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

Допущено до захисту
Завідувач кафедри комп’ютерних наук,
доктор техн. наук, професор
Андрій БОНДАРЧУК
« 01 » червня 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

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

Тема: ІНФОРМАЦІЙНА ВЕБПЛАТФОРМА ПІДТРИМКИ ВЛАСНИКІВ ДОМАШНІХ ТВАРИН

Виконала
студентка групи ІНб-2-22-4.0д.
Рудавіна Катерина Дмитрівна

_____
(підпис)
Науковий керівник
кандидат технічних наук, доцент
Мельник І.Ю.

_____
(підпис)

Київ - 2026

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

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

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА
«Інформаційна вебплатформа підтримки власників домашніх тварин»

Виконавець - студентка групи ІНб-2-22-4.0д,
спеціальності 122 Комп'ютерних наук,
освітньої програми 122.00.01 Інформатика
Рудавіна Катерина Дмитрівна

1. Вихідні дані до роботи: Сучасні підходи до проєктування систем із використанням стеку React (Vite) для фронтенду та Node.js/Express для серверної частини. Вимоги до побудови масштабованих SPA-застосунків із використанням реляційної бази даних SQLite та Prisma ORM для забезпечення цілісності даних. Проєкт реалізується з урахуванням стандартів доступності WCAG 2.1, підтримки мультимовності та вимог Закону України «Про захист персональних даних».
2. Основними завданнями бакалаврської роботи є: Аналіз предметної галузі та порівняльне дослідження існуючих платформ для визначення оптимального набору функцій підтримки власників тварин. Дослідження методів проєктування компонентно-орієнтованих інтерфейсів та архітектури «клієнт-сервер» на базі REST API для забезпечення високої швидкодії вебплатформи. Визначення вимог до функціонування модулів електронної медичної картки тварини, системи геолокаційних оголошень та механізмів сповіщень. Проєктування структури бази даних та розробка рольової моделі доступу для розмежування прав користувачів. Розробка адаптивного користувацького інтерфейсу вебплатформи. Програмна реалізація бізнес-логіки сервісів, впровадження механізмів безпеки JWT та проведення верифікації функціоналу шляхом автоматизованого тестування.
3. Пояснювальна записка: Обсяг - приблизно 90 стор. формату А4 комп'ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.
4. Графічні матеріали: Комп'ютерна презентація доповіді. Додаток А.
5. Додатки: Додаток А - графічні зображення розробленої платформи.
6. Строки подання роботи на кафедру: « 01 » червня 2026 р..

Науковий керівник:
кандидат технічних наук
доцент
_____ Мельник І.Ю.
_____ дата

Виконавець:
студентка групи
ІНб-2-22-4 Од
_____ Рудавіна К.Д.
_____ дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк виконання етапу роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	01.10.2025 15.10.2025	Виконано
2	Аналіз предметної галузі, постановка задачі та формування вимог до системи	16.10.2025 20.11.2025	Виконано
3	Огляд сучасних рішень, аналіз літературних джерел та вибір оптимальних технологій для реалізації системи	21.11.2025 20.01.2026	Виконано
4	Розробка архітектури програмного забезпечення та проектування основних модулів системи.	21.01.2026 15.02.2026	Виконано
5	Інтеграція та тестування основних функціональних модулів	16.02.2026 28.02.2026	Виконано
6.	Проведення автоматизованого тестування, аналіз результатів, виявлення та усунення помилок	01.03.2026 04.03.2026	Виконано
7.	Створення звіту з кваліфікаційної роботи	14.03.2026 04.04.2026	Виконано
8.	Подання роботи на перевірку, проходження антиплагіату, внесення правок керівника	10.04.2026 01.05.2026	Виконано
9.	Підготовка презентаційних матеріалів, тексту доповіді та захист кваліфікаційної роботи	16.05.2026 15.06.2026	

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

Індивідуальний план склала
Рудавіна К.Д.

РЕФЕРАТ

Кваліфікаційна робота: 109 с., 25 рис., 5 табл., 25 джерел.

Актуальність: полягає у гострій потребі українського ринку в єдиному інтегрованому та локалізованому цифровому рішенні для підтримки власників домашніх тварин. Наявні міжнародні аналоги (такі як Petkey чи PetDesk) банально не враховують нашої регіональної та мовної специфіки. В свою чергу, в складних умовах воєнного стану критично зросла необхідність в інструментах саме оперативного реагування. Зокрема, йдеться про швидкий пошук численних загублених улюбленців через систему Lost&Found, зручну координацію місцевих волонтерів та безперебійний доступ до актуальних ветеринарних послуг.

Об’єкт дослідження: процес надання інформаційної підтримки для власників домашніх тварин із безпосереднім використанням сучасних вебтехнологій.

Предмет дослідження: інформаційна вебплатформа підтримки власників домашніх тварин, яка технічно забезпечує збереження та обробку даних про улюбленців, соціальну взаємодію користувачів і швидкий доступ до корисної сервісної інформації.

Мета роботи: детальне проєктування та подальша практична реалізація інформаційної вебплатформи підтримки власників домашніх тварин, яка б ефективно забезпечувала зручний доступ до необхідної інформації, локальних сервісів та інструментів для комунікації між користувачами.

Завдання дослідження:

1. проведено комплексний аналіз поточної предметної галузі, а також досліджено функціонал вже існуючих профільних вебплатформ;
2. визначено основні системні та бізнес-вимоги до функціонування майбутньої інформаційної вебплатформи;
3. спроєктовано загальну архітектурну структуру застосунку та логічно виділено його базові функціональні модулі;

4. розроблено зручний, сучасний та адаптивний користувацький інтерфейс;

5. фізично реалізовано програмну серверну та клієнтську частини вебплатформи;

6. виконано ретельне тестування усіх основних функцій розробленої системи.

Отримані результати: Безпосередньо у кваліфікаційній роботі на практиці було розроблено та успішно реалізовано всю програмну частину платформи. Зокрема, у систему повністю інтегровано надійний модуль автентифікації користувачів, функціональний "профіль домашнього улюбленця", а також здійснено прив'язку зовнішніх картографічних сервісів для роботи інтерактивної карти місць. Окрім цього, розроблено зручну систему сповіщень та повноцінний соціальний форум.

Методи дослідження: системний аналіз предметної галузі, порівняльний аналіз вже існуючих вебсервісів, моделювання загальної структури платформи, проєктування користувацького інтерфейсу, а також безпосередня програмна реалізація та подальше автоматизоване тестування створеного вебзастосунку. Практичне значення: створено готовий до розгортання веб-сервіс, який може застосовуватися в освітніх закладах, невеликих організаціях чи для індивідуальних потреб - без залежності від комерційних хмарних платформ, із повним контролем користувача над власними даними.

Практичне значення: **:** практичне значення одержаних результатів полягає у створенні готового до розгортання веб-застосунку «Інформаційна веб-платформа підтримки власників домашніх тварин », який інтегрує в межах єдиного інтерфейсу базові повсякденні сценарії догляду за твариною та може бути безпосередньо використаний у реальній експлуатації. Розроблена платформа забезпечує комплексне вирішення основних задач власника домашньої тварини: ведення електронного профілю улюбленця з історією щеплень і медичних подій, отримання своєчасних сповіщень-нагадувань, пошук pet-friendly локацій на інтерактивній карті, обмін досвідому спільноті через тематичний форум, а також

публікацію та пошук оголошень про загублених чи знайдених тварин у модулі Lost&Found.

Розроблений програмний продукт реалізовано як повноцінний SPA-застосунок на базі сучасного стеку технологій(React, Node.js, Express, Prisma ORM, SQLite з можливістю безшовної міграції на PostreSQL), що забезпечує його високу масштабованість, кросплатформність і незалежність від комерційних хмарних платформ. Впроваджена рольова модель доступу (RBAC) з трьома рівнями привілеїв (користувач, модератор, адміністратор) та механізми безпеки на основі JWT- токенів і bcrypt-хешування паролів гарантують надійний захист персональних даних користувачів відповідно до вимог Закону України «Про захист персональних даних».

Результати кваліфікаційної роботи можуть бути використані: ветеринарними клініками та зоозахисними організаціями як інструмент комунікації з власниками тварин, волонтерськими ініціативами (UA-animals, KARG та ін.) для координації допомоги і пошуку загублених тварин в умовах воєнного стану, індивідуальними користувачами для систематизації догляду за домашніми улюбленцями. Окремі модулі платформи (геомодуль на базі Leaflet, система автентифікації, механізм модерації UGC-контенту) можуть бути перенесені у суміжні предметні галузі. Крім того, програмний код, архітектурні рішення та модель бази даних можуть слугувати навчально-методичним матеріалом у дисциплінах із веб-програмування, проєктування інформаційних систем та баз даних для студентів спеціальності 122 «Комп`ютерні науки».

ПЕРЕЛІК ВИКОРИСТАНИХ СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

API - Application Programming Interface - програмний інтерфейс прикладного застосунку

BBox - Bounding Box - обмежувальна прямокутна область видимої частини карти

CRUD - Create, Read, Update, Delete - базові операції створення, читання, оновлення та видалення даних

ORM -Object-Relational Mapping - об’єктно-реляційне відображення

OS - Operating System - операційна система

RBAC - Role-Based Access Control - рольова модель розмежування доступу

REST - Representational State Transfer - архітектурний стиль побудови вебсервісів

SPA - Single Page Application -односторінковий вебзастосунок

WCAG - Web Content Accessibility Guidelines - рекомендації щодо доступності вебконтенту

ЗМІСТ

ВСТУП

У сучасних умовах кількість домашніх тварин постійно зростає. Разом із цим зростає і потреба власників у швидкому доступі до достовірної інформації, різних сервісів та інструментів, які пов’язані з доглядом за твариною. На сьогодні питання обліку даних про тварин, контролю ветеринарних заходів, пошуку спеціалізованих закладів, а також взаємодії з іншими власниками, часто вирішуються за допомогою різних окремих ресурсів. Через це користувачу доводиться користуватися одразу декількома сервісами. Це, в свою чергу, знижує зручність їх використання.

Актуальність даної кваліфікаційної роботи полягає в тому, що на ринку України існує потреба в інтегрованому та локалізованому рішенні для підтримки власників домашніх тварин. На сьогодні вже існують міжнародні сервіси, наприклад Petkey та PetDesk. Однак вони не завжди враховують локальні мовні, регіональні та організаційні особливості, які є важливими саме для українських користувачів. В умовах війни дана проблема стала ще більш відчутною. Через постійні обстріли, переміщення населення та збільшення кількості загублених і безпритульних тварин, особливо в прифронтових районах, зростає потреба в оперативних інструментах для розміщення оголошень загубився/знайшовся (Lost&Found), координації волонтерської допомоги та швидкого доступу до місцевих ветеринарних послуг.

Також слід зазначити, що в Україні вже діє велика кількість організацій та ініціатив, які займаються допомогою тваринам. До них можна віднести UAnimals, «Зоопатруль Україна», Kyiv Animal Rescue Group (KARG), Wild Animals Rescue Center, Animal House, UPAW, а також волонтерські команди, зокрема VETMARKET Pluriton. Інтеграція можливостей взаємодії з подібними структурами у локалізовану вебплатформу дає змогу підвищити оперативність допомоги. Крім того, це підвищує і загальну ефективність сервісу для українських користувачів.

Мета даної кваліфікаційної роботи полягає у проєктуванні та реалізації інформаційної вебплатформи підтримки власників домашніх тварин. Дана

вебплатформа повинна забезпечувати зберігання та обробку інформації про тварин, доступ до сервісних даних, а також можливість взаємодії користувачів між собою.

Для досягнення поставленої мети в роботі необхідно вирішити наступні завдання:

1. провести аналіз предметної галузі та існуючих вебплатформ;
2. визначити основні вимоги до функціонування інформаційної вебплатформи;
3. спроєктувати її структуру та основні модулі; розробити користувацький інтерфейс;
4. реалізувати програмну частину вебплатформи;
5. виконати тестування основних функцій розробленої системи.

Об’єктом дослідження є процес інформаційної підтримки власників домашніх тварин із використанням вебтехнологій.

Предметом дослідження є інформаційна вебплатформа підтримки власників домашніх тварин та її функціональні можливості

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Роль інформаційних технологій у сфері догляду за домашніми тваринами.

Предметна область даної роботи пов’язана з утриманням домашніх тварин у міському середовищі. Вона охоплює чималу сукупність інформаційних процесів та різноманітних взаємодій між власником і ветеринарними закладами. Сюди також відносяться і сервісні організації, наприклад, магазини зоотоварів, грумінг, готелі або сервіси вигулу. Крім того, важливу роль відіграє локальна спільнота власників.

На практиці потреби власника не обмежуються лише рідкісними візитами до лікаря. Значну частину займають щоденні рутинні задачі. Це годування, щоденний вигул та контроль загального стану тварини. Також є періодичні події, тобто щеплення, обробка від паразитів чи планові огляди. В свою чергу, часто трапляються і непередбачувані ситуації. Наприклад, зникнення тварини, потреба в екстреній консультації або необхідність швидко знайти найближчу клініку.

Важливою особливістю цієї предметної області є те, що в ній існують як комерційні, так і публічні муніципальні платформи. Для прикладу можна взяти місто Київ. Там функціонує міський ресурс «Реєстр домашніх тварин». Він підтримує питання адопції та публікації оголошень про загублених чи знайдених тварин, які обов'язково проходять модерацію. Також там містяться довідкові правила утримання тварин [13]. іВ той же час паралельно існують і приватні платформи. Вони призначені переважно для тимчасового догляду за тваринами під час від’їзду власника. Хорошим прикладом є сервіс TrustedHousesitters, який організовує взаємодію між власниками та доглядальниками у форматі спільноти або маркетплейсу.

З позиції обробки інформації в межах даної предметної області виділено наступні типові сутності та дані:

– ідентифікаційні дані тварини: кличка, вид, порода, стать, вік, особливі прикмети та фотографії;

– медичні дані: інформація про щеплення, хронологія візитів, рекомендації лікаря та алергії. Також сюди входять нагадування про майбутні процедури;

– події та рутини: календар подій, повторювані нагадування та збережена історія їх виконання;

– геопросторові дані: координати клінік і магазинів, маршрути, а також інформація про відстань і доступність;

– комунікаційні дані: обговорення на платформі (форум), опубліковані питання і відповіді, коментарі користувачів та дії модераторів;

– інцидентні дані модуля Lost&Found: оголошення про зниклих тварин, контактна інформація, місце, час інциденту і поточний статус оголошення.

Зараз для сфери цифрової підтримки власників тварин дуже характерною є проблема фрагментації. Тобто різні задачі зазвичай вирішуються в різних застосунках. Ветеринарні записи ведуться в одному місці, карта відкривається в іншому, а форум - взагалі окремо. Через це користувач витрачає набагато більше часу і часто змушений дублювати дані. Крім того, суттєво підвищується ризик пропустити щось важливе (наприклад, планове щеплення). Усе це має прямий вплив на якість догляду за твариною.

В даній роботі обрано напрям створення єдиної інформаційної вебплатформи. Вона об'єднує ключові задачі предметної області в одному зручному середовищі. Такий підхід є цілком доцільним. Він реалізований з огляду на те, що вебплатформа:

– забезпечує доступ з різних пристроїв. При цьому не потрібна прив’язка до операційної системи чи встановлення застосунку;

– дозволяє зберігати дані централізовано. Це стосується профілів тварин, розкладів, оголошень та самого форуму;

– підтримує швидкі канали сповіщення. Використовуються Web Push та email, щоб звести до мінімуму ризик пропуску важливих процедур;

– дає можливість використовувати геосервіси. Для пілотної версії обрано територію України (зокрема місто Київ), що допомагає швидко шукати корисні заклади;

– створює надійну базу для спілкування. Це реалізовано через вбудований форум та окремий модуль Lost&Found.

Отже, можна зробити висновок, що предметна область має виражену потребу в комплексному рішенні. Воно має бути в першу чергу орієнтоване на щоденні сценарії власника. В свою чергу, розробка інтегрованої вебплатформи є повністю обґрунтованою. Цей продукт має високу цінність для користувача. Крім того, його реалізація чудово підходить для дипломного проєкту (сюди входить аутентифікація, профілі тварин, геомодуль, система сповіщень, форум, Lost&Found та розподіл ролей на User, Moderator і Admin).

1.2 Аналіз існуючих веб-платформ та мобільних сервісів для власників тварин

У межах даного підрозділу проводиться аналіз наявних вебрішень і мобільних сервісів, які використовуються для підтримки власників домашніх тварин. Основна мета такого аналізу полягає у визначенні їх функціональних можливостей, а також у виявленні обмежень відносно концепції інтегрованої вебплатформи. При цьому важливо розглядати не тільки наявність певних функцій. Не менш важливими є якість їх реалізації, зручність використання та відповідність повсякденним сценаріям власника тварини.

Для порівняння вебрішень доцільно використати такі критерії:

– інтегрованість даних, тобто чи поєднуються профіль тварини, події, нагадування, оголошення та комунікація в одному середовищі;

– підтримка повсякденних сценаріїв, зокрема рутини, планових медичних подій та історії виконання;

– надійність і гнучкість сповіщень, а саме канали їх отримання, повтори та контроль доставки;

– релевантність геомодуля, тобто актуальність і практична користь пошуку закладів;

– комунікація та модерація, що включає форум, коментарі, скарги та рольову модель;

– безпека облікового запису, зокрема механізми автентифікації, відновлення доступу і захисту даних.

Підтримка власників домашніх тварин у цифровому середовищі зазвичай реалізується через набір окремих сервісів. Кожен із них покриває певний сценарій. Наприклад, ведення профілю тварини та її медичних даних, планування подій догляду, пошук потрібних послуг або установ, спілкування між користувачами, а також дії у випадку втрати тварини. З метою формування вимог до розроблюваного вебзастосунку було виконано огляд і порівняння функціональних можливостей ряду репрезентативних рішень. Вони відображають різні підходи до організації таких сценаріїв.

Перший клас рішень орієнтований на автоматизацію взаємодії ветеринарної установи з клієнтом. Прикладом такого сервісу є PetDesk. Дана платформа позиціонується як рішення для ветеринарних практик і декларує функції нагадувань, комунікацій, онлайн-запису та підтримки інтеграцій із системами керування практикою [2]. У межах такого підходу головною є ефективність роботи клініки. В свою чергу, повсякденні сценарії власника, наприклад форумна взаємодія, пошук корисних місць у місті або гейміфікація, не є центральними функціями такого сервісу.

Другий клас рішень формується навколо ідентифікації тварини та механізмів її повернення у разі втрати. До таких сервісів можна віднести Petkey Pro. Він орієнтований на притулки, ветеринарів і служби контролю тварин. В описі сервісу акцент зроблено на ідентифікації та активації мікрочипів, функції backtrack, швидкому контакті з власником, а також на можливості подати повідомлення про знайдену тварину [1]. Подібні рішення добре показують

важливість сценарію Lost&Found. Проте зазвичай вони не розглядають платформу як щоденний інструмент для власника тварини.

Окрему групу становлять рішення, які поєднують цифровий профіль тварини з певними сервісними функціями для власника. Прикладом є Animal ID. Даний сервіс описує цифровий профіль тварини, збереження документів і медичних даних, календар і нагадування, а також ідентифікацію через QR-паспорт та оповіщення у випадку сканування ідентифікатора [3]. Такий підхід уже є ближчим до задач довготривалого ведення даних про тварину. Однак він сам по собі не забезпечує розвинені механізми спільнотного обміну досвідом, форумної взаємодії або повноцінної модерації користувацького контенту.

Наступний напрям складають сервіси, які орієнтовані на ринок послуг догляду. Rover позиціонує себе як платформу для пошуку та бронювання догляду за тваринами. Це може бути перетримка, вигул, візити до тварини та інші послуги. Також сервіс використовує механізми комунікації між сторонами та формування довіри, зокрема відгуки, перевірки і гарантії [5]. У таких рішеннях основним центром є підбір виконавця та організація конкретної послуги. В той же час комплексна підтримка власника через інтеграцію профілю, подій, інформаційних модулів і спільноти не є головною метою такого продукту.

Окремо слід виділити сервіси, які використовують механізми залучення користувачів та елементи lifestyle-асистента. В даному випадку мова йде про гейміфікацію. Проєкт Dogiz, який зараз інтегрується з платформою Vetted, демонструє підхід, де щоденні дії власника перетворюються на активність у сервісі. На сторінці Dogiz/Vetted описано нарахування Care Coins за рутинні дії та прогулянки, використання бейджів і лідербордів, а також функцію Dr. Poop. Вона передбачає AI-аналіз стану за фото як приклад персоналізованих підказок у догляді [4]. Наявність такого класу рішень підтверджує практичну доцільність підтримки щоденних сценаріїв. Саме вони допомагають утримувати активність користувачів у сервісі.

Крім комерційних продуктів, у даній предметній області існують і муніципальні вебресурси. Вони підтримують суспільно значущі сценарії, зокрема

облік, інформування, публікацію оголошень про загублених або знайдених тварин, а також адопцію. Такі ресурси можуть передбачати і модерацію оголошень. Прикладом є київський «Реєстр домашніх тварин», який містить відповідні розділи та форми взаємодії з користувачами [6]. Також існують приватні платформи тимчасового догляду за тваринами під час подорожей власника.

Одним із таких сервісів є TrustedHousesitters. Він позиціонується як спільнота, що поєднує власників і доглядальників у форматі пошуку pet sitter або house sit [7]. Подібні рішення розширюють перелік сценаріїв предметної області. Але, як правило, вони фокусуються на окремій задачі і не замінюють комплексний інструмент для щоденного ведення даних та планування догляду. Розглянуті рішення відрізняються насамперед своїм основним фокусом. Частина сервісів орієнтована на взаємодію з ветеринарними закладами та керування медичними подіями. Інші рішення зосереджені на ідентифікації тварини та відновленні контакту з власником. Окремі продукти працюють як маркетплейс послуг. Також існують сервіси, які підтримують залученість користувача через елементи гейміфікації. Така спеціалізація пояснює, чому більшість систем покриває лише окремі сценарії. В свою чергу, комплексна підтримка щоденних задач власника часто залишається поза межами одного продукту. У таблиці 1.1 наведено порівняльний аналіз функціональних можливостей розглянутих сервісів та концепції розроблюваного рішення. Узагальнення показує, що наявні платформи мають сильні сторони переважно в межах окремого домінуючого сценарію. Це можуть бути ветеринарні процеси, ідентифікація та повернення тварини, маркетплейс послуг або гейміфікований догляд.

Для теми даної кваліфікаційної роботи пріоритетним є інший центр. Ним виступає власник тварини та узгоджений набір щоденних функцій у межах єдиного вебзастосунку. До таких функцій відноситься профіль і базові медичні відомості, календар подій догляду з вебнагадуваннями, інтеграція з мапою для пошуку релевантних установ і місць, модуль спільноти у форматі Q&A, а також Lost&Found як інтегрований сценарій. Таке поєднання дозволяє зменшити

функціональну розрізненість дій користувача. Крім того, воно забезпечує єдиний інформаційний простір для підтримки власника домашньої тварини. Зіставлення функціональних можливостей проаналізованих сервісів із запропонованою концепцією, що наведено у таблиці 1.1, демонструє відсутність цілісного рішення. Тобто наявні сервіси не поєднують одночасно профілі тварин, події і нагадування, практично корисний геомодуль, комунікацію спільноти та модуль Lost&Found із підтримкою модерації.

Отже, актуальною є розробка інтегрованої інформаційної вебплатформи, у якій модулі працюють як єдина система. Така платформа повинна підтримувати як регулярні, так і епізодичні сценарії власника домашньої тварини. Саме це зумовлює формування невирішених питань, концепції та вимог, які наведені у підрозділі 1.3.

Таблиця 1.1

Порівняльний аналіз функціональних можливостей розглянутих вебплатформ/сервісів

Функціональний модуль	Ваша платформа (концепт)	PetDesk (USA)	Animal ID (UA/Int)	Rover (Global)	Dogiz (UK/Israel)
Профіль та медкарта	+	++(глибока інтеграція)	+(QR-паспорт)	-	+
Мапа та локації	++(з маршрутами)	-	-	+(тільки ситтери)	++(акцент на прогулянки)
Форум / Ком'юніті	++ (Q&A система)	-	-	-	+
Lost & Found	+	-	++ (основний фокус)	-	-
Пуш-сповіщення	+	++	+	+	+
Гейміфікація	++ (бейджі/маршрути)	-	-	-	++

Легенда таблиці: - немає, + є базово, ++ розвинено/ключовий фокус.

Узагальнення типових сценаріїв використання платформи дозволяє виділити ключові етапи взаємодії власника з системою. На рисунку 1.1 представлено життєвий цикл взаємодії власника з платформою. Він охоплює послідовність кроків від реєстрації та створення профілю тварини до

налаштування подій, отримання сповіщень, використання геомодуля, комунікації на форумі та активації модуля Lost&Found у разі інцидентів. Така циклічна модель підтверджує необхідність інтегрованого підходу. У цьому випадку оновлення даних та зворотний зв'язок забезпечують безперервну підтримку догляду за твариною. Саме це і лягло в основу вимог, наведених у підрозділі 1.3.

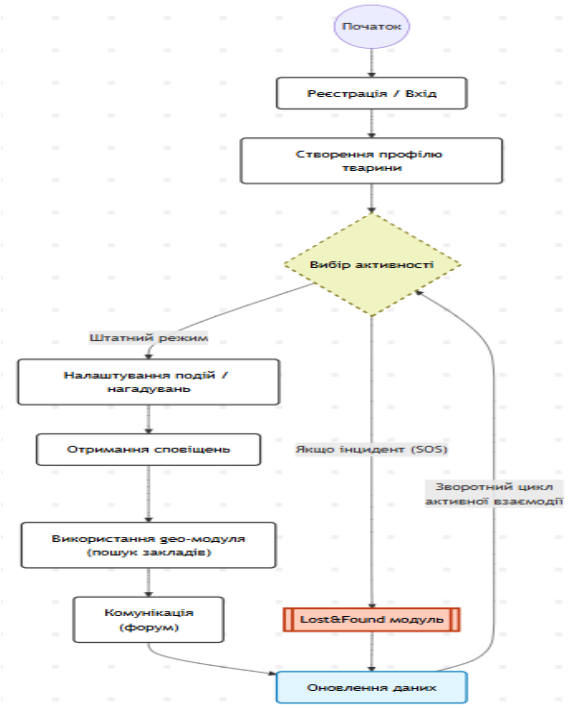


Рисунок 1.1. Життєвий цикл взаємодії власника з платформою

1.3 Визначення потреб користувачів та функціональних вимог до платформи.

Результати аналізу предметної області, який було наведено у підрозділі 1.1, а також узагальнення функціональних можливостей існуючих рішень у таблиці 1.1, дозволяють сформулювати основну проблему. Вона полягає в тому, що у користувача немає єдиного інформаційного простору для ведення даних про тварину та виконання типових щоденних задач. Через це виникає фрагментація даних. Також зростає ризик пропуску важливих подій, наприклад щеплень, планових процедур або інших дій, пов'язаних із доглядом. Крім того, геосервіси у розрізнених застосунках часто мають обмежену практичну цінність. Окремо слід зазначити і недостатню підтримку комунікації між власниками тварин, а

також слабку керованість UGC-контенту без наявності модераційних механізмів. В свою чергу, важливими залишаються ризики, пов’язані з безпекою облікового запису та приватністю даних користувача.

Запропонований підхід передбачає створення інтегрованої інформаційної вебплатформи підтримки власників домашніх тварин. В даній платформі функціональні модулі мають працювати не окремо, а узгоджено. До таких модулів відносяться керування обліковим записом, профілі тварин, нагадування та сповіщення Web Push/email, геомодуль пошуку закладів для України з пілотуванням у місті Києві, форум і модуль Lost&Found. Усі вони повинні використовувати спільну модель даних. Також передбачається рольова модель доступу з ролями User, Moderator та Admin. Це необхідно для керування контентом, підвищення довіри до платформи та забезпечення базового контролю за діями користувачів. На основі цього далі визначаються невирішені питання, концептуальні рішення та вимоги до системи.

З огляду на це, можна виокремити ряд невирішених питань, які є важливими саме для власника тварини як основного користувача платформи.

По-перше, це питання цілісності даних і формування єдиного інформаційного простору. Наявні рішення часто створюють навколо себе ізольовані набори даних. В одних сервісах профіль тварини глибоко пов’язаний із процесами певної установи. В інших головною функцією є ідентифікація тварини, а щоденні події догляду залишаються другорядними. У результаті користувач змушений розподіляти інформацію між різними системами. Медичні дані зберігаються окремо, нагадування налаштовуються окремо, а пошук локацій виконується ще в іншому застосунку. Це збільшує ризик втрати актуальності даних, дублювання записів і знижує загальну користь цифрової підтримки. З точки зору проєктування вебзастосунку це формує вимогу створити узгоджену модель даних. У такій моделі профіль тварини повинен бути базовою сутністю, з якою логічно пов’язані події, нагадування, оголошення Lost&Found та взаємодія у спільноті.

По-друге, відкритим залишається питання системної підтримки щоденної рутини. Як видно з таблиці 1.1, модулі сповіщень і нагадувань у різних системах присутні. Але їх роль і глибина реалізації відрізняються. Наприклад, у PetDesk нагадування є частиною сервісної комунікації, де ініціатором часто виступає клініка. В той же час потреба власника тварини є ширшою. Вона включає регулярні процеси: вигул, годування, обробки, вакцинації та візити. Отже, невирішеним лишається формування нагадувань як керованого процесу. Він повинен включати налаштування повторюваності, пріоритетів, підтвердження виконання та збереження історії подій. Для вебплатформи це означає необхідність вибору підходу до планування подій і реалізації push-сповіщень. Вони мають бути достатньо надійними та доречними для щоденного використання.

По-третє, геосервіси у багатьох рішеннях не забезпечують власнику контекстно релевантної підтримки. За таблицею 1.1 видно, що мапи й локації або відсутні, або підпорядковані конкретній вузькій задачі. Наприклад, у Rover мапа фактично допомагає знайти ситтерів, а в Dogiz акцент зроблено на прогулянках. Для цільової платформи власника тварини важливими є інші категорії. Це ветеринарні клініки, корисні магазини, точки для прогулянок, а у перспективі - маршрути. Таким чином, невирішеним залишається питання поєднання довідкової карти з практичними сценаріями. Мова йде про пошук, фільтрацію, побудову маршруту та інтеграцію з щоденними потребами. Наприклад, прогулянка може розглядатися як активність, що пов’язана з гейміфікацією або планом дня.

По-четверте, комунікація користувачів і пошук практичного досвіду залишаються недостатньо представленими в більшості розглянутих платформ. Це також підтверджується результатами таблиці 1.1. Навіть якщо спільнотний компонент існує, він не завжди реалізований як структурована Q&A-система з пошуком. В свою чергу, така система повинна масштабуватися за темами, породами або видами тварин і типами проблем. Це породжує окрему проєктну задачу. Необхідно визначити структуру публікацій, коментарів і механізмів

пошуку, а також базові правила якості та модерації. Без цього спільнота може перетворитися на хаотичний потік контенту.

По-п’яте, окрему увагу потрібно приділити сценарію Lost&Found. У частині рішень він реалізований достатньо сильно, наприклад в Animal ID. Проте часто він або взагалі відсутній, або не інтегрований із профілем тварини та її повсякденними даними. Для власника втрата тварини є кризовою ситуацією. Тут критичними є швидкість, точність і зручність публікації інформації. Тому невирішеним залишається питання створення такого модуля, який, з одного боку, використовує дані з профілю тварини, тобто фото, прикмети та контактні дані, а з іншого - враховує безпеку і приватність. Наприклад, які саме контакти показувати публічно та як запобігати можливим зловживанням.

