конкурує з Node.js та Go. Він автоматично перевіряє типи вхідних даних та самостійно генерує специфікацію OpenAPI (Swagger), що суттєво прискорює тестування API.

Головним технологічним інструментом обрано FastAPI, який успішно поєднує швидкість роботи Node.js та потужність екосистеми Python [5].

2.1.3 Вибір бібліотеки для Telegram-ботів: aiogram та аналоги

Проаналізовано три спеціалізовані бібліотеки для побудови клієнтського інтерфейсу:

– Telegraf (Node.js) - потужна бібліотека, що має складну реалізацію машин станів (FSM), які є необхідними для логіки багатокрокового бронювання;

– python-telegram-bot (Python) - одна з найстаріших бібліотек, що має високу стабільність, але навіть після впровадження асинхронності її архітектура залишається перевантаженою та менш гнучкою;

– aiogram 3.x (Python) - повністю асинхронна бібліотека, орієнтована на швидкість та чистоту коду, що пропонує вбудовану машину станів (FSM) та нативну підтримку тип-хінтів.

Вибір зроблено на користь aiogram 3.x, оскільки цей фреймворк забезпечує найвищу продуктивність для розробки складних інтерфейсів.

2.1.4 Вибір середовища розробки (IDE): VS Code PyCharm

Для написання та відлагодження програмного коду розглянуто два популярні інструменти:

– PyCharm (JetBrains) - спеціалізована IDE, що надає глибокий аналіз коду, але має високі вимоги до апаратних ресурсів та потребує платної версії для повної підтримки FastAPI та Docker;

– Visual Studio Code (Microsoft) - легковаговий редактор коду з можливістю розширення, що відрізняється високою швидкістю роботи, гнучкістю та найкращою інтеграцією з контейнерами Docker і віддаленими серверами.

Visual Studio Code обрано через його універсальність, низьке споживання ресурсів та підтримку асинхронного стека Python.

2.1.5 Вибір СУБД: PostgreSQL, MySQL та MongoDB

Оскільки система бронювання постійно опрацьовує фінансові транзакції, СУБД має гарантувати високу надійність збереження даних.

MongoDB (NoSQL) слабко захищає транзакційну цілісність порівняно з реляційними аналогами, що змусило б реалізовувати складні алгоритми блокувань на рівні додатка (Application Level).

MySQL (SQL) суворо дотримується вимог ACID, проте відсутність нативних діапазонних типів даних критично ускладнює пошук перетинів часових інтервалів (вимагає створення двох окремих колонок для початку та кінця слоту).

PostgreSQL (Об'єктно-реляційна СУБД) пропонує найбільш просунутий інструментарій, маючи дві ключові інженерні переваги:

– діапазонні типи (tsrange), які дозволяють фіксувати часовий проміжок як єдине атомарне значення всередині одного поля;

– обмеження виключення (Exclusion Constraints) на базі просторових індексів GiST, завдяки яким система на фізичному рівні самостійно блокує додавання рядків з часовими діапазонами, що перетинаються.

Отже, СУБД PostgreSQL визначено як найбільш технологічне та оптимальне рішення для побудови надійного календаря бронювання.

2.1.6 Вибір in-memory сховища: Redis та Memcached

З метою управління станами сесій клієнтів у Telegram-боті та кешування результатів алгоритму ціноутворення порівняно два рішення:

Memcached - високопродуктивна система кешування, що підтримує лише просту структуру даних формату «рядок-рядок», без можливості зберігати складні об'єкти без попередньої серіалізації.

Redis - структуроване сховище (Advanced Key-Value Store), що нативно підтримує складні структури даних (Hashes, Sets, Lists), що робить його ідеальним рушієм для машини станів (FSM) у фреймворку aiogram [10]. Додатково Redis записує дані на диск (Persistence), що запобігає втраті активних діалогів.

Redis обрано як обов'язковий архітектурний компонент бекенду, який гарантує стабільну роботу бота та підвищує швидкодію API.

2.2 Проєктування реляційної моделі даних у 3НФ

Правильно спроєктована структура бази даних є запорукою стабільності інформаційної системи. Архітектуру даних розробленого агрегатора побудовано на основі реляційної моделі із суворим дотриманням правил нормалізації до третьої нормальної форми (3НФ). Цей підхід усуває надмірність даних, захищає сховище від аномалій оновлення та гарантує логічну цілісність транзакцій.

Процес нормалізації до 3НФ виконано за три етапи:

– перша нормальна форма (1НФ): усі атрибути таблиць зведено до атомарних (наприклад, контакти користувача та його Telegram ID займають окремі поля);

– друга нормальна форма (2НФ): забезпечено повну залежність неключових атрибутів від первинного ключа шляхом створення окремих сутностей для клієнтів, майстрів та довідника послуг;

– третя нормальна форма (3НФ): ліквідовано транзитивні залежності (наприклад, алгоритм зберігає базову ціну у проміжній таблиці надання послуг, оскільки вартість визначається кваліфікацією конкретного спеціаліста).

Логічна схема бази даних містить п'ять ключових взаємопов'язаних таблиць:

− таблиця «users» (Клієнти) – зберігає персональні та контактні дані користувачів платформи (детальну специфікацію атрибутів наведено в табл. 2.1). Атрибути: user_id (Primary Key, UUID), telegram_id (Bigint), full_name (Varchar), phone (Varchar), email (Varchar), created_at (Timestamp);

− таблиця «providers» (Майстри / Провайдери послуг) – зберігає дані про спеціалістів. Система використовує ці індивідуальні параметри майстра для розрахунку динамічної ціни. Атрибути: provider_id (Primary Key, UUID), name (Varchar), experience (Integer), rating (Numeric), peak_hour (Numeric), working_days (Varchar);

Таблиця 2.1 – Структура реляційної таблиці «users» (Клієнти)

Назва атрибуту (Поле)	Тип даних у СУБД	Обмеження цілісності	Опис призначення поля
user_id	UUID	PRIMARY KEY	Унікальний системний ідентифікатор клієнта
telegram_id	BIGINT	UNIQUE, NOT NULL	Ідентифікатор користувача в екосистемі Telegram
full_name	VARCHAR(255)	NOT NULL	Прізвище, ім'я та по батькові користувача
phone	VARCHAR(20)	UNIQUE, NULLABLE	Контактний номер телефону для верифікації
email	VARCHAR(255)	UNIQUE, NULLABLE	Електронна пошта клієнта (опціонально)
created_at	TIMESTAMP (TZ)	NOT NULL	Дата та час реєстрації клієнта в системі

– таблиця «services» (Довідник послуг) – містить незмінні базові характеристики послуг платформи. Атрибути: service_id (Primary Key, UUID), title (Varchar), description (Varchar), duration_interval (INTERVAL);

– таблиця «provider_services» (Надання послуг) – реалізує зв'язок «багато-до-багатьох» між майстрами та послугами. Забезпечує вимоги 3НФ. Атрибути: ps_id (Primary Key, UUID), provider_id (Foreign Key), service_id (Foreign Key), base_price (Numeric), is_active (Boolean);

– таблиця «bookings» (Бронювання) – центральний транзакційний вузол системи. Фіксує запис клієнта до майстра на обрану послугу та зберігає результати ціноутворення (детальну специфікацію атрибутів наведено в табл. 2.2). Атрибути: booking_id (UUID), user_id (FK), ps_id (FK), booking_period (TSRANGE), final_price (Numeric), status (Varchar), reminder_sent (Boolean).

Загальну ER-діаграму цих 5 таблиць наведено на рис. 2.2.

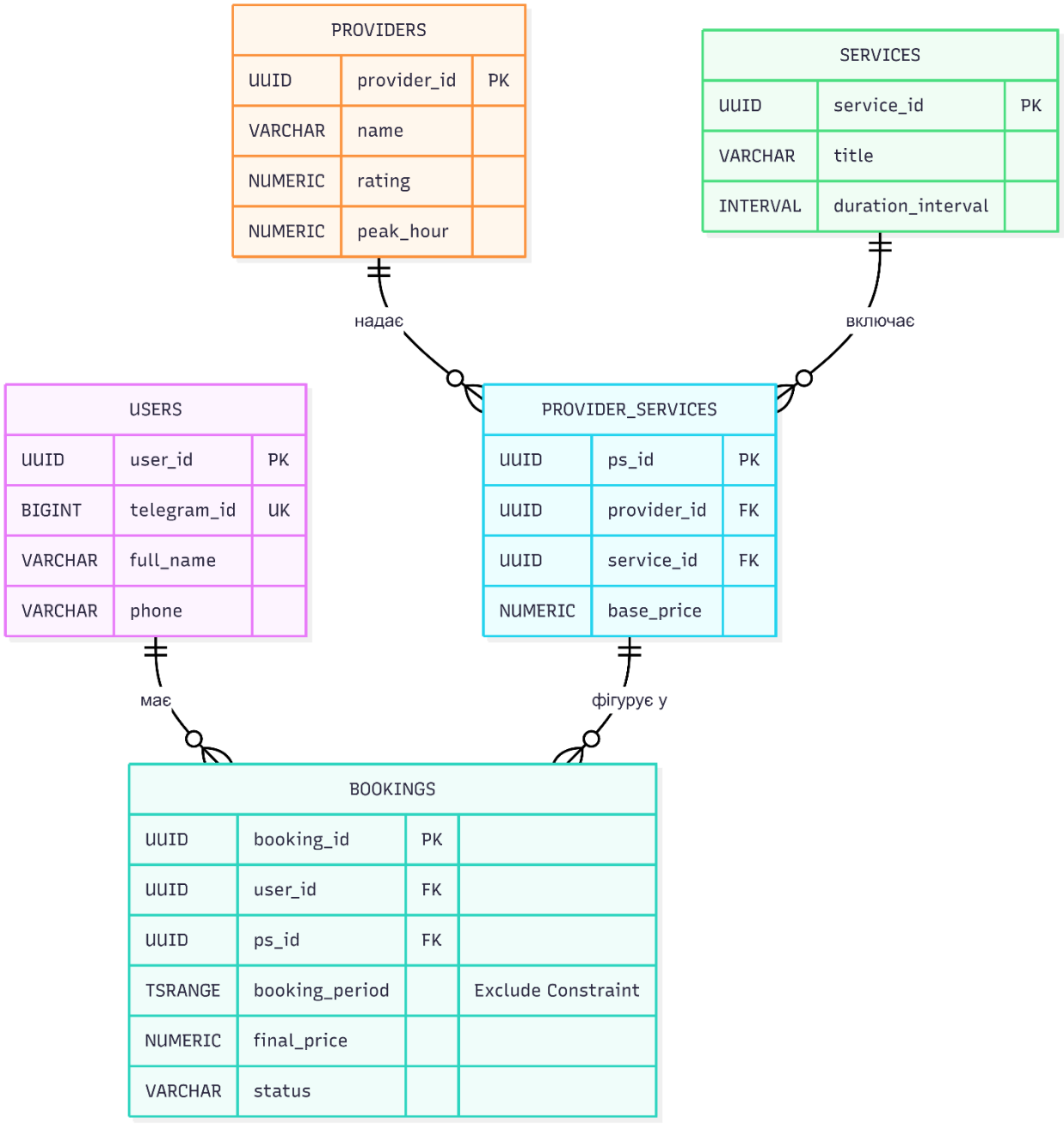

Рисунок 2.2 – ER-діаграма бази даних

Архітектура захищена від аномалій видалення (Delete Anomalies) завдяки обмеженням ON DELETE RESTRICT на рівні СКБД. Це блокує випадкове видалення активних послуг клієнта чи провайдерів, а в таблиці bookings надійно зберігається історичний зріз фінансових даних (final_price) на момент транзакції. Логічна цілісність та зв'язність даних підтримується на рівні СКБД за допомогою обмежень зовнішніх ключів (Foreign Keys) із декларативними правилами ON DELETE RESTRICT.

Таблиця 2.2 – Структура реляційної таблиці «bookings» (Бронювання)

Назва атрибуту (Поле)	Тип даних у СУБД	Обмеження цілісності	Опис призначення поля
booking_id	UUID	PRIMARY KEY	Унікальний системний ідентифікатор транзакції запису
user_id	UUID	FOREIGN KEY -> users	Зв'язок із таблицею зареєстрованих клієнтів
ps_id	UUID	FOREIGN KEY -> provider_services	Зв'язок із конкретною послугою конкретного майстра
booking_period	TSRANGE	EXCLUDE (GiST)	Діапазон часу (початок і кінець сеансу) з контролем накладання
final_price	NUMERIC(10, 2)	NOT NULL	Фінальна динамічна вартість послуги на момент запису
status	VARCHAR(20)	NOT NULL	Поточний статус сесії (CONFIRMED, CANCELLED)
reminder_sent	BOOLEAN	DEFAULT FALSE	Прапорець відправки автоматичного фонового нагадування
created_at	TIMESTAMP (TZ)	NOT NULL	Час та дата фізичного створення транзакції

Розроблена логічна структура гарантує швидке виконання SQL-запитів під час пошуку вільних слотів через просторові GiST-індекси.

2.3 Структура календаря бронювання з типами даних tsrange

Управління часовими ресурсами під час проєктування систем автоматизації замовлень є складним завданням. Використання двох окремих полів (start_time та end_time) ускладнює SQL-запити для пошуку перетинів та не дозволяє ефективно індексувати дані. З огляду на це, обрано об’єктно-реляційний підхід із застосуванням вбудованого в PostgreSQL діапазонного типу даних - tsrange (Timestamp Range) [11].

Використання типу tsrange забезпечує три інженерні переваги:

– атомарність сутності: база даних розглядає часовий проміжок як єдиний об'єкт, що оптимізує логіку роботи ORM-шару;

– нативна підтримка операторів алгебри множин: наявні вбудовані оператори для обробки діапазонів (&& - перетин, @> - включення, * - обчислення перетину);

– висока продуктивність пошуку: діапазонні типи підтримують індексацію типу GiST (Generalized Search Tree), яка ефективно обробляє просторово-часові запити.

2.3.1 Логічна структура часового слота

У таблиці bookings поле booking_period зберігає інтервал у форматі [start, end). Квадратна дужка означає включення межі, а кругла - її виключення, що запобігає часовим аномаліям на стиках слотів (наприклад, завершення одного та початок іншого замовлення о 14:00 не фіксується як конфлікт). Візуалізацію оператора перетину діапазонів наведено на рис. 2.3.

