Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп’ютерних наук

Допущено до захисту
Завідувач кафедри комп’ютерних наук
доктор технічних наук, професор
Андрій БОНДАРЧУК

« 01» червня 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр»
спеціальність 122 Комп’ютерні науки
освітня програма 122.00.01 Інформатика

Тема: МОДУЛЬ ІНФОРМАЦІЙНО-УПРАВЛЯЮЧОЇ СИСТЕМИ ГОТЕЛЬНО-РЕСТОРАННОГО КОМПЛЕКСУ

Виконав
студент групи ІНб-2-22-4.0д
Стукаленко Максим Миколайович

_____
(підпис)

Науковий керівник
кандидат технічних наук, доцент
Мельник І. Ю.

_____
(підпис)

Київ – 2026

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

«Затверджую»
Завідувач кафедри комп'ютерних наук,
доктор технічних наук, професор
Андрій БОНДАРЧУК
« 31 » жовтня 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА
«Модуль інформаційно-управляючої системи готельно-ресторанного комплексу»

Виконавець – студент групи ІНб-2-22-4.0д,
спеціальності 122 Комп'ютерні наук,
освітньої програми 122.00.01 Інформатика
Стукаленко Максим Миколайович

1. Вихідні дані: Законодавство та нормативно-правові акти України у сфері інформаційних технологій; наукові публікації та технічна документація з питань автоматизації готельно-ресторанного бізнесу, побудови REST API та розробки програмного забезпечення; офіційна документація фреймворків Django та Django REST Framework; публікації з аналізу існуючих програмних рішень класів PMS та POS.

2. Основні завдання: Здійснити аналіз існуючих рішень готельно-ресторанного комплексу та провести порівняльний аналіз існуючих програмних рішень класів PMS, POS, комплексних систем та CRM; сформулювати функціональні та нефункціональні вимоги до модуля ІУС ГРК; обґрунтувати вибір архітектурного стилю REST API та технологічного стеку Django + Django REST Framework; розробка модулю інформаційно-управляючої системи готельно-ресторанного комплексу з реалізацією функціоналу: CRUD для моделей, алгоритми для функціоналу; проведення тестування розробленого модуля на коректність роботи; оформити пояснювальну записку відповідно до вимог стандартів та методичних рекомендацій кафедри.

3. Пояснювальна записка: Обсяг – 45 стор. Формату А4 комп'ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.

4. Графічні матеріали: Комп'ютерна презентація доповіді.

5. Додатки: не передбачено

6. Строк подання роботи на кафедру: «01» червня 2026 р.

Науковий керівник	Виконавець:
к.т.н., доцент	
_____Мельник І.Ю.	_____Стукаленко М.М.
_____ дата	_____ дата

Календарний план

№	Етапи виконання роботи	Термін виконання	Примітка
1	Затвердження теми кваліфікаційної роботи бакалавра	31.10.25	Виконано
2	Аналіз проблеми, вивчення аналогів, формування цілей і завдань	01.11.25 - 11.01.26	Виконано
3	Аналіз предметної області готельно-ресторанного комплексу та порівняльний аналіз існуючих програмних рішень (PMS, POS, CRM)	26.01.26 - 15.03.26	Виконано
4	Проектування модуля ІУС ГРК: вибір архітектурного стилю REST API, технологічного стеку Django+DRF, розробка ER-діаграм	16.03.26 - 31.03.26	Виконано
5	Розробка програмного модуля ІУС ГРК з реалізацією CRUD-операцій та бізнес-алгоритмів	01.04.26 - 15.04.26	Виконано
6	Тестування функціональності модулів	16.04.26 - 30.04.26	Виконано
7	Впровадження та оцінка результатів	01.05.26 - 15.05.26	
8	Підготовка пояснювальної записки, графічних матеріалів, додатків	25.05.25 - 12.06.26	Виконано
9	Захист кваліфікаційної роботи	15.06.26	

«Затверджую»
Науковий керівник
к.т.н., доцент, Мельник І.Ю.

Індивідуальний план склав
Стукаленко М.М.

РЕФЕРАТ

Кваліфікаційна робота: 45 с., 4 рисунків, 5 таблиць, 34 джерел.

Актуальність: ефективне функціонування готельно-ресторанного комплексу сьогодні неможливе без комплексної автоматизації бізнес-процесів. Більшість існуючих систем на ринку є монолітними та дорогими в обслуговуванні, а доступні рішення класів PMS і POS охоплюють лише один із двох операційних блоків підприємства. Це зумовлює потребу в розробці гнучкого, модульного рішення, придатного для малого та середнього готельно-ресторанного бізнесу.

Об'єкт дослідження: інформаційні процеси управління ресурсами та операційною діяльністю готельно-ресторанного комплексу.

Предмет дослідження: методи моделювання предметної області, алгоритмічне забезпечення та програмна реалізація модуля інформаційно-управляючої системи на базі архітектурного стилю REST та фреймворку Django REST Framework.

Мета роботи: проєктування інформаційної моделі та алгоритмічного забезпечення модуля інформаційно-управляючої системи готельно-ресторанного комплексу на основі архітектури REST API з подальшою програмною реалізацією та верифікацією коректності розроблених алгоритмів.

Завдання:

1. здійснити аналіз предметної області ГРК та провести порівняльний аналіз існуючих програмних рішень класів PMS, POS, комплексних систем та CRM;
2. сформулювати функціональні та нефункціональні вимоги до модуля ІУС ГРК та спроектувати його інформаційну модель і архітектуру з обґрунтуванням вибору архітектурного стилю REST API і технологічного стеку Django + Django REST Framework;

3. розробити та реалізувати алгоритмічне забезпечення ключових бізнес-процесів ГРК;

4. провести тестування розробленого модуля та аналіз результатів.

Основні результати: розроблено модуль інформаційно-управляючої системи готельно-ресторанного комплексу на базі архітектури REST API з використанням фреймворку Django REST Framework. Спроектовано модульну архітектуру у вигляді трьох незалежних застосунків (Hotel, Restaurant, Dashboard) із єдиною базою даних та реляційними моделями даних для 16 сутностей. Реалізовано API-інтерфейс із 54 ендпоінтами з повним CRUD, уніфікованим форматом помилок та двошаровою валідацією вхідних даних. Розроблено алгоритм автоматичного пошуку вільного номера на основі перевірки перетину часових інтервалів, що унеможливлює подвійне бронювання, алгоритм резервування столика з буфером ±2 години, а також механізм пакетного імпорту та зіставлення платіжних реєстрів у форматі Excel. Реалізовано консолідований дашборд комплексу, що агрегує ключові фінансові та операційні показники обох підрозділів у реальному часі. Проведено тестування методом чорного ящика з охопленням 31 тест-кейсу та перевіркою всіх 54 ендпоінтів – усі тести пройдено успішно..

Практичне значення дослідження полягає у впровадженні розробленого модуля як самостійного програмного рішення для автоматизації операційної діяльності підприємств малого та середнього готельно-ресторанного бізнесу, а також як готової технологічної основи для побудови повноцінних веб- та мобільних інформаційних систем галузі гостинності. Відкрита модульна архітектура забезпечує масштабованість продукту шляхом підключення нових функціональних підмодулів без зміни існуючого коду.

Ключові слова: інформаційно-управляюча система, готельно-ресторанний комплекс, REST API, Django REST Framework, автоматизація бронювання, модульна архітектура, CRUD, тестування API.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface – програмний інтерфейс застосунку.

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

DRF – Django REST Framework

ГРК – готельно-ресторанний комплекс

ІУС – інформаційно-управляюча система

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

PMS – Property Management System – система управління готелем

POS – Point of Sale – система автоматизації торгівлі

СУБД – система управління базами даних

UI – User Interface – інтерфейс користувача

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний ринок гостинності характеризується високою конкуренцією та постійним зростанням вимог до якості обслуговування. Ефективне функціонування готельно-ресторанного комплексу (ГРК) сьогодні неможливе без комплексної автоматизації бізнес-процесів: від миттєвого бронювання номерів до детального обліку сировини на складі та моніторингу фінансових потоків [1]. Більшість існуючих систем на ринку є монолітними та дорогими в обслуговуванні, це створює потребу в розробці гнучких, модульних рішень [9]. Використання архітектури REST API дозволяє відокремити бізнес-логіку від інтерфейсу користувача, забезпечуючи високу швидкість роботи та можливість інтеграції з різними клієнтськими додатками [25].

Мета дослідження. Проєктування інформаційної моделі та алгоритмічного забезпечення модуля інформаційно-управляючої системи готельно-ресторанного комплексу на основі архітектури REST API з подальшою програмною реалізацією та верифікацією коректності розроблених алгоритмів.

Об'єкт дослідження. Інформаційні процеси управління ресурсами та операційною діяльністю готельно-ресторанного комплексу.

Предмет дослідження. Методи моделювання предметної області, алгоритмічне забезпечення та програмна реалізація модуля інформаційно-управляючої системи на базі архітектурного стилю REST та фреймворку Django REST Framework.

Мета дослідження. Для досягнення мети було поставлено такі завдання:

1. Здійснити аналіз предметної області ГРК та провести порівняльний аналіз існуючих програмних рішень.

2. Спроектувати інформаційну модель та архітектуру модуля ІУС ГРК з обґрунтуванням вибору архітектурного стилю REST API і технологічного стеку.

3. Розробити та реалізувати алгоритмічне забезпечення ключових бізнес-процесів ГРК.

4. Провести тестування розробленого API та аналіз результатів.

Методи дослідження. У роботі використано методи системного аналізу, об’єктно-орієнтованого проектування, принципи побудови REST-сервісів та реляційного моделювання баз даних.

Практичне значення отриманих результатів полягає у розробці інформаційної моделі та алгоритмічного забезпечення модуля ІУС ГРК, придатних для безпосереднього впровадження на підприємствах галузі. Результати роботи можуть бути використані як основа для проєктування повноцінних інформаційно-управляючих систем підприємств малого та середнього бізнесу у сфері готельно-ресторанного обслуговування.

Результати роботи апробовані шляхом публікації тез доповіді на конференції Інформаційні технології – 2026: зб. тез XII Всеукраїнської науково-практичної конференції молодих учених, 14 травня 2026 р. м. Київ / Київський столичний університет імені Бориса Грінченка с. 175-179 [31].

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Характеристика готельно-ресторанного комплексу як об'єкта автоматизації

Готельно-ресторанний комплекс (ГРК) є багатофункціональним підприємством індустрії гостинності, що містить два операційно відмінних, але інформаційно взаємозалежних блоки: засіб розміщення та заклад харчування. З точки зору системного аналізу, ГРК є складною системою з розподіленою відповідальністю, розгалуженою ієрархією підрозділів та високою інтенсивністю інформаційного обміну між ними.

Готельний блок забезпечує надання послуг розміщення, що включають управління номерним фондом різних категорій, організацію додаткових сервісів та адміністрування циклу перебування гостя від бронювання до виселення. Ресторанний блок охоплює організацію харчування гостей готелю та зовнішніх відвідувачів: обслуговування в залі, доставку до номерів, організацію банкетних заходів, управління меню та контроль товарно-матеріальних запасів.

Організаційна структура ГРК охоплює такі функціональні підрозділи: служба прийому та розміщення, господарська служба, служба харчування, інженерно-технічна служба, фінансово-бухгалтерський відділ та служба безпеки. Кожен підрозділ генерує власний потік операційних даних і водночас є споживачем інформації від суміжних служб, створюючи необхідність централізованої інформаційної системи.