По-шосте, у контексті платформи, яка акумулює персональні дані та підтримує сценарії спільноти, принциповим є питання інформаційної безпеки. У межах концепції, що наведена в таблиці 1.1, передбачено модуль аутентифікації та менеджер паролів. Це підсилює цінність продукту для користувача. В той же час така функціональність підвищує відповідальність щодо захисту облікового запису, контролю доступу та збереження конфіденційності. Отже, до невирішених питань належать вибір підходів до автентифікації, політик доступу, зберігання чутливих даних і мінімізації ризиків компрометації.

Виявлені прогалини, які наведені у таблиці 1.1, визначають вибір напрямів дослідження та обґрунтовують концепцію розроблюваного вебзастосунку як інтегрованої інформаційної платформи підтримки власників домашніх тварин. На відміну від сервісів із вузьким центром, наприклад клініка, ідентифікація або маркетплейс послуг, запропонована концепція зміщує центр уваги до власника та його щоденних задач. Вона передбачає поєднання модулів у межах єдиного середовища. До них належать профіль тварини та базові медичні відомості, мапа локацій із маршрутизацією, система нагадувань і push-сповіщень, модуль спільноти у форматі Q&A з пошуком, а також Lost&Found як інтегрований кризовий сценарій. Додатково, з урахуванням практик залучення користувачів, які підтверджуються прикладом Dogiz, доцільним є використання гейміфікації.

Бейджі та мотиваційні механіки можуть підтримувати регулярну активність і формувати звичку користування сервісом.

Таким чином, концепція вебзастосунку зумовлює необхідність дослідити та обґрунтувати такі напрями:

- структурну модель даних і взаємозв’язок модулів навколо профілю тварини;
- механізми планування подій догляду та реалізацію push-сповіщень;
- інтеграцію геосервісів для пошуку корисних локацій і побудови маршрутів;
- організацію спільнотного модуля, а саме структуру контенту, пошук і базові правила якості;
- специфіку Lost&Found як публічного сценарію з урахуванням приватності;
- підходи до захисту облікового запису та збереження чутливих даних у межах модулів аутентифікації та менеджера паролів.

У сукупності ці напрями формують логічну основу подальших рішень щодо методів і засобів реалізації у наступних розділах роботи. Також вони забезпечують обґрунтований вибір функціональної концепції платформи відповідно до результатів порівняльного аналізу, наведеного у таблиці 1.1.

1.4 Формування вимог до інформаційної веб-платформи підтримки власників тварин

Вимоги до системи були сформовані на основі ґрунтовного аналізу існуючих вебрішень, який наведено у підрозділі 1.2. Також обов'язково враховувались невирішені питання, які описані в підрозділі 1.3. При проєктуванні платформи визначено, що вона повинна відповідати міжнародним стандартам доступності [23]. Наприклад, враховуються потреби людей з певним порушенням зору. В свою чергу, детальне порівняння функціональних можливостей розглянутих сервісів (див. табл. 1.1) свідчить про один важливий факт. Більшість наявних на ринку рішень покриває лише якісь окремі задачі. В той же час

запропонована платформа максимально орієнтована на інтегровану підтримку. Тобто повсякденні сценарії власника домашньої тварини зібрані в єдиному зручному вебзастосунку. Для правильної організації роботи у межах даного проєкту передбачено три основні ролі: користувач, модератор та адміністратор.

Функціональні вимоги чітко визначають перелік можливостей системи. Вони встановлюють конкретні сценарії, які має підтримувати розроблена вебплатформа.

FR-1. Аутентифікація через Google. Система має забезпечувати швидкий вхід користувача з використанням його особистого Google-акаунта (технології OAuth/OpenID Connect). Тобто локальна реєстрація за допомогою класичного пароля не потрібна і не вимагається.

FR-2. Керування сесією користувача. Після успішної аутентифікації створюється надійна авторизована сесія користувача. В свою чергу, система має також забезпечувати коректне завершення цієї сесії (повний вихід з акаунту).

FR-3. Рольова модель доступу. Даний модуль реалізує чітке розмежування прав доступу залежно від призначеної ролі:

– користувач - має розширену змогу керувати власними даними, зокрема це профілі тварин чи нагадування. Також йому доступно створення та редагування або видалення власного контенту. Сюди відносяться пости, залишені коментарі чи оголошення у модулі Lost&Found;

– модератор - здійснює постійну модерацію користувацького контенту як на форумі, так і в модулі Lost&Found. Цей процес завжди відбувається виключно згідно з правилами платформи;

– адміністратор - має повний доступ. Це керування всіма користувачами та їхніми ролями. Він володіє значно розширеними правами модерації. Крім того, адміністратор має безперешкодний доступ до тонких системних налаштувань у межах реалізованого функціоналу.

FR-4. Профіль користувача та налаштування. Платформа має надавати користувачу окрему сторінку профілю. На ній відображається базова ідентифікаційна інформація, наприклад ім’я та пошта, які автоматично підтягнуті з Google. Також тут реалізовано можливість самостійно керувати налаштуваннями всіх сповіщень.

FR-5. Створення та керування профілями тварин. Користувач отримує зручну можливість створювати один або навіть відразу кілька профілів своїх домашніх тварин. Відповідно, при цьому доступні базові операції перегляду, швидкого редагування або повного видалення такого запису.

FR-6. Атрибути профілю тварини. Кожен створений профіль має містити обов'язковий мінімальний набір даних. До нього входить кличка, стать тварини, її порода або загальний тип, фотографія (опційно за наявності), а також додаткові текстові примітки.

FR-7. Облік записів про щеплення. Дана функція призначена для додавання та подальшого перегляду точної інформації про щеплення для кожної тварини окремо. У системі зберігається назва чи тип конкретної вакцини, дата виконання процедури, власні примітки власника. За потреби легко вказується наступна планова дата або періодичність повторення.

FR-8. Події та історія догляду. В системі забезпечується комплексне ведення історії різних подій. Тобто тут фіксуються візити до ветеринарного лікаря, регулярні обробки чи інші значущі дії, які безпосередньо пов’язані з певною твариною.

FR-9. Створення нагадувань. Будь-який користувач може легко створювати персональні нагадування про безліч подій. Наприклад: вигул, щоденне годування, майбутній візит до лікаря або планове щеплення. При цьому гнучко вказуються такі параметри: точна дата й час, текстовий опис та опційна прив’язка до профілю. Також можна зручно налаштувати повторюваність нагадування.

FR-10. Web Push сповіщення. Система повністю підтримує надсилання браузерних push-сповіщень. Обов'язково треба зазначити, що це відбувається

виключно за умови отримання відповідної добровільної згоди від самого користувача.

FR-11. Email-сповіщення. Платформа має стабільно підтримувати розсилку email-сповіщень. Вони зазвичай можуть виступати як основний або як надійний резервний канал для доставки нагадувань. Відправка відбувається безпосередньо на прив'язану адресу Google-акаунта.

FR-12. Налаштування сповіщень. Залежно від власних потреб користувач може самостійно керувати категоріями сповіщень. Це означає, що можна вмикати або вимикати якісь окремі типи нагадувань та налаштовувати загальні базові параметри для їх подальшого отримання.

FR-13. Карта корисних місць (пілотно - місто Київ). Система має надавати користувачу зручну інтерактивну карту локацій. На ній зібрані місця, які є корисними для власників тварин. Основну увагу сфокусовано на території України. В свою чергу, для пілотної демонстрації функціоналу обрано саме місто Київ. На мапі відображаються клініки, зоомагазини тощо.

FR-14. Пошук і фільтри на карті. Для забезпечення максимальної зручності користувач має змогу здійснювати швидкий пошук конкретних локацій. Також передбачено детальну фільтрацію за окремими категоріями. Наприклад, можна шукати лише «клініки» або виключно «магазини».

FR-15. Перегляд детальної інформації про локацію. Для будь-якої обраної точки на інтерактивній карті система має виводити її основні дані. Це повна назва, точна адреса, контактна інформація та інші корисні відомості, які доступні в базі.

FR-16. Форум: створення публікацій. У межах форуму користувач має змогу створювати свої власні публікації або починати нові обговорення. Для цього достатньо вказати короткий заголовок та основний текст свого питання.

FR-17. Форум: коментарі. Інші зареєстровані користувачі мають можливість вільно залишати свої коментарі чи відповіді до вже створених публікацій.

FR-18. Форум: пошук. Дана система підтримує дуже швидкий і зручний пошук у створених публікаціях. Це істотно допомагає людям швидко знаходити релевантні теми чи схожі проблеми. Як мінімум, пошук працює за вказаним заголовком або основним текстом публікації.

FR-19. Модерація форуму. Для підтримання здорової атмосфери та порядку модератор або адміністратор має змогу приховувати чи повністю видаляти певний контент. Це стосується будь-яких публікацій або коментарів, які критично порушують правила платформи.

FR-20. Lost&Found: створення оголошень. Користувач може оперативно створювати детальні оголошення типу «загублено/знайдено». Вони зазвичай містять повний опис інциденту, опційне фото тварини, дату і час, а також приблизну локацію події та контактну інформацію власника.

FR-21. Lost&Found: перегляд, пошук і фільтрація. Розроблено підтримку швидкого перегляду стрічки таких оголошень. Також доступний пошук та розширена фільтрація. Зокрема за типом, датою створення або конкретною локацією.

FR-22. Lost&Found: статуси оголошень. Щоб завжди підтримувати високу актуальність інформації в цьому важливому модулі, система забезпечує можливість закривати свої оголошення. Тобто користувач самостійно переводить своє публічне оголошення в неактуальний стан, коли його проблему вже було успішно вирішено.

FR-23. Модерація Lost&Found. Аналогічно до системи форуму, модератор або адміністратор має змогу регулярно перевіряти та модерувати ці оголошення. У разі виявлення відвертого спаму чи серйозного порушення правил платформи такий контент швидко приховується або безповоротно видаляється.

Нефункціональні вимоги визначають ключові характеристики загальної якості системи. Саме вони здатні повністю забезпечити її безпечну, стабільну та максимально зручну експлуатацію в реальних умовах щоденного використання. Для платформи підтримки власників домашніх тварин ці вимоги є критичними.

Це зумовлено саме тим, що система постійно працює з реальною персональною інформацією користувачів та містить багато відкритого публічного контенту (особливо на форумі та у модулі Lost&Found). Крім того, на ній реалізовані механізми активних сповіщень та геосервісів, які потребують стабільності.

NFR-1. Безпека та конфіденційність. При проєктуванні системи визначено, що вона має забезпечувати:

– чітке розмежування прав доступу безпосередньо на рівні серверної логіки. Це відбувається відповідно до ролей системи: користувач, модератор чи адміністратор;

– надійний захист від найпоширеніших вебзагроз. До таких відносяться атаки XSS, CSRF, різноманітні SQL-ін’єкції та несанкціонований доступ до даних;

– сувору мінімізацію збирання персональних даних. Тобто зберігаються лише ті дані, які є критично необхідними для технічної реалізації заявленого функціоналу платформи;

– надзвичайно безпечну обробку даних, які отримуються під час входу через сервіс Google. Система не повинна зберігати жодних надлишкових відомостей або дублювати те, що їй об'єктивно не потрібно;

– строгий контроль доступу до користувацького контенту. Для звичайного користувача діє жорстка заборона на редагування або видалення будь-яких чужих матеріалів. Винятком можуть бути лише дозволені випадки, що знаходяться в межах компетенцій команди модерації.

NFR-2. Надійність і стійкість до помилок. Платформа має:

– дуже коректно та безвідмовно обробляти різні помилки введення або виконання. Це легко досягається через надійну валідацію даних як на самому клієнті, так і глибоко на сервері. Також користувачу повинні обов'язково виводитись зрозумілі текстові повідомлення про помилку;

– не допускати випадкової чи системної втрати даних під час типових технічних збоїв. Наприклад, при раптовому самостійному перезавантаженні сторінки чи короткочасних проблемах зі з'єднанням мережі;

– повністю забезпечувати цілісність усіх збережених баз даних. Тобто повністю унеможливити створення нагадування без чіткої прив’язки до конкретного користувача. Аналогічно має бути технічно неможливо «зламати» або пошкодити деревоподібну структуру коментарів.

NFR-3. Продуктивність та час відгуку. Система повинна стабільно забезпечувати прийнятний та швидкий для людини час відгуку при виконанні основних сценаріїв використання. Сюди відноситься:

– миттєвий перегляд профілю тварини, всього списку поточних нагадувань, а також загальної стрічки форуму та наявних публічних оголошень Lost&Found;

– виконання надшвидкого пошуку по форуму або модулю Lost&Found. Цей важливий процес має відбуватись абсолютно без відчутних затримок для користувача;

– плавна та коректна робота інтерактивної карти, а також усіх дрібних елементів взаємодії з нею. Мова йде про масштабування, вибір потрібної локації та застосування різних фільтрів. (При цьому треба зазначити, що конкретні числові метрики всього цього будуть визначені та перевірені вже під час тестування системи безпосередньо у розділі 3.)

NFR-4. Зручність використання (UX) та адаптивність. Система безумовно має:

– мати повністю адаптивний інтерфейс для різноманітних мобільних пристроїв. Це життєво важливо, оскільки сценарії її використання «на прогулянці» чи просто «в дорозі» є чи не найбільш типовими для власника тварини;

– забезпечувати максимально просту, інтуїтивну та логічну навігацію між ключовими модулями застосунку. Ними виступають профіль тварини, система нагадувань, карта, форум та розділ Lost&Found;

– підтримувати швидке виконання всіх базових дій власника за мінімальну кількість кліків. Особливо це стосується процесу створення тварини,

додавання нового планового щеплення, створення повторюваного нагадування або публікації нового питання на платформі.

NFR-5. Сумісність. Система має стабільно та коректно працювати у великій кількості актуальних версій популярних браузерів. Окремо завжди враховується той факт, що сучасні браузери дуже часто накладають певні обмеження на Web Push-сповіщення через власні політики безпеки. Тому розроблена система має:

– дуже обережно і коректно вчасно запитувати у користувача офіційний дозвіл на надсилання таких сповіщень;

– забезпечувати максимально надійний альтернативний канал зв'язку (наприклад, звичайний email). Цей резервний сценарій миттєво спрацьовує у випадках, коли Web Push з будь-яких причин недоступний або його раніше обдумано вимкнув сам користувач.

NFR-6. Масштабованість та розширюваність. Закладена архітектура розробленої системи має бути дуже гнучкою та заздалегідь передбачати можливість її майбутнього стрімкого розвитку. Це дозволить:

– реалізувати подальше масштабне розширення географії роботи платформи. Починаючи від пілотного міста (Київ) і поступово переходячи до інших великих міст по всій території України;

– забезпечити можливе додавання нових нетипових типів нагадувань або зовсім інших каналів повідомлень (наприклад, інтеграція з популярним месенджером Telegram). При цьому базове ядро робочої системи не має потребувати складної і довгої переробки;

– закласти можливість технічно розширювати поточні рамки функцій модерації. Тобто з часом додавати нові категорії скарг, поглиблені автоматичні фільтри або розширені журнали дій.

NFR-7. Підтримуваність та якість коду. Програмна реалізація системи обов'язково має забезпечувати:

– чітку модульність. В програмі відбувається дуже логічне розділення компонентів за відповідними підсистемами. Це обліковий запис, профілі різних тварин, нагадування, форум, Lost&Found та весь функціонал карти;

– зручну можливість легкого автоматизованого тестування ключових розроблених функцій. Мінімальною вимогою для цього є перевірка основних сценаріїв використання платформи та всіх наявних ролей доступу;

– детальне ведення технічного журналювання, або так званого логування. Це активно робиться для максимально швидкої діагностики всіляких помилок, що виникають. Також система фіксує важливі дії модерації чи адміністрування у межах доцільного.

NFR-8. Локалізація та єдність інтерфейсу. Інтерфейс функціональної платформи має бути виключно україномовним, що природно розглядається як основний і безальтернативний варіант. Також в усіх впроваджених модулях повинна використовуватися логічно єдина і повністю узгоджена технічна термінологія.

1.5 Постановка задачі та критерії оцінювання результату

На підставі проведеного аналізу предметної галузі, який описано у підрозділі 1.1, а також визначених невирішених питань (див. п. 1.3) та сформованих вимог (див. п. 1.4), ставиться чітка задача. Необхідно розробити інформаційну вебплатформу для підтримки власників домашніх тварин. Ця платформа має інтегрувати основні повсякденні сценарії використання в межах єдиного вебзастосунку. Крім того, вона повинна забезпечувати керований доступ до всіх функцій відповідно до трьох визначених ролей: користувач, модератор та адміністратор. В свою чергу, порівняльні результати з таблиці 1.1 повністю підтверджують доцільність саме інтегрованого підходу, замість використання безлічі розрізнених сервісів.

Для досягнення поставленої мети в роботі необхідно вирішити наступні завдання:

1. спроєктувати загальну архітектуру та структуру платформи. Також продумати взаємодію її модулів з обов'язковим урахуванням ролей (користувач, модератор, адміністратор);

2. спроєктувати модель даних. Вона потрібна для надійного зберігання профілів тварин, записів про їх щеплення, щоденних подій догляду чи нагадувань. Також сюди входять публікації і коментарі на форумі та оголошення модуля Lost&Found;

3. реалізувати гнучкий механізм аутентифікації через Google (опціонально). Крім того, налаштувати механізми авторизації, тобто рольовий доступ для основних сценаріїв використання платформи;

4. реалізувати безпосередньо самі функціональні модулі платформи:

a. профіль домашньої тварини (включає основні дані, фотографії, щеплення та історію медичних чи рутинних подій);

b. нагадування та цільові сповіщення (використовуються Web Push безпосередньо у браузері та резервний email-канал);

c. зручний геомодуль (інтерактивна карта корисних місць по всій Україні, де пілотним містом виступає Київ);

d. форум для спільноти (можливість створювати публікації, писати коментарі, користуватися пошуком та проводити модерацію контенту);

e. модуль Lost&Found (створення нових оголошень, пошук за допомогою фільтрів, зміна статусів та модерація);

5. провести повноцінне тестування усіх основних сценаріїв. Після цього необхідно виконати оцінювання отриманих результатів за чітко визначеними критеріями.

Оцінювання кінцевого результату виконання даної роботи здійснюється за такими критеріями:

– функціональна повнота. Це означає, що успішно реалізовано всі заявлені модулі та ключові сценарії використання. Усе має повністю відповідати вимогам, описаним у розділі 1.4;

– коректність роботи виділеної рольової моделі. Розмежування прав доступу між звичайним користувачем, модератором та адміністратором має працювати стабільно. Це обов'язково перевіряється на типових тест-кейсах;

– коректність та гарантована цілісність даних. Система повинна завжди виконувати валідацію введеної інформації. Вона не має допускати будь-яких неузгоджених або просто некоректних станів даних (наприклад, технічно неможливо створити подію без прив’язки до конкретного користувача);

– надійність модуля нагадувань і сповіщень. Усі створені нагадування коректно зберігаються та вчасно спрацьовують. В свою чергу, самі повідомлення стабільно доставляються доступними каналами відповідно до налаштувань, які обрав користувач;

– продуктивність та загальна зручність використання. Інтерфейс застосунку має бути придатним для комфортної практичної роботи. Базові дії повинні виконуватися без надмірної кількості зайвих кроків, а пошук форуму чи оголошень працювати швидко та без затримок;

– забезпечення базового рівня безпеки. Тобто реалізовано надійний контроль доступу. Платформа захищена від типових помилок або некоректних запитів. Також персональні дані використовуються виключно в межах, які необхідні для нормального функціонування платформи.

Перевірка досягнення усіх вищезазначених критеріїв виконується шляхом тестування. Вона включає:

– безпосереднє виконання цілого набору тест-кейсів для основних модулів та сценаріїв;

– проведення окремої перевірки прав доступу для кожної визначеної ролі системи;

– практичну перевірку доставки Web Push та email-повідомлень в умовах контрольних сценаріїв;

– детальну фіксацію результатів такого тестування. Їх узагальнення та фінальні висновки будуть наведені вже у розділі 3.

Висновки до розділу 1

У першому розділі було виконано детальний аналіз предметної області, яка пов'язана з підтримкою власників домашніх тварин. Також тут визначено типові сценарії взаємодії користувачів з різними інформаційними сервісами. Під час дослідження було показано, що ключові задачі власника охоплюють не лише регулярні процеси догляду (ведення профілю, планові медичні події, рутина та постійний контроль виконання). Значну частку складають і епізодичні ситуації, наприклад, терміновий пошук тварини у випадку її втрати. Крім того, власники мають виражену потребу в комфортних засобах комунікації з іншими користувачами. В межах розділу також було сформовано класи даних, які є характерними для цієї предметної області. До них відносяться персональні дані користувача, дані самої тварини, медичні події, геодані, а також генерований користувацький контент для форуму і публічні оголошення Lost&Found. Окремо визначено мінімальні вимоги до якості та загальної надійності роботи розроблюваної системи.

У підрозділі 1.2 було ретельно проаналізовано наявні на ринку вебрішення та встановлено цікавий факт. Більшість із них мають досить вузьку спеціалізацію. Вони зосереджені або виключно на ветеринарній взаємодії, або на ідентифікації тварини, або взагалі працюють як звичайний маркетплейс послуг (чи інструмент гейміфікації). Додатково було розглянуто муніципальні сервіси, яскравим прикладом яких є київський «Реєстр домашніх тварин», та спеціалізовані приватні платформи тимчасового догляду. Це дозволило сформувати максимально повну картину існуючих підходів. Детальне порівняння можливостей аналогів із запропонованою концепцією повністю підтвердило наявність значних прогалин. Здебільшого вони пов’язані зі слабкою інтегрованістю даних, відсутністю модерації, незручною підтримкою повсякденних сценаріїв та проблемами надійності сповіщень.

Отже, на основі отриманих результатів повністю обґрунтовано доцільність розробки нової інтегрованої вебплатформи. Дана система гармонійно поєднує в

собі модулі управління акаунтом, детальні профілі тварин, надійну систему нагадувань (через Web Push та email), зручний геомодуль (для України, з пілотуванням у Києві), активний форум та життєво необхідний модуль Lost&Found. Робота платформи суворо контролюється через рольову модель доступу (User, Moderator, Admin).

Сформульовані в цьому розділі невирішені питання, концептуальні підходи та вимоги виступають надійним підґрунтям. Саме на їх основі в наступному розділі буде здійснюватися подальше проєктування архітектури, розробка моделі даних та побудова ключових сценаріїв використання вебзастосунку.

РОЗДІЛ 2
МЕТОДОЛОГІЯ ТА МЕТОДИ ПРОЄКТУВАННЯ Й РОЗРОБЛЕННЯ ІНФОРМАЦІЙНОЇ ВЕБПЛАТФОРМИ ПІДТРИМКИ ВЛАСНИКІВ ДОМАШНІХ ТВАРИН

2.1 Методологічні засади проєктування вебплатформи

Розробка інформаційної вебплатформи підтримки власників домашніх тварин (далі - Платформа) виконується на основі системного підходу. За такого підходу предметна область розглядається не фрагментами, а як сукупність тісно взаємопов’язаних процесів. Сюди відноситься ведення профілю тварини, планування щоденного догляду, отримання сервісної інформації та активна публікація користувацького контенту на форумі. Окрему увагу приділено кризовим ситуаціям, зокрема модулю Lost&Found. Такий підхід відіграє ключову роль. Він дозволяє ефективно уникнути фрагментації даних. Тобто він забезпечує користувачу справжній «єдиний інформаційний простір», відсутність якого була визначена як головна проблема ще у розділі 1.

Проєктування даної платформи спирається на наступні фундаментальні принципи:

- модульність. Усі базові функції логічно групуються у відповідні підсистеми. Це керування обліковим записом, профілі тварин, нагадування та події догляду, геомодуль, форум, а також Lost&Found. Кожна з цих підсистем має чітко визначені межі своєї відповідальності та власні інтерфейси для взаємодії;
- єдність і узгодженість даних. Для всіх базових сутностей (користувач, контент, довідкові об’єкти чи оголошення) передбачається єдина цілісна модель даних. Вона має чіткі зв'язки та строгі правила валідації. В свою чергу, це суттєво зменшує дублювання та підвищує загальну узгодженість інформації між різними модулями;
- безпека за замовчуванням (security-by-default). Цей принцип означає, що для всіх критичних операцій застосовуються виключно серверні механізми контролю. Сюди відноситься автентифікація та рольова модель доступу (RBAC).

Також обов'язково проводиться надійна валідація вхідних даних, яка абсолютно не залежить від поведінки клієнтської частини застосунку;

– підтримка повсякденних та інцидентних сценаріїв. Усі проєктні рішення приймаються з обов'язковим урахуванням типових щоденних дій користувача (наприклад, перегляд профілю тварини, робота з різними довідниками чи спілкування на форумі). В той же час враховуються і ситуації підвищеної терміновості. Зокрема, у сценаріях Lost&Found критично важливими параметрами є швидкість публікації оголошення та релевантність відображення супутньої інформації.

Для якісного вирішення задач пошуку по масиву публікацій або оголошеннях доцільним є застосування механізму повнотекстового пошуку. Тобто використання інвертованого індексу та ранжування результатів за релевантністю. З технічної точки зору це може бути реалізовано стандартними засобами SQLite FTS5. Або ж, у випадку майбутньої міграції проєкту на СУБД PostgreSQL, такі задачі ефективно вирішуються через функцію tsvector із застосуванням сучасних індексів GIN.

Отже, у межах даної кваліфікаційної роботи практична новизна підходу полягає не у створенні чергового вузькоспеціалізованого сервісу. Вона полягає саме в ефективній інтеграції ключових модулів у єдину вебплатформу. Дана платформа має повністю узгоджені правила доступу та забезпечує керовану публікацію користувацького контенту. Крім того, вона використовує правильні масштабовані підходи для швидкого оброблення великих масивів довідкових і просторових даних.

2.2 Методика формування вимог та сценаріїв використання

Вихідною основою для формування вимог до Платформи є результати аналізу предметної області та вже розглянутих існуючих рішень, що були наведені у розділі 1. Крім того, обов’язково враховуються сформульовані функціональні та нефункціональні вимоги, які описані у п. 1.4. Для того щоб забезпечити логічний перехід від загальних вимог до етапів безпосереднього проєктування та подальшої

реалізації, у роботі застосовано методику типу «вимога → сценарій → критерії приймання». Саме такий підхід дозволяє деталізувати кожну окрему вимогу не лише на рівні формального опису, а й через реальну поведінку користувача та системи.

У межах даної методики кожна вимога розкривається через три основні складові:

1. сценарій використання (use case) - тобто послідовність дій користувача та відповідних реакцій системи;
2. умови коректності - правила валідації, визначені обмеження цілісності даних і перевірки доступу;
3. критерії приймання - перевірювані ознаки того, що певна функція реалізована правильно. Це може бути очікувана поведінка інтерфейсу, коректна API-відповідь або зміна стану даних.

Застосування такої методики має кілька важливих переваг. По-перше, вона забезпечує трасування вимог до конкретно реалізованих функцій. Тобто кожна функція в системі має обґрунтування у вигляді початкової вимоги. По-друге, це дає чітку основу для формування тестових перевірок. В свою чергу, по-третє, такий підхід уніфікує правила коректності як для клієнтської, так і для серверної частини застосунку. Це особливо важливо для систем, де значна частина логіки пов’язана з доступом, валідацією та обробкою користувацького контенту.

У межах даної роботи як цільовий підхід до зберігання даних розглядається централізована модель на основі реляційної бази даних SQLite із використанням ORM. Такий варіант обрано не випадково. Він забезпечує цілісність, узгодженість та відтворюваність даних, що є важливим для модулів профілю тварини, нагадувань, форуму та Lost&Found. Водночас на етапі прототипування для окремих підсистем допускається використання локального збереження даних у браузері як тимчасового технічного рішення. У таких випадках це фіксується у відповідних сценаріях як альтернативна гілка. Саме тому на рисунках 2.5-2.6 показано не лише цільову реалізацію, а й проміжний прототипний варіант.

Визначення критичних сценаріїв (use cases). На основі сформованих вимог було визначено перелік пріоритетних сценаріїв використання, які безпосередньо впливають на архітектурні рішення та подальші перевірки працездатності системи. Саме ці сценарії розглядаються як критичні, оскільки вони охоплюють основні функціональні модулі платформи.

Перш за все, до таких сценаріїв належить автентифікація та контроль доступу. Сценарій автентифікації користувача включає введення облікових даних, отримання токена або сесії, а також подальше використання цього токена при зверненні до захищених функцій. В даному випадку особливо важливим є не лише сам факт успішного входу, а й правильна перевірка доступу до операцій, які не можуть виконуватись без авторизації. Типову послідовність взаємодії за схемою «користувач → клієнт → сервер → БД» наведено на рис. 2.1.

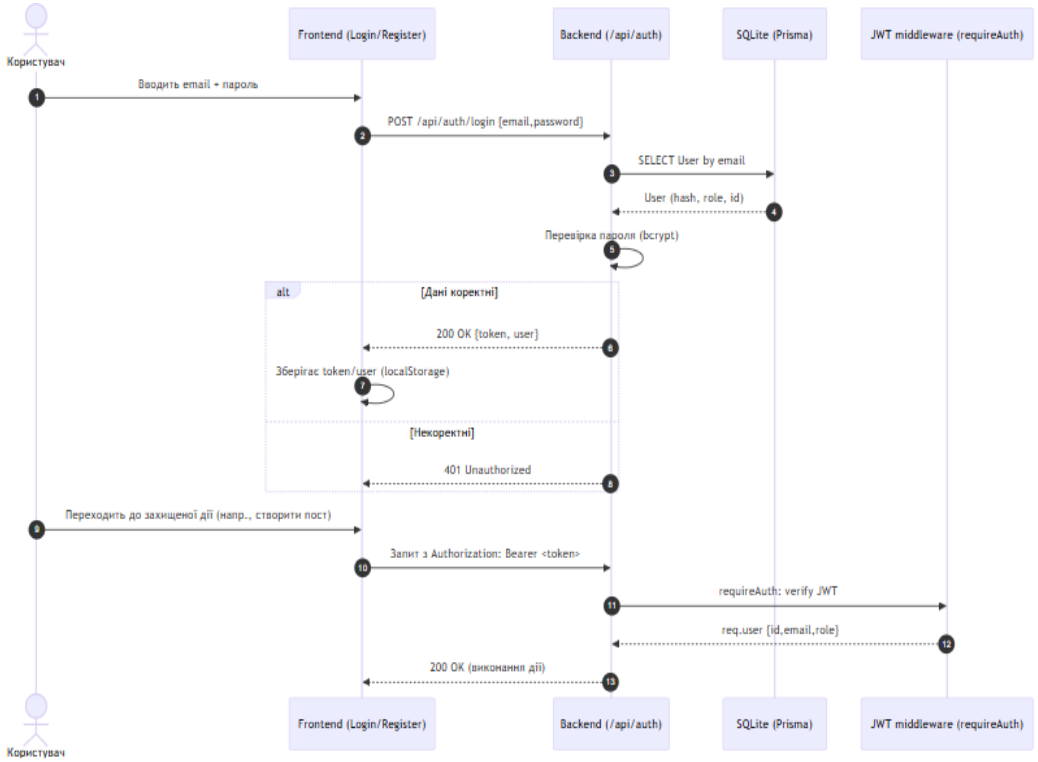


Рисунок 2.1. Сценарій автентифікації користувача та отримання токена доступу (JWT).

Зокрема, ця схема наочно ілюструє базовий потік передачі даних між абсолютно всіма рівнями застосунку. В свою чергу, такий самий суворий

алгоритм маршрутизації та перевірки прав застосовується і для інших модулів платформи, включаючи створення оголошень Lost&Found чи роботу із загальним форумом. Саме завдяки такому стандартизованому підходу вдається надійно ізолювати серверну бізнес-логіку. Безпосередньо на практиці ці визначені *use cases* стануть міцним технічним фундаментом для написання робочого коду та подальшого комплексного тестування системи.