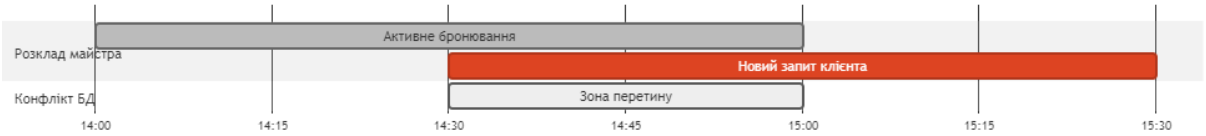

Рисунок 2.3 – Візуалізація оператора перетину діапазонів (tsrange overlap)

Комбінація типу tsrange та асинхронного обробника на FastAPI дозволяє виконувати перевірку доступності майстра безпосередньо всередині SQL-запиту без завантаження зайвих масивів даних у пам’ять Python-додатка. Це мінімізує мережеві затримки та знижує навантаження на CPU сервера. Таке рішення підвищує загальну масштабованість системи опрацювання замовлень.

Головною архітектурною перевагою використання tsrange та індексів GiST є впровадження обмеження виключення (Exclude Constraint) на рівні схеми бази даних (через оператор перетину &&). СУБД автоматично скасовує транзакції INSERT або UPDATE, якщо новий слот перетинається з активним бронюванням майстра.

Цей підхід гарантує абсолютну консистентність розкладу, повністю ліквідовуючи проблему стану гонитви (race conditions). Перевірка цілісності виконується ядром PostgreSQL за математичний час O(log N), що є значно надійнішим за програмні перевірки на рівні бекенду.

2.4 Архітектурне розв'язання проблеми стану гонитви

Системи масового бронювання стикаються з проблемою конкурентного доступу (race condition), коли кілька користувачів одночасно намагаються зайняти однаковий часовий слот. За відсутності надійних механізмів синхронізації база даних генерує аномалію «втраченого оновлення» (lost update) [4]-[6], спричиняючи дублювання записів, що є неприпустимим.

2.4.1 Транзакційні аномалії та рівні ізоляції у PostgreSQL

Проаналізовано стандартні рівні ізоляції транзакцій у СУБД PostgreSQL [11]:

– Read Committed (за замовчуванням): кожна операція бачить лише зафіксовані дані на момент початку запиту, що не захищає від перетину інтервалів;

– Repeatable Read: блокує нетипове читання, проте дозволяє декільком транзакціям успішно завершитися за умови модифікації різних рядків (що й відбувається під час створення нових записів);

– Serializable: гарантує повну ізоляцію, проте суттєво знижує продуктивність бази даних, генеруючи значну кількість помилок серіалізації та вимагаючи впровадження логіки повторного виконання запитів (retry logic).

Результати аналізу доводять, що застосування виключно рівнів ізоляції не є ефективним рішенням архітектурної проблеми.

2.4.2 Обґрунтування обмежень виключення

Для гарантування абсолютної консистентності даних на фізичному рівні в архітектуру системи інтегровано механізм обмежень виключення (Exclusion Constraints) у поєднанні з просторовими індексами GiST [11]. Теоретичну основу застосованого підходу до керування конкурентним доступом (concurrency control) складають концепції, описані у [4].

На відміну від класичного обмеження UNIQUE, яке перевіряє лише точну рівність значень, обмеження EXCLUDE дозволяє задати довільний оператор порівняння - зокрема, оператор перетину діапазонів (&&), що є критично важливим для захисту часових слотів від накладання.

Формальний опис цього обмеження для таблиці bookings реалізовано засобами SQL:

```
ALTER TABLE bookings
ADD CONSTRAINT prevent_overlapping_bookings
EXCLUDE USING gist (
  ps_id WITH =,
  booking_period WITH &&
) WHERE (status != 'CANCELLED');
```

Ядро СУБД обробляє це обмеження за такою логікою:

– ps_id WITH = - перевіряється рівність ідентифікатора (пошук конкретного майстра та послуги);

– booking_period WITH && - перевіряється перетин часових діапазонів через оператор &&;

– WHERE (status != 'CANCELLED') - ігноруються скасовані записи, що дозволяє миттєво звільнити слот.

Таким чином, логіку синхронізації повністю перенесено з прикладного рівня на максимально швидкий рівень ядра бази даних.

2.4.3 Система ізоляції: Redis Soft Lock та PostgreSQL Hard Lock

Оскільки процес створення запису клієнтом потребує часу (вибір слоту, оплата, підтвердження), використання лише обмежень PostgreSQL здатне погіршити користувацький досвід (UX). Для запобігання ситуаціям, коли слот займається іншим клієнтом під час оплати, розроблено дворівневу систему ізоляції конкурентних запитів:

– Soft Lock (Рівень кешування): під час вибору вільного часу бекенд створює тимчасовий ключ у Redis формату lock:slot:{ps_id}:{start_time} із часом життя (TTL) 300 секунд [10], що приховує слот з пошукової видачі та гарантує клієнту безпечний коридор для транзакції;

– Hard Lock (Рівень бази даних): після успішної оплати запис транзакції здійснюється в PostgreSQL. У випадку мережевого збою обмеження ExcludeConstraint спрацьовує як фінальний рубіж захисту від дублювання, після чого сервер примусово видаляє тимчасовий ключ із Redis.

2.5 Проєктування архітектури взаємодії через Webhook

Взаємодію між клієнтським інтерфейсом (Telegram) та сервером (FastAPI) побудовано на основі подієво-орієнтованої архітектури (Event-Driven Architecture) [12]. З двох методів отримання оновлень (Long Polling та Webhook) обрано архітектуру Webhook для експлуатаційного середовища (production). На відміну від Long Polling, який створює безперервне фонове навантаження на мережу, Webhook реалізує push-модель [13]: сервери Telegram самостійно відправляють асинхронний HTTP POST-запит на визначений маршрут бекенду лише в момент настання події, що гарантує миттєву реакцію системи та ефективно масштабується.

Життєвий цикл обробки події складається з п'яти етапів, загальну діаграму послідовності (Sequence Diagram) яких наведено на рис. 2.4:

– генерація події (клієнт взаємодіє з ботом, обираючи вільний часовий слот);

– доставка (платформа Telegram формує JSON-об'єкт із даними про подію та надсилає його на маршрут /webhook);

– маршрутизація (FastAPI приймає запит та передає об'єкт у диспетчер aiogram через асинхронний метод feed_update);

– виконання бізнес-логіки (диспетчер аналізує стан діалогу FSM з Redis, викликає обробник, після чого бекенд звертається до PostgreSQL або модуля ціноутворення);

– завершення транзакції (сервер формує відповідь та повертає статус HTTP 200 OK, підтверджуючи успішне опрацювання події).

Такий підхід забезпечує повністю неблокуючу (non-blocking) обробку запитів, що дозволяє бекенду одночасно опрацьовувати значний обсяг вхідних подій без перевантаження циклу подій (event loop). Крім того, делегування управління станами діалогів до індустріального сховища Redis робить архітектуру сервера незалежною від стану (stateless).

Комбінація механізму Webhook та асинхронності FastAPI дозволяє системі одночасно обробляти сотні запитів. Сервер не блокує основний потік виконання коду. Це гарантує високу швидкість відгуку інтерфейсу бота.

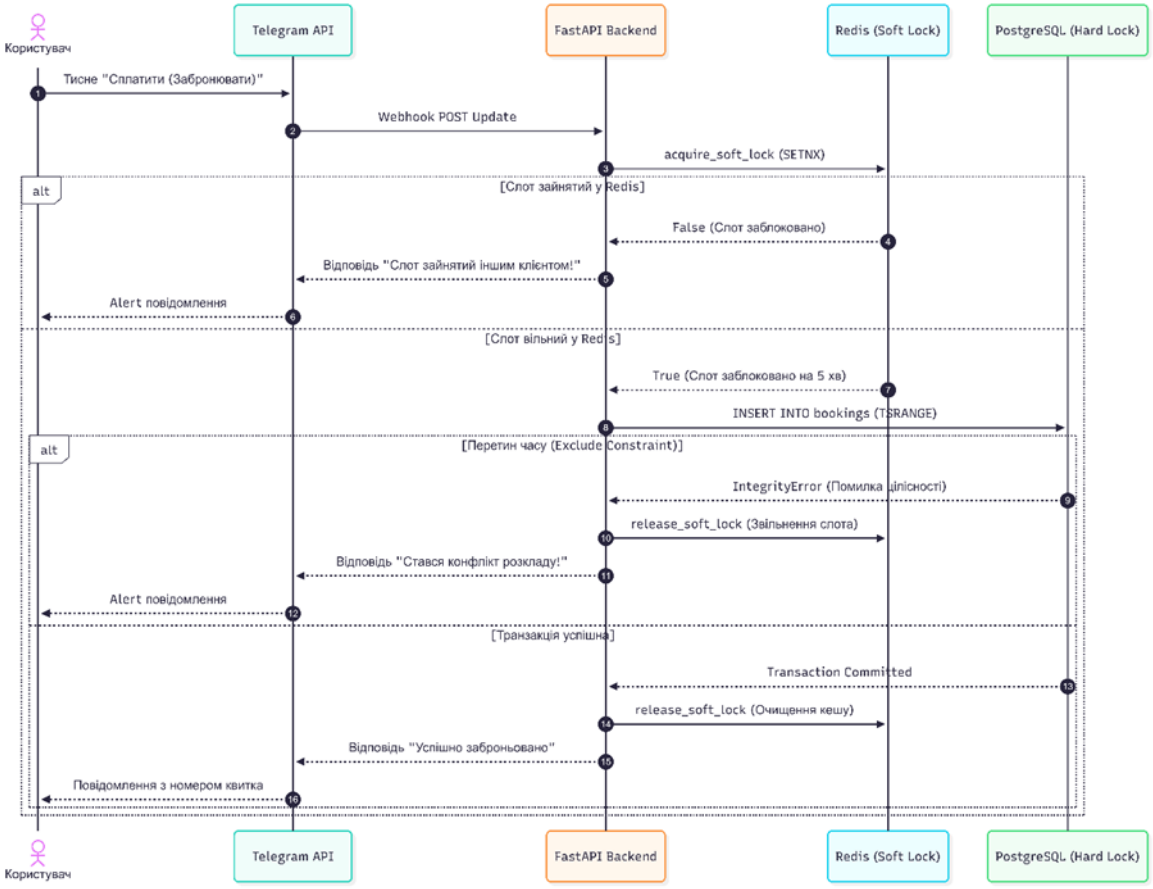

Рисунок 2.4 – Діаграма послідовності (Sequence Diagram) обробки бронювання

2.6 Контейнеризація та розгортання системи через Docker

З метою забезпечення абсолютної ідентичності середовищ розробки, тестування та експлуатації застосовано технологію контейнеризації на базі Docker [14]. Це рішення забезпечує чотири головні переваги:

– ізоляція залежностей: бекенд (FastAPI), база даних (PostgreSQL) та сховище (Redis) функціонують в окремих контейнерах, що унеможливлює конфлікти версій бібліотек;

– декларативне управління інфраструктурою: топологія мережі описана у конфігураційному файлі docker-compose.yml (Infrastructure as Code);

– оркестрація та мережева взаємодія: сервіси спілкуються у віртуальній мережі за локальними DNS-іменами, що підвищує безпеку та усуває необхідність відкриття портів бази даних для зовнішнього світу;

– швидке масштабування: контейнеризація дозволяє миттєво розгорнути додаткові екземпляри бекенду для адаптації до пікових навантажень.

Архітектурна схема розгортання охоплює три основні контейнери: app, db та cache.

2.7 Висновки до розділу 2

У другому розділі здійснено проєктування архітектури програмної системи та логічної структури бази даних. За результатами розробки зроблено такі висновки:

– обґрунтовано вибір технологічного стека, основу якого формують асинхронні інструменти (Python, FastAPI та aiogram 3.x), що є оптимальним рішенням для побудови високонавантажених подієво-орієнтованих сервісів;

– розроблено реляційну модель даних у третій нормальній формі (3НФ), яка усуває надмірність інформації та гарантує гнучкість системи агрегатора;

– інтегровано об’єктно-реляційний підхід для управління часом за допомогою діапазонних типів tsrange та обмежень виключення (Exclusion Constraints) на рівні ядра PostgreSQL, що гарантує абсолютний захист розкладу від станів гонитви;

– спроєктовано систему мережевої взаємодії через механізм Webhook, який миттєво доставляє події та раціональніше використовує ресурси сервера;

– визначено стратегію розгортання інфраструктури через Docker-контейнери, що забезпечує ізоляцію та стабільну роботу програми.

Спроєктована архітектура формує надійний інженерний фундамент та повністю готова до етапу програмної реалізації.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

Третій розділ присвячено безпосередній програмній реалізації розробленої архітектури. У процесі виконання роботи здійснено написання вихідного коду серверної та клієнтської частин, налаштування модульної структури проєкту та інтеграцію ключових алгоритмів бронювання й динамічного ціноутворення. Опис програмної реалізації наведено з дотриманням принципів об’єктно-орієнтованого програмування та сучасних стандартів асинхронної розробки мовою Python.

3.1 Реалізація асинхронного інтерфейсу на базі aiogram

Клієнтський інтерфейс розроблено у вигляді чат-бота в месенджері Telegram на базі асинхронного фреймворку aiogram 3.x [15]. Зазначений інструмент дозволив побудувати асинхронну подієво-орієнтовану логіку, завдяки якій система опрацьовує кожен запит користувача паралельно, не блокуючи основний потік виконання коду.

Логіку взаємодії розподілено на ізольовані модулі – обробники подій (хендлери), які відповідають за конкретні етапи сценарію. З метою збереження чистоти коду застосовано механізм маршрутизації (Router), який поділяє обробники на дві групи:

- командні обробники, призначені для опрацювання стартових команд (/start, /admin, /master);
- Callback-обробники, які забезпечують опрацювання натискань на Inline-кнопки. Даний інструмент виступає головним засобом взаємодії, оскільки передає структуровані дані у форматі CallbackData, усуваючи необхідність очікування текстового введення від користувача.

Валідація вхідних подій здійснюється через вбудовані фільтри. Зокрема, забезпечено сувору перевірку прав доступу (за ідентифікатором ADMIN_ID) для входу до панелі адміністратора, що дозволяє миттєво

відхиляти некоректні запити на початковому етапі виконання коду. Графічний інтерфейс вітального вікна користувача під час запуску системи представлено на рис. 3.1.