Аналіз операційної діяльності ГРК [1] дозволяє виділити такі ключові бізнес-процеси, що підлягають автоматизації:

– Управління бронюванням. Процес охоплює прийом запитів на бронювання через різні канали (прямий, онлайн, телефонний), перевірку доступності номерного фонду, формування підтвердження резервації та

управління листом очікування. Критичним ризиком при відсутності автоматизації є ситуація подвійного бронювання одного номера для різних гостей, що призводить до репутаційних та фінансових втрат.

– Управління номерним фондом. Передбачає ведення актуального реєстру номерів із зазначенням категорії, технічних характеристик (площа, поверх, планування, оснащення) та динамічного статусу.

– Управління замовленнями. Охоплює прийом та обробку замовлень у ресторані й барі, передачу замовлень на виробництво або друк чеків на кухонних принтерах, облік виконання замовлень та формування рахунків.

– Управління резервуванням столиків. Включає прийом заявок на бронювання столиків із прив'язкою до конкретного часового слоту, перевірку доступності з урахуванням тривалості обслуговування, підтвердження резервації та управління статусом столиків у реальному часі.

– Управління товарно-матеріальними запасами. Передбачає облік сировини та напівфабрикатів з підтримкою одиниць виміру (кг, г, л, мл), контроль залишків, формування замовлень постачальникам при досягненні мінімального порогу та списання за нормативами рецептур.

– Управління персоналом. Охоплює ведення кадрового реєстру з посадовою прив'язкою, облік контактних даних та ідентифікаційних номерів (ІПН/РНОКПП), а також формування зведеної інформації про фонд оплати праці в розрізі підрозділів.

– Формування управлінської звітності. Генерація оперативних і зведених звітів: показник завантаженості номерного фонду, середня ціна номера, виручка на доступний номер, аналіз продажів ресторану в розрізі страв та часових інтервалів, фінансовий звіт по комплексу.

Інформаційна архітектура ГРК характеризується складною системою потоків даних, які можна класифікувати за такими ознаками:

За напрямком: вхідні (дані від клієнтів, постачальників, зовнішніх систем

бронювання), внутрішні (міжсервісний обмін між підрозділами) та вихідні (звіти, підтвердження, фіскальні документи).

За типом даних: нормативно-довідкова інформація – класифікатори номерів, тарифні плани, меню, посадові довідники; транзакційні дані – записи бронювань, замовлень, платежів; аналітичні дані – агреговані показники, динамічні звіти.

За критичністю: дані реального часу (статуси номерів, поточні замовлення) потребують синхронного оновлення та негайної доступності для всіх служб; архівні дані (закриті бронювання, оплачені рахунки) зберігаються для аналітики та аудиту.

Ключовою проблемою при відсутності централізованої інформаційної системи є порушення цілісності та актуальності даних: дублювання записів у розрізнених облікових інструментах, неузгоджені статуси між підрозділами та неможливість отримання картини стану підприємства в режимі реального часу.

1.2 Аналіз існуючих інформаційних систем

Property Management System (PMS) є основним класом програмних рішень для автоматизації операційної діяльності засобів розміщення. Функціональне ядро PMS охоплює модулі: управління бронюваннями, служба прийому та розміщення, управління номерним фондом, управління тарифами, звітність. Провідними представниками є OPERA Cloud PMS [16], Protel Cloud PMS [17], Cloudbeds [18], Clock PMS+ [19], Mews [20].

Архітектурно сучасні PMS реалізовані як SaaS-рішення (Software as a Service) з хмарним розгортанням, це знижує капітальні витрати на IT-інфраструктуру. Вони підтримують інтеграцію з глобальними системами дистрибуції, менеджерами каналів та системами управління доходами [5].

У контексті ГРК PMS-системи мають суттєві обмеження: ресторанна функціональність або відсутня, або представлена лише у вигляді ручного внесення позицій до гостьового рахунку без повноцінного управління

замовленнями, меню та запасами. Крім того, ліцензійна вартість провідних рішень є суттєвим бар'єром для підприємств малого та середнього бізнесу.

Point of Sale (POS) – клас систем автоматизації торгівлі та обслуговування в закладах харчування. Функціональний склад: прийом і обробка замовлень, управління меню з модифікаторами та варіантами страв, друк чеків та фіскальних документів, облік продажів, управління столиками, базова аналітика продажів. Серед поширених рішень: Poster POS [21], Toast [22], Lightspeed Restaurant [23], Oracle Simphony [24].

Сучасні POS-системи реалізуються як хмарні рішення з веб-інтерфейсом та мобільними додатками для офіціантів, підтримують offline-режим роботи при втраті з'єднання. Однак вони спроектовані виключно для ресторанного сегменту: відсутня будь-яка інтеграція з готельним блоком, неможливе автоматичне перенесення ресторанних замовлень до гостьового рахунку, фінансова звітність формується відокремлено від готельного блоку [10, 13].

Ряд вендорів пропонує інтегровані рішення, які об'єднують PMS та POS у єдиній екосистемі зі спільною базою даних та наскрізним обліком гостей: Hotelogix, Maestro PMS, Opera з модулем Food & Beverage Management.

Ці рішення підтримують наскрізний гостьовий рахунок, централізовану звітність та єдиний профіль гостя. Проте вони орієнтовані на сегмент середнього та великого бізнесу що відображається у вартості впровадження та складності адаптації: підприємство змушене підлаштовувати власні процеси під жорстку логіку системи, а кастомізація потребує залучення вендора.

Загальні CRM-платформи (Salesforce, HubSpot, Pipedrive) дозволяють вести профілі клієнтів та автоматизувати маркетингові комунікації, проте не підтримують специфічних галузевих функцій: управління номерним фондом, дейт-специфічне бронювання, облік ресторанних замовлень. Облікові системи забезпечують бухгалтерський та складський облік, але є незручними для оперативного використання персоналом готелю та ресторанного залу.

Жоден із розглянутих класів рішень не забезпечує одночасно функціональну повноту, цінову доступність та гнучкість налаштування,

необхідні для підприємства малого та середнього масштабу (табл. 1.1) [7, 9]. Це обумовлює доцільність розробки спеціалізованого модульного рішення.

Таблиця 1.1

Порівняльний аналіз існуючих програмних рішень

Критерій	**PMS-системи**	**POS/Ресторанні системи**	**Комплексні рішення**	**CRM / Облікові системи**
Управління номерним фондом	Повністю	Відсутнє	Повністю	Відсутнє
Бронювання та резервування	Повністю	Відсутнє	Повністю	Часткове
Управління рестораном / POS	Часткове або відсутнє	Повністю	Повністю	Відсутнє
Єдиний рахунок гостя	Ускладнене	Відсутнє	Підтримується	Відсутнє
Облік персоналу	Базовий	Базовий	Середній	Розширений
Фінансова звітність	Готельний блок	Ресторанний блок	Консолідована	Загальна
Інтеграція між блоками	Слабка	Слабка	Висока	Відсутня
Гнучкість налаштування	Середня	Середня	Низька	Висока
Масштабованість	Висока	Висока	Висока	Висока
Вартість впровадження	Середня/Висока	Середня	Висока	Середня
Простота використання	Середня	Висока	Складна	Середня
Орієнтація на галузь	Готель	Ресторан	Готель + Ресторан	Загальний бізнес

1.3 Постановка задачі дослідження

На підставі аналізу предметної області та виявлених недоліків існуючих рішень визначається основний функціональний склад модуля інформаційно-управляючої системи (ІУС) ГРК:

а) Готельний підмодуль:

- Управління довідником номерного фонду (CRUD-операції над об'єктами Room, RoomType) з підтримкою статусної моделі (Available / Need Cleaning / Not Available)
- Управління бронюваннями з алгоритмом автоматичного пошуку вільного номера за типом і датами
- Управління гостьовими платежами та інтеграція з платіжними реєстрами
- Управління кадровим реєстром готелю (Personal, WorkPose)
- Облік інвентарю номерів (Stuff)
- Формування звіту завантаженості та фінансової статистики

б) Ресторанний підмодуль:

- Управління меню
- Управління замовленнями із зв'язком страва–столик та автоматичним розрахунком суми
- Управління резервуванням столиків
- Управління товарно-матеріальними запасами
- Управління кадровим реєстром ресторану
- Формування фінансової статистики ресторанного блоку

Модуль аналітики:

- Консолідований endpoint агрегації ключових показників обох блоків у реальному часі
- Показник завантаженості готелю
- Фінансові показники обох блоків та загальна виручка комплексу

Функціональні вимоги:

1. Система повинна забезпечувати повний CRUD (Create, Read, Update, Delete) для сутностей: Room, RoomType, Booking, Personal, WorkPose, Stuff, Payment (Hotel); Table, Dish, Order, Reserve, Raw, Personal, WorkPose, Payment (Restaurant).

2. Алгоритм пошуку вільного номера повинен виключати номери, для яких існують активні бронювання з перетином часового інтервалу.

3. Алгоритм резервування столика повинен виключати зайняті столики.

4. Система повинна підтримувати імпорт платіжних реєстрів з збереженням в базу даних для подальшої аналітики.

5. Система повинна надавати агреговані аналітичні endpoint: статистика платежів за діапазоном дат, звіт завантаженості номерного фонду, звіт по комплексу, звіт по персоналу.

6. Усі вхідні дані повинні проходити серверну валідацію з поверненням структурованих повідомлень про помилки у форматі JSON.

Нефункціональні вимоги:

1. Надійність. Система повинна забезпечувати цілісність даних за рахунок транзакційної обробки операцій запису та механізму foreign key constraints на рівні СУБД.

2. Розширюваність. Модульна архітектура системи повинна дозволяти додавання нових підмодулів без внесення змін до існуючого коду.

3. Документованість. API повинен мати автоматично генеровану документацію у форматі OpenAPI 3.0 (Swagger UI) [4, 15] на основі анотацій drf-spectacular.

4. Сумісність. Система повинна реалізовувати REST-архітектурний стиль із поверненням відповідей у форматі JSON та дотриманням HTTP-методів і кодів стану.

На підставі проведеного аналізу обирається наступний технологічний підхід:

– Архітектурний стиль – REST API (Representational State Transfer). Обґрунтування: stateless-природа REST забезпечує горизонтальну масштабованість без необхідності зберігання серверних сесій; уніфікований інтерфейс дозволяє підключати будь-які клієнтські застосунки без змін на стороні сервера; висока підтримка в індустрії та зрілість інструментарію.

– Backend-фреймворк – Django + Django REST Framework (DRF). Обґрунтування: Django реалізує архітектурний патерн MTV (Model-Template-View), забезпечує вбудовану ORM (Object-Relational Mapping) з підтримкою

міграцій схеми БД, адміністративну панель. DRF надає декларативний механізм серіалізації, класи представлень (views), систему дозволів (permissions) та вбудовану підтримку OpenAPI [2].

– СУБД – SQLite. Для розробки та тестування використовується SQLite завдяки відсутності необхідності окремого сервера.

– Документація drf-spectacular (OpenAPI 3.0). Автоматична генерація специфікації API з анотацій коду забезпечує актуальність документації та можливість тестування через Swagger UI.

1.4 Висновки до розділу 1

У першому розділі проведено системний аналіз предметної області – готельно-ресторанного комплексу як об'єкта автоматизації. Встановлено, що ГРК є складною системою з розподіленою відповідальністю, в якій готельний і ресторанний блоки функціонують як операційно відмінні, але інформаційно взаємозалежні підсистеми.

