МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії

програмного забезпечення

Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

SaaS-платформа для притулку/готелю для тварин

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма «Інженерія програмного забезпечення»

Здобувачка _____ **Світлана БЕЗРЯДІНА**

«__» _____ 2026 р.

Керівник роботи

ст. викладачка _____ **Світлана БОРОЛЬОВА**

«__» _____ 2026 р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
Євген ДАВИДЕНКО
«___» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувача

Безрядіної Світлани

1. Тема кваліфікаційної роботи SaaS-платформа для притулку/готелю для тварин затверджена наказом ректора ЧНУ ім. Петра Могили № 349 від «26» грудня 2025 р.
2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.
3. Очікуваним результатом роботи є веборієнтована SaaS-платформа, яка забезпечить притулкам зручний інструмент для ведення електронного обліку підопічних, а зооготелям – автоматизований модуль для управління.
4. Перелік питань, що підлягають розробці: аналіз предметної області та огляд існуючих систем автоматизації для притулків та зооготелів; обґрунтування вибору стеку технологій та програмних засобів; проєктування мікросервісної

архітектури системи та розробка структури реляційної бази даних; реалізація серверної частини, що включає створення незалежних сервісів авторизації, обліку тварин та бронювання вольєрів; розробка клієнтської частини з інтерактивним інтерфейсом та розмежуванням прав доступу користувачів; інтеграція хмарного сховища медіафайлів і налаштування контейнеризації для розгортання платформи, а також проведення тестування розроблених модулів та оцінка ефективності готової системи.

5. Перелік графічних матеріалів: Презентація

6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «26» грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **SaaS-платформа для притулку/готелю для тварин**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КБР	25.12.2025	26.12.2025	Виконано
2.	Огляд літератури та аналіз існуючих CRM-систем для зообізнесу	15.01.2026	19.01.2026	Виконано
3.	Складання календарного плану	20.01.2026	21.01.2026	Виконано
4.	Аналіз вимог: визначення ролей користувачів (Готель/Притулок), сценаріїв бронювання та обліку тварин	22.01.2026	26.01.2026	Виконано
5.	Проєктування мікросервісної архітектури, схеми бази даних та вибір стеку технологій	02.02.2026	13.03.2026	Виконано
6.	Реалізація серверної частини: розробка сервісів авторизації, обліку тварин та алгоритмів бронювання вольєрів	16.03.2026	13.04.2026	Виконано
7.	Розробка клієнтської частини та тестування бізнес-логіки	14.04.2026	25.04.2026	Виконано
8.	Відгук керівника КБР	05.05.2026	08.05.2026	Виконано
9.	Оформлення КБР та презентації	08.05.2026	22.05.2026	Виконано
10.	Попередній захист	27.05.2026	27.05.2026	Виконано
11.	Рецензування			
12.	Завершення оформлення КБР та презентації			
13.	Захист кваліфікаційної роботи			

Здобувачка _____ **Світлана БЕЗРЯДІНА**

«__» _____ 20__ р.

Керівник роботи

ст. викладачка _____ **Світлана БОРОВЛЬОВА**

«__» _____ 20__ р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

«SaaS-платформа для притулку/готелю для тварин»

Здобувачка 409 гр.: Безрядіна Світлана

Керівник: ст. вкладачка Боровльова Світлана

Актуальність роботи зумовлена необхідністю цифровізації та оптимізації процесів обліку тварин, бронювання вольєрів та управління ресурсами у сфері зообізнесу та благодійності шляхом створення єдиного інформаційного середовища.

Метою роботи є розробка веборієнтованої інформаційної системи з використанням мікросервісної архітектури для підвищення ефективності управління діяльністю організацій сфери догляду за тваринами.

Об’єктом роботи є процеси інформаційного забезпечення та управління діяльністю організацій у сфері тимчасового утримання тварин (притулків та готелів).

Предметом роботи є методи та програмні засоби створення розподілених вебсистем класу SaaS для автоматизації обліку тварин та управління бронюваннями.

Кваліфікаційна робота складається із вступу, 4 розділів, висновків та переліку джерел посилання.

У вступі обґрунтовано актуальність обраної теми, визначено мету, основні завдання, об'єкт і предмет роботи, а також розкрито практичне значення одержаних результатів.

У першому розділі проведено комплексний аналіз предметної області, досліджено функціональні особливості притулків та готелів для тварин, проведено порівняльний огляд існуючих програмних рішень на ринку.

Другий розділ присвячено науково-методичному аналізу сучасного стану моделей та методів розробки SaaS-платформ на основі фахових публікацій. У розділі досліджено стратегії мультитенантності, обґрунтовано використання мікросервісного підходу, проведено математичне моделювання процесів валідації

даних та сформовано детальну специфікацію вимог до програмного забезпечення за міжнародними стандартами.

У третьому розділі представлено архітектурне проєктування хмарної платформи, що охоплює розробку комплексу інфраструктурних UML-діаграм, моделювання життєвого циклу запитів через шлюз безпеки API Gateway, а також розробку нормалізованої логічної структури бази даних та побудову ERD-діаграми з розподілом на спільні й ізольовані схеми тенантів..

У четвертому розділі детально описано безпосередню програмну реалізацію асинхронної серверної частини мікросервісів та високореактивного клієнтського SPA-застосунку. Представлено результати інтеграції з хмарним сервісом, наведено дані комплексного API-тестування та модульного автоматизованого тестування, а також описано процес налаштування контейнеризації Docker для розгортання платформи.

У висновках узагальнено результати виконаної роботи, підтверджено виконання поставлених завдань та визначено шляхи подальшого розширення функціоналу створеного вебзастосунку.

Кваліфікаційна робота викладена на 82 сторінках машинописного тексту, складається із вступу, 4 розділів, загальних висновків, переліку джерел посилання з 21 найменувань. Праця містить 8 таблиць та 23 рисунків.

Ключові слова: SaaS-платформа, вебзастосунок, мікросервісна архітектура, FastAPI, Vue.js, мультитенантність, облік тварин, бронювання вольєрів.

ABSTRACT

to the bachelor's qualification work

"SaaS platform for animal shelters/hotels"

Student of group 409: Bezriadina Svitlana

Supervisor: Senior Lecturer Borovlova Svitlana

The relevance of this work stems from the need to digitize and optimize processes related to animal registration, enclosure reservations, and resource management in the fields of animal care and philanthropy by creating a unified information environment.

The goal of this work is to develop a web-based information system using microservice architecture to improve the efficiency of managing the activities of organizations in the animal care sector.

The focus of this work is on the processes of information support and management of organizations involved in the temporary care of animals (shelters and boarding facilities).

The subject of this work is the methods and software tools for creating distributed SaaS-class web systems to automate animal accounting and reservation management.

The qualification work consists of an introduction, 4 chapters, conclusions, and a list of references.

In the introduction, the relevance of the chosen topic is justified, the goal, main tasks, object, and subject of the study are defined, and the practical significance of the obtained results is disclosed.

The first chapter provides a comprehensive analysis of the subject area, explores the functional characteristics of animal shelters and pet hotels, performs a comparative review of existing software solutions on the market.

The second chapter is devoted to a scientific and methodological analysis of the current state of models and methods for SaaS platform development based on specialized academic publications. The chapter investigates multi-tenancy strategies, justifies the use of a microservices approach, performs mathematical modeling of data validation processes, and establishes a detailed software requirements specification according to international standards.

The third chapter presents the architectural design of the cloud platform, which encompasses the development of a set of infrastructural UML diagrams, the modeling of the request lifecycle through the API Gateway, and the development of a normalized logical database structure, including the construction of an ERD diagram divided into shared and isolated tenant schemas.

The fourth chapter details the software implementation of the asynchronous server-side microservices and the highly reactive client-side Single Page Application. It presents the results of integration with the cloud service, provides data from comprehensive API testing and automated unit testing, and describes the Docker containerization setup process for deploying the platform.

In the conclusions, the results of the performed work are summarized, the fulfillment of the tasks set at the beginning is confirmed, and ways for further expansion of the created web application's functionality are identified.

The qualification work is presented on 82 pages of typewritten text, consists of an introduction, 4 sections, general conclusions, a list of references with 21 titles. The work contains 8 tables and 23 figures.

Keywords: SaaS platform, web application, microservices architecture, FastAPI, Vue.js, multi-tenancy, animal accounting, aviary booking.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 3
ВСТУП 4
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 6
1.1 Опис предметної області 6
1.2 Структурні та функціональні особливості 7
1.3 Аналіз готових рішень 10
Висновки до розділу 1 14
2 ДОСЛІДЖЕННЯ МЕТОДІВ ТА СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ 15
2.1 Архітектурне рішення та стек технологій 15
2.2 Методи та технології побудови мультитенантних хмарних систем 19
2.3 Методи проєктування мікросервісних систем 21
2.4 Специфікація вимог до програмного забезпечення 24
2.5 Постановка завдання на розробку 31
Висновки до розділу 2 32
3 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА МОДЕЛЮВАННЯ СИСТЕМИ 34
3.1 Обґрунтування вибору архітектурних рішень та стеку технологій 34
3.2 Сценарії використання та діаграма прецендертів 37
3.3 Діаграма діяльності 42
3.4 Діаграма класів 44
3.5 Діаграма компонентів 46
3.6 Діаграма розгортання 49
3.7 ERD Діаграма 52
Висновки до розділу 3 53
4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ 54
4.1 Розробка серверної частини платформи 54
4.2 Розробка клієнтської частини платформи 61
4.3 Інтерфейс програми 69
4.4 Тестування програмного забезпечення 79
Висновки до розділу 4 82
ВИСНОВКИ 84
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 86

ПЕРЕЛІК СКОРОЧЕНЬ

СКБД – Система керування базами даних

ACID – сукупність властивостей, що гарантують надійність виконання транзакцій у базі даних (Atomicity, Consistency, Isolation, Durability)

API – набір визначень та протоколів, що дозволяють різним програмним компонентам взаємодіяти між собою (Application Programming Interface)

ASGI – стандарт інтерфейсу для асинхронних вебсерверів та застосунків (Asynchronous Server Gateway Interface)

CORS – механізм, що дозволяє вебсторінкам запитувати ресурси з іншого домену (Cross-Origin Resource Sharing)

CRM – система управління відносинами з клієнтами (Customer Relationship Management)

HTTP / HTTPS – основні протоколи передачі даних у мережі Інтернет (HyperText Transfer Protocol / Secure)

JSON – текстовий формат обміну даними, що базується на парах «ключ-значення» (JavaScript Object Notation)

JWT – відкритий стандарт для створення токенів доступу, що використовуються для автентифікації (JSON Web Token)

MSA – архітектурний стиль, що структурує застосунок як набір невеликих автономних сервісів (Microservices Architecture)

SaaS – модель надання програмного забезпечення як послуги через хмарні сервіси (Software as a Service)

SOA – сервіс-орієнтована архітектура побудови програмного забезпечення (Service-Oriented Architecture)

SPA – односторінковий вебзастосунок, що завантажує лише одну HTML-сторінку та динамічно оновлює її (Single Page Application)

ВСТУП

У сучасному світі цифровізація бізнес-процесів стала ключовою умовою для ефективного функціонування будь-якої організації. Сфера догляду за тваринами, яка охоплює як притулки для безпритульних тварин, так і комерційні зооготелі, також потребує впровадження сучасних технологій. Водночас на сьогоднішній день галузь зіштовхується зі значним технологічним відставанням.

Більшість притулків в Україні працюють на основі волонтерства та благодійної підтримки, використовуючи для ведення обліку тварин застарілі паперові формати чи окремі електронні таблиці. Це ускладнює пошук інформації про підопічних, спричиняє втрату даних і знижує загальну продуктивність роботи персоналу. З іншого боку, комерційні зооготелі прагнуть автоматизувати бронювання, управління номерним фондом та покращити взаємодію з клієнтами, аби зберігати конкурентоспроможність у своїй ніші. Існуючі на ринку рішення зазвичай представлені або дорогими універсальними CRM-системами, які важко адаптувати під специфіку роботи з тваринами, або ж вузькоспеціалізованим програмним забезпеченням, яке охоплює лише окремі напрями діяльності.

Таким чином, створення спеціалізованої SaaS-платформи, що поєднуватиме функції для притулків та зооготелів, постає актуальним науково-практичним завданням. Використання мікросервісної архітектури в такій системі дозволить забезпечити її гнучкість, масштабованість і надійність.

Метою роботи є розробка веборієнтованої інформаційної системи з використанням мікросервісної архітектури для підвищення ефективності управління діяльністю організацій сфери догляду за тваринами.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1) провести аналіз предметної області та існуючих програмних аналогів для автоматизації роботи притулків та зооготелів;

2) визначити функціональні та нефункціональні вимоги до системи, розробити сценарії використання для різних ролей користувачів;

3) спроєктувати архітектуру системи, обґрунтувати вибір мікросервісного підходу (розподіл на сервіси авторизації, обліку тварин та бронювання);

4) розробити структуру бази даних для зберігання інформації про організації, тварин, послуги та бронювання;

5) реалізувати серверну частину з використанням технології rest арі та контейнеризації;

6) реалізувати клієнтську частину у вигляді вебзастосунку для адміністраторів та менеджерів;

7) провести тестування розробленої платформи та оцінити її ефективність.

Об’єктом роботи є процеси інформаційного забезпечення та управління діяльністю організацій у сфері тимчасового утримання тварин (притулків та готелів).

Предметом роботи є методи та програмні засоби створення розподілених вебсистем класу SaaS для автоматизації обліку тварин та управління бронюваннями.

Практичне значення одержаних результатів полягає у створенні діючого програмного продукту, який дозволяє:

– притулкам систематизувати базу даних підопічних, зберігати історію та медіа-дані у хмарі, спростити процес адміністрування;

– готелям для тварин автоматизувати процес бронювання вольєрів, уникнути накладок у графіку, вести облік додаткових послуг та формувати клієнтську базу.

Застосування мікросервісної архітектури забезпечує можливість незалежного масштабування модулів системи та спрощує її подальшу підтримку та розвиток.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У цьому розділі проводиться комплексний системний аналіз предметної області, який є фундаментальним етапом для проєктування ефективної інформаційної системи. Оскільки об’єктами автоматизації виступають заклади опіки над тваринами з різними операційними моделями (некомерційні притулки та комерційні зооготелі), виникає необхідність у детальному дослідженні їхніх бізнес-процесів, документообігу та структурних особливостей.

1.1 Опис предметної області

На нинішній стадії розвитку інформаційного суспільства та стрімкої еволюції хмарних обчислень, автоматизація бізнес-процесів перетворюється на обов'язковий чинник для результативного ведення діяльності установ будь-якого спрямування. Сучасні тенденції вказують на масовий перехід від традиційних локальних систем збереження інформації до гнучких хмарних рішень, які забезпечують безперебійний доступ до даних та їхню надійну синхронізацію. Сфера догляду за тваринами, яка охоплює як некомерційні (муніципальні та благодійні) притулки, так і платні комерційні готелі для тимчасового проживання, не є винятком і активно інтегрується у цифровий простір.

Діяльність таких закладів супроводжується необхідністю безперервно генерувати, обробляти та надійно зберігати великі масиви специфічної інформації. Документообіг організацій з опіки тварин мусить впоратися з обробкою не лише базових реєстрів підопічних, а й деталізованих медичних карток, що містять історію хвороб та специфічні дієтичні рекомендації. Окрім того, критично важливим є адміністрування розкладів бронювання місць (контроль завантаженості вольєрів), управління ресурсами закладу та ведення актуальних списків власників або потенційних опікунів. Ведення такого обліку власноруч, на паперових аркушах чи за допомогою розрізнених офісних застосунків, неминуче тягне за собою втрату інформації, неузгодженість дій команди та погіршення рівня наданих послуг.

З огляду на зазначену проблематику, у даній роботі зосереджено увагу на процесах керування та автоматизації обліку в установах, котрі займаються тимчасовим чи постійним розміщенням тварин. Аналіз предметної області охоплює увесь період знаходження тварини у закладі – починаючи з моменту її фіксації в системі, проходження карантину та первинного ветеринарного огляду, і аж до моменту вибуття з готелю або передачі притулком у нову сім'ю.

Відповідно до вищезазначеного, об’єктом дослідження кваліфікаційної роботи виступають процеси управління, інформаційного супроводу та автоматизації діяльності організацій, що забезпечують утримання та догляд за тваринами.

Вирішення завдань автоматизації вимагає застосування сучасних архітектурних підходів, які б дозволили обслуговувати безліч різних організацій в межах однієї платформи без ризику витоку чи змішування конфіденційної інформації. Тому предметом дослідження виступають методології, архітектурні патерни та програмні інструменти для створення хмарних SaaS-систем із підтримкою мультитенантності, збудованих на мікросервісній основі.

Такі системи мають гарантувати стійку ізоляцію даних користувачів на рівні баз даних та високу гнучкість незалежних програмних модулів. Застосування мікросервісного підходу дозволяє забезпечити стабільну роботу сервісів, необхідних для паралельного адміністрування як медичної документації, так і складної системи бронювання, формуючи єдиний, масштабований цифровий простір для потреб зооіндустрії.

1.2 Структурні та функціональні особливості

Для побудови ефективної інформаційної системи проведено детальний системний аналіз організаційної структури та операційної діяльності об’єктів автоматизації. Оскільки розроблювана SaaS-платформа орієнтована на два типи установ – притулки для безпритульних тварин та комерційні зооготелі – аналіз враховує як спільні риси, так і специфічні відмінності у їхніх бізнес-процесах.

Організаційна структура таких об’єктів передбачає чіткий розподіл ролей, кожна з яких висуває власні функціональні вимоги до системи.

Ключовою фігурою в ієрархії є адміністратор організації, який виконує функції глобального управління тенантом. До його ключових обов’язків належить реєстрація нових співробітників, налаштування профілів закладу, моніторинг ресурсів (кількості вільних вольєрів) та аналіз завантаженості готелю чи притулку. Функціональна частина системи для цієї ролі повинна забезпечувати надійне керування обліковими записами з чітким розмежуванням прав доступу, що є базовою нефункціональною вимогою до безпеки. З іншого боку, рядові працівники закладу потребують інструментарію для ведення поточної документації щодо кожної тварини. Це не обов'язково складні медичні діагнози, а скоріше фіксація базових показників: дати останнього годування, прийому вітамінів, проведення планових обробок від паразитів чи просто замітки про зміну поведінки. Такий «журнал догляду» має бути максимально спрощеним. Це дозволяє персоналу швидко вносити записи, не відволікаючись від основної роботи, при цьому система автоматично структурує ці дані за хронологією.

Функціональні особливості притулків вимагають від системи підтримки детальних анкет тварин, де зберігається інформація про їхній характер, особливості соціалізації та медіафайли для ідентифікації. Оскільки персонал притулку часто змінюється, критично важливо, щоб кожен новий працівник міг швидко ознайомитися з «медичною історією» тварини та зрозуміти поточний статус підопічного. В зооготелях функціональний акцент зміщується на модуль управління бронюванням. Працівник готелю повинен мати можливість швидко перевірити доступність вольєрів на конкретні дати та зафіксувати прибуття нової тварини. Головним завданням тут є уникнення помилок при плануванні, зокрема запобігання овербукінгу, коли на одне місце претендують декілька тварин. Система має автоматично валідувати дати заїзду та виїзду, що мінімізує вплив людського фактора на бізнес-процеси.

Окрім функціональних аспектів, архітектура SaaS-платформи повинна задовольняти ряд жорстких нефункціональних вимог. Першою і найважливішою є

вимога до мультитенантності та сегрегації даних. Оскільки система є багатокористувацькою, необхідно забезпечити такий рівень архітектурної ізоляції, при якому жодна організація не матиме доступу до даних іншої, навіть за умови використання спільного екземпляра сервера та бази даних. Це досягається впровадженням стратегії Schema-per-Tenant, де для кожного нового клієнта в базі даних PostgreSQL динамічно створюється окрема схема. Такий підхід не лише гарантує конфіденційність інформації, а й підвищує відмовостійкість системи, дозволяючи проводити резервне копіювання чи міграцію даних окремих тенантів без зупинки всієї платформи.

Іншою важливою нефункціональною вимогою є масштабованість та відмовостійкість, що реалізується через мікросервісну архітектуру. Розподіл системи на окремі сервіси (авторизація, база тварин, бронювання) дозволяє платформі залишатися працездатною навіть при великому навантаженні на один з модулів. Наприклад, активне завантаження фотографій тварин персоналом притулку не повинно сповільнювати роботу модуля бронювання в готелі. Крім того, система мусить мати здатність до адаптації, зважаючи на те, що персонал нерідко користується планшетними чи мобільними пристроями просто біля вольєрів, коли виконує свої щоденні обходи. Важливо, аби зовнішній вигляд системи – її інтерфейс – коректно демонструвався на екранах різного розміру, не втрачаючи при цьому комфорту при заповненні інформації, що часто відбувається у русі.

Надійність системи підкріплюється валідацією вхідних даних на рівні API та обробкою виняткових ситуацій. Це гарантує, що при веденні записів догляду або оформленні бронювання не виникне критичних збоїв, які могли б призвести до втрати інформації. Таким чином, об’єкт автоматизації потребує гнучкого хмарного рішення, яке поєднує в собі простоту щоденних записів працівників та потужний механізм управління ресурсами для адміністрації, забезпечуючи при цьому найвищий рівень захисту та швидкості доступу до даних.

1.3 Аналіз готових рішень

Для формування точних вимог до проєктованої інформаційної системи та визначення її конкурентних переваг, проведено детальний аналіз існуючих програмних рішень. Ринок програмного забезпечення націленого на сферу зоопослуг, існує певний набір рішень, однак переважна частка цих розробок є надто вузькоспрямованою: вони або цілком націлені на потреби некомерційних притулків для тварин, або ж обслуговують виключно комерційні готельні комплекси для улюбленців.

Платформа PetPoint

PetPoint (рисунок 1.1) є однією з найстаріших та найпоширеніших систем управління даними для муніципальних притулків та організацій із порятунку тварин, переважно на ринку Північної Америки. Вона орієнтована на масштабний облік популяцій безпритульних тварин [1].

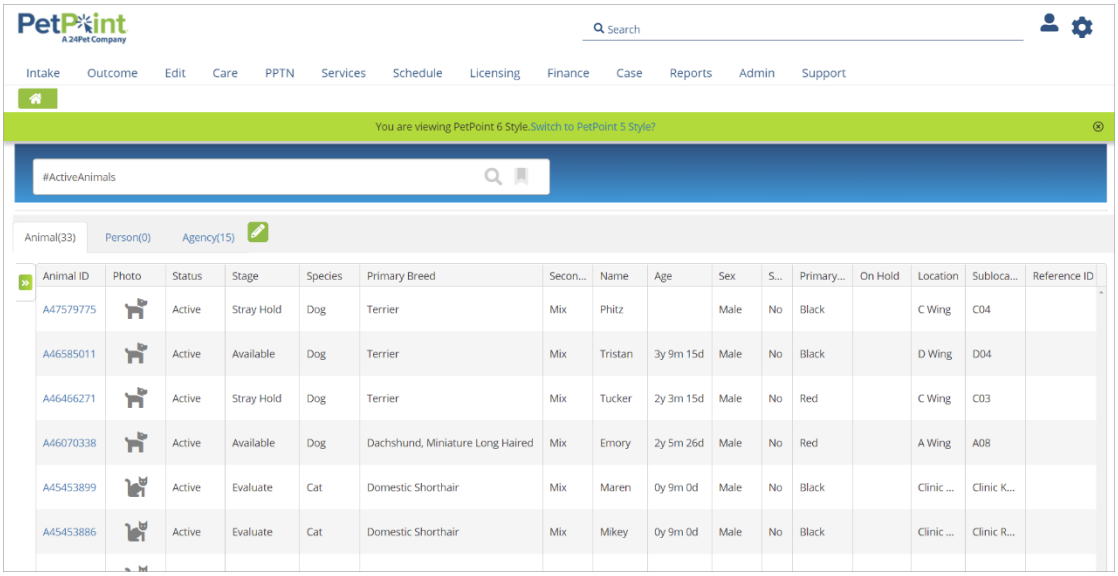

Рисунок 1.1 – Інтерфейс платформи

Архітектура та стек технологій

PetPoint побудована на базі класичної монолітної архітектури enterprise-рішень попереднього покоління (імовірно, стек Microsoft: ASP.NET та СКБД MS SQL Server). Система функціонує як хмарний застосунок, проте через монолітну структуру має складнощі з гнучким масштабуванням окремих модулів.