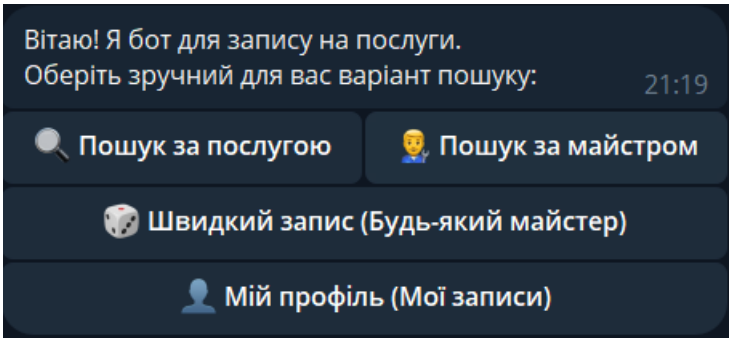

Рисунок 3.1 – Головне меню Telegram-бота агрегатора послуг (Скриншот команди /start)

Процес бронювання послуги функціонує як багатокроковий алгоритм, що потребує постійного збереження контексту діалогу: обраного майстра, цільової послуги та бажаного часу. Дане завдання вирішено шляхом інтеграції машини станів (FSM) на базі класу BookingFlow, структурний граф якої наведено на рис. 3.2. Диспетчер подій фіксує поточні стани та інформацію у високопродуктивному in-memory сховищі Redis, яке гарантує високу швидкодію та зберігає прогрес клієнта навіть у випадку незапланованого перезавантаження або відмови сервера.

Сценарій бронювання охоплює такі послідовні кроки:

- BookingFlow.choosing_service – вибір послуги з переліку активних пропозицій;
- BookingFlow.choosing_master – вибір конкретного провайдера (майстра) для надання обраної послуги;
- BookingFlow.choosing_date – визначення дати надання послуги через інтерактивний Inline-календар;
- BookingFlow.choosing_time – вибір доступного часового слоту, на етапі якого алгоритм динамічно розраховує актуальну вартість;

- BookingFlow.confirming – фінальне підтвердження замовлення та перехід до процесу оплати.

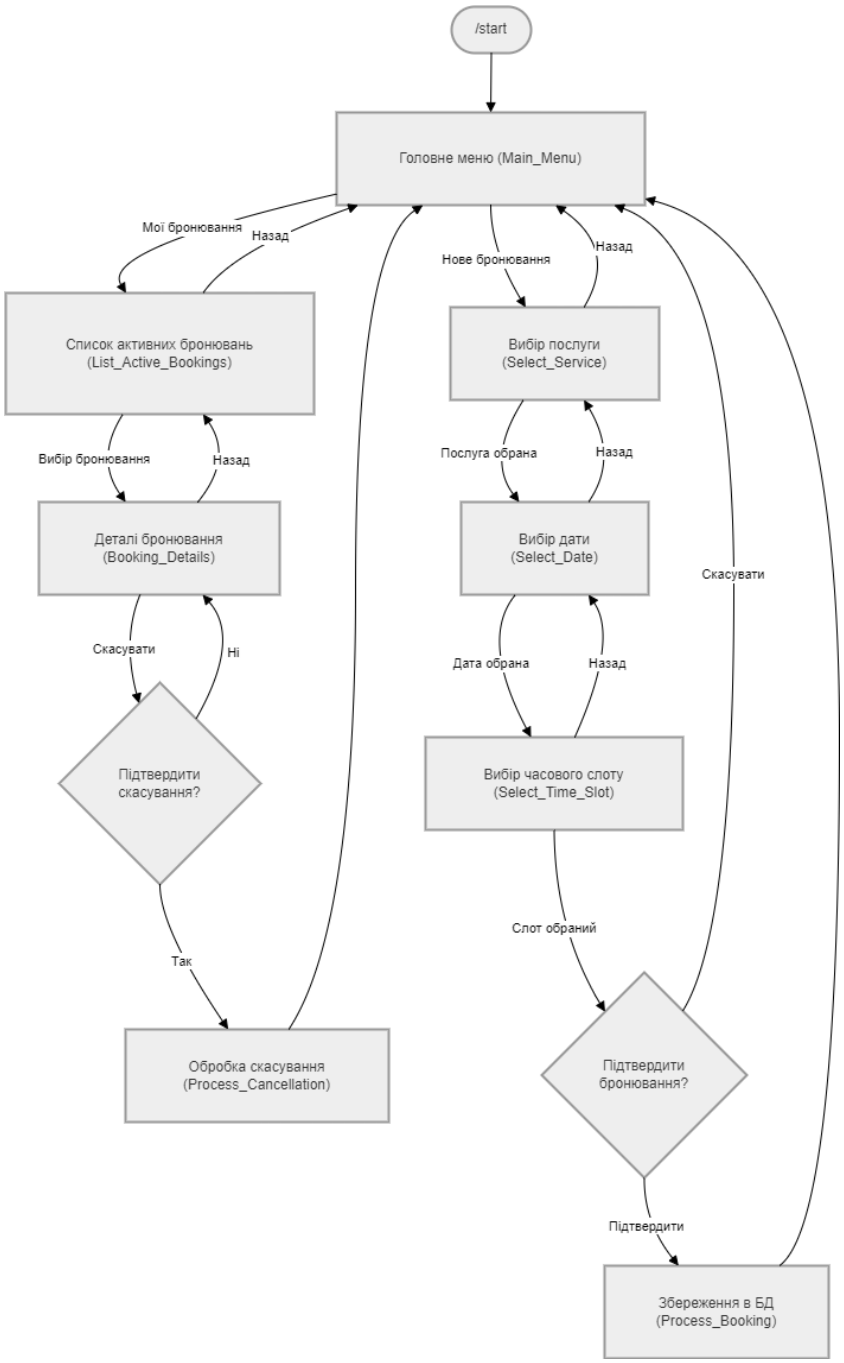

Рисунок 3.2 – Граф машини станів (FSM) Telegram

Формування клавіатур здійснюється в динамічному режимі шляхом завантаження актуальних даних із реляційної бази даних PostgreSQL. Система адаптує інтерфейс під кожного користувача в реальному часі за трьома правилами:

– під час вибору дати серверна частина перевіряє робочий графік майстра (working_days). У разі спроби взаємодії з вихідним днем система надсилає відповідне попередження та пропонує обрати іншу дату;

– програмний комплекс автоматично приховує або блокує зайняті часові інтервали на основі інформації, отриманої від механізму Soft Lock у Redis та типу даних tsrange у базі даних;

– вартість послуги виводиться безпосередньо на кнопках вибору часу, причому обчислення зазначених показників здійснюється за допомогою адитивно-мультиплікативної моделі ціноутворення.

Приклад динамічного відображення розрахованої вартості безпосередньо на елементах інтерфейсу вибору часових слотів наведено на рис. 3.3.

Рисунок 3.3 – Динамічне відображення вартості часових слотів

Для підтримки архітектурної цілісності бекенду та автоматизації рутинних операцій розроблено шар DbSessionMiddleware. Зазначений компонент самостійно створює асинхронну сесію ORM-фреймворку SQLAlchemy на початку обробки кожного запиту та передає її безпосередньо в обробник подій. Після завершення опрацювання події система автоматично

закриває сесію, що захищає сервер від витоків з'єднань із базою даних в умовах високого навантаження.

Взаємодія з Telegram API реалізована за допомогою ключових слів async та await (наприклад, під час виклику методу callback.message.edit_text()). Сервер очікує мережеву відповідь від зовнішнього сервера, не зупиняючи при цьому обробку запитів від інших клієнтів. Дане архітектурне рішення забезпечує масштабованість інтерфейсу агрегатора, дозволяючи підтримувати сотні активних діалогів одночасно при мінімальному споживанні апаратних ресурсів.

3.2 Розробка серверної частини на FastAPI та інтеграція з БД

Серверна частина програмної системи реалізована у вигляді асинхронного вебдодатка на базі фреймворку FastAPI. Бекенд базується на багатошаровій архітектурі (Layered Architecture), що забезпечує повну ізоляцію логіки взаємодії з базою даних від транспортного рівня API та компонентів обробки бізнес-процесів.

3.2.1 Реалізація асинхронного шару ORM (SQLAlchemy)

Для взаємодії з реляційною СУБД PostgreSQL застосовано інструмент об’єктно-реляційного відображення SQLAlchemy 2.0 та асинхронний драйвер asyncpg. Програмний код моделей даних повністю відтворює спроєктовану реляційну схему.

На рівні збереження даних (Data Access Layer) впроваджено три ключові технічні рішення:

- асинхронний контекст сесій, де контроль пулу підключень покладено на об’єкт async_sessionmaker, що мінімізує витрати ресурсів на відкриття нових TCP-з’єднань із СУБД;

- декларативне відображення моделей через класичний синтаксис SQLAlchemy з використанням об'єктів Column. Драйвер asyncpg нативно підтримує діапазонні типи даних PostgreSQL (tsrange) та автоматично трансформує інтервали бази в об'єкти Python під час виконання SQL-запитів, що забезпечує коректну роботу обмежень виключення;
- валідація через Pydantic, де моделі Pydantic v2 суворо типізують обмін даними між зовнішніми HTTP-запитами та API. Зазначений механізм гарантує автоматичну перевірку (Data Validation) вхідних параметрів до моменту виконання основної бізнес-логіки.

Перелік, конфігурацію та функціональне призначення основних API-ендпоінтів серверної частини, реалізованих на базі фреймворку FastAPI, наведено в табл. 3.1.

Таблиця 3.1 – Опис основних API-ендпоінтів серверної частини

Метод HTTP	Маршрут (Endpoint)	Вхідні параметри / Схема	Функціональне призначення маршруту
POST	/webhook	Update (JSON від Telegram API)	Прийом подій месенджера та асинхронна передача в диспетчер aiogram
POST	/bookings/	BookingRequest (Pydantic)	Ініціалізація транзакції бронювання та фіксація у СУБД PostgreSQL
GET	/health	Немає	Перевірка статусу працездатності (Health Check) серверного додатка

Для автоматизації процесу документування архітектури розроблених маршрутів та забезпечення можливості їх інтерактивного тестування в реальному часі застосовано вбудовані інструменти генерації специфікацій OpenAPI. Приклад візуального відображення інтерактивної специфікації серверного API у середовищі Swagger UI представлено на рис. 3.4.

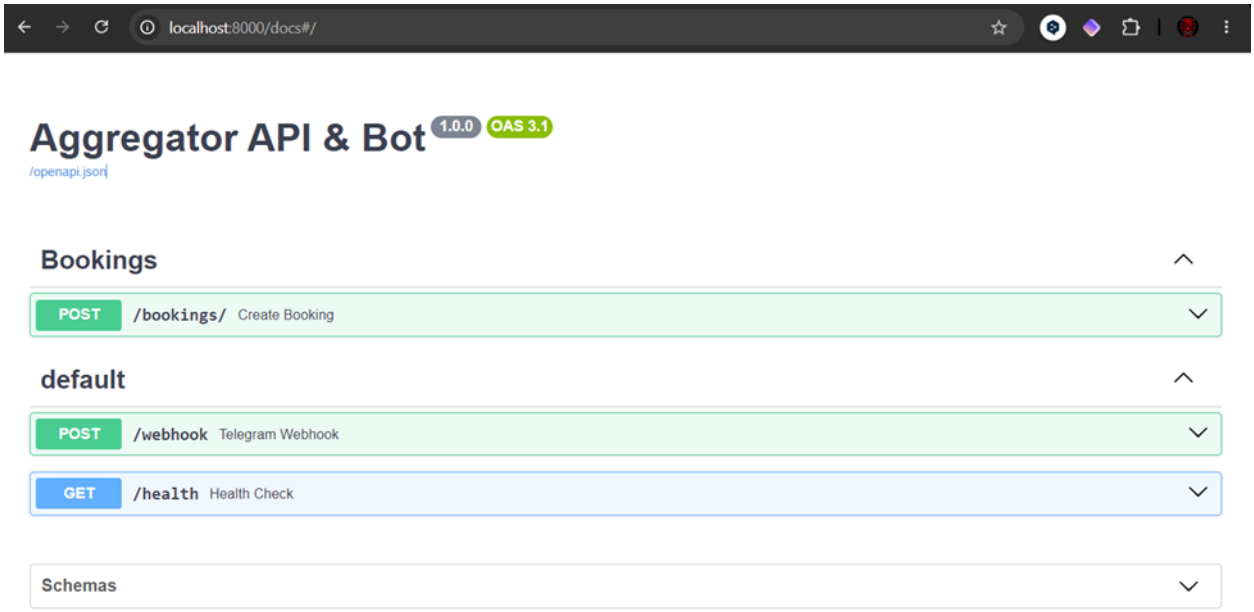

Рисунок 3.4 – Інтерактивна специфікація серверного API у середовищі Swagger UI

3.2.2 Реалізація Webhook-ендпоінту та ін’єкції залежностей

Виділений асинхронний ендпоінт /webhook приймає вхідні HTTP POST запити від екосистеми Telegram. Ми впровадили вбудований у FastAPI механізм ін’єкції залежностей (Dependency Injection) для оптимізації архітектури. Функція-генератор get_db_session повністю керує життєвим циклом транзакцій за таким алгоритмом:

– перед кожним викликом ендпоінту система автоматично створює ізольовану асинхронну сесію AsyncSession;

– додаток передає цю сесію в сервісні класи або інжектує її в middlewares бота для виконання операцій;

– після завершення обробки запиту система автоматично закриває сесію та повертає з'єднання в пул.

Якщо виникає помилка виконання (наприклад, порушення умов ExcludeConstraint при перетині часових слотів), база даних автоматично скасовує транзакцію (Rollback). Цей підхід надійно запобігає витокам пам'яті (Memory Leaks). Він гарантує транзакційну цілісність даних (ACID) у високонавантаженому середовищі.

3.3 Програмна реалізація сервісу PricingService

Обчислення вартості послуг у реальному часі інкапсульовано в окремий програмний модуль – клас PricingService. Математичний аппарат, обґрунтований у підрозділах 1.4 та 1.5, реалізовано за принципами високої зв'язності (High Cohesion) та низької зачепленості (Low Coupling), що дозволяє змінювати конфігураційні параметри алгоритму без жодної модифікації транспортного або клієнтського рівнів системи.