Визначено ключові бізнес-процеси, що підлягають автоматизації: управління бронюваннями та номерним фондом, управління замовленнями та резервуванням столиків, управління персоналом і запасами, формування консолідованої управлінської звітності.

Порівняльний аналіз існуючих рішень (PMS, POS, комплексні системи, CRM) показав відсутність на ринку доступного, гнучкого та функціонально повного рішення для підприємств малого та середнього масштабу. Комплексні рішення є економічно невиправданими та негнучкими; вузькоспеціалізовані – не забезпечують міжблокової інтеграції.

Сформульовано функціональні та нефункціональні вимоги до модуля ІУС, обґрунтовано вибір технологічного стеку та архітектурного підходу, що відповідає вимогам розширюваності, стабільності та документованості системи.

РОЗДІЛ 2
ПРОЕКТУВАННЯ МОДУЛЯ ІНФОРМАЦІЙНО-УПРАВЛЯЮЧОЇ СИСТЕМИ ГОТЕЛЬНО-РЕСТОРАННОГО КОМПЛЕКСУ

2.1 Вибір технологічного стеку та обґрунтування архітектурних рішень

На підставі аналізу функціональних та нефункціональних вимог, сформульованих у розділі 1, як основний архітектурний стиль для розробки серверної частини модуля ІУС обрано REST (Representational State Transfer). REST є набором архітектурних обмежень і не є протоколом або стандартом – він встановлює принципи організації взаємодії між клієнтом і сервером через HTTP.

Ключовими принципами REST, що визначили вибір цього архітектурного стилю, є наступні:

– Stateless (відсутність стану на сервері): кожен HTTP-запит містить усю необхідну для його обробки інформацію; сервер не зберігає стан сесії між запитами. Це забезпечує горизонтальну масштабованість системи - будь-який сервер у кластері може обробити будь-який запит незалежно від попередніх.

– Uniform Interface (уніфікований інтерфейс): ресурси ідентифікуються через URI (наприклад, /api/hotel/room/), стан передається через представлення (JSON), а взаємодія здійснюється через стандартні HTTP-методи (GET, POST, PUT, PATCH, DELETE).

– Client-Server (розділення клієнта і сервера): серверна частина не залежить від конкретної реалізації клієнта – до одного REST API можуть звертатись веб-застосунок, мобільний додаток, desktop-клієнт або стороння система.

– Cacheable (кешованість): HTTP-відповіді можуть бути кешовані, що знижує навантаження на сервер при повторних запитах до рідко змінюваних ресурсів (наприклад, списку типів номерів або меню ресторану).

Порівняно з альтернативними архітектурними підходами – GraphQL та gRPC – REST є більш зрілим та широко підтримуваним рішенням. GraphQL надає гнучкість формування запитів на стороні клієнта, проте потребує значно складнішої схеми, що є надлишковим для системи з чітко визначеними сутностями та операціями [26]. gRPC забезпечує найвищу продуктивність, однак його підтримка у браузерах потребує додаткового проксі-рівня, а поріг входу суттєво вищий. Для системи класу CRUD-based бізнес-логіки ГРК REST є оптимальним вибором за критерієм "функціональність/складність" [11].

Для реалізації серверної частини обрано Django [12] у поєднанні з Django REST Framework (DRF). Серед ключових переваг Django для даного проекту:

– Вбудована ORM: Django ORM забезпечує декларативне оголошення структури бази даних у вигляді Python-класів (models.py) та автоматичну генерацію SQL-схеми через систему міграцій. Це виключає необхідність ручного написання DDL-запитів та забезпечує незалежність від конкретної СУБД.

– Адміністративна панель (Django Admin): автоматично генерована CRUD-панель для управління даними через веб-інтерфейс суттєво прискорює розробку та тестування, знижує потребу в додаткових інструментах управління даними.

– Спільнота: Django існує з 2005 року та має детальну документацію та велику спільноту.

Django REST Framework як розширення для REST API. DRF надає декларативний шар абстракції для побудови REST API на базі Django:

– Serializers: класи, що відповідають за двосторонню трансформацію між Python-об'єктами (моделями Django) та JSON-представленням. DRF Serializers підтримують вкладену серіалізацію, валідацію полів та об'єктного рівня, що дозволяє централізувати логіку перевірки даних.

– API Views та ViewSets: DRF надає як функціональні (function-based views з декоратором @api_view), так і класові (class-based views, ModelViewSet) підходи до визначення обробників запитів. У даному проекті використовуються функціональні view для більшого контролю над логікою обробки.

– Browsable API: вбудований HTML-рендерер DRF автоматично генерує інтерактивний веб-інтерфейс для тестування API-ендпоінтів у браузері без додаткових інструментів.

Порівняно з альтернативами – FastAPI та Flask – Django+DRF є більш повним рішенням. FastAPI забезпечує кращу продуктивність завдяки асинхронному виконанню, проте для CRUD-систем з реляційними БД перевага FastAPI нівелюється накладними витратами на ORM. Flask є мінімалістичним фреймворком і потребує вибору та інтеграції сторонніх компонентів для кожної задачі, що збільшує час розробки. Для академічного проекту з чітко визначеними вимогами Django+DRF забезпечує найкраще співвідношення зрілості інструментарію та швидкості розробки [29].

База даних. Для зберігання даних у середовищі розробки та прототипування обрано SQLite – вбудовану файлову СУБД, що не потребує встановлення окремого сервера. SQLite є стандартним вибором для розробки Django-проектів: Django за замовчуванням налаштований на роботу з SQLite, що дозволяє розгорнути проект без конфігурування зовнішніх сервісів [14]. Реляційна модель даних SQLite повністю відповідає структурі сутностей системи, а Django ORM забезпечує переносимість на PostgreSQL або MySQL при переході до промислового розгортання.

Документування API. Для автоматичного генерування специфікації API у форматі OpenAPI 3.0 використовується пакет drf-spectacular. Він інспектує URL-конфігурацію та View-класи Django REST Framework і формує YAML/JSON-опис усіх ендпоінтів, включаючи схеми запитів і відповідей. На основі сформованої специфікації drf-spectacular надає Swagger UI –

інтерактивну веб-консоль для тестування API. Це забезпечує актуальність документації та можливість тестування ендпоінтів без додаткових клієнтів.

Парсинг Excel-реєстрів. Для імпорту платіжних реєстрів у форматі .xlsx використовується бібліотека pandas [6] разом з openpyxl [3]. pandas забезпечує зручне читання та трансформацію табличних даних, а перейменування стовпців (функція rename) дозволяє відобразити українськомовні заголовки реєстру на поля моделі Payment.

2.2 Проектування архітектури системи

Модуль ІУС ГРК реалізовано як єдиний Django-проект з модульною структурою Django-застосунків (apps). Таке рішення відповідає принципу "розділення відповідальності" та дозволяє незалежно розробляти та тестувати окремі підсистеми [30].

Проект складається з трьох Django-застосунків:

– Hotel – містить моделі, серіалізатори, view та URL-конфігурацію готельного підмодуля. Відповідає за управління номерним фондом, бронюваннями, клієнтами, персоналом готелю та інвентарем номерів.

– Restaurant – містить аналогічні компоненти для ресторанного підмодуля. Відповідає за управління столиками, меню, замовленнями, резервуванням, запасами та персоналом ресторану.

– Dashboard (кореневий застосунок) – містить єдиний агрегуючий ендпоінт /api/dashboard/, котрий об'єднує ключові показники обох підмодулів у реальному часі.

Кожен застосунок дотримується однакової внутрішньої структури файлів: models.py (моделі даних), serializers.py (серіалізатори та валідація), views.py (обробники HTTP-запитів), services.py (бізнес-логіка, ізольована від шару HTTP), urls.py (URL-маршрутизація), errors.py (централізовані повідомлення про помилки). Виокремлення бізнес-логіки у services.py дозволяє тестувати логіку незалежно від HTTP-шару (див. рис. 2.1).

Система реалізована у багаторівневій архітектурі, де кожен рівень виконує чітко визначену функцію та взаємодіє лише з суміжними рівнями.

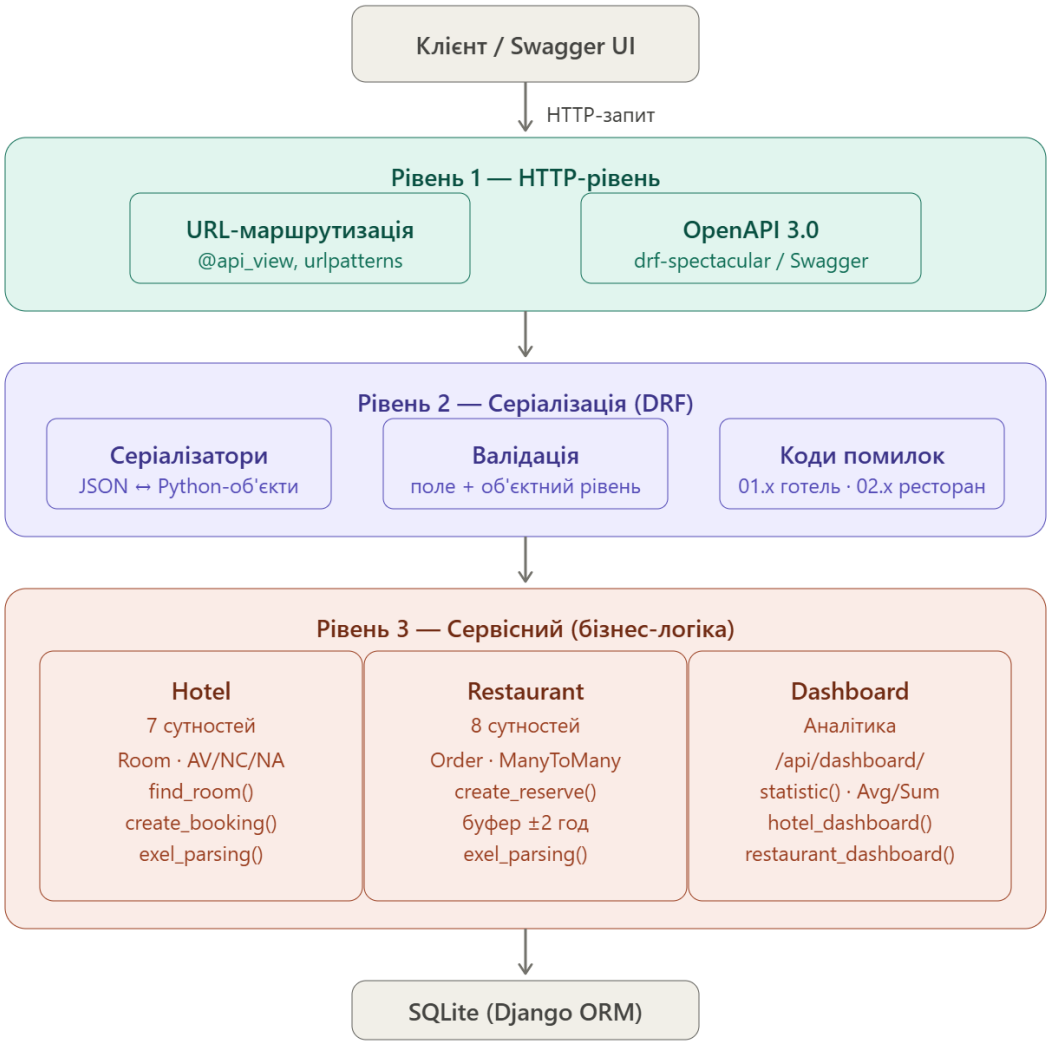

Рисунок 2.1. Загальна архітектура модуля ІУС ГРК

Рівень HTTP (View Layer). Функціональні view, декоровані @api_view, приймають HTTP-запити, делегують валідацію серіалізаторам, викликають відповідні сервісні функції та формують HTTP-відповіді з правильними кодами стану. View не містять бізнес-логіки – вони виконують роль тонкого "клею" між HTTP-протоколом і логікою застосунку. Усі view повертають відповіді у форматі JSON через клас Response Django REST Framework [28].