До переваг системи належать: глибокий аналітичний апарат із можливістю генерації складних статистичних звітів, інтеграція з міжнародними реєстрами мікрочипів та потужний модуль відстеження повного життєвого циклу тварини. Головними недоліками є застарілий користувацький інтерфейс, уповільнене оновлення функцій через монолітність архітектури, а також повна відсутність модуля бронювання для готелів та інструментів комерційного ціноутворення.

Хмарна система Shelterluv

Shelterluv (рисунок 1.2) – це сучасне SaaS-рішення, створене як альтернатива застарілим монолітам. Основний акцент системи зроблено на оптимізації роботи з волонтерами, обробці пожертв та автоматизації передачі тварин у нові сім'ї [2].

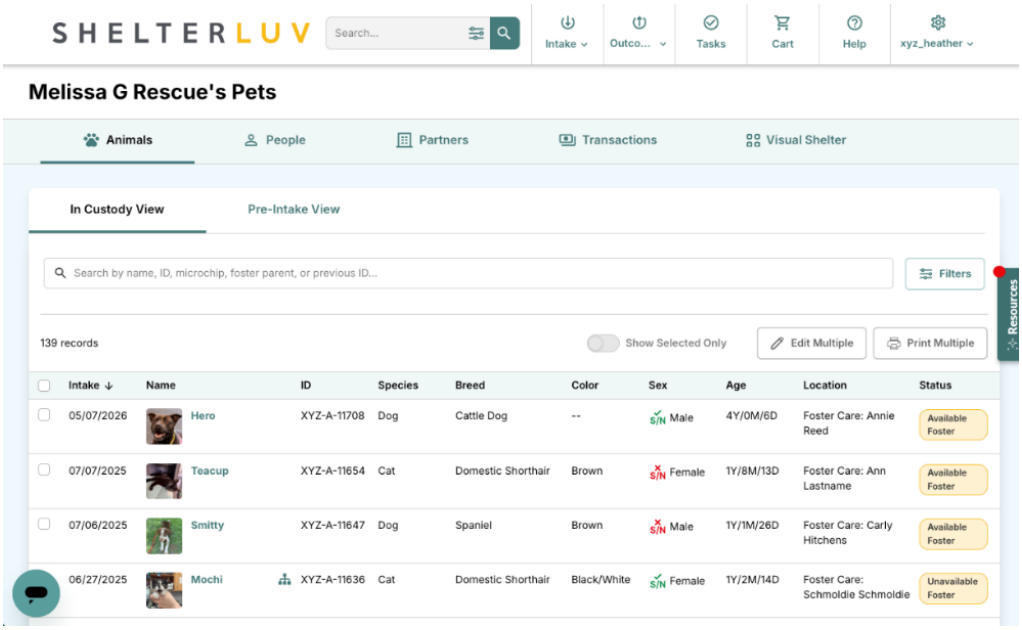

Рисунок 1.2 – Логотип хмарної системи

Архітектура та стек технологій

Застосунок реалізовано на базі модульного підходу з ймовірним використанням JavaScript-фреймворків (Node.js на бекенді та React/Vue на фронтенді). База даних використовує хмарні сервіси AWS або Google Cloud із застосуванням нереляційних сховищ для швидкого завантаження медіаконтенту та гнучких форм опитування..

До переваг використання Shelterluv належать: сучасний інтуїтивний SPA-інтерфейс, адаптований під мобільні пристрої, наявність відкритих API для

інтеграції з вебсайтами притулків та вбудовані модулі онлайн-платежів й краудфандингу. Серед недоліків виділяють: менш деталізований медичний модуль без підтримки гнучкого «щоденника догляду», відсутність інструментів управління номерним фондом вольєрів та обмежені можливості адаптації під українське законодавство.

Платформа Revelation Pets

Revelation Pets (рисунок 1.3) – це вузькоспеціалізована SaaS-платформа, призначена виключно для комерційних організацій: зооготелів, центрів денної перетримки та грумінг-салонів [3].

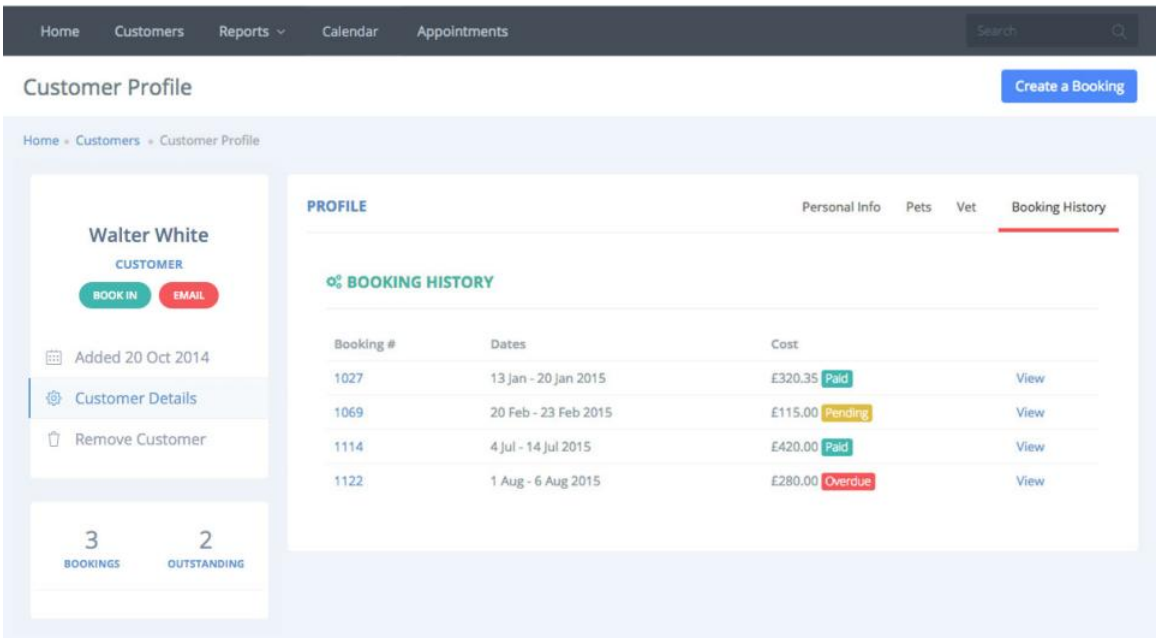

Рисунок 1.3 – Логотип платформи

Архітектура та стек технологій

Платформа базується на клієнт-серверній хмарній архітектурі. Серверна частина, імовірно, реалізована на фреймворках Ruby on Rails або Django з використанням реляційних СКБД (PostgreSQL або MySQL), що забезпечує надійний контроль транзакцій та фінансових розрахунків. Клієнтський інтерфейс активно використовує спеціалізовані бібліотеки для розширеної візуалізації графіків та календарів.

Серед ключових переваг платформи: високоякісний візуальний модуль календаря бронювань із вбудованим механізмом запобігання овербукінгу,

автоматизована система комунікації з клієнтами (розсилка нагадувань і фотозвітів) та зручний інструментарій для калькуляції вартості проживання з урахуванням додаткових послуг. Водночас суттєвими недоліками є практично повна відсутність модуля медичного обліку (фіксуються лише базові щеплення), неможливість ведення історії походження безпритульних тварин та висока вартість підписки, що стає критичним бар'єром для використання системи некомерційними та благодійними організаціями..

Для наочного відображення результатів аналізу, основні характеристики розглянутих систем та цільові показники розроблюваної платформи зведено у таблицю 1.1.

Таблиця 1.1 – Порівняльний аналіз програмних рішень у сфері догляду за тваринами

Характеристика	**PetPoint**	**Shelterluv**	**Revelation Pets**	**Проєктована система**
Цільова аудиторія	Великі притулки	Сучасні притулки	Готелі, перетримки	Притулки та готелі
Архітектурний підхід	Монолітна	Cloud-native SaaS	SaaS	Мікросервіси
Технологія зберігання (Ізоляція)	Спільна БД (логічна ізоляція)	Спільна БД (логічна ізоляція)	Спільна БД	Schema-per-Tenant
Медичний журнал (записи догляду)	Високий рівень (складний)	Базовий рівень	Мінімальний	Гнучкий журнальні записи
Модуль бронювання вольєрів	Відсутній	Відсутній	Високий рівень	Високий рівень (з валідацією)
Інтерфейс користувача	Застарілий	Сучасний (SPA)	Сучасний	Сучасний (Vue 3, SPA)

Підсумовуючи результати аналізу, можна констатувати, що на ринку відсутнє комплексне рішення, яке б однаково ефективно задовольняло потреби як некомерційних притулків (детальний медичний облік), так і комерційних готелів (керування бронюванням). Існуючі системи мають або застарілу монолітну архітектуру, або обмежуються логічною ізоляцією даних у спільній базі. Це обґрунтовує актуальність розробки власної SaaS-платформи на основі мікросервісів із застосуванням підходу Schema-per-Tenant для фізичної ізоляції даних, що забезпечить високу безпеку та гнучкість системи.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи було проведено комплексний системний аналіз предметної області, що стосується процесів управління та автоматизації діяльності організацій з опіки тварин. Дослідження специфіки бізнес-процесів показало, що сучасні притулки та комерційні зооготелі стикаються зі схожими проблемами обробки масивів даних, проте мають різні функціональні пріоритети. Для притулків критично важливим є гнучкий облік стану тварин у вигляді щоденників догляду, тоді як для готелів першочерговим є надійний механізм управління бронюванням вольєрів із запобіганням конфліктам дат.

Огляд та порівняльний аналіз сучасних програмних рішень, таких як PetPoint, Shelterluv та Revelation Pets, дозволив виявити суттєві прогалини на ринку спеціалізованого програмного забезпечення. Встановлено, що більшість існуючих систем є або занадто вузькоспрямованими, або базуються на застарілих монолітних архітектурах. Крім того, виявлено проблему в підходах до зберігання інформації: аналоги здебільшого використовують логічну ізоляцію даних у спільній базі, що створює потенційні ризики порушення конфіденційності для організацій-клієнтів.

2 ДОСЛІДЖЕННЯ МЕТОДІВ ТА СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

У даному розділі проводиться аналіз статей та дослідження сучасних методів та технологій, необхідних для проєктування надійної та масштабованої SaaS-платформи, призначеної для автоматизації діяльності притулків та зооготелів. На основі аналізу предметної області, здійсненого в попередньому розділі, виникає необхідність детального обґрунтування вибору архітектурних рішень для побудови хмарного середовища.

2.1 Архітектурне рішення та стек технологій

Після проведення системного аналізу предметної області та виявлення недоліків існуючих рішень виникає об'єктивна необхідність у формуванні чітких архітектурних вимог до майбутньої платформи. Розробка програмного забезпечення для зооіндустрії вимагає врахування економічних, технічних та безпекових факторів, що безпосередньо впливають на вибір загальної архітектури системи та стеку технологій.

Backend-частина

Для реалізації серверної логіки проводився аналіз популярних фреймворків: Django, Flask та FastAPI. Django є надлишковим для мікросервісної архітектури, а Flask не має вбудованої підтримки асинхронності на рівні ядра. Тому вибір було зупинено на FastAPI [4]. Він базується на стандартах ASGI, що дозволяє обробляти асинхронні запити. Це критично для мікросервісів, які очікують відповіді від бази даних. Вбудована підтримка анотацій типів (Pydantic) забезпечує автоматичну валідацію вхідних даних, а генерація документації Swagger UI [5] значно полегшує тестування ендпоінтів.

Frontend-частина

Для створення користувацького інтерфейсу за принципом SPA розглядалися бібліотека React та фреймворк Vue 3 [6]. Оскільки React вимагає підключення великої кількості сторонніх залежностей для маршрутизації та управління станом, перевагу було віддано Vue 3. Використання Composition API дозволяє створювати

чистий, повторно використовуваний код, що необхідно при розробці складних форм медичних карток. Екосистема Vue, зокрема інструмент збірки Vite та менеджер стану Pinia [7], дозволяє ефективно синхронізувати дані між модулями без перезавантаження сторінки.

Система управління базами даних

Для зберігання інформації порівнювалися реляційні (MySQL, PostgreSQL) та нереляційні (MongoDB) СКБД. На відміну від NoSQL рішень, реляційні бази забезпечують строгу цілісність даних, що є життєво необхідним для модуля бронювання (запобігання перетину дат). PostgreSQL було обрано як базову технологію через її нативну підтримку механізму схем, що є фундаментом для стратегії Schema-per-Tenant [8]. Також можливість роботи з типом даних JSONB дозволяє гнучко зберігати специфічні характеристики тварин різних притулків у межах жорсткої реляційної структури.

Інфраструктура та допоміжні сервіси: Docker та Cloudinary

Для забезпечення портативності мікросервісів розглядалося використання класичних віртуальних машин та контейнеризації. Вибір зроблено на користь технології Docker [9], оскільки вона ізолює конфлікти залежностей та значно пришвидшує розгортання. Для роботи з медіафайлами (фотографії тварин) локальне зберігання на сервері було відкинуто через швидке вичерпання дискового простору. Альтернативним і більш ефективним рішенням обрано інтеграцію з хмарним CDN-сервісом Cloudinary [10], що дозволяє розвантажити сервер застосунку та пришвидшити доставку зображень кінцевому користувачу.

Модель розгортання SaaS

Організації з догляду за тваринами, незалежно від їхнього комерційного статусу, переважно є представниками малого бізнесу або неприбуткового сектору. Це означає, що вони не мають власних IT-відділів, серверного обладнання та бюджетів на складне впровадження програмних продуктів (On-Premise рішень). Тому єдиною економічно доцільною моделлю розповсюдження розроблюваної системи є Software as a Service (SaaS). Хмарна модель розгортання знімає з кінцевого користувача обов'язки щодо підтримки інфраструктури та налаштування

резервного копіювання. Такий підхід знижує поріг входження для нових клієнтів і дозволяє розробникам централізовано впроваджувати оновлення [11].

Вибір архітектурного патерну: мікросервіси проти моноліту

Традиційно вебзастосунки початкового рівня створюються за принципом монолітної архітектури, де весь функціонал (база даних, бізнес-логіка, інтерфейс) тісно пов'язаний у єдиному виконуваному файлі чи середовищі. Незважаючи на простоту розробки на початкових етапах, моноліт швидко стає некерованим при збільшенні кількості користувачів та розширенні функціоналу. Зважаючи на те, що проєктована платформа оброблятиме паралельні процеси різної інтенсивності, обрано мікросервісний архітектурний підхід.

Розподіл системи на незалежні мікросервіси дозволяє досягти високої відмовостійкості. Якщо з технічних причин сервіс бронювання тимчасово вийде з ладу, працівники притулку все одно зможуть додавати записи до медичних карток тварин, оскільки сервіс обліку функціонує автономно. Крім того, мікросервісна архітектура дозволяє масштабувати лише ті вузли, які зазнають найбільшого навантаження, що оптимізує витрати на хмарний хостинг [12].

Стратегія мультитенантності

Ключовою проблемою будь-якої багатокористувацької SaaS-системи є забезпечення надійної ізоляції даних клієнтів. Існує три основні підходи до вирішення цього завдання: окрема база даних для кожного клієнта, єдина база з логічним розділенням, через ідентифікатор тенанта в кожній таблиці та використання окремих схем бази даних.

Використання окремих баз даних є занадто дорогим та складним в адмініструванні при великій кількості дрібних клієнтів. Логічне розділення в єдиній таблиці є небезпечним, оскільки помилка в коді може призвести до витоку конфіденційної інформації одного притулку іншому. Тому в даній роботі обрано компромісний та найбільш ефективний підхід – Schema-per-Tenant на базі СКБД PostgreSQL.

Ця стратегія передбачає, що всі клієнти фізично знаходяться на одному сервері бази даних, але для кожної нової організації під час реєстрації динамічно

створюється власна, повністю ізольована схема (набір таблиць). Таким чином, запити до бази даних виконуються виключно в контексті поточної схеми авторизованого користувача. Це гарантує абсолютну конфіденційність даних і дозволяє легко робити бекапи чи мігрувати окремих клієнтів без впливу на сусідні організації.

Для систематизації та наочного подання результатів аналізу, порівняння розглянутих альтернатив та остаточно обраного стеку технологій і архітектурних патернів сформовано зведену таблицю 1.2.

Таблиця 1.2 – Зведена характеристика вибору стеку технологій та архітектури

Категорія	Розглянуті альтернативи	Обране рішення	Обґрунтування вибору
Серверна частина (Backend)	Django, Flask, FastAPI	FastAPI	Нативна підтримка асинхронності (ASGI), вбудована валідація даних (Pydantic), автогенерація документації (Swagger UI), ідеальність для мікросервісів.
Клієнтська частина (Frontend)	React, Vue 3	Vue 3	Наявність Composition API для написання чистого коду, менша залежність від сторонніх бібліотек, нативна інтеграція з Vite та Pinia для управління станом.
Система управління базами даних (СКБД)	MongoDB, MySQL, PostgreSQL	PostgreSQL	Нативна підтримка логічних схем для мультитенантності, строга реляційна цілісність даних (ACID), можливість гнучкої роботи з типом JSONB.
Інфраструктура та розгортання	Віртуальні машини, Docker	Docker	Ізоляція конфліктів залежностей, висока портативність мікросервісів, значне пришвидшення та стандартизація процесів розгортання.
Зберігання медіафайлів	Локальний диск сервера, Cloudinary	Cloudinary (CDN)	Запобігання швидкому вичерпанню дискового простору бекенд-сервера, оптимізація та пришвидшення доставки зображень клієнтам.
Архітектурний підхід	Моноліт, Мікросервіси	Мікросервісна архітектура	Висока відмовостійкість (ізоляція збоїв окремих модулів), можливість незалежного точкового масштабування високонавантажених вузлів.
Модель розповсюдження	On-Premise, SaaS	SaaS (Хмара)	Зняття з клієнтів (притулків/готелів) тягаря адміністрування інфраструктури, централізоване оновлення, економічна доцільність.
Стратегія мультитенантності	Окрема БД, Спільна БД (за ID), Schema-per-Tenant	Schema-per-Tenant	Оптимальний баланс: гарантована конфіденційність та ізоляція даних клієнтів без надмірних витрат на адміністрування сотень окремих БД.

2.2 Методи та технології побудови мультитенантних хмарних систем

Розвиток концепції хмарних обчислень за останні п’ять років зумовив перехід від традиційних методів розгортання програмного забезпечення до моделі SaaS. У наукових публікаціях фахових видань, зокрема в журналах «Інженерія програмного забезпечення» та «Комп’ютерно-інтегровані технології», сучасна архітектура SaaS розглядається як складна багатокористувацька система, ключовим викликом якої є забезпечення мультитенантності. Мультитенантність визначається як архітектурний принцип, за якого один екземпляр програмного забезпечення обслуговує декількох незалежних клієнтів, гарантуючи при цьому повну ізоляцію їхніх даних та конфігурацій [13].

Аналіз наукових праць, таких як дослідження архітектурних патернів ізоляції у роботах, дозволяє класифікувати методи побудови мультитенантних систем за рівнем спільного використання ресурсів бази даних [14]. Виділяють три основні моделі:

1) Модель із спільними таблицями (Shared Database, Shared Schema). Дані всіх тенантів зберігаються в єдиному наборі таблиць, а розмежування здійснюється за допомогою додаткового поля-ідентифікатора. У фаховій літературі зазначається, що цей метод є найбільш економічно вигідним з точки зору використання дискового простору, проте він несе значні ризики безпеки. Помилка у логіці запитів може призвести до несанкціонованого доступу до даних іншого клієнта.

2) Модель із окремими базами даних (Database-per-Tenant). Кожен клієнт отримує власний екземпляр бази даних. Наукові дослідження підтверджують, що це забезпечує найвищий рівень фізичної ізоляції, проте створює надмірні витрати на адміністрування та споживання пам’яті сервера при масштабуванні до сотень і тисяч клієнтів.

3) Модель із окремими схемами (Shared Database, Separate Schema). Цей підхід, відомий як Schema-per-Tenant, став предметом активного вивчення у 2021–2024 роках. Він передбачає використання однієї бази даних, у якій для кожного тенанта динамічно створюється окрема логічна схема. Це дозволяє фізично

розділити таблиці користувачів на рівні СКБД, зберігаючи переваги централізованого управління сервером.

Для глибшого аналізу проведено порівняння цих методів за ключовими інженерними метриками (таблиця 2.1).

Таблиця 2.1 – Порівняльний аналіз моделей ізоляції даних у мультитенантних системах

Критерій порівняння	**Shared Schema**	**Database-per-Tenant**	**Schema-per-Tenant**
Ступінь ізоляції даних	Логічна (низька)	Фізична (висока)	Логіко-фізична (середня/висока)
Складність масштабування	Низька	Висока	Середня
Ефективність використання ресурсів	Максимальна	Мінімальна	Оптимальна
Складність оновлення схем БД	Легка	Дуже складна	Автоматизована (через міграції)
Безпека (Data Leakage risk)	Висока	Майже відсутня	Низька

Як зазначають дослідники у статті про методи захисту даних у хмарних СКБД, вибір підходу Schema-per-Tenant для систем, що оперують медичними або персональними даними тварин, є найбільш обґрунтованим. Це дозволяє реалізувати механізм «перемикання контексту» на рівні запитів до бази даних, де застосунок автоматично встановлює активну схему, search_path у PostgreSQL, залежно від ідентифікатора авторизованого користувача.

Окрім класичного підходу Schema-per-Tenant, у наукових працях останніх років активно розглядається метод Row Level Security як альтернатива логічному розділенню даних. RLS дозволяє визначати політики доступу на рівні окремих рядків таблиці, що теоретично спрощує архітектуру бази даних, оскільки всі тенанти використовують одну схему. Однак, порівняльний аналіз продуктивності

показує, що при зростанні обсягу даних понад 1 млн записів, накладні витрати на перевірку політик для кожного рядка стають критичними.

Вибір моделі окремих схем для даного проєкту обґрунтовується також зручністю адміністрування. Математично модель ізоляції на рівні схем можна представити як відображення множини тенантів $T = \{t_1, t_2, ..., t_n\}$ на множину ізольованих просторів імен $S = \{s_1, s_2, ..., s_n\}$ у межах однієї СКБД, де для будь-якого запиту q, ініційованого тенантом t_i, область видимості даних обмежена виключно схемою s_i. Це забезпечує детерміновану безпеку: дані тенанта t_j є фізично недосяжними для запиту від t_i, оскільки вони знаходяться поза активним шляхом пошуку.

Важливим аспектом сучасних досліджень є також питання Dynamic Provisioning – здатності системи автоматично розгортати інфраструктуру для нового клієнта. У публікаціях за 2023 рік наголошується, що використання Docker-контейнерів у поєднанні з мультитенантними БД дозволяє досягти концепції "Infrastructure as Code", що мінімізує втручання людини в процес реєстрації нового притулку чи готелю.

Крім того, аналіз методів автентифікації в розподілених хмарних системах показує поступову відмову від сесійної моделі на користь бездержавних протоколів, зокрема JWT [15]. Наукове обґрунтування цього вибору полягає в тому, що JWT дозволяє передавати ідентифікатор тенанта безпосередньо у корисному навантаженні токена, що значно прискорює маршрутизацію запитів у мікросервісному середовищі без додаткових звернень до центральної бази сесій. Таким чином, на основі аналізу фахових джерел можна зробити висновок, що поєднання мікросервісної архітектури з мультитенантним підходом Schema-per-Tenant у середовищі PostgreSQL є найбільш актуальним та технічно доцільним методом для створення масштабованих SaaS-платформ у зооіндустрії.

2.3 Методи проєктування мікросервісних систем

У сучасній інженерії програмного забезпечення проєктування масштабованих хмарних систем тісно пов’язане з вибором архітектурного стилю.

Згідно з дослідженнями у сфері SOA, традиційний монолітний підхід поступається MSA, у контексті систем із високим рівнем динамічного навантаження. У наукових публікаціях за 2021–2024 роки зазначається, що використання MSA дозволяє досягти незалежного розгортання компонентів та ізоляції відмов [16]. Для розроблюваної SaaS-системи це означає, що модулі авторизації, обліку тварин та управління простором функціонують як автономні вузли. Загальну схему організації такої архітектури та її порівняння з традиційними підходами наведено на рисунку 2.1.