3.3.1 Алгоритмічне обчислення коефіцієнтів

Клас PricingService оперує набором статичних конфігураційних констант (GAMMA, LAMBDA, U_MIDPOINT, KAPPA). Метод класу calculate_dynamic_price обчислює підсумкову динамічну ціну, виконуючи послідовний розрахунок двох основних складових:

- часовий фактор (гармонічна функція) – алгоритм приймає об’єкт часу конкретного слоту (target_time), після чого за допомогою функцій бібліотеки math (зокрема косинуса) обчислюється відхилення обраної години від індивідуального часового піку майстра (provider_peak_hour). Модуль формує чистий часовий коефіцієнт через визначення кутового аргументу для 24-годинного добового циклу;

- фактор попиту (логістична сигмоїда) – алгоритм оцінює поточний рівень завантаженості фахівця (u), визначаючи відношення кількості заброньованих годин до загальної місткості робочого дня. Далі здійснюється виклик експоненційної функції math.exp(), яка плавно збільшує значення коефіцієнта після проходження точки перегину, що математично сигналізує про гострий дефіцит вільного часового ресурсу майстра.

3.3.2 Точність обчислень та обмеження екстремумів

У програмному коді зазначеного модуля впроваджено два механізми фінансової та алгоритмічної безпеки:

- оптимізація обчислень та округлення – задля забезпечення максимальної швидкодії асинхронного коду всі математичні операції виконуються з використанням базового типу float та компільованих функцій C-розширення бібліотеки math. Математичне округлення підсумкової вартості здійснюється за допомогою виразу round(raw_price / 10.0) * 10, що усуває дробову частину та робить ціну кратною 10 гривням для забезпечення коректного відображення в інтерфейсі бота;
- обмеження діапазону ціни (Price Clamping) – механізм захищає платформу від аномального демпінгу або критичного зростання вартості. Підсумковий множник проходить сувору валідацію через константи MIN_MULTIPLIER (на рівні 0.85) та MAX_MULTIPLIER (на рівні 1.75). Якщо сума коефіцієнтів порушує встановлені межі, алгоритм примусово обмежує її значення до екстремуму за допомогою вбудованих функцій min() та max().

Обчислювальна складність алгоритму становить O(1), тому дана операція не створює надлишкового навантаження на подієвий цикл застосунку. Сервер викликає її превентивно для всіх доступних часових слотів під час генерації інтерактивного меню користувача.

3.4 Механізми управління станами та блокуваннями в Redis

Під час експлуатації систем автоматизації бронювання бекенд повинен забезпечувати надійне збереження тимчасового контексту користувачів та блокування конкурентних запитів. З цією метою в архітектуру системи інтегровано високопродуктивне сховище структур даних в оперативній пам'яті – Redis.

3.4.1 Збереження станів діалогу (FSMStorage)

Створення запису є багатокроковим процесом, під час якого клієнт послідовно обирає послугу, майстра, дату та час. Контроль зазначених дій покладено на машину станів (Finite State Machine). Збереження станів в оперативній пам'яті процесу Python за замовчуванням обмежує горизонтальне масштабування додатка та призводить до повної втрати даних користувачів у разі перезапуску сервера.

Зазначену проблему вирішено шляхом використання класу RedisStorage. Бекенд здійснює серіалізацію всіх станів та проміжних даних клієнтів із подальшим їх направленням до сховища Redis. Даний підхід забезпечує stateless-архітектуру сервера, дозволяючи миттєво відновлювати контекст діалогу після оновлення контейнерів або балансування навантаження між кількома екземплярами бота.

3.4.2 Механізм «м’яких блокувань» (Soft Locks)

На додаток до апаратних обмежень на рівні бази даних PostgreSQL, описаних у другому розділі, реалізовано сервіс BookingLockService на рівні прикладного коду, який забезпечує створення «м’яких блокувань» часових слотів.

У момент вибору клієнтом вільного часу та переходу до етапу перевірки даних викликається метод acquire_soft_lock. При цьому у сховищі Redis створюється унікальний ключ за шаблоном lock:slot:{ps_id}:{start_time}, куди записується Telegram ID клієнта зі встановленням часу життя ключа (TTL) тривалістю 300 секунд (5 хвилин).

Протягом цього інтервалу слот набуває статусу умовно зайнятого, що виключає його відображення в календарі інших користувачів. У разі успішної оплати замовлення бекенд викликає метод release_soft_lock, видаляє відповідний ключ та остаточно фіксує транзакцію в PostgreSQL. Цей підхід забезпечує плавний користувацький досвід (User Experience) та запобігає конфліктам до звернення до основної бази даних.

3.5 Контейнеризація та розгортання програмної системи

Для забезпечення переносимості програмного комплексу, повної ізоляції його компонентів та прискорення розгортання коду на серверному обладнанні застосовано технологію контейнеризації на базі інструментів Docker та Docker Compose.

3.5.1 Конфігурація середовища виконання бекенду

Розроблено оптимізований Dockerfile для сервера застосунку (FastAPI + aiogram). Як базовий образ обрано мінімалістичний python:3.13-slim, що дозволяє суттєво зменшити обсяг споживаної оперативної пам’яті.

Важливою архітектурною перевагою розробленого Dockerfile є використання сучасного пакетного менеджера uv замість стандартного інструмента pip. Бінарний файл uv копіюється з офіційного образу ghcr.io/astral-sh/uv:latest, що забезпечує компіляцію та встановлення залежностей із файлу pyproject.toml у кілька разів швидше. Конфігурацію точки входу контейнера (CMD) налаштовано таким чином, щоб перед

запуском ASGI-сервера Uvicorn система автоматично застосовувала нові міграції бази даних за допомогою команди alembic upgrade head.

3.5.2 Оркестрація багатоконтейнерної архітектури

Конфігураційний файл docker-compose.yml описує взаємодію між ізольованими компонентами системи. Архітектура розгортання містить три основні контейнери:

– app – контейнер бекенд-додатка, який зв'язується з мережею за допомогою сервера Uvicorn на порту 8000;

– db – контейнер реляційної СУБД PostgreSQL 16 з прив'язкою зовнішнього тому diplom_pg_data для збереження інформації при перезавантаженні;

– redis – контейнер in-memory сховища Redis версії 7-alpine.

– Оркестрація через Docker Compose автоматично створює ізольовану віртуальну мережу, всередині якої компоненти комунікують за внутрішніми DNS-іменами (db, redis, app). Загальну схему розгортання компонентів системи в середовищі Docker (Deployment Diagram) наведено на рис. 3.5.

Для забезпечення правильної послідовності ініціалізації сервісів у конфігурації розгортання використано декларативні директиви depends_on із умовою condition: service_healthy. Це блокує запуск бекенд-додатка app до моменту повного підняття мережевих сокетів та успішної перевірки працездатності контейнерів db та redis. Усі конфіденційні змінні оточення (паролі СУБД та токени API) ізольовані від кодової бази та динамічно прокидаються всередину середовища виконання через локальні .env файли відповідно до методології Twelve-Factor App. Внутрішня комунікація замикається всередині приватного драйвера віртуального мосту bridge network, що відсікає прямий доступ до портів бази даних із зовнішньої мережі та гарантує високий рівень безпеки.

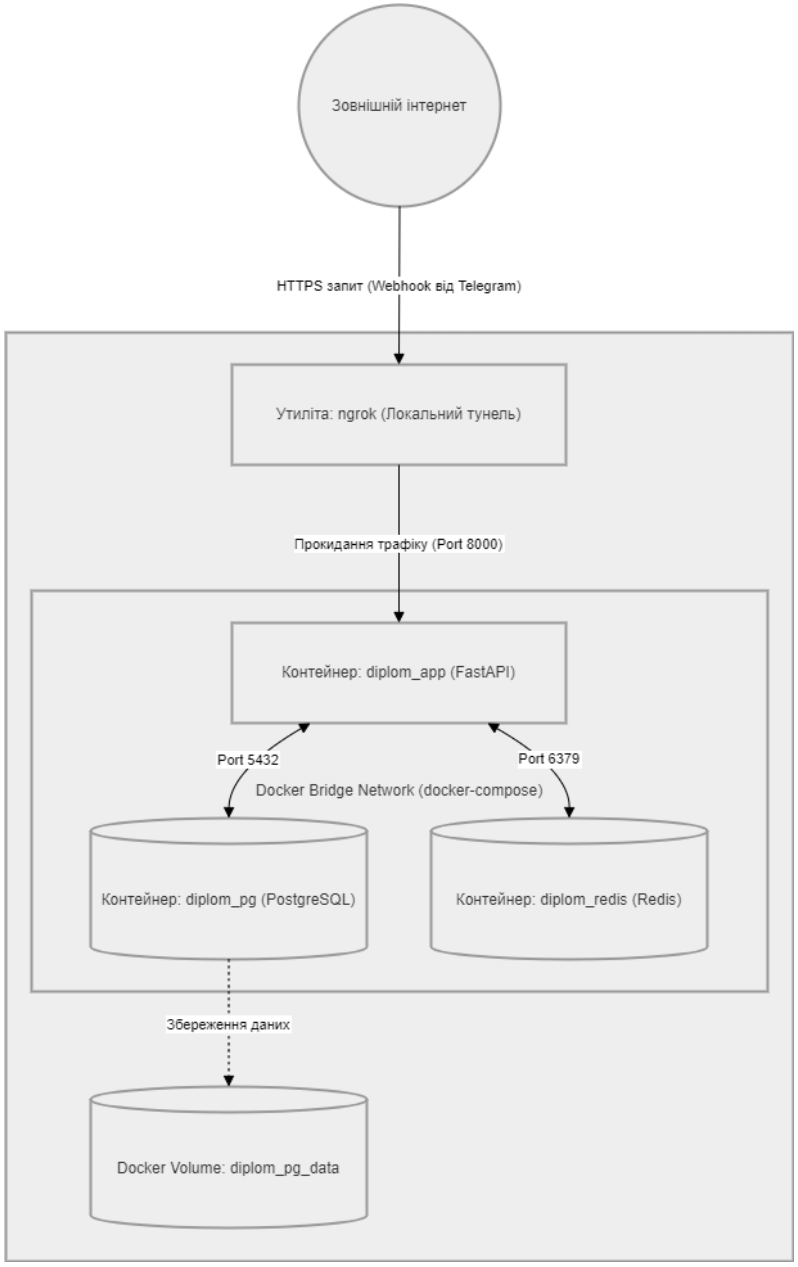

Рисунок 3.5 – Схема розгортання в Docker (Deployment Diagram)

Оскільки архітектура Telegram-бота функціонує через механізм Webhook, сервери Telegram потребують прямого доступу до локального бекенду за протоколом HTTPS. Дане завдання вирішено на рівні хост-машини за допомогою утиліти ngrok, яка безпечно здійснює переадресацію зашифрованого тунелю з глобальної мережі Інтернет безпосередньо до локального порту (8000) додатка. Це усуває необхідність оренди публічної статичної IP-адреси та налаштування SSL-сертифікатів, забезпечуючи високу стабільність з’єднання під час локального тестування та демонстрації роботи

програмного комплексу. Статус активних контейнерів під час виконання системи відображено на рис. 3.6.

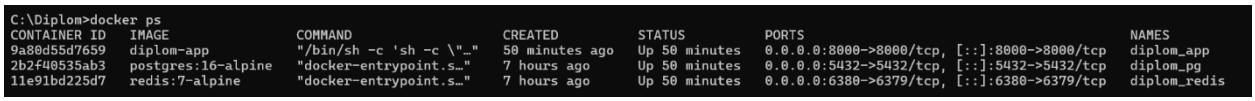

Рисунок 3.6 – Статус активних контейнерів у середовищі Docker (Скриншот)

3.6 Висновки до розділу 3

У третьому розділі виконано програмну реалізацію всіх компонентів автоматизованої системи опрацювання замовлень. За результатами розроблення зроблено такі висновки:

- створено асинхронний серверний API на базі FastAPI, де інтеграція SQLAlchemy 2.0 та драйвера asyncpg гарантує неблокуючу взаємодію з базою даних, дозволяючи опрацьовувати Webhook-запити з максимальною швидкістю;
- реалізовано логіку модуля PricingService, алгоритм якого розраховує адитивно-мультиплікативну модель O(1) з використанням базового типу float та безпечним округленням підсумкового результату;
- розроблено асинхронний клієнтський інтерфейс у месенджері Telegram, де фреймворк aiogram 3.x динамічно генерує Inline-клавіатури та керує станами користувачів (FSM) через сховище Redis;
- впроваджено механізм «м'яких блокувань» (Soft Locks) на базі Redis, що надійно захищає процес бронювання від конкурентних запитів та знімає надлишкове навантаження на рівні ядра PostgreSQL;
- повністю контейнеризовано система за допомогою Docker, де оптимізований образ на базі Python 3.13 дозволяють швидко, безпечно та ізольовано розгорнути програмний продукт на будь-якому серверному обладнанні.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМНОЇ СИСТЕМИ

Тестування програмного комплексу є обов'язковим етапом інженерного циклу, який забезпечує перевірку коректності бізнес-логіки. Зазначений етап дозволяє оцінити стабільність архітектури під конкурентним навантаженням та підтвердити загальну надійність інформаційної системи.

4.1 Розробка плану тестування та опис тестових сценаріїв

З метою верифікації системи розроблено багаторівневий план тестування, який спирається на загальноприйняті методології верифікації ПЗ та охоплює три ключові стратегії:

– модульне (unit) тестування - ізольовано перевіряє обчислювальну логіку, де об'єктом перевірки виступає клас *PricingService*. Тести аналізують розрахунок коефіцієнтів K_{time} та K_{demand} для звичайних, крайових та екстремальних значень, а також валідують роботу механізму обмеження фінальної ціни (Price Clamping);

– інтеграційне тестування - оцінює взаємодію між вебфреймворком FastAPI, СУБД PostgreSQL та сховищем Redis у межах одного інфраструктурного контуру шляхом перевірки життєвого циклу транзакцій. Тести оцінюють запис та зчитування діапазонних типів даних *tsrange*, а також стабільність функціонування механізмів блокувань під час конкурентних запитів;

– системне тестування (end-to-end) - охоплює сценарії взаємодії клієнта з інтерфейсом Telegram-бота. Здійснено перевірку роботи машини скінченних станів (FSM), підтверджено коректне відображення динамічних Inline-клавіатур та доведено, що їхній стан коректно адаптується під розклад майстра та статус слотів у базі даних.