Ще одним важливим сценарієм є ведення профілю тварини та подій догляду. Даний сценарій охоплює створення та редагування основної інформації про тварину, а також додавання пов’язаних записів, наприклад даних про щеплення, візити до ветеринара чи інші події догляду. Як і в попередньому випадку, у роботі фіксуються одразу два варіанти - прототипний та цільовий. У першому дані можуть тимчасово зберігатися локально, тоді як у другому передбачається повноцінне збереження в SQLite. Відповідні альтернативи наведено на рис. 2.2, який зображено на наступній сторінці.

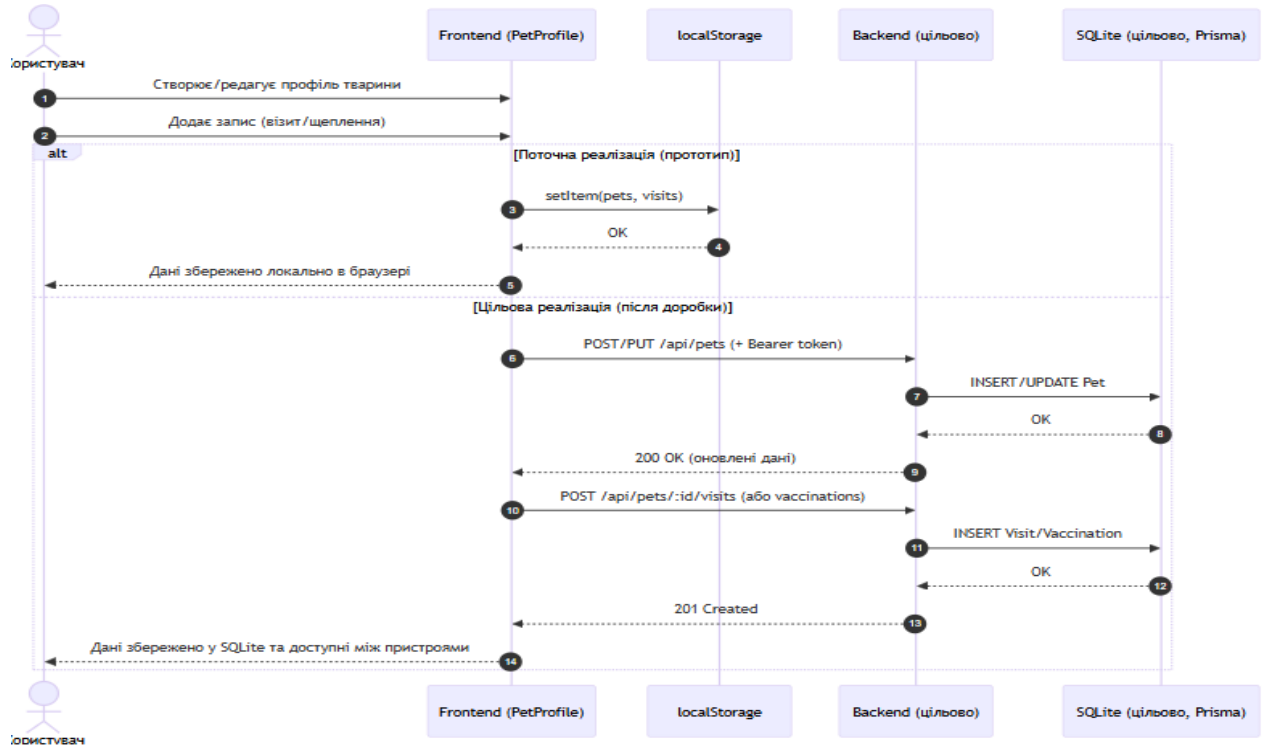


Рисунок 2.2. Сценарій ведення профілю тварини та подій догляду: поточна прототипна та цільова реалізація.

Окреме місце займає геомодуль, який відповідає за відображення корисних місць на карті. Для цього модуля важливо забезпечити не лише зручність використання, а й прийнятну продуктивність. Саме тому в роботі використовується сценарій завантаження лише тих об’єктів, які знаходяться в межах поточної видимої області карти (BBox), із додатковою можливістю фільтрації за категоріями. Такий підхід дозволяє обмежити обсяг переданих даних і, відповідно, зменшити навантаження на клієнтську та серверну частини. Логіку цього сценарію, а також межі відповідальності між клієнтом і сервером, подано на рис. 2.3.

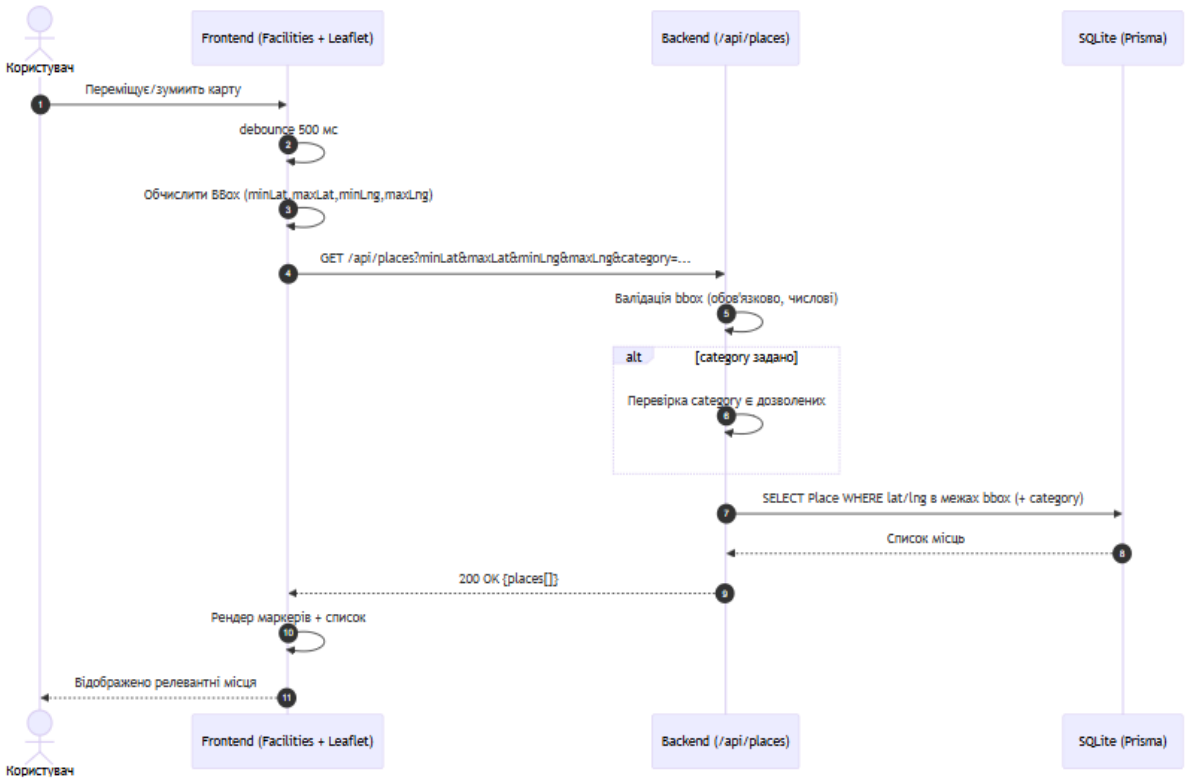


Рисунок 2.3. Сценарій геомодуля: BBox-вибірка об’єктів і оновлення маркерів при зміні області перегляду.

Для коректного наповнення карти довідковими об’єктами також було виділено окремий сценарій керованого наповнення довідкових даних, тобто workflow модерації. Його суть полягає в тому, що користувач може запропонувати новий об’єкт, після чого модератор або адміністратор виконує перевірку і приймає рішення approve або reject. Якщо пропозиція схвалюється, об’єкт публікується в

системі. Такий підхід дозволяє зменшити ризик появи на карті некоректних, недостовірних або небажаних даних. Відповідний workflow із фіксацією рішень модерації наведено на рис. 2.4.

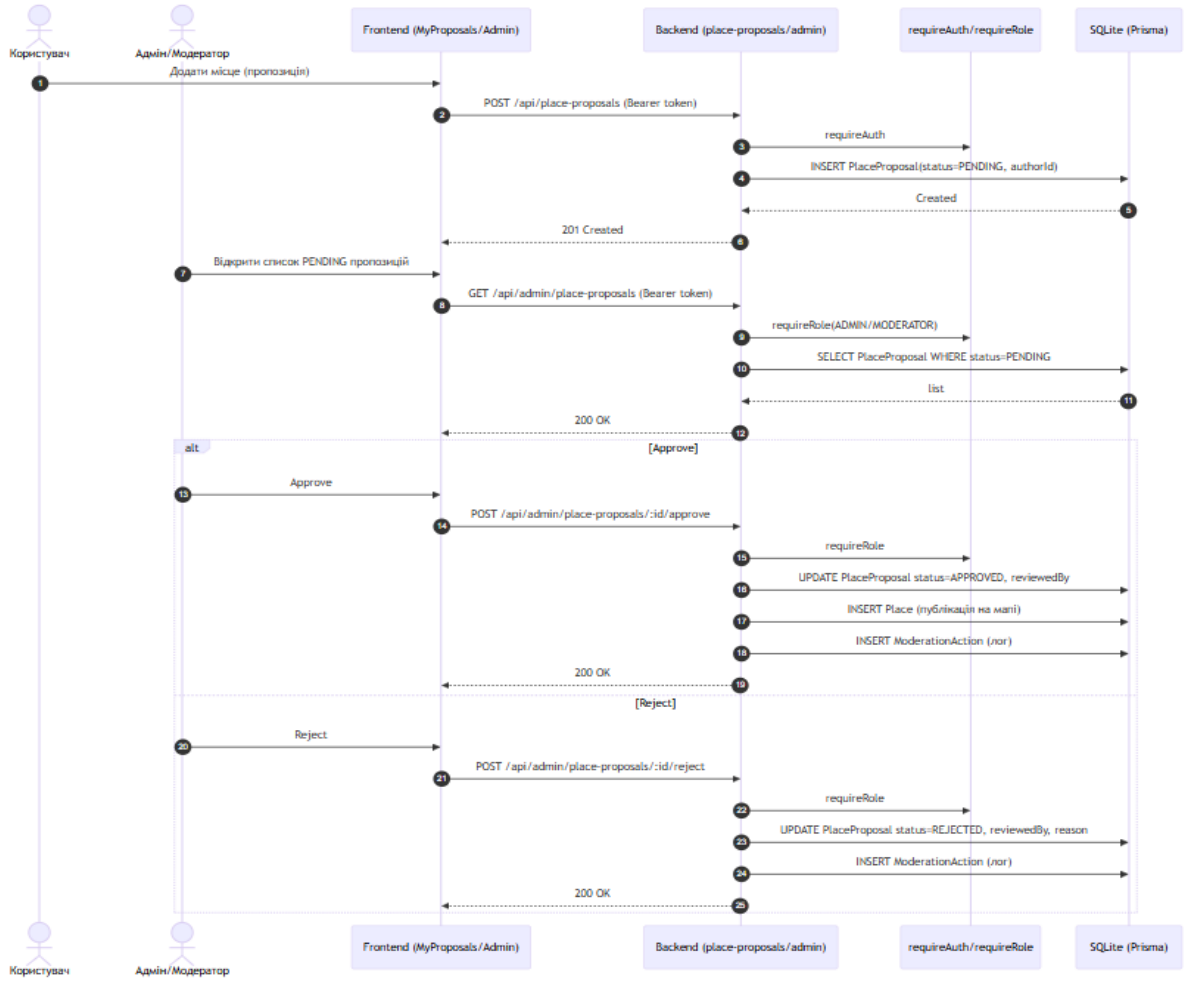


Рисунок 2.4. Workflow модерації пропозицій місць та публікації на карті.

Для модуля Lost&Found критичним є сценарій створення оголошення та його подальшого відображення іншим користувачам. У цьому випадку важливо враховувати, що даний модуль працює в межах кризових ситуацій, а тому особливу роль відіграють швидкість публікації, простота заповнення форми та актуальність інформації. З огляду на етапність розроблення в методиці окремо розмежовуються поточний прототипний варіант і цільова серверна реалізація з підтримкою модерації. Саме це і відображено на наступній сторінці на рис. 2.5. Зокрема, наведена схема чітко демонструє логічну еволюцію обробки даних: від

простого локального стейту до повноцінної і безпечної взаємодії з API. В свою чергу, додавання обов'язкового етапу модерації у цільовій реалізації дозволяє надійно захистити базу від спаму та відфільтрувати нерелевантний контент. Технічно це означає, що система перестає бути просто прототипом і стає повністю готовою до роботи з реальними користувачами.

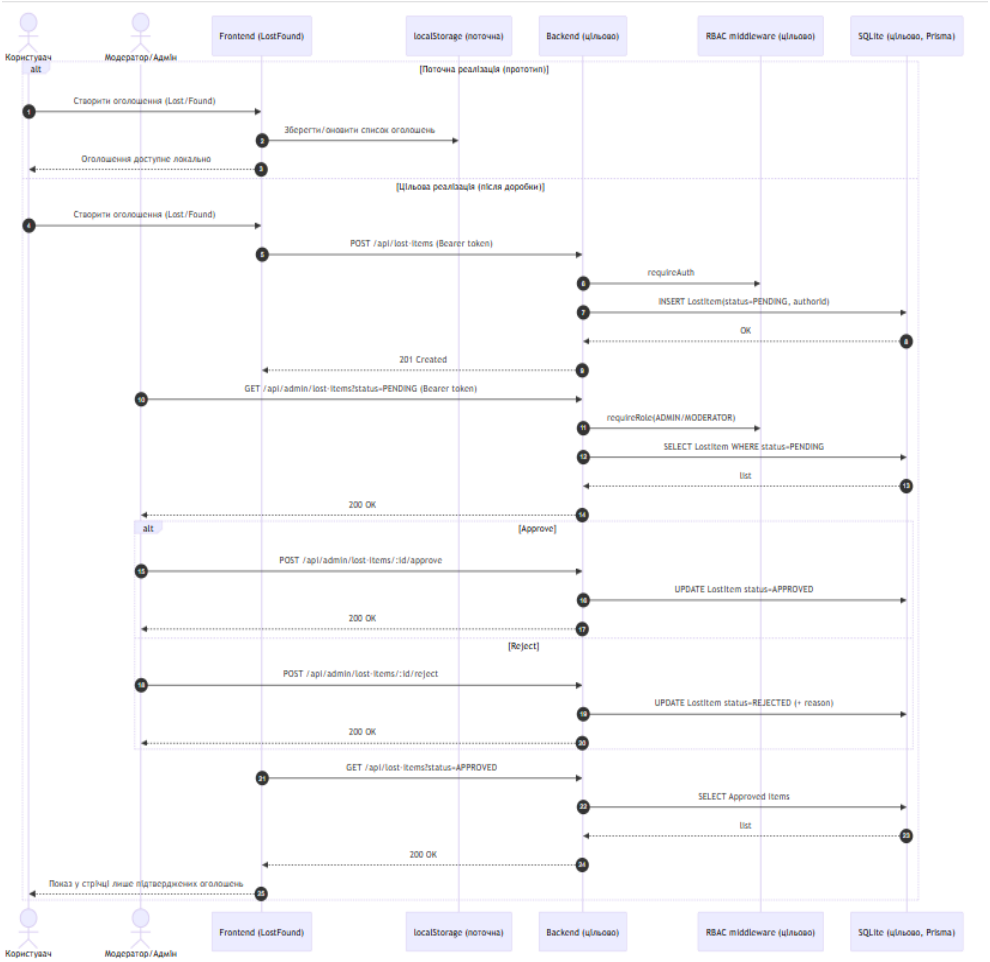


Рисунок 2.5. Сценарій Lost&Found: поточна прототипна реалізація та цільова реалізація з модерацією.

Не менш важливим модулем є форум, тобто підсистема для роботи з користувацьким контентом (UGC-content). Для нього критичними визначено сценарії створення нового поста, додавання коментарів та виконання дій, які потребують перевірки прав доступу. Наприклад, це може бути видалення матеріалу або його модерація. Такі дії мають виконуватись лише у разі наявності відповідних повноважень. Узагальнену послідовність операцій, а також точку контролю доступу до цих функцій, наведено на рис. 2.6.

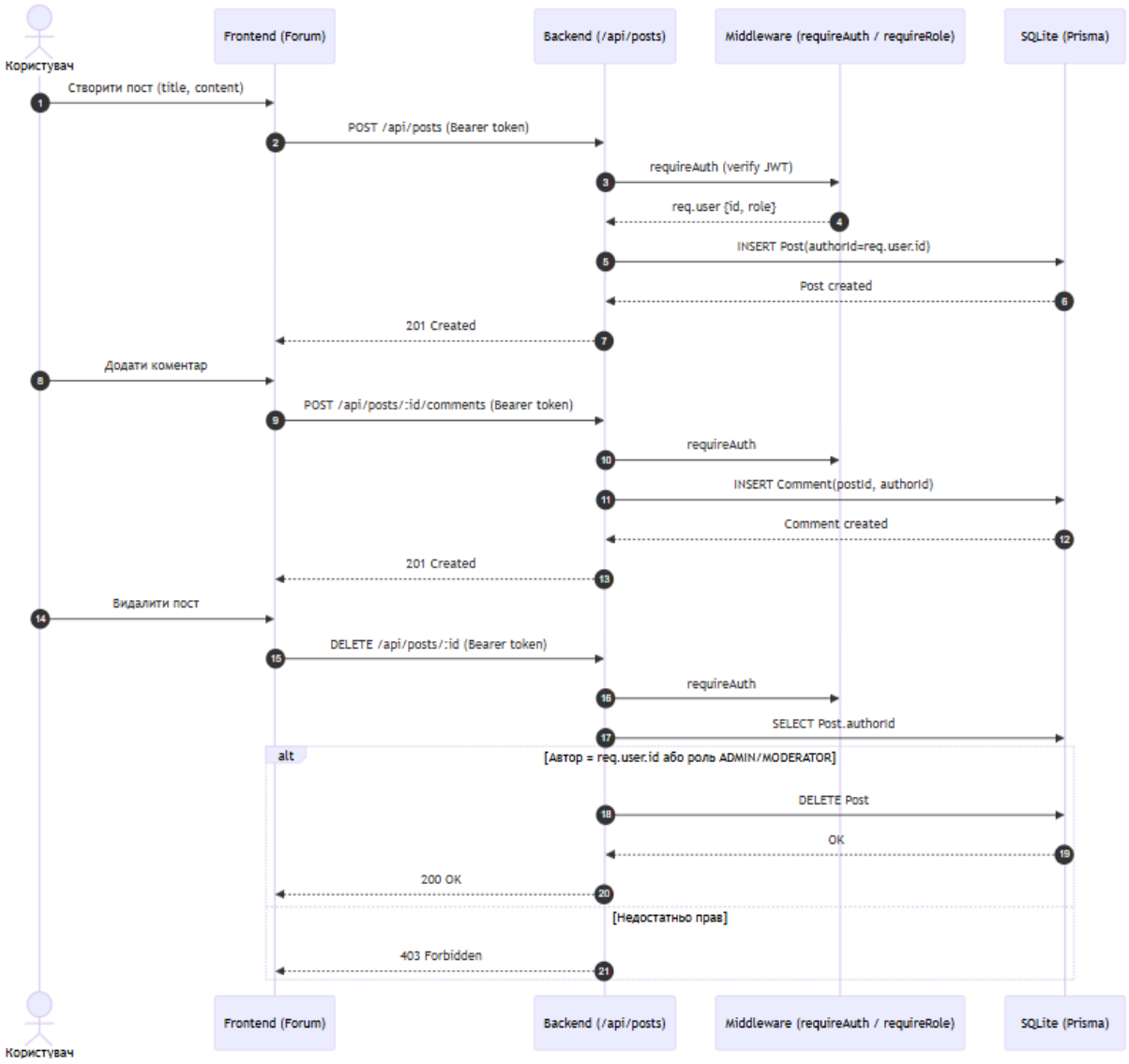


Рисунок 2.6. Сценарій роботи форуму: створення поста, коментування та видалення з урахуванням RBAC.

Окремо у роботі також розглядається сценарій входу через Google, послідовність якого наведено на рис. 2.7. У даному випадку реалізується єдиний вхід до системи з подальшою синхронізацією профілю користувача з базою даних. Після завершення цього процесу клієнтська частина отримує токен доступу, який надалі використовується для виклику API. Такий підхід забезпечує більш зручний процес входу та дозволяє відмовитися від класичної локальної реєстрації за логіном і паролем.

Окрім цього, інтеграція зовнішнього провайдера авторизації суттєво підвищує загальний рівень безпеки застосунку. Адже системі більше не

доводиться самостійно зберігати та обробляти хеші паролів безпосередньо у власній базі. В свою чергу, таке архітектурне рішення надійно мінімізує можливі ризики витоку персональних даних і значно розвантажує серверну логіку.

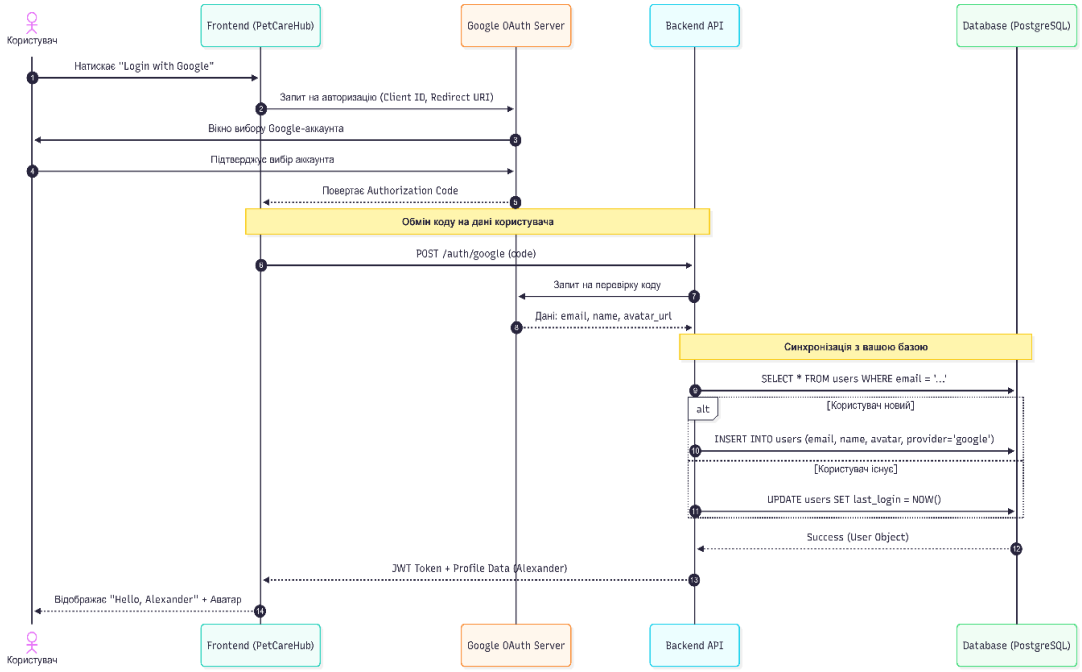


Рисунок 2.7. Sequence diagram: Google OAuth login

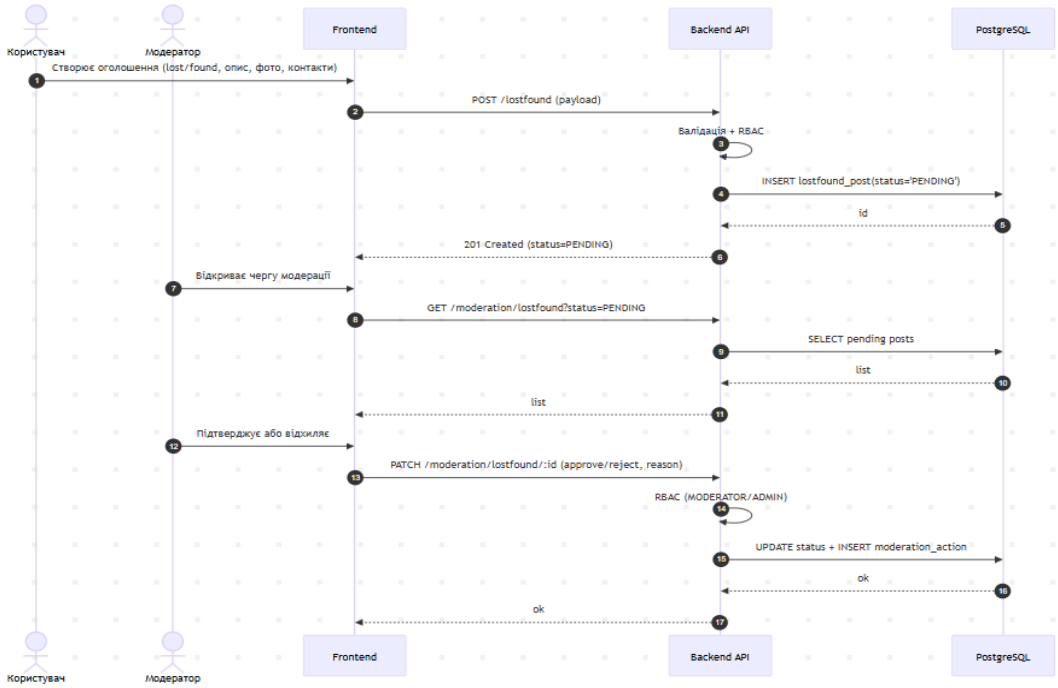


Рисунок 2.8. Sequence diagram: Lost&Found (створення → модерація)

Формалізація умов коректності та критеріїв приймання. Для демонстрації того, як саме застосовується описана вище методика, доцільно розглянути приклад геомодуля, схема роботи якого наведена на рис. 2.3. Згідно з поставленими вимогами система повинна відображати на карті лише ті об’єкти, які є релевантними для поточної області перегляду, а також підтримувати фільтрацію за категоріями. Іншими словами, клієнт не повинен отримувати весь набір даних одразу. Завантажуються лише ті точки, які справді потрібні в конкретний момент взаємодії з картою.

У цьому випадку формалізуються такі умови коректності:

– параметри меж видимої області карти (BBox) є обов’язковими та мають бути числовими;

– значення категорії, якщо воно передається у запиті, повинно належати до заздалегідь визначеного переліку допустимих категорій;

– перевірка вхідних даних обов’язково виконується на сервері незалежно від поведінки клієнта.

– В свою чергу, для цього сценарію визначаються такі критерії приймання:

– при коректно заданому BBox система повертає лише ті об’єкти, координати яких дійсно потрапляють у встановлені межі;

– якщо додатково вказано категорію, у відповіді містяться лише об’єкти відповідного типу;

– при відсутніх або некоректних параметрах система повертає повідомлення про помилку, а клієнтська частина не використовує такі дані для оновлення маркерів на карті.

Отже, сформовані сценарії використання та критерії приймання надалі застосовуються як основа для демонстрації працездатності окремих модулів. Крім того, саме вони використовуються під час побудови тестових перевірок у розділах, присвячених реалізації системи та верифікації отриманих результатів.

Такий підхід робить процес розроблення більш послідовним, а самі результати - більш обґрунтованими та перевірюваними.

2.3 Метод проєктування архітектури та розмежування модулів

При проєктуванні інтегрованої інформаційної вебплатформи було визначено, що найкраще підходить класична клієнт-серверна архітектура. Вона забезпечує чіткий поділ відповідальності між різними компонентами. Також вона дозволяє надійно централізувати всі бізнес-правила безпосередньо на сервері. Архітектурні рішення у даній роботі приймалися за відомим методом [24] модульної декомпозиції за доменами (domain-based decomposition). В свою чергу, він тісно поєднується з важливими принципами separation of concerns (розділення відповідальності) та security-by-default (безпека за замовчуванням).

Загальна архітектурна схема. Отже, система логічно поділяється на дві великі підсистеми:

– Frontend (SPA): даний модуль реалізує безпосередньо структуру користувацького інтерфейсу та зручну маршрутизацію. Він відповідає за візуалізацію даних, інтерактивну роботу з картою та форми створення чи редагування контенту. Frontend постійно взаємодіє з сервером через HTTP-запити. При цьому швидкий обмін даними здійснюється у стандартному форматі JSON;

– Backend (REST API): цей модуль повністю забезпечує бізнес-логіку системи та надійний контроль доступу (RBAC). Також він відповідає за виконання операцій над даними через інструменти ORM, модерацію текстового контенту та пошукові алгоритми. За потреби саме тут відбувається планування нагадувань або ж інтеграція з різними зовнішніми каналами сповіщень.

Такий чіткий поділ дозволяє ефективно ізолювати UI-логіку від базових правил предметної області. Тобто істотно зменшуються ризики виникнення несанкціонованих дій. Це досягається за рахунок того, що абсолютно всі критичні перевірки виконуються виключно на сервері. На рисунку 2.9 детально зображено узагальнену архітектуру Платформи. Там також показано канали взаємодії між

клієнтською частиною застосунку, її сервером, базою даних і зовнішніми сервісами.

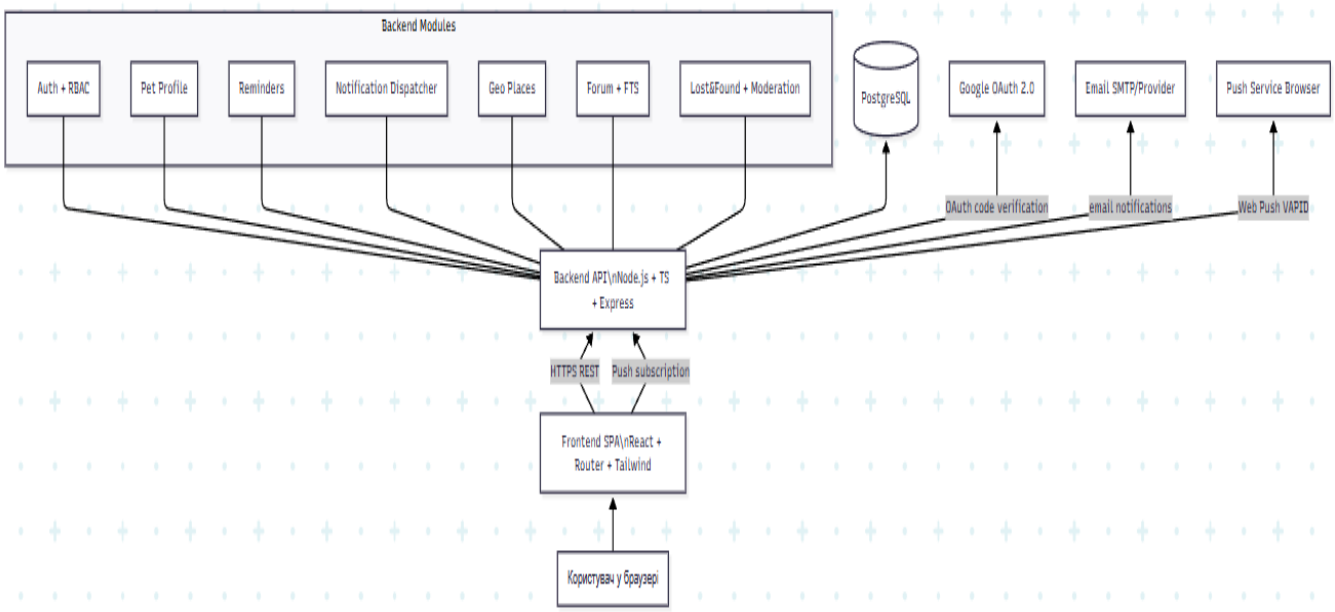


Рисунок 2.9. Загальна архітектура Платформи (компонентна схема)

Архітектурно платформа проєктується як єдиний модульний REST-сервіс. Тут кожен окремий функціональний напрям має свій власний специфічний набір маршрутів (ендпоінтів) та відповідних політик доступу. Ключовим принципом є те, що авторизація завжди суворо перевіряється на backend. Системі не важливо, як саме організовано збереження токена чи сесії на клієнті (frontend), оскільки перевірка все одно йде через сервер.