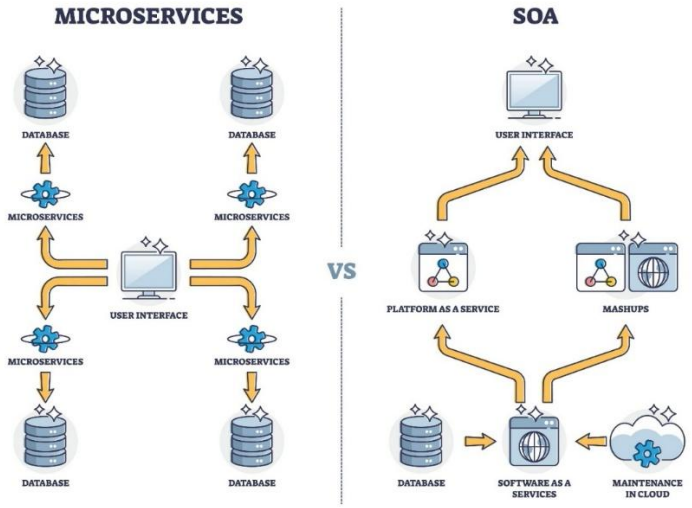

Рисунок 2.1 – Порівняння MSA та SOA

Ключовим аспектом проєктування мікросервісів є організація взаємодії між ними. Науковий аналіз патернів розподілених систем вказує на доцільність використання архітектурного шаблону «API Gateway». Цей патерн виступає єдиною точкою входу для клієнтських запитів, забезпечуючи маршрутизацію, балансування навантаження та агрегацію даних. Такий підхід суттєво зменшує кількість мережевих запитів від клієнтського застосунку, що є критично важливим для забезпечення високої продуктивності платформи на мобільних пристроях працівників притулків [17].

У контексті проєктування мікросервісів для SaaS-платформ особливої уваги заслуговує метод контейнеризації як спосіб уніфікації середовища виконання.

Дослідження методів оптимізації хмарної інфраструктури вказують на те, що використання Docker-контейнерів дозволяє реалізувати патерн Sidecar. Цей патерн передбачає винесення допоміжних функцій (логування, моніторинг, проксіювання запитів) в окремий контейнер, що працює в парі з основним сервісом. Це дозволяє розробнику зосередитися виключно на бізнес-логіці сервісів обліку тварин або бронювання.

Також аналіз сучасного інструментарію показує, що для забезпечення відмовостійкості доцільно використовувати стратегію Health Checks, інтегровану в оркестрацію контейнерів. Це дозволяє автоматично перезапускати мікросервіс у разі виявлення критичного збою, що підвищує загальну доступність системи до рівня 99.9%. З наукової точки зору, такий підхід мінімізує "середній час до відновлення" (MTTR), що є ключовою метрикою якості в інженерії програмного забезпечення.

Вибір інструментальних засобів для реалізації серверної частини також базується на сучасних метриках продуктивності. У фаховій літературі часто проводиться порівняльний аналіз сучасних вебфреймворків [18]. Відповідно до бенчмарків, опублікованих у дослідженнях архітектури вебзастосунків, фреймворк FastAPI мовою Python демонструє пропускну здатність, співставну з рішеннями на базі Node.js та Go. Висока продуктивність FastAPI досягається завдяки використанню стандарту ASGI та циклу подій бібліотеки uvicorn [19].

Наукове обґрунтування використання асинхронного програмування полягає в тому, що більшість операцій у SaaS-системі є I/O-bound, зокрема це запити до бази даних PostgreSQL та звернення до хмарного сховища медіафайлів. Асинхронний підхід дозволяє серверу не блокувати основний потік виконання під час очікування відповіді від бази даних, що радикально підвищує кількість одночасно оброблюваних з'єднань. Крім того, інтеграція бібліотеки Pydantic у FastAPI реалізує концепцію строгої типізації даних під час виконання, що, як доводять дослідження, знижує кількість критичних помилок на етапі експлуатації ПЗ на 40%. Це особливо важливо для модуля бронювання, де валідація часових інтервалів має бути абсолютно точною для уникнення овербукінгу [20].

Щодо клієнтської частини, сучасна інженерія інтерфейсів спирається на парадигму SPA. Дослідження методів рендерингу показують, що перенесення логіки формування інтерфейсу на бік клієнта значно знижує навантаження на сервер. Для забезпечення високої реактивності інтерфейсу було обрано фреймворк Vue 3.

Інновацією Vue 3, яка активно обговорюється в науковій спільноті, є використання Composition API [21]. Цей метод дозволяє інкапсулювати реактивний стан та бізнес-логіку інтерфейсу в окремі функції, що вирішує проблему фрагментації коду, характерну для попередніх підходів. Використання реактивної парадигми забезпечує миттєве оновлення статусу вольєрів чи медичних карток на екрані користувача без необхідності повного перезавантаження сторінки застосунку.

Отже, результати дослідження інструментарію підтверджують, що синергія патерну API Gateway, асинхронного бекенду на базі FastAPI та реактивного фронтенду на Vue 3 утворює науково обґрунтований стек технологій. Він повністю відповідає вимогам масштабованості, швидкодії та надійності, які висуваються до сучасних мультитенантних SaaS-систем.

2.4 Специфікація вимог до програмного забезпечення

1. Призначення та межі проєкту

1.1 Призначення системи

SaaS-платформа призначена для надання закладам опіки над тваринами (притулкам та комерційним зооготелям) інструментарію для комплексної автоматизації їхньої діяльності. Система забезпечує управління профілями тварин, оперативне ведення щоденних записів догляду, управління фондом вольєрів, а також здійснення та контроль бронювань із запобіганням овербукінгу.

1.2 Погодження, ухвалені в програмній документації

– специфікація розроблена на основі вимог до інформаційних систем у сфері послуг та стандарту архітектури мультитенантних застосунків;

– ухвалено використання сучасних вебтехнологій (SPA, мікросервіси) для забезпечення кросплатформної сумісності та швидкодії;

– ухвалено термін «тенант» для позначення ізольованого робочого простору окремої організації-клієнта.

1.3 Межі проєкту

– проєкт охоплює розробку клієнтської частини (вебзастосунку) та серверної інфраструктури (API) для обробки даних організацій;

– реалізація ізоляції даних здійснюється на рівні бази даних (Schema-per-Tenant);

– не охоплює апаратне забезпечення чи розробку нативних мобільних застосунків (iOS, Android), але передбачає адаптивний дизайн для мобільних пристроїв браузера;

– не включає розробку модулів бухгалтерського обліку підприємства та розрахунку заробітної плати.

2. Загальний опис

2.1 Сфера застосування

Вебзастосунок використовується для автоматизації процесів у некомерційних притулках для тварин, центрах перетримки, ветеринарних реабілітаційних центрах та комерційних зооготелях.

2.2 Характеристики користувачів

– адміністратори закладів: персонал, який має доступ до панелі управління тенантом для налаштування системи, додавання вольєрів та управління співробітниками;

– працівники з догляду (персонал): користувачі, які використовують застосунок для щоденної роботи (додавання профілів тварин, внесення записів про стан здоров'я, створення бронювань);

– рівень підготовки: користувачі мають базові навички роботи з вебзастосунками, система повинна мати інтуїтивно зрозумілий інтерфейс.

2.3 Загальна структура і склад системи

– клієнтська частина – вебінтерфейс, побудований на основі Vue 3 з адаптивним дизайном;

– серверна частина – API-сервер мікросервісної архітектури, побудований на мові Python із базою даних PostgreSQL;

– інтеграція – підключення до хмарного сховища Cloudinary для збереження медіафайлів тварин;

– адмінпанель – модуль для адміністраторів організації із функціями управління налаштуваннями закладу.

2.4 Загальні обмеження

– застосунок працює лише за наявності стабільного інтернет-з'єднання;

– обмеження на кількість одночасних користувачів залежить від серверної інфраструктури хмарного провайдера.

– використання безкоштовного рівня хмарних сервісів обмежує обсяг збережених фотографій до 25 ГБ загалом для всіх тенантів;

– залежність від працездатності сторонніх сервісів: у разі їх недоступності система повинна відображати тимчасові заглушки замість реальних фото.

3. Функції системи

3.1 Авторизація та аутентифікація

3.1.1 Опис функції: Дозволяє користувачам входити в систему та ідентифікувати свій робочий простір.

3.1.2 Вхідна і вихідна інформація:

– вхідна – логін (email), пароль;

– вихідна – токен доступу, інформація про користувача (роль, ID тенанта).

3.1.3 Функціональні вимоги:

– шифрування паролів за допомогою алгоритмів хешування (bcrypt);

– автоматичне перемикання контексту бази даних (search_path) на основі токена.

3.2 Управління профілями тварин та записами догляду

3.2.1 Опис функції: Користувач може створювати картки тварин, завантажувати фотографії та вести хронологічний журнал стану.

3.2.2 Вхідна і вихідна інформація:

– вхідна – дані тварини (вид, порода, вік), текст нотатки, медіафайли;

– вихідна – структурований профіль тварини, історія записів.

3.2.3 Функціональні вимоги:

– фільтрація профілів за статусом;

– збереження дати та автора кожного медичного чи поведінкового запису.

3.2.4 Правила обробки даних тварин

– система повинна вимагати обов'язкове заповнення полів: «Кличка», «Вид», «Стать» та «Дата надходження»;

– кожен профіль повинен мати унікальний ідентифікатор, що виключає конфлікти при агрегації даних у мікросервісах;

– завантаження медіафайлів обмежене розміром 5 МБ на один файл та форматами .jpg, .jpeg, .png для забезпечення швидкодії інтерфейсу.

3.3 Здійснення та контроль бронювань

3.3.1 Опис функції: Користувач може закріплювати тварину за певним вольєром на вказаний період.

3.3.2 Вхідна і вихідна інформація:

– вхідна – ID тварини, ID вольєра, дата заїзду, дата виїзду;

– вихідна – статус транзакції бронювання, оновлений статус вольєра.

3.3.3 Функціональні вимоги:

– алгоритмічна перевірка доступності вольєра перед виконанням бронювання;

– блокування операції при виявленні часових конфліктів.

3.3.4 Валідаційні обмеження алгоритму бронювання

– дата виїзду тварини не може бути раніше або дорівнювати даті заїзду;

– система повинна автоматично блокувати можливість бронювання вольєра, якщо хоча б один день із обраного діапазону вже зайнятий іншою твариною;

– редагування дат активного бронювання можливе лише за умови, що нові дати не створюють конфліктів із майбутніми записами.

4. Вимоги до інформаційного забезпечення

4.1 Джерела і зміст вхідної інформації

– дані про тварин та їхній стан надаються безпосередньо працівниками закладів;

– інформація про структуру вольєрів (категорії, місткість) формується адміністраторами.

4.2 Нормативно-довідкова інформація

– довідник видів та порід тварин;

– класифікатор статусів тварин та категорій номерів готелю.

4.3 Вимоги до способів організації, збереження та ведення інформації

– дані зберігаються в реляційній базі даних;

– ізоляція даних здійснюється через стратегію Schema-per-Tenant;

– регулярне резервне копіювання даних.

4.4 Структура обміну даними

– обмін інформацією між компонентами системи здійснюється у форматі JSON;

– всі відповіді сервера повинні містити стандартизовані коди статусів HTTP (200 для успіху, 400 для помилок валідації, 401 для проблем авторизації, 500 для внутрішніх збоїв);

– медіадані передаються у вигляді прямих URL-посилань на CDN Cloudinary, що дозволяє фронтенду кешувати зображення та знижувати навантаження на мережу.

5. Вимоги до технічного забезпечення

- сервери – віртуальні приватні сервери або хмарні контейнери з підтримкою масштабування;
- мережа – високошвидкісне з'єднання з підтримкою HTTPS;
- клієнтські пристрої – сучасні браузери (Chrome, Safari, Firefox) на ПК, планшетах або мобільних пристроях.

6. Вимоги до програмного забезпечення

6.1 Архітектура програмної системи

– мікросервісна архітектура з окремими модулями для авторизації, обліку тварин та бронювань;

– RESTful API для взаємодії між клієнтом і сервером.

6.2 Системне програмне забезпечення

– ОС серверів – Linux (Ubuntu);

– вебсервер / Reverse Proxy – Nginx.

6.3 Мережне програмне забезпечення

– протокол HTTPS із сертифікатом SSL/TLS.

6.4 Програмне забезпечення ведення інформаційної бази

– СКБД – PostgreSQL;

– інструменти міграції даних – Alembic.

6.5 Мова і технологія розробки

– frontend – Vue 3, Vite, Tailwind CSS;

– backend – Python 3.12, FastAPI;

– база даних – PostgreSQL;

– інструменти – Docker для контейнеризації та розгортання.

7. Вимоги до зовнішніх інтерфейсів

7.1 Інтерфейс користувача

– адаптивний дизайн, сумісний із роздільною здатністю від мобільних екранів до десктопів;

– інтуїтивно зрозумілий інтерфейс формату SPA.

7.2 Апаратний інтерфейс

– не потребує спеціалізованого обладнання, працює на стандартних ПК і мобільних пристроях.

7.3 Програмний інтерфейс

– REST API для забезпечення взаємодії фронтенду з мікросервісами бекенду.

7.4 Комунікаційний протокол

– HTTPS для всіх запитів з метою забезпечення конфіденційності даних.

8. Властивості програмного забезпечення

8.1 Доступність

- час роботи системи: 99.9%.

8.2 Супроводжуваність

- модульна структура коду для спрощення оновлень;
- автоматично згенерована документація API (через Swagger/OpenAPI).
- код бекенд-частини повинен супроводжуватися документацією за стандартом OpenAPI, доступною за адресою /docs кожного мікросервісу;
- використання системи контролю версій Git із чітким описом комітів є обов'язковим для відстеження змін у логіці мультитенантності.

8.3 Переносимість

- повна сумісність із хмарними платформами завдяки контейнерам Docker.

8.4 Продуктивність

- час відповіді на запит: не більше 300 мілісекунд при стандартному навантаженні.

8.5 Надійність

- використання ACID-транзакцій для запобігання втраті даних при бронюванні;
- автоматичне резервне копіювання бази даних.

8.6 Безпека

- захист від типових веб-вразливостей (SQL-ін'єкцій, XSS);
- строга ізоляція даних між тенантами на рівні СКБД;
- перевірка авторизації через jwt.
- захист від перебору (brute-force): впровадження механізму тимчасового блокування ip-адреси після 10 невдалих спроб входу протягом 5 хвилин;
- cors-політики – доступ до api дозволений лише з офіційного домену вебзастосунку для запобігання cross-origin запитам;
- data sanitization – всі вхідні рядкові дані повинні проходити очистку від html-тегів для запобігання xss-атакам у журналах догляду.

9. Інші вимоги

- логування дій користувачів, пов'язаних із видаленням даних;

– можливість масштабування системи шляхом додавання нових мікросервісів без зупинки роботи поточних модулів.

2.5 Постановка завдання на розробку

Беручи до уваги проведені дослідження предметної області, порівняльний аналіз чинних програмних аналогів, а також сформований комплекс функціональних і нефункциональних вимог, стає можливим сформулювати остаточну науково-практичну постановку завдання на розробку програмного забезпечення. Метою практичної реалізації є створення хмарної платформи для закладів опіки тварин, що включає виконання наступних етапів:

1) Проєктування та розгортання мультитенантної архітектури бази даних PostgreSQL. Необхідно розробити реляційну структуру даних, що підтримує модель Schema-per-Tenant. Система повинна містити спільну схему для збереження глобальних реєстрів організацій та облікових записів, а також забезпечувати динамічне створення та ізоляцію індивідуальних схем тенантів при реєстрації нових клієнтів. На рівні ядра СКБД мають бути налаштовані механізми автоматичного перемикання контексту виконання запитів на основі ідентифікації поточного користувача.

2) Розробка серверної мікросервісної інфраструктури та шлюзу безпеки. Потрібно реалізувати розподілену логіку бекенду на базі асинхронного фреймворку FastAPI, виділивши автономні сервіси для авторизації, ведення обліку тварин притулків та комерційного бронювання готелів. Критичним елементом є конструювання центрального шлюзу API Gateway, який має виконувати функції єдиної точки входу, здійснювати дешифрацію та валідацію токенів JWT, перевіряти рольові обмеження доступу персоналу та забезпечувати розумну проксі-маршрутизацію HTTP-трафіку до відповідних ізольованих Docker-контейнерів.

3) Створення доменного модуля некомерційного обліку та медичного контролю притулку. Необхідно розробити програмні ендпоінти та клієнтські форми для ведення цифрових карток підопічних тварин. Важливим завданням є

імплементація підсистеми ветеринарного супроводу для виведення хронологічних таймлайнів медичних оглядів та вакцинацій. Цей модуль повинен містити інструменти для предиктивного контролю складських припасів та інтерактивного керування щоденними нарядами на вигул та годування.

4) Розробка комерційного контуру готелю та безконфліктних алгоритмів бронювання.Завдання передбачає створення підсистеми управління номерним фондом зооготелю, прайс-листами додаткових сервісів та профілями комерційних клієнтів. Ключовою інженерною вимогою є розробка серверної логіки валідації часових інтервалів проживання. Алгоритм перевірки повинен повністю виключати можливість подвійного бронювання або перетину дат заселення однієї й тієї самої кімнати різними тваринами, гарантуючи абсолютну транзакційну цілісність фінансових та логістичних потоків готелю.

5) Конструювання високореактивного адаптивного користувацького інтерфейсу. На базі фреймворку Vue 3 та компонентної бібліотеки Ant Design Vue необхідно створити монолітний клієнтський пакет SPA. Графічний інтерфейс повинен миттєво перебудовуватися в режимі реального часу залежно від типу організації користувача, відображаючи притулок або готель та повністю приховуючи нерелевантні бізнес-інструменти, забезпечуючи безшовний UX-досвід.

Висновки до розділу 2

Узагальнюючи результати другого розділу, визначено, що успішна реалізація SaaS-платформи залежить від синергії обраного стеку технологій. Використання фреймворку FastAPI забезпечує швидку обробку асинхронних запитів бекенду, а Vue 3 дозволяє створити високореактивний інтерфейс для зручної роботи персоналу. Фундаментом збереження ізольованих даних обрано СКБД PostgreSQL, а для оптимізації медіафайлів — хмарне сховище Cloudinary.

Проведений аналіз методів та технологій побудови хмарних систем підтвердив, що для застосунків, які оперують конфіденційними даними багатьох незалежних організацій, найбільш дієвою є стратегія Schema-per-Tenant. Такий

підхід забезпечує оптимальний баланс між жорсткою ізоляцією інформації на рівні схем бази даних PostgreSQL та ефективним використанням серверних ресурсів, що повністю нівелює ризики несанкціонованого доступу до даних іншого клієнта.

Дослідження методологій проєктування мікросервісних систем дозволило формалізувати розподілений архітектурний каркас платформи. Доведено, що використання патерну API Gateway виступає критичним інфраструктурним елементом для централізованої координації та безпечної маршрутизації запитів у розподіленому Docker-середовищі. Важливим аспектом проєктування стала інтеграція бездержавного протоколу автентифікації JWT, який спрощує обмін ідентифікаційними даними між ізольованими мікросервісами без деградації загальної продуктивності системи.

Черговим результатом розділу стала розробка детальної специфікації вимог до програмного забезпечення, структурованої за міжнародними інженерними стандартами. Формалізація призначення системи, опис рольових моделей користувачів та деталізація функціональних блоків некомерційного обліку тварин і комерційного управління бронюваннями дозволили трансформувати бізнес-завдання притулків та готелів у чітку систему технічних орієнтирів, визначивши жорсткі властивості надійності, безпеки й доступності платформи.

Фінальним етапом виступило формування остаточної постановки завдання на розробку. У межах цього завдання було чітко декомпоновано інженерні кроки створення системи, типізовано вхідні й вихідні потоки даних для благодійного та комерційного контурів, а також визначено критерії оцінки успішності проєкту через автоматизоване тестування. У сукупності сформовані теоретичні дослідження, специфікація вимог та постановка завдання створюють вичерпний, науково обґрунтований інженерний базис для переходу до етапу безпосереднього архітектурного проєктування, побудови інфраструктурних UML-моделей та детального моделювання структури бази даних у наступному розділі кваліфікаційної роботи..

3 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА МОДЕЛЮВАННЯ СИСТЕМИ

У цьому розділі наведено опис процесу архітектурного проєктування та інформаційного моделювання розробленої хмарної SaaS-системи. Детально розкрито логіку побудови мікросервісної інфраструктури, змодельовано життєвий цикл обробки запитів через шлюз безпеки API Gateway за допомогою апарату UML-діаграм, а також спроєктовано нормалізовану логічну структуру реляційної бази даних із підтримкою стратегії мультитенантності на рівні ізольованих схем. Розділ демонструє перехід від теоретичної специфікації вимог до формалізованих інженерних моделей, які створюють надійний фундаментальний каркас для безпосередньої програмної реалізації та подальшого розгортання системи в наступному розділі.

3.1 Обґрунтування вибору архітектурних рішень та стеку технологій

Для забезпечення високої продуктивності, масштабованості та надійності функціонування системи було обрано сучасний технологічний стек, який повністю задовольняє інженерні вимоги до розробки багатокористувацьких хмарних SaaS-рішень. Ключовим архітектурним принципом практичної реалізації є використання мікросервісного стилю. Це дозволило декомпозувати загальну бізнес-логіку платформи на окремі, логічно ізольовані та автономні функціональні блоки, які мають можливість незалежного масштабування та оновлення без зупинки всього програмного комплексу.

Проєктна інфраструктура базується на концепції інфраструктури як коду. Усі конфігураційні файли середовища розгортання, інструкції складання сервісів та правила маршрутизації мережевого трафіку зафіксовані в програмних маніфестах, що дозволяє автоматизувати процес розгортання платформи на продуктових серверах, забезпечити ідентичність середовища розробки та продуктового середовища, а також повністю нівелювати вплив людського фактора.

Серверна частина мікросервісів розроблена на базі асинхронного фреймворку FastAPI. Його використання у проєкті дозволяє ефективно обробляти велику кількість конкурентних HTTP-запитів в однопоточному циклі подій завдяки

неблокуючим операціям введення-виведення. Валідація вхідних та вихідних структур даних на рівні програмних інтерфейсів виконується за допомогою декларативних схем бібліотеки Pydantic, що гарантує сувору типізацію та безпеку під час виконання коду. Автоматична генерація інтерактивної документації за стандартом OpenAPI спрощує процес налагодження та інтеграції клієнтської частини з серверними ендпоінтами.

Клієнтська частина системи функціонує за принципом SPA та реалізована за допомогою фреймворку Vue 3 з використанням архітектурного підходу Composition API. Менеджмент глобального стану додатка в межах браузера клієнта здійснюється через реактивне сховище Pinia. Таке рішення забезпечує динамічне оновлення елементів користувацького інтерфейсу в режимі реального часу без примусового перезавантаження вебсторінки, суттєво оптимізуючи обсяг передаваного мережевого трафіку та підвищуючи швидкість відгуку системи.

Реляційна база даних проєкту спроєктована під управлінням СКБД PostgreSQL. Реалізація архітектурної стратегії мультитенантності за типом Schema-per-Tenant забезпечується на рівні сесій підключення до бази даних. Під час обробки кожного вхідного запиту мікросервіс ідентифікує орендаря та динамічно виконує SQL-інструкцію SET search_path TO tenant_[id]. Це дозволяє використовувати один фізичний екземпляр СКБД та загальний пул підключень, гарантуючи при цьому ізоляцію даних на рівні логічних просторів імен, що відповідає високим стандартам безпеки та конфіденційності в SaaS-індустрії.

Специфікація фізичної інфраструктури та контейнеризації

Фізична топологія інфраструктури SaaS-платформи базується на системній контейнеризації Docker та оркестрації Docker Compose. Кожен мікросервіс, проксі-сервер та сервер баз даних ізольовані у власних легковагових контейнерах на рівні ядра операційної системи Linux Host, що передбачає поділ на закриті внутрішні мережі та суворе керування постійним зберіганням даних.

Для організації мережевої взаємодії всередині Docker Host створено віртуальну підмережу типу Docker Bridge. Публічний доступ із мережі Інтернет дозволений лише до контейнера шлюзу безпеки та маршрутизатора (API Gateway).

Усі інші внутрішні вузли платформи та СКБД PostgreSQL не мають відкритих назовні портів, що унеможливлює прямі зовнішні атаки. Комунікація між сервісами всередині мережі хоста відбувається за внутрішніми DNS-іменами контейнерів через закритий TCP-протокол.

Управління конфігураціями й секретами реалізовано через змінні оточення, які передаються в контейнери під час їхнього старту із захищених файлів налаштувань (.env) відповідно до методології Twelve-Factor App. Задля забезпечення стабільності та відмовостійкості інфраструктури для всіх критичних контейнерів налаштовано політику автоматичного перезапуску, яка гарантує відновлення роботи сервісів у разі внутрішніх збоїв.