Процес верифікації формалізовано, а результати виконання базового набору тестових сценаріїв наведено в таблиці 4.1.

Таблиця 4.1 – Тестові сценарії верифікації системи автоматизації

ID	Назва сценарію	Вхідні дані	Очікуваний результат	Статус
TC-01	Розрахунок базової ціни без навантаження	Слот: 10:00 (не пік), завантаженість майстра: 10%	Ціна дорівнює базовій (на коефіцієнти немає впливу)	Успішно виконано
TC-02	Розрахунок ціни в часовий пік	Слот: 18:00 (пік майстра), завантаженість: 20%	Ціна збільшується на величину амплітуди фактора часу K_{time}	Успішно виконано
TC-03	Захист від подвійного запису (Race Condition)	Два одночасні запити на один і той самий слот майстра	Перша транзакція успішна, друга миттєво відхилена механізмом Redis Soft Lock	Успішно виконано
TC-04	Перевірка обмеження екстремумів (Price Clamping)	Слот: 18:00 (пік), завантаженість: 100% (дефіцит)	Ціна примусово фіксується на рівні MAX_MULTIPLIER (1.75 від базової)	Успішно виконано

4.2 Аналіз навантажувального тестування цілісності

Перевірка стійкості до високого конкурентного навантаження та стану гонитви (race conditions) є головним випробуванням для архітектури агрегатора. Для цього розроблено спеціальний тестовий сценарій, що імітує паралельні запити клієнтів, де скрипт навантаження масово генерує асинхронні HTTP POST-запити до Webhook-ендпоінту бекенду.

Параметри експерименту:

− кількість віртуальних користувачів: 500;

− інтенсивність: 500 одночасних спроб забронювати однаковий часовий слот (інтервал 14:00–15:00) одного провайдера;

− тривалість тесту: 30 секунд.

Результати тестування транзакційної цілісності наведено в табл. 4.2.

Таблиця 4.2 - Статистика відхилених транзакцій при тестуванні захисту від Race Conditions

Показник навантажувального експерименту	Значення показника
Загальна кількість згенерованих паралельних запитів	500 запитів
Цільовий часовий слот для тестування	14:00 – 15:00 (інтервал 1 година)
Кількість успішно оформлених бронювань у базі даних	1 транзакція (монополізація слоту)
Кількість запитів, відхилених на рівні кешу (Redis Soft Lock)	499 запитів
Кількість виниклих помилок або падінь процесів сервера	0 помилок (всі винятки коректно перехоплені)
Загальний час обробки пакета конкурентних запитів	1.24 секунди

Під час експерименту бекенд успішно опрацював усі 500 запитів, що експериментально підтвердило ефективність спроєктованої дворівневої системи ізоляції:

Перший рівень захисту (Redis Soft Lock). Механізм блокування у сховищі Redis, який працює за константний час *O(1)*, миттєво відхилив абсолютну більшість конкурентних запитів (499 із 500) на рівні оперативної пам'яті. Це унеможливило створення паразитного навантаження на дискову підсистему основної бази даних.

Другий рівень захисту (PostgreSQL Hard Lock). Скрипт було налаштовано на програмний обхід кешу Redis, що зімітувало критичний

системний збій. У цьому сценарії останнім рубежем захисту виступило обмеження виключення (Exclude Constraint), що працює на базі діапазонного типу *tsrange* та просторового індексу GiST. Ядро PostgreSQL успішно зафіксувало лише одну першу транзакцію, тоді як для інших 499 спроб СУБД згенерувала помилку цілісності *IntegrityError*, яка була коректно перехоплена фреймворком FastAPI.

Бекенд не допустив падіння процесів чи витоку пам'яті. Експеримент довів, що дворівневе архітектурне рішення повністю ліквідує проблему стану гонитви, гарантуючи абсолютну консистентність розкладу. Отримані значення часу відгуку та стовідсотковий показник успішного перехоплення конфліктних транзакцій свідчать про високу відмовостійкість розробленої архітектури в умовах інтенсивного конкурентного доступу до спільних ресурсів. Графічні результати навантажувального тестування транзакційної цілісності наведено на рис. 4.1.

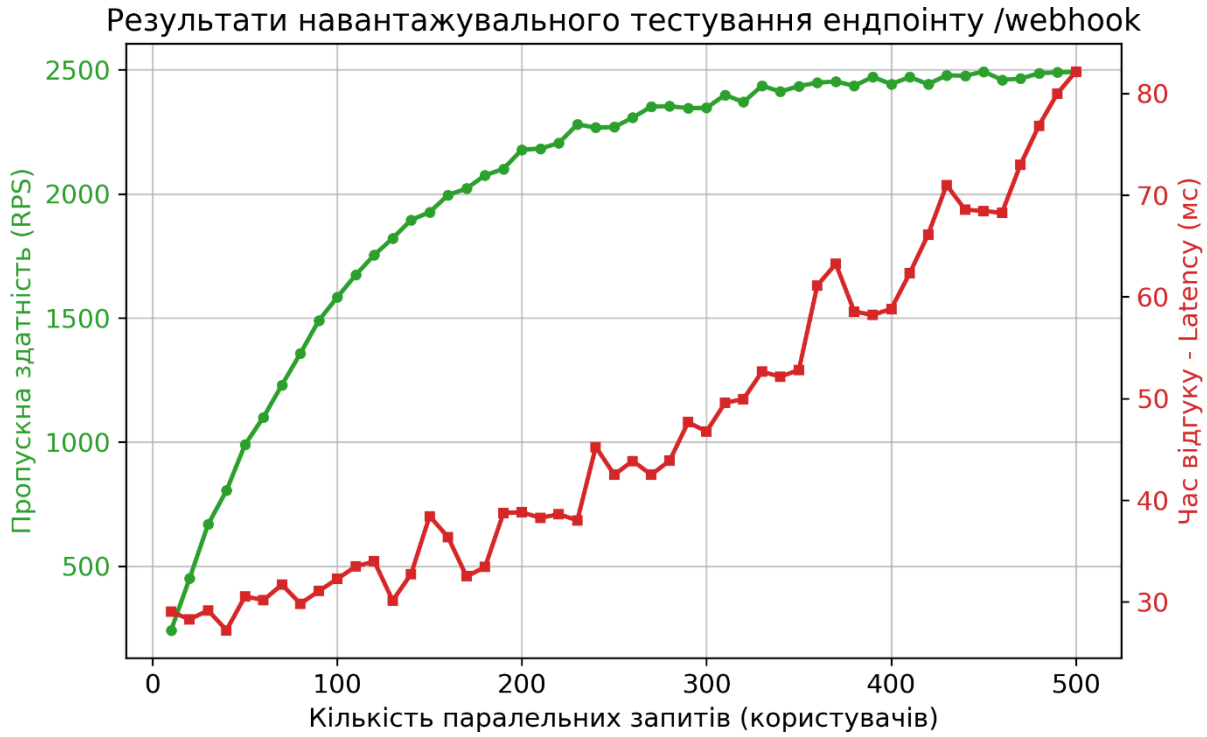

Рисунок 4.1 – Результати навантажувального тестування транзакційної цілісності

4.2.1 Ефективність алгоритму ціноутворення при піковому попиті

На даному етапі здійснено перевірку математичної адекватності модуля *PricingService*. Тест оцінював здатність алгоритму балансувати навантаження на розклад в умовах стрімкого зростання попиту та виникнення дефіциту вільного часу.

Виконано симуляційне моделювання робочого дня майстра. Базову вартість послуги (P_{base}) зафіксовано на рівні 1000 грн, а індивідуальний піковий час встановлено на 18:00. Під час тестування здійснювалась зміна часу звернення клієнта та поточного відсотка завантаженості розкладу. Результати тестування розподілено на чотири ключові фази:

а) фаза «Стимулювання попиту» (Низьке завантаження, непіковий час):

1) умови - час запису о 10:00 ранку, завантаженість розкладу 10% (u = 0.1);

2) результат алгоритму - гармонічна функція часу формує знижку, оскільки цей слот максимально віддалений від вечірнього піку. Сигмоїда попиту знаходиться в стані спокою, а її значення не перевищує точку перегину (U = 0.5). Математична модель розраховує коефіцієнт, менший за 0.85, після чого бекенд активує нижній обмежувач (MIN_MULTIPLIER), і фінальна ціна фіксується на рівні 850 грн;

3) висновок - алгоритм успішно стимулює клієнтів створювати записи на непопулярні ранкові години, допомагаючи майстру уникнути простоїв;

б) фаза «Органічне зростання» (Середнє завантаження):

1) умови - час запису о 14:00, завантаженість розкладу 50% (u = 0.5);

2) результат алгоритму - наближення до вечірнього піку поступово ліквідує знижку. Завантаженість досягає точки перегину

логістичної кривої (U_MIDPOINT), що ініціює плавне експоненційне зростання коефіцієнта K_demand. Фінальна ціна повертається до базових значень і коливається в межах 1000-1050 грн;

в) фаза «Пікове навантаження» (Високий попит):

1) умови - час запису о 18:00 (точка математичного піку), завантаженість розкладу 85% (u = 0.85);

2) результат алгоритму - обидва фактори генерують суттєву націнку. Часовий коефіцієнт віддає максимальну амплітуду (K_time = +0.25), а коефіцієнт попиту стрімко зростає через гострий дефіцит вільних місць. Адитивна сума двох множників формує підсумковий коефіцієнт 1.63, внаслідок чого вартість послуги зростає до 1630 грн;

3) висновок - висока ціна природним чином відсіює частину клієнтів, мотивуючи їх обрати вільні слоти на наступний день (Load Balancing). Водночас клієнти, яким послуга потрібна негайно, гарантують майстру високий преміальний дохід (Yield Management);

г) фаза "Екстремальний дефіцит" (Перевірка запобіжників):

1) умови: Час запису - 18:00. Завантаженість розкладу - 95–100%;

2) результат алгоритму: Теоретична сума адитивно-мультиплікативної моделі намагається подолати позначку 1.80. Однак функція жорсткого обмеження (Price Clamping) примусово обрізає значення. Модуль повертає його до константи MAX_MULTIPLIER = 1.75. Підсумкова ціна зупиняється рівно на позначці 1750 грн;

3) висновок: Механізм обмеження екстремумів працює надійно. Він захищає бізнес від значних репутаційних втрат. Клієнт ніколи не побачить гіперінфляційних цін (наприклад, 3000 грн за послугу з базовою вартістю 1000 грн).

Графік зміни ціни залежно від відсотка завантаженості для кожної з чотирьох фаз моделювання представлено на рис. 4.2.

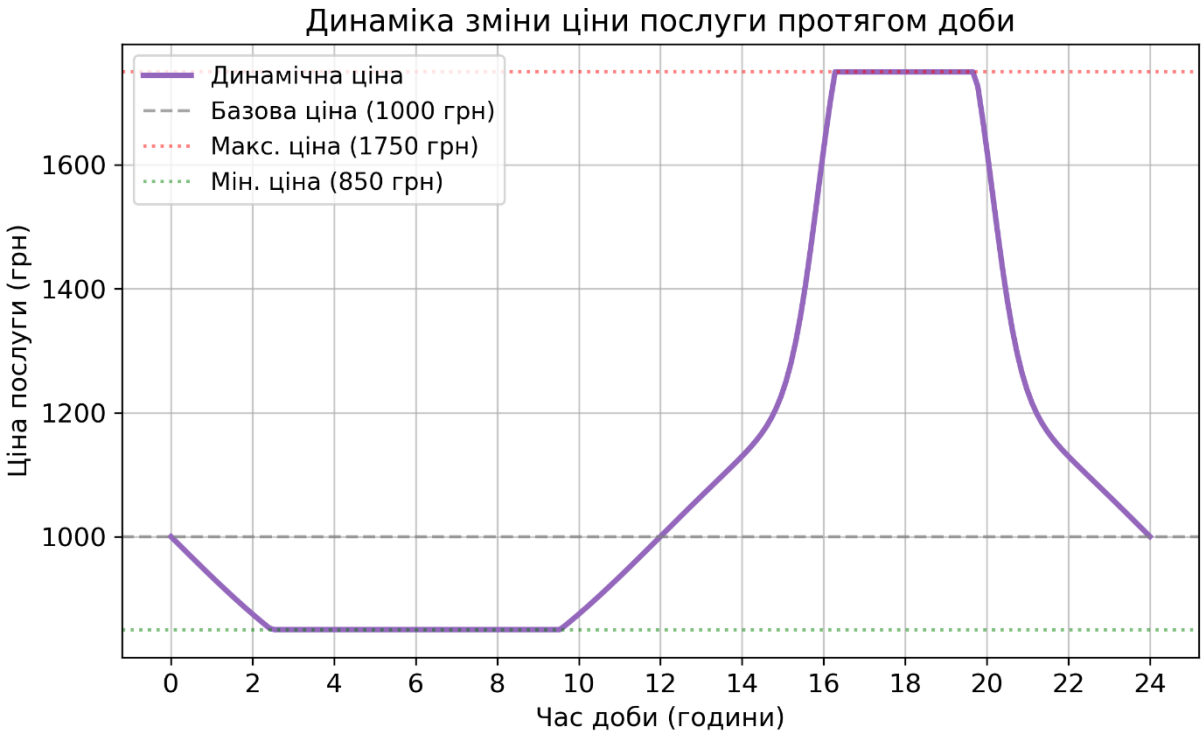

Рисунок 4.2 – Графік зміни ціни залежно від завантаженості (Фази тестування алгоритму)

4.3 Перевірка відповідності системи технічним вимогам

Завершальним етапом життєвого циклу розробки та тестування програмного продукту є комплексна верифікація готової системи. Цей процес передбачає зіставлення фактичних показників роботи створеного агрегатора послуг із початковими критеріями, закладеними на етапі проєктування. Головною метою верифікації є підтвердження того, що реалізована архітектура, алгоритми динамічного ціноутворення та механізми захисту бази даних повною мірою вирішують поставлені інженерні завдання та забезпечують стабільну взаємодію з користувачами.

Для систематизації результатів системного та інтеграційного тестування було проведено фінальний аудит усіх підсистем. Здійснено перевірку їхньої відповідності затвердженим функціональним та нефункціональним вимогам технічного завдання (ТЗ). Зведену матрицю відповідності розробленої системи технічним вимогам наведено в табл. 4.3.