Рівень серіалізації та валідації (Serializer Layer). Серіалізатори DRF виконують дві ключові функції: перетворення JSON-запиту в Python-словник (десеріалізація) та перетворення Django-моделі у JSON-відповідь

(серіалізація) [27]. Метод validate() в кожному серіалізаторі реалізує бізнес-правила валідації: перевірку формату ІПН, довжини номера телефону, коректності діапазону дат, позитивності числових значень. При виявленні помилки серіалізатор повертає структурований JSON з кодом помилки та повідомленням через функцію error_response() з модуля errors.py.

Рівень сервісів (Service Layer). Модуль services.py кожного застосунку містить функції, що реалізують складну бізнес-логіку: алгоритм пошуку вільного номера, алгоритм резервування столика, агрегацію аналітичних показників, парсинг Excel-реєстрів. Сервісні функції взаємодіють безпосередньо з Django ORM (моделями) і не залежать від HTTP-контексту.

Рівень даних (Model Layer). Django-моделі описують структуру реляційної бази даних. ORM автоматично генерує SQL-запити на основі викликів Python API (Model.objects.filter(), .aggregate(), .create() тощо), забезпечуючи незалежність від конкретної СУБД.

2.3 Проектування моделей даних

Готельний підмодуль оперує наступними сутностями, реалізованими у вигляді Django-моделей (рис. 2.2).

Модель WorkPose (Посада) є базовою довідниковою сутністю, що визначає кадровий класифікатор підприємства. Містить унікальну назву посади (поле name з обмеженням unique=True), необов'язковий опис (definition) та заробітну плату (salary типу DecimalField з точністю до двох знаків). Зв'язок з моделлю Personal реалізовано як ForeignKey з посиланням по полю name (to_field='name') замість стандартного первинного ключа, що спрощує читаємість запитів.

Модель Personal (Персонал) зберігає кадровий реєстр співробітників. Ключові поля: ідентифікаційний номер платника податків (inn, max_length=10, unique=True), контактні дані (email_address, phone_number обидва з унікальністю), фото (ImageField з параметром upload_to='personal'). Зовнішній

ключ work_pos на WorkPose з поведінкою SET_NULL при видаленні посади забезпечує цілісність даних.

Модель RoomType (Тип кімнати) є довідником категорій номерного фонду з унікальною назвою та необов'язковим описом. Використовується для класифікації номерів та є критичним полем в алгоритмі пошуку доступних кімнат.

Модель Room (Кімната) є центральною сутністю готельного підмодуля. Унікальний номер кімнати (SmallIntegerField) слугує природним ключем. Статусна модель реалізована через CharField з обмеженим набором значень (choices): AV (Available), NC (Need Cleaning), NA (Not Available). Зовнішній ключ на RoomType та поле price (DecimalField) дозволяють динамічно розраховувати вартість бронювання.

Модель Client (Клієнт) зберігає контактні дані гостей готелю. Поля phone_number та email_address є унікальними. Зовнішній ключ на Client у моделі Booking реалізований як ForeignKey з посиланням на первинний ключ.

Модель Booking (Бронювання) є ключовою транзакційною сутністю готельного блоку. Поле pay_id (унікальний ідентифікатор оплати) є пов'язуючим ключем між системою бронювань та платіжним реєстром. Дати початку і кінця бронювання (date_start, date_end типу DateField) є основою алгоритму перевірки доступності номерів. Булеві прапорці is_pay та is_active відстежують стан оплати та активності бронювання.

Модель Stuff (Інвентар номерів) забезпечує прив'язку матеріальних цінностей до конкретних кімнат через ForeignKey на Room.

Модель Payment (Оплата) є спільною для обох підмодулів і зберігає записи про здійснені платежі, імпортовані з банківських виписок у форматі Excel. Унікальний payment_id дозволяє уникнути дублювання записів при повторному імпорті.

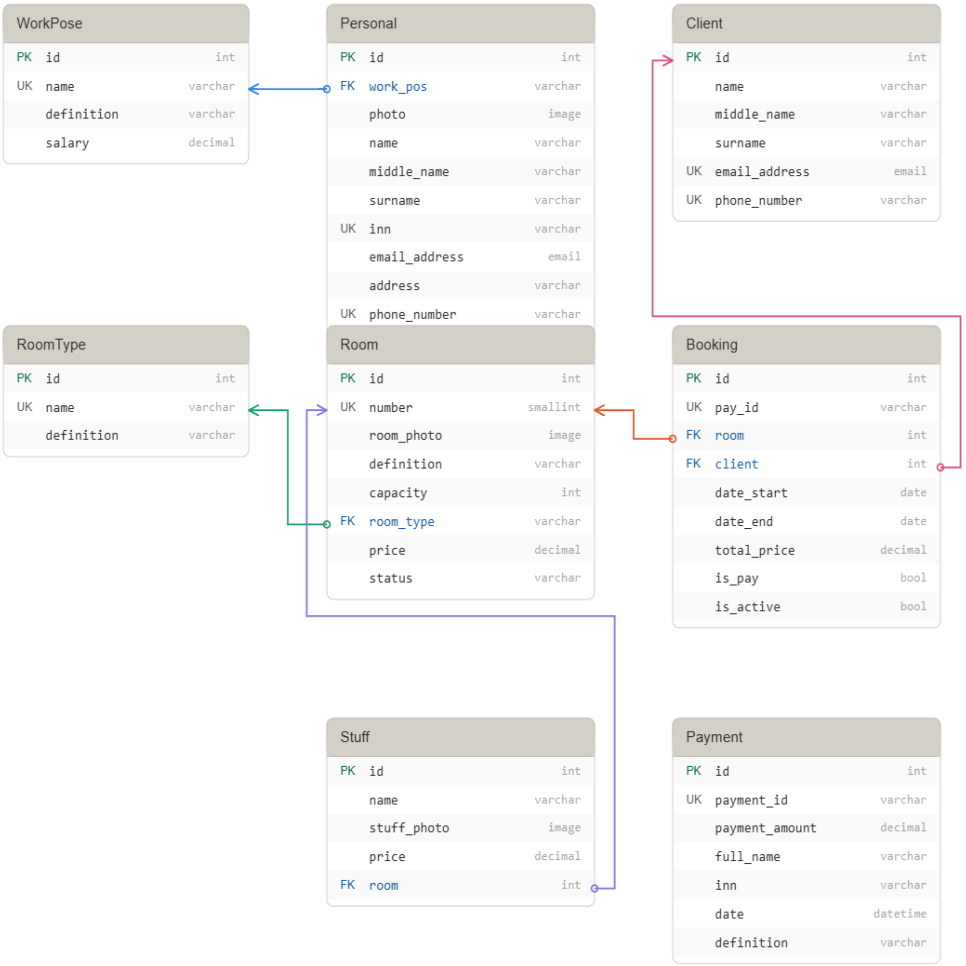

Рисунок 2.2. Фізична модель бази даних готельного підмодуля

Ресторанний підмодуль включає наступні моделі.

Модель Table (Столик) містить унікальний номер столика та булевий прапорець is_available, що відображає поточну доступність. Модель є основою для алгоритму резервування.

Модель Dish (Страва) описує позиції меню. Поле is_available дозволяє тимчасово приховувати страву з можливості замовлення (наприклад, при відсутності інгредієнтів) без видалення запису. Фотографія страви зберігається як ImageField.

Модель Order (Замовлення) є центральною транзакційною сутністю ресторанного блоку. Зв'язок ManyToMany з Dish реалізовано через стандартну проміжну таблицю Django, дозволяючи прив'язати до одного замовлення

довільну кількість страв. Поле total_price перераховується сервісною функцією check_total_pay() при зміні складу замовлення.

Модель Reserve (Резервування столика) фіксує факт бронювання конкретного столика на певний час. Прапорець is_active дозволяє скасовувати бронювання без фізичного видалення запису, що є важливим для аналітики.

Модель Raw (Сировина) забезпечує облік складських запасів. Поле unit з обмеженим набором одиниць вимірювання (Kg, G, L, Ml) дозволяє зберігати дані про різнорідні продукти в уніфікованому форматі.

Ресторанний підмодуль використовує власні сутності WorkPose, Personal та Payment з аналогічною структурою до готельного підмодуля, але в окремому Django-застосунку, що забезпечує незалежність кадрового обліку двох підрозділів (рис. 2.3).

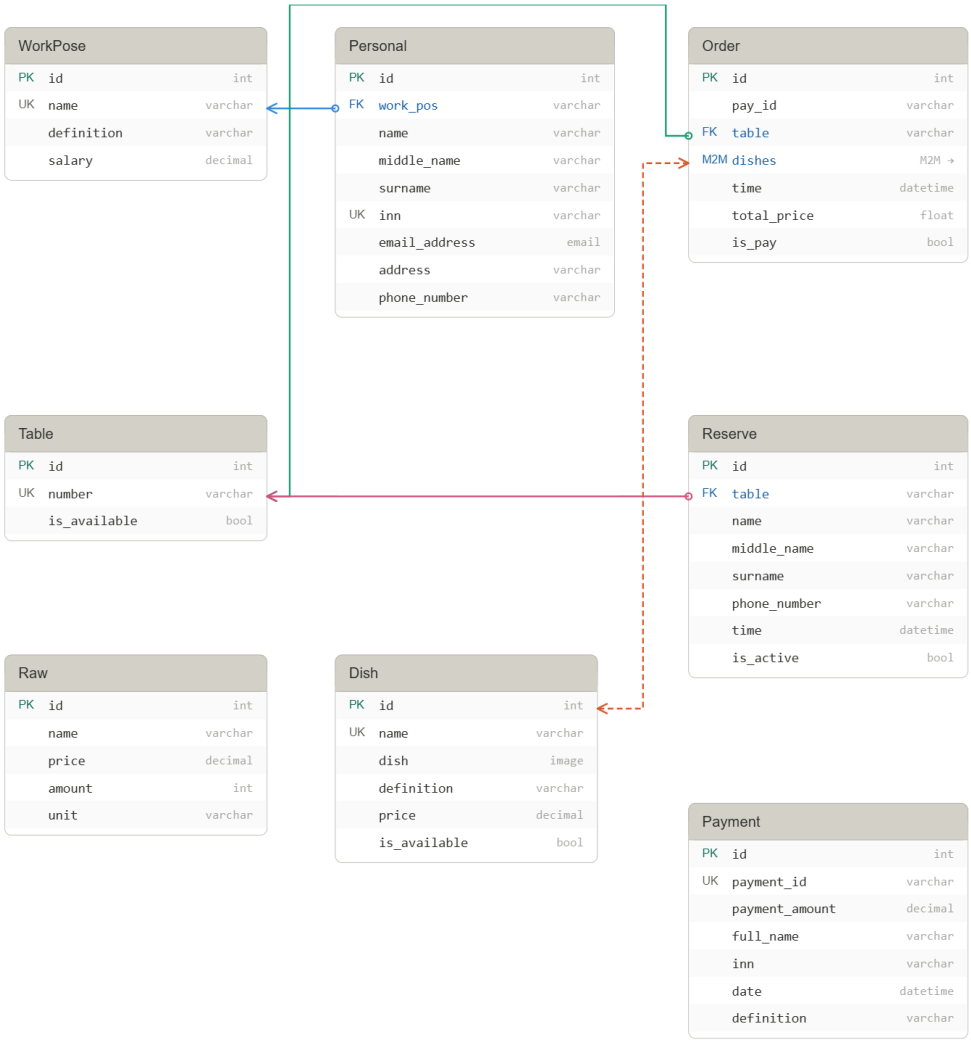

Рисунок 2.3. Фізична модель бази даних ресторанного підмодуля

2.4 Проектування бізнес-логіки та алгоритмів

Алгоритм find_room(), реалізований у services.py готельного підмодуля (див. рис. 2.4), є ключовим компонентом системи бронювань. Задача алгоритму: за заданим типом кімнати та діапазоном дат [date_start, date_end) знайти перший доступний номер.