Оскільки контейнери є ефемерними, для персистентності інформації СКБД PostgreSQL налаштована на роботу з іменованими томами Docker Volumes. Фізичний каталог бази даних /var/lib/postgresql/data всередині контейнера зв'язаний із постійним сховищем хост-машини. Це гарантує цілісність та збереженість медичних карток, записів про тварин та журналів бронювань під час оновлення образів чи технічних робіт.

Рівень клієнта представлений пристроєм кінцевого користувача, на якому у веббраузері виконується скомпільований пакет статичних файлів (HTML/JS/CSS) SPA Vue.js. Клієнтський застосунок взаємодіє із зовнішнім світом за двома напрямками: відправляє асинхронні HTTP-запити до основного сервера платформи та здійснює пряме завантаження важкого медіаконтенту на розподілені сховища за протоколом HTTPS..

Рівень шлюзу (API Gateway) реалізований на базі високопродуктивного вебсервера Nginx, який функціонує в режимі зворотного проксі-сервера. Nginx є єдиною публічною точкою входу системи. Він аналізує URL-шлях кожного вхідного запиту та за допомогою директиви proxy_pass перенаправляє його на відповідний внутрішній мікросервіс. Окрім маршрутизації, цей рівень забезпечує базову безпеку: здійснює фільтрацію шкідливих запитів, додає захисні HTTP-заголовки та проводить валідацію CORS-політик.

Рівень сервісів складається з набору незалежних Docker-контейнерів, у яких розгорнуто застосунки FastAPI під управлінням ASGI-серверів Uvicorn. Сервіси логічно розділені за доменними областями: Auth Service відповідає за генерацію та верифікацію бездержавних JWT-токенів; Animal Service обробляє бізнес-логіку ведення карток та медичних записів тварин; Booking Service містить алгоритми розрахунку зайнятості вольєрів та менеджменту номерного фонду. Автономність контейнерів захищає систему від каскадних збоїв.

Рівень зберігання даних представлений контейнером PostgreSQL 16. Доступ до бази даних мають виключно мікросервіси з внутрішньої мережі Docker. Завдяки динамічній зміні простору схем межах одного інстансу, СКБД ізолює транзакції різних клієнтів. Використання Docker Volumes забезпечує фізичне відокремлення файлів бази даних від життєвого циклу самого контейнера, що гарантує абсолютну відмовостійкість збереження критично важливої інформації.

Зовнішні сервіси включають хмарну інфраструктуру Cloudinary. Фронтенд-застосунок Vue.js завантажує фотографії тварин безпосередньо в Cloudinary через захищене API, минаючи бекенд-сервери. Хмара оптимізує зображення під різні розширення екранів та повертає у систему унікальний текстовий URL-рядок. Бекенд зберігає в PostgreSQL лише це посилання. Такий підхід радикально знижує навантаження на дискову підсистему та пропускну здатність мережі основного сервера платформи, делегуючи доставку контенту глобальній мережі CDN.

Візуальне відображення описаної багаторівневої інфраструктури, її фізичних вузлів та протоколів взаємодії буде детально представлено у підрозділі, присвяченому Діаграмі розгортання

3.2 Сценарії використання та діаграма прецендертів

Для високорівневого моделювання функціональних вимог, визначення меж проєктувальної системи та візуалізації взаємодії між користувачами й безпосередньо програмним комплексом було застосовано апарат Use Case інженерії. Цей підхід дозволяє зафіксувати повний перелік сервісів, які розподілена

SaaS-платформа зобов'язана надавати кожній рольовій категорії користувачів, та формалізувати політику розмежування прав доступу на основі ролей (рис.3.1).

Клієнт (Власник тварини)

Цей актор є зовнішнім користувачем платформи, який взаємодіє з системою через її публічну частину. Для забезпечення повноти процесу дистанційного обслуговування для нього визначено ключовий прецедент:

– самостійне онлайн-бронювання – надає можливість у реальному часі переглядати вільний номерний фонд готелю та надсилати заявку на розміщення улюбленця. Ця функція через зв'язок розширення <<extend>> інтегрується у загальний модуль управління замовленнями.

Працівник організації (базова роль)

Цей актор уособлює операційний персонал (доглядачів, менеджерів, штатних ветеринарів), який здійснює щоденне обслуговування закладу. Для забезпечення повноти внутрішніх бізнес-процесів за цією роллю закріплено наступний спектр прецедентів:

– управління картками твари – реєстрація нових профілів, завантаження медіаконтенту та ведення загальних облікових анкет підопічних;

– ведення медичних записів – фіксація історії оглядів, встановлених діагнозів, ветеринарних призначень та графіків профілактичної вакцинації;

– управління бронюваннями – обробка та підтвердження вхідних заявок, а також реєстрація процедур безпосереднього заселення та виселення тварин;

– перегляд статусу вольєрів – моніторинг поточної зайнятості та санітарного стану кімнат номерного фонду;

– облік складу – контроль наявності та витрат матеріальних ресурсів (кормів, медикаментів, засобів гігієни).

Адміністратор організації

Має найвищий рівень повноважень у межах конкретного простору компанії-орендаря. Окрім успадкування всіх операційних функцій звичайного працівника, адміністратор наділений ексклюзивними прецедентами для повноцінного бізнес-менеджменту:

– управління персоналом – реєстрація нових облікових записів співробітників та призначення їм внутрішньосистемних ролей;

– конфігурація вольєрів – додавання нових житлових місць, встановлення їхніх технічних параметрів, габаритних обмежень та цінових тарифів;

– управління щоденними задачами працівника – формування операційних нарядів для персоналу, розподіл завдань з догляду та контроль за їхнім виконанням;

– видалення критичних даних – прецедент із підвищеними вимогами до безпеки, що дозволяє безповоротно вилучати системні логічні записи з бази даних, унеможливлюючи випадкове руйнування інформації звичайними працівниками.

Важливим інженерним рішенням, відображеним на діаграмі для гарантування безпеки багатокористувацького середовища, є використання зв'язків включення <<include>>. Усі операційні прецеденти ролей Адміністратора та Працівника обов'язково включають базовий прецедент «Автентифікація в системі».

Рисунок 3.1 – Діаграма прецедентів

Для деталізації взаємодії акторів, структуризації бізнес-логіки та виявлення альтернативних шляхів виконання алгоритмів розроблено розширені текстові специфікації прецедентів. Далі наведено деталізацію основного операційного сценарію реєстрації нової тварини в системі, який обробляється мікросервісом обліку (табл. 3.1)

Таблиця 3.1 – Сценарій використання «Додавання нової тварини»

Атрибут прецеденту	**Опис**
Назва	Додавання профілю нової тварини
Область дії	PetCloud SaaS
Рівень	User Goal
Головний актор	Працівник або адміністратор притулку
Зацікавлені сторони	Працівник - швидка реєстрація. Притулок - отримання валідних, структурованих даних для ведення коректного обліку.
Передумови	Користувач авторизований
Гарантія успіху	Профіль тварини додано до бази даних, інформацію безпечно збережено в ізольованій схемі тенанта.
Основний сценарій	– працівник відкриває розділ «підопічні» у веб-інтерфейсі; – натискає кнопку ініціалізації форми «Додати тварину»; – заповнює обов'язкові поля (ім'я, вік, вид, опис) та прикріплює графічний файл (фотографію); – натискає кнопку збереження; – клієнтський застосунок асинхронно завантажує фото до зовнішнього хмарного сховища та отримує URL-посилання; – сформований пакет даних відправляється на єдиний шлюз (API Gateway), який маршрутизує його до відповідного мікросервісу; – система валідує дані на сервері та виконує транзакцію запису в базу даних; – нова тварина з'являється у загальному списку на екрані користувача.
Розширення	– порушення валідації – працівник залишив обов'язкове поле порожнім або ввів некоректний формат даних. Система блокує відправку запиту і підсвічує поле червоним кольором.. – помилка мережі – шлюз не може зв'язатися з мікросервісом. Система виводить повідомлення: «Помилка з'єднання з сервером. Спробуйте пізніше».
Спеціальні умови	Завантаження медіафайлів має відбуватися на стороннє хмарне сховище для розвантаження серверів платформи, а в БД зберігається лише текстове посилання.

У межах функціонального модуля зооготелю ключовим бізнес-процесом є оформлення проживання. Під час проєктування даного сценарію було ухвалено архітектурне рішення щодо впровадження механізму «пошуку за явним запитом» для оптимізації навантаження на мережевий рівень. Замість виконання миттєвої відправки запитів до бази даних під час введення кожного символу номера телефону, система ініціює запит до бекенду виключно після явного натискання кнопки користувачем.

Це рішення є критично важливим для масштабованої SaaS-платформи, де одночасна робота сотень працівників з різних організацій могла б згенерувати надмірну кількість паралельних запитів на шлюз API Gateway. Специфікацію цього процесу наведено в табл. 3.2.

Таблиця 3.2 – Сценарій використання «Оформлення бронювання вольєра»

Атрибут прецеденту	**Опис**
Назва	Оформлення бронювання вольєра
Область дії	PetCloud SaaS
Рівень	User Goal
Головний актор	Працівник готелю або адміністратор
Зацікавлені сторони	Власник тварини - гарантія резервування місця на обраний період. Адміністратор - швидке оформлення послуги та 100% уникнення накладок у графіку номерного фонду.
Передумови	Користувач авторизований. У базі даних готелю попередньо налаштовані вольєри та перелік додаткових послуг.
Гарантія успіху	Транзакцію успішно закрито: створено запис про бронювання та сформовано необхідні зв'язки з переліком обраних послуг.
Основний сценарій	– працівник відкриває модуль «Календар бронювань»; – вводить номер телефону власника; – натискає кнопку ініціалізації пошуку; – система ідентифікує клієнта і підтягує перелік його тварин із бази даних; – працівник обирає тварину та вказує бажані дати заїзду/виїзду; – система динамічно відфільтровує та показує вільні вольєри; – працівник обирає вольєр та додає необхідні супутні послуги; система автоматично розраховує фінальну вартість і зберігає дані.

Кінець таблиці 3.2

Розширення	– Клієнта не знайдено. За вказаним номером телефону записів немає. Система пропонує створити новий профіль власника та тварини. – Відсутність місць – на обрані дати немає жодного вільного вольєра відповідного розміру. Система блокує кнопку збереження та пропонує змінити дати.
Спеціальні умови	Процедура збереження бронювання має виконуватися в межах єдиної атомарної транзакції (ACID) у реляційній базі даних. Якщо під час запису додаткових послуг виникає збій, основний запис про бронювання вольєра також має бути скасовано (Rollback).

Наведені специфікації сценаріїв використання відіграють ключову роль у загальному життєвому циклі розробки програмного забезпечення. Вони слугують фундаментальною основою для проєктування екранних форм користувацького інтерфейсу, розробки маршрутизаторів API та виступають базою для формування тест-кейсів під час проведення подальшого інтеграційного та наскрізного тестування платформи.

3.3 Діаграма діяльності

Динамічний аспект функціонування розподіленої мікросервісної архітектури SaaS-платформи, який визначає логіку керування вхідними інформаційними потоками та координацію міжкомпонентної взаємодії, формалізується за допомогою діаграми діяльності (рис. 3.2). Ця модель деталізує покроковий алгоритм роботи центрального шлюзу безпеки API Gateway, реалізованого на базі зворотного проксі-сервера Nginx, який виступає єдиною захищеною точкою входу для зовнішніх клієнтських запитів від Vue 3 SPA.

Побудована схема охоплює повний життєвий цикл обробки HTTP-запиту: від його перехоплення на мережевому периметрі до безпечної маршрутизації всередині ізольованої віртуальної підмережі Docker Bridge.

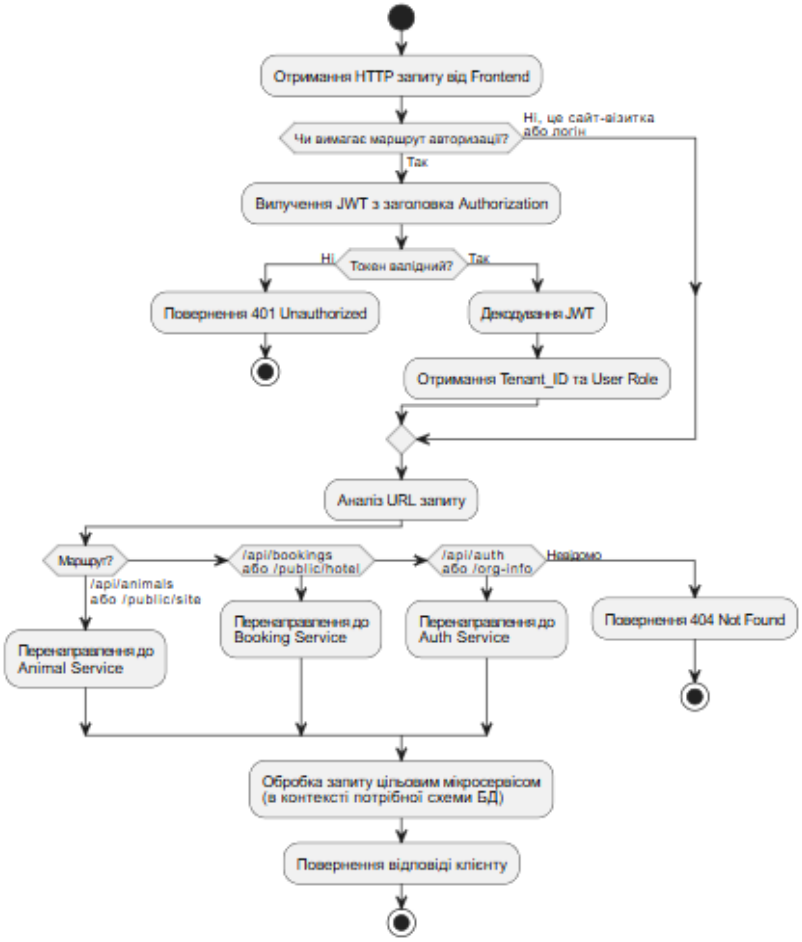

Рисунок 3.2 – Діаграма діяльності

Описаний на діаграмі процес складається з кількох послідовних етапів, що утворюють лінійні та розгалужені алгоритмічні контури:

– Ініціалізація та перехоплення запиту – шлюз індукує виконання алгоритму в момент отримання HTTP-запиту від фронтенд-частини. Першим логічним розгалуженням є перевірка типу маршруту. Система аналізує конфігураційний файл проксі-сервера, щоб визначити, чи належить запитуваний URL до категорії публічних ендпоінтів.

– Контур авторизації та безпеки – Якщо маршрут вимагає суворої перевірки прав доступу (наприклад, операції з картками тварин чи журналами бронювань), шлюз виконує процедуру вилучення бездержавного маркера безпеки JWT із заголовка Authorization. У разі відсутності токена або провалу його криптографічної верифікації (токен підроблений чи термін його дії вичерпано), діяльність миттєво переходить до фінального вузла відхилення із поверненням

клієнту HTTP-статусу 401 Unauthorized, що запобігає будь-якому навантаженню на внутрішні сервіси

– Декодування контексту тенанта – за умови успішної валідації JWT, шлюз безпеки асинхронно декодує корисне навантаження (Payload) маркера. На цьому етапі з криптографічного підпису вилучаються метадані користувача: його рольовий ідентифікатор та унікальний маркер організації (tenant_id). Цей крок є критично важливим для збереження стійкості стратегії мультитенантності, оскільки саме отриманий tenant_id далі супроводжуватиме запит для динамічного перемикання простору схем у PostgreSQL.

– Маршрутизація за URL-шляхом – після етапу безпеки API Gateway проводить аналіз URL-шляху запиту. Відповідно до таблиці маршрутизації відбувається перенаправлення (проксіювання) трафіку до цільового мікросервісу: запити з префіксом /api/animals направляються до Animal Service, /api/bookings – до Booking Service, а /api/auth – до Auth Service. За відсутності відповідного маршруту система генерує виключення з кодом помилки 404 Not Found.

Така організація процесу забезпечує високий ступінь інкапсуляції бізнес-логіки та дозволяє централізовано керувати безпекою, логуванням та масштабуванням мікросервісів, мінімізуючи при цьому навантаження на внутрішні компоненти системи.

3.4 Діаграма класів

Діаграма класів відображає концептуальну модель даних та логічну структуру взаємозв’язків між сутностями, що забезпечують роботу SaaS-платформи (рис. 3.3). Архітектурно система поділена на три функціональні домени, які корелюють з мікросервісною структурою та реалізацією мультитенантності через механізм схем PostgreSQL.

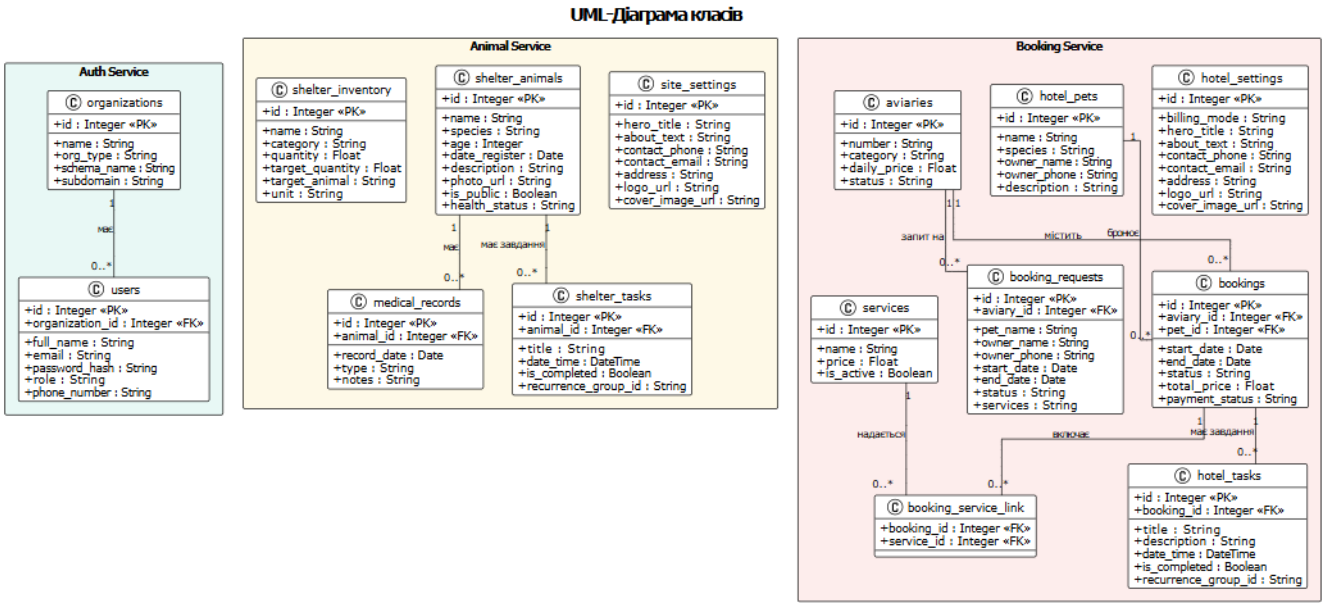

Рисунок 3.3 – Діаграма класів

У межах глобального простору авторизації (Auth Service) виділено ключову інфраструктурну сутність **organizations**, яка містить ідентифікаційні дані орендаря, включаючи специфічне поле назви логічної схеми бази даних, що керує мультитенантністю. До цього класу через суворий зв'язок відношення «один-до-багатьох» підпорядковується клас **users**, який репрезентує облікові записи персоналу. Це гарантує жорстку ідентифікаційну прив'язку кожного співробітника до його цільової організації без можливості міжтенантної міграції.

Архітектура мікросервісу обліку (Animal Service) базується на центральному транзакційному класі **shelter_animals**, який агрегує основну інформацію про підопічних. Від нього успадковують ідентифікатори два залежні класи: **medical_records**, призначений для ведення ветеринарної історії, та **shelter_tasks** для управління щоденними розкладами догляду. У межах цього ж мікросервісу функціонують два незалежні класи, що не мають прямого реляційного зачеплення з тваринами: **shelter_inventory** для моніторингу складських запасів та **site_settings** для управління контентом публічної вебсторінки конкретного притулку.

Домен комерційного бронювання (Booking Service) оперує найбільш розгалуженою транзакційною моделлю, ядром якої виступає клас **bookings**. Ця сутність консолідує інформацію про замовлення, приймаючи зовнішні ключі від незалежних класів **hotel_pets** (клієнтські профілі тварин) та **aviaries** (номерний

фонд готелю) за принципом відношення 1:N. Для забезпечення можливості додавання кількох платних послуг до одного замовлення, логічний зв'язок «багато-до-багатьох» між сутностями замовлення та каталогом послуг **services** розв'язано через проміжну асоціативну сутність **booking_service_link**. Крім того, архітектура цього домену включає клас **booking_requests** для тимчасового зберігання непідтверджених зовнішніх заявок від клієнтів та клас операційних завдань **hotel_tasks**, безпосередньо прив'язаний до конкретного бронювання.

Загалом, спроєктована даталогічна модель повністю задовольняє вимоги третьої нормальної форми, мінімізує надлишковість даних та ідеально мапується на фізичні таблиці СКБД PostgreSQL за допомогою механізмів бібліотеки SQLAlchemy, гарантуючи надійну цілісність інформації під час паралельної роботи мікросервісів.

3.5 Діаграма компонентів

Для відображення внутрішньої логіко-структурної організації програмних модулів, визначення меж їхньої автономності, специфікації інтерфейсів взаємодії та системних залежностей між розподіленими сервісами SaaS-платформи було розроблено діаграму компонентів, яку наведено на рисуінку 3.4.

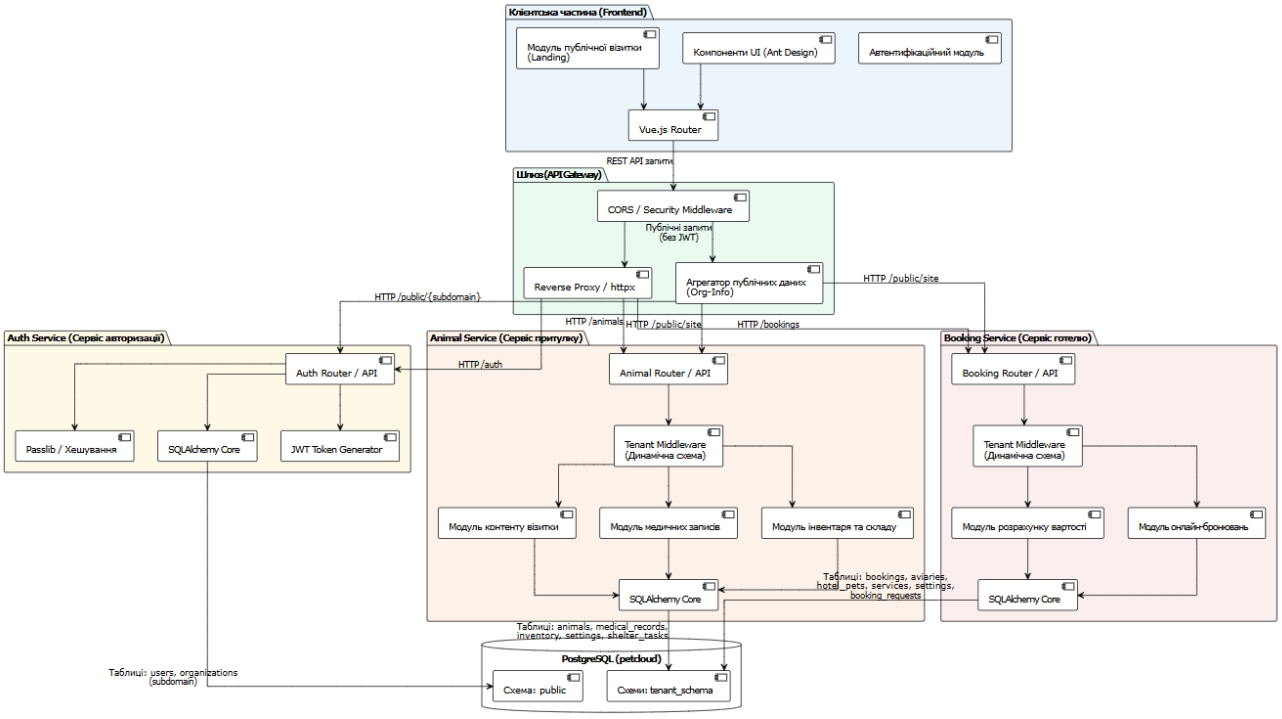

Рисунок 3.4 – Діаграма компонентів

Наведена діаграма компонентів деталізує декомпозицію архітектури проєкту на чотири взаємопов'язані, але логічно та фізично ізольовані рівні, що забезпечує гнучкість, низьку зв'язність та високу міцність програмних модулів:

Клієнтська частина

Цей архітектурний рівень відповідає за представлення даних та взаємодію з користувачем. Він розроблений за технологією SPA і містить наступні ключові субкомпоненти:

– компоненти UI – графічний інтерфейс користувача, побудований на базі бібліотеки компонентів Ant Design, що забезпечує стандартизований та адаптивний дизайн;

– модуль публічної візитки – забезпечує рендеринг відкритих сторінок організацій для зовнішніх відвідувачів, які не потребують авторизації в системі;

– автентифікаційний модуль – відповідає за управління сесіями на стороні клієнта та збереження токенів доступу;

– Vue.js Router – забезпечує маршрутизацію на стороні клієнта, керуючи переходами між екранними формами без перезавантаження сторінки. Усі зовнішні HTTP-запити відправляються до серверної частини через цей рівень.