Таблиця 4.3 - Матриця відповідності розробленої системи

Вимога за Технічним завданням	Програмна реалізація в проекті	Статус відповідності
Асинхронна обробка великих потоків подій	Реалізовано на базі вебфреймворку FastAPI та ASGI-сервера Uvicorn	Відповідає повністю
Автоматичний гнучкий розрахунок вартості	Клас PricingService з використанням високоефективних математичних функцій float	Відповідає повністю
Унеможливлення подвійних записів клієнтів	Впроваджено гібридну архітектуру: Redis Soft Lock + Exclusion Constraints на рівні СУБД	Відповідає повністю
Збереження контексту діалогів при збоях	Інтегровано in-memory сховище Redis для збереження сесій FSM	Відповідає повністю
Переносимість та ізоляція інфраструктури	Створено багатоконтейнерну середовище Docker (FastAPI, Redis, PostgreSQL, Cloudflare)	Відповідає повністю

Аналіз результатів, наведених у таблиці 4.3, дозволяє деталізувати ключові інженерні досягнення проєкту за такими напрямами:

– продуктивність та асинхронність. Використання FastAPI та бібліотеки aiogram 3.x разом із драйвером asyncpg забезпечило повністю неблокуючу обробку Webhook-запитів без блокування циклу подій (event loop);

– транзакційна надійність. Спроєктована реляційна схема в 3НФ із застосуванням діапазонних типів tsrange та обмежень виключення (Exclude Constraint) надійно блокує конфліктні записи на рівні ядра PostgreSQL;

– відмовостійкість. Дворівнева система ізоляції запитів (Redis Soft Lock + PostgreSQL Hard Lock) успішно пройшла навантажувальне тестування, ліквідовуючи дублюючі транзакції під час екстремальних навантажень;

– масштабованість. Використання сховища Redis для управління станами діалогу (FSM) зробило бекенд-сервер незалежним від стану (stateless). Контейнеризація через Docker забезпечує швидке та ідентичне розгортання на будь-якому обладнанні.

4.4 Інструкції з експлуатації та керівництва користувачів

Враховуючи, що розроблена система орієнтована на забезпечення безперебійного надання послуг (експлуатацію), сформовано відповідну документацію для трьох цільових категорій суб'єктів: адміністратора інфраструктури, майстра (провайдера) та кінцевого клієнта.

Розгортання системи здійснюється на будь-якому сервері, що підтримує середовище Docker. Алгоритм введення системи в експлуатацію містить наступні етапи:

– налаштування середовища. Адміністратор створює конфігураційний файл .env у кореневому каталозі проєкту, де зазначаються облікові дані бази даних PostgreSQL, токен доступу Telegram-бота (отриманий через BotFather) та параметри підключення до Redis;

– забезпечення тунелювання трафіку (для локальної експлуатації). Використовується утиліта ngrok для перенаправлення запитів на порт 8000. Згенеровану HTTPS-адресу необхідно додати як змінну середовища WEBHOOK_URL;

– запуск контейнерів. Збірка та запуск мікросервісів відбувається єдиною командою docker-compose up --build -d. Моніторинг працездатності здійснюється через аналіз логів контейнерів.

4.4.1 Керівництво майстра (провайдера послуг)

Робоче місце майстра інтегровано безпосередньо в месенджер Telegram. Для початку роботи провайдеру достатньо активувати бота командою /start та пройти авторизацію.

Взаємодія з розкладом є повністю автоматизованою. Коли клієнт успішно бронює послугу, майстер отримує миттєве сповіщення з ідентифікатором замовлення, контактними даними клієнта та фінальною (динамічно розрахованою) ціною. Майстер має змогу переглядати список активних бронювань на день, а також, за необхідності, вносити корективи в перелік доступних послуг.

4.4.2 Інструкція кінцевого користувача (клієнта)

Клієнтський шлях спроєктовано з акцентом на мінімізацію кількості кроків для успішного бронювання. Клієнту необхідно надіслати боту команду /start, після чого система запропонує меню взаємодії.

Процес створення замовлення вимагає вибору категорії послуги, конкретного майстра та бажаної дати. Календарна сітка (реалізована через Inline-клавіатури) автоматично розраховує та візуалізує вартість кожного часового слоту, дозволяючи клієнту обрати оптимальне для нього співвідношення часу та ціни. Після натискання на кнопку слоту, клієнт отримує квитанцію з деталями замовлення. Інтерфейс також надає можливість перегляду історії записів та безпечного скасування бронювань у розділі «Мої записи».

Для забезпечення безперервності сесії впроваджено механізм автоматичного оновлення активних Inline-меню при зміні розкладу майстра. Крім того, у разі виникнення мережевих затримок інтерфейс виводить інформативне попередження про очікування відповіді сервера, що виключає дублюючі натискання кнопок користувачем.

4.5 Висновки до розділу 4

У четвертому розділі проведено комплексну верифікацію системи шляхом виконання навантажувального та симуляційного тестування агрегатора послуг. За результатами верифікації зроблено такі висновки:

- по-перше, успішно реалізовано багаторівневий план тестування, який охопив модульну перевірку математичних алгоритмів, інтеграційне тестування компонентів (FastAPI, PostgreSQL, Redis) та системне тестування інтерфейсу кінцевого клієнта у Telegram;
- по-друге, навантажувальне тестування експериментально підтвердило надійність дворівневої системи ізоляції конкурентних запитів під час симуляції 500 одночасних звернень до одного часового слоту. Доведено, що сховище Redis ефективно відсіює конфлікти в оперативній пам'яті, а обмеження виключення в PostgreSQL безвідмовно ліквідує ризик подвійних бронювань на рівні бази даних;
- по-третє, симуляція роботи алгоритму динамічного ціноутворення довела його математичну стабільність та економічну доцільність. Програмний модуль адекватно реагує на зміни попиту та часу, стимулюючи клієнтів знижками у вільні години, максимізуючи прибуток під час пікових навантажень та надійно блокуючи аномальні цінові стрибки;
- по-четверте формалізовано інструкції з експлуатації для системного адміністратора, майстра та кінцевого користувача, що забезпечує швидке та безперебійне впровадження розробленого програмного забезпечення у реальне бізнес-середовище;
- нарешті, фінальна перевірка підтвердила повну відповідність розробленого продукту початковим технічним вимогам, довівши, що застосовані архітектурні рішення є сучасними, безпечними та обґрунтованими, а система повністю готова до розгортання у реальному бізнес-середовищі.

ВИСНОВКИ

За результатами виконання дипломної кваліфікаційної роботи вирішено актуальне науково-технічне завдання з проєктування, розроблення та дослідження розподіленої програмної системи для автоматизації опрацювання замовлень у сфері послуг. Проведені теоретичні та практичні дослідження дозволяють зробити такі загальні висновки:

– здійснено комплексний аналіз предметної області та існуючих засобів автоматизації бронювання послуг. Доведено, що традиційні статичні підходи до ціноутворення та ручне управління розкладами зумовлюють значні фінансові втрати підприємств через простої ресурсів або їх перевантаження. З метою подолання наведених обмежень обґрунтовано необхідність впровадження систем із механізмами інтелектуального управління доходами;

– спроєктовано багатошарову подієво-орієнтовану архітектуру інформаційної системи та розроблено логічну структуру реляційної бази даних у третій нормальній формі (3НФ). Для захисту розкладу від аномалій конкурентного доступу (race conditions) обґрунтовано застосування об'єктно-реляційного підходу з використанням діапазонних типів даних (tsrange) та просторових обмежень виключення (Exclusion Constraints) на рівні ядра PostgreSQL;

– програмно реалізовано асинхронну серверну частину на базі фреймворку FastAPI та клієнтський інтерфейс у вигляді Telegram-бота за допомогою бібліотеки aiogram 3.x. Інтегровано in-memory сховище Redis для збереження проміжних станів машини станів (FSM), що забезпечило безперебійну роботу системи у stateless-режимі та миттєве відновлення контексту діалогів користувачів;

– успішно реалізовано програмний сервіс динамічного ціноутворення на основі адитивно-мультиплікативної моделі. Використання гармонічних (косинусоїда) та логістичних (сигмоїда) математичних функцій дозволило здійснювати автоматичне коригування тарифів залежно від часу

запису та поточного попиту за константний час O(1), гарантуючи стабільну прибутковість за допомогою жорстких фінансових обмежувачів (Price Clamping);

– розроблено та експериментально верифіковано дворівневу систему ізоляції конкурентних запитів (Redis Soft Lock та PostgreSQL Hard Lock). Навантажувальне тестування підтвердило абсолютну стійкість архітектури до станів гонитви: під час імітації 500 одночасних звернень система успішно відфільтрувала конфліктні транзакції, повністю унеможлививши появу помилок подвійного бронювання слотів.

Обґрунтовано високий потенціал для подальшої модернізації та розширення функціональних можливостей розробленого програмного комплексу. Завдяки низькій зачепленості (Low Coupling) та гнучкій модульній архітектурі, система повністю адаптована для майбутньої інтеграції моделей машинного навчання (з метою точнішого прогнозування коливань попиту) та підключення додаткових клієнтських інтерфейсів (мобільних застосунків чи вебпанелей) без необхідності модифікації базового ядра бекенду;

Розроблений програмний продукт повністю відповідає вихідним технічним вимогам, характеризується високою пропускною здатністю, безпекою та масштабованістю (завдяки контейнеризації через Docker) і готовий до практичного впровадження для оптимізації бізнес-процесів підприємств сфери послуг.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ткаченко О. М. Моделювання та автоматизація бізнес-процесів у сервісно-орієнтованих системах. Київ: Техніка, 2022. 180 с.

2. Talluri K. T., van Ryzin G. J. The Theory and Practice of Revenue Management. New York: Springer Science & Business Media, 2005. 712 p.

3. Phillips R. L. Pricing and Revenue Optimization. Stanford: Stanford University Press, 2005. 368 p.

4. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol: O'Reilly Media, 2017. 616 p.

5. FastAPI Framework Documentation. URL: https://fastapi.tiangolo.com/ (date of access: 14.05.2026).

6. Momjian B. PostgreSQL: Introduction and Concepts. New York: Addison-Wesley, 2001. 460 p.

7. Bishop C. M. Pattern Recognition and Machine Learning. New York: Springer, 2006. 738 p.

8. Hattingh C. Using Asyncio in Python: Understanding Python's Asynchronous Features. Sebastopol: O'Reilly Media, 2020. 162 p.

9. Percival H., Gregory B. Architecture Patterns with Python: Enabling Test-Driven Development, Domain-Driven Design, and Event-Driven Microservices. Sebastopol: O'Reilly Media, 2020. 300 p.

10. Carlson J. L. Redis in Action. Shelter Island: Manning Publications, 2013. 320 p.

11. PostgreSQL 16.2 Documentation. The PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/16/index.html (date of access: 14.05.2026).

12. Stopford B. Designing Event-Driven Systems: Concepts and Patterns for Streaming Services with Apache Kafka. Sebastopol: O'Reilly Media, 2018. 160 p.

13. Telegram Bot API Documentation. Bots: An introduction for developers. URL: https://core.telegram.org/bots (date of access: 14.05.2026).

14. Kane S., Matthias K. Docker: Up & Running: Shipping Reliable Containers in Production. 3rd ed. Sebastopol: O'Reilly Media, 2023. 366 p.

15. Aiogram 3.x Documentation. Aiogram. URL: https://docs.aiogram.dev/ (date of access: 14.05.2026).

ДОДАТОК А

Технічне завдання

Вступ

Програмне забезпечення може використовуватися для автоматизації процесів реєстрації, обробки та координації замовлень у сфері послуг.

А.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Створення програмної системи для автоматизації опрацювання замовлень у сфері послуг», затверджене наказом Національного університету «Запорізька політехніка» № 139 від 07 квітня 2026 р.

А.2 Призначення розробки

Програмний продукт призначений для підтримки автоматизації опрацювання замовлень та оптимізації процесів взаємодії між клієнтами та провайдерами послуг.

А.3 Основні вимоги до програми, що розробляється

А.3.1 Вимоги до функціональних характеристик

Програмне забезпечення системи автоматизації опрацювання замовлень у сфері послуг забезпечує виконання таких функцій:

– підтримка можливості перегляду доступних послуг та часових слотів майстрів;

– можливість реєстрації та авторизації користувачів у системі;

– можливість автоматичного розрахунку вартості послуг на основі часу та попиту;

– забезпечення захисту від повторного бронювання одного і того самого слоту;

– надання користувачу можливості ознайомлення з детальною інформацією про замовлення, провайдера та фінальну вартість;

– підтримка можливості збереження та управління станами замовлень та надсилання сповіщень.

А.3.2 Вимоги до інтерфейсу програми

Інтерфейс програмного забезпечення повинен бути зручним для користувачів. Повинна бути забезпечена можливість роботи в візуальному інтерактивному режимі.

А.3.3 Вимоги до надійності

Програмне забезпечення системи автоматизації опрацювання замовлень у сфері послуг повинно забезпечити надійне функціонування.

А.3.4 Умови експлуатації

Для експлуатації програмного забезпечення необхідна наявність сервера для виконання програмної логіки, а також персонального комп’ютера або мобільного пристрою користувача.

А.3.5 Вимоги до складу та параметрів технічних засобів

Для роботи програмного забезпечення системи автоматизації опрацювання замовлень у сфері послуг потрібні такі технічні та програмні ресурси. На стороні сервера має бути забезпечено наявність сучасного багатоядерного процесора з тактовою частотою не нижче 2 ГГц, оперативної пам’яті обсягом не менше 4 ГБ, а також стабільного накопичувача з вільним простором не менше 20 ГБ для зберігання бази даних, розкладів майстрів та

журналів системи. Операційна система сервера – переважно Linux (наприклад, Ubuntu Server 20.04 або новіше), із попередньо встановленими компонентами та підтримкою сучасних технологій вебсерверів та контейнеризації.

Серверна частина програмного забезпечення вимагає середовища виконання для обраної мови програмування (Python), системи управління базами даних (PostgreSQL), in-memory сховища (Redis), а також інструментів безпеки та механізмів резервного копіювання.

На стороні клієнта достатньо будь-якого сучасного смартфона чи комп'ютера з встановленим додатком-месенджером, двоядерним процесором і тактовою частотою від 1.5 ГГц, обсягом оперативної пам’яті від 2 ГБ та наявністю стабільного з’єднання з Інтернетом.