Принцип розмежування модулів. Даний метод розмежування повністю базується на одному простому принципі. Він звучить так: «Один домен - один набір API-операцій і одна політика доступу». Тобто для кожного функціонального напряму виділяється чітка група маршрутів (ендпоінтів), а також пов’язаний з ними унікальний набір правил:

– Автентифікація та користувачі. Сюди здебільшого входять реєстрація та вхід, безпечне отримання токена доступу та базові операції профілю;

– Форум (UGC). Цей домен містить пости й коментарі користувачів, правила їх редагування або видалення, а також модерацію;

– Мапа корисних місць. Модуль призначений для отримання списку об’єктів, фільтрації, зручного пошуку та створення вибірки виключно в межах видимої області карти (BBox);

– Пропозиції користувачів і модерація. Тут відбувається створення користувачем пропозиції нового місця. Модератор її переглядає, після чого йде підтвердження або відхилення. Також обов'язково ведеться фіксація його дій;

– Lost&Found. Цей модуль працює зі створенням чи переглядом оголошень. Також він контролює зміну їхніх станів (наприклад, переведення оголошення з активного в закрите).

В свою чергу, для підтримки майбутнього розширення платформи закладається ще один великий домен. Мова йде про «Профіль тварини / медичні записи / події догляду / нагадування». Він зможе легко додаватися до вже стабілізованих модулів без зміни їхньої загальної архітектури. Це робитиметься шляхом додавання нових маршрутів та сутностей.

Серверний контроль критичних операцій. Для всіх типових операцій, таких як створення, редагування або видалення (CRUD), у роботі методично закріплено одне важливе правило. Впроваджено строгий контроль прав доступу, який здійснюється виключно на backend. Тобто система не покладається лише на перевірки на стороні frontend. Причина такого рішення досить проста і зрозуміла. Клієнтський код може бути легко модифікований або скомпрометований. Тому перевірки на клієнті розглядаються лише як елемент «зручності» для користувача, а не як реальний засіб безпеки. В той же час, сервер виконує наступні маніпуляції:

– перевірку автентифікації (чи наявний та валідний переданий токен);

– перевірку призначеної ролі та конкретних дозволів користувача на виконання дії;

– внутрішню валідацію всіх даних отриманого запиту;

– безпосереднє виконання операції над базою даних (БД) або ж повернення контрольованої помилки.

Рольова модель та метод реалізації RBAC. З метою забезпечення максимальної керованості користувацького контенту, а також для надійного захисту персональної інформації, у Платформі було застосовано рольову модель доступу RBAC (Role-Based Access Control). У межах даної роботи активно використовуються три основні ролі: User (звичайний користувач), Moderator (модератор) та Admin (адміністратор).

Роль User призначена безпосередньо для вільного виконання всіх базових сценаріїв застосунку. Тобто це створення та легке редагування власних даних (профілі тварин, події догляду, налаштування нагадувань). Сюди ж відноситься створення власного контенту (нові пости, коментарі чи оголошення в Lost&Found). В свою чергу, роль Moderator забезпечує якісну модерацію публічного контенту згідно з правилами платформи. Він займається приховуванням або видаленням чужих матеріалів та регулярною обробкою оголошень (наприклад, зміною параметрів видимості). Також він здійснює контроль загальної якості довідкових даних у геомодулі, тобто розглядає пропозиції нових місць від людей. Роль Admin має максимально розширені повноваження. Вони включають серйозні адміністративні операції. Це керування всіма користувачами та їхніми ролями, а також доступ до системних налаштувань. Адміністраторові доступний і повний набір звичайних модераційних дій. Методично RBAC задається у вигляді зрозумілої матриці доступів. Вона має таку схему: (конкретна операція або дія) → (перелік ролей, яким дозволено таке виконання). На серверній стороні це реалізується як швидка централізована перевірка, котра обробляється ще перед виконанням обробника маршруту (наприклад, як middleware/guard або як перевірка безпосередньо в самому контролері). Такий підхід забезпечує абсолютно однакові правила для всіх клієнтів системи. Це вагомо зменшує ризик будь-якого обходу обмежень, оскільки клієнтська частина апріорі не розглядається як повністю довірене середовище. Для виконання найкритичніших операцій використовується поєднання відразу двох типів перевірок. Це перевірка ролі та, паралельно, перевірка належності об’єкта. Перевірка ролі (role-level authorization) означає, чи

взагалі системно дозволена певна дія для конкретної ролі. Наприклад, модерація контенту дозволена лише для Moderator або Admin. Перевірка належності об’єкта (resource ownership) означає, чи є ресурс фактично «власним» для ініціатора. Наприклад, User може редагувати та видаляти лише свої власні пости чи оголошення. Однак, у передбачених політикою випадках, Moderator може виконувати ці самі дії і щодо чужого контенту. На рисунку 2.10 зображено детальну послідовність проходження перевірок автентифікації та авторизації (RBAC) об'єктом.

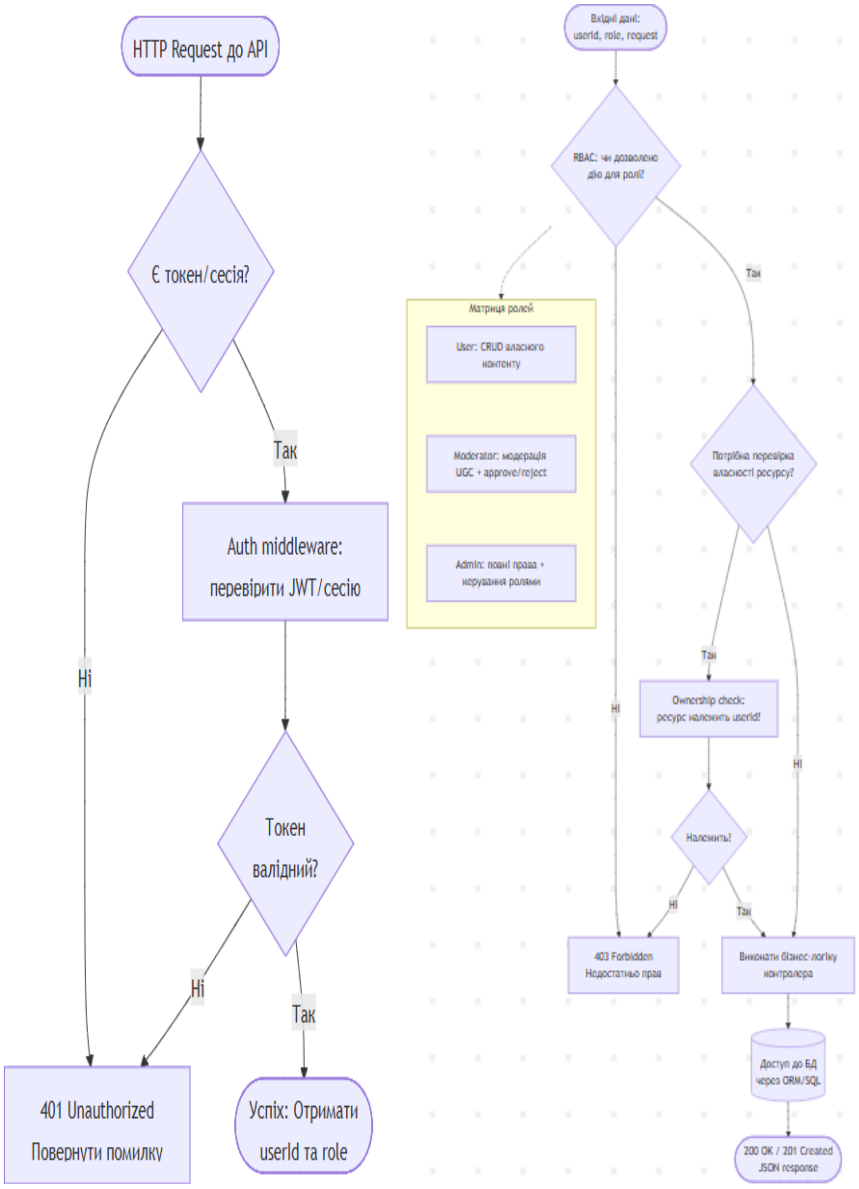


Рисунок 2.10. Схема серверної перевірки доступу (RBAC) для ролей User / Moderator / Admin.

Загалом, типовий ланцюжок серверних перевірок для успішного отримання доступу до захищених ресурсів включає в себе наступні кроки: (1) перевірку автентифікації користувача на основі токена, (2) отримання унікального ідентифікатора та ролі з бази, (3) швидку перевірку дозволів згідно з політикою доступу платформи, і вже (4) повноцінне виконання бізнес-операції (або ж повернення помилки доступу на кшталт 403 Forbidden).

2.4 Метод проєктування моделі даних та забезпечення цілісності

При проєктуванні моделі даних для Платформи було вирішено застосувати класичний ітераційний підхід. Тобто весь процес створення структури бази даних логічно поділяється на три послідовні етапи.

Перший етап - це концептуальне проєктування. Тут на основі попереднього аналізу предметної області (з розділу 1) та визначених сценаріїв використання (п. 2.2) виділяються всі ключові сутності. Також базово фіксуються типи зв’язків між ними. В свою чергу, для даної Платформи основними сутностями виступають: безпосередньо Користувач (User), Тварина (Pet), Подія догляду чи візит (Event/Visit), Публікація на форумі (Post), Оголошення (LostItem) та відповідний Об’єкт на мапі (Place).

Другий етап - логічне проєктування. Для кожної виділеної сутності чітко визначається набір її атрибутів. Одночасно підбираються оптимальні типи даних, а також первинні та зовнішні ключі. Важливо зазначити, що на цьому етапі проводиться глибока нормалізація всіх відносин (зокрема до третьої нормальної форми). Вона критично потрібна для повного усунення надмірності та забезпечення логічної цілісності бази. Завдяки цьому реалізуються стандартні зв’язки типу «один-до-багатьох» (наприклад, один користувач може мати багато постів). За потреби також використовуються складніші зв’язки «багато-до-багатьох» через створення додаткових допоміжних таблиць (якщо це необхідно для ролей платформи чи категорій).

Третій етап полягає у фізичному проєктуванні. Тут вже визначаються специфічні параметри зберігання інформації в обраній СУБД (в межах проєкту це

SQLite). Крім того, налаштовуються спеціальні індекси для суттєвого прискорення вибірок даних. Зокрема мова йде про просторові індекси для координат на мапі та текстові індекси для системи пошуку. Також безпосередньо на рівні схеми даних встановлюються суворі обмеження (constraints).

Отже, для гарантування абсолютної цілісності та коректності інформації на Платформі розроблено надійний багаторівневий підхід. Він включає наступні механізми: - Валідація на рівні прикладного інтерфейсу (API). Усі формати, обов'язковість полів та допустимі значення суворо перевіряються системою ще до моменту звернення до самої бази даних.

Наприклад, це стосується валідації існуючих категорій місць або наявних статусів для оголошень. - Декларативні обмеження власне всередині СУБД. Проєктом передбачено систематичне використання базових механізмів контролю. До них відносяться NOT NULL чи UNIQUE (використовується, наприклад, для унікального email користувача). Також застосовуються зовнішні ключі (FOREIGN KEY). Вони необхідні для каскадного видалення або блокування створення так званих «сирітських записів». Наявність ще й перевірок CHECK гарантує захист від неконсистентних станів даних навіть у разі логічних помилок у програмному коді. - Забезпечення типобезпеки безпосередньо на рівні ORM. Оскільки в роботі використовується сучасний інструмент Prisma ORM, він дозволяє автоматично синхронізувати схему бази даних із внутрішніми типами мови програмування (а саме TypeScript). У результаті це мінімізує імовірність виникнення помилок невідповідності типів ще на етапі поточного розроблення функціоналу.

Окрему велику увагу було приділено питанню забезпечення швидкого та ефективного пошуку. Тобто, для великих текстових сутностей (таких як пости форуму або оголошення в Lost&Found) передбачається спеціальна розумна індексація. Вона повністю сумісна з обраним методом оброблення тексту. В свою чергу, для геомодуля запроваджено оптимізацію запитів виключно за координатами (lat, lng). Це робиться для дуже швидкої фільтрації локацій, які потрапляють у межі обраної видимої області екрана (BBox).

Також варто прояснити поточний статус сутностей Pet, Vaccination та CareEvent. Усі вони логічно передбачені саме цільовою фінальною моделлю даних (що добре видно на рис. 2.11). Однак, у поточній робочій версії прототипу дані для цих модулів поки що зберігаються просто локально: безпосередньо в браузері самого користувача.

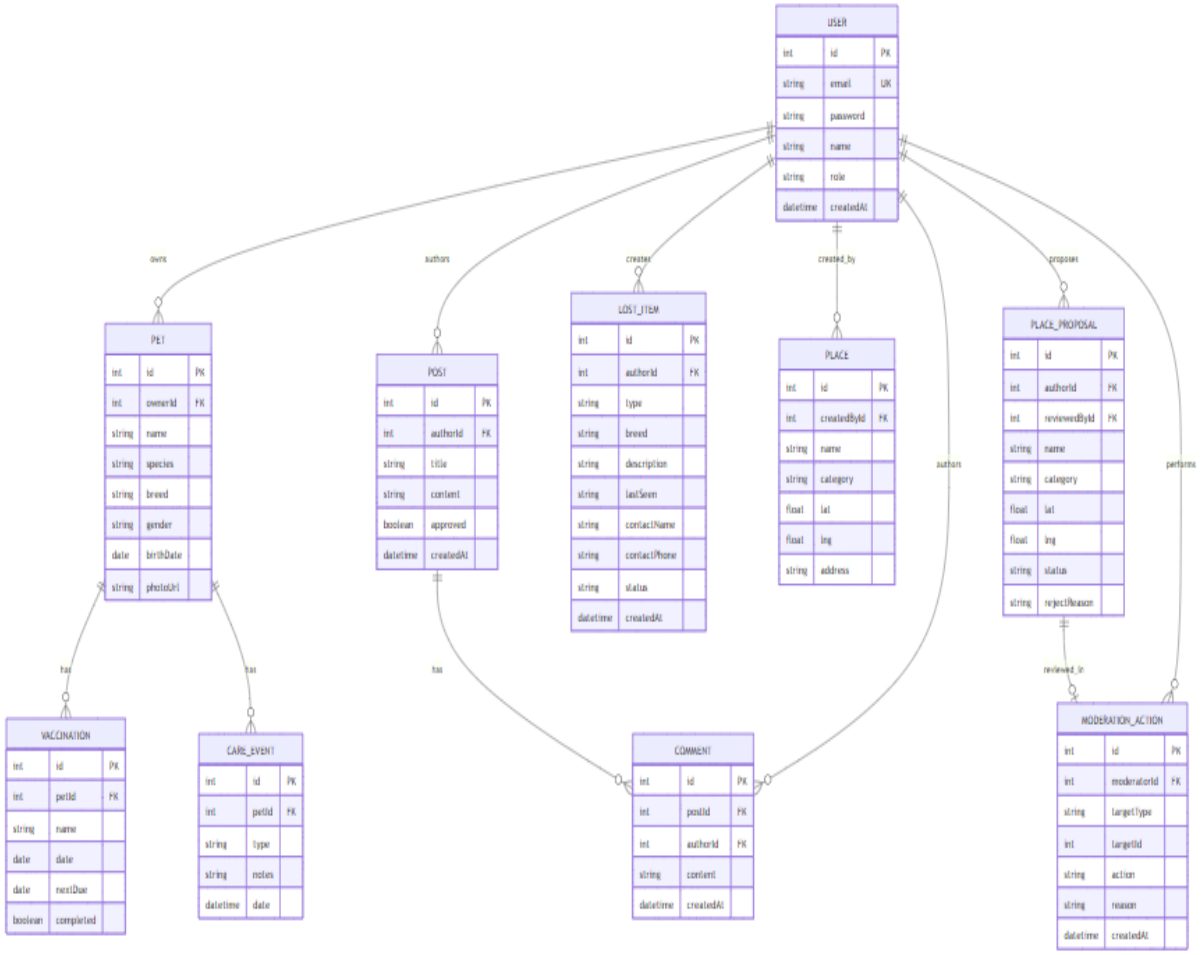


Рисунок 2.11. Логічна модель даних Платформи

2.5 Алгоритмічні методи: пошук, ранжування та планування сповіщень

При проєктуванні системи відразу розглядалося питання її майбутнього промислового розгортання та масштабування. Для таких цілей передбачається міграція бази даних на PostgreSQL. Дана СУБД надає значно розширені можливості індексації та повнотекстового пошуку. Зокрема, йдеться про використання функції tsvector разом із спеціальними індексами GIN. Однак, у

поточній реалізації розроблюваної платформи було використано SQLite. Відповідно, усі інфраструктурні рішення для PostgreSQL наразі визначено як логічну перспективу подальшого розвитку проєкту.

Метод пошуку у форумі та Lost&Found. Пошук на платформі реалізується виключно як захищена серверна операція. Даний підхід забезпечує відразу кілька вагомих переваг. По-перше, це відмінна масштабованість. Тобто системі не потрібно щоразу передавати весь масив наявного контенту на сторону клієнта для його локальної фільтрації. По-друге, гарантується сталість і швидкість результатів навіть при стрімкому зростанні обсягу даних. Також з'являється зручна можливість ранжування отриманої інформації за її релевантністю.

В якості цільового методу для такої задачі було свідомо обрано повнотекстовий пошук у SQLite через використання розширення FTS5. Він чудово відповідає обраній СУБД та надає наступні можливості:

– вільно індексувати потрібні текстові поля (наприклад, заголовок, основний контент чи опис проблеми);

– швидко і точно виконувати запити безпосередньо за ключовими словами або цілими фразами;

– своєчасно отримувати числову оцінку релевантності (так званий ранг). В подальшому цей показник ефективно використовується системою для правильного сортування результатів.

Отже, результатом роботи алгоритму пошуку є певна множина документів (це можуть бути пости або оголошення). Для всіх знайдених елементів швидко обчислюється оцінка релевантності. В результаті формується остаточна відсортована видача для користувача. Слід також зазначити, що для забезпечення повної узгодженості даних між основними таблицями та індексом FTS передбачається постійна підтримка його актуальності. На практиці це реалізується дуже просто: через контрольоване автоматичне оновлення індексованого представлення щоразу при створенні або редагуванні контенту.

Алгоритм ранжування оголошень Lost&Found (ВАЖЛИВО: потребує реалізації у розділі 3). Для стрічки Lost&Found важливо відображати не лише “найновіше”, а й “найкорисніше” для користувача. Тому застосовується гібридне ранжування, яке поєднує текстову релевантність, актуальність і повноту оголошення.

Запропоновано скорингову функцію:

$$S_{core}=w_t \cdot S_{text}+w_r \cdot S_{recency}+w_c \cdot S_{complete}+w_p \cdot S_{photo}$$

де:

- S_{text} - оцінка релевантності тексту (наприклад, ранжування FTS для запиту користувача);
- $S_{recency}$ - оцінка “свіжості” оголошення (чим новіше, тим вище);
- $S_{complete}$ - оцінка заповненості (наявність ключових полів: тип оголошення, дата/час, локація, контакти, опис);
- S_{photo} - бінарний коефіцієнт наявності фото (0 або 1);
- w_t, w_r, w_c, w_p - вагові коефіцієнти, що підбираються емпірично

Як приклад функції “свіжості” можна використати експоненційне згасання:

$$S_{recency}=e^{-\lambda.\Delta t}$$

де:

- Δt - час від моменту публікації (у годинах або днях)
- λ - коефіцієнт згасання.

Алгоритм ранжування застосовується після фільтрації оголошень за умовами видимості (наприклад, лише APPROVED та не приховані модератором). Практично це дозволяє реалізувати ранжування на рівні SQL-запиту (через обчислення Score у SELECT і сортування ORDER BY Score DESC) або на рівні бекенду (обчислення після отримання кандидатів із БД).

Метод планування нагадувань і доставки сповіщень. Для забезпечення максимальної надійності роботи нагадувань було використано системний підхід під назвою «заплановано в БД → доставляється воркером». Основна його ідея

полягає у тому, що він чітко розділяє весь процес на два логічні етапи. Першим є власне створення нагадування, тобто надійне збереження всіх його налаштувань та параметрів безпосередньо у базі даних. Другим етапом виступає сама доставка, яка працює як фонова періодична обробка запланованих подій.

В свою чергу, з методичної точки зору найбільш важливими є наступні властивості такого механізму:

– ідемпотентність. Це означає, що будь-яка повторна або технічна спроба доставки сповіщення ні в якому разі не повинна створювати некоректні дублікати в системі;

– відмовостійка обробка збоїв. Якщо обраний канал зв'язку (наприклад, push-сповіщення у браузері чи email-адреса) виявився тимчасово недоступним, це не призводить до втрати самого нагадування. Натомість система обережно переводить його у стан очікування для повторної спроби відправки;

– детальне логування процесів. В системі відбувається постійна та автоматична фіксація результатів кожної доставки події. Це робиться для того, щоб у майбутньому мати змогу проводити глибокий аналіз помилок чи затримок.

Такий підхід є цілком обґрунтованим і логічним стосовно сучасних реалій. Адже у звичайному вебсередовищі користувач може часто знаходитись офлайн, а вкладка його браузера - бути просто закритою. Сервер, своєю чергою, також може бути тимчасово перезапущений під час оновлення. Саме тому реальна стійкість системи реалізується виключно через надійне збереження стану нагадування у БД та його подальше виконання повністю незалежним фоновим процесом (воркером).

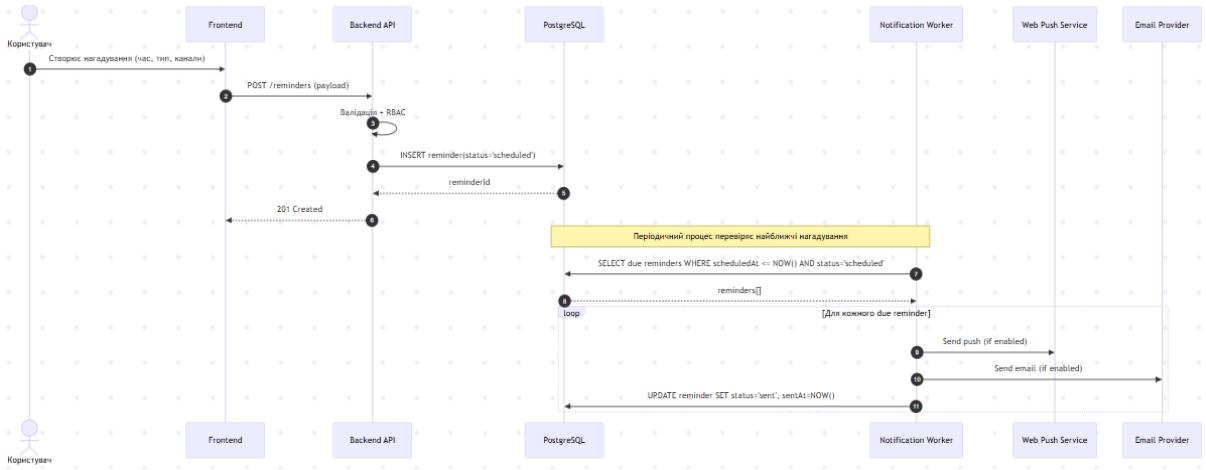


Рисунок 2.12. Sequence diagram: Reminder → Notification delivery

2.6 Методика тестування та обґрунтування достовірності результатів

Для того щоб надійно обґрунтувати достовірність отриманих результатів у процесі розробки Платформи, було визначено чітку сукупність перевірок. Усі вони безпосередньо відповідають критеріям, які раніше були детально описані у підрозділі 1.5. З метою досягнення найвищої якості кінцевого продукту, в роботі було вирішено застосувати комплексне тестування. Воно логічно і послідовно розділене відразу на кілька рівнів:

– модульне тестування (unit). Даний етап орієнтований виключно на точкову перевірку базових функцій валідації [17] та встановлених правил доступу. Також сюди відноситься перевірка різних дрібних допоміжних процедур. Як приклад, це може бути математичне обчислення скорингу для ранжування оголошень (якщо воно реалізоване у вигляді окремої функції);

– інтеграційне тестування. Цей рівень створений для перевірки правильної взаємодії API безпосередньо з базою даних системи. Наприклад, тестується такий стандартний логічний ланцюжок: створення профілю тварини, подальше додавання запису про щеплення та швидке створення відповідного нагадування. Крім того, тут ретельно перевіряється робота контролю доступу (RBAC) для різних системних ролей у зв'язці;

– функціональне (або сценарне) тестування. На цьому етапі відбувається реальне виконання ключових сценаріїв системи. Це процес

надійного OAuth-входу, спроба публікації питання на форумі чи створення оголошення у Lost&Found з подальшою модерацією контенту. При цьому обов'язково ведеться строга фіксація як очікуваних, так і реальних фактичних результатів роботи системи.

В свою чергу, для об'єктивної перевірки заявлених раніше нефункціональних вимог активно використовуються цілком практичні критерії успіху:

– абсолютна коректність доступів. Тобто система має технічно гарантувати фізичну неможливість редагування або видалення будь-якого чужого контенту для ролі звичайного користувача;

– висока продуктивність модулів пошуку. Цей аспект передбачає чітке вимірювання часу відгуку системи на найпоширеніших типових запитах. Також обов'язково проводиться перевірка коректності фінального сортування видачі;

– загальна стійкість системи сповіщень. Для цього спеціально імітуються тестові сценарії для відправки push-повідомлень чи email-листів. Головною метою тут є повне підтвердження того факту, що навіть при збоях заплановані системні нагадування ніколи не губляться.

2.7 Висновки до розділу 2.

У другому розділі було детально описано та обґрунтовано загальну методологію проєктування і розроблення Платформи. Дана система розглядається виключно як високоінтегрований вебзастосунок із класичною клієнт-серверною архітектурою. При проєктуванні було повністю та успішно застосовано модульну декомпозицію за доменами, а також виконано централізацію всіх бізнес-правил виключно на серверній стороні. Крім того, у розділі було чітко визначено і затверджено підхід до логічного розмежування зон відповідальності між клієнтом (frontend) та сервером (backend). Окремо було зафіксовано вкрай важливий принцип: абсолютно всі критичні перевірки (автентифікація, авторизація та сувора валідація) завжди мають виконуватися на рівні REST API. В свою чергу, для зручності розробки було сформульовано дієву методику переходу від сухих

вимог безпосередньо до реалізації. Вона представлена у вигляді зрозумілого ланцюжка «вимога → сценарій → критерії приймання». Це чудово забезпечує подальшу прозору трасованість та перевірюваність усіх функціональних можливостей платформи. Також було запропоновано та застосовано метод проєктування моделі даних. Він органічно розділений на три етапи: концептуальний, логічний та фізичний. Даний метод активно поєднує в собі початкову строгу валідацію на рівні API та вбудовані обмеження самої СУБД для безперервної підтримки цілісності збережених даних. У межах розділу також було описано основні алгоритмічні методи. Вони є критично необхідними для технічної реалізації пошуку та формування правильної релевантної видачі у модулях форуму і Lost&Found. Додатково було продумано цікавий підхід до гібридного ранжування оголошень, який базується на сукупності різних сигналів. Наприклад, гнучко враховується текстова релевантність, актуальність оголошення за часом створення, повнота опису та навіть банальна наявність фотографії. Також було ґрунтовно доведено доцільність методу планування нагадувань і доставки сповіщень. Його суть полягає саме у збереженні поточного стану події у базі даних та виділенні незалежної автономної процедури безпосередньо для самої доставки. Такий підхід значно підвищує загальну стійкість системи, особливо в умовах раптових збоїв чи під час перебування клієнта в офлайн-режимі. Наприкінці було запропоновано ефективний підхід до поетапного тестування системи. Впровадження модульного, інтеграційного та функціонального тестування повністю дозволяє обґрунтувати високу достовірність результатів. Це переконливо доводить фактичну відповідність платформи початково сформульованим вимогам.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ВЕБПЛАТФОРМИ ПІДТРИМКИ ВЛАСНИКІВ ДОМАШНІХ ТВАРИН

У даному розділі представлено детальний опис етапу безпосередньої практичної реалізації розробленої інформаційної вебплатформи (далі - Платформа). Основну увагу тут було приділено внутрішній структурі написаного програмного коду та процесу конфігурації бази даних. Також досить детально розглядається реалізація найбільш ключових модулів застосунку. До них традиційно належать процеси автентифікації, загальний форум, розділ Lost&Found та, звичайно, геомодуль. Окремо пояснюються механізми взаємодії між розробленою клієнтською і серверною частинами. Варто одразу зазначити, що поточна версія системи технічно реалізована саме як повноцінний функціональний прототип Платформи.

3.1 Середовище розробки та обраний стек

Для повноцінної і якісної розробки інформаційної платформи було вирішено використати сучасний та дуже популярний стек веб-технологій. Зокрема, серверну частину (Backend) побудовано на базі платформи Node.js із обов'язковим застосуванням суворої типізації мови TypeScript. В свою чергу, для клієнтської частини (Frontend) було обрано органічну зв'язку бібліотеки React та надшвидкого збирача Vite [8,16]. Щоб забезпечити максимальну мобільність та високу швидкість розгортання проєкту саме на етапі створення прототипу, у ролі системи керування базами даних (СУБД) використано легку SQLite. Однак, у системі вже надійно закладено чітку перспективу подальшої безшовної міграції на потужнішу PostgreSQL для етапу активної промислової експлуатації.

Реалізація клієнт-серверної моделі та spa. В основі розробленої системи лежить класична та перевірена часом модель сучасного односторінкового застосунку (Single Page Application, або просто SPA).

Конкретно клієнтську частину (Frontend) повністю побудовано на базі актуальної бібліотеки React версії 18. В якості інструменту збірки застосовано Vite. Саме такий вибір дозволяє забезпечити фактично миттєву реакцію графічного інтерфейсу на будь-які дії користувача. При цьому візуальна зміна сторінок відбувається без відчутного для ока повного перезавантаження. Для швидкої та зручної стилізації компонентів було інтегровано популярний CSS-фреймворк Tailwind CSS [12]. Він значною мірою забезпечив надійну адаптивність дизайну під абсолютно різні розміри екранів, включаючи всі поширені мобільні пристрої.

Що стосується серверної частини (Backend), то вона логічно реалізована як функціональне RESTful API. Сама розробка велася за допомогою середовища Node.js [14] та дуже гнучкого мікрофреймворку Express [18]. Принципово важливим рішенням стало те, що абсолютно весь серверний код було написано суто на мові TypeScript. Це допомогло забезпечити строгу статичну типізацію робочого проєкту. В результаті, такий підхід суттєво зменшив загальну кількість дрібних помилок ще на на самих ранніх етапах написання логіки.

Безпосередній обмін необхідними даними між клієнтом та сервером було налаштовано через стандартний протокол HTTP. Вся інформація зручно передається у форматі JSON. При цьому, для простого виконання асинхронних запитів на стороні фронтенду було додатково інтегровано відому бібліотеку Axios.

Конфігурація бази даних та використання ORM Prisma. Для максимально комфортної формалізації створеної раніше моделі даних, а також задля безпечного доступу до самої БД, у роботі дуже активно використовується інструмент Prisma ORM. У поточній робочій реалізації прототипу було налаштовано провайдер sqlite. Цей варіант дозволяє неймовірно зручно зберігати всі згенеровані дані просто у спеціальному локальному файлі з назвою dev.db.

Основні аспекти практичної реалізації цього шару даних виглядають наступним чином: - безпосередній опис розробленої загальної схеми даних виконано у спеціальному конфігураційному файлі schema.prisma. В свою чергу,

це дозволяє генератору Prisma автоматично створювати зручний типізований клієнт для написання запитів та дуже легко керувати послідовними міграціями БД; - задля більш ефективної роботи системи було реалізовано декілька базових індексів. Вони призначені насамперед для суттєвого прискорення алгоритмів пошуку користувачів за їхньою унікальною електронною поштою, а також для швидкого фільтрування різноманітних оголошень за потрібними категоріями; - в самій структурі передбачено можливість швидкої зміни поточного провайдера на postgresql. Це робиться буквально виправленням одного слова у файлі конфігурації. Більше того, це зовсім не вимагає переписувати усю вже готову бізнес-логіку запитів до бази.

Нижче наведено фрагмент коду, що відображає зміст файлу конфігурації.

Лістинг 3.1 - Фрагмент файлу schema.prisma з описом ключових моделей

```
model User {
    id        Int      @id @default(autoincrement())
    email     String   @unique
    role      String   @default("USER")
    posts     Post[]
    placeProposals PlaceProposal[]}
  model Post {
    id        Int      @id @default(autoincrement())
    title     String
    approved  Boolean  @default(false)
    authorId  Int
    author    User     @relation(fields: [authorId], references: [id])
  }
  model Place {
    id        Int      @id @default(autoincrement())
    name      String
    lat       Float
    lng       Float
  }
  model PlaceProposal {
    id        Int      @id @default(autoincrement())
    status    String   @default("PENDING")
    authorId  Int
    placeId   Int?     @unique
```

Реалізація геомодуля та механізмів пошуку. Досить важливою складовою частиною Платформи є геомодуль. Фізично його реалізовано у вигляді зручної інтерактивної карти, яка візуалізує всю корисну інфраструктуру для власників домашніх тварин. Саму безпосередню інтеграцію потрібних картографічних інструментів у клієнтську частину було успішно виконано за допомогою відкритої

бібліотеки Leaflet [9]. Для кращої сумісності використано її адаптацію під назвою React-Leaflet. В якості базового джерела картографічних даних (тобто самих тайлів карти) був застосований повністю безкоштовний сервіс OpenStreetMap [15]. Саме це вдале технічне рішення відразу забезпечило повну незалежність платформи від різноманітних комерційних платних API (таких як, наприклад, Google Maps).

У межах створеного геомодуля наразі повністю реалізовано дві великі функціональні можливості: - візуалізація об'єктів на карті. Практично було налаштовано відображення гарних маркерів для абсолютно різних категорій місць (це можуть бути ветеринарні клініки, спеціалізовані зоомагазини або просто звичайні майданчики для вигулу собак). Крім того, для кожного такого маркера було реалізовано інформативні спливаючі вікна (Popups). Під час кліку на маркер вони показують необхідну детальну інформацію про цей об'єкт. Зокрема туди виводиться назва місця, його фізична адреса, приблизний графік роботи та залишені контактні дані; - динамічне програмне керування картою. В системі було прописано та реалізовано зручні механізми автоматичного фокусування поля камери на обраному об'єкті. Також додано працюючу фільтрацію всіх маркерів у реальному часі, яка напряму залежить від спеціально обраної користувачем категорії на бічній панелі.

В свою чергу, механізми гнучкого пошуку на платформі були практично реалізовані відразу на двох базових рівнях: - серверна фільтрація даних. Спеціально для роботи модуля «Місця» (Facilities) розроблено окремі API-ендпоїнти. Вони повноцінно підтримують передачу вхідних параметрів для запитів, щоб фільтрувати дані ще глибоко на рівні бази даних сервера. Сам пошук при цьому здійснюється шляхом перевірки співпадіння за назвою об'єкта або його категорією. Технічно це елегантно реалізовано за допомогою операторів contains та equals у запитах від Prisma ORM - дуже швидкий клієнтський пошук. Для того, щоб стабільно забезпечувати практично миттєвий відгук візуального інтерфейсу (особливо в тих розділах, де постійно відображається дуже велика кількість

дрібних елементів), було використано бібліотеку Fuse.js. Дана бібліотека повністю локально реалізує потужний алгоритм нечіткого пошуку, який базується на відомому обчисленні відстані Левенштейна.

Також тут варто обов'язково зазначити, що для таких великих розділів як «Форум» та «Lost&Found», вже архітектурно повністю підготовлено ґрунт до переходу на систему повнотекстового пошуку (Full-Text Search, FTS). Тобто, у поточній схемі бази даних завчасно виділено відповідні текстові поля під майбутню індексацію контенту. Це дозволить вже згодом, при подальшому масштабуванні проєкту та перенесенні бази на СУБД PostgreSQL, дуже легко і просто реалізувати якісне ранжування результатів за їх релевантністю. І, що найголовніше, це відбудеться взагалі без глобальної переробки логіки коду прикладного рівня.

Нижче можна побачити, як саме виглядає логіка серверної фільтрації в самому коді.

Лістинг 3.2. Приклад реалізації фільтрації об'єктів на сервері (placeController.ts)

```
export const getPlaces = async (req: Request, res: Response) => {
  const { minLat, maxLat, minLng, maxLng, category } = req.query;const where: any =
{
    lat: { gte: parseFloat(minLat as string), lte: parseFloat(maxLat as string) },
    lng: { gte: parseFloat(minLng as string), lte: parseFloat(maxLng as string) },
  };if (category) where.category = category as PlaceCategory;
   const places = await prisma.place.findMany({ where, take: 500, orderBy: {
createdAt: 'desc' } });  res.json({ data: places });};
 export const createPlace = async (req: Request, res: Response) => {
   const { name, category, lat, lng, address } = req.body;
   const place = await prisma.place.create({   data: { name, category,
lat:parseFloat(lat), lng: parseFloat(lng),address, createdById: req.user?.id }});
   res.status(201).json({ message: 'Place created successfully', data: place });
```

3.2 Реалізація архітектури та модулів платформи

Модульна декомпозиція та склад реалізованих підсистем**.** Архітектурно Платформа була розроблена як набір окремих функціональних модулів. Вони надійно об’єднані між собою єдиною спільною моделлю даних на базі Prisma та SQLite, а також REST API. Слід зазначити, що поточна практична реалізація містить одразу два різні типи модулів. По-перше, це вже повністю інтегровані full-

stack модулі (тобто працююча зв'язка Frontend + Backend + БД). По-друге, це модулі, наразі реалізовані переважно на клієнтській стороні з використанням локального збереження у localStorage. Вони виступають швидким прототипним рішенням до моменту повного завершення серверного CRUD у найближчому майбутньому.

До переліку вже повністю реалізованих модулів належать наступні: - автентифікація та авторизація (Auth). Даний модуль технічно містить класичний JWT-флоу, безпечне хешування паролів, визначає глобальний контекст авторизації у клієнтській частині та виконує відповідні middleware-перевірки доступу безпосередньо на сервері; - загальний форум (Forum). Він відповідає за створення постів та коментарів, виконує всі необхідні CRUD-операції через API, а також контролює правильні зв’язки сутностей через Prisma; - геомодуль корисних місць (Places/Facilities). Це інтерактивна карта на базі Leaflet, яка дозволяє вільно отримувати та зручно фільтрувати місця через запити до API (в тому числі здійснювати пошук за точними координатами або заданою видимою областю екрана); панель адміністрування та модерації (Admin). Модуль спеціально призначений для суворого RBAC-контролю доступу і дозволяє зручно модерувати користувацькі пропозиції нових місць; - домашня сторінка (Home Page). Тут на практиці реалізовано інтерактивну навігацію та умовний рендеринг інтерфейсу (Conditional Rendering). Тобто вигляд сторінки напряму залежить від поточної ролі авторизованого користувача (Admin/Moderator або звичайний User). Крім того, на сторінці відбувається загальна агрегація ключових сервісів платформи з їх гарним адаптивним відображенням.

В свою чергу, до частково реалізованих модулів (вважаються станом прототипу) належать: - Lost&Found. На даний момент на стороні фронтенду успішно реалізовано процес додавання, пошуку та фільтрації, включаючи механізм нечіткого fuzzy-пошуку [20] за допомогою бібліотеки Fuse.js. Проте збереження інформації поки що здійснюється просто локально у localStorage. Варто зазначити, що у схемі бази даних вже завчасно створена модель LostItem, однак серверні маршрути і контролери для її повної інтеграції наразі тимчасово

не підключені; - профіль тварини (Pet Profile). Для цього розділу вже повністю розроблено зручний UI та відповідну логіку керування введеними даними (за допомогою кастомних хуків React). Самі ж дані тимчасово зберігаються також у localStorage (з чіткою прив'язкою до унікального userId) до етапу фінального завершення розробки серверного модуля сутностей «питомці/медичні записи».

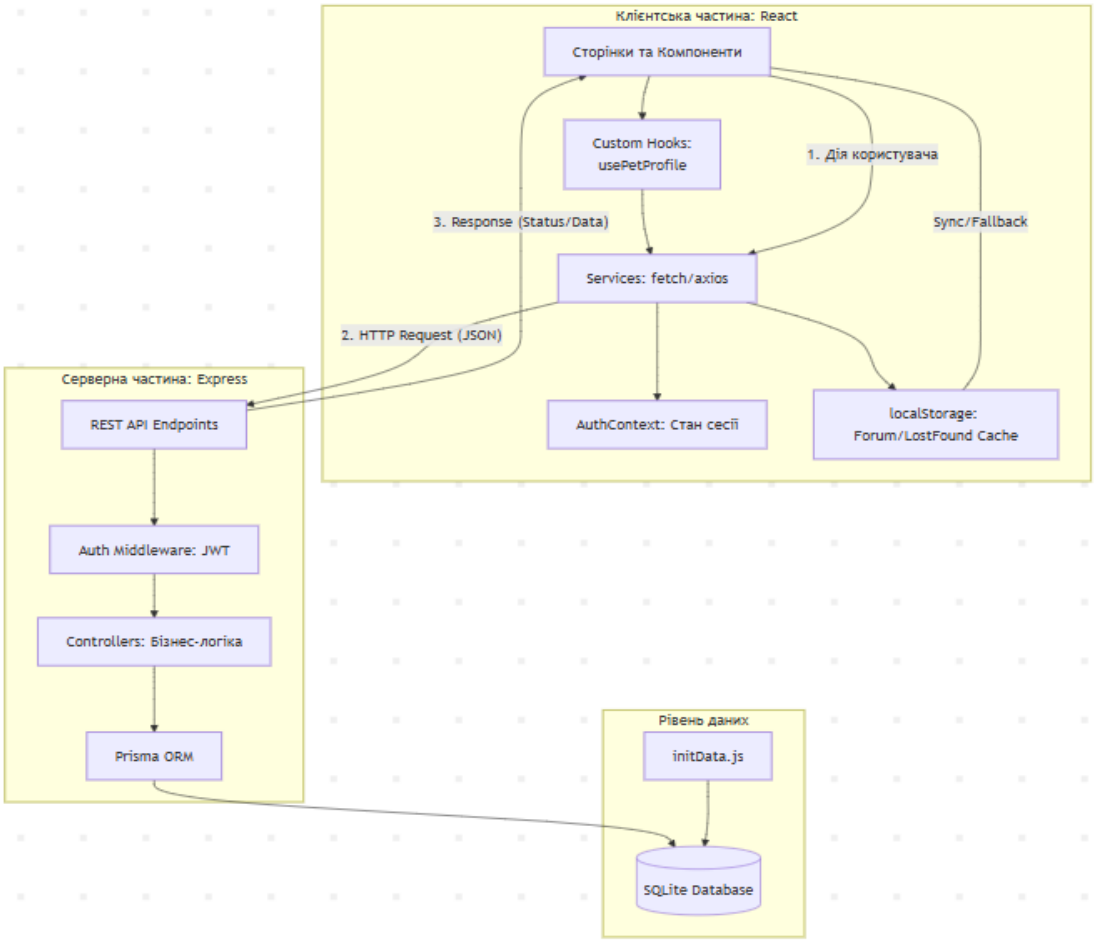


Рисунок 3.1. Узагальнена схема модулів платформи та потоків даних

Реалізація серверної частини: маршрутизація, контролери, безпека**.** Як зазначалося раніше, серверна частина платформи повністю побудована як об'ємне REST API на платформі Node.js та фреймворку Express. Для доступу до даних ефективно використовується Prisma, а весь серверний код написаний мовою TypeScript. Програмна логіка була чітко структурована за окремими сутнісними доменними зонами (наприклад, Auth, Forum, Places, Admin тощо). При цьому загальна структура поділяється на кілька логічних рівнів: - рівень маршрутів (routes). Він визначає конкретний набір доступних HTTP-ендпоїнтів та

безпосередньо зв’язує кожен з них із відповідним логічним обробником; - рівень контролерів (controllers). Даний шар надійно інкапсулює у собі саму бізнес-логіку системи. Сюди належать ключові процеси створення, читання, оновлення чи повного видалення сутностей, а також валідація даних і базові перевірки; - рівень проміжного програмного забезпечення (middleware). Саме на цьому рівні було реалізовано більшість наскрізних механізмів безпеки системи. Насамперед це перевірка переданого токена JWT. Також тут виконуються обов'язкові суворі рольові перевірки (за моделлю RBAC) для оперативного надання доступу до захищених адміністративних функцій.

Зокрема, у модулі автентифікації (Auth) вже впроваджено повноцінний цикл безпеки. Він включає в себе стандартні операції реєстрації та входу користувача. Далі йде автоматичне формування та гарантована перевірка JWT. Окрім цього, обов'язково здійснюється криптографічно надійне хешування паролів усіх користувачів за допомогою популярної бібліотеки bcrypt. Завершує цей ланцюжок системна перевірка ролей користувача на сервері для надання або блокування доступу до різноманітних захищених ресурсів платформи.

Слід також додати, що в таблиці 3.1 наочно представлено основні групи REST API-ендпоїнтів, які логічно розділені безпосередньо за своїми модулями (Auth, Forum, Places, Admin).

Таблиця 3.1

Групи REST API-ендпоїнтів за модулями (Auth/Forum/Places/Admin)

Модуль	Ендпоїнт (Path)	Метод	Опис операції
Authentication (Auth)	/api/auth/register /api/auth/login /api/auth/me	POST POST GET	Реєстрація нового користувача Аутентифікація та отримання JWT-токена Отримання даних поточного профілю
Forum (Posts)	/api/posts /api/posts /api/posts/:id/like /api/posts/:id	GET POST POST DELETE	Отримання списку всіх постів форуму Створення нового допису (з підтримкою зображень) Додавання/видалення «лайка» до посту Видалення власного допису користувачем

Продовження таблиці 3.1

Lost & Found	/api/lost-found /api/lost-found	GET POST	Список оголошень про загублених тварин Створення оголошення про знахідку/втрату
Places (Facilities)	/api/places /api/places/search /api/proposals	GET GET POST	Отримання списку pet-friendly локацій Пошук локацій за фільтрами (тип, місто) Подання заявки на додавання нової локації
Pets & Profile	/api/pets /api/pets /api/pets/:id	GET POST DELETE	Отримання списку вихованців користувача Додавання нової картки тварини Видалення профілю тварини з системи
Admin Panel	/api/admin/users /api/admin/places /api/admin/stats	GET PATCH GET	Управління списком користувачів (Admin only) Модерація та підтвердження нових локацій Отримання системної статистики платформи

В свою чергу, на лістингу 3.3 показано невеликий фрагмент коду middleware, що відповідає саме за перевірку JWT та відпрацювання рольової моделі (RBAC) з робочого файлу middleware/auth.ts

Лістинг 3.3 - Фрагмент middleware перевірки JWT та ролей (RBAC) (middleware/auth.ts)

```
export const requireAuth = async (req: Request, res: Response, next: NextFunction) =>
{const authHeader = req.headers.authorization;
  if (!authHeader?.startsWith('Bearer ')) return res.status(401).json({ error: 'No
token' });
  try {
    const decoded = jwt.verify(authHeader.substring(7), getJwtSecret()) as any;
 req.user = { id: decoded.userId, email: decoded.email, role: decoded.role };next();
  } catch (err) { res.status(401).json({ error: 'Invalid token' }); };
 export const requireRole = (allowedRoles: Role[]) => (req: Request, res: Response,
next: NextFunction) => {
  if (!req.user || !allowedRoles.includes(req.user.role as Role)) {
    return res.status(403).json({ error: 'Insufficient permissions' });  }  next();};
```

Окрему увагу під час розробки було приділено створенню спеціального адміністративного модуля. Доступ донього системно обмежений і дозволений лише для ролей ADMIN або MODERATOR. Метою даного інструменту є зручне внутрішнє керування зібраними даними та своєчасна модерація користувацьких пропозицій нових локацій у розділі Places. Такий підхід дозволяє дуже комфортно і, головне, централізовано контролювати всі критичні системні операції. Тобто

розробнику взагалі не потрібно постійно дублювати однакову логіку перевірок доступу в кожному окремому контролері.

Реалізація клієнтської частини: сторінки, компоненти та інтеграція з api. Вся візуальна клієнтська частина платформи була повністю реалізована як сучасний SPA-застосунок на базі React. Під час розробки цього шару було враховано кілька ключових особливостей.

По-перше, для підтримки глобального стану авторизації було використано стандартний механізм AuthContext. Це дозволяє легко і стабільно керувати поточним токеном та даними користувача абсолютно в усіх місцях застосунку.

По-друге, весь безпосередній доступ до серверних ресурсів було акуратно інкапсульовано у спеціальні сервіси (зокрема, це authService.js, forumService.js, placeService.js).

Якщо ж говорити про реалізацію, то на рисунку 3.2 досить детально зображено вигляд сторінок автентифікації (зліва) та форми реєстрації нового користувача (справа).

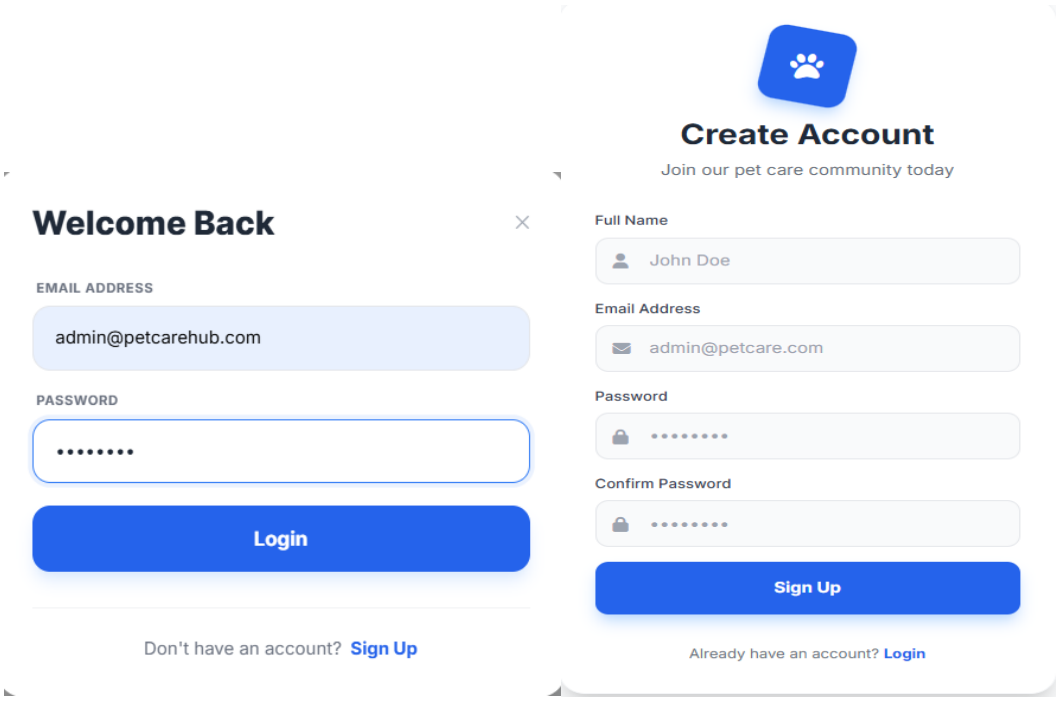


Рисунок 3.2. Сторінки авторизації: Login/Register

Таке рішення значущо полегшує подальшу підтримку коду проєкту, а також дозволяє неодноразово використовувати готові набори запитів у

найрізноманітніших компонентах. По-третє, фізичний модуль карти (Facilities) якісно збудований через набори Leaflet-компонентів, які миттєво відображають масив даних, що надходить з API. Що ж стосується власне домашньої сторінки платформи, то її було реалізовано з використанням вже згаданих механізмів умовного рендерингу. Таким чином відбувається динамічне відображення саме того інтерфейсу, що відповідає стану авторизації конкретно в цей момент. Додатково сюди ж інтегровано стрічку останніх подій, що працює через внутрішній сервіс повідомлень.

Загалом, домашня сторінка (Home Page) виступає в ролі такого собі центрального навігаційного вузла системи. Вона забезпечує дуже зручний та адаптивний доступ до всіх трьох ключових сервісів платформи: безпосередньо інтерактивної карти, загального форуму спільноти та корисного розділу розшуку пропавших тварин. Як вже відмічалося, візуальний інтерфейс саме цієї сторінки вміє динамічно підлаштовуватися. Система визначає роль користувача і надає йому відповідні інструменти для управління контентом. Для прикладу, візуальну демонстрацію інтерфейсу такої головної сторінки з боку адміністратора можна роздивитись на рис. А.1 (Додаток А). Вона завжди містить додаткову панель швидкого доступу до аналітики та модерації платформи. Натомість, вигляд домашньої сторінки від імені звичайного користувача наведено на рис. Б.2. Тут логічний акцент свідомо зміщено саме на персоналізовані сервіси та найсвіжіші оголошення іншимх учасників.

З метою забезпечення ефективного зворотного зв’язку, а також для покращення соціальної інтеграції сайту, у системі було спеціально розроблено додатковий інформаційний блок. Він містить зручні інтерактивні посилання на офіційні сторінки проєкту в популярних мережах - Instagram, Telegram, Facebook та TikTok. Ця додаткова навігація гарантує людям дуже швидкий доступ до зовнішніх каналів підтримки платформи. Даний модуль інтегровано суворо у футер сайту, що незмінно гарантує його доступність користувачу на абсолютно будь-якій сторінці застосунку. Такий підхід суттєво підвищує загальний рівень

залученості цільової аудиторії та робить процес отримання сторонньої підтримки набагато легшим через їхні улюблені месенджери.

Безпосередньо розроблений та успішно інтегрований модуль інтерактивної карти, яка візуалізує всю збережену в базі інфраструктуру, наочно зображено на рисунку 3.3.

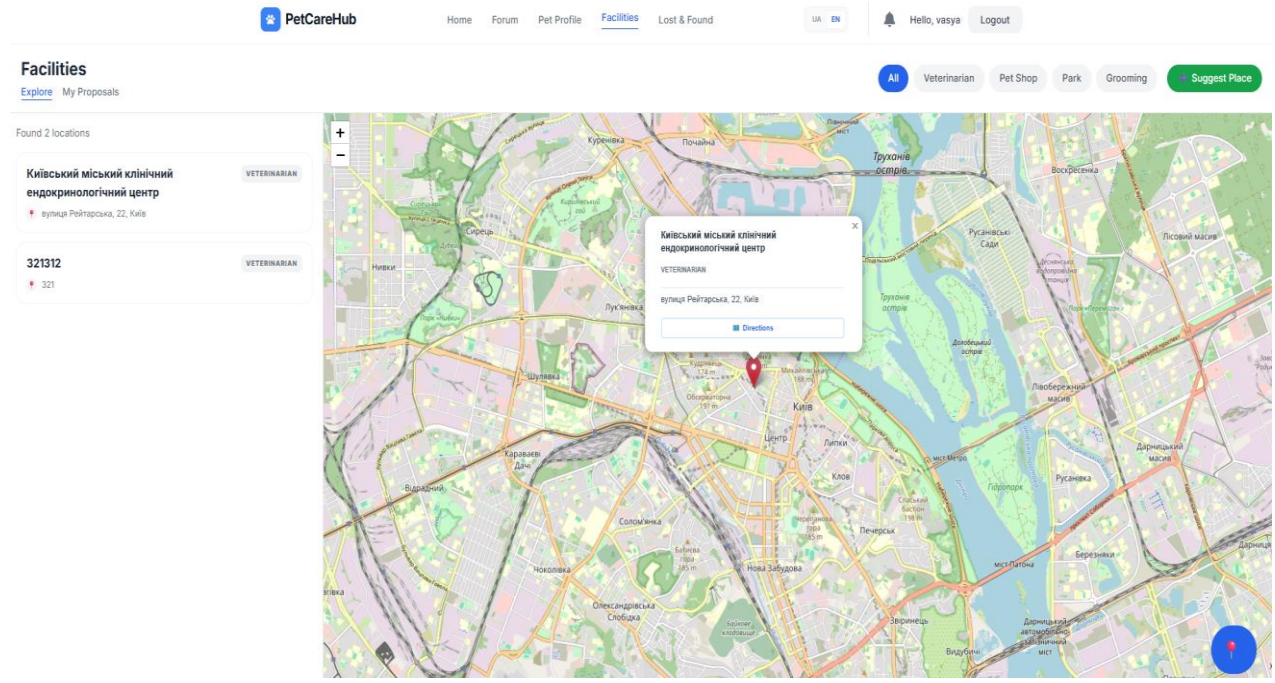


Рисунок 3.3. Інтерфейс карти з відображенням маркерів об’єктів

В свою чергу, розроблений модуль Facilities безпосередньо відповідає за відображення актуальних об'єктів інфраструктури на інтерактивній карті від OpenStreetMap. При цьому надійно реалізовано зручну функцію швидкого фільтрування всіх присутніх локацій за категоріями (наприклад, Veterinarian, Pet Shop, Park або Grooming). Якщо користувач натискає на обраний ним маркер, відразу відкривається інформативне спливаюче вікно (popup). У ньому чітко показується точна назва місця, його фізична адреса, а також обов'язкове корисне посилання на зовнішній великий картографічний сервіс.

Якщо ж детально розглядати повний цикл взаємодії користувача з даним модулем, то він логічно охоплює відразу декілька основних сценаріїв. Перш за все, абсолютно будь-який зареєстрований користувач має можливість подати власну пропозицію щодо додавання нового корисного місця. Це робиться через

спеціальну зручну форму «Suggest a Place». Під час її заповнення необхідно обрати точну геолокацію об'єкта на самій карті, вказати його текстову назву, відповідну категорію, а також базову адресу розташування та години роботи (рис. Б.13). Відразу після успішної подачі така нова пропозиція автоматично отримує системний статус PENDING. З цього моменту вона відображається у особистому кабінеті користувача-автора у окремій вкладці «My Proposals». Варто зазначити, що там само постійно знаходяться і ті місця, які вже були раніше успішно погоджені системою та отримали статус APPROVED (рис. Б.14). Після схвалення така нова точка майже миттєво стає видимою на загальній карті для всіх відвідувачів платформи (рис. Б.15).

З боку адміністратора платформи цей процес виглядає наступним чином. Спочатку йому надходить автоматичне системне сповіщення про щойно створену користувачем пропозицію. Це відбувається через внутрішній комунікаційний модуль повідомлень (рис. Б.16). Далі, вже в самій адміністративній панелі, зібрана черга таких пропозицій зручно відображається у спеціальному захищеному розділі «Place Proposals». Тут для кожного нового сформованого запису передбачено дві головні функціональні кнопки для модерації: «Approve» (схвалити) та «Reject» (відхилити) (рис. Б.17). При цьому, перед остаточним погодженням об'єкта, система завжди перевіряє обрану дію і запитує звичайне додаткове підтвердження в адміністратора (рис. Б.18). Якщо все добре і кнопка натиснута, то система повідомляє про успішне додавання нової локації (рис Б.19). Саме після цієї дії перевірене місце фактично відразу з'являється на загальній інтерактивній карті (рис. Б.20).

В свою чергу, автор цієї поданої пропозиції одразу після рішення отримує радісне сповіщення про схвалення його запиту. Воно ненав'язливо відображається безпосередньо у його особистій стрічці нотифікацій, що розташована поряд із картою (рис. Б.21). Однак, на практиці інколи виникають ситуації, коли у поданих користувачем даних є певні невідповідності, помилки або грубі порушення правил. У такому разі модератор або адміністратор має повне право відхилити цей проблемний запит. Для цього спеціально використовується логічна форма

відхилення. У ній перевіряючий обов'язково вказує аргументовану текстову причину відмови перед відправкою (рис. Б.22). Відразу після виконання цього кроку система самостійно надсилає користувачу швидке сповіщення про те, що його пропозицію було наразі відхилено (рис. Б.23). А у його персональному розділі «My Proposals» статус цього запису назавжди актуалізується та змінюється на REJECTED. Там само детально відображається і залишений пояснювальний коментар модератора. Даний підхід гарантовано забезпечує максимальну прозорість усього процесу прийняття рішень під час модерації (рис. Б.24).

Окрім роботи із картою місць, також у системі було повноцінно розроблено та реалізовано великий соціальний модуль форуму. Ознайомитися з його візуальним представленням можна на рисунку 3.4.

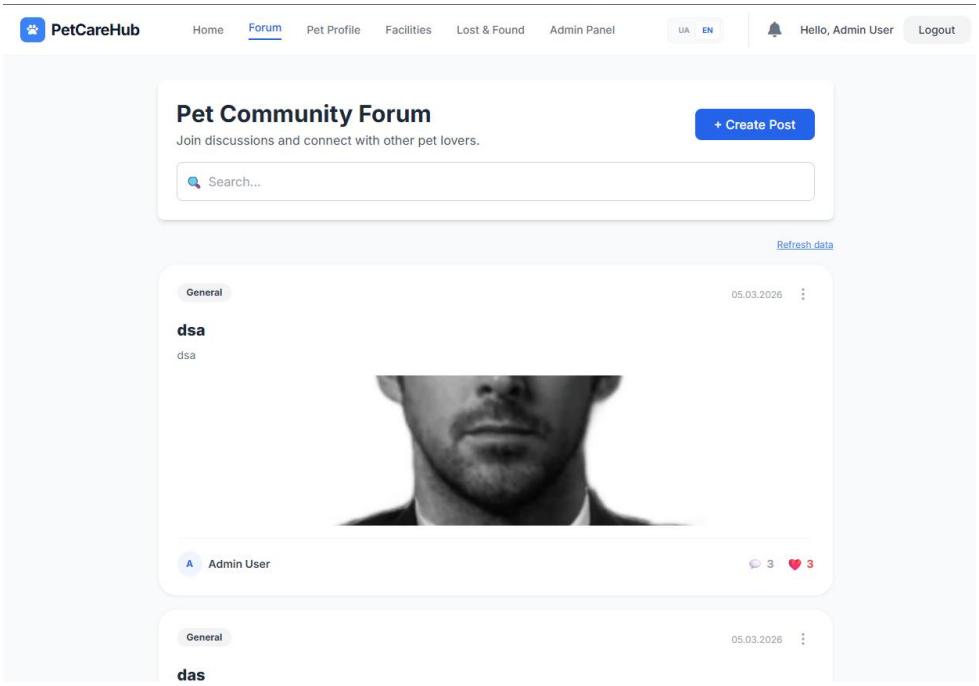


Рисунок 3.4. Модуль форум для користувачів

Основна мета даного модуля Forum полягає у коректному виведенні загальної центральної стрічки користувацьких публікацій. Він має якісну внутрішню підтримку рубрикації контенту за визначеними категоріями платформи, працюючу вбудовану систему лайків та зручні коментарі. Всі перелічені дані швидко отримуються та оновлюються виключно через захищені запити до REST API. Важливим архітектурним рішенням стало те, що можливість

створення повністю нових постів доступна лише для успішно автентифікованих у системі користувачів. У той же час, звичайний перегляд самої стрічки із публікаціями залишається абсолютно відкритим публічним ендпоїнтом.

Лістинг 3.4 - Приклад виклику API з клієнтського сервісу (фрагмент authService.js або forumService.js)

```
const forumService = {
// Створення поста з Bearer токеном
async createPost(postData, token) {
  const response = await fetch('/api/posts', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${token}`
    },    body: JSON.stringify(postData),
    cache: 'no-store'});
  const data = await handleResponse(response);
  return data.data;
},// Лайк поста (тільки Authorization заголовок)
async likePost(postId, token) {
  const response = await fetch(`/api/posts/${postId}/like`, {
    method: 'POST',
    headers: { 'Authorization': `Bearer ${token}` } });
```

Серед найбільш ключових особливостей даної кодової реалізації при налаштуванні викликів API варто виділити наступні моменти:

– handleResponse. Це спеціально створена розробником спільна утилітарна функція. Вона необхідна для швидкої перевірки статусу response.ok та подальшої централізованої обробки можливих помилок при роботі з мережею;

– Authorization: Bearer ${token}. Представляє собою цілком стандартний і надійний спосіб правильної передачі JWT-токена саме для авторизації на закритих захищених ендпоїнтах сервера;

– cache: 'no-store'. Цей технічний параметр вказується у запитах браузера виключно для того, щоб свідомо запобігти будь-якому небажаному кешуванню оброблених динамічних даних. Це ефективно рятує від проблеми відображення застарілої інформації користувачу.

Сам процес та безпосередню реалізацію візуальної форми для створення користувачем нового поста наочно представлено на рисунку 3.5.

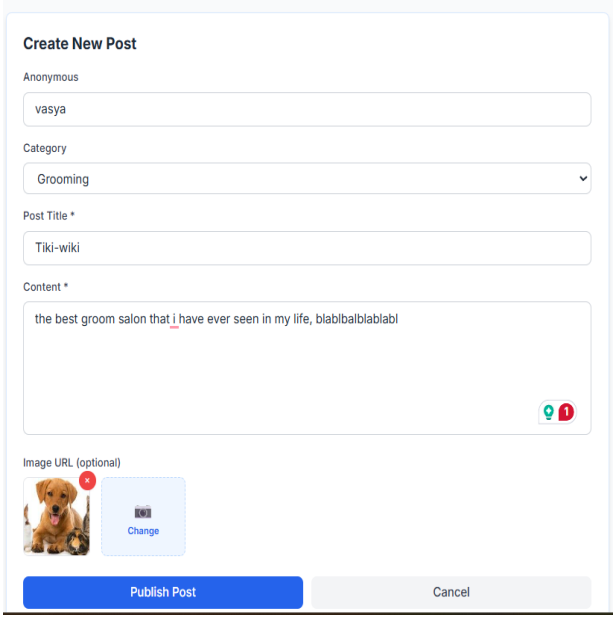


Рисунок 3.5. Сторінка форум (форма додавання поста)

Дана форма дуже гнучка і повністю підтримує зручний вибір необхідної категорії з випадаючого списку. Також вона дозволяє легке додавання стороннього тематичного зображення через його зовнішню URL адресу. Крім того, на етапі налаштувань передбачено зручну можливість самої публікації тексту як просто від імені анонімного автора, так і від повністю іменованого реального профілю. Технічно весь цей процес реалізований саме через відправку форматованого POST-запиту від клієнта до серверного шляху /api/posts/.

Особливості частково реалізованих модулів. Щодо досить важливого модуля Lost&Found, то наразі в ньому вже успішно реалізовано весь повноцінний UX-ланцюжок подій, проте діє він поки що виключно на клієнтській стороні (фронтенді). Тобто, в системі вже чудово працює візуальне додавання нових карток з оголошеннями, їх локальний пошук користувачами, динамічна фільтрація та комфортний перегляд. Окрім цього, вже зараз інтегровано якісний fuzzy-пошук, який працює на основі популярної бібліотеки Fuse.js. Його візуальну складову обробки запитів можна побачити на рисунку 3.6. Оскільки поточний продукт виступає як прототип, на цьому етапі розробки усі введені тестові дані тимчасово зберігаються просто у localStorage браузера самого клієнта. Дане

рішення було свідомо реалізоване, адже воно дозволяє дуже наочно демонструвати весь багатий робочий функціонал цього модуля без прямої жорсткої програмної перешкоди у вигляді необхідності в повністю готовій серверній частині. Водночас обов'язково треба зазначити, що у підготовленій затвердженій схемі бази даних у Prisma вже надійно фігурує цільова модель LostItem.

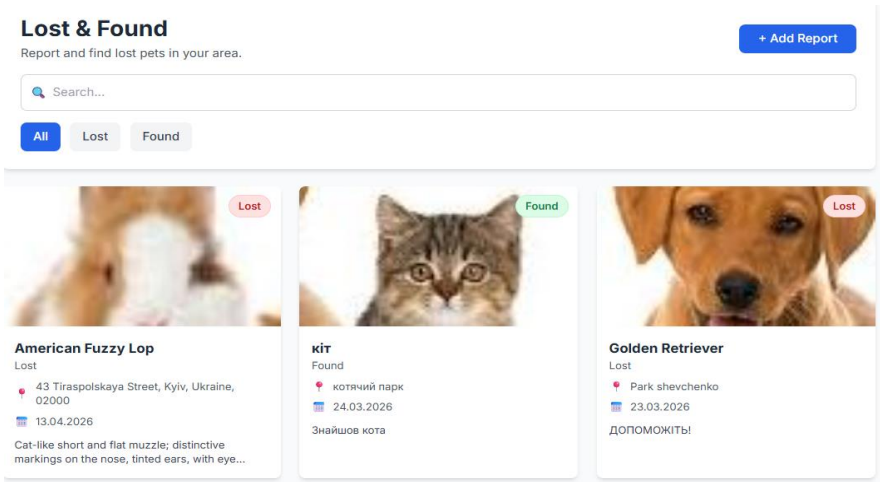


Рисунок 3.6. Модуль LostFound (фільтрація)

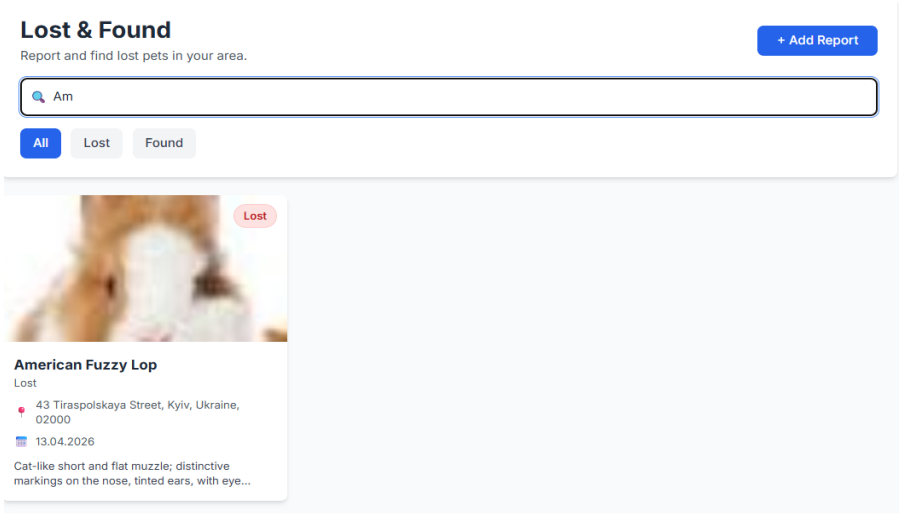


Рисунок 3.6. Модуль LostFound (продовження)

Цей крок створює дуже міцну основу під майбутнє швидке і безшовне підключення повного серверного CRUD (тобто відповідних контролерів і маршрутів), а також гарантує перенесення локальних тимчасових даних вже у справжню повноцінну БД бази.

На практиці сам модуль Lost & Found охайно відображає зручні візуальні картки всіх актуальних активних оголошень. Особливістю цього розділу є підтримка швидкої фільтрації карток за їх базовим статусом на вибір (тобто Lost або Found). Впроваджений раніше у логіку fuzzy-пошук повністю і дуже ефективно забезпечує точне правильне знаходження цільових результатів. Це чудово і наглядно продемонстровано як приклад на нижньому скріншоті системи з успішним пошуком за неповним частковим запитом «Am». Природно, для того щоб успішно подати повністю нове оголошення, користувач має заповнити заздалегідь підготовлену форму подачі такої інформації. Її фактична та візуальна реалізація дуже детально представлена на рисунку 3.7.

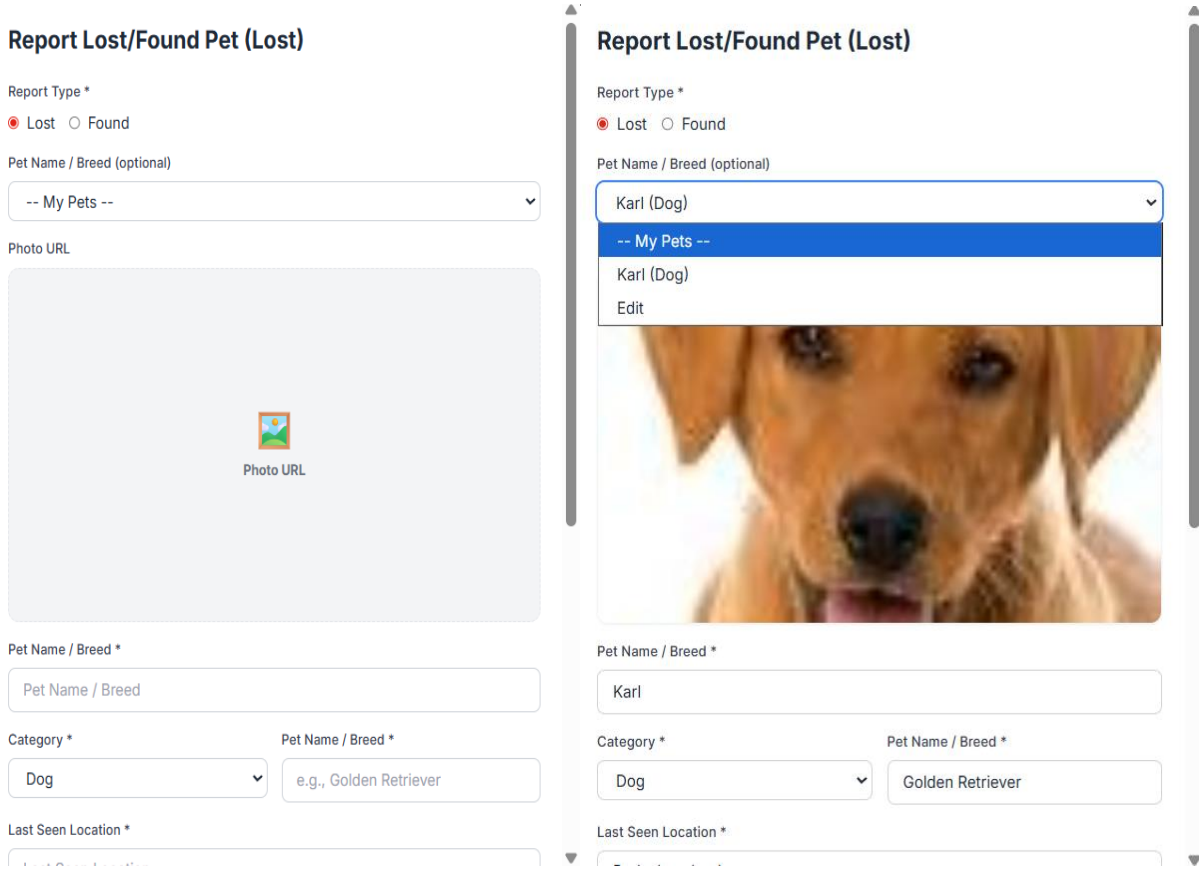


Рисунок 3.7. Модуль подачі загубленої домашньої тварини

Абсолютно аналогічно до попередньої ситуації було зібрано і модуль із назвою Pet Profile. Його технічно реалізовано саме як відокремлений безпечний клієнтський простір, спеціально призначений для гнучкого керування

персональними даними про домашню тварину. Тут усі поточні введені параметри та текст так само тимчасово і безпечно зберігаються у браузерному localStorage.

Проте цього разу вони мають чітку математичну внутрішню прив’язку безпосередньо до актуального значення userId поточного власника тварини (це рішення добре ілюструється на малюнку 3.8). Слід наголосити, що таке гнучке технічне налаштування також застосовано виключно як суто тимчасове рішення. Воно ефективно працює лише до моменту повної фінальної та глибокої реалізації на бекенді великої серверної моделі з довгою назвою «питомці/медичні записи/події».

Таблиця 3.4

Порівняння способів збереження даних у модулях (DB через Prisma vs localStorage)

Характеристика	localStorage (Поточний стан фронтенду)	DB через Prisma (Бекенд)
Місце зберігання	Браузер користувача (локально)	Централізована база даних (на сервері)
Доступність даних	Тільки на тому ж пристрої та в тому ж браузері	З будь-якого пристрою після входу в акаунт
Спільний доступ	Дані бачить лише один користувач	Дані доступні всім (напр., пости на форумі)
Безпека	Низька (вразливість до XSS, дані у відкритому тексті)	Висока (захист через JWT, валідація на сервері)
Обсяг даних	Обмежений (~5-10 МБ)	Практично необмежений
Складність реалізації	Дуже низька (getItem/setItem)	Середня (потребує API-ендпоїнтів та схем Prisma)
Типи даних	Тільки рядки (потрібен JSON.stringify)	Складні типи, зв'язки (Relations), індекси
	Типи даних	Тільки рядки (потрібен JSON.stringify)

Під час роботи модуль Pet Profile зручно та інтуїтивно відображає головний інформаційний профіль тварини одразу з кількома окремими вкладками для навігації. До них, зокрема, згідно проєкту відносяться важливі розділи Health &

Visits, My Forum Posts та згаданий раніше розділ Lost & Found. Наприклад, для кожної конкретної обраної тварини зі списку там завжди доступна ключова медична інформація про її поточний статус складної вакцинації.

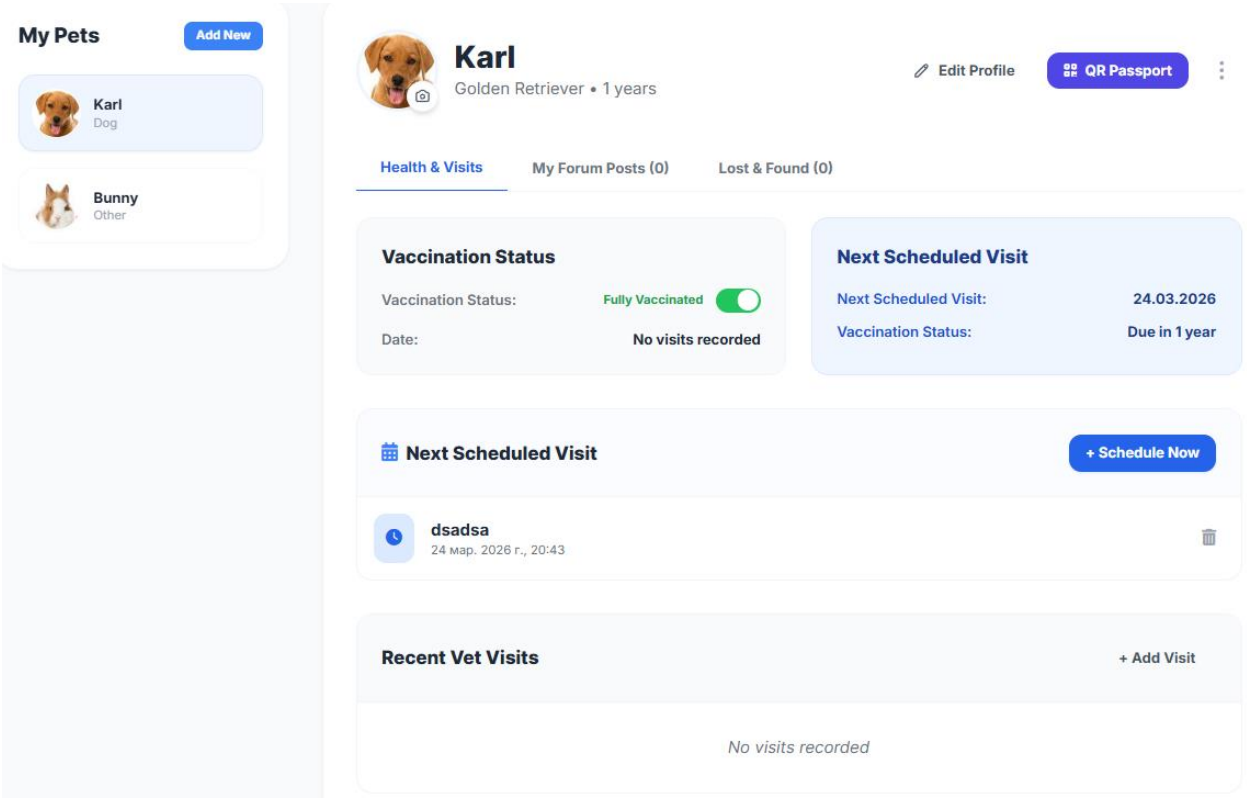


Рисунок 3.8. Модуль Pet Profile

Там же відслідковуються усі акуратно заплановані майбутні візити до ветеринарного лікаря або останні фактичні відвідування найближчої клініки за записами. Якщо ж є потреба у детальнішій візуальній демонстрації розроблених сценаріїв взаємодії типового користувача з цим модулем (таких операцій як безпосереднє додавання абсолютно нової тварини в систему, планування її чергового візиту, правильна валідація заповнених форм та навіть успішна генерація інтерактивного QR-паспорта безпеки), то всі відповідні скріншоти та схеми було повністю наведено у Додатку Б роботи (зокрема, див. рис. Б.3 - Б.8 та додатково рис. Б.25).

3.3 Реалізація авторизації та правил доступу

Для надійного гарантування безпеки та коректної автентифікації користувачів у системі було використано класичний механізм JSON Web Tokens (JWT) [11]. Він ефективно забезпечує цілісність переданих даних через використання цифрового підпису. Слід зазначити, що такий технічний підхід повністю відповідає чинним нормам Закону України «Про захист персональних даних» [21]. Зокрема, в частині захисту конфіденційної інформації від несанкціонованого доступу.

В свою чергу, у самих токенах свідомо зберігаються лише мінімальні необхідні ідентифікатори. Це суттєво мінімізує будь-які ризики витоку персональних даних безпосередньо під час мережевого обміну. На основі глибокої перевірки цих токенів у системі реалізовано продуману рольову модель доступу (RBAC). Вона дозволяє дуже гнучко розмежовувати повноваження між учасниками. Сама структура цієї моделі, а також конкретні права різних груп користувачів, більш детально описані нижче.

Рольова модель та правила доступу. У платформі було успішно реалізовано рольову модель доступу (RBAC). Базово вона спирається на наявність трьох ключових ролей для користувачів: - Користувач. Має право вільно створювати та редагувати власні особисті дані (наприклад, свій профіль, картки тварин, нагадування). Також він створює контент (нові пости, коментарі, публікує оголошення у Lost&Found) та просто переглядає загальну карту корисних місць; - Модератор. Ця роль має значно розширені повноваження щодо модерації загального користувацького контенту. Серед них: приховування, відхилення або автоматична публікація матеріалів, базова обробка поточних скарг, а також зміна статусів для оголошень у системі Lost&Found; - Адміністратор. Має абсолютно повні права в системі. Включно з глобальним керуванням ролями самих користувачів та будь-якими системними довідниками.

З точки зору програмної реалізації RBAC безпосередньо на backend, сама перевірка доступу ефективно виконується як проміжне програмне забезпечення

(middleware). Цей механізм працює за чітким алгоритмом. По-перше, він акуратно валідує поточний JWT або сесію. По-друге, витягує звідти userId та відповідну роль (role). І нарешті, він автоматично блокує або дозволяє подальше виконання запиту суворо згідно з прийнятими політиками доступу платформи.

Лістинг 3.5 - Фрагмент ROLES та типу Role (backend/src/utils/constants.ts)

```
 export const ROLES = {
   USER: 'USER',
   MODERATOR: 'MODERATOR',
   ADMIN: 'ADMIN',
 } as const;
export type Role = typeof ROLES[keyof typeof ROLES];
/**
 * Validates if a value is a valid role.*/
export const isValidRole = (value: string): value is Role => {
  return Object.values(ROLES).includes(value as Role);};
```

Представлена в лістингу кодова реалізація використовує важливе ключове слово as const. Воно потрібне для створення повністю незмінного об'єкта. Внаслідок цього система безпомилково динамічно виводить тип Role (а саме 'USER' | 'MODERATOR' | 'ADMIN'). Такий підхід блискуче забезпечує абсолютно сувору типізацію по всьому коду бекенда

Автентифікація з використанням JWT у заголовку authorization**.** Відразу після успішного входу користувача до системи, сервер самостійно формує захищений JWT-токен. Надалі клієнтська частина платформи обов'язково передає його у кожному своєму захищеному запиті. Це робиться через використання стандартного HTTP-заголовка у такому форматі: Authorization: Bearer <token>

Сама ж перевірка переданого токена виконується через спеціальний middleware з назвою requireAuth. Якщо токен дійсно є валідним та живим, цей middleware гарантовано декодує зашифровані дані користувача. Після цього він робить їх глобально доступними для всіх наступних обробників конкретного запиту (наприклад, просто зберігаючи їх у об'єкті req.user). Можна побачити реалізацію безпосередньо на лістингу номер 3.6.

Лістинг 3.6 - Фрагмент requireAuth (перевірка JWT та заповнення req.user) (backend/src/middleware/auth.ts)

```
export const requireAuth = async (req: Request, res: Response, next: NextFunction)
=> {
  const authHeader = req.headers.authorization;
```

```
  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ error: 'No token provided' });
  } try {
    const token = authHeader.substring(7);
    const decoded = jwt.verify(token, getJwtSecret()) as {
      userId: number; email: string; role: string;
    };req.user = { id: decoded.userId, email: decoded.email, role: decoded.role };
    next();
  } catch (jwtError) {
    return res.status(401).json({ error: 'Invalid or expired token' });}};
```

Інтерфейс адміністративної панелі містить чотири розділи: Dashboard із системною статистикою платформи, Place Proposals - черга пропозицій користувачів з можливістю погодження або відхилення, Forum Posts та Lost & Found для модерації контенту, а також розділ Users для керування обліковими записами. Зовнішній вигляд панелі адміністратора з чергою пропозицій представлено на рисунку 3.9.

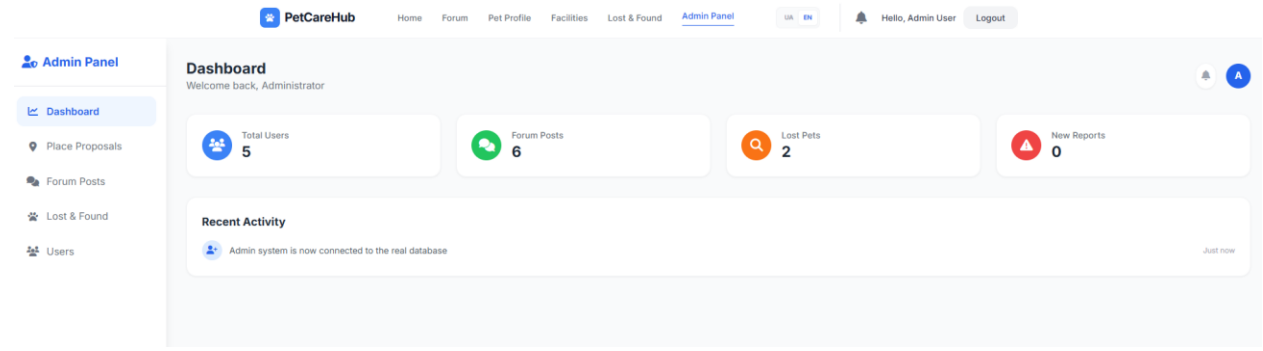


Рисунок 3.9. Модуль головної панелі Адміністратора

Реалізація middleware для перевірки ролей**.** З метою безпечного обмеження доступу виключно до адміністративних або модераторських дій, у системі було реалізовано перевірку ролей. Вона гнучко виконується через логічну обгортку requireRole(allowedRoles: Role[]). Ця функція дуже просто порівнює поточну роль користувача (яку було зчитано з req.user.role) із переліком допустимих ролей. Цей перелік завжди заздалегідь жорстко заданий для конкретного системного маршруту.

Отже, уся логіка RBAC безпосередньо на backend реалізована у два зрозумілих етапи:

1. Використовується базова перевірка requireAuth. Тобто система допускає до виконання лише тих користувачів, які вже мають валідну JWT-сесію;

2. Відпрацьовує перевірка requireRole([...]). Вона додатково аналізує, чи взагалі належить цей конкретний авторизований користувач до списку дозволених ролей (до прикладу, чи має він статус ADMIN або MODERATOR для певної закритої дії).

Лістинг 3.7 - Фрагмент requireRole (перевірка req.user.role) (backend/src/middleware/auth.ts)

```
export const requireRole = (role: string) => {
return (req: any, res: any, next: any) => {
    // Перевірка, чи користувач автентифікований та чи має потрібну роль
    if (!req.user || req.user.role !== role) {
      return res.status(403).json({
        message: 'Access denied: insufficient permissions.'
      });    }
    next(); // Користувач має роль, дозволяємо виконання наступного обробника};};
```

Матриця доступів та узгодження з реалізацією**.** З метою наочної та зручної формалізації усіх визначених політик доступу, в роботі успішно застосовується спеціальна зведена матриця (табл. 3.5). В свою чергу, вона дуже чітко та об'ємно відображає підсумкові цільові правила побудованої моделі RBAC безпосередньо для усіх основних системних дій у межах розробленої Платформи.

Таблиця 3.5

Матриця доступів (RBAC) для основних дій Платформи

Дія/модуль	Користувач	Модератор	Адміністратор
Перегляд публічних поств/оголошень	так	так	так
Створення постів/коментарів	так	так	так
Редагування/видалення власного поста/коментаря	так	так	так
Приховування/відхилення чужого контенту	ні	так	так
Створення/редагування профілю тварин	так	так	так
Модерація Lost&Found(approve/reject/hide)	ні	так	так
Призначення ролей користувачам	ні	ні	так

Фактично захищені ендпоїнти. Задля кращої наочності налаштувань безпеки, нижче наведено інформацію про поточні захищені ендпоїнти та суворі вимоги до ролей при їх використанні.

Таблиця 3.6

Захищені ендпоїнти та вимоги до ролей.

Модуль	Метод	Шлях	Захист	Дозволені ролі
Admin places proposals	GET	/api/admin/place-proposals/	requireAuth + requireRole	ADMIN, MODERATOR
Admin places proposals	POST	/api/admin/place-proposals/:id/approve	requireAuth + requireRole	ADMIN, MODERATOR
Admin places proposals	POST	/api/admin/place-proposals/:id/reject	requireAuth + requireRole	ADMIN, MODERATOR
Forum	POST	/api/posts/	requireAuth	USER, MODERATOR, ADMIN
Forum	DELETE	/api/posts/:id	requireAuth	USER, MODERATOR, ADMIN (з урахуванням правил "власний/чужий" у бізнес-логіці)
Forum	POST	/api/posts/:id/comments	requireAuth	USER, MODERATOR, ADMIN
Forum	POST	/api/posts/:id/like	requireAuth	USER, MODERATOR, ADMIN
Forum	POST	/api/posts/:id/like	requireAuth	USER, MODERATOR, ADMIN
Forum	DELETE	/api/posts/comments/:id	requireAuth	USER, MODERATOR, ADMIN (власний/чужий)
ace proposals	POST	/api/place-proposals/	requireAuth	USER, MODERATOR, ADMIN
Place proposals	GET	/api/place-proposals/my	requireAuth	USER, MODERATOR, ADMIN

В свою чергу, до переліку так званих відкритих публічних ендпоїнтів належать наступні:

– POST /api/auth/register, POST /api/auth/login (вони вільно відповідають за вхід та реєстрацію нових людей);

– GET /api/places/ (відповідає за перегляд місць на інтерактивній карті);

– GET /api/posts/ (дозволяє дивитись загальну стрічку публікацій форуму);

– GET /api/posts/:id (забезпечує детальний перегляд обраного поста).

Лістинг 3.8 - Фрагмент adminRoutes.ts із підключенням requireAuth та requireRole

```
const router = Router();
// Усі адмінські маршрути вимагають авторизації та ролі ADMIN або MODERATOR.
// Ці мідлвери застосовуються до всіх ендпоїнтів, визначених нижче.
router.use(requireAuth);
router.use(requireRole([ROLES.ADMIN, ROLES.MODERATOR]));
router.get('/place-proposals', getAdminProposals);
 router.post('/place-proposals/:id/approve', approveProposal);
 router.post('/place-proposals/:id/reject', rejectProposal);
 router.get('/users', getAllUsers);
 router.delete('/users/:id', deleteUser);
 router.patch('/users/:id/role', updateUserRole);
```

В системі присутні наступні ключові конфігураційні файли:

1. Внутрішній файл backend/src/routes/postRoutes.ts (модуль Форум). Тут для забезпечення великої функціональної гнучкості використовується відразу два підходи. Це optionalAuth спеціально для тих, хто просто публічно переглядає стрічку, а також класичний requireAuth вже безпосередньо для можливості виконання дій самим користувачем.

Лістинг 3.9 - Фрагмент postRoutes.ts / placeProposalRoutes.ts із підключенням requireAuth

```
 // Публічні маршрути (optionalAuth дозволяє бачити 'isLiked', якщо токен є)
router.get('/', optionalAuth, getPosts);
router.get('/:id', optionalAuth, getPostById);
// Захищені маршрути (тільки для авторизованих користувачів)
router.post('/', requireAuth, createPost);
router.delete('/:id', requireAuth, deletePost);
router.post('/:id/like', requireAuth, likePost);
router.post('/:id/comments', requireAuth, addComment);
```

2. Внутрішній файл backend/src/routes/proposalRoutes.ts (модуль Пропозиції місць). Абсолютно всі маршрути в цьому конкретному файлі строго і завжди потребують авторизації. Більше того, для створення нових пропозицій сюди додатково було прописано і додано спеціальний лімітер запитів. Це зроблено для елементарного захисту від спаму.

```
const router = Router();
// Захищені маршрути для подачі та перегляду власних пропозицій
router.post('/', requireAuth, proposalLimiter, createProposal);
router.get('/my', requireAuth, getMyProposals);
export default router;
```

Забезпечення цілісності даних у модулі admin (транзакції Prisma). Важливо також обов'язково відмітити спосіб безпечної реалізації досить складних адміністративних операцій (зокрема мова йде про процес погодження запропонованого місця модератором). Вони професійно реалізовані з широким використанням так званих атомарних транзакцій у Prisma (prisma.$transaction).

Саме це серйозне технічне рішення гарантує збереження абсолютної узгодженості даних. Особливо у тих випадках, коли одна системна дія неминуче включає в себе відразу кілька послідовних змін у БД. Наприклад, як при ситуації, коли йде одночасне створення запису в сутності Place та паралельне оновлення його статусу в PlaceProposal. Що це дає на практиці? Відповідь проста: у разі виникнення будь-якої, навіть найменшої програмної помилки процесу, внесені часткові зміни ні за яких обставин не фіксуються в пам'яті. Таким чином, система дуже ефективно запобігає небезпечній і болючій десинхронізації станів у базі даних.