Рівень шлюзу

Виконує роль єдиної фасадної точки доступу до системи (Reverse Proxy), що ізолює внутрішню мережу мікросервісів. Складається з:

– CORS / Security Middleware – проміжне програмне забезпечення для валідації політик спільного використання ресурсів та базової фільтрації шкідливого трафіку;

– Reverse Proxy – компонент асинхронного проксіювання, який аналізує URL-префікси та перенаправляє HTTP-запити до відповідних внутрішніх мікросервісів;

– агрегатор публічних даних (Org-Info) – спеціалізований компонент, який зчитує піддомен вхідного запиту і без перевірки токена дозволяє отримувати відкриту інформацію про заклади.

Рівень мікросервісів додатка

Складається з набору автономних Docker-контейнерів, у яких розгорнуто застосунки FastAPI під управлінням високопродуктивних ASGI-серверів Uvicorn. Кожен сервіс інкапсулює власну ізольовану бізнес-логіку та оперує наступними внутрішніми модулями й компонентами:

– Auth Service (Сервіс авторизації) – даний мікросервіс відповідає за реєстрацію нових тенантів, автентифікацію користувачів та контроль сесій, що реалізується через компонент Auth Router / API. Внутрішня логіка сервісу містить JWT Token Generator для кодування й випуску бездержавних маркерів доступу та бібліотеку Passlib для безпечного хешування паролів. Взаємодія з базою даних здійснюється за допомогою інструменту SQLAlchemy Core, який виконує операції запису та зчитування виключно в межах спільної глобальної схеми public для таблиць користувачів та організацій;

– Animal Service (Сервіс притулку) – призначений для обробки бізнес-логіки у сфері операційного обліку закладу, приймаючи вхідні REST-запити через програмний інтерфейс Animal Router / API. Центральним елементом архітектури є проміжне програмне забезпечення Tenant Middleware (Динамічна схема), яке вилучає контекст організації та перевизначає шлях пошуку таблиць для залежних модулів, серед яких функціонують модуль контенту візитки, модуль медичних записів (для ведення ветеринарних журналів) та модуль інвентаря та складу (для моніторингу залишків ресурсів). Усі ці модулі взаємодіють із рівнем доступу SQLAlchemy Core, який здійснює транзакційний запис даних у таблиці animals, medical_records, inventory, settings та shelter_tasks цільової ізольованої схеми;

– Booking Service (Сервіс готелю) – забезпечує повний спектр функціоналу для комерційного керування готелем для тварин, де вхідною точкою є програмний інтерфейс Booking Router / API. Запит проходить через ізолюючий шар Tenant Middleware (Динамічна схема), після чого передається до спеціалізованого модуля розрахунку вартості, який калькулює ціну проживання на основі тарифів вольєрів, та модуля онлайн-бронювань, що координує резервування місць у реальному часі й запобігає овербукінгу. Збереження результатів обробки забезпечується через рівень SQLAlchemy Core, який надсилає транзакції до таблиць bookings, aviaries,

hotel_pets, services, settings та booking_requests, розміщених у відповідній схемі тенанта.

Рівень бази даних

представляє собою єдиний інстанс реляційної СКБД petcloud, логічно розділений на дві ключові зони для дотримання високих стандартів безпеки та ізоляції в SaaS-індустрії. Перша зона – це глобальна схема public, яка є спільною для всієї платформи та зберігає централізовані таблиці користувачів і зареєстрованих організацій. Друга зона представлена сукупністю повністю відокремлених логічних просторів імен у вигляді схем tenant_schema, кожна з яких динамічно створюється під час реєстрації нової компанії та містить індивідуальний набір операційних таблиць притулку або готелю, доступні мікросервісам лише після виконання інструкції перемикання шляху пошуку

Така архітектурна організація компонентів гарантує абсолютну конфіденційність інформації клієнтів та дозволяє незалежно масштабувати, тестувати й оновлювати кожен окремий модуль платформи.

3.6 Діаграма розгортання

Діаграма розгортання відображає фізичну топологію обчислювальних вузлів, мережеві протоколи взаємодії, середовища виконання та особливості інфраструктурного розміщення компонентів платформи. Ця модель завершує цикл проєктування системи, демонструючи, яким чином мікросервіси та логічні модулі відображаються на різних апаратних ресурсах, а також фіксує межі безпеки мережевого периметра платформи (рис. 3.5).

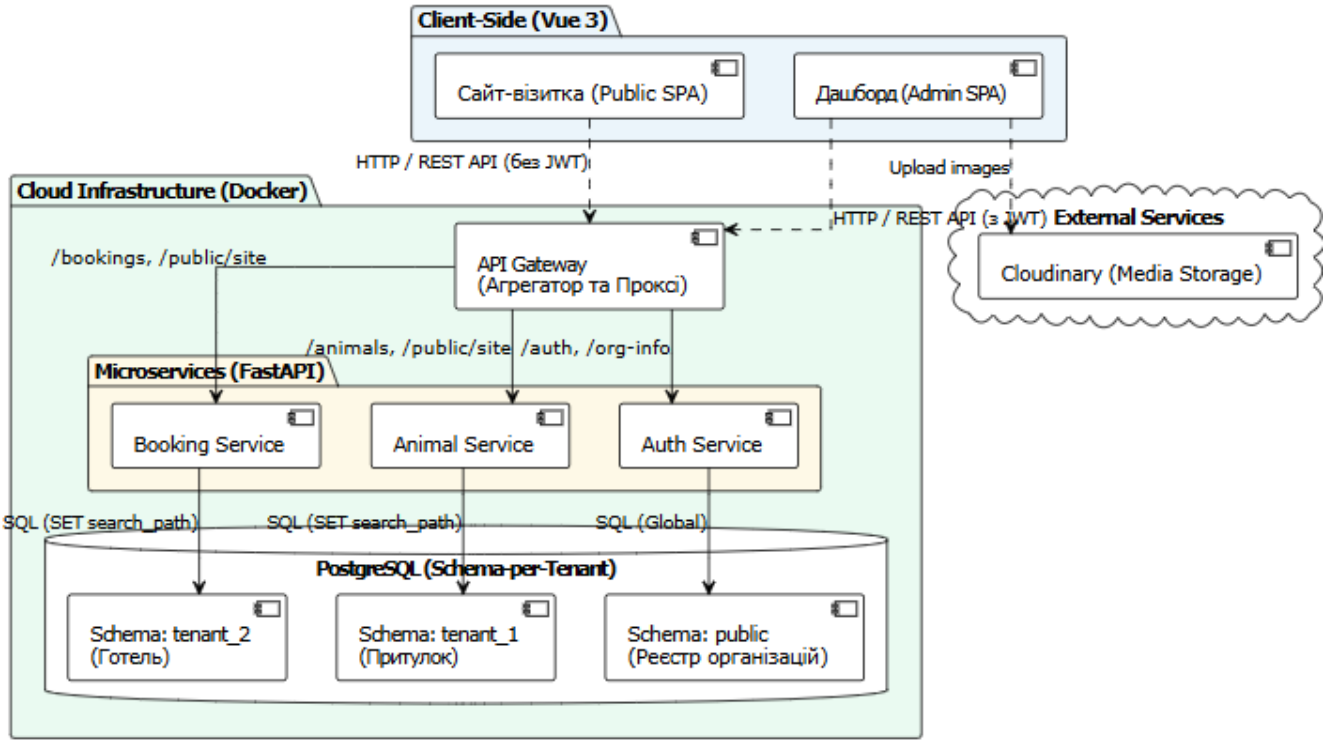

Рисунок 3.5 – Діаграма розгортання

Топологічна структура розподіленої інфраструктури, представлена на схемі, побудована за багаторівневим принципом і розподілена між трьома основними середовищами виконання:

Вузол клієнта (Client-Side) – представляє собою апаратний пристрій кінцевого користувача, у веббраузері якого виконується скомпільований пакет статичних файлів односторінкового застосунку. Цей рівень розділений на два автономні функціональні блоки: Дашборд (Admin SPA), що слугує внутрішнім кабінетом для авторизованого персоналу організацій, та Сайт-візитку (Public SPA), який є відкритим вебпорталом для зовнішніх відвідувачів та клієнтів. Взаємодія з основним сервером платформи відбувається виключно за бездержавним протоколом HTTP/HTTPS через надсилання асинхронних REST API запитів. При цьому Admin SPA наділений повноваженнями здійснювати пряме бінарне завантаження важкого графічного контенту (фотографій тварин) на зовнішній хмарний вузол, минаючи основні обчислювальні потужності системи.

Хмарна інфраструктура контейнеризації – центральний обчислювальний кластер, ізольований на рівні ядра операційної системи Linux Host за допомогою інструментів Docker та Docker Compose. Мережевий периметр цього вузла

захищений: єдиною публічною точкою входу, доступною з глобальної мережі Інтернет, є контейнер шлюзу безпеки API Gateway. Усі інші внутрішні компоненти розміщені всередині закритої віртуальної підмережі Docker Bridge, не мають відкритих назовні портів і комунікують між собою за внутрішніми DNS-іменами контейнерів.

Контейнерний кластер мікросервісів – розміщений всередині Docker-госта і складається з трьох ізольованих середовищ виконання застосунків FastAPI під управлінням ASGI-серверів Uvicorn. Шлюз API Gateway аналізує URL-шлях вхідного HTTP-запиту і виконує його зворотне проксіювання: запити до маршрутів /auth та /org-info перенаправляються на Auth Service; запити до операційних ендпоінтів обліку /animals та відкритих сайтів /public/site шлюзуються на Animal Service; транзакції бронювання кімнат /bookings направляються на Booking Service. Така архітектурна топологія повністю унеможливлює виникнення каскадних збоїв на фізичному рівні.

Вузол бази даних – представляє собою виділений контейнер СКБД PostgreSQL, робота якого налаштована через іменовані томи Docker Volumes для забезпечення постійного зберігання даних (персистентності) на жорсткому диску хост-машини. Фізичний екземпляр бази даних суворо розділений на логічні простори імен для реалізації стратегії Schema-per-Tenant. Мікросервіс авторизації (Auth Service) взаємодіє зі спільною глобальною схемою public (Реєстр організацій) за допомогою прямих SQL-запитів. Операційні мікросервіси Animal Service та Booking Service підключаються до індивідуальних, повністю ізольованих схем клієнтів (schema_tenant_1 для притулку, schema_tenant_2 для готелю), перемикання контексту між якими відбувається динамічно під час кожного сеансу зв'язку шляхом виконання інструкції SET search_path.

Зовнішні хмарні сервіси – представлена розподіленою інфраструктурою сховища медіафайлів Cloudinary. Цей вузол винесено за межі основного хостингу платформи для оптимізації дискового простору. Cloudinary приймає зображення від клієнтського застосунку, оптимізує їх та повертає в систему унікальний текстовий URL-рядок, який бекенд-мікросервіси зберігають у відповідних

таблицях PostgreSQL, делегуючи подальшу доставку графічного контенту кінцевим користувачам глобальній мережі CDN.

Спроєктована топологія діаграми розгортання повністю задовольняє сучасні інженерні вимоги до побудови відмовостійких хмарних систем, забезпечує високий рівень безпеки даних за рахунок приховування внутрішньої мережі контейнерів та мінімізує витрати на інфраструктуру завдяки ефективному пулінгу ресурсів в межах єдиного Docker Host.

3.7 ER Діаграма

Для забезпечення структурованого зберігання даних, підтримки цілісності інформації та реалізації стратегії мультитенантності, було розроблено логічну модель бази даних. Вона представлена у вигляді діаграми «сутність-зв'язок», яка ілюструє фізичну та логічну архітектуру сховищ, розділених на спільні та специфічні схеми даних.

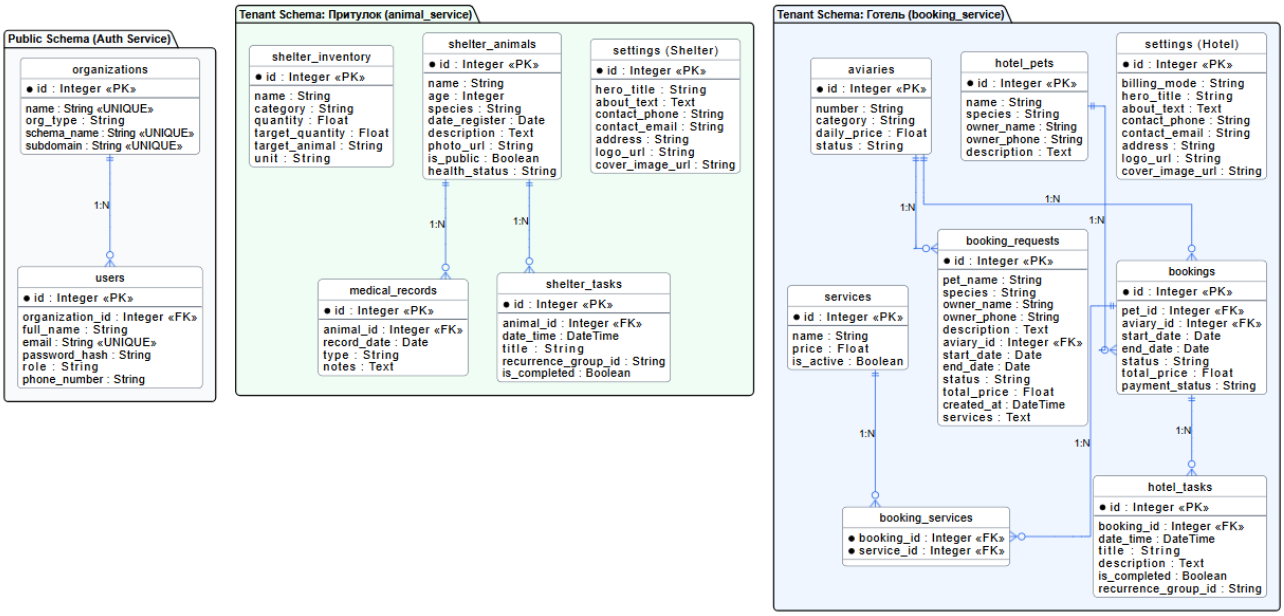

Рисунок 3.6 – ER діаграма

Логічна структура бази даних побудована за ієрархічним принципом, що забезпечує чітке розмежування доступу:

1. Public Schema (Auth Service) – містить глобальні таблиці, до яких мають доступ усі компоненти системи для автентифікації користувачів. Основною

сутністю є organizations, яка зберігає метадані тенантів (назви, типи, піддомени), та users, що відповідає за облікові записи персоналу з прив'язкою до конкретної організації.

2. Tenant Schemas (Domain isolation) – кожна організація функціонує у власній ізольованій схемі, що унеможливлює витік даних між різними тенантами.

Схема «Притулок» зосереджена навколо сутності shelter_animals. З нею пов'язані дочірні таблиці med_records, shelter_tasks та shelter_inventory.

Схема «Готель» орієнтована на бізнес-процеси бронювання. Ключовою сутністю є bookings, яка акумулює дані про замовлення, прив'язуючи hotel_pets до aviaries. Таблиця booking_services реалізує зв'язок «багато-до-багатьох» між замовленнями та додатковими послугами, що дозволяє динамічно формувати фінальну вартість заїзду.

Висновки до розділу 3

У третьому розділі виконано комплексне проєктування логічної архітектури, бізнес-логіки та інфраструктури хмарної SaaS-платформи на базі мікросервісного стеку FastAPI, Vue 3 та СКБД PostgreSQL. Побудована діаграма діяльності деталізувала алгоритми функціонування шлюзу API Gateway щодо перехоплення, JWT-автентифікації та захищеної маршрутизації вхідних HTTP-запитів. Спроєктована діаграма класів заклала підгрунтя для ізоляції даних за стратегією Schema-per-Tenant. Компонентна декомпозиція дозволила успішно розподілити загальну бізнес-логіку на три автономні сервіси, що повністю унеможливлює виникнення каскадних збоїв. Фізична топологія розподіленої інфраструктури, візуалізована на діаграмі розгортання, зафіксувала системну контейнеризацію Docker та Docker Compose. Делегування збереження медіаконтенту зовнішньому хмарному CDN-сховищу Cloudinary дозволило оптимізувати пропускну здатність серверів. Отримані проєктні рішення утворили цілісний інженерний фундамент, повністю готовий до етапу безпосередньої програмної реалізації.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

Цей розділ присвячено етапу безпосередньої розробки вихідного коду хмарної SaaS-платформи та перевірці її працездатності. Головною метою даного етапу є практична імплементація архітектурних рішень, даталогічних моделей та алгоритмів маршрутизації, що були спроєктовані та обґрунтовані у попередньому розділі.

Процес програмної реалізації системного комплексу декомпозовано на кілька логічних стадій. На першому етапі розглядається створення серверної інфраструктури (блок бекенду), що включає конфігурацію єдиного шлюзу API Gateway, програмування ізольованих мікросервісів на базі асинхронного фреймворку FastAPI та імплементацію складного механізму мультитенантності на рівні взаємодії з СКБД PostgreSQL. Наступним кроком описується розробка реактивного клієнтського застосунку (блок фронтенду) за допомогою архітектури SPA на базі фреймворку Vue 3. Завершальною стадією є проведення функціонального тестування готового програмного продукту для підтвердження його відповідності початковим бізнес-вимогам та перевірки коректності обробки інформації в умовах ізольованих просторів орендарів.

4.1 Розробка серверної частини платформи

Програмна реалізація серверного рівня платформи базується на принципах мікросервісної архітектури, де кожен функціональний домен (авторизація, облік тварин, бронювання) виділено в автономний сервіс. Такий підхід забезпечує високу модульність, спрощує масштабування окремих компонентів та ізолює бізнес-логіку. Центральним елементом інтеграції та оркестрації цих сервісів є інструмент Docker Compose, який описує фізичну топологію мережі, параметри середовища виконання та правила взаємодії між контейнерами.

Серверна частина реалізована за допомогою сучасного асинхронного фреймворку FastAPI, що дозволяє досягти високої продуктивності при обробці конкурентних запитів. Для забезпечення узгодженості між середовищами розробки

та виробничим середовищем, кожний сервіс інкапсульовано у Docker-контейнер із власною файловою системою та конфігураційним оточенням.

Для забезпечення чистоти архітектури, ізоляції компонентів та спрощення спільної розробки, файлова структура бекенд-частини платформи організована за принципом розподіленої відповідальності. Кожен мікросервіс розташований у власній окремій директорії, містить індивідуальні інструкції для збирання контейнера та оперує суворо визначеним набором файлів, що відповідають за роутинг, бізнес-логіку та взаємодію з базою даних (рис. 4.1).

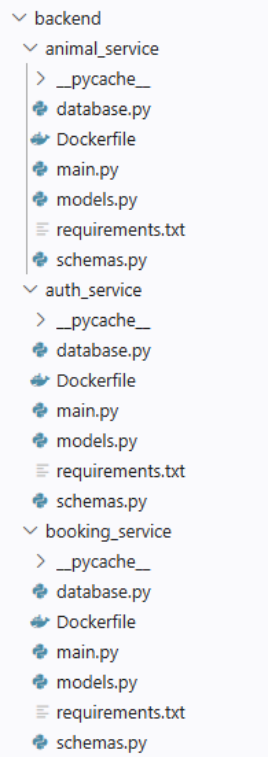

Рисунок 4.1 – Структура бекенду

Узагальнений опис структури папок та призначення ключових файлів у мікросервісах auth_service, animal_service та booking_service виглядає так:

– main.py – центральний файл ініціалізації мікросервісу, де створюється екземпляр додатка FastAPI, конфігуруються CORS-політики, підключаються проміжні шари (middlewares) та реєструються API-маршрутизатори;

– database.py – модуль керування підключенням до бази даних PostgreSQL. Саме тут налаштовується пул з'єднань, створюється фабрика асинхронних сесій

SQLAlchemy та реалізуються механізми динамічного керування шляхом пошуку простору імен (SET search_path) для забезпечення ізоляції тенантів;

– models.py – декларативні ORM-моделі для відображення об'єктів коду на реляційні таблиці бази даних;

– schemas.py – декларативні схеми валідації та серіалізації даних на базі бібліотеки Pydantic. Вони визначають строгі контракти для вхідних JSON-пакетів (Request Body) та вихідних відповідей сервера (Response Body);

– requirements.txt – зафіксований перелік зовнішніх бібліотек та залежностей, необхідних для стабільної роботи сервісу;

– Dockerfile – покрокова інструкція для автоматичного збирання легковагового контейнерного образу (на базі Docker-образу Python slim), яка копіює вихідний код, встановлює залежності, ізолює внутрішній порт контейнера та запускає вебсервер Uvicorn.

Така структуризація дозволяє гнучко розвивати, тестувати й оновлювати кожен сервіс окремо, повністю нівелюючи ризик виникнення каскадних помилок у розподіленій архітектурі платформи.

4.1.1 Реалізація API Gateway

Для централізованого управління мережевим трафіком та забезпечення єдиної точки входу (Single Point of Entry) для клієнтського застосунку було спроєктовано та реалізовано прикладний шлюз API Gateway. Програмну імплементацію шлюзу виконано мовою Python з використанням асинхронного фреймворку FastAPI.

На базовому рівні ініціалізації шлюзу реалізовано налаштування політик CORS за допомогою вбудованого CORSMiddleware. Це дозволяє безпечно приймати асинхронні запити від фронтенд-частини на базі Vue.js, забезпечуючи коректну передачу облікових даних та підтримку всіх необхідних HTTP-методів.

Головним обчислювальним компонентом API Gateway є універсальний асинхронний контролер маршрутизації, який перехоплює всі вхідні HTTP-запити та аналізує їхній URL-шлях. На основі розробленої системи лексичних правил

шлюз динамічно визначає цільовий мікросервіс платформи всередині Docker-мережі. Наприклад, запити з префіксами /auth, /login або /org-info маршрутизуються до auth_service; домени /animals, /medical-records та /tasks обслуговуються сервісом animal_service; а транзакції /bookings та /aviaries перенаправляються до booking_service. Окремим алгоритмічним блоком виділено обробку публічних маршрутів (префікс /api/public/), які агрегують відкриті дані для сайтів-візиток без вимоги валідації безпекових маркерів.

Важливою архітектурною перевагою реалізованого шлюзу є наявність механізму інтелектуальної контекстної маршрутизації на основі JWT-маркерів. Для загальносистемних ендпоінтів, які існують у кількох мікросервісах одночасно, шлюз перехоплює заголовок Authorization. За допомогою криптографічної бібліотеки jose система декодує корисне навантаження токена та зчитує поле org_type. Якщо поточний користувач належить до притулку, запит динамічно перенаправляється до Animal Service, в іншому випадку – до Booking Service.

Безпосереднє проксіювання (Reverse Proxying) реалізовано через асинхронні сесії бібліотеки httpx. Gateway формує нову цільову URL-адресу, прокидає параметри запиту (query parameters), зчитує бінарне тіло та копіює заголовки, попередньо очищуючи заголовок host для уникнення мережевих конфліктів. Використання конструкції httpx.AsyncClient гарантує неблокуюче виконання запитів, а вбудований блок обробки винятків автоматично генерує стандартизовану відповідь із HTTP-кодом 503 (Service Unavailable) у разі тимчасової недоступності цільового мікросервісу, запобігаючи неконтрольованому зависанню клієнтського застосунку.

4.1.2 Програмування мікросервісів на FastAPI

Розробка функціональних модулів бекенд-частини платформи базується на декларативному та асинхронному програмуванні в середовищі фреймворку FastAPI. Логіка кожного мікросервісу декомпозована на ізольовані рівні: рівень маршрутизатора, рівень серверної валідації (схеми Pydantic) та об'єктно-реляційне відображення.