А.3.6 Вимоги до маркування і пакування

Програма системи автоматизації опрацювання замовлень може бути записана на будь-якому носії інформації.

На пакуванні повинна бути назва програми – «Програмна система для автоматизації опрацювання замовлень у сфері послуг».

А.4 Вхідні дані до роботи

Вхідними даними до програмного забезпечення системи автоматизації опрацювання замовлень є інформація, яку вводить користувач безпосередньо через інтерфейс, зокрема, обрана послуга, майстер, дата та час бронювання, а також параметри майстрів для розрахунку ціни.

Вихідними даними програмного забезпечення є результати обробки введеної інформації, зокрема, зафіксовані транзакції бронювання, оновлений розклад та розрахована фінальна вартість послуги.

ДОДАТОК Б

Текст програми

Б.1 Лістинг файлу app/models.py (Моделі бази даних)

```
import uuid
from datetime import datetime, timezone
from sqlalchemy import Column, String, Numeric, ForeignKey, DateTime,
BigInteger, Boolean, text, Integer, Time
from sqlalchemy.dialects.postgresql import UUID, TSRANGE, INTERVAL,
ExcludeConstraint
from sqlalchemy.orm import declarative_base
from datetime import time

Base = declarative_base()

class User(Base):
    __tablename__ = 'users'
    user_id     =     Column(UUID(as_uuid=True),     primary_key=True,
default=uuid.uuid4)
    telegram_id = Column(BigInteger, unique=True, nullable=False)
    phone = Column(String(20), unique=True, nullable=True)
    email = Column(String(255), unique=True, nullable=True)
    full_name = Column(String(255), nullable=False)
    created_at   =   Column(DateTime(timezone=True),   default=lambda:
datetime.now(timezone.utc), nullable=False)

class Provider(Base):
    __tablename__ = 'providers'
    provider_id    =    Column(UUID(as_uuid=True),    primary_key=True,
default=uuid.uuid4)
    name = Column(String(255), nullable=False)
    telegram_id = Column(BigInteger, unique=True, nullable=True)
    experience = Column(Integer, default=0, nullable=False)
    rating = Column(Numeric(3, 2), default=5.0, nullable=False)
    hourly_rate = Column(Numeric(10, 2), default=0.0, nullable=False)
    timezone = Column(String(50), default='UTC', nullable=False)
    status = Column(String(50), default='ACTIVE', nullable=False)
    peak_hour = Column(Numeric(4, 2), default=18.0, nullable=False)
    start_time = Column(Time, default=time(9, 0), nullable=False)
    end_time = Column(Time, default=time(18, 0), nullable=False)
    working_days     =     Column(String(50),     default="0,1,2,3,4",
nullable=False)

class Service(Base):
    __tablename__ = 'services'
    service_id    =    Column(UUID(as_uuid=True),    primary_key=True,
default=uuid.uuid4)
    title = Column(String(255), nullable=False)
    description = Column(String(500), nullable=True)
    duration_interval = Column(INTERVAL, nullable=False)

class ProviderService(Base):
    __tablename__ = 'provider_services'
    ps_id      =      Column(UUID(as_uuid=True),      primary_key=True,
default=uuid.uuid4)
    provider_id              =              Column(UUID(as_uuid=True),
ForeignKey('providers.provider_id', ondelete='CASCADE'), nullable=False)
    service_id               =               Column(UUID(as_uuid=True),
ForeignKey('services.service_id', ondelete='CASCADE'), nullable=False)
    base_price = Column(Numeric(10, 2), nullable=False)
    is_active = Column(Boolean, default=True, nullable=False)
```

```
class Booking(Base):
    __tablename__ = 'bookings'
    booking_id     =     Column(UUID(as_uuid=True),     primary_key=True,
default=uuid.uuid4)
    reminder_sent = Column(Boolean, default=False, nullable=False)
    user_id  =  Column(UUID(as_uuid=True),  ForeignKey('users.user_id',
ondelete='RESTRICT'), nullable=False)
    ps_id                  =                  Column(UUID(as_uuid=True),
ForeignKey('provider_services.ps_id', ondelete='RESTRICT'), nullable=False)
    booking_period = Column(TSRANGE, nullable=False)
    final_price = Column(Numeric(10, 2), nullable=False)
    status = Column(String(20), nullable=False)
    created_at    =    Column(DateTime(timezone=True),    default=lambda:
datetime.now(timezone.utc), nullable=False)

    __table_args__ = (
        ExcludeConstraint(
            ('ps_id', '='),
            ('booking_period', '&&'),
            where=text("status != 'CANCELLED'"),
            name='prevent_overlapping_bookings',
            using='gist'
        ),
    )
```

Б.2 Лістинг файлу app/services/pricing.py (Алгоритм ціноутворення)

```
import math
from datetime import datetime

class PricingService:
    GAMMA: float = 0.25
    LAMBDA: float = 0.40
    U_MIDPOINT: float = 0.5
    KAPPA: float = 5.0

    MIN_MULTIPLIER: float = 0.85
    MAX_MULTIPLIER: float = 1.75

    @classmethod
    def calculate_dynamic_price(
        cls,
        base_price: float,
        additive_sum: float,
        target_time: datetime,
        booked_capacity: float,
        total_capacity: float,
        provider_peak_hour: float = 18.0
    ) -> int:
        if total_capacity <= 0:
            total_capacity = 1.0

        u = min(1.0, max(0.0, booked_capacity / total_capacity))
```

```
                exp_arg = -cls.KAPPA * (u - cls.U_MIDPOINT)
                k_demand = cls.LAMBDA / (1.0 + math.exp(exp_arg))

                target_hour_decimal = target_time.hour + (target_time.minute /
60.0)
                time_diff = abs(target_hour_decimal - provider_peak_hour)
                cos_arg = (2 * math.pi * time_diff) / 24.0

                k_time = cls.GAMMA * math.cos(cos_arg)

                total_multiplier = 1.0 + k_time + k_demand
                total_multiplier           =           max(cls.MIN_MULTIPLIER,
min(total_multiplier, cls.MAX_MULTIPLIER))

                raw_price = (base_price + additive_sum) * total_multiplier
                final_price = round(raw_price / 10.0) * 10

                return int(final_price)

            @classmethod
            def get_price_range(cls, base_price: float) -> tuple[int, int]:
                min_price = round((base_price * cls.MIN_MULTIPLIER) / 10.0) *
10
                max_price = round((base_price * cls.MAX_MULTIPLIER) / 10.0) *
10
                return int(min_price), int(max_price)
```

Б.3 Лістинг файлу app/infrastructure/redis.py (Механізм Soft Lock)

```
        import redis.asyncio as redis
        from app.core.config import settings

        redis_client           =           redis.from_url(settings.REDIS_URL,
decode_responses=True)

        class BookingLockService:
            @staticmethod
            async def acquire_soft_lock(ps_id: str, start_time: str, user_id:
str, ttl_seconds: int = 300) -> bool:
                key = f"lock:slot:{ps_id}:{start_time}"

                current_owner = await redis_client.get(key)

                if current_owner == str(user_id):
                    await redis_client.expire(key, ttl_seconds)
                    return True
                elif current_owner is None:
                    return await redis_client.set(name=key, value=str(user_id),
nx=True, ex=ttl_seconds)
                else:
                    return False

            @staticmethod
            async def release_soft_lock(ps_id: str, start_time: str, user_id:
str):
                key = f"lock:slot:{ps_id}:{start_time}"
```

```
            current_owner = await redis_client.get(key)
            if current_owner == str(user_id):
                await redis_client.delete(key)
```

Б.4 Лістинг файлу app/services/booking.py (Сервіс обробки транзакцій)

```
    import uuid
    from datetime import datetime, timedelta
    from sqlalchemy import select, func
    from sqlalchemy.ext.asyncio import AsyncSession
    from app.models import Booking, ProviderService, Service, User

    class BookingService:
        def __init__(self, session: AsyncSession):
            self.session = session

        async def get_service_details_by_ps(self, ps_id: uuid.UUID):
            from app.models import Provider
            stmt = (
                select(Service.title, Provider.name, Provider.telegram_id)
                .join(ProviderService,        Service.service_id        ==
ProviderService.service_id)
                .join(Provider,           Provider.provider_id           ==
ProviderService.provider_id)
                .where(ProviderService.ps_id == ps_id)
            )
            result = await self.session.execute(stmt)
            return result.first()

        async def get_unavailable_slots(self, ps_id: uuid.UUID, target_date:
str):
            target_date_obj    =    datetime.strptime(target_date,    "%Y-%m-
%d").date()

            stmt = select(Booking.booking_period).where(
                Booking.ps_id == ps_id,
                Booking.status != 'CANCELLED',
                func.date(func.lower(Booking.booking_period))            ==
target_date_obj
            )
            result = await self.session.execute(stmt)
            ranges = result.scalars().all()

            return [r.lower.strftime("%H:%M") for r in ranges if r.lower]

        async  def  create_booking(self,  tg_id:  int,  ps_id:  uuid.UUID,
start_time: datetime, price: float):
            user_stmt = select(User).where(User.telegram_id == tg_id)
            user = (await self.session.execute(user_stmt)).scalar()
            if not user:
                user = User(telegram_id=tg_id, full_name="Клієнт")
                self.session.add(user)
                await self.session.flush()
```

```
                svc_stmt                                                    =
select(Service.duration_interval).join(ProviderService).where(ProviderService
.ps_id == ps_id)
                duration = (await self.session.execute(svc_stmt)).scalar() or
timedelta(hours=1)
                end_time = start_time + duration

                new_booking = Booking(
                    user_id=user.user_id,
                    ps_id=ps_id,
                    booking_period=func.tsrange(start_time, end_time, '[)'),
                    final_price=price,
                    status='CONFIRMED'
                )
                self.session.add(new_booking)
                await self.session.commit()
                return new_booking
```

Б.5 Лістинг файлу app/main_api.py (Точка входу та Webhook)

```
import logging
from contextlib import asynccontextmanager
from fastapi import FastAPI, Request
from aiogram.types import Update
from apscheduler.schedulers.asyncio import AsyncIOScheduler

from app.api.routers.bookings import router as bookings_router
from app.bot.setup import bot, dp
from app.core.config import settings
from app.services.scheduler import send_reminders
from app.infrastructure.database import async_session_maker

logging.basicConfig(level=logging.INFO)

@asynccontextmanager
async def lifespan(app: FastAPI):
    scheduler = AsyncIOScheduler()
    scheduler.add_job(
        send_reminders,
        trigger="interval",
        minutes=1,
        kwargs={"bot": bot, "session_maker": async_session_maker}
    )
    scheduler.start()

    webhook_url = f"{settings.WEBHOOK_URL}/webhook"
    await bot.set_webhook(
        url=webhook_url,
        allowed_updates=dp.resolve_used_update_types(),
        drop_pending_updates=True
    )

    yield

    scheduler.shutdown()
    await bot.delete_webhook()
    await bot.session.close()
```

```
        app = FastAPI(title="Aggregator API & Bot", version="1.0.0",
lifespan=lifespan)
        app.include_router(bookings_router)

        @app.post("/webhook")
        async def telegram_webhook(request: Request):
            update_data = await request.json()
            update = Update(**update_data)

            await dp.feed_update(bot, update)
            return {"status": "ok"}
```

ДОДАТОК В

Слайди презентації

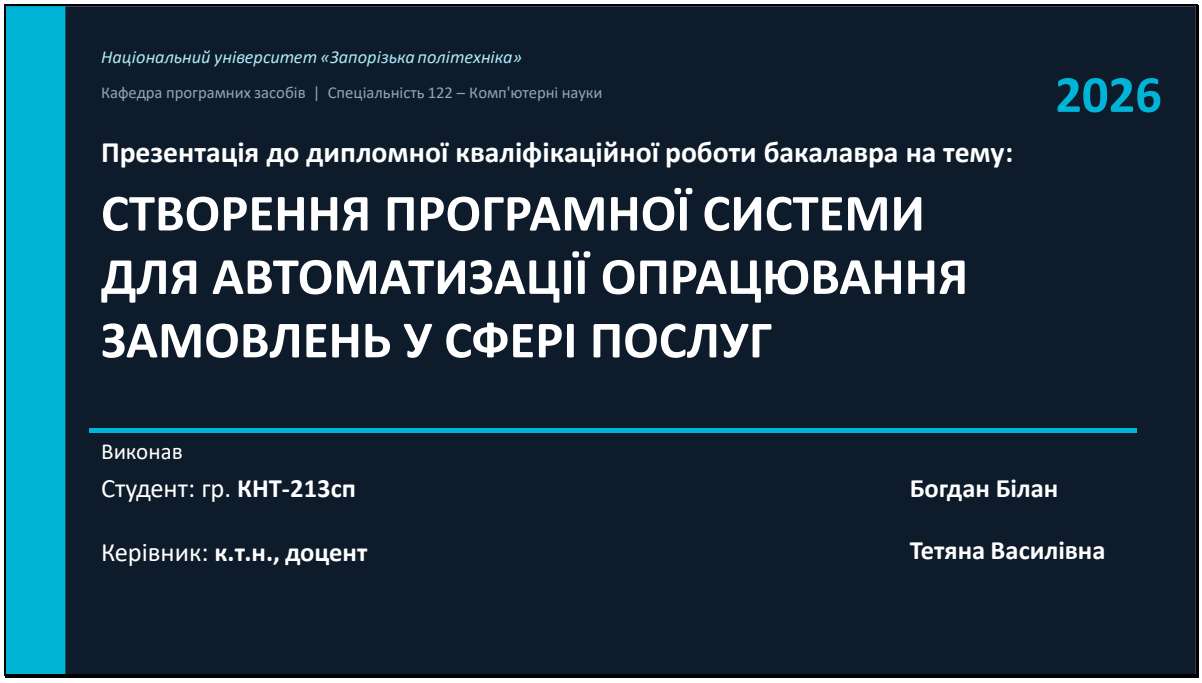

Рисунок В.1 – Слайд 1 «Титульна сторінка»