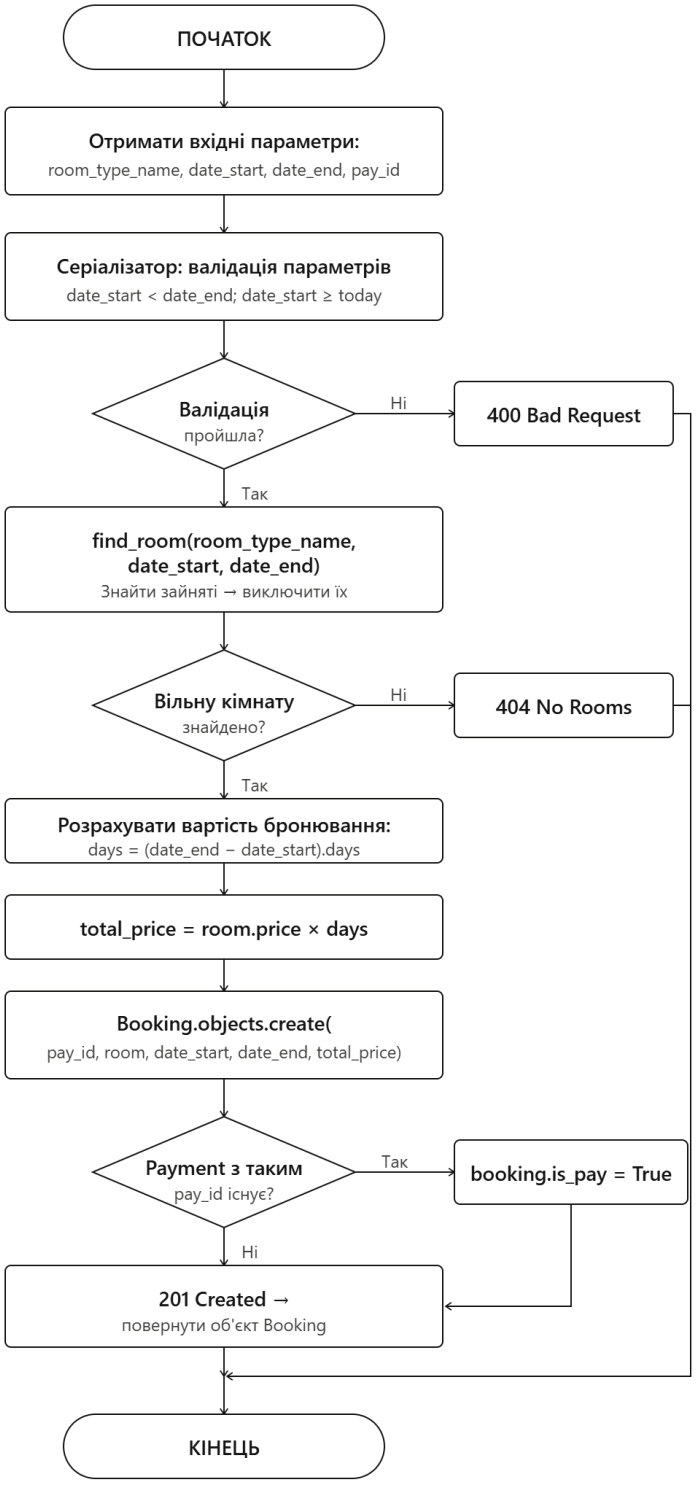

Рисунок 2.4. Схема алгоритму створення бронювання

Алгоритм реалізовано у два кроки:

1. Визначення зайнятих номерів: через фільтрацію активних бронювань (is_active=True) з перетином заданого часового інтервалу. Умова перетину інтервалів записується як date_start__lt=date_end AND date_end__gt=date_start, що відповідає класичному алгоритму перевірки перетину відрізків. Запит повертає множину номерів кімнат, що вже заброньовані.

2. Пошук доступного номера: через виключення зайнятих номерів та фільтрацію за типом кімнати. Метод .first() повертає перший доступний запис або None.

```
def find_room(room_type_name, date_start, date_end):
    occupied_room = Booking.objects.filter(
        is_active=True,
        date_start__lt=date_end,
        date_end__gt=date_start
    ).values_list('room__number', flat=True)

    available_room = Room.objects.filter(
        room_type__name=room_type_name
    ).exclude(number__in=occupied_room).first()

    return available_room
```

Лістинг 2.1. Алгоритм пошуку вільного номера

Після знаходження доступного номера функція create_booking() розраховує загальну вартість бронювання на основі кількості ночей (date_end - date_start).days та ціни кімнати. Додатково виконується перевірка наявності відповідного запису оплати в таблиці Payment (на випадок, якщо оплата надійшла раніше, ніж бронювання).

Алгоритм create_reserve(), реалізований у services.py ресторанного підмодуля, вирішує задачу автоматичного розподілу столиків при резервуванні. На відміну від готельного бронювання (де тривалість визначається датами), ресторанне резервування прив'язане до конкретного часу з фіксованим "вікном" обслуговування (±2 години) (лістинг 2.2).

Алгоритм реалізовано наступним чином:

1. Визначення часового вікна: часовий інтервал можливого конфлікту резервувань обчислюється як [time - 2 години, time + 2 години]. Цей

підхід означає, що для кожного замовленого столика резервується "буфер" у 4 години – час, протягом якого столик вважається зайнятим.

2. Пошук зайнятих столиків: активні резервування (is_active=True), що перетинаються з визначеним часовим вікном, формують множину зайнятих столиків.

3. Вибір вільного столика: перший столик, що не входить до множини зайнятих, стає місцем резервування. Якщо вільних столиків немає – алгоритм повертає помилку, що сигналізує про неможливість резервування.

```
def create_reserve(time, name, middle_name, surname, phone_number):
    start_limit = time - timedelta(hours=2)
    end_limit = time + timedelta(hours=2)

    occupied_tables = Reserve.objects.filter(
        time__range=(start_limit, end_limit),
        is_active=True
    ).values_list('table__number', flat=True)

    available_table = Table.objects.exclude(
        number__in=occupied_tables
    ).first()

    if not available_table:
        return None

    reserve = Reserve.objects.create(
        table=available_table, time=time,
        name=name, middle_name=middle_name,
        surname=surname, phone_number=phone_number
    )
    return reserve
```

Лістинг 2.2. Алгоритм резервування столика

Для інтеграції з банківськими виписками реалізовано функцію exel_parsing() (присутня в обох підмодулях), вона забезпечує пакетний імпорт платежів з Excel-файлів.

Алгоритм функціонує у три кроки:

1. Файл .xlsx зчитується бібліотекою pandas. Стовпці перейменовуються з українськомовних назв банківського реєстру ('id_оплати', 'сума', 'ПІБ', 'РНОКПП', 'дата', 'прич. платежу') на відповідні поля моделі Payment. Перетворений DataFrame конвертується в список словників.

2. Виконується зіставлення платежів з існуючими записами: для кожного запису реєстру перевіряється наявність бронювання або замовлення

з відповідним pay_id. При знаходженні збігу встановлюється прапорець is_pay = True. Це дозволяє автоматично підтверджувати оплату без ручного ввімкнення кожного запису.

3. Перетворені дані повертаються view-функції для подальшого збереження через серіалізатор PaymentSerializer, який забезпечує валідацію кожного запису перед записом у базу даних.

Функції hotel_dashboard() та restaurant_dashboard() формують агреговані показники стану кожного підмодуля. Дашборд реалізовано як єдиний GET-ендпоінт, що викликає обидві функції та об'єднує результати.

Функція hotel_dashboard() повертає такі показники:

– Статус номерного фонду: загальна кількість, розподіл за статусами (Available / Need Cleaning / Not Available), кількість активних бронювань на поточний день та відсоток завантаженості (occupancy rate).

– Фінансові показники: виручка поточного місяця (сума підтверджених платежів) та кількість незакритих (неоплачених) бронювань.

Функція restaurant_dashboard() повертає:

– Статус залу: загальна кількість столиків, кількість вільних та зайнятих, кількість активних резервувань на поточний день.

– Показники замовлень: загальна кількість замовлень на поточний день, кількість оплачених та неоплачених.

– Фінансові показники: виручка поточного дня та поточного місяця.

– Показники меню: кількість доступних та недоступних страв.

Консолідований ендпоінт /api/dashboard/ додатково розраховує загальну виручку комплексу за поточний місяць як суму виручок обох підмодулів, що є ключовим управлінським показником для керівництва ГРК.

2.5 Проектування API-інтерфейсу

URL-простір API організовано за принципом ієрархічної вкладеності ресурсів з дотриманням принципів REST. Список всіх ендпоінтів готельного модуля наведено в таблиці 2.1

Таблиця 2.1

Структура URL-маршрутів API готельного підмодуля

URL-шаблон	**HTTP-метод**	**Призначення**
/api/hotel/personal/	GET, POST	Список персоналу / Додати співробітника
/api/hotel/personal/detail/<pk>	GET, PUT, DELETE	Деталі, редагування, видалення співробітника
/api/hotel/room/	GET, POST	Список кімнат / Додати кімнату
/api/hotel/room/detail/<pk>	GET, PUT, DELETE	Деталі, редагування, видалення кімнати
/api/hotel/room/status_av/<pk>	PATCH	Встановити статус Available
/api/hotel/room/status_na/<pk>	PATCH	Встановити статус Not Available
/api/hotel/booking/	GET, POST	Список бронювань / Переглянути
/api/hotel/booking/create/	POST	Створити бронювання (алгоритм пошуку кімнати)
/api/hotel/booking/detail/<pk>	GET, PUT, DELETE	Деталі, редагування, видалення бронювання
/api/hotel/cancel_booking/<pk>	PATCH	Скасувати бронювання
/api/hotel/pay_booking/<pk>	PATCH	Підтвердити оплату
/api/hotel/payment/	GET, POST	Список платежів / Додати платіж
/api/hotel/excel_parsing/	POST	Імпорт платіжного реєстру з Excel
/api/hotel/stats/	GET	Фінансова статистика за діапазоном дат
/api/hotel/occupancy/	GET	Звіт завантаженості номерного фонду
/api/hotel/personal_report/	GET	Звіт по персоналу в розрізі посад

Усі ендпоінти API повертають відповіді у форматі JSON. Для успішних операцій структура відповіді визначається серіалізатором відповідної моделі. Для помилкових ситуацій встановлено уніфікований формат відповіді з помилкою, що дає змогу клієнту однозначно ідентифікувати тип помилки:

```
{
  "error": {
    "error_code": "02.01",
    "message": "ІПН повинен складатись лише з цифр"
  }
}
```

Лістинг 2.3. Формат JSON-відповіді при помилці валідації

Коди помилок побудовані за дворівневою системою: перша частина (01 або 02) ідентифікує підмодуль (Hotel або Restaurant), друга частина – конкретний тип помилки. Така система дозволяє клієнтській стороні обробляти помилки програмно, не покладаючись на текст повідомлення.

HTTP-коди стану використовуються у відповідності до семантики REST:

– 200 OK – успішне отримання або оновлення даних.

– 201 Created – успішне створення нового ресурсу.

– 400 Bad Request – помилка валідації вхідних даних.

– 404 Not Found – ресурс за вказаним ідентифікатором не знайдено.

– 405 Method Not Allowed – метод не підтримується для даного ендпоінта.

Валідація вхідних даних реалізована на двох рівнях: рівні поля та рівні об'єкта.

Валідація на рівні поля виконується автоматично Django REST Framework на основі оголошень типів полів у моделі та серіалізаторі (наприклад, EmailField автоматично перевіряє формат email-адреси, DecimalField – числовий тип та кількість знаків).

Валідація на рівні об'єкта реалізована у методі validate() кожного серіалізатора. Метод validate() отримує словник усіх валідованих полів і може перевіряти їх у сукупності:

– PersonalSerializer перевіряє: цифровий формат ІПН (inn.isdigit()), довжину ІПН (10 знаків), цифровий формат телефону (phone_number.isdigit()), допустиму довжину телефону (10 або 12 знаків для форматів +380XXXXXXXXX та 0XXXXXXXXX).

– BookingSerializer перевіряє: дата початку не раніше поточного дня (date_start >= date.today()), дата початку раніше дати кінця (date_start < date_end).

– StatSerializer перевіряє: обидва кінці діапазону знаходяться в минулому, початок діапазону раніше його кінця.

– CreateReserveSerializer перевіряє: час резервування мінімум за 2 години від поточного моменту.

При виявленні помилки валідації метод підіймає виняток serializers.ValidationError з результатом функції error_response(), що дає уніфікований формат відповіді.

2.6 Висновки до розділу 2

У другому розділі виконано повне проектування модуля інформаційно-управляючої системи готельно-ресторанного комплексу. Обґрунтовано вибір технологічного стеку: архітектурний стиль REST API, фреймворк Django у поєднанні з Django REST Framework, СУБД SQLite для прототипування та бібліотека pandas для обробки Excel-реєстрів.

Спроектовано модульну архітектуру системи у вигляді трьох незалежних Django-застосунків (Hotel, Restaurant, Dashboard) з уніфікованою внутрішньою структурою (models, serializers, views, services, urls, errors). Багатошарова архітектура (HTTP-шар, шар серіалізації/валідації, сервісний шар) забезпечує ізоляцію бізнес-логіки від HTTP-контексту та спрощує тестування.

Спроектовано реляційні моделі даних для обох підмодулів з визначенням зв'язків між сутностями, обмежень унікальності та поведінки при видаленні (ON DELETE SET NULL). Розроблено ключові алгоритми системи: алгоритм пошуку вільного номера на основі перевірки перетину часових інтервалів, алгоритм резервування столика з часовим буфером ±2 години, механізм імпорту та зіставлення платіжних реєстрів. Спроектовано структуру API-інтерфейсу з дотриманням конвенцій REST, уніфікованим форматом помилок та двошаровою системою валідації вхідних даних.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МОДУЛЯ ІНФОРМАЦІЙНО-УПРАВЛЯЮЧОЇ СИСТЕМИ ГОТЕЛЬНО-РЕСТОРАННОГО КОМПЛЕКСУ

3.1 Реалізація структури проекту та налаштування середовища

Модуль ІУС ГРК реалізовано у вигляді єдиного Django-проекту з трьома незалежними застосунками. Файлова структура проекту організована наступним чином:

```
hotel_restaurant_system/
    manage.py
    hotel_restaurant_system/
        settings.py
        urls.py
        wsgi.py
    Hotel/
        models.py
        serializers.py
        views.py
        services.py
        urls.py
        errors.py
    Restaurant/
        models.py
        serializers.py
        views.py
        services.py
        urls.py
        errors.py
    Dashboard/
        views.py
        urls.py
```

Лістинг 3.1. Файлова структура Django-проекту

Кореневий файл urls.py проекту підключає маршрути всіх трьох застосунків через механізм include() з відповідними префіксами: /api/hotel/ для готельного підмодуля, /api/restaurant/ для ресторанного та /api/ для дашборду. Додатково підключаються маршрути drf-spectacular для автоматичної генерації OpenAPI-специфікації (/api/schema/) та Swagger UI (/api/docs/).

Файл settings.py налаштовує базові параметри проекту (переглянути залежності можна в табл. 3.1). До списку INSTALLED_APPS додано всі три застосунки разом з rest_framework та drf_spectacular. База даних налаштована на SQLite з файлом db.sqlite3 у кореневій директорії. Конфігурація Django

REST Framework задається через словник REST_FRAMEWORK, де визначаються класи рендерингу відповідей. Конфігурація drf-spectacular задається через SPECTACULAR_SETTINGS з визначенням назви API та версії.

Таблиця 3.1

Залежності проекту

Пакет	**Версія**	**Призначення**
Django	6.0.3	Основний веб-фреймворк (MTV-архітектура, ORM, Admin)
djangorestframework	3.16.1	Побудова REST API, серіалізатори, @api_view
drf-spectacular	0.29	Генерація OpenAPI 3.0 специфікації та Swagger UI
pandas	3.0.1	Читання та трансформація Excel-реєстрів (.xlsx)
openpyxl	3.1.5	Рушій читання .xlsx (залежність pandas)
pillow	12.0.0	Обробка зображень для полів ImageField у моделях

3.2 Реалізація готельного підмодуля

Моделі готельного підмодуля реалізовані у файлі Hotel/models.py відповідно до спроектованої ER-діаграми.

Модель Room реалізує статусну модель через параметр choices поля status. Це обмежує множину допустимих значень на рівні Django-валідації та формує відповідне обмеження у СУБД. Поле room_type є ForeignKey на RoomType з параметром to_field='name', що дозволяє звертатися до типу кімнати за назвою, що підвищує читаємість API-відповідей.

Модель Booking використовує on_delete=SET_NULL для зовнішніх ключів на Room та Client. При видаленні кімнати або клієнта відповідне поле стає NULL, але запис бронювання зберігається – що важливо для збереження фінансової історії. Поле pay_id оголошено унікальним (unique=True), що запобігає дублюванню бронювань з однаковим платіжним ідентифікатором.

```
class Booking(models.Model):
    pay_id = models.CharField(max_length=255, unique=True,
default="Error")
    room        = models.ForeignKey(Room, on_delete=models.SET_NULL,
                      null=True, to_field="number")
    client = models.ForeignKey(Client, on_delete=models.SET_NULL,
null=True)
    date_start = models.DateField()
    date_end = models.DateField()
    total_price = models.DecimalField(max_digits=10, decimal_places=2,
                      null=True, blank=True)
    is_pay = models.BooleanField(default=False)
    is_active = models.BooleanField(default=True)
```

Лістинг 3.2. Реалізація моделі Booking

Серіалізатори готельного підмодуля реалізовані у файлі Hotel/serializers.py. Більшість є підкласами ModelSerializer, що автоматично генерує поля на основі визначення моделі. Метод validate() реалізує специфічну бізнес-логіку перевірки на рівні об'єкта.

PersonalSerializer перевіряє формат і довжину ідентифікаційного номера платника податків та номера телефону. Валідація реалізована через методи рядка isdigit() та len() без залежності від зовнішніх бібліотек. При невалідному вводі метод validate() підіймає serializers.ValidationError з результатом функції error_response(), що формує уніфікований JSON зі структурою {error: {error_code, message}}.

CreateBookingSerializer є підкласом Serializer (не ModelSerializer), оскільки приймає параметри для алгоритму пошуку (room_type_name, date_start, date_end, pay_id), а не прямі поля моделі. Це відокремлює інтерфейс API від внутрішньої структури БД.

```
class CreateBookingSerializer(serializers.Serializer):
    room_type_name = serializers.CharField(max_length=255)
    date_start     = serializers.DateField()
    date_end       = serializers.DateField()
    pay_id         = serializers.CharField(max_length=255)

    def validate(self, data):
        date_start = data.get("date_start")
        date_end   = data.get("date_end")
        if date_start >= date_end:
            raise serializers.ValidationError(error_response("01.10"))
        if date_start < date.today():
            raise serializers.ValidationError(error_response("01.11"))
        return data
```

Лістинг 3.3. Реалізація CreateBookingSerializer

View-функції реалізовані у Hotel/views.py з використанням декоратора @api_view, що трансформує Python-функцію у DRF-сумісний HTTP-обробник. Декоратор отримує список дозволених методів і автоматично повертає 405 Method Not Allowed для решти. Стандартний CRUD-патерн реалізовано через пари list/detail функцій – наприклад, room_list (GET список та POST створення) та room_detail (GET деталі, PUT оновлення, DELETE видалення).

```
@api_view(["GET", "POST"])
def room_list(request):
    if request.method == "GET":
        rooms = Room.objects.all()
        return Response(RoomSerializer(rooms, many=True).data)
    serializer = RoomSerializer(data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data,
status=status.HTTP_201_CREATED)
    return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

@api_view(["GET", "PUT", "DELETE"])
def room_detail(request, pk):
    try:
        room = Room.objects.get(pk=pk)
    except Room.DoesNotExist:
        return Response(error_response("01.17"),
                        status=status.HTTP_404_NOT_FOUND)
    if request.method == "GET":
        return Response(RoomSerializer(room).data)
    if request.method == "PUT":
        s = RoomSerializer(room, data=request.data)
        if s.is_valid():
            s.save()
            return Response(s.data)
        return Response(s.errors,
status=status.HTTP_400_BAD_REQUEST)
    room.delete()
    return Response(status=status.HTTP_204_NO_CONTENT)
```

Лістинг 3.4. Реалізація room_list та room_detail

Для операцій зміни статусу, що не вкладаються у стандартний CRUD, реалізовані окремі ендпоінти. Функція status_av() встановлює status="AV" через PATCH-запит без тіла – достатньо pk кімнати в URL. Функція new_booking() приймає параметри CreateBookingSerializer, викликає

create_booking() з сервісного шару та повертає створене бронювання або 404 при відсутності вільних номерів.

Функція excel_parsing() приймає файл через request.FILES, передає його у сервісну функцію exel_parsing(), що повертає список словників. Кожний словник серіалізується через PaymentSerializer у циклі, це дозволяє зберігати коректні записи та пропускати дублікати без переривання всього імпорту.

3.3 Реалізація ресторанного підмодуля

Центральна особливість ресторанного підмодуля – зв'язок ManyToMany між моделями Order та Dish. Django автоматично створює проміжну таблицю restaurant_order_dishes при виконанні міграцій, що дозволяє прив'язувати до одного замовлення довільну кількість страв.

Для оновлення складу замовлення реалізовано ендпоінт update_order_dishes(), він приймає список ідентифікаторів страв та замінює поточний набір через метод .set(). Після оновлення викликається сервісна функція check_total_pay(), яка перераховує загальну суму замовлення на основі поточних цін страв та зберігає лише змінені поля через update_fields – це мінімізує кількість SQL-запитів:

```
def check_total_pay(pk):
    order = Order.objects.filter(pk=pk).first()
    if order.is_pay:
        return
    total     = order.dishes.aggregate(Sum("price"))
    new_total = total["price__sum"] or 0
    if new_total != order.total_price:
        order.total_price = new_total
        order.save(update_fields=["total_price"])
```

Лістинг 3.5. Функція перерахунку суми замовлення

Функція create_reserve() у Restaurant/services.py реалізує пошук вільного столика через фільтрацію Django ORM. Часовий діапазон конфлікту обчислюється як [time - 2 год, time + 2 год], зайняті столики визначаються через time__range з прапорцем is_active=True. Вільний столик знаходиться методом .exclude().first(). Весь пошук виконується двома SQL-запитами незалежно від кількості столиків.

Ендпоінт new_reserve() приймає спрощений формат через CreateReserveSerializer: поля full_name, phone_number та time. Серіалізатор перевіряє, чи час резервування не менш ніж через 2 години від поточного моменту (порівняння з datetime.now() + timedelta(hours=2)), а номер телефону складається виключно з цифр довжиною 10 або 12 знаків.

Для управління доступністю страв реалізовано два окремих ендпоінти: cancel_dish() встановлює is_available=False (приховує страву), confirm_dish() повертає is_available=True. Збереження виконується через save(update_fields=["is_available"]) для мінімального SQL-запиту. Аналогічну пару утворюють cancel_table() та confirm_table() для управління доступністю столиків. Такий підхід зберігає прив'язку до існуючих замовлень та резервувань без порушення цілісності даних.

3.4 Реалізація аналітичних функцій та дашборду

Функція statistic() використовує агрегацію Django ORM через .aggregate() з функціями Avg, Sum та Count. Це формує єдиний SQL-запит з агрегатними функціями замість завантаження всіх записів у Python. Фільтрація за діапазоном дат реалізована через lookup date__range, що транслюється у SQL-умову BETWEEN. Ендпоінт stats() отримує параметри через request.query_params, валідує через StatSerializer і передає у statistic().

Функція hotel_dashboard() агрегує показники готельного блоку в реальному часі. Розподіл кімнат за статусами отримується комбінацією .values("status").annotate(Count("id")), що генерує GROUP BY SQL-запит. Результат перетворюється на словник {AV: N, NC: N, NA: N} для зручного доступу. Відсоток завантаженості розраховується як відношення активних бронювань на поточний день до загальної кількості кімнат.

Функція staff_report() використовує annotate() для збагачення запиту WorkPose агрегованими полями: staff_count (кількість співробітників) та total_salary (сумарний фонд оплати праці по посаді). Метод .values() формує SELECT лише для потрібних колонок без завантаження пов'язаних об'єктів. В

SQL це еквівалентно запиту SELECT name, COUNT(*), SUM(salary) FROM workpose JOIN personal GROUP BY name.

Консолідований ендпоінт dashboard() об'єднує результати hotel_dashboard() та restaurant_dashboard(). Загальна виручка комплексу розраховується як їх сума з явною обробкою значення None через оператор or 0, що виникає при відсутності платежів за поточний місяць.

```
@api_view(["GET"])
def dashboard(request):
    hotel      = hotel_dashboard()
    restaurant = restaurant_dashboard()
    total = (
        (hotel["finances"]["revenue_this_month"] or 0) +
        (restaurant["finances"]["revenue_this_month"] or 0)
    )
    return Response({
        "hotel":           hotel,
        "restaurant":      restaurant,
        "complex_summary": {"total_revenue_this_month": total}
    }, status=status.HTTP_200_OK)
```

Лістинг 3.6. Реалізація консолідованого дашборду

3.5 Тестування системи

Тестування розробленого модуля проводилося за методологією чорного ящика – перевірка функціональної відповідності специфікації без аналізу внутрішньої реалізації (табл. 3.2).

Основними інструментами тестування слугували Swagger UI, доступний за адресою /api/docs/, автоматично згенерований drf-spectacular та Postman [8]. Swagger UI надає інтерактивний веб-інтерфейс для формування HTTP-запитів до всіх ендпоінтів без додаткових інструментів. Postman, в свою чергу є спеціалізованим інструментом для розробки, тестування та документування API, що надає зручний графічний інтерфейс для формування HTTP-запитів довільного типу, збереження їх у колекціях і написання автоматизованих тестів на JavaScript. На відміну від Swagger UI, Postman дозволяє: зберігати змінні середовища для повторного використання значень, налаштовувати заголовки запитів та Content-Type, виконувати послідовні сценарії з передачею даних між запитами.

Таблиця 3.2

Результати тестування CRUD-операцій

Сутність	**Тест-кейс**	**Метод**	**HTTP-код**	**Очікувано**	**Статус**
Room	Отримати список	GET	200	Список об'єктів	OK
Room	Створити нову кімнату	POST	201	Новий об'єкт	OK
Room	Ціна ≤ 0	POST	400	Помилка 01.07	OK
Room	Дублікат номера кімнати	POST	400	Unique violation	OK
Booking	Створити бронювання	POST	201	Новий об'єкт	OK
Booking	date_start > date_end	POST	400	Помилка 01.09	OK
Booking	date_start < today	POST	400	Помилка 01.08	OK
Personal	ІПН – не 10 цифр	POST	400	Помилка 01.02	OK
Personal	ІПН містить літери	POST	400	Помилка 01.01	OK
Dish	Ціна = 0	POST	400	Помилка 02.06	OK
Order	Оновити склад страв	PUT	200	Перерахована сума	OK
Reserve	Час < now + 2 год	POST	400	Помилка 02.14	OK
Room	Видалити неіснуючий	DELETE	404	Помилка 01.17	OK

Тестування охоплювало три категорії перевірок:

– Функціональне тестування – перевірка коректності виконання бізнес-операцій (CRUD, пошук кімнат, резервування, імпорт реєстрів, формування аналітики).

– Тестування валідації – перевірка реакції системи на некоректні вхідні дані та граничні значення.

– Тестування цілісності даних – перевірка коректності роботи зовнішніх ключів та обмежень унікальності.

Тестування базових операцій проводилося для кожної сутності обох підмодулів.

Тестування алгоритму find_room() проводилося за чотирма сценаріями:

1. Базовий сценарій: три кімнати типу "Standard", вільний діапазон дат. Очікується повернення першої доступної кімнати та коректний розрахунок total_price. Результат: тест пройдено.

2. Сценарій часткового перекриття: кімната №101 зайнята на 10–20 квітня, запит на 15–25 квітня. Очікується повернення кімнати №102. Результат: тест пройдено – алгоритм коректно виключив №101.

3. Сценарій повного заповнення: всі кімнати типу "Standard" заброньовані. Очікується 404 з відповідним повідомленням. Результат: тест пройдено.

4. Сценарій суміжних бронювань: кімната №101 зайнята на 1–10 квітня, запит на 10–20 квітня. Суміжне бронювання не є конфліктом (умова перетину інтервалів). Очікується успішне бронювання від 10 квітня. Результат: тест пройдено.

Тестування алгоритму create_reserve() охоплювало чотири сценарії:

1. Базовий сценарій: столик №1 вільний, запит на 19:00. Очікується автоматичне призначення столика №1. Результат: тест пройдено.

2. Сценарій конфлікту: столик №1 зайнятий о 19:00, новий запит на 20:00 (в межах буфера ±2 год). Очікується призначення столика №2. Результат: тест пройдено.

3. Сценарій поза буфером: столик №1 зайнятий о 19:00, запит на 21:01. Очікується повторне призначення столика №1. Результат: тест пройдено.

4. Сценарій валідації часу: запит на час менш ніж через 2 год від поточного моменту. Очікується 400 від CreateReserveSerializer. Результат: тест пройдено – система повернула помилку 02.14.

Тестування exel_parsing() проводилося у Postman з файлом .xlsx, що містив 10 записів. З них 5 мали pay_id, що збігалися з існуючими бронюваннями, а 5 – нові записи. Результати: всі 10 записів збережено в таблицю Payment; для 5 відповідних бронювань автоматично встановлено is_pay=True. Повторний імпорт того ж файлу, з доданими чотирьма новими

записами, повернув 400 для записів з дублікатами payment_id, решта оброблена без помилок – що підтвердило коректність обмеження unique та незалежну обробку записів у циклі.

Консолідований дашборд та аналітичні ендпоінти тестувалися після заповнення бази даних тестовими записами (табл. 3.3).

Таблиця 3.3

Результати тестування аналітичних ендпоінтів

Ендпоінт	**Перевірений показник**	**Статус**	**Примітка**
/api/dashboard/	Загальна виручка = сума обох блоків	OK	
/api/dashboard/	revenue = 0 при порожній БД (не None)	OK	or 0
/api/hotel/occupancy/	Завантаженість при 0 бронюваннях = 0%	OK	
/api/hotel/occupancy/	Завантаженість при 100% бронюваннях	OK	
/api/hotel/stats/	Статистика за вказаним діапазоном дат	OK	
/api/hotel/stats/	start_date > today – помилка 400	OK	StatSerializer
/api/hotel/stats/	start_date > end_date – помилка 400	OK	StatSerializer
/api/hotel/personal_report/	Кількість персоналу по посадах	OK	GROUP BY
/api/restaurant/stats/	Фінансова статистика ресторану	OK	

3.6 Висновки до розділу 3

У третьому розділі виконано повну реалізацію та тестування модуля інформаційно-управляючої системи готельно-ресторанного комплексу.

Цілісність даних забезпечується механізмом Django ORM: foreign key constraints на рівні СУБД, поведінка SET_NULL при видаленні пов'язаних записів та обмеження unique на критичних полях виключають дублювання та каскадне видалення фінансових даних.

Модульна архітектура дозволяє додавати нові підмодулі (наприклад, SPA-центр або конференц-зал) без зміни існуючого коду – лише через новий Django-застосунок і підключення маршруту в кореневому urls.py. Принцип Open/Closed дотримується на архітектурному рівні.

Swagger UI доступний за адресою /api/docs/ і надає автоматично актуальну документацію всіх 54 ендпоінтів з описом схем запитів та відповідей. Специфікація OpenAPI 3.0 оновлюється автоматично при зміні коду без ручного втручання.

Логіка API відповідає стандартам HTTP (GET / POST / PUT / PATCH / DELETE) та повертає JSON-відповіді з коректними кодами стану (200 / 201 / 400 / 404 / 204). Це гарантує сумісність з будь-яким HTTP-клієнтом незалежно від платформи.

Побудовано структуру Django-проекту з трьома застосунками (Hotel, Restaurant, Dashboard) з уніфікованою файловою організацією. Для основних застосунків (Hotel, Restaurant) реалізовано повний стек компонентів: моделі з міграціями, серіалізатори з валідацією, view-функції з @api_view та сервісний шар. Створено ключові алгоритми системи: пошук вільного номера за перетином часових інтервалів (два SQL-запити незалежно від розміру БД), резервування столика, імпорт та зіставлення платіжних реєстрів. Реалізовано консолідований дашборд комплексу, що агрегує показники обох підмодулів в єдиній відповіді.

Тестування методом чорного ящика з використанням Swagger UI (інтерактивне тестування безпосередньо в браузері на основі автоматично згенерованої OpenAPI-специфікації) охопило 13 тест-кейсів CRUD-операцій, 4 сценарії алгоритму пошуку кімнати, 4 сценарії резервування столика, сценарій імпорту реєстру та 9 тест-кейсів аналітичних ендпоінтів – всього 31 тест-кейс. Також за допомогою Postman було перевірено коректну роботу всіх ендпоінтів, після зіставлення колекції тестів. Всі тест-кейси пройдено успішно.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи досягнуто поставленої мети – розроблено функціональний модуль інформаційно-управляючої системи готельно-ресторанного комплексу на основі архітектури REST API.

1. Здійснено системний аналіз предметної області ГРК та порівняльний аналіз існуючих програмних рішень. Визначено ГРК як складну систему з розподіленою відповідальністю, виділено сім ключових бізнес-процесів, що підлягають автоматизації: управління бронюванням, номерним фондом, замовленнями, резервуванням столиків, товарно-матеріальними запасами, персоналом та управлінською звітністю. Проведено порівняльний аналіз класів PMS (OPERA Cloud, Protel, Cloudbeds, Mews), POS (Poster, Toast, Lightspeed, Oracle Simphony), комплексних систем (Hotelogix, Maestro PMS) та CRM-рішень. Встановлено, що PMS-системи не мають повноцінної ресторанної функціональності, POS-системи – готельної інтеграції, а комплексні рішення є недоступними за вартістю для малого та середнього бізнесу. На підставі виявлених недоліків сформульовано функціональні та нефункціональні вимоги до розроблюваного модуля.

2. Спроектовано інформаційну модель та архітектуру модуля ІУС ГРК з обґрунтуванням вибору архітектурного стилю REST API і технологічного стеку. Обґрунтовано використання архітектурного стилю REST API, який забезпечує технологічну незалежність від клієнтської частини та сумісність з будь-яким HTTP-клієнтом. Як технологічний стек обрано Django та Django REST Framework, що забезпечує MTV-архітектуру, ORM для роботи з базою даних і зручні інструменти побудови API-серіалізаторів. Спроектовано модульну архітектуру у вигляді трьох незалежних Django-застосунків (Hotel, Restaurant, Dashboard) із багатошаровою організацією компонентів. Розроблено реляційні моделі даних для 16 сутностей обох підмодулів та ER-діаграми, спроектовано API-інтерфейс із 54 ендпоінтами з уніфікованим форматом помилок і двошаровою валідацією вхідних даних.

3. Розроблено та реалізовано алгоритмічне забезпечення ключових бізнес-процесів ГРК. Реалізовано три ключові алгоритми системи: алгоритм автоматичного пошуку вільного номера на основі перевірки перетину часових інтервалів (виконується двома SQL-запитами незалежно від розміру бази даних), що унеможливлює подвійне бронювання; алгоритм резервування столика для коректного управління завантаженістю ресторанного залу; механізм пакетного імпорту та зіставлення платіжних реєстрів у форматі Excel, що автоматизує підтвердження оплат без ручного втручання. Реалізовано повний CRUD для всіх моделей обох підмодулів, консолідований дашборд комплексу та аналітичні ендпоінти з агрегацією даних через Django ORM.

4. Проведено тестування розробленого API та аналіз результатів. Тестування виконано методом чорного ящика з використанням Swagger UI та Postman. Загалом охоплено 31 тест-кейс: 13 тест-кейсів CRUD-операцій і валідації, 4 сценарії алгоритму пошуку вільного номера, 4 сценарії алгоритму резервування столика, сценарій пакетного імпорту платіжного реєстру та 9 тест-кейсів аналітичних ендпоінтів. За допомогою Postman перевірено коректну роботу всіх 54 ендпоінтів. Усі тест-кейси пройдено успішно, що підтверджує відповідність реалізованої системи визначеним функціональним і нефункціональним вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Buhaieva M., Malykhin O., Hrynko Y. Business process management in the hotel and restaurant industry. Central ukrainian scientific bulletin. economic sciences. 2024. No. 12(45). P. 162–177. URL: https://doi.org/10.32515/2663-1636.2024.12(45).162-177 (date of access: 10.05.2026).

2. Django REST framework. Django REST framework. URL: https://www.django-rest-framework.org/ (date of access: 10.05.2026).

3. openpyxl – openpyxl 3.1.3 documentation. openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files – openpyxl 3.1.3 documentation. URL: https://openpyxl.readthedocs.io/ (date of access: 10.05.2026).

4. OpenAPI specification v3.0.3. Home. URL: https://spec.openapis.org/oas/v3.0.3.html (date of access: 10.05.2026).

5. Ткаченко О. А., Ткаченко О. І., Щетинін А. О. Веб-орієнтовані системи управління готелями – ефективна цифровізація готельного бізнесу. ITSynergy. 2024. № 1. С. 43–61. URL: https://doi.org/10.53920/its-2024-1-4 (дата звернення: 10.05.2026).

6. Pandas documentation – pandas 3.0.2 documentation. pandas - Python Data Analysis Library. URL: https://pandas.pydata.org/docs/ (date of access: 10.05.2026).

7. Нещадим Л., Тимчук С. Автоматизація бізнес-процесів підприємств індустрії гостинності як чинник підвищення економічної ефективності. Економіка та суспільство. 2022. № 36. URL: https://doi.org/10.32782/2524-0072/2022-36-38 (дата звернення: 10.05.2026).

8. What is API Testing? A Guide to Testing APIs | Postman. Postman: The World's Leading API Platform | Sign Up for Free. URL: https://www.postman.com/api-platform/api-testing/ (date of access: 10.05.2026).

9. Гудзова О. О. Автоматизовані системи управління готелями. Наукові видання – Львівський торговельно-економічний університет. URL: https://journals-lute.lviv.ua/journal/15_2013/22.pdf (дата звернення: 10.05.2026).

10. Лисюк Т. Інноваційні рішення в готельно-ресторанному бізнесі: технології автоматизації та персоналізації послуг. Економіка та суспільство. 2024. № 67. URL: https://doi.org/10.32782/2524-0072/2024-67-13 (дата звернення: 10.05.2026).

11. Sikora R., Postoliuk A. Comparative analysis of the effectiveness of architectural styles rest, graphql, and grpc for scalable microservice systems. Herald of Khmelnytskyi National University. Technical sciences. 2025. Vol. 355, no. 4. P. 560–568. URL: https://doi.org/10.31891/2307-5732-2025-355-79

12. Django documentation. Django Project. URL: https://docs.djangoproject.com/en/6.0/topics/db/queries/ (date of access: 10.05.2026).

13. Чайка І. М., Дністрянська Н. І. Системи автоматизації ресторанного бізнесу в україні та їхні маркетингові можливості. Індустрія туризму і гостинності в центральній та східній європі. 2025. № 12. С. 63–67. URL: https://doi.org/10.32782/tourismhospcee-12-9 (дата звернення: 10.05.2026).

14. SQLite. SQLite Home Page. URL: https://sqlite.org/ (date of access: 10.05.2026).

15. What Is OpenAPI?. Swagger Docs. URL: https://swagger.io/docs/specification/v3_0/about/ (date of access: 10.05.2026).

16. Opera Cloud PMS | Oracle Hospitality. fw_error_www. URL: https://www.oracle.com/ua/hospitality/hotel-property-management/hotel-pms-software/ (date of access: 10.05.2026).

17. Cloud-Based Hotel Property Management Software (PMS) | Planet. Planet - Connecting payments and software. URL: https://www.weareplanet.com/hotel-pms (date of access: 10.05.2026).

18. Cloudbeds. Hotel Property Management System. URL: https://www.cloudbeds.com/property-management-system/ (дата звернення: 12.04.2026).

19. Clock: Hotel management software. Clock: Hotel management software that simply works. URL: https://www.clock-software.com/ (date of access: 10.05.2026).

20. Best Hotel Property Management System 2026 | Mews PMS. Mews. URL: https://www.mews.com/en/property-management-system (date of access: 10.05.2026).

21. Poster POS – програма автоматизації закладів громадського харчування | Poster POS. Poster POS. URL: https://joinposter.com/ua (дата звернення: 10.05.2026).

22. Toast. Toast POS. URL: https://pos.toasttab.com/ (дата звернення: 12.04.2026).

23. Restaurant ePOS System. Lightspeed Commerce. URL: https://www.lightspeedhq.com/uk/pos/restaurant/ (date of access: 10.05.2026).

24. Simphony POS Systems. Oracle Simphony POS Systems. URL: https://www.oracle.com/asean/food-beverage/micros/ (date of access: 10.05.2026).

25. Tarkar M., Parker A. APIs and Restful APIs. International journal of trend in scientific research and development. 2018. Volume-2, Issue-5. P. 319–322. URL: https://doi.org/10.31142/ijtsrd15797 (date of access: 10.05.2026).

26. Comparison of Rest vs Graphql: Performance, Authentication and Scalability / K. Sadaf et al. International conference on research and development in information, communication, and computing technologies, Nagapattinam, India, 4–5 April 2025. 2025. P. 369–377. URL: https://doi.org/10.5220/0013930100004919 (date of access: 10.05.2026).

27. Serializers - Django REST framework. Django REST framework. URL: https://www.django-rest-framework.org/api-guide/serializers/ (date of access: 10.05.2026).

28. Viewsets and routers. Django REST framework. URL: https://www.django-rest-framework.org/tutorial/6-viewsets-and-routers/ (date of access: 10.05.2026).

29. Bednarz B., Miłosz M. Benchmarking the performance of Python web frameworks. Journal of Computer Sciences Institute. 2025. Vol. 36. P. 336–341. URL: https://doi.org/10.35784/jcsi.7738 (date of access: 11.05.2026).

30. Anusha Reddy Guntakandla. Modular architecture: a scalable and efficient system design approach for enterprise applications. World journal of advanced research and reviews. 2025. Vol. 26, no. 1. P. 3114–3126. URL: https://doi.org/10.30574/wjarr.2025.26.1.1340 (date of access: 11.05.2026).

31. Придибайло, О. Б., & Бондарчук, А. П. (2018). Розробка інформаційних технологій для сервісно-орієнтовного проектування інформаційних систем. Сучасний захист інформації, (1), 6-10.

32. Abramov, V., Astafieva, M., Boiko, M., Bodnenko, D., Bushma, A., Vember, V., ... & Yaskevych, V. (2021). Theoretical and practical aspects of the use of mathematical methods and information technology in education and science.

33. Машкіна, І. В., Абрамов, В. О., Носенко, Т. І., Іваніченко, Є. В., & Яскевич, В. О. (2024). Навчально-методичний посібник до виконання і захисту кваліфікаційної роботи бакалавра спеціальностей 122 Комп’ютерні науки для здобувачів вищої освіти денної форми навчання.

34. Стукаленко М. М., & Мельник І.Ю. Модуль інформаційно-управляючої системи готельно-ресторанного комплексу на основі rest API. Збірник матеріалів Всеукраїнської конференції молодих учених "Інформаційні технології”. 2026. № 13. С. 175-179. URL: https://zcit.kubg.edu.ua/index.php/journal/article/view/92 (дата звернення: 23.05.2026).