Рівень об'єктно-реляційного мапінгу предметної області описується за допомогою декларативної бази SQLAlchemy. Класи сутностей жорстко зв'язані з відповідними реляційними таблицями, а зв'язки типу «один-до-багатьох» (1:N) між карткою тварини та її медичною історією чи щоденними завданнямит догляду реалізовані через визначення зовнішніх ключів. Приклад опису базових ORM-моделей обліку тварин та медичних записів наведено у лістингу нижче.

```
class ShelterAnimal(Base):
    __tablename__ = "shelter_animals"
    id = Column(Integer, primary_key=True)
    name = Column(String)
    age = Column(Integer)
    species = Column(String)
    date_register = Column(Date, default=date.today)
    description = Column(Text)
    photo_url = Column(String)
    is_public = Column(Boolean, default=False)
    health_status = Column(String, default="Здоровий")
```

Рівень серверної валідації базується на декларативних моделях бібліотеки Pydantic v2. Сувора типізація вхідних пакетів даних запобігає передачі некоректних форматів. Для кожної сутності розроблено дві базові схеми: створення об'єкта та транзакційну схему відповіді сервера, яка наслідує базові поля та додає системні атрибути. Приклад реалізації схем валідації наведено нижче у лістингу.

```
class AnimalCreate(BaseModel):
    name: str
    age: int
    species: str
    description: Optional[str] = None
    photo_url: Optional[str] = None
    health_status: Optional[str] = None
    is_public: Optional[bool] = False

class AnimalResponse(AnimalCreate):
    id: int
    date_register: Optional[date] = None

    class Config:
        from_attributes = True
```

Використання внутрішнього класу конфігурації з прапорцем from_attributes = True є критично важливим для мікросервісної архітектури, оскільки це дозволяє

Pydantic автоматично зчитувати дані безпосередньо з ORM-об'єктів SQLAlchemy, суттєво прискорюючи мапінг даних та формування JSON-відповідей.

Рівень представлення та ініціалізації додатка реалізовано у файлах main.py. застосунок ініціює екземпляр FastAPI(), підключає проміжне програмне забезпечення CORS для взаємодії з клієнтським портом та інтегрує хмарне сховище медійного контенту Cloudinary. Обробка бізнес-логіки та асинхронна робота з СКБД реалізована за допомогою механізму впровадження залежностей (Dependency Injection) через конструкцію Depends(get_db_with_schema). Нижче у лістингу продемонстровано приклад реалізації ендпоінту для додавання нової тварини в базу даних.

```
app = FastAPI(title="PetCloud Animal Service")

@app.post("/animals", response_model=schemas.AnimalResponse)
def create_animal(animal: schemas.AnimalCreate, db: Session =
Depends(get_db_with_schema)):
    db.expire_on_commit = False

    new_animal = models.ShelterAnimal(
        name=animal.name,
        age=animal.age,
        species=animal.species,
        description=animal.description,
        photo_url=animal.photo_url,
        health_status=animal.health_status,
        is_public=animal.is_public or False
    )

    db.add(new_animal)
    db.flush()
    db.refresh(new_animal)
    db.commit()
    return new_animal
```

Завдяки такій трирівневій структурі (Models-Schemas-Routers), код мікросервісу FastAPI залишається чистим, легко підтримується та масштабується, а також забезпечує чітку ізоляцію бізнес-процесів у межах виділеного домену системи.

4.1.3 Практична реалізація мультитенантності

Найбільш складним архітектурним завданням при розробці мультитенантної SaaS-платформи є забезпечення абсолютної ізоляції даних різних організацій-орендарів в межах одного фізичного екземпляра СКБД PostgreSQL. У межах даного проєкту цю проблему успішно розв'язано на рівні динамічного керування логічними просторами імен за стратегією Schema-per-Tenant. Практичну логіку створення та ізоляції просторів імплементовано у модулі database.py.

Процес ініціалізації нового тенанта відбувається асинхронно в момент реєстрації компанії. Система викликає інфраструктурну функцію create_tenant_schema(schema_name, org_type), яка через пряме підключення до рушія бази даних engine.connect() виконує інструкцію створення логічної схеми в PostgreSQL та динамічно генерує в її межах ізольований набір операційних таблиць, що відповідають профілю організації.

```
def create_tenant_schema(schema_name: str, org_type: str):
    with engine.connect() as connection:
        connection.execute(text(f'CREATE SCHEMA IF NOT EXISTS "{schema_name}";'))
        connection.execute(text(f'SET search_path TO "{schema_name}";'))

        if org_type == "shelter":
            tables_to_create = [
                models.ShelterAnimal.__table__,
                models.MedicalRecord.__table__,
                models.ShelterTask.__table__,
                models.ShelterInventory.__table__,
                models.SiteSettings.__table__
            ]
        else org_type == "hotel":
            tables_to_create = [
                models.HotelPet.__table__,
                models.Aviary.__table__,
                models.Service.__table__,
                models.Booking.__table__,
                models.BookingServiceLink.__table__,
                models.BookingRequest.__table__,
                models.HotelTask.__table__,
                models.HotelSettings.__table__
            ]

        models.Base.metadata.create_all(bind=connection, tables=tables_to_create)
        connection.commit()
```

Безпосередня ізоляція SQL-транзакцій під час обробки HTTP-запитів поточних користувачів реалізована за допомогою спеціальної контекстної

залежності get_db_with_schema. Ця функція виступає проміжним фільтром, який автоматично перехоплює HTTP-запит, вилучає з його заголовка або авторизаційного токена унікальне ім'я схеми тенанта й перевизначає системний шлях пошуку таблиць. Нижче наведено вихідний код реалізації даного інфраструктурного механізму.

```
def get_db_with_schema(schema_name: str = Depends(get_tenant_schema)):
    db = SessionLocal()
    try:
        db.execute(text(f'SET search_path TO "{schema_name}"'))
        yield db
    finally:
        db.close()
```

Команда db.execute(text(f'SET search_path TO "{schema_name}";')) перевизначає поточну робочу зону СКБД для поточної сесії зв'язку. Усі наступні ORM-операції SQLAlchemy виконуватимуться виключно всередині вказаної схеми, роблячи дані інших організацій абсолютно невидимими та недосяжними для поточного користувача. Після завершення HTTP-транзакції конструкція finally гарантує закриття сесії й повернення з'єднання до спільного пулу підключень.

Така програмна реалізація мультитенантності забезпечує найвищий рівень безпеки та конфіденційності даних у SaaS-індустрії, мінімізує витрати на утримання хмарної інфраструктури та дозволяє обслуговувати тисячі ізольованих організацій в межах єдиної розгорнутої бази даних.

4.2 Розробка клієнтської частини платформи

Клієнтська частина хмарної SaaS-платформи спроєктована для забезпечення високої швидкодії, реактивності та зручного користувацького досвіду. В якості основного інструменту розробки інтерфейсу обрано прогресивний JavaScript-фреймворк Vue 3 з використанням архітектурного підходу Composition API. Застосунок функціонує за принципом SPA, що дозволяє динамічно перемальовувати DOM-дерево та оновлювати контент без примусового перезавантаження вебсторінки веббраузером, суттєво зменшуючи мережеві затримки

4.2.1 Реалізація мережевого шару та архітектура API-маршрутів

Для організації безпечної та структурованої HTTP-комунікації між компонентами Vue та центральним шлюзом бекенду використовується бібліотека Axios. Увесь мережевий шар платформи абстраговано та централізовано у конфігураційному файлі api.js. Важливим інженерним рішенням є використання механізму перехоплювачів, що дозволяє автоматизувати процес додавання маркера безпеки JWT до кожного вихідного запиту. У лістингу нижче наведено реалізацію перехоплювача.

```
axios.interceptors.request.use((config) => {
    const token = localStorage.getItem('token');

    if (token) {
        config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
}, (error) => {
    return Promise.reject(error);
});
```

Для забезпечення суворої типізації та зручності взаємодії всі API-запити згруповано за логічними об'єктами (authApi, animalsApi, bookingApi), кожен з яких відповідає за свою доменну область.

Першим та базовим рівнем абстракції є об'єкт authApi, який інкапсулює логіку взаємодії із сервісом авторизації. Його методи забезпечують безпечний вхід до системи, реєстрацію нових організацій та управління обліковими записами персоналу в межах глобальної бази даних платформи (табл. 4.1).

Таблиця 4.1 – Повний реєстр методів authApi

HTTP	Маршрут	Вхідні дані	Повертає
POST	/login	Об’єкт із username, password	JWT-токен, метадані користувача
	/register	Анкетні дані компанії та адміна	Статус реєстрації
	/auth/workers	Дані нового працівника та роль	Об'єкт створеного працівника

Кінець таблиці 4.1

GET	/org-info/{subdomain}	Піддомен (URL param)	Дані публічної візитки
	/auth/workers	–	Масив об'єктів (перелік персоналу)
	/auth/org	–	Об’єкт профілю організації
PUT	/auth/org/subdomain	Об'єкт із новим subdomain	Оновлений статус організації
	/auth/workers/{id}	ID (URL), оновлені дані працівника	Оновлений запис облікового запису
DELETE	/auth/workers/{id}	ID працівника (через URL param)	Статус успішного видалення

Для функціонального забезпечення організацій типу «притулок» розроблено об'єкт animalsApi. Він надає фронтенд-компонентам набір методів для повноцінного ведення карток тварин, фіксації медичних записів, моніторингу складських запасів та організації щоденних завдань догляду (табл. 4.2).

Таблиця 4.2 – Повний реєстр методів об’єкта animalApi

HTTP	**Маршрут**	**Вхідні дані**	**Повертає**
POST	/animals	Анкетні дані тварини (JSON)	Створений об’єкт тварини
	/medical-records	Дані процедури/огляду (JSON)	Створений медичний запис
	/inventory	Дані про товар (категорія, кількість)	Створена позиція на складі
	/tasks	Дані завдання (JSON)	Створений розклад завдання
GET	/shelter/stats	–	Зведені статистичні метрики притулку
	/public/site	schema_name (query param)	Дані для публічного лендінгу
	/animals	–	Масив профілів усіх тварин
	/animals/{id}	ID тварини (через URL param)	Детальний профіль однієї тварини
	/medical-records/animal/{id}	ID тварини (через URL param)	Історія медичних записів
	/inventory	–	Масив залишків на складі
	/tasks	–	Перелік усіх щоденних завдань

Кінець таблиці 4.2

PUT	/animals/{id}	ID (URL), оновлені дані	Оновлений об'єкт тварини
	/inventory/{id}	ID товару (URL), нові залишки	Оновлена позиція складу
	/hotel_tasks/{id}	ID (URL), зміна статусу виконання	Оновлений стан задачі
DELETE	/animals/{id}	ID тварини (через URL param)	Статус видалення картки
	/medical-records/{id}	ID медичного запису (URL)	Статус видалення процедури
	/inventory/{id}	ID товару (через URL param)	Статус списання/видалення
	/tasks/{id}	ID завдання (через URL param)	Видалена задача
	/tasks/{id}/recurrence	ID завдання (через URL param)	Видалена ціла група повторюваних задач

Комерційний модуль платформи, орієнтований на роботу зооготелів, обслуговується через об'єкт bookingApi. Цей інтерфейс відповідає за отримання бізнес-статистики, управління номерним фондом вольєрів, обробку зовнішніх клієнтських заявок та безпосереднє оформлення транзакцій бронювання місць (табл. 4.3).

Таблиця 4.3 – Повний реєстр методів об’єкта bookingApi

HTTP	**Маршрут**	**Вхідні дані**	**Повертає**
POST	/settings/upload	Медіафайл (об'єкт FormData)	URL-посилання з хмари Cloudinary
	/services	Дані нової послуги (ціна, назва)	Створений запис сервісу
	/hotel_pets	Анкетні дані власника/улюбленця	Створена клієнтська картка
	/aviaries	Характеристики вольєра (JSON)	Створений об'єкт вольєра
	/bookings	Об’єкт резервації (дати, послуги)	Підтверджений запис бронювання
	/public/booking-requests	schema_name, дані клієнта (JSON)	Створена зовнішня заявка
	/hotel_tasks	Дані завдання (JSON)	Створене операційне завдання

Кінець таблиці 4.3

GET	/hotel/stats	–	Зведені статистичні метрики готелю
	/settings	–	Поточні налаштування готелю
	/public/hotel/site	schema_name (query param)	Дані для публічного сайту готелю
	/services/all	–	Всі додаткові послуги (прайс-лист)
	/services	–	Тільки активні послуги для модалки
	/hotel_pets	–	База карток власників та тварин
	/aviaries	–	Перелік усього номерного фонду
	/bookings	–	Масив поточних бронювань
	/bookings/aviary/{id}	ID вольєра (через URL param)	Історія та зайнятість кімнати
	/booking-requests	–	Усі вхідні заявки від клієнтів
	/booking-requests/pending		Лише необроблені заявки
	/hotel_tasks		Перелік завдань персоналу готелю
PUT	/settings	Новий конфігураційний об'єкт	Оновлені налаштування
	/services/{id}	ID послуги (URL), нові параметри	Оновлений запис сервісу
	/aviaries/{id}	ID (URL), нові характеристики	Оновлений об'єкт вольєра
	/booking-requests/{id}	ID заявки, зміна статусу	Оновлений статус заявки
	/hotel_tasks/{id}	ID (URL), зміна статусу виконання	Оновлений стан задачі
DELETE	/services/{id}	ID послуги (через URL param)	Статус видалення послуги
	/aviaries/{id}	ID вольєра (через URL param)	Статус видалення номера
	/booking-requests/{id}	ID заявки (через URL param)	Видалена або відхилена заявка
	/hotel_tasks/{id}	ID завдання (через URL param)	Статус видалення задачі
	/hotel_tasks/{id}/recurrence	ID завдання (через URL param)	Статус видалення групи задач

Використання єдиного реєстру API-методів гарантує узгодженість даних, суттєво спрощує підтримку маршрутів у разі зміни бекенд-ендпоінтів та робить

процес розробки нових Vue-компонентів інтуїтивно зрозумілим. Будь-яка зміна на стороні серверних мікросервісів потребує коригування лише одного відповідного методу в об'єкті абстракції, без необхідності редагування логіки в десятках компонентів інтерфейсу.

4.2.2 Менеджмент стану

Для управління глобальним станом застосунку та забезпечення реактивної синхронізації даних між непов'язаними компонентами інтерфейсу в клієнтську частину платформи інтегровано офіційну бібліотеку керування станом Pinia. Це дозволяє уникнути надлишкового прокидання пропсів (props drilling) крізь глибокі рівні вкладеності компонентів Vue та централізувати збереження транзакційних метаданих.

Фундаментальним елементом безпеки та підтримки концепції мультитенантності на клієнтській стороні є глобальне сховище useAuthStore, яке повністю контролює життєвий цикл сесії користувача. На початковому етапі життєвого циклу застосунку виконується декларативне визначення реактивного стану сховища. Реалізацію структури збереження контексту користувача та його ідентифікаційних маркерів безпеки наведено у лістингу нижче.

```
state: () => ({
    user: JSON.parse(localStorage.getItem('user')) || null,
    token: localStorage.getItem('token') || null,
    orgType: localStorage.getItem('orgType') || null,
    role: localStorage.getItem('role') || null,
}),
```

Під час ініціалізації процесу автентифікації, після успішної перевірки облікових даних на стороні API Gateway, отриманий криптографічний JWT-токен, рольова приналежність, а також унікальний ідентифікатор організації та її тип (org_type) записуються в реактивний стан сховища. Для запобігання втрати робочої сесії при випадковому або примусовому оновленні сторінки веббраузером користувача, дані дублюються у постійне локальне сховище браузера localStorage.

Використання обчислюваних реактивних властивостей сховища є інженерною основою для забезпечення гнучкості та динамічності інтерфейсу SPA-

застосунку. Геттери функціонують як кезовані обчислювальні контури, що миттєво реагують на зміну глобального стану авторизації. Практичну реалізацію аналітичних геттерів сховища представлено у лістингу нижче.

```
getters: {
    isShelter: (state) => state.orgType === 'shelter',
    isHotel: (state) => state.orgType === 'hotel',
    isAuthenticated: (state) => !!state.token,
    userName: (state) => state.user ? state.user.full_name : 'Гість',
    isAdmin: (state) => state.role === 'admin'
}
```

Завдяки впровадженню описаної архітектури геттерів, головні компоненти навігації (зокрема, бічна панель Sidebar) динамічно перебудовують конфігурацію свого меню в режимі реального часу. Наприклад, якщо систему активує адміністратор некомерційного притулку, реактивно рендериться інтерфейс медичних карток, складського інвентаря та специфічних черг завдань. У разі успішного входу працівника комерційного готелю для тварин, інтерфейс автоматично адаптується під бізнес-домен, відображаючи інтерактивний календар бронювань, картки номерного фонду вольєрів та калькулятори додаткових послуг. Таке архітектурне рішення забезпечує безшовний мультитенантний клієнтський досвід у межах єдиного скомпільованого клієнтського пакета платформи.

4.2.3 Інтеграція з Cloudinary

Сучасні SaaS-платформи оперують великими обсягами графічної інформації. Збереження бінарних даних зображень безпосередньо у реляційній базі даних PostgreSQL є інженерним антипатерном, оскільки це суттєво знижує загальну продуктивність СКБД, ускладнює резервне копіювання та збільшує витрати на інфраструктуру. Для вирішення цієї проблеми та оптимізації роботи з медіаконтентом (фотографіями тварин, логотипами притулків, знімками вольєрів) в архітектуру платформи було інтегровано зовнішній спеціалізований хмарний сервіс Cloudinary. Таке рішення забезпечує автоматичну компресію зображень та їх миттєву доставку кінцевим користувачам через глобальну мережу доставки контенту.

Процес управління медіафайлами охоплює взаємодію клієнтської та серверної частин. На фронтенді Vue 3 користувач обирає графічний файл, який пакується у стандартний веб-об'єкт FormData і через абстракцію мережевого шару відправляється на відповідний ендпоінт шлюзу.

На рівні бекенд-мікросервісів прийом та переадресацію файлів реалізовано за допомогою офіційної Python-бібліотеки cloudinary. З міркувань кібербезпеки критичні облікові дані підключення не зберігаються у вихідному коді, а динамічно зчитуються зі змінних оточення (Environment Variables). Програмну реалізацію конфігурації та функції завантаження наведено у лістингу.

```
def configure_cloudinary():
    cloud_name = os.getenv("CLOUDINARY_CLOUD_NAME")
    api_key = os.getenv("CLOUDINARY_API_KEY")
    api_secret = os.getenv("CLOUDINARY_API_SECRET")
    if not cloud_name or not api_key or not api_secret:
        raise RuntimeError("Cloudinary credentials are not set in environment
variables")
    cloudinary.config(
        cloud_name=cloud_name,
        api_key=api_key,
        api_secret=api_secret,
        secure=True
    )

def upload_image_to_cloudinary(file: UploadFile, folder: str):
    file_content = file.file.read()
    result = cloudinary.uploader.upload(
        file_content,
        folder=folder,
        resource_type="image"
    )
    return result
```

Обробка HTTP-запитів на завантаження здійснюється через спеціалізовані маршрути, захищені політикою перевірки прав адміністратора Depends. Коли файл надходить до ендпоінту, бекенд перехоплює його як UploadFile, зчитує в оперативну пам'ять і передає в інфраструктуру Cloudinary із вказівкою чіткої цільової директорії.

```
@app.post("/settings/upload", response_model=schemas.ImageUploadResponse)
async def upload_settings_image(file: UploadFile = File(...), _admin: dict =
Depends(require_admin)):
    result = upload_image_to_cloudinary(file, folder="petcloud/shelter/settings")
    return {"url": result["secure_url"], "public_id": result["public_id"]}
```

```
@app.post("/animals/upload", response_model=schemas.ImageUploadResponse)
async def upload_animal_image(file: UploadFile = File(...), _admin: dict =
Depends(require_admin)):
    result = upload_image_to_cloudinary(file, folder="petcloud/shelter")
    return {"url": result["secure_url"], "public_id": result["public_id"]}
```

У результаті успішної транзакції хмарний сервіс генерує унікальне криптографічне посилання secure_url. Сервер повертає цей рядок у форматі JSON-відповіді на фронтенд. Після цього клієнтський застосунок використовує отриманий текстовий URL для фінального збереження картки тварини чи налаштувань профілю в реляційній базі PostgreSQL. Такий підхід повністю розділяє логіку зберігання структурованих даних і важкого медіаконтенту, гарантуючи високу масштабованість розроблюваної SaaS-платформи.

4.3 Інтерфейс програми

Графічний інтерфейс користувача розроблюваної SaaS-платформи виступає безпосереднім інструментом взаємодії персоналу організацій-орендарів та зовнішніх відвідувачів із розподіленою мікросервісною архітектурою бекенду. Візуальну частину системи реалізовано як адаптивний та реактивний вебзастосунок SPA на базі фреймворку Vue 3. Для забезпечення сучасної естетики, уніфікації графічних елементів, компонування складних табличних даних та кастомних календарів у якості базової бібліотеки UI-компонентів було інтегровано фреймворк Ant Design Vue.

Впровадження готового набору високорівневих компонентів дозволило суттєво скоротити терміни розробки інтерфейсу та зосередитися на бізнес-логіці динамічного підлаштування інтерфейсу під конкретний тип організації.

4.3.1 Головна панель управління та модуль автентифікації

Вхідною точкою для користувачів платформи є екранна форма автентифікації (рис. 4.2). Вона містить поля для введення облікових даних працівника, валідація яких у реальному часі здійснюється за допомогою реактивних правил зв'язування даних v-model. Після успішної перевірки токена на

стороні API Gateway, користувач перенаправляється на головний екран системи – робочу панель (рис. 4.3).

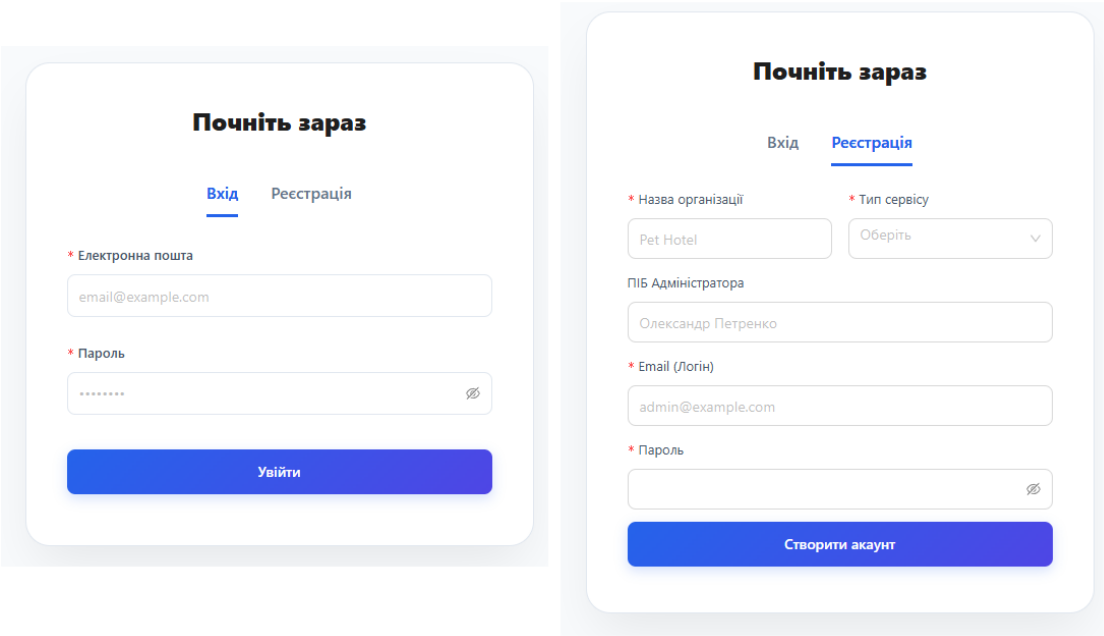

Рисунок 4.2 – Форма авторизації та реєстрації

Головна панель управління спроєктована за принципом високої модульності та композиції незалежних інтерфейсних шарів. Каркас сторінки побудовано на базі контейнерної сітки a-layout від UI-фреймворку Ant Design Vue. Логічна структура панелі управління складається з трьох ключових інфраструктурних компонентів.

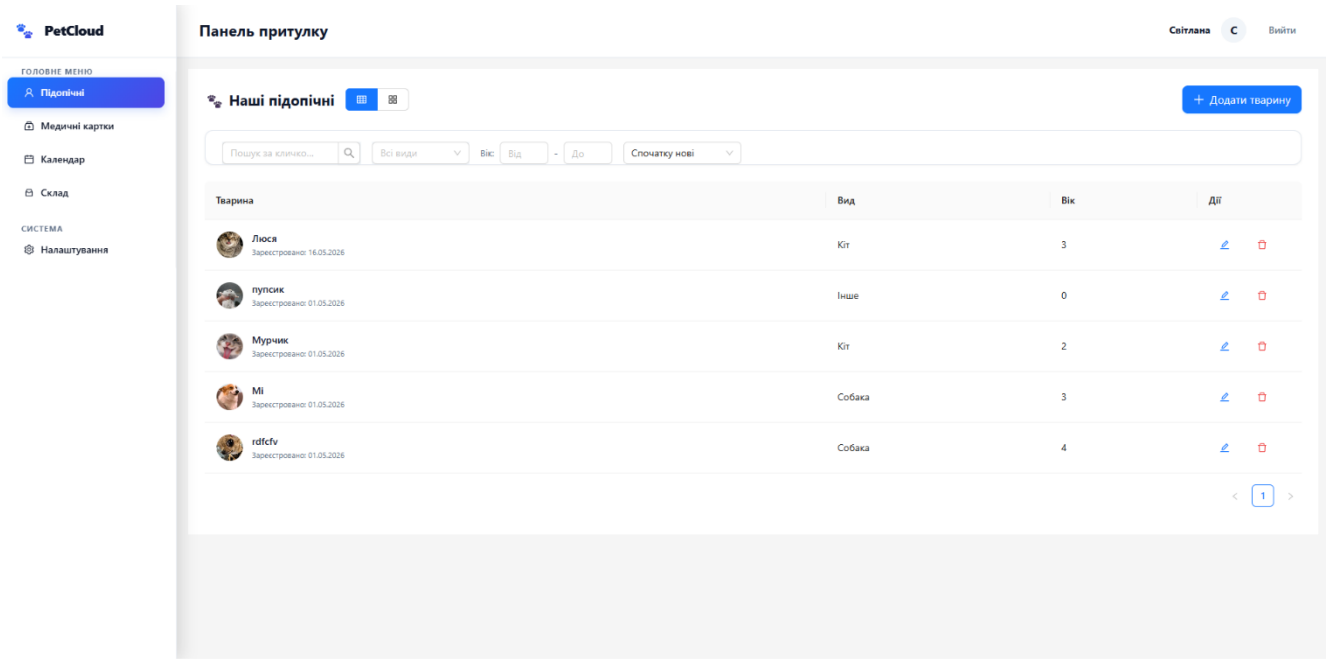

Рисунок 4.3 – Графічний інтерфейс головної панелі

Компонент бічного меню – відповідає за глобальну навігацію платформою. Він містить логотип платформи «PetCloud» та інтерактивне меню a-menu. Спираючись на глобальний стан Pinia-сховища authStore, компонент реалізує гнучку трансформацію написів: для користувачів із типом організації hotel головний пункт меню відображається як «Клієнти», тоді як для некомерційних притулків назва автоматично змінюється на «Підопічні». Навігація між робочими просторами реалізована без перезавантаження сторінок за допомогою генерації подій emit('change-view', view) при кліку на відповідні елементи меню.

Компонент верхнього заголовка – забезпечує виведення динамічного заголовка поточної сторінки та керування інтерактивними сповіщеннями. У правій частині компонента розміщено елемент a-popover, інтегрований із глобальним інфраструктурним сховищем notificationStore. При кліку на іконку сповіщень система розгортає компактний список a-list із увімкненим асинхронним індикатором завантаження a-spin. У цьому списку відображаються поточні операційні завдання на сьогодні, термін виконання яких минає найближчим часом, із зазначенням точного часу та імені тварини.

Центральна контентна зона – динамічний робочий простір, архітектура якого базується на патерні умовного перемикання представлень (Dynamic View Switching) у режимі реального часу. Контентна область зв'язана з реактивною змінною currentView батьківського додатка. Залежно від обраного пункту в бічному меню та бізнес-профілю організації, система через каскад директив v-if / v-else-if ізолює та монтує у DOM-дерево браузера відповідні високорівневі доменні модулі.

Для організації взаємодії між ізольованими компонентами навігації та центральною контентною зоною реалізовано асинхронний контролер зміни станів. Розроблена архітектура координації компонентів інтерфейсу через централізовані події та реактивні прапорці Pinia дозволяє повністю уникнути хаотичного перевантаження сторінок та дублювання коду. Вебзастосунок функціонує як монолітний клієнтський пакет, який гнучко трансформує свій внутрішній простір

під операційні потреби конкретного тенанта, забезпечуючи високу швидкість відгуку інтерфейсу та безшовну роботу користувачів.

4.3.2 Графічний інтерфейс модуля притулку

Функціональний блок некомерційних притулків для тварин інтегрує в єдиному користувацькому просторі інструменти для ведення цифрових карток підопічних, моніторингу ветеринарної історії, координації щоденних графіків догляду та контролю складських запасів. Центром робочого простору притулку є сторінка «Наші підопічні». Інтерфейс надає працівнику гнучку систему відображення бази даних тварин за допомогою перемикача режимів. Перший режим представляє структуровану таблицю, оптимізовану для швидкого сканування великих масивів даних, сортування та масового аналізу характеристик. Другий режим активує візуальну сітку карток, де кожна картка містить фотографію тварини, завантажену з хмари, її кличку, вік та вид, що забезпечує наочність при повсякденній роботі (рис. 4.4). Панель фільтрації дозволяє здійснювати миттєвий текстовий пошук, а також вибіркову фільтрацію за біологічним видом чи статусом публічності. Керівник має можливість прямо з інтерфейсу картки за допомогою інтерактивного перемикача керувати прапорцем is_public, що автоматично синхронізує відображення тварини на зовнішньому сайті-візитці притулку.

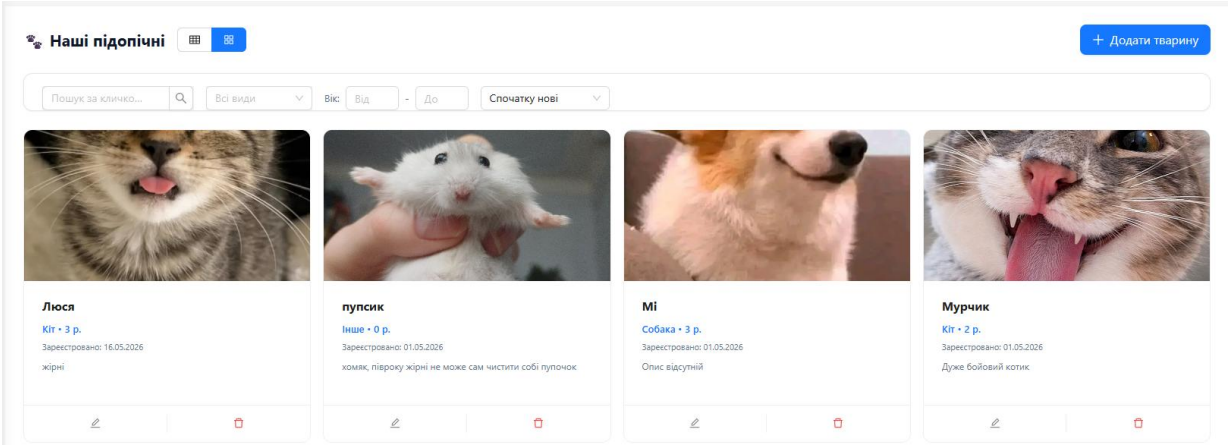

Рисунок 4.4 – Режим відображенян карток

Для створення нових та редагування наявних карток використовується модальне вікно (рис. 4.5). Воно реалізоване на базі компонента a-modal і містить

вертикальну форму валідації. Окрім стандартних текстових полів та селекторів виду, у вікно вбудовано кастомний завантажувач медіафайлів a-upload. Цей компонент перехоплює бінарний файл, візуалізує індикатор завантаження і через асинхронний тригер взаємодіє з ендпоінтом шлюзу, передаючи об'єкт FormData. Отриманий від Cloudinary URL-рядок автоматично мапується у поле форми та виводиться як прев'ю майбутньої аватарки тварини.

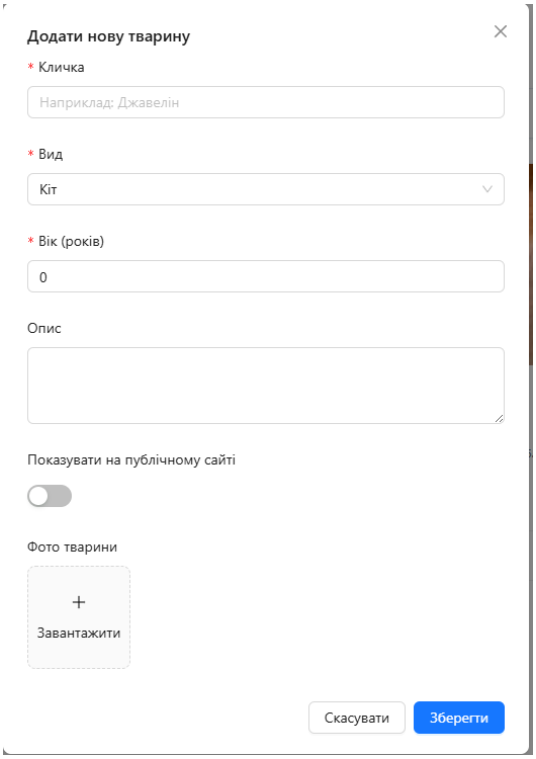

Рисунок 4.5 – Модальне вікно додавання тварини

Рівень ветеринарного контролю та медичного супроводу представлений сторінкою «Медичний журнал» та експертним інтерфейсом медичної картки (рис. 4.6). Журнал відображає перелік пацієнтів із можливостями пошуку за кличкою та хронологічного сортування. При обранні конкретної тварини розгортається широкоформатне модальне вікно, простір якого розділено на два логічні контури.

Ліва інфраструктурна панель – відображає персистентний профіль пацієнта, його фотографію, вид та вік. Ключовим інтерактивним елементом тут виступає безрамковий селектор статусу здоров'я тварини. Зміна значення у дропдауні (наприклад, перемикання зі статусу «Здоровий» на «Карантин» чи «На лікуванні») миттєво змінює колірне кодування картки за допомогою динамічних класів CSS та

відправляє фоновий PUT-запит до мікросервісу для оновлення даталогічного стану сутності.

Права контентна зона – побудована на базі вкладок a-tabs. Перша вкладка містить вертикальний інтерактивний таймлайн медичної історії, де у зворотному хронологічному порядку виводяться картки оглядів, вакцинацій та операцій із зазначенням типу процедури, дати та детальних текстових нотаток лікаря. Друга вкладка надає ергономічну форму для миттєвого додавання нового медичного запису.

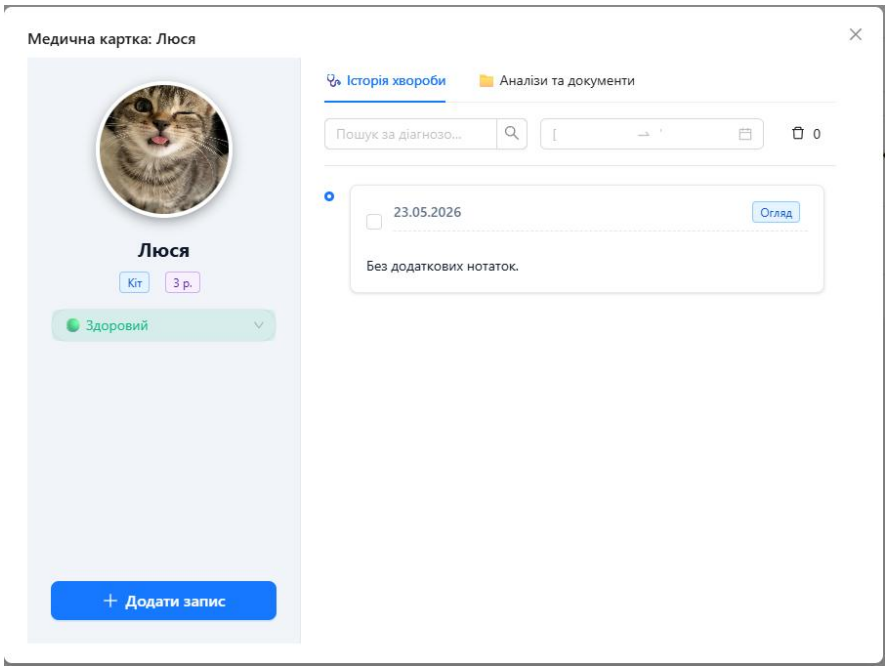

Рисунок 4.6 – Медична картка пацієнта

Координація щоденної діяльності персоналу здійснюється через модуль «Календар завдань» (рис. 4.7). Інтерфейс підтримує два режими представлення: повноформатний інтерактивний сітковий календар та лінійний операційний список на день. У режимі календаря кожна комірка дня динамічно рендерить масив бейджів, які відображають назви запланованих задач (годування, прийом ліків, вигул). Клікнувши на конкретний день, працівник переходить до списку завдань, де реалізовано патерн швидкого виконання: кожна задача забезпечена прапорцем. При встановленні прапорця інтерфейс реагує миттєво – текст завдання закреслюється, його прозорість знижується, а на бекенд надходить асинхронний запит для фіксації виконання транзакції.

Менеджмент матеріально-технічного забезпечення притулку покладено на сторінку «Склад та припаси» (рис. 4.7). Інтерфейс використовує систему тематичних вкладок для швидкої фільтрації ресурсів за категоріями (Корм, Ліки, Амуніція, Господарські товари). Основна зона екрана рендерить картки складських позицій. Для забезпечення предиктивного аналізу залишків у кожну картку вбудовано лінійний індикатор прогресу. Алгоритм обчислює відсоткове співвідношення поточної кількості припасів до цільового критичного мінімуму, заданого адміністратором. Якщо рівень залишків падає нижче встановленого порогу, картка динамічно отримує червоне колірне кодування.

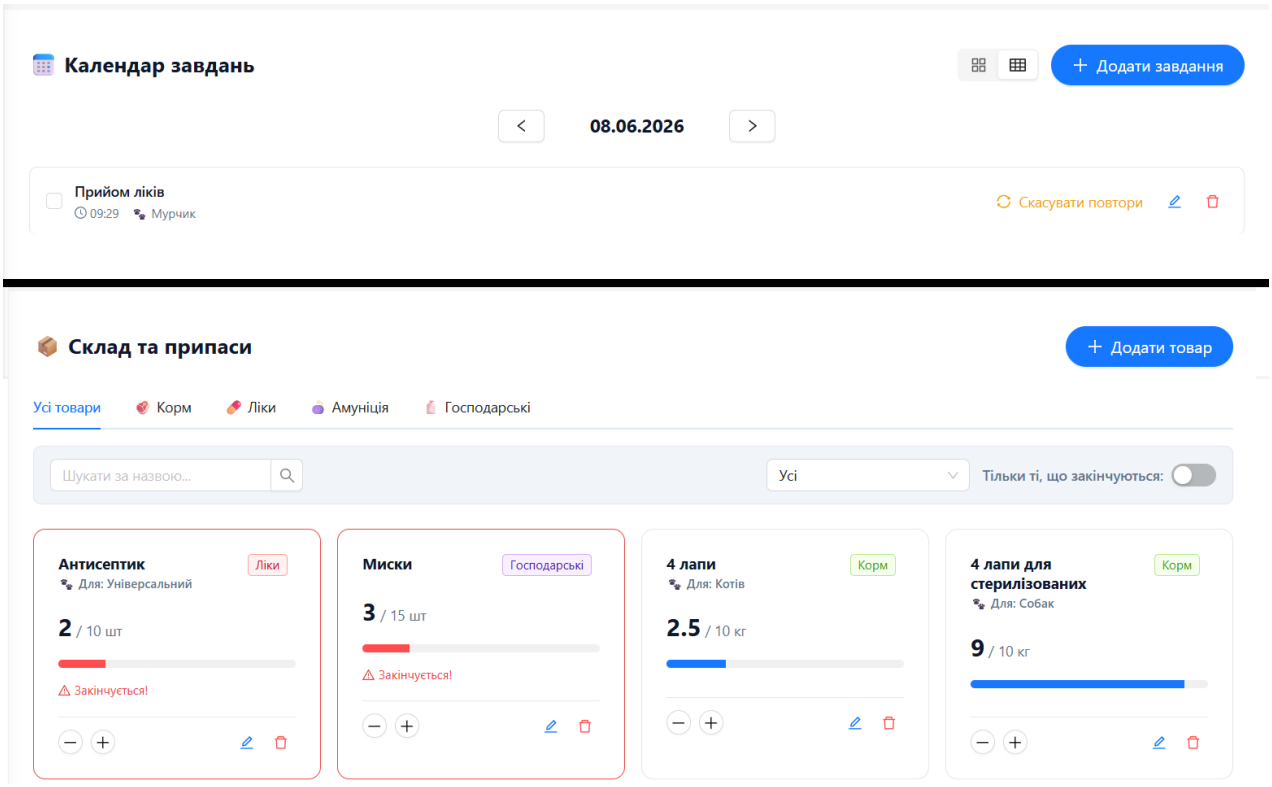

Рисунок 4.7 - Інтерфейс календаря щоденних завдань та управління складом

Загалом, спроєктований графічний інтерфейс модуля притулку забезпечує високий рівень ергономіки, централізує управління гетерогенними даними та ізолює логіку некомерційного закладу в межах єдиного, чутливого до контексту веб-інтерфейсу.

4.3.3 Інтерфейс комерційного модуля готелю

Комерційний домен SaaS-платформи, орієнтований на автоматизацію діяльності зооготелів, консолідує у межах єдиного клієнтського простору інструменти для фінансового обліку, моніторингу завантаженості номерного фонду, координації додаткових сервісів та запобігання конфліктам овербукінгу. Графічні модулі цього рівня спроєктовані з акцентом на прозорість транзакційних потоків, предиктивне планування заселення кімнат та мінімізацію людського фактора під час реєстрації клієнтів. Повні вихідні коди компонентів представлення готелю винесено у додатки до дипломної роботи.

Центральною консоллю управління комерційними процесами є сторінка «Управління бронюваннями» (рис. 4.8). У верхній частині розташовано операційну панель із селекторами сортування та фільтрації, які дозволяють миттєво групувати замовлення за логічним станом (Актуальне, Завершене, Скасоване) або часовим виміром (Найближчі заїзди, Спочатку нові). Реєстр бронювань рендериться у вигляді гнучкої сітки карток booking-card.

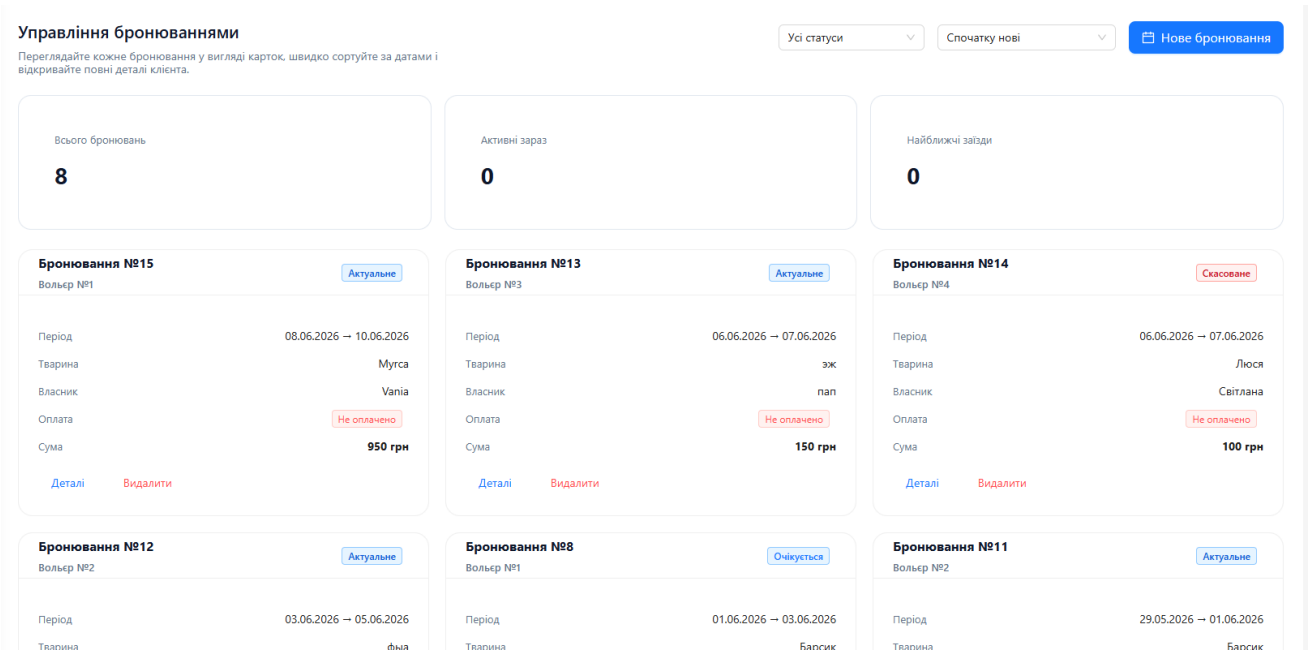

Рисунок 4.8 - Інтерфейс консолі управління бронюваннями

Головним інструментом фіксації комерційних угод є широкоформатне модальне вікно оформлення нового заїзду. Простір форми логічно розбитий на послідовні контури. Процес реєстрації тварини побудовано за принципом поетапного інтелектуального алгоритму.

Ідентифікація клієнта – працівник вводить номер телефону власника у пошукове поле a-input-search. Якщо клієнта знайдено в базі, поля ПІБ власника та кличка тварини заповнюються автоматично, мінімізуючи часові витрати на ручне введення.

Параметризація термінів та перевірка конфліктів – користувач вказує часовий інтервал проживання за допомогою двокалендарного компонента a-range-picker. Зміна масиву дат автоматично ініціює фоновий запит до сервера для вилучення переліку вольєрів, вільних саме у цей проміжок часу. Скисок вибору номерів динамічно оновлює свій вміст, повністю блокуючи можливість вибору вже зайнятої кімнати.

Калькуляція додаткових сервісів та бюджету – форма надає мультиселектор для підключення супутніх послуг. Інтерфейс реалізує патерн миттєвого прорахунку: реактивний наглядач відстежує будь-які зміни у полях обраного вольєра та масиву ідентифікаторів послуг, автоматично перераховуючи підсумкову вартість. Перед збереженням замовлення персонал бачить фінальний фінансовий підсумок транзакції у реальному часі.

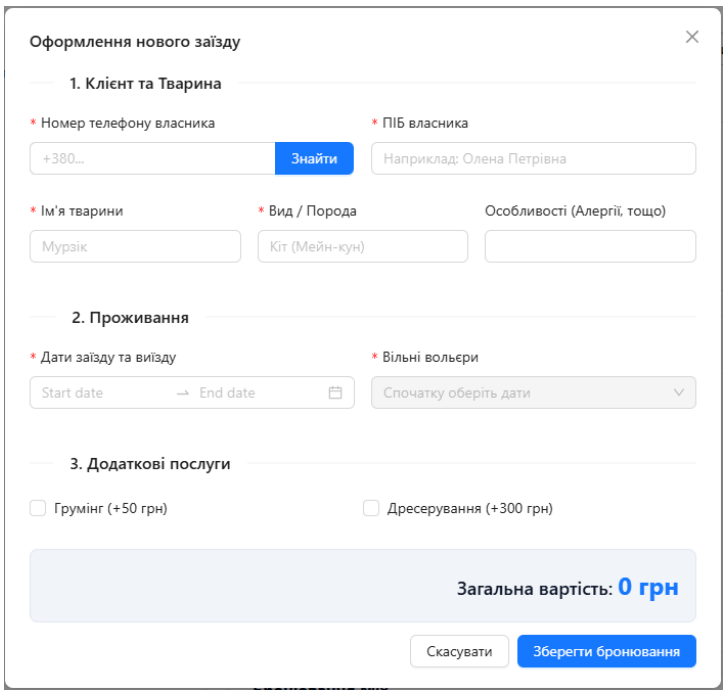

Рисунок 4.8 – Форма оформлення заїзду

Для управління матеріальними активами та прайс-листом готелю розроблено автономні сторінки «Фонд вольєрів» та «Додаткові послуги». Обидва модулі використовують сувору табличну розмітку a-table із підтримкою меж bordered. Якщо вольєр закривається на ремонт або санітарне обслуговування, його статус динамічно змінюється на «Виведений з експлуатації» із присвоєнням червоного індикатора, що автоматично вилучає його з алгоритмів доступності форми бронювання. У модулі прайс-листа реалізовано механізм безпечного архівації послуг: замість фізичного видалення запису з бази даних, що порушило б реляційні зв'язки минулих транзакцій, інтерфейс дозволяє перемістити послугу в архів.