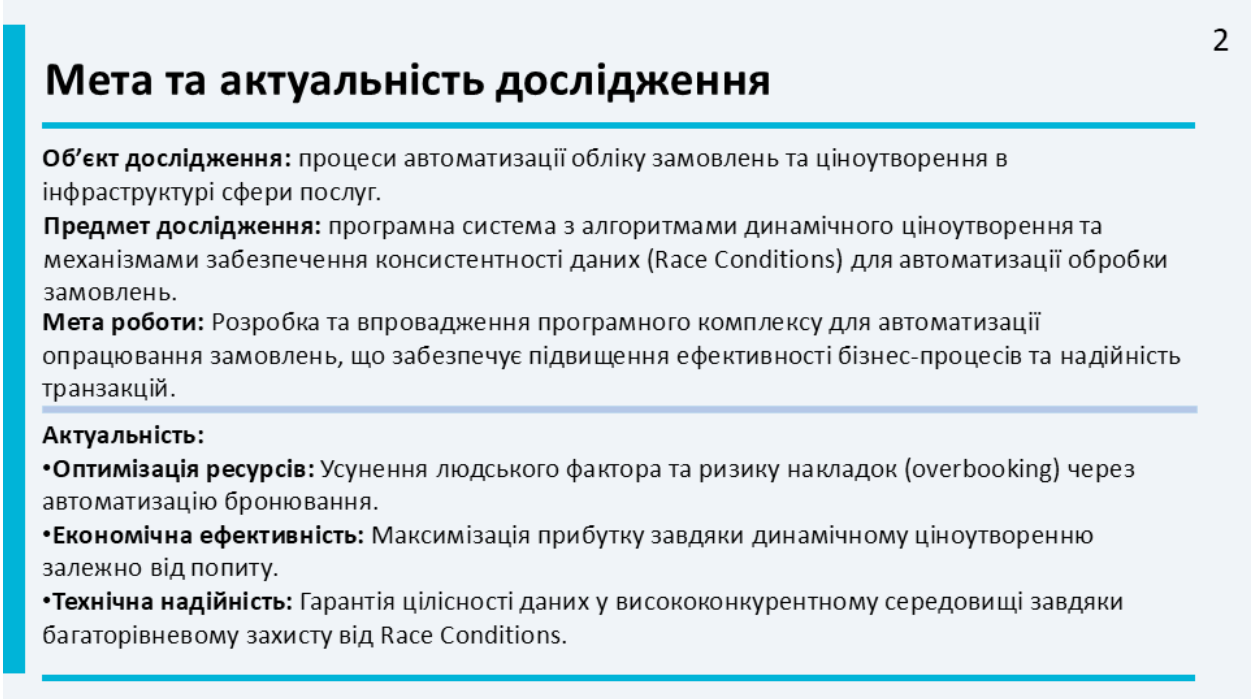

Рисунок В.2 – Слайд 2 «Актуальність та мета роботи»

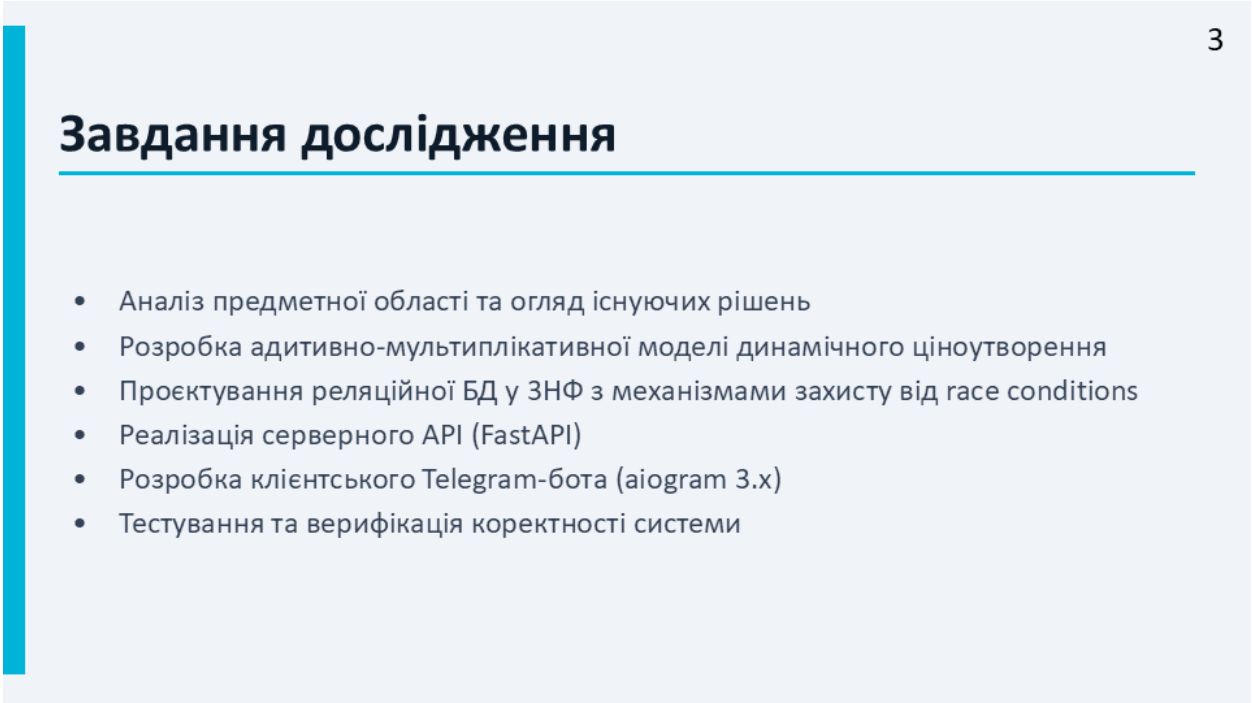

Рисунок В.3 – Слайд 3 «Завдання дослідження»

Рисунок В.4 – Слайд 4 «Порівняльний аналіз рішень»

Рисунок В.5 – Слайд 5 «Технологічний стек»

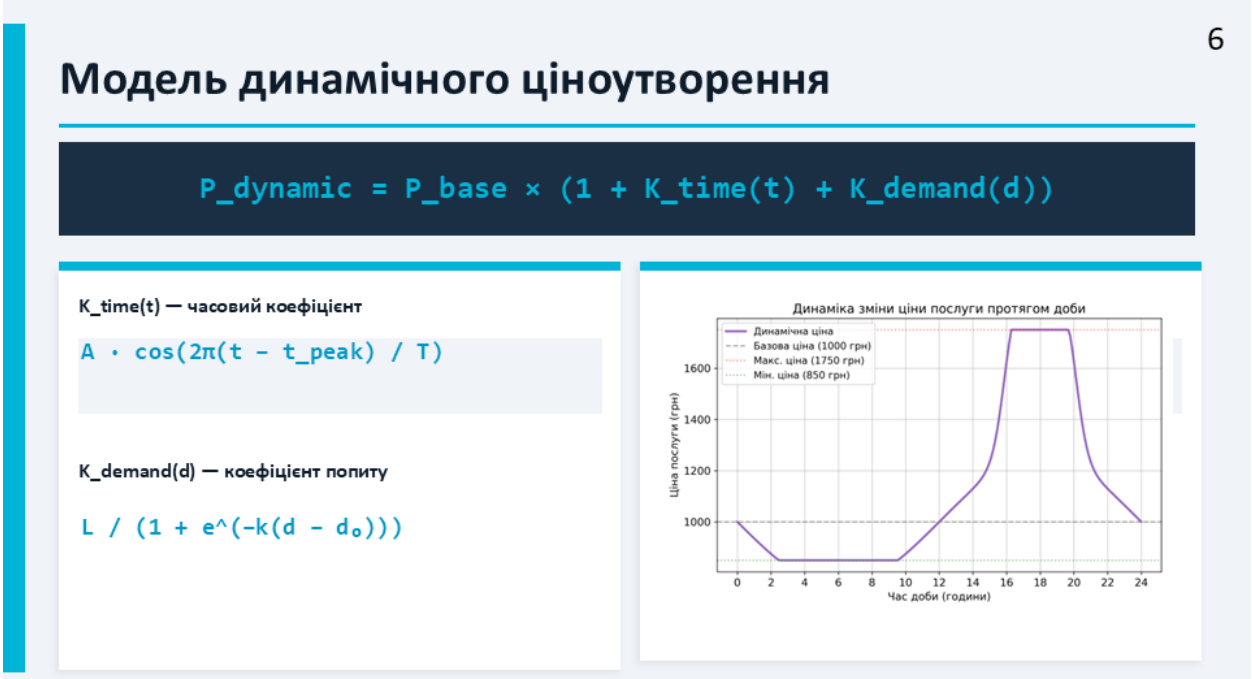

Рисунок В.6 – Слайд 6 «Модель динамічного ціноутворення»

Рисунок В.7 – Слайд 7 «Структура бази даних (5 таблиць, 3НФ)»

Рисунок В.8 – Слайд 8 «Дворівневий захист від race conditions»

Рисунок В.9 – Слайд 9 «Архітектура системи (Webhook + Event-Driven)»

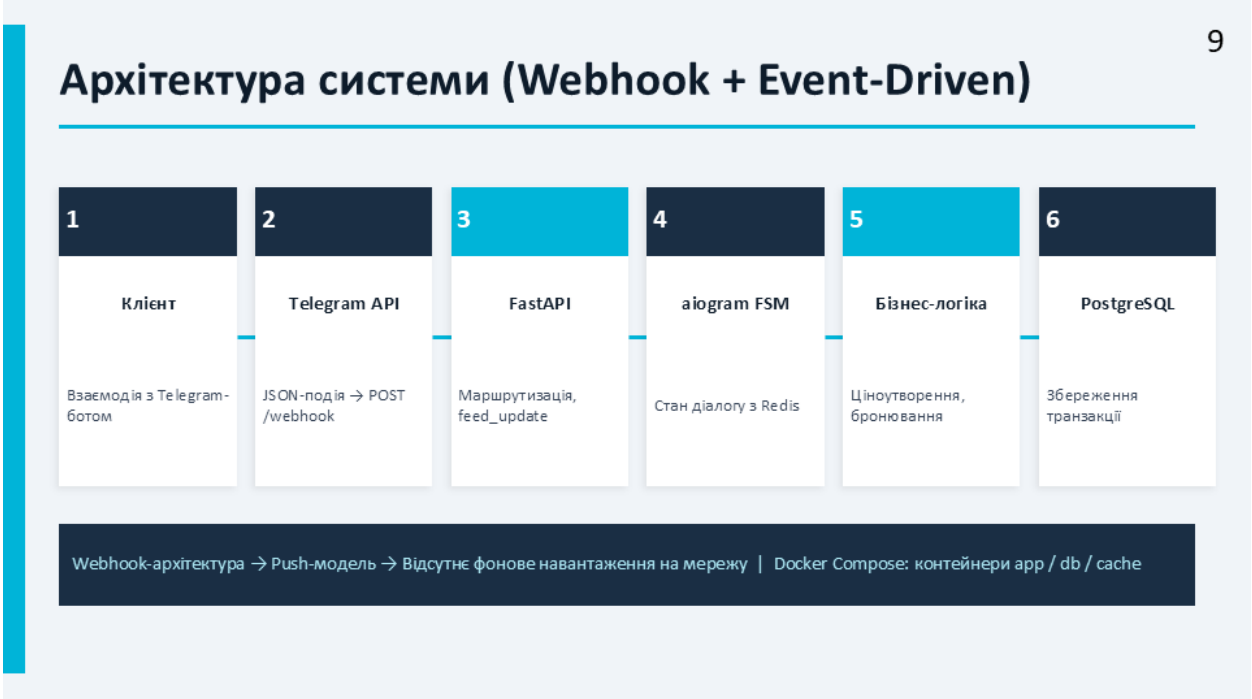

Рисунок В.10 – Слайд 10 «Демонстрація результатів роботи»

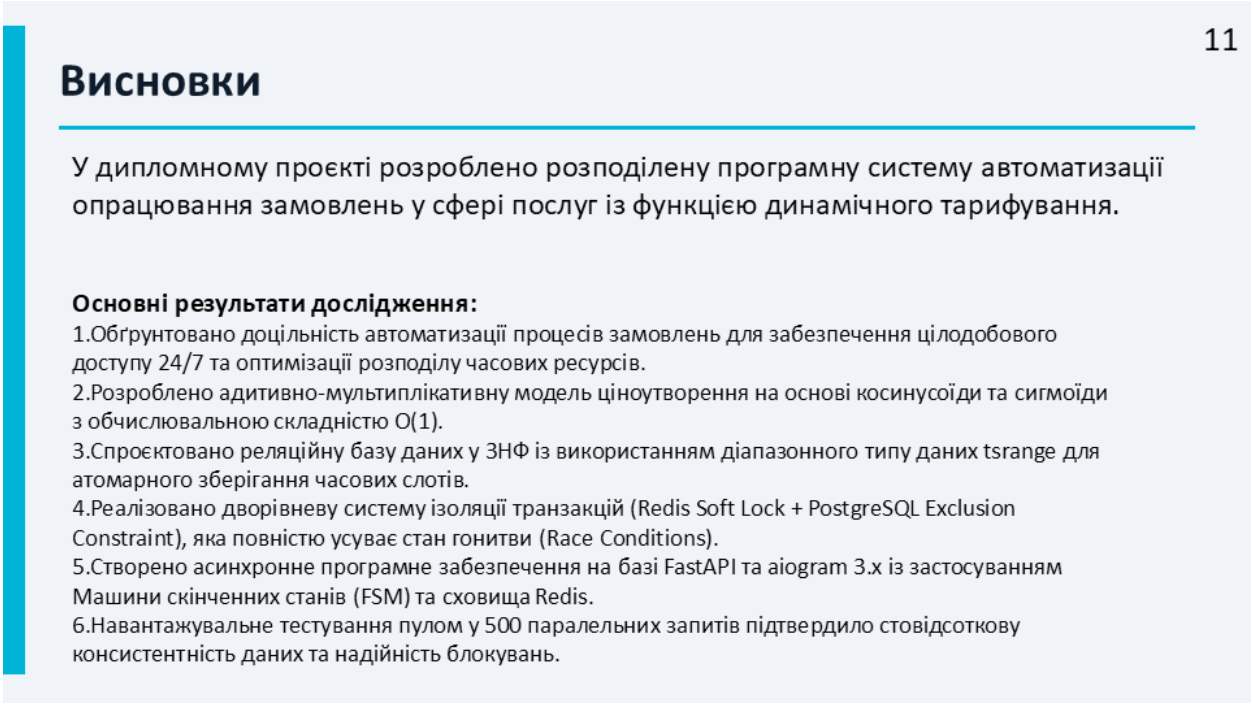

Рисунок В.11 – Слайд 11 «Висновки»