Лістинг 3.10 - Фрагмент prisma.$transaction у adminController.ts (approve/reject)

```
const result = await prisma.$transaction(async (tx) => {
 // 1. Створення об'єкта Place (нової локації на мапі)
 const newPlace = await tx. place.create({
   data: { name: proposal.name, category: proposal.category, lat: proposal.lat,
lng: proposal.lng,
     address: proposal.address, createdById: proposal.authorId, approvedById:
actorId, approvedAt: new Date() }
 });
 // 2. Оновлення статусу пропозиції на APPROVED та прив'язка до нової локації
 const updatedProposal = await tx.placeProposal.update({
```

```
   where: { id: proposalId },
   data: { status: 'APPROVED', reviewedById: actorId, reviewedAt: new Date(),
placeId: newPlace.id }});
 // 3. Створення нотифікації користувачу, що подав пропозицію
 await tx.notification.create({
   data: { userId: proposal.authorId, type: 'PROPOSAL_APPROVED',
     message: `Your place "${proposal.name}" has been approved and is now on the
map!` }
 });
 return { place: newPlace, proposal: updatedProposal };
});
```

3.4 Реалізація бази даних і моделі сутностей

Для того, щоб повністю та якісно покрити потреби усіх розроблених функціональних модулів, у базі даних було спроєктовано та реалізовано цілу низку ключових сутностей. Зокрема, до цього переліку увійшли наступні:

– User. Вона відповідає за безпечне збереження персональних даних користувача. Сюди входять його ідентифікатор (або зовнішній провайдер авторизації), актуальний email, ім’я, посилання на аватар та відповідна системна роль;

– Pet. Ця сутність призначена для зберігання детального профілю домашньої тварини. Вона містить такі поля як кличка, стать, точна порода, завантажене фото та вказана дата народження;

– Vaccination. Містить усі робочі записи про зроблені щеплення. Зберігає тип застосованої вакцини, дату її введення, наступну очікувану дату візиту та додаткові текстові нотатки;

– CareEvent. Фіксує різні події рутинного догляду за твариною (наприклад, факт вигулу, години годування або ж разовий візит до лікаря тощо);

– Reminder. Відповідає за системні нагадування для користувача. Ця модель зберігає тип нагадування, точний час його спрацювання, параметри повторюваності та обрані канали доставки;

– PushSubscription. Спеціальна сутність для збереження підписок браузера. Вона необхідна для коректної відправки повідомлень за технологією Web Push;

– ForumPost та ForumComment. Використовуються для збереження безпосередньо контенту форуму платформи та залишених коментарів під ним;- LostFoundPost. Зберігає створені оголошення у розділі Lost/Found, а також їхні поточні статуси модерації;

– ModerationAction. Сутність, що призначена для ведення внутрішнього журналу модерації. Вона детально фіксує хто саме, яку дію зробив, і вказує причину такого рішення.

Також варто зазначити, що у контексті роботи із сутностями локацій у системі було налаштовано відповідні маршрутизатори запитів. По-перше, це маршрути самих локацій (placeRoutes.ts). Ці ендпоїнти безпосередньо відповідають за відкритий перегляд вже існуючих затверджених точок на загальній мапі. Крім того, вони обробляють запити на пряме додавання абсолютно нових об'єктів (звичайно, якщо це дозволено системними правами користувача).

На наступній сторінці наведено короткий опис тест-кейсів для цих маршрутів:

– GET /api/places (Доступ: Публічний, Роль: Будь-хто). Очікуваний результат: 200 OK - успішне отримання повного списку всіх верифікованих локацій.

– GET /api/places/:id (Доступ: Публічний, Роль: Будь-хто). Очікуваний результат: 200 OK - правильне отримання детальної інформації про одну конкретну точку за її ID.

– POST /api/places (Доступ: requireAuth, Роль: User). Очікуваний результат: 401 Unauthorized - закономірна відмова системи при спробі створити нову точку без переданого токена безпеки.

По-друге, це маршрути пропозицій нових місць (proposalRoutes.ts). Вони активно використовуються самими користувачами для подання спеціальних запитів на додавання нових корисних об'єктів у систему. Наприклад, це можуть бути різноманітні Pet-friendly кафе, тренувальні майданчики, зоомагазини тощо.

Обмеження цілісності та валідація. З метою забезпечення абсолютної коректності збережених даних, у архітектурі платформи було вдало поєднано

відразу два різні підходи до перевірок. Насамперед, це сувора програмна валідація даних безпосередньо на рівні самого API. Вона автоматично перевіряє передані типи даних, наявність усіх обов'язкових полів, допустимі діапазони введення дат та коректний формат електронної пошти. Наступним кроком йдуть додаткові жорсткі фізичні обмеження вже на рівні самої БД. Вони потрібні спеціально для того, щоб навіть у разі виникнення якоїсь непередбаченої помилки в коді програми, записані дані ніколи не стали неконсистентними. Це надзвичайно важливий системний запобіжник.

В свою чергу, як конкретні приклади таких принципових технічних обмежень у базі можна навести наступні:

– для поля users.email примусово встановлено обмеження UNIQUE NOT NULL, що унеможливлює створення дублікатів акаунтів;

– для поля pets.owner_id використано зовнішній ключ (FK), який посилається на users.id із завчасно налаштованою каскадною поведінкою. Вона чітко узгоджена з бізнес-логікою додатка (зокрема, зазвичай тут використовується правило RESTRICT, щоб випадково та безповоротно не видалити важливу історію медичних записів);

– спеціально для великих модулів Lost&Found та загального форуму всі робочі статуси програмно реалізовано як перелік фіксованих значень. Тобто, технічно це enum у генераторі Prisma, а також додатковий CHECK/enum безпосередньо у самій СУБД PostgreSQL;

– задля забезпечення високої швидкості роботи системи, для найчастіших системних запитів було створено відповідні індекси. Зокрема йдеться про такі поля як forum_post.created_at, lostfound_post.status та, звичайно, reminder.scheduled_at. Завдяки цьому пошук потрібної інформації відбувається практично миттєво.

3.5 Реалізація ключових сценаріїв (нагадування, Lost&Found)

Дані програмні сценарії безпосередньо описують найбільш типові взаємодії звичайного користувача з Платформою. Саме завдяки їм вдається надійно та комплексно перевірити повноту спроєктованої моделі даних, коректну роботу API та дієвість налаштувань RBAC на практиці. Нижче наведено кілька таких прикладів. В свою чергу, вони виступають базовими сценаріями для проведення фінального тестування і наочної демонстрації працездатності всієї системи.

Сценарій створення нагадування та доставки сповіщення (Web Push + email). Щоб гарантувати стовідсоткову надійність роботи механізму сповіщень, при проєктуванні системи було вирішено логічно розділити цей процес на два окремих етапи. По-перше, відбувається саме створення користувачем нагадування та його безпечна фіксація в БД. По-друге, вже сформоване сповіщення відправляється до адресата окремим фоновим диспетчером (або так званим воркером) суворо на основі збережених у базі даних.

Такий структурний підхід дозволяє повністю уникнути неприємних ситуацій, коли важливе нагадування просто “загубилося”. Наприклад, через раптовий перезапуск сервера чи тимчасову недоступність зовнішнього каналу зв'язку. Більше того, у межах безпосередньої роботи на базі PostgreSQL було вдало використано підхід керування конкурентним виконанням. Технічно це реалізовано як “вибірка з блокуванням” за допомогою спеціальної SQL-команди SELECT FOR UPDATE SKIP LOCKED. Вона надійно гарантує, що два різні екземпляри воркера ніколи одночасно не оброблять одне й те саме нагадування.

З метою ретельної перевірки коректності доставки таких email-сповіщень у процесі розробки був спеціально використаний зручний тестовий поштовий сервіс Mailtrap. Відповідна візуальна реалізація цієї перевірки представлена на рисунку 3.10.

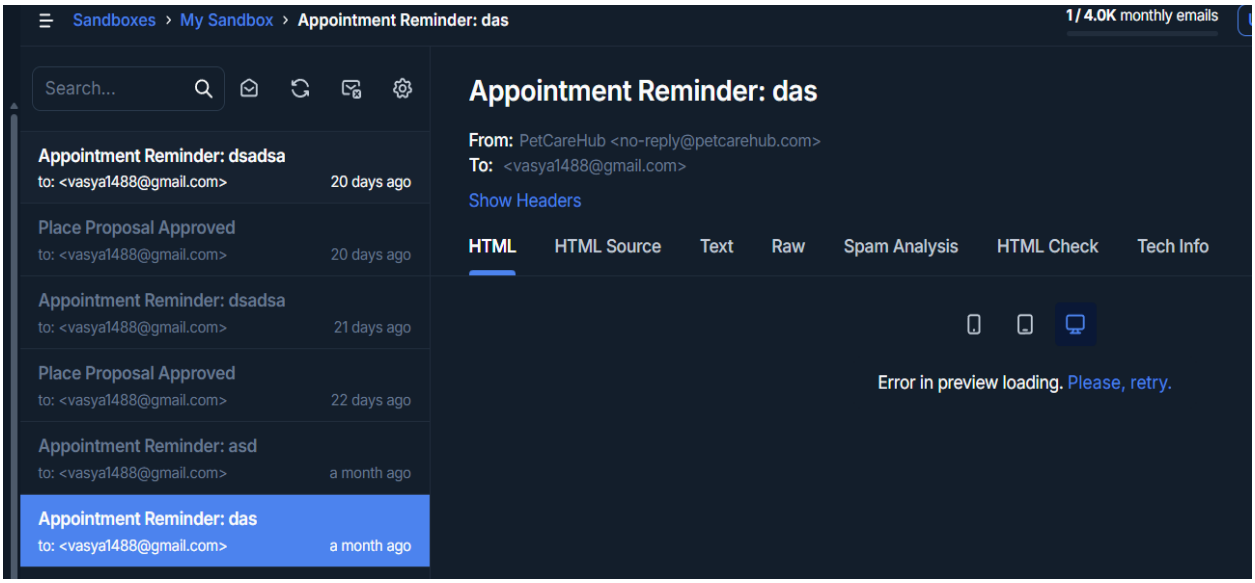


Рисунок 3.10. Доставка email-сповіщень користувачу

Сам інтерфейс платформи Mailtrap наочно підтверджує, що відправка системних листів (зокрема, таких категорій, як Appointment Reminder та Place Proposal Approved) на адресу тестового користувача відбувається абсолютно успішно. Це беззаперечно свідчить про гнучку і повністю коректну поточну роботу серверного модуля email-нотифікацій.

Сценарій публікації Lost&Found з модерацією. Оскільки розділ Lost&Found передбачає обов'язкову та сувору модерацію створеного контенту, у логіку роботи бази було закладено відповідні статуси для нових оголошень. На практиці цей життєвий алгоритм виглядає доволі лінійно та прозоро. Спочатку щойно створений запис автоматично отримує початковий статус PENDING. Далі, після своєї перевірки, модератор обов'язково змінює його або на APPROVED (схвалено і відправлено в стрічку), або ж на REJECTED (відхилено).

Також варто зазначити, що було передбачено додатковий кризовий статус HIDDEN. Він миттєво застосовується системою у разі виявлення грубих порушень з боку користувача вже після факту публікації. В свою чергу, використання таких чітких статусних маркерів дозволяє надійно зберегти усю внутрішню історію дописів. Завдяки цьому забезпечується повна і відкрита прозорість дій з боку самих модераторів перед іншими учасниками платформи.

3.6 Тестування та аналіз результатів

Безпосереднє тестування описаних у попередніх підрозділах механізмів суворо базується на відомому принципі «найменших привілеїв». Тобто, абсолютно всі критичні для функціонування системи операції (як-от створення нових користувацьких постів чи подача пропозицій щодо локацій) вимагають обов'язкової попередньої перевірки переданого JWT-токена. Під час самого тестування основний технічний акцент закономірно робиться саме на чіткій валідації статусів доступу. Зокрема, ідеально відпрацьовується поведінка сервера при видачі коду 401 Unauthorized для повністю анонімних і невідомих запитів. Водночас обов'язково перевіряється і статус 403 Forbidden. Він неодмінно має виникати за будь-якої спроби маніпуляції чужими персональними даними або ж у разі намагань отримання несанкціонованого доступу до закритих адмін-функцій.

Окрім перевірок доступу, окремим і дуже важливим етапом тестування є перевірка інтегрованих механізмів захисту від спаму (так званий Rate Limiting). Його наявність якісно гарантує стабільність роботи нашого API навіть при цілеспрямованій великій кількості інтенсивних запитів на створення нових ресурсів. Відповідно, такий строгий комплексний підхід надійно забезпечує загальну цілісність усіх збережених даних та відмінно захищає систему від будь-якого несанкціонованого втручання. Фрагмент перевірки можна побачити на листінгу 3.11

Лістинг 3.11 – Фрагмент тестування безпеки та контролю доступу (JWT, RBAC, Rate Limiting) у файлі security.test.ts

```
describe('Security & Access Control (Sections 3.4-3.5)', () => {
it('should return 401 Unauthorized for anonymous post creation', async () => {
  const res = await request(app).post('/api/posts').send({ title: 'Spam' });
  expect(res.status).toBe(401); // Перевірка відсутності JWT});
it('should return 403 Forbidden when deleting another user content', async () => {
  const post = await createTestPost(otherUser.id);
  const res = await request(app)
    .delete(`/api/posts/${post.id}`)
    .set('Authorization', `Bearer ${differentUserToken}`);
  expect(res.status).toBe(403); // Принцип найменших привілеїв});
it('should return 429 Too Many Requests on proposal spam', async () => {
  for (let i = 0; i < 6; i++) { // Поза лімітом у 5 запитів
```

```
    const res = await request(app).post('/api/proposals').set('Authorization',
`Bearer ${token}`);
    if (i === 5) expect(res.status).toBe(429); // Перевірка Rate Limiting }});});
```

Разом з тим, на рисунку 3.11 можна досить наочно побачити вже зафіксований успішний результат проходження фінальних тестів спеціально для серверної частини (backend-у) платформи. Натомість на рисунку 3.12, відповідно, зображено статус пройдених перевірок уже безпосередньо по клієнтській стороні (frontend).

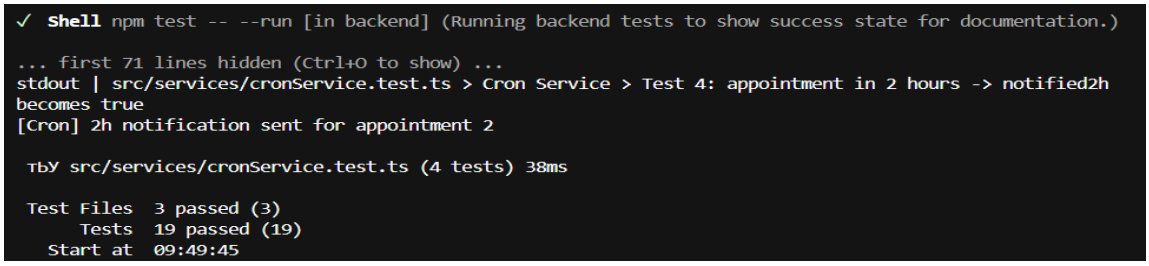


Рисунок 3.11. Протокол успішного тестування API та механізмів авторизації

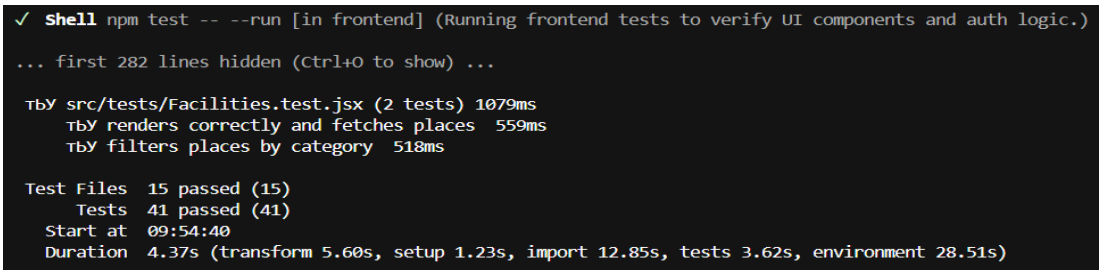


Рисунок 3.12. Верифікація компонентів інтерфейсу: перевірка захищених роутів та валідація форм

Отже, отримані логічні результати проведеного тестування стовідсотково підтверджують загальну коректність практичної реалізації системи зручної авторизації та суворого розмежування доступу на всіх рівнях спроєктованої платформи. В першу чергу, налаштований механізм middleware-ланцюжка (а саме зв'язка requireAuth → requireRole) чудово забезпечує швидку та послідовну перевірку всіх прав користувача. При чому, що найважливіше, це робиться без жодного непотрібного дублювання бізнес-логіки у самих контролерах. В свою чергу, постійне використання інструменту транзакцій prisma.$transaction безапеляційно гарантує необхідну атомарність при виконанні дійсно критичних адміністративних операцій. В цілому, сукупність абсолютно всіх успішно

проведених перевірок - починаючи від базової валідації складених JWT-токенів і аж до глибокого тестування інтерактивних компонентів інтерфейсу - яскраво свідчить про високу готовність розробленої вебплатформи PetCareHub до етапу активної дослідної експлуатації вже в суворих умовах реального навантаження.

3.7 Висновки до розділу 3.

У даному розділі було досить детально розглянуто та крок за кроком описано процес безпосередньої практичної реалізації створеної інформаційної вебплатформи підтримки власників домашніх тварин. На основі вже проведеної роботи можна зробити наступні ключові висновки:

По-перше, для проєкту було успішно обрано та практично впроваджено сучасний технологічний стек розробки. Зокрема, використання бібліотеки React для клієнтського фронтенду у тісному поєднанні з серверною платформою Node.js та статично типізованою мовою TypeScript дозволило створити дійсно швидкий та надійний вебзастосунок формату SPA. В свою чергу, саме таке рішення стовідсотково забезпечило високу швидкість відгуку візуального інтерфейсу та загальну стабільність обробки запитів сервером.

По-друге, безпосередньо на практиці було реалізовано гнучку модульну архітектуру системи. Вдале поєднання вже повністю інтегрованих full-stack модулів (таких як Автентифікація, загальний Форум та Геомодуль) із тимчасовими функціональними прототипами на клієнті (це розділи Lost&Found та Профіль тварини) дало змогу побудувати платформу з дуже чітким розмежуванням бізнес-логіки. Також варто зазначити, що активне використання генератора Prisma ORM суттєво спростило поточну взаємодію з локальною базою даних SQLite. Більше того, це відразу заклало міцний технічний фундамент для майбутньої швидкої міграції проєкту на потужну СУБД PostgreSQL.

По-третє, під час розробки особлива увага приділялася питанням безпеки та правильному розмежуванню прав доступу. Завдяки використанню класичного механізму JWT-токенів та робочій рольовій моделі (RBAC) вдалося досягти дуже непоганого рівня захисту персональних даних усіх користувачів. При цьому,

безпосередня реалізація middleware-ланцюжків на самому сервері дозволила дуже комфортно і централізовано контролювати доступ людей до закритих адміністративних функцій платформи. Тобто розробнику взагалі не довелося дублювати однакові перевірки у різному коді. Нарешті, загальна ефективність розробленої системи була успішно підтверджена шляхом її комплексного тестування. Проведені перевірки серверного API, а також паралельна верифікація графічного інтерфейсу клієнта, наочно довели правильність роботи всіх ключових сценаріїв застосунку.

ВИСНОВКИ

Безпосередньо у результаті виконання даної кваліфікаційної роботи було детально досліджено поточний стан інформаційної підтримки власників домашніх тварин. Під час аналізу встановлено, що разом зі стрімким зростанням загальної кількості улюбленців закономірно підвищується і потреба у зручних, повністю інтегрованих цифрових рішеннях. В свою чергу, виявлено, що існуючі на сьогодні підходи зазвичай базуються на використанні дуже розрізнених ресурсів. Тобто вони банально не здатні забезпечити дійсно належного рівня ефективності та оперативності взаємодії для кінцевого користувача.

Також проведений глибокий аналіз предметної галузі та наявних популярних міжнародних платформ показав одну важливу деталь. Вони далеко не повною мірою враховують специфічні локальні особливості саме українського ринку. Зокрема, це стосується мовних, організаційних та регіональних аспектів. Додатково було чітко встановлено, що в поточних складних умовах воєнного стану значно зростає критична потреба в інструментах дуже оперативного реагування. Насамперед мова йде про швидкий пошук загублених тварин, загальну координацію локальної волонтерської діяльності та легкий доступ до базових ветеринарних послуг.

Виходячи з цього, у ході роботи було визначено та детально сформовано основні вимоги до майбутньої інформаційної вебплатформи. При проєктуванні розроблено її загальну архітектурну структуру та логічно виділено ключові функціональні модулі системи. Окрема технічна увага при цьому приділялася створенню максимально зручного користувацького інтерфейсу. Адже саме він безпосередньо забезпечує ефективну та комфортну взаємодію людей із запропонованим цифровим рішенням.

Відповідно, головним практичним результатом роботи стала безпосередня розробка та фізична реалізація інформаційної вебплатформи. Дана система надійно забезпечує збереження та безпечну обробку персональних даних про домашніх тварин. Також вона надає швидкий доступ до необхідної сервісної

інфраструктури та повноцінно підтримує соціальну взаємодію між самими користувачами. Серед іншого, у функціоналі передбачено зручну можливість використання платформи для розміщення екстрених оголошень у розділі Lost&Found. Це технічне рішення значущо підвищує загальну швидкість реагування спільноти у відповідних форс-мажорних ситуаціях.

На фінальному етапі розроблена система успішно пройшла комплексне тестування. Отримані результати стовідсотково підтвердили коректність її поточного функціонування та абсолютну відповідність усім поставленим на початку вимогам. Якщо ж говорити про практичне значення отриманих результатів, то воно полягає у готовності використання розробленої Платформи як дійсно ефективного інструменту для підтримки власників тварин в Україні. Крім того, проєкт вже має закладений потужний технічний потенціал для свого подальшого масштабування. В майбутньому це дозволить легко інтегрувати систему з діючими волонтерськими центрами та спеціалізованими ветеринарними організаціями.

СПИСОК ВИКОРИССТАНОЇ ЛІТЕРАТУРИ

1. Petkey Pro - Petkey ™. *Pet Microchip ID Lookup & Registration for Dogs & Cats - Petkey ™*. URL: https://petkey.org/Pro (date of access: 07.02.2026).

2. Veterinary Software Built to Reclaim Time and Connect With Clients. *Veterinary Software Built to Reclaim Time and Connect With Clients*. URL: https://petdesk.com/ (date of access: 07.02.2026).

3. База реєстрації домашніх тварин. *Pet registration platform | Animal-id.net*. URL: https://animal-id.net/ua/ (дата звернення: 07.02.2026).

4. Vetted Pet Health | Daily Pet Care App with Rewards for Dog Owners. *Vetted Pet Health | Daily Pet Care App with Rewards for Dog Owners*. URL: https://www.vettedpethealth.com/ (date of access: 07.02.2026).

5. URL: https://www.rover.com/.

6. Реєстр домашніх тварин (Київська міська державна адміністрація) - https://pets.kyivcity.gov.ua/ (date of access: 07.02.2026)

7. TrustedHousesitters - Find Pet Sitters & House Sits Worldwide - https://www.trustedhousesitters.com/ (date of access: 07.02.2026)

8. React Documentation. React - A JavaScript library for building user interfaces. URL: https://react.dev/ (date of access: 15.04.2026).

9. Leaflet - a JavaScript library for interactive maps. Leaflet - an open-source JavaScript library for mobile-friendly interactive maps. URL: https://leafletjs.com/%20 (date of access: 15.04.2026).

10. Prisma Documentation. Prisma | Next-generation ORM for Node.js and TypeScript. URL: https://www.prisma.io/docs (date of access: 15.04.2026).

11. JWT.io. JSON Web Tokens - jwt.io. URL: https://jwt.io/ (date of access: 15.04.2026).

12. Tailwind CSS Documentation. Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. URL: https://tailwindcss.com/docs (date of access: 15.04.2026).

13. Про ідентифікацію та реєстрацію тварин. Закон України від 15.12.2009 № 1776-VI. URL: https://zakon.rada.gov.ua/laws/show/1445-17

14. Node.js documentation. Node.js v20.x documentation. URL: https://nodejs.org/docs/latest/api/ (date of access: 15.04.2026).

15. OpenStreetMap Foundation. OpenStreetMap - the free wiki world map. URL: https://www.openstreetmap.org/ (date of access: 15.04.2026).

16. Vite Documentation. Vite - Next Generation Frontend Tooling. URL: https://vitejs.dev/guide (date of access: 04.05.2026

17. Vitest Documentation. Vitest - A blazing fast unit test framework powered by Vite. URL: https://vitest.dev/ (date of access: 04.05.2026)

18. Express.js Guide. Express - Fast, unopinionated, minimalist web framework for Node.js. URL: https://expressjs.com/ (date of access: 04.05.2026).

19. i18next Documentation. i18next - The internationalization framework for browser or any other javascript runtime. URL: https://www.i18next.com/ (date of access: 04.05.2026).

20. Fuse.js. Fuse.js - Lightweight fuzzy-search library in JavaScript. URL: https://www.fusejs.io/ (date of access: 04.05.2026

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

22. Державна служба України з питань безпечності харчових продуктів та захисту споживачів. Ідентифікація та реєстрація тварин. URL: https://dpss.gov.ua/ (дата звернення: 04.05.2026).

23. W3C. Web Content Accessibility Guidelines (WCAG) 2.1. URL: https://www.w3.org/TR/WCAG21/ (date of access: 04.05.2026).

24. Крискун, І. М., Зайченко, С. П., & Білавка, В. Б. Штучний інтелект у розробці і управлінні іт-проектами. організатори конференції, 214.

25. Lucide Icons. Lucide - Beautiful & consistent icons. URL: https://lucide.dev/docs (date of access: 04.05.2026).

26. Бондарчук, А. П., Срочинська, Г. С., & Твердохліб, М. Г. (2015). Основи інфокомунікаційних технологій. Навчальний посібник [Електронний

ресурс]. Державний університет теле-комунікацій, Київ.–2015.–76 с.–Режим доступу: http://www. dut. edu. ua/uploads/l_840_37756081. pdf, 5.

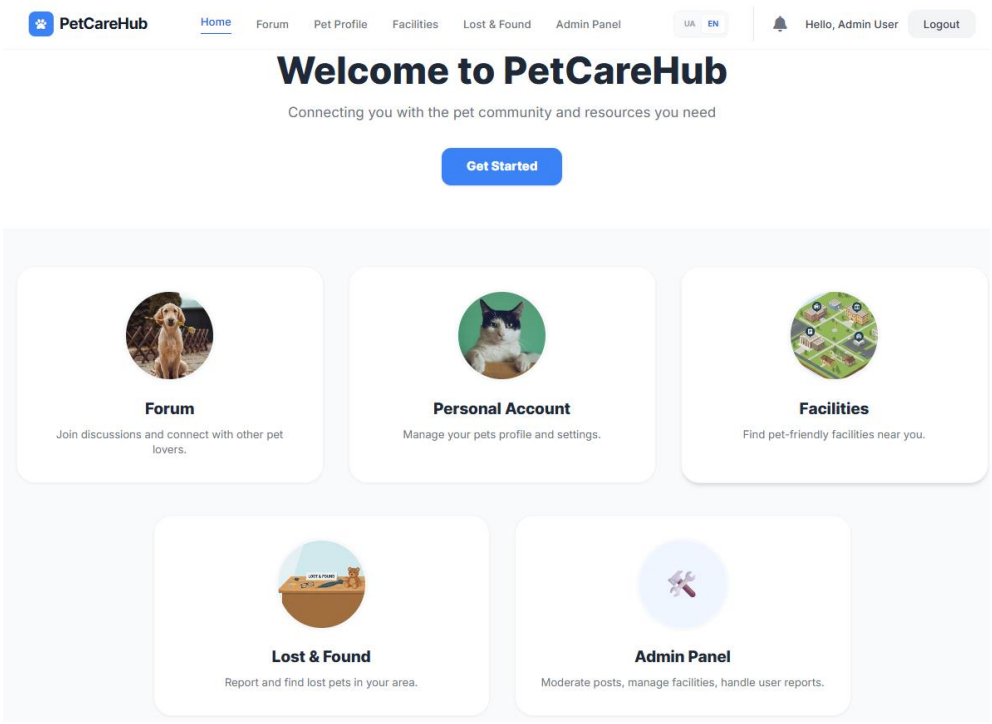
27. Zhurakovskyi, B., Toliupa, S., Druzhynin, V., Bondarchuk, A., & Stepanov, M. (2021). Calculation of quality indicators of the future multiservice network. In Future Intent-Based Networking: On the QoS Robust and Energy Efficient Heterogeneous Software Defined Networks (pp. 197-209). Cham: Springer International Publishing.

28. Галай, Я. О., Бондарчук, А. П., Ткаленко, О. М., Полоневич, О. В., & Зінченко, О. В. (2021). Розроблення системи оцінювання безпеки розумних будинків на основі IoT. Зв'язок, (1), 55-59.

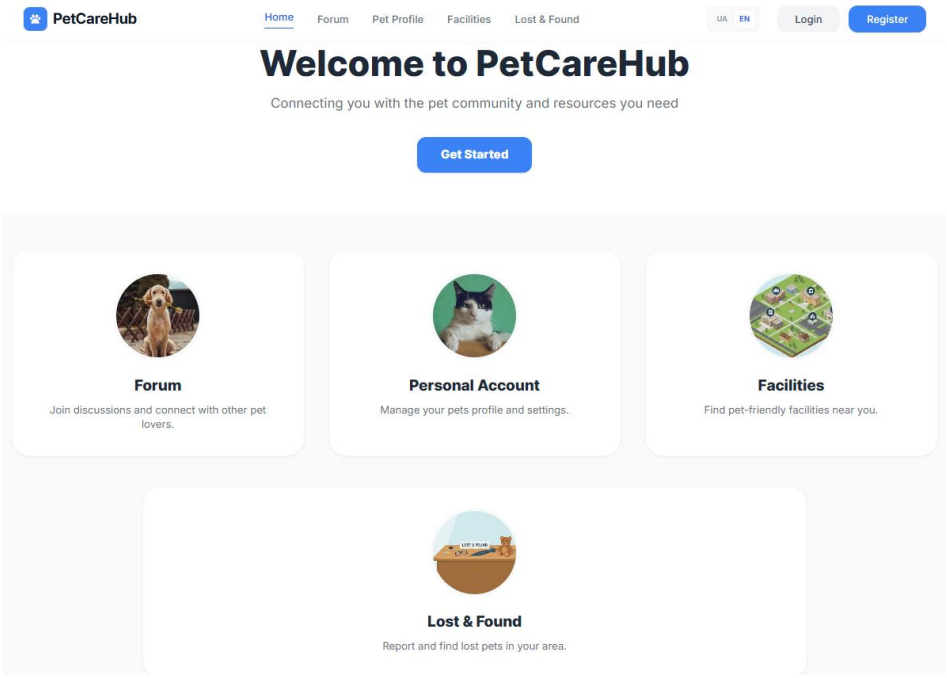
29. Abramov, V., Astafieva, M., Boiko, M., Bodnenko, D., Bushma, A., Vember, V., ... & Yaskevych, V. (2021). Theoretical and practical aspects of the use of mathematical methods and information technology in education and science.

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

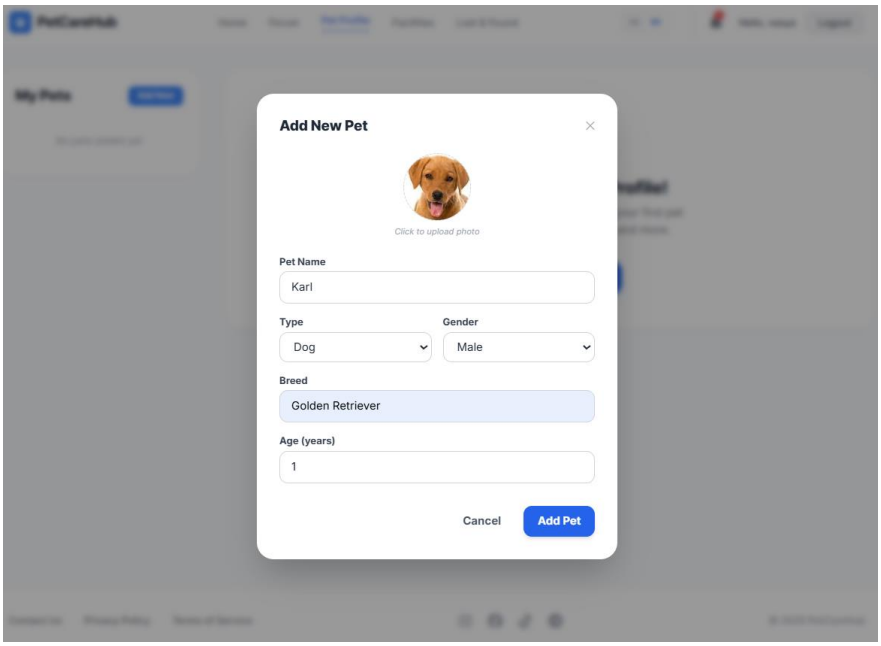
ДОДАТОК А



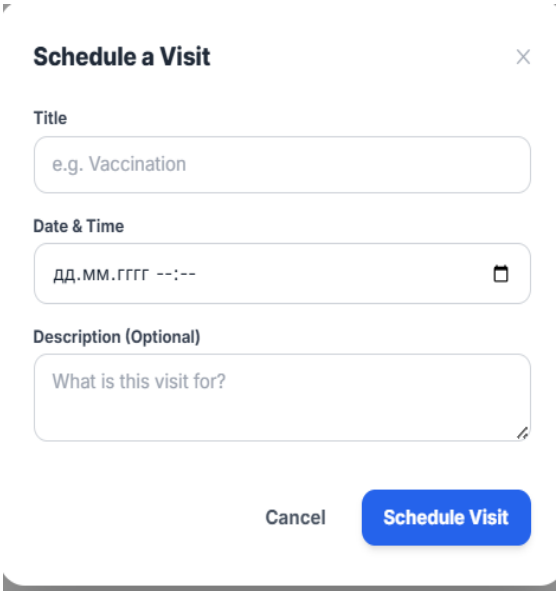
А.1.Головна сторінка(для адміна)



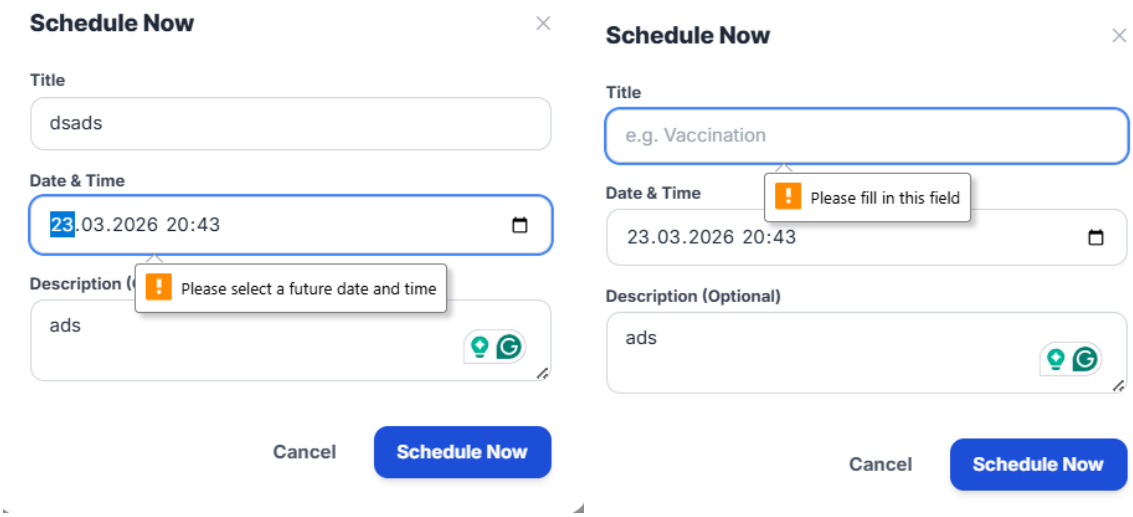
А.2.Головна сторінка(користувач)



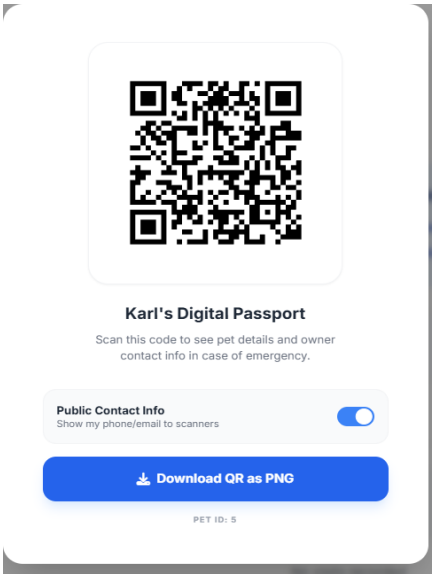
А.3.Модуль домашнього улюбленця(заповнення даних)



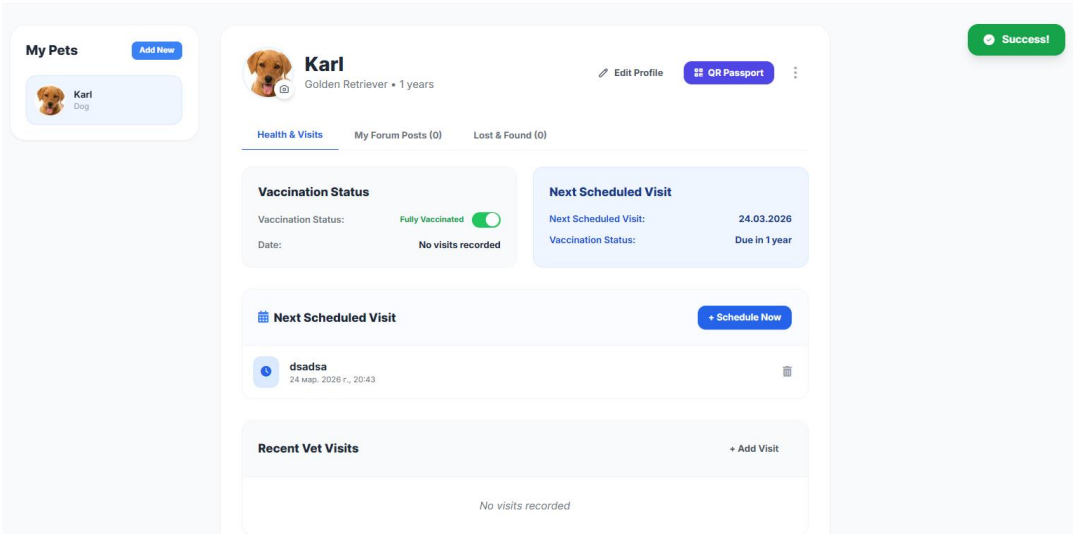
А.4.Модуль домашнього улюбленця(додавання візиту)



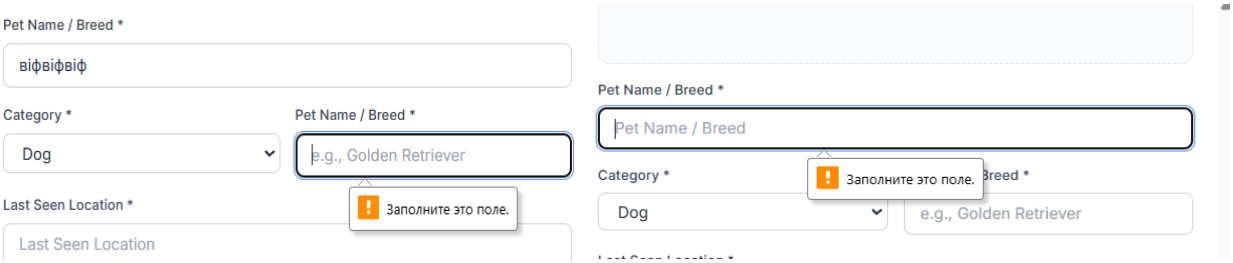
А.5.Модуль домашнього улюбленця(валідація даних)



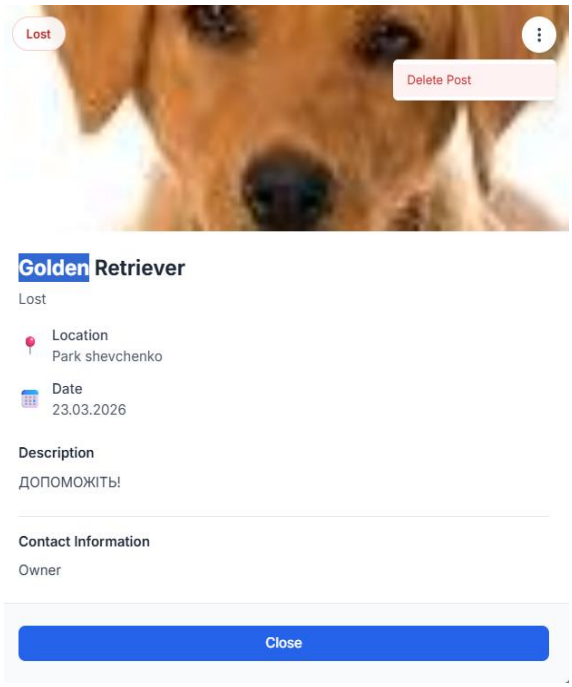
А.6.Модуль унікальний qr-код домашнього улюбленця



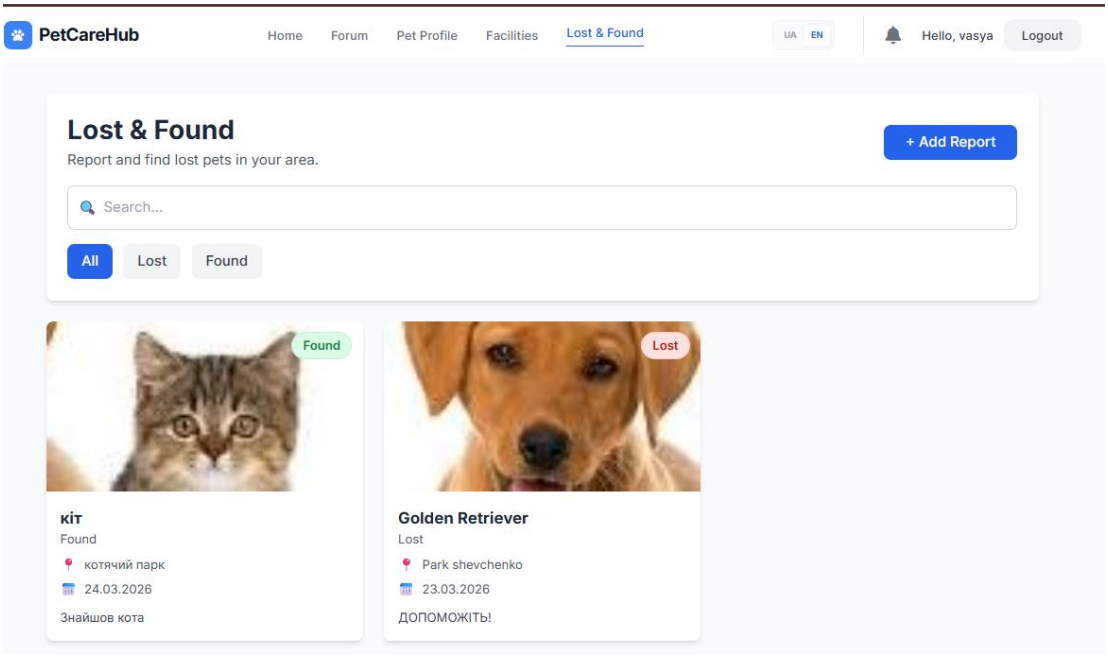
А.7.Модуль домашнього улюбленця



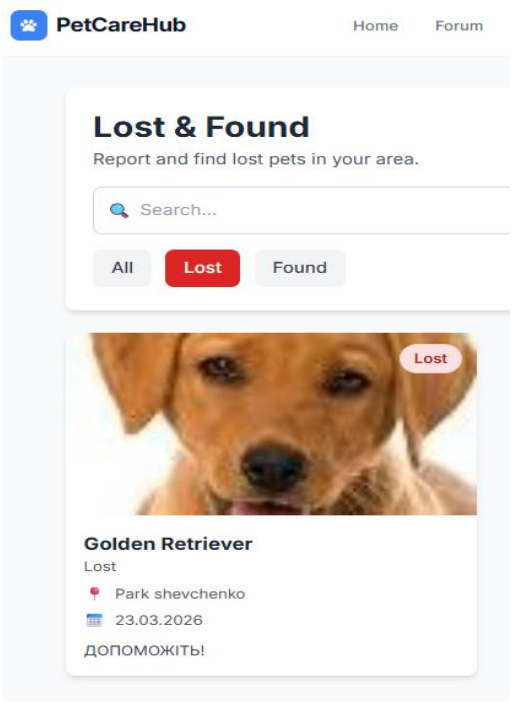
А.8.Модуль подачі загубленої домашньої тварини (валідація даних)



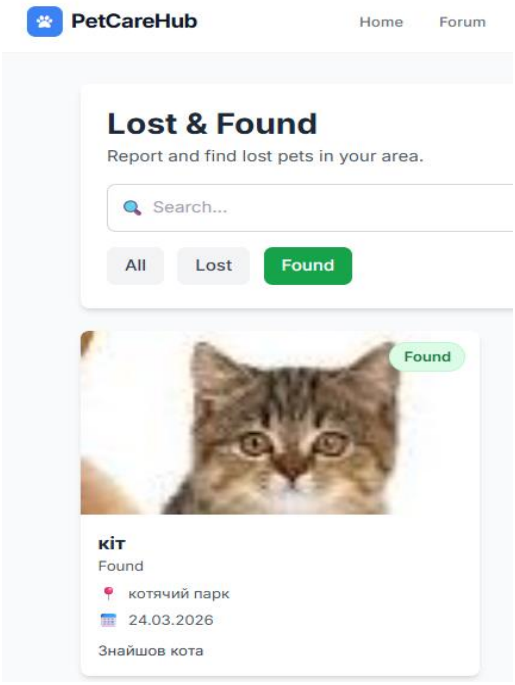
А.9.Модуль подачі загубленої домашньої тварини (валідація даних)



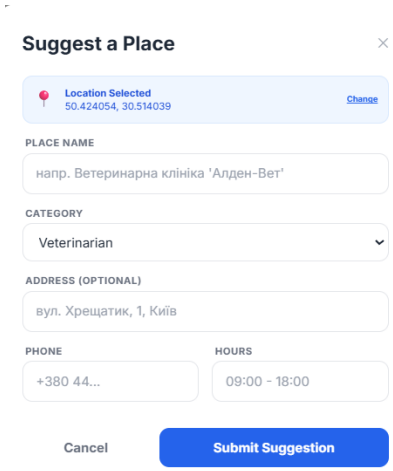
А.10.Модуль фільтрації репортів (всі)



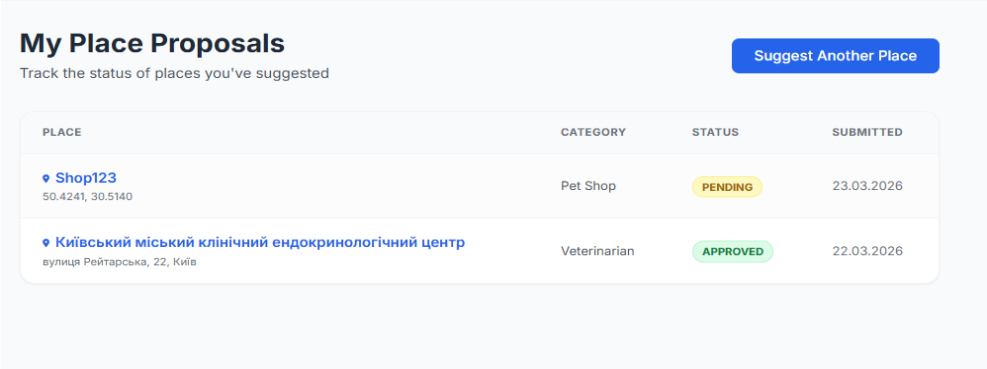
А.11.Модуль фільтрації репортів (загублені)



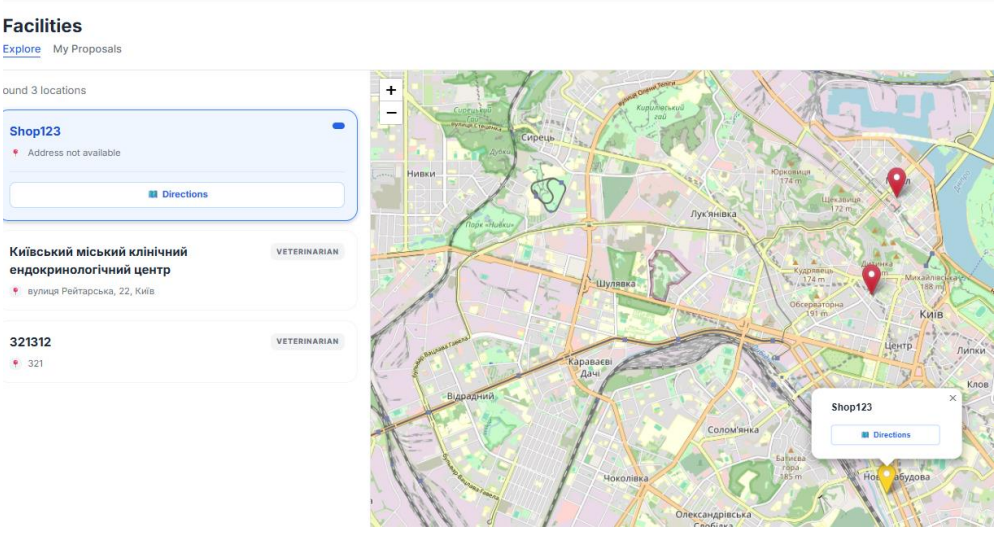
А.12.Модуль фільтрації репортів (знайдені)



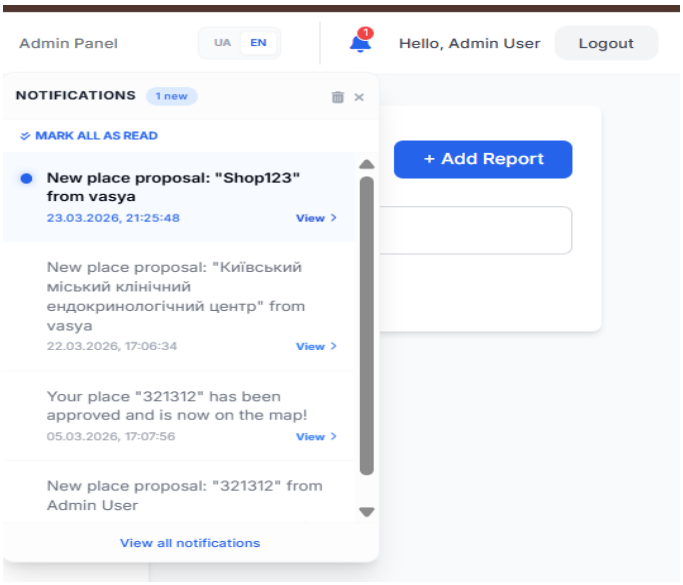
А.13.Модуль додавання точки на карті



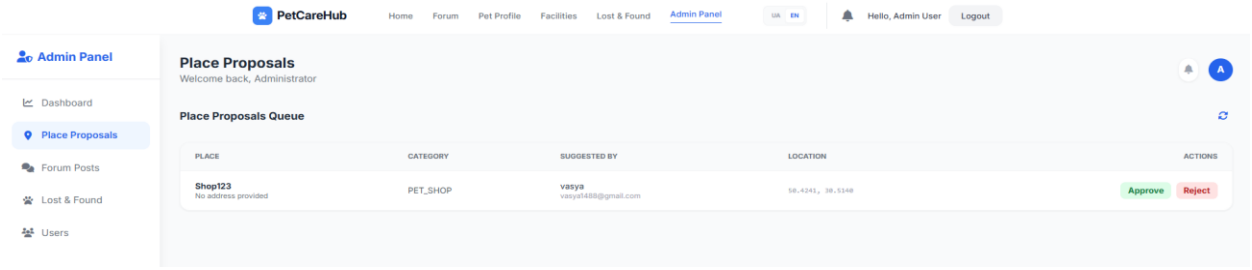
А.14.Модуль пропозиції місць на карті



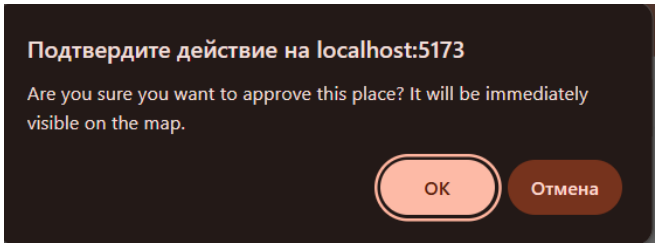
А.15.Продовження модуля пропозиції(відображення на карті)



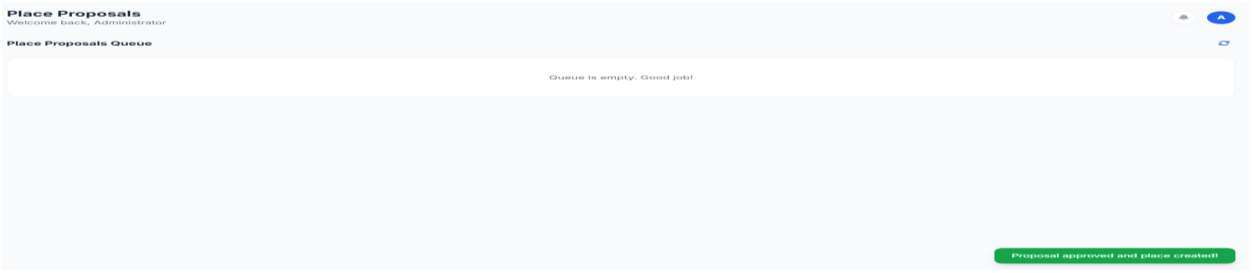
А.16.Модуль повідомлень



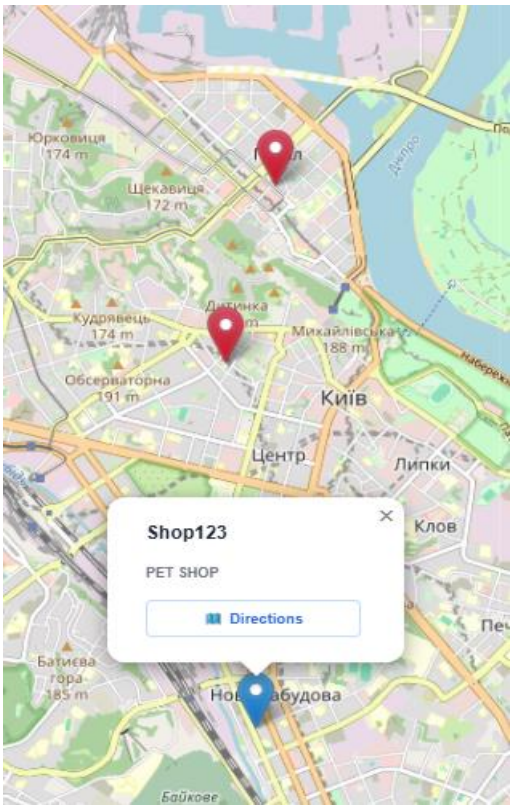
А.17.Модуль погодження місць на карті, адміністратором



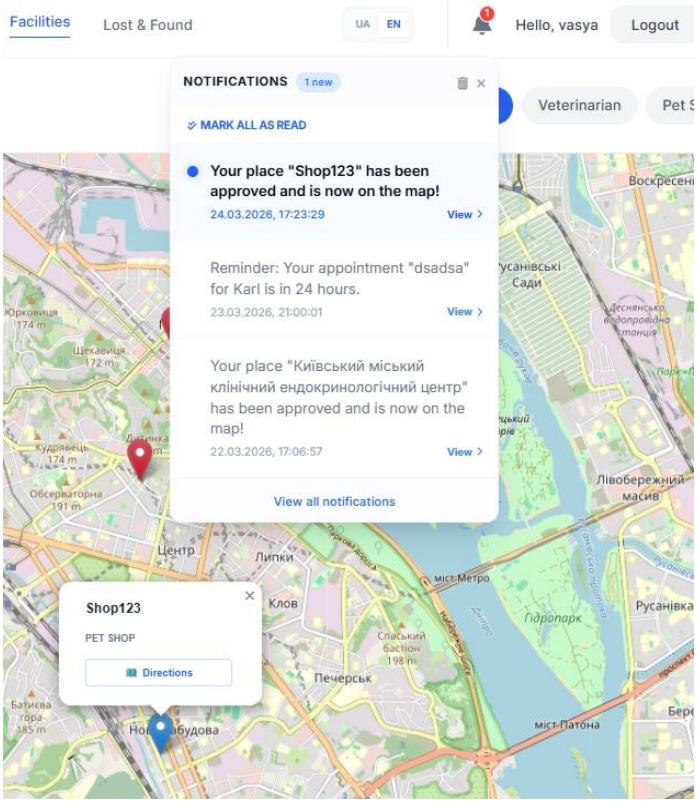
А.18.Редакція пропозицій на карті



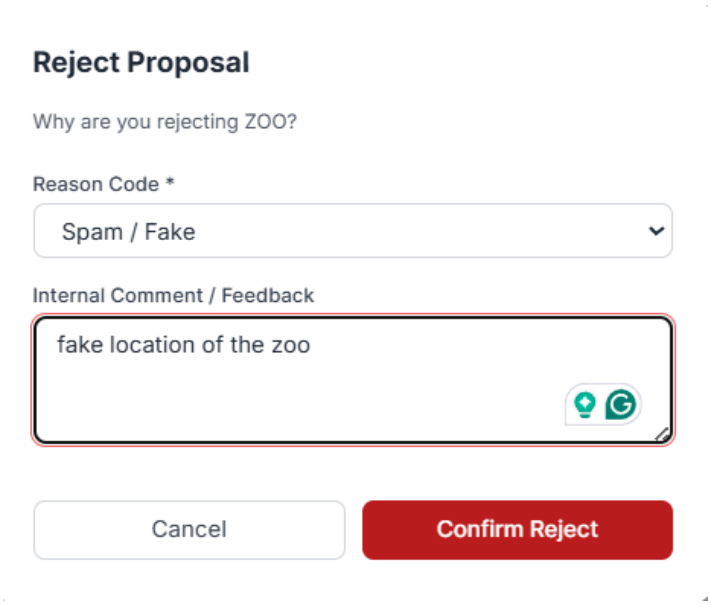
А.19.Продовження модуля редакції пропозицій на карті(успіх)



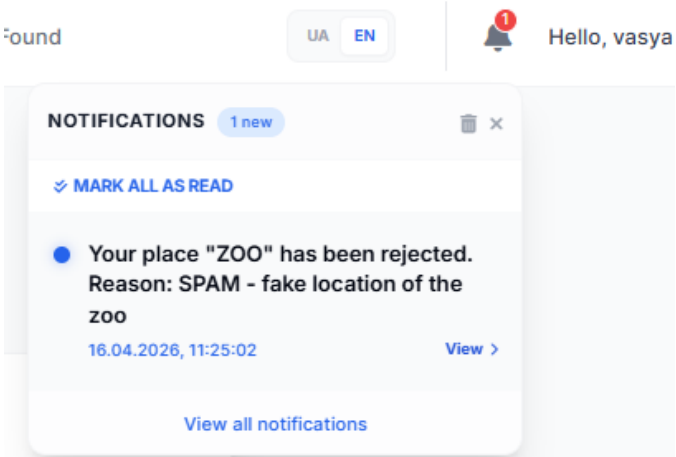
А.20.Відображення точки на карті



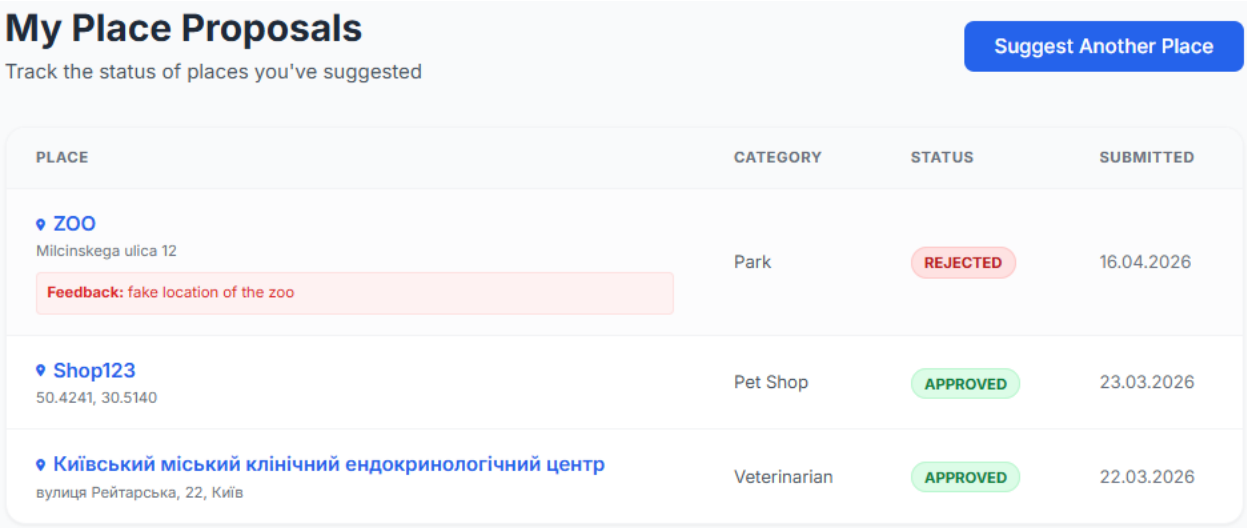
А.21.Сповіщення для користувача



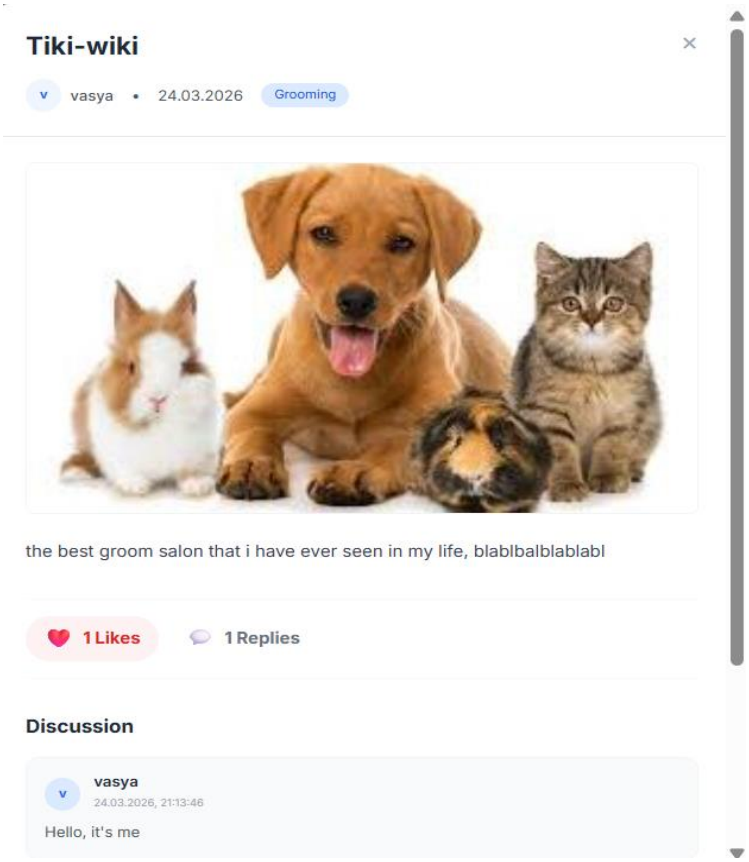
А.22.Модуль відхилення точки на карті адміністратором



А.23.Сповіщення для користувача про відхилення пропозиції



А.24.Відображення в профілі користувача пропозицій



А.25.Відображення поста на форуме