Операційне управління щоденними завданнями догляду за готельними тваринами реалізовано так само як в модулі притулку. Але окрім стандартних представлень класичного календаря задач та лінійного списку нарядів для персоналу, сюди вбудовано спеціалізовану панель обробки зовнішніх онлайн-заявок від клієнтів (рис. 4.9). Ця панель слугує буферною зоною, де акумулюються запити, надіслані власниками тварин із відкритих публічних сайтів-візиток готелів. Кожна картка заявки містить анкетні дані, бажані дати та перелік послуг, а також забезпечена інтерактивними кнопками затвердження або відхилення. Працівник може одним кліком перетворити вхідну веб-заявку на повноцінне замовлення з автоматичним бронюванням вибраного вольєра.

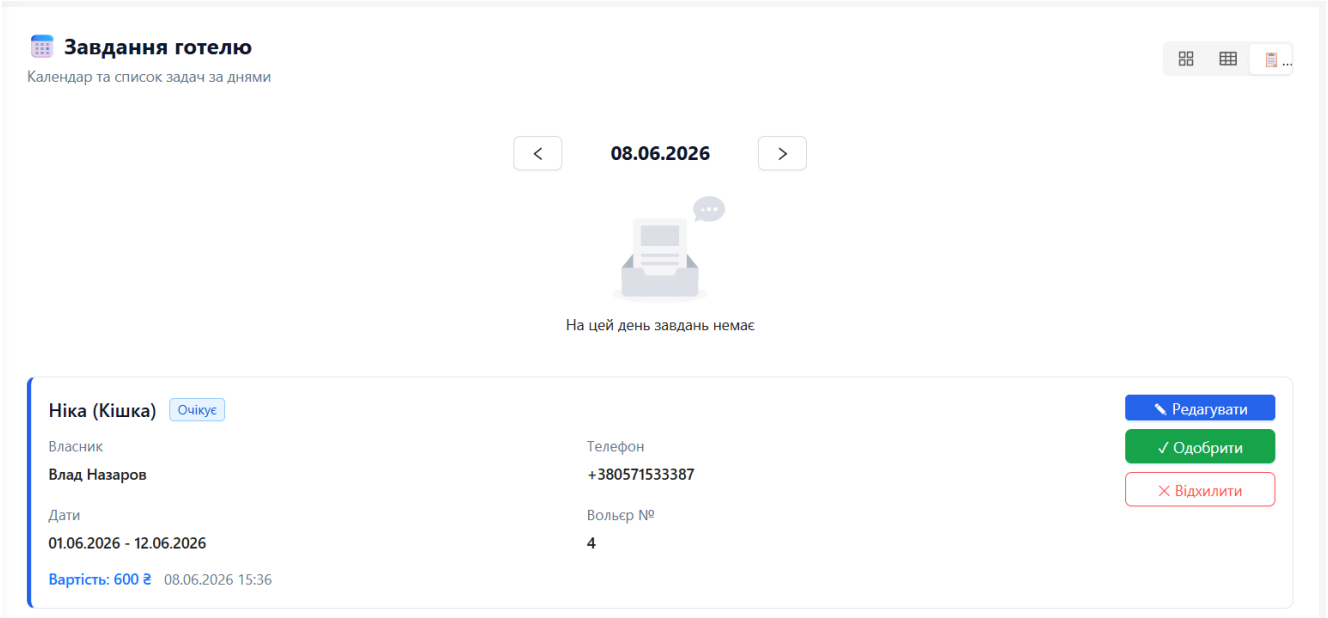

Рисунок 4.9 – Панель онлайн-заявок клаєнтів

Програмна гнучкість комерційного інтерфейсу досягається завдяки суворому відстеженню реактивних станів форми. Як приклад мінімального інфраструктурного коду, у лістингу нижче наведено архітектурний фрагмент логіки очищення та реактивної синхронізації внутрішнього стану форми бронювання готелю за допомогою вбудованого наглядача watch.

```
watch(() => props.open, (newVal) => {
  if (newVal) {
    Object.assign(form, {
      pet_id: null, owner_phone: '', owner_name: '', pet_name: '',
      species: '', description: '', dates: [], aviary_id: null, service_ids: []
    });
    availableAviaries.value = [];
    fetchServices();
    fetchHotelSettings();
  }
});
```

Впровадження описаних інтерфейсних рішень дозволило створити ергономічне та відмовостійке цифрове середовище для управління готельним бізнесом у сфері догляду за тваринами. Платформа успішно централізує комерційну інформацію, повністю унеможливлює помилки дублювання номерів та забезпечує безшовну обробку клієнтських запитів, суттєво підвищуючи операційну ефективність організацій-орендарів.

4.4 Тестування програмного забезпечення

Тестування програмного забезпечення є критично важливим етапом життєвого циклу розробки, який гарантує відповідність готового продукту початковим функціональним та нефункціональним вимогам. Зважаючи на розподілену мікросервісну архітектуру розробленої SaaS-платформи та суворі вимоги до ізоляції даних орендарів, стратегія тестування охоплювала перевірку програмних інтерфейсів, тестування механізмів безпеки та наскрізне функціональне тестування клієнтського застосунку.

Для верифікації серверної частини використовувалися можливості бекенд-фреймворку FastAPI, який автоматично генерує інтерактивну документацію за стандартом OpenAPI (Swagger UI), а також інструментарій платформи Postman. Це дозволило проводити модульне тестування кожного ендпоінту мікросервісів

безпосередньо через графічний інтерфейс. Під час тестування перевірялася коректність обробки вхідних JSON-пакетів та правильність повернення HTTP-кодів статусу (наприклад, 200 OK при успішній транзакції). Результат успішного модульного тестування мікросервісу обліку тварин наведено на рис. 4.10.

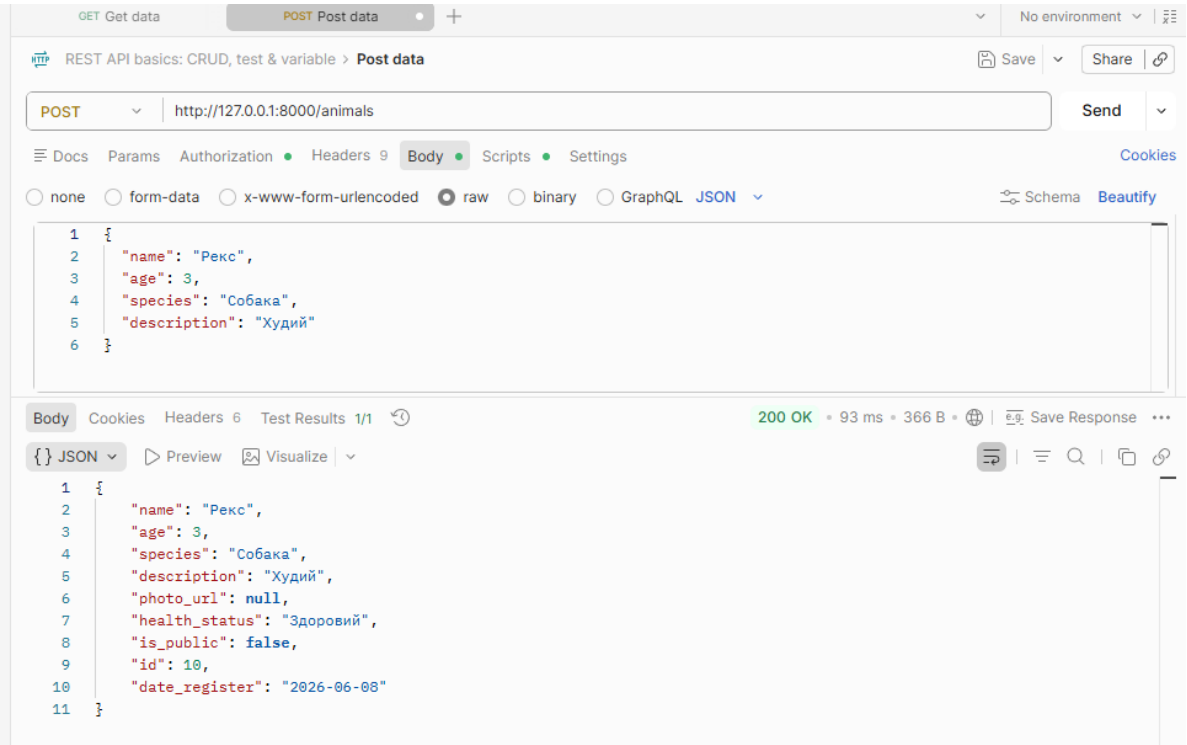

Рисунок 4.10 – Результат модульного тестування

Особливу увагу було приділено інтеграційному тестуванню центрального шлюзу API Gateway. Перевірка здійснювалася шляхом симуляції HTTP-запитів із різними станами безпекових маркерів у середовищі Postman. Запити без заголовка Authorization або з підробленим підписом JWT миттєво відхилялися шлюзом із поверненням помилки 401 Unauthorized (рис. 4.11), що підтвердило надійність захисту внутрішньої Docker-мережі від зовнішніх атак.

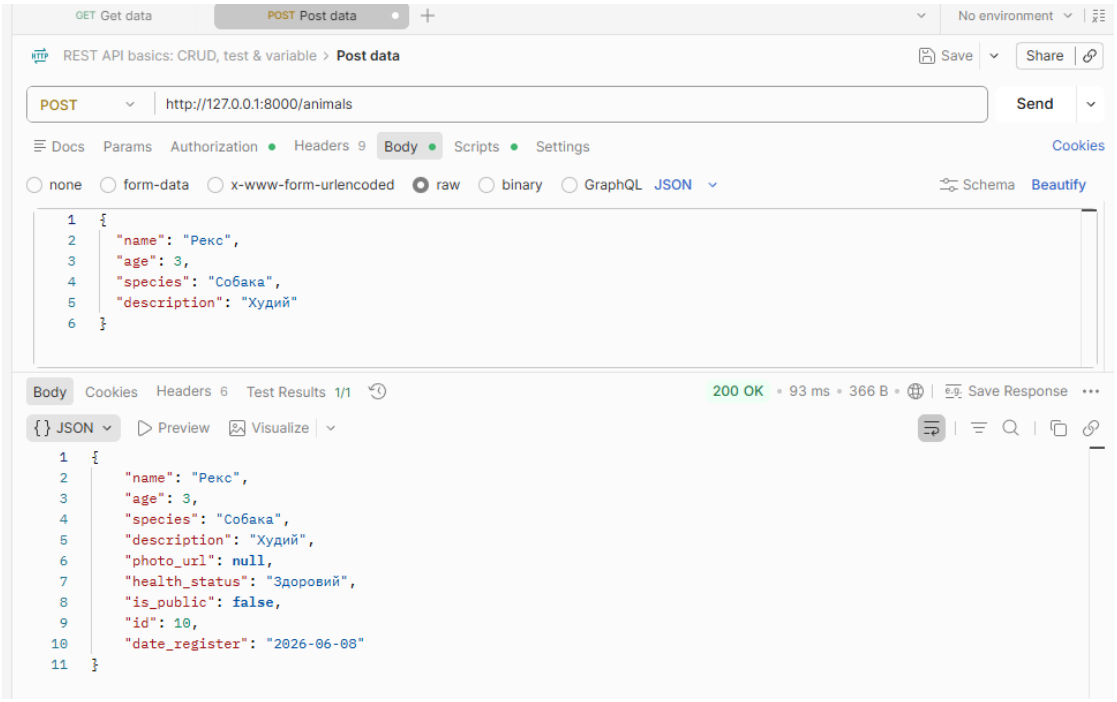

Рисунок 4.11 – Перевірка відпрайцювання механізму JWT-автентифікації

Для забезпечення високої надійності клієнтської частини та запобігання появі регресійних дефектів під час подальшого масштабування платформи, в архітектуру проєкту було інтегровано автоматизоване середовище модульного тестування. Оскільки клієнтський застосунок базується на екосистемі Vue 3 та збирачі Vite, основним інструментом для написання та виконання тестів було обрано сучасний фреймворк Vitest у поєднанні з офіційною бібліотекою Vue Test Utils.

Стратегія модульного тестування фронтенду фокусується на двох ключових напрямках: перевірка ізольованої бізнес-логіки (зокрема, реактивних сховищ Pinia) та перевірка поведінки графічних UI-компонентів у віртуальному DOM.

Першим рівнем є тестування глобального стану системи без монтування графічного інтерфейсу. Для цього перевіряються обчислювані властивості та дії сховищ. Наприклад, для модуля useAuthStore розроблено набір тестових сценаріїв, які імітують зміну типу організації і перевіряють, чи коректно відпрацьовують аналітичні прапорці isShelter та isHotel. Результат такого тестування представлено на рисунку 4.12

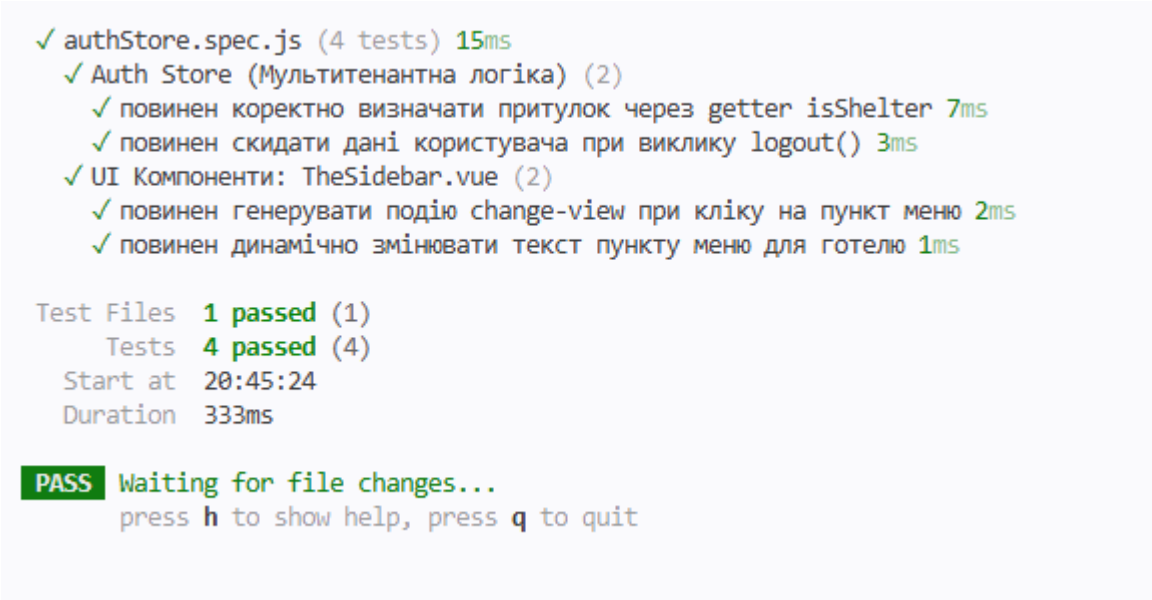

Рисунок 4.12 – Результат тестування через Vitest

Другим етапом є компонентне тестування самих SFC-файлів. За допомогою Vue Test Utils компоненти ізольовано монтуються у віртуальне середовище. Написані тести перевіряють коректність рендерингу умовних блоків v-if, відпрацювання обробників кліків (@click) та емісію користувацьких подій (emit) до батьківських інстанцій.

Наприклад, успішно протестовано інфраструктурний компонент бокового меню (рис. 4.12). Тестовий сценарій програмно імітує клік по пункту меню «Прайс-лист» і за допомогою методу wrapper.emitted() підтверджує, що компонент дійсно згенерував подію change-view із правильним строковим параметром services.

Впровадження практики Test-Driven Development та регулярний запуск модульних тестів дозволяють оперативно виявляти логічні помилки на етапі розробки, що суттєво підвищує загальну відмовостійкість графічного інтерфейсу платформи перед її розгортанням у виробничому середовищі.

Висновки до розділу 4

У четвертому розділі дипломної роботи було здійснено повний цикл практичної реалізації та тестування спроєктованої хмарної SaaS-платформи. Процес розробки було декомпозовано на створення серверної інфраструктури,

програмування клієнтського вебзастосунку та налаштування інтеграційних механізмів.

На рівні бекенду успішно імплементовано мікросервісну архітектуру з використанням асинхронного фреймворку FastAPI. Завдяки розробці централізованого шлюзу API Gateway забезпечено єдину точку доступу до сервісів та надійну маршрутизацію трафіку. Практично реалізовано архітектурний патерн мультитенантності у базі даних PostgreSQL, що гарантує абсолютну криптографічну та транзакційну ізоляцію інформації кожної організації-орендаря.

Клієнтську частину платформи розроблено у вигляді односторінкового застосунку на базі фреймворку Vue 3 з використанням архітектури Composition API. Завдяки інтеграції глобального сховища стану Pinia досягнуто реактивної адаптації графічного інтерфейсу під специфіку бізнес-домену поточного користувача. Для оптимізації зберігання та доставки медіаконтенту, зокрема фотографій тварин, успішно виконано інтеграцію із зовнішнім хмарним сховищем Cloudinary.

Завершальним етапом стала комплексна перевірка працездатності програмного продукту. Проведене модульне та інтеграційне тестування серверних ендпоінтів за допомогою інструментарію Postman, а також автоматизоване тестування логіки клієнтських компонентів у середовищі Vitest підтвердили стабільність, відмовостійкість та повну відповідність розробленої системи початковим вимогам. Платформа готова до дослідної експлуатації та впровадження у реальні бізнес-процеси цільових організацій.

ВИСНОВКИ

У кваліфікаційній роботі бакалавра успішно розроблено хмарну багатокористувацьку SaaS-платформу для автоматизації операційної діяльності закладів опіки над тваринами, а також повністю вирішено актуальну науково-практичну задачу та реалізовано всі поставлені пункти завдання, а саме:

1) проведено ґрунтовний системний аналіз предметної області та існуючих програмних аналогів для автоматизації роботи притулків та зооготелів, що дозволило виявити критичні точки в управлінні ресурсами закладів;

2) визначено комплекс функціональних і жорстких нефункціональних вимог до системи, а також розроблено розширені сценарії використання для різних рольових моделей користувачів;

3) спроєктовано гнучку та стійку до навантажень архітектуру системи, а також обґрунтовано доцільність застосування мікросервісного підходу з розподілом загальної бізнес-логіки на автономні сервіси авторизації, обліку тварин та бронювання;

4) розроблено нормалізовану структуру реляційної бази даних для безпечного зберігання інформації про організації, тварин, медичні послуги, складські залишки та журнали бронювань;

5) програмно реалізовано асинхронну серверну частину мікросервісів із використанням технології REST API та інструментів системної контейнеризації;

6) реалізовано клієнтську частину у вигляді високореактивного вебзастосунку для адміністраторів, менеджерів та лінійного персоналу організацій;

7) проведено всебічне тестування розробленої хмарної платформи та оцінено її загальну операційну й технологічну ефективність.

Науково-методичне обґрунтування архітектурних рішень, підкріплене аналізом сучасних публікацій у профільних фахових виданнях, дозволило обрати та адаптувати найбільш ефективну стратегію мультитенантності – модель Schema-per-Tenant на базі СКБД PostgreSQL. Це рішення гарантує сувору логічну та фізичну ізоляцію інформації кожного окремого організації-орендаря (тенанта) платформи у межах єдиного екземпляра бази даних, забезпечуючи найвищий

рівень безпеки конфіденційних даних. Вибір мікросервісного архітектурного підходу у поєднанні з технологіями FastAPI на серверній частині та Vue 3 на клієнтській частині дозволив спроєктувати систему, що володіє стійкістю до високих паралельних навантажень, можливістю точкового лінійного масштабування та високою швидкістю відгуку екранних форм користувацького інтерфейсу.

Професійне інженерне проєктування програмного продукту підкріплено розробкою детальної специфікації вимог згідно з міжнародними стандартами розробки ПЗ. Сформований комплекс UML-моделей дозволив візуалізувати внутрішню компонентну структуру сервісів та життєвий цикл обробки HTTP-запитів через центральний шлюз безпеки. Математичне моделювання алгоритмів валідації часових інтервалів формалізувало логіку запобігання конфліктам овербукінгу при бронюванні вольєрів.

Етап програмної реалізації та тестування довів абсолютну дієздатність закладених архітектурних рішень. Реалізований API Gateway успішно забезпечує динамічну контекстну маршрутизацію на основі JWT-маркерів. Клієнтський вебзастосунок отримав ергономічний інтерфейс, який у режимі реального часу трансформує свій функціонал під потреби конкретного тенанта, уникаючи надлишковості графічних елементів. Інтеграція зовнішнього хмарного сховища Cloudinary дозволила оптимізувати обробку медіафайлів та зняти навантаження з основної бази даних.

Загалом, результати виконаних етапів дослідження, проєктування, розробки та тестування повністю підтвердили високу практичну цінність створеної SaaS-платформи для цифровізації сфери догляду за тваринами. Спроєктований програмний продукт відповідає сучасним критеріям інженерії програмного забезпечення, не містить критичних дефектів алгоритмічної логіки та є повністю готовим до впровадження у реальну операційну діяльність профільних закладів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. PetPoint. URL: https://www.24pet.com/products/petpoint (Accessed: 23.05.2026).

2. Shelterluv. URL: https://www.shelterluv.com/ (last acccess: 23.05.2026).

3. RevelationPets. URL: https://www.revelationpets.com/ (Accessed: 23.05.2026).

4. Tiangolo (Sebastián Ramírez). FastAPI Documentation. URL: https://fastapi.tiangolo.com/ (Accessed: 23.05.2026).

5. OAI. OpenAPI Specification 3.1.0. URL: https://spec.openapis.org/oas/v3.1.0 (Accessed: 23.05.2026).

6. Vue.js Core Team. Vue.js 3 Documentation URL: https://vuejs.org/ (Accessed: 23.05.2026).

7. Pinia Documentation. The intuitive store for Vue.js. URL: https://pinia.vuejs.org/ (Accessed: 23.05.2026).

8. PostgreSQL Global Development Group. PostgreSQL 16 Documentation: 5.9. Schemas. URL: https://www.postgresql.org/docs/current/ddl-schemas.html (Accessed: 23.05.2026).

9. Docker Inc. Docker Documentation: Containerization overview. URL: https://docs.docker.com/ (Accessed: 23.05.2026).

10. Cloudinary. Image and Video API Documentation. URL: https://cloudinary.com/documentation (Accessed: 23.05.2026).

11. Юрченко М. Ю., Негоденко О. В., Довженко Т. П., Дзядович О. С. Застосування моделі хмарної піраміди та методу SaaS для підвищення рівня захищеності великих даних. *Зв'язок*. 2023. № 5. С. 48–52. DOI: 10.31673/2412-9070.2023.054751.

12. Fowler M., Lewis J. Microservices. *MartinFowler.com*. 2014. URL: https://martinfowler.com/articles/microservices.html (Accessed: 23.05.2026).

13. Кошелюк В. А., Корень В. В., Тимчук В. В. Архітектурні моделі мультитенантності в сучасних SaaS-системах: порівняльний аналіз та методологія

вибору. *Комп'ютерно-інтегровані технології: освіта, наука, виробництво*. 2025. № 61. С. 98–103. DOI: 10.36910/6775-2524-0560-2025-61-14.

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

15. Internet Engineering Task Force (IETF). RFC 7519: JSON Web Token (JWT). URL: https://datatracker.ietf.org/doc/html/rfc7519 (Accessed: 23.05.2026).

16. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd Edition. O'Reilly Media, 2021. 614 p.

17. Richardson C. Microservices Patterns: With examples in Java. Manning Publications, 2018. 520 p.

18. Корчев Є., Божуха Л. Проєктування та розробка інформаційної системи управління замовленнями у закладах харчування з кросплатформеним доступом. *Інформаційні технології в металургії та машинобудуванні*. 2025. DOI: 10.34185/1991-7848.itmm.2025.01.101.

19. Tiangolo (Sebastián Ramírez). FastAPI Documentation: Concurrency and async / await. URL: https://fastapi.tiangolo.com/async/ (Accessed: 23.05.2026).

20. Pydantic Documentation. Data validation using Python type hints. URL: https://docs.pydantic.dev/latest/ (Accessed: 23.05.2026).

21. Vue.js Core Team. Vue.js 3 Documentation: Composition API FAQ. URL: https://vuejs.org/guide/extras/composition-api-faq.html (Accessed: 23.05.2026).