

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Платформа координації громадської допомоги
тваринам»**

Виконав студент групи ПЗ-22-2
ВЄЩИКОВ Олег Миколайович

Керівник роботи:
ВЛАСЕНКО Олег Васильович

Рецензент:
ЛОБАНЧИКОВА Надія Миколаївна

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії програмного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ВЄЩИКОВ Олег Миколайович**

Керівник роботи: **ВЛАСЕНКО Олег Васильович**

Тема роботи: **«Платформа координації громадської допомоги тваринам»,**

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., № 92**

Вихідні дані для роботи: **інформаційні технології з координації громадської допомоги тваринам. Використання веб-орієнтованих та мобільних технологій для створення крос-платформеної системи допомоги тваринам.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Власенко О.В.	01.03.2026р.	01.03.2026р
2	Власенко О.В.	01.03.2026р	01.03.2026р
3	Власенко О.В.	01.03.2026р	01.03.2026р

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з застосунку платформи допомоги тваринам FurryHelp та пояснювальної записки. Пояснювальна записка містить 3 розділи, 28 ілюстрацій, 18 таблиць, список із 30 використаного джерела та додатки.

Метою роботи є створення сучасного, зручного та функціонального веб-сервісу, що дозволяє користувачам публікувати оголошення про загублених та знайдених тварин, взаємодіяти з організаціями допомоги тваринам, отримувати геолокаційні сповіщення про тварин поблизу, а також спілкуватися в режимі реального часу.

Об'єктом дослідження є процес створення онлайн-платформи для допомоги безпритульним тваринам з елементами соціальної взаємодії та геолокаційного сервісу.

Предметом дослідження є методи розробки адаптивного вебзастосунку з реалізацією функцій геолокаційної карти, роботи з базами даних оголошень і організацій, авторизацією користувачів, системою сповіщень у реальному часі через WebSocket та Firebase push-notifications.

У роботі застосовано методи об'єктно-орієнтованого програмування, побудови REST API, інтеграції з геолокаційними базами даних PostGIS, інструменти фронтенд- та бекенд-розробки, адаптивного дизайну, а також механізми WebSocket-комунікації та push-сповіщень.

КЛЮЧОВІ СЛОВА: ПЛАТФОРМА ДОПОМОГИ ТВАРИНАМ, ВЕБЗАСТОСУНОК, ГЕОЛОКАЦІЯ, WEBSOCKET, REACT, DJANGO, POSTGIS, FIREBASE, PUSH-СПОВІЩЕННЯ, ОРГАНІЗАЦІЇ.

					ІПЗ.КР.Б – 121 – 26 - ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Вециков О.М.			Платформа координації громадської допомоги тваринам	Літ.	Арк.	Аркушів
Керівник		Власенко О.В.					4	80
						Житомирська політехніка, група ІПЗ-22-2		
Рецидент		Лобанчикова Н.М.						
Зав.каф		Вакалюк Т.А.						

ABSTRACT

The bachelor's qualification thesis consists of a web application – the FurryHelp animal assistance platform – and an explanatory report. The report contains 3 chapters, 28 figures, 18 tables, a list of 30 references, and appendices.

The aim of the thesis is to develop a modern, user-friendly, and functional online service that allows users to post announcements about lost and found animals, interact with animal welfare organizations, receive geolocation-based notifications about nearby animals, and communicate in real time.

The object of research is the process of creating an online platform for assisting homeless animals with elements of social interaction and geolocation services.

The subject of the research is the methods of developing an adaptive web application implementing features such as a geolocation map, interaction with databases of announcements and organizations, user authentication, real-time notification systems via WebSocket and Firebase push notifications.

KEYWORDS: ANIMAL ASSISTANCE PLATFORM, WEB APPLICATION, GEOLOCATION, WEBSOCKET, REACT, DJANGO, POSTGIS, FIREBASE, PUSH NOTIFICATIONS, ORGANIZATIONS.

					ІІЗ.КР.Б – 121 – 26 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		4

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення

ІТ – інформаційні технології

БД – база даних

СУБД – система управління базами даних

API – Application Programming Interface

REST – Representational State Transfer

JWT – JSON Web Token

WS – WebSocket

FCM – Firebase Cloud Messaging

GIS – Geographic Information System

SPA – Single Page Application

ORM – Object-Relational Mapping

CRUD – Create, Read, Update, Delete

HTTP – HyperText Transfer Protocol

CORS – Cross-Origin Resource Sharing

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		5

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1. Аналіз використання інформаційних технологій	9
1.1. Постановка задачі	9
1.2. Аналіз аналогів програмного продукту	11
1.3. Вибір архітектури платформи	15
1.4. Обґрунтування вибору інструментальних засобів	17
Висновки до першого розділу	20
РОЗДІЛ 2. Проєктування та розробка платформи FurryHelp	20
2.1. Визначення варіантів використання та структури системи	22
2.2. Розробка бази даних	27
2.3. Проєктування та реалізація алгоритмів роботи	33
2.4. Реалізація функціоналу	38
Висновки до другого розділу	40
РОЗДІЛ 3. Інтерфейс та порядок роботи з системою	41
3.1. Порядок встановлення та налаштування системи	41
3.2. Структура інтерфейсу та порядок взаємодії з платформою	47
3.3. Тестування роботи програмного продукту	61
Висновки до третього розділу	64
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	66
ДОДАТКИ.....	69

ВСТУП

Питання захисту безпритульних тварин, а також пошуку загублених домашніх улюбленців в Україні залишається гострою соціальною проблемою, яка потребує сучасних технологічних підходів для свого вирішення. Щороку тисячі тварин опиняються на вулиці внаслідок різних факторів, проте єдиного застосунку чи навіть іншого механізму реагування на такі ситуації досі не існує. Зазвичай люди та волонтери шукають допомоги у локальних спільнотах Facebook, міських форумах чи регіональних Telegram-каналах. Однак детальний системний аналіз показує, що головна проблема таких неспеціалізованих майданчиків полягає в їхній повній технологічній непристосованості до виконання оперативних пошукових завдань та обробки просторових даних.

Через специфічні алгоритми формування стрічки новин, життєво важливі оголошення миттєво зникають під масивом іншого контенту, розважальних дописів чи реклами, стрімко втрачаючи охоплення цільової аудиторії вже в перші години після публікації. Крім того, координати місця інциденту в соціальних мережах зазвичай описуються текстом у довільній формі, що критично ускладнює орієнтацію на місцевості та унеможливлює автоматизовану обробку локацій. Найбільшим недоліком є відсутність просторового таргетування люди, які живуть у сусідньому будинку, мають фізичну можливість оперативно прийти на місце події і допомогти, часто не бачать відповідних дописів. Водночас повідомлення нецільовим чином транслюються на все місто, створюючи інформаційний шум та знижуючи загальну ефективність волонтерських ініціатив. Відсутність структурованої бази даних призводить до того, що інформація дублюється, втрачає актуальність, а координація між різними групами допомоги стає неможливою.

Актуальність даної роботи безпосередньо зумовлена об'єктивною відсутністю в Україні єдиної спеціалізованої платформи, яка б об'єднала розрізнені ініціативи та інтегрувала в собі структуровану базу оголошень, точну інтерактивну геолокацію, систему миттєвих push-сповіщень та реєстр перевірених волонтерських організацій в єдину екосистему. Існуючі західні аналоги,

					ПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		7

на кшталт Petfinder чи Lost My Doggie, організаційно та юридично орієнтовані виключно на власні ринки, використовують закриті бази даних та абсолютно не адаптовані до українських реалій, інфраструктури та актуальних потреб вітчизняного волонтерського руху.

Метою роботи є проектування та розробка кросплатформного сервісу FurryHelp (вебверсії та мобільного застосунку) для координації пошуку зниклих і знайдених тварин за допомогою геолокаційних підписок, інтерактивних карт та комунікації користувачів у реальному часі.

Об'єктом дослідження є процес створення клієнт-серверної інформаційної платформи для координації взаємодії волонтерів, громадських організацій та власників тварин із використанням геоданих та інструментів комунікації в реальному часі.

Предметом дослідження виступають архітектурні методи та сучасні технології програмної розробки: інтеграція геозапитів PostGIS, асинхронний обмін даними через протокол WebSocket та ASGI-сервер, протоколи безпечної JWT-авторизації, побудова SPA-інтерфейсу на базі бібліотеки React з інтерактивними картами та методи мобільної адаптації вебдодатків засобами фреймворку Capacitor.

Практична цінність проєкту полягає в тому, що розроблена система FurryHelp є не просто навчальним прототипом, а повноцінним, готовим до експлуатації програмним продуктом, який розгорнуто в хмарній інфраструктурі. Платформа формує єдиний цифровий простір та повністю автоматизує весь цикл допомоги: від точної фіксації інциденту на карті до миттєвої розсилки сповіщень користувачам у заданому радіусі. Інтегровані модулі верифікації фондів, регіональної модерації вмісту та автономного кешування даних дозволяють ефективно використовувати додаток як рядовим громадянам, так і професійним організаціям для порятунку та подальшої адопції тварин.

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. Аналіз використання інформаційних технологій

1.1. Постановка задачі

Процес пошуку загублених домашніх тварин або надання допомоги безпритульним тваринам у міському середовищі вимагає високої швидкості обміну інформацією. Зараз більшість громадян у разі виявлення або втрати тварини використовують неспеціалізовані майданчики, переважно тематичні групи у соціальних мережах або локальні канали у месенджерах. Користувач створює публікацію, додає фотографію, описує ситуацію та чекає на відгуки інших людей. Основний недолік такого підходу полягає в тому, що через постійне оновлення стрічки новин оголошення швидко опускається вниз і вже за кілька годин стає непомітним для більшості підписників.

Іншою суттєвою проблемою є низька точність прив'язки до місцевості. Географічні координати події зазвичай вказуються приблизно, у вигляді текстового опису орієнтирів чи назв вулиць. Через це люди, які перебувають за кілька кварталів від місця інциденту і мають реальну можливість допомогти або зафіксувати переміщення тварини, просто не отримують інформацію або отримують її із запізненням, коли тварина вже змінила своє розташування.

Ці чинники стали головною причиною для проєктування та розробки спеціалізованої платформи FurryHelp. Мета проєкту полягає не у повній заміні соціальних мереж, а у створенні ефективного інструменту, де кожне оголошення про тварину зберігається як чітко структурований запис у базі даних. Кожна публікація обов'язково містить точні географічні координати, тип тварини, її поточний статус, фотографії та дані про автора. Створення такої структури дозволяє фільтрувати інформацію за заданими параметрами, відображати актуальні мітки на інтерактивній карті міста та автоматично надсилати сповіщення тим користувачам, які оформили географічну підписку на оновлення у конкретному радіусі навколо свого будинку чи роботи.

Окрім звичайних власників тварин, інформаційна система розрахована на активну взаємодію з притулками та ветеринарними клініками. Для таких

					ІІЗ.КР.Б – 121 – 26 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		9

установ передбачено окремий тип профілю, який активується після проходження процедури верифікації адміністратором. Організації отримують можливість вести облік тварин та розширювати охоплення своїх оголошень за рахунок підвищених лімітів на публікації. Для забезпечення порядку та чистоти контенту в системі реалізовано роль регіонального модератора (менеджера). Його завдання полягає в обробці скарг від користувачів та модерації оголошень виключно в межах свого міста чи області, що дозволяє децентралізувати управління платформою.

Для наочного представлення функціональних вимог до системи, основні інженерні задачі платформи FurryHelp, їхній зміст та конкретні програмні інструменти реалізації зведено у таблицю 1.1.

Таблиця 1.1

Основні задачі системи та їх реалізація

Задача	Зміст	Реалізація
Структуроване оголошення	Стандартизовані поля замість довільного тексту	Модель Report, ReportSerializer, форма CreateReportPage
Геолокація	Координати події для карти і сповіщень	PointField, Leaflet, bbox-фільтрація
Real-time оновлення	Нові оголошення без перезавантаження сторінки	Django Channels, CityConsumer
Верифікація організацій	Контроль збору коштів і статусу	OrganizationRequest, сигнал approve_organization
Модерація	Скарги й обмеження прав за регіоном	Complaint, manager_region, Django admin
Життєвий цикл	Автоматичне архівування неактивних оголошень	archive_stale_reports, bump_used, expiry_warned

1.2. Аналіз аналогів програмного продукту

Перед розробкою були розглянуті три майданчики, якими реально користуються люди в подібних ситуаціях: Lost My Doggie, Petfinder і Facebook-групи [31].

Lost My Doggie це спеціалізований сервіс для пошуку загублених тварин. Головна ідея швидко розіслати повідомлення людям поруч. Це близько до того, що робить FurryHelp, але сервіс не адаптований під українські міста, не має регіональної модерації й не дає карту оголошень у реальному часі [28].

Petfinder це великий каталог тварин, переважно для адопції. Добре пропрацьовані профілі організацій і фільтри пошуку. Але оперативний сценарій "щойно знайшов собаку, хочу повідомити сусідів" там не є основним [27].

Facebook-групи найпоширеніший варіант, бо поріг входу нульовий. Але саме тому там хаос: немає єдиного формату, дописи дублюються, геолокація вказана словами, а автоматичних сповіщень за радіусом не існує.

У рамках дослідження було проведено аналіз існуючих аналогів та платформ-прототипів, що функціонують на ринку. Це дозволило визначити їхні ключові переваги та функціональні недоліки для подальшого проектування власної системи.

The image shows a screenshot of the 'Lost My Doggie' website. At the top is an orange header with the logo and tagline 'Find lost and stray dogs quickly'. Below the header is a white form with the heading 'ENTER THE PET'S INFO TO IMMEDIATELY START NOTIFYING THE LOCAL AREA.' The form contains a 'PET STATUS' section with radio buttons for 'Lost' (selected) and 'Found/Stray'. It then has input fields for 'Pet Name*' (with a note 'if unsure, put unknown'), 'Kind of Pet*' (a dropdown menu showing 'Dog'), 'Last Seen Address*' (with a placeholder 'Введіть місцезнаходження'), 'Email Address*', and 'Phone*'. At the bottom of the form is a blue button with a white dog icon and the text 'FIND MY PET'.

Рис 1.1 – Платформа Lost My Doggie

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

Lost My Doggie це спеціалізований закордонний сервіс, орієнтований виключно на розшук зниклих домашніх тварин.

Сильні сторони це вузька спеціалізація саме на загублених тваринах, ефективна ідея швидких локальних сповіщень для жителів району, а також максимально зрозумілий та покроковий сценарій подачі повідомлення про зникнення улюбленця.

Проблема платформи вона орієнтована на іноземний ринок і не є універсальним українським рішенням. У ній відсутні інструменти роботи з регіональними менеджерами, немає інтеграції з PostGIS-картами для точного геопросторового аналізу, відсутній загальний каталог організацій (притулків, волонтерських центрів) та гнучке розмежування ролей для них.



Рис 1.2 – Платформа Petfinder

Petfinder одна з найбільших у світі баз даних для пошуку та влаштування безпритульних тварин у сім'ї.

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

Сильні сторони це надзвичайно розвинений та структурований каталог тварин, які готові до адопції. Тісна та автоматизована взаємодія із професійними притулками та rescue-організаціями. Наявність детальної фільтрації за багатьма параметрами тварини наприклад вік, розмір, порода, сумісність.

Проблема це головний вектор системи спрямований саме на довготривалу адопцію (пошук нової домівки), а не на оперативний розшук критично загублених або щойно знайдених тварин у конкретному радіусі підписки користувача.



Пошук загублених тварин Україна /Поиск потерянных животных Украина >

🔓 Загальнодоступна група · 19 290 учасників

Приєднатися до групи

Вибране

Відео Reels

Світлини

Події



Напишіть щось...



Найактуальніші



Наталі Бохонок

1 год · 🌐



Потеряшки Одесса · [Приєднатися](#)

Вероника Коваленко · 2 год · 🌐

Добрый день! В районе Ярморочной
найдена собака породы Малинуа, ... більше

Рис 1.3 – Локальні Facebook-групи

Facebook-групи на сьогодні залишаються основним стихійним інструментом для пошуку тварин в Україні.

Сильні сторони це мінімальний поріг входу для будь-якого користувача (соцмережа вже встановлена у більшості), величезна кількість живих локальних спільнот (наприклад, за районами чи містами), а також можливість миттєво отримувати коментарі та підказки від людей, які знаходяться поруч.

Проблеми тут це повна відсутність єдиного формату даних (кожен пише оголошення як хоче, часто забуваючи вказати місто чи контакти). Стрічка новин швидко оновлюється, через що оголошення «губляться». Відсутні геозапити (пошук на карті), автоматичні підписки на радіус зникнення, централізована верифікація волонтерських організацій та контроль життєвого циклу оголошення (невідомо, чи тварину вже знайшли, чи оголошення досі актуальне).

Таблиця 1.2

Аналіз конкурентних рішень у сфері пошуку тварин

Критерій	Lost My Doggie	Petfinder	Facebook-групи	FurryHelp
Структуровані оголошення	Так	Так	Частково	Так
Інтерактивна карта	Частково	Частково	Ні або вручну	Так
Геопідписка за радіусом	Частково	Ні	Ні	Так
WebSocket-оновлення	Не основний фокус	Не основний фокус	Ні	Так
Верифікація організацій	Обмежено	Так	Залежить від групи	Так
Регіональна модерація	Не основний фокус	Не основний фокус	Залежить від адмінів	Так
Мобільна оболонка	Так/веб	Так	Так	Так
Життєвий цикл оголошення	Частково	Частково	Ні	Так

У результаті аналізу від спеціалізованих сервісів взято ідею оперативних сповіщень, від каталогів підхід до профілів організацій, а слабкі сторони соціальних груп закриті структурованою базою даних, фільтрами й автоматичною модерацією.

1.3. Вибір архітектури платформи

Програмна платформа FurryHelp побудована за сучасною моделлю розділеної клієнт-серверної архітектури, що забезпечує чітке розмежування зон відповідальності між рівнем відображення даних та рівнем бізнес-логіки. Клієнтська частина реалізована у вигляді односторінкового вебдодатка (Single Page Application, SPA), який відповідає за рендеринг користувацького інтерфейсу, клієнтську маршрутизацію, управління станами, інтерактивну роботу з картами та нативну мобільну оболонку. Своєю чергою, серверна частина виступає ізольованим API-шлюзом, який інкапсулює в собі всю складну бізнес-логіку, включаючи процеси автентифікації, перевірку прав доступу, обробку геопросторових запитів, генерацію сповіщень та автоматичне архівування неактуальних даних [7].

В якості фундаментального фреймворку для розробки серверної частини було обрано Python-фреймворк Django у зв'язці з бібліотекою Django REST Framework (DRF). Головним аргументом на користь цього вибору стала висока зрілість екосистеми та наявність потужного об'єктно-реляційного відображення (ORM) із вбудованою системою міграцій. DRF надає готові інструменти для серіалізації даних, налаштування політик доступу (permissions), обмеження частоти запитів (throttling) та маршрутизації через ViewSet, що дозволяє уникнути написання шаблонного коду з нуля. Хоча альтернативні сучасні рішення, такі як FastAPI, потенційно забезпечують вищу пропускну здатність (throughput) при обробці запитів, для проєкту FurryHelp критично важливою була швидкість розробки надійної бізнес-логіки та безшовна інтеграція з модулем GeoDjango [4], який є індустріальним стандартом для роботи з геопросторовими базами даних.

Окрему увагу під час проєктування архітектури було приділено протоколам передачі даних. Використання традиційного синхронного стандарту WSGI виявилось концептуально непридатним для цього проєкту, оскільки технологія WebSocket тут виступає не як додаткова функція, а як ядро системи реального часу. Стрічка активних подій міста, система коментарів та механізм

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		15

миттєвих геосповіщень вимагають постійного двостороннього обміну даними. Для забезпечення підтримки довготривалих з'єднань архітектуру було переведено на стандарт асинхронного серверного шлюзу (ASGI). Реалізація цього підходу здійснюється за допомогою вебсервера Daphne та бібліотеки Django Channels, де в ролі транспортного шару (channel layer) для обміну подіями між серверними процесами виступає високошвидкісне in-memory сховище Redis.

Для розробки клієнтської частини перевагу було надано бібліотеці React, яка перевершила альтернативні фреймворки Vue та Angular завдяки гнучкому компонентному підходу, широкій екосистемі та концепції віртуального DOM. Фреймворк Angular був відхилений через надмірну архітектурну складність для проєкту даного масштабу. Додатковою перевагою React стала його природна сумісність з інструментом збірки Vite, який забезпечує миттєве гаряче перезавантаження модулів під час розробки, а також безшовна інтеграція з бібліотекою Leaflet для роботи з картами та технологією Capacitor для фінальної компіляції вебдодатка у нативний мобільний застосунок [7]. Загальна схема взаємодії описаних компонентів представлена на рисунку 1.4.

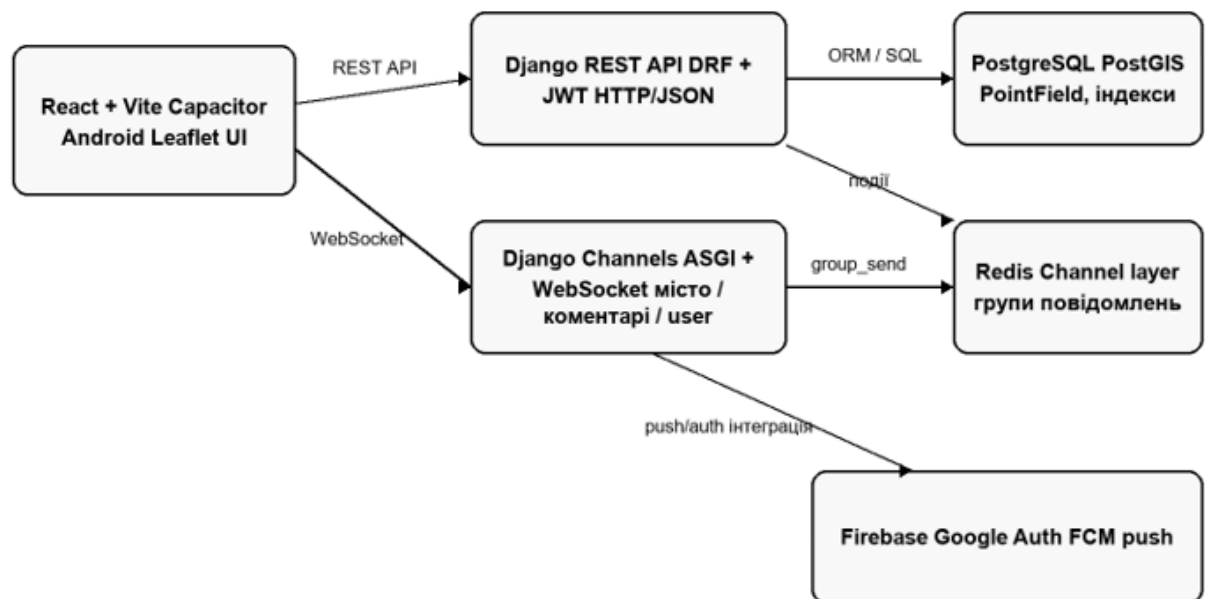


Рис. 1.4 – Загальна схема взаємодії компонентів FurryHelp

Архітектурні компоненти та їх обґрунтування

Компонент	Роль у системі	Причина вибору
React + Vite	Клієнтський інтерфейс, маршрути, форми, карта, мобільна оболонка.	Швидка збірка, компонентність, підтримка сучасного frontend-стеку.
Django REST Framework	REST API, serializers, permissions, throttling, ViewSet.	Зменшує кількість ручного API-коду, добре поєднується з ORM.
Django Channels	WebSocket-канали для карти, коментарів і повідомлень.	Працює у Django-екосистемі та підтримує ASGI.
Redis	Channel layer для обміну подіями між процесами.	Простий і швидкий брокер повідомлень для Channels.
PostgreSQL + PostGIS	Реляційне зберігання даних і геопросторові запити.	Підтримує PointField, Distance, індекси і надійні транзакції.
Firebase	Google Auth і Firebase Cloud Messaging.	Дозволяє інтегрувати Google-вхід і push-повідомлення.
Capacitor	Android-оболонка для React-застосунку.	Дає доступ до native API без переписування клієнта.

1.4. Обґрунтування вибору інструментальних засобів

Головний критерій при виборі кожного інструменту чи він закриває конкретну потребу, яку не закрити простіше.

PostgreSQL з PostGIS потрібен для підписок на геозони. Відстань між точкою підписки і точкою оголошення рахується прямо в SQL через Distance() база повертає лише релевантні записи, без перебору всіх підписок у Python. Без PostGIS цей запит або неможливий, або катастрофічно повільний при зростанні даних [5].

Pillow використовується для обробки фотографій: при завантаженні кожне зображення конвертується у WebP і стискається до максимум 1024×1024 пікселів із якістю 80%. Це зменшує розмір файлів у 2–3 рази і попутно sanitize-ить завантажені файли конвертація через PIL ефективно прибирає потенційно шкідливий вміст [22].

IndexedDB на клієнті дозволяє зберігати офлайн-чергу оголошень разом з фотографіями. localStorage для цього не підходить він не тримає бінарні дані і має ліміт 5 МБ. Коли з'єднання відновлюється, offlineQueue.js формує FormData і відправляє збережені записи на сервер [23].

Simple JWT обраний тому, що SPA і Capacitor не використовують серверні сесії. Access-токен живе 60 хвилин, refresh добу. Axios interceptor на клієнті автоматично оновлює токен, не перериваючи запит [18].

Таблиця 1.4

Порівняння СУБД для геопросторових та реляційних задач

Критерій	PostgreSQL + PostGIS	MySQL	MongoDB
Геопросторові типи	PointField, geography, Distance, ST-функції PostGIS.	Є spatial-можливості, але екосистема Django GIS слабша для цієї задачі.	Є geospatial indexes, але модель даних не така зручна для реляційних зв'язків FurryHelp.
Реляційні зв'язки	Сильна сторона: FK, constraints, indexes, транзакції.	Також сильна сторона.	Потрібне ручне моделювання зв'язків або дублювання даних.
Django/GeoDjango	Нативна підтримка через django.contrib.gis.db.backends.postgis.	Можливо, але з меншим набором GIS-практик.	Не є природним вибором для Django ORM.

продовж. табл. 1.4

Сценарій підписок за радіусом	Оптимальний: обчислення відстані у базі.	Можливий, але менш типовий у цьому стеку.	Можливий, але складніше поєднати з усіма реляційними ролями.
Підсумок	Обрано.	Не обрано через гіршу відповідність GIS-стеку.	Не обрано через складніші зв'язки та модераційні сценарії.

Таблиця 1.5

Використані інструменти та їх практичне обґрунтування

Інструмент	Де використано	Практичне обґрунтування
Simple JWT	CustomLoginView, refresh, blacklist.	Підходить для SPA і мобільної оболонки, не вимагає серверної сесії для кожного API-запиту.
django-filter	ReportViewSet.	Дає фільтрацію за типом тварини, статтю, типом оголошення і містом.
Pillow	compress_image, thumbnail для карти.	Перевіряє та стискає зображення, зменшує ризик великих файлів.
WhiteNoise	Static files у production.	Спрощує обслуговування статичних файлів у контейнері.
Firebase Admin SDK	Перевірка Google ID token і FCM.	Пов'язує frontend Google Auth із backend JWT та push-повідомленнями.
IndexedDB	offlineQueue.js.	Зберігає оголошення з фото офлайн, що неможливо зробити надійно через localStorage.

Аналітична частина проєкту базується на UML-нотаціях та діаграмах активності [1; 2]. Це дозволило чітко окреслити межі системи, ролі користувачів та логіку процесів, уникнувши перевантаження текстом.

Вибір бекграунд-стеку (Django, DRF, PostgreSQL/PostGIS, Redis, Daphne) обґрунтовано архітектурною документацією [3; 4; 5; 6; 9]. Комплекс цих технологій забезпечує не лише швидку розробку, а й готову підтримку геопросторових запитів, асинхронного зв'язку та безпечної JWT-авторизації.

Клієнтська частина проєктувалася з акцентом на швидкість, адаптивність та роботу в офлайн. Для реалізації карт, маршрутизації, локального збереження даних та push-сповіщень обрано стек на базі React, Leaflet, Capacitor та Firebase [7; 8; 13; 14; 15; 16].

Вимоги до якості формувалися з урахуванням принципів чистої архітектури, безпеки та контейнеризації [10; 11]. Це гарантує, що система буде не просто функціональною, а стійкою до навантажень та зручною для подальшого супроводу.

Висновки до першого розділу

У першому розділі роботи було детально проаналізовано предметну область задачі, визначено основні проблеми координації громадської допомоги тваринам та обґрунтовано комплекс технологічних рішень, які покладені в основу розробки хмарної платформи.

Встановлено, що існуючі на ринку аналогічні програмні рішення, такі як сервіси Lost My Doggie, Petfinder або тематичні Facebook-групи, вирішують задачу пошуку зниклих та прилаштування безпритульних тварин лише частково. Головним їхнім недоліком є відсутність комплексності: жоден із сервісів не поєднує в собі структуровані оголошення з автоматичною геопідпискою за радіусом, системою верифікації громадських організацій, локальною модерацією та автоматичним контролем життєвого циклу записів. Саме ця

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

комбінація функцій є ключовою відмінністю платформи FurryHelp, яка дозволяє закрити всі потреби волонтерів та звичайних громадян в межах єдиного продукту.

На основі проведеного аналізу вимог до системи було сформовано та обґрунтовано оптимальний архітектурний стек розробки. Використання інструменту Django REST Framework забезпечує швидке створення стабільного REST API та надійну реалізацію бізнес-логіки. Інтеграція фреймворку Django Channels разом із сховищем Redis вирішує задачу підтримки асинхронних WebSocket-з'єднань для миттєвого оновлення стрічки подій міста та коментарів у реальному часі.

Для роботи з геоданими обрано зв'язку PostgreSQL та розширення PostGIS, що дозволяє перенести складні просторові запити та розрахунки радіусів безпосередньо на рівень системи управління базами даних. Клієнтська частина платформи для браузерів та операційної системи Android базується на бібліотеці React зі збірщиком Vite та кросплатформеною оболонкою Capacitor, що спрощує підтримку та розробку інтерфейсу.

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		21

РОЗДІЛ 2. Проектування та розробка платформи FurryHelp

2.1. Визначення варіантів використання та структури системи

Перш ніж проектувати базу даних і API, необхідно чітко визначити, хто користується системою і що саме кожен з них може робити. У FurryHelp передбачено чотири ролі: звичайний користувач, організація, регіональний менеджер і адміністратор. Кожна роль має свій набір дозволів, і ці межі закладені не лише на рівні інтерфейсу, а безпосередньо в логіці backend.

Звичайний користувач основна аудиторія платформи. Він реєструється, підтверджує email, заповнює профіль і починає публікувати оголошення. Йому доступна карта з усіма активними записами, фільтри за типом тварини і містом, коментарі під оголошеннями, підписки на геозони і можливість подати скаргу на підозрілий або повторний допис. Якщо контакти іншого користувача приховані, він може надіслати ContactRequest запит на відкриття контактних даних.

Організація це верифікований притулок, ветклініка або волонтерський центр. Статус організації не видається автоматично: користувач подає заявку OrganizationRequest із назвою, ЄДРПОУ, адресою, описом і координатами. Менеджер або адміністратор розглядає заявку і у разі схвалення система автоматично створює запис Organization і встановлює прапор `user.is_organization = True`. Після цього організація отримує профіль у каталозі, власні публікації, відгуки від користувачів і підвищений ліміт на кількість оголошень 20 за 15 хвилин замість 3 для звичайного акаунту. Крім того, лише організація може додавати посилання на збір коштів, і воно проходить перевірку домену в сервізаторі.

Менеджер є проміжною адміністративною роллю, яка виникла з практичної потреби: платформа орієнтована на кілька міст, і одна людина-адміністратор не може оперативно обробляти скарги й заявки по всій країні. Водночас давати кожному модератору повний доступ до всіх даних небезпечно. Рішення полягає у прив'язці до регіону через поле `manager_region`. Технічно менеджер не має окремої таблиці в базі даних це проху-модель Manager над моделлю User

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

і Django-група Managers. Методи `can_manage_city` і `manager_city_q` автоматично обмежують `queryset`: менеджер бачить і може змінювати лише ті записи, місто яких збігається з його регіоном. Це забезпечує ізоляцію даних без дублювання моделі користувача.

Адміністратор має повний доступ до Django admin без будь-яких регіональних обмежень. Він керує всіма користувачами, організаціями, заявками, оголошеннями і скаргами. Роль адміністратора в системі загальна підтримка й вирішення ситуацій, які виходять за межі повноважень менеджера.

Таблиця 2.1

Ролі та їхні можливості в системі

Актор	Права та можливості	Технічна реалізація
Користувач	Реєстрація, вхід, редагування профілю, створення оголошень, коментарі, скарги, підписки на геозони, контактні запити	User, JWT, ReportViewSet, CommentViewSet, SubscriptionViewSet, ContactRequestViewSet
Організація	Профіль у каталозі, пости, збір коштів через дозволені домени, підвищений ліміт оголошень	User.is_organization, Organization, OrganizationPost, OrganizationRequest
Менеджер	Модерація скарг, заявок і користувачів у своєму регіоні	Група Managers, proxy-модель Manager, поле <code>manager_region</code> , методи <code>can_manage_city</code>
Адміністратор	Повний доступ до всіх записів і налаштувань	<code>is_superuser</code> , Django admin

Після визначення ролей треба описати конкретні сценарії взаємодії. Кожен такий сценарій складається з трьох речей: користувач, який робить дію, обов'язкова умова для початку і результат, який система зберігає або повертає.

Реєстрація це старт для кожного нового користувача. Коли людина заповнює форму, система шле лист із посиланням для підтвердження пошти. Увійти на сайт можна тільки після того, як поле `email_verified` стане `True`. Це захищає систему від реєстрацій на фейкові адреси і дозволяє нормально надсилати сповіщення на емейл.

Увійти в акаунт можна двома способами. Перший спосіб є стандартним це користувач вводить логін і пароль через `CustomLoginView`, а система повертає йому `access` і `refresh` JWT-токени. Другий спосіб працює через Google клієнт отримує `Firebase ID token`, відправляє його на бекенд, де функція `verify_id_token` перевіряє підпис Google, і після цього видається токен сайту. Обидва шляхи дають однаковий результат — авторизовану сесію.

Створення оголошення зроблено у вигляді однієї форми з логічними блоками. Там є тип і стать тварини, опис, місто, фотографії та точка на карті, якщо вона потрібна. Перед збереженням система перевіряє ліміти на спам. Якщо користувач додав фото, бібліотека `Pillow` автоматично стискає їх у формат `WebP`. Якщо в оголошенні є посилання на збір грошей, серіалізатор перевіряє домен, бо додавати такі посилання можуть тільки організації [22].

Підписка на геозону дозволяє користувачу вибрати точку на карті та радіус у кілометрах. Також можна налаштувати фільтр за типом тварини чи оголошення. Коли хтось публікує новий запис у цій зоні, `Django Signal` автоматично створює об'єкт `Notification` і відправляє `WebSocket`-подію прямо на пристрій підписника [24].

Функція `Bump` це можливість один раз підняти оголошення в топ і продовжити термін його дії. Вона доступна тільки автору і лише тоді, коли прапорець `bump_used` у цьому циклі ще не активований. Після натискання кнопки оновлюється поле `last_activity_at`, а прапорці `expiry_warned` і `bump_used` скидаються назад.

Скаргу на чуже активне оголошення може подати будь-який авторизований користувач. Один юзер може поскаржитися на конкретний пост лише

					ПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		24

один раз, що контролюється обмеженням на рівні бази даних. Коли оголошення набирає 10 скарг, система сама переводить його в архів, а автор отримує сповіщення.

Заявку на реєстрацію організації можна відправити тільки один раз, за це відповідає зв'язок OneToOneField у базі. Коли менеджер або адмін схвалює цю заявку, Django Signal створює або оновлює профіль організації та виставляє прапорець `is_organization = True`. Одночасно з цим користувачу йде сповіщення про результат перевірки.

Відновлення пароля працює через емейл. Користувач пише свою адресу, система створює токен і надсилає його в посиланні. Коли людина переходить за посиланням і вводить новий пароль, цей токен одразу анулюється.

Модерація регіонів налаштована в адмінці Django. Локальний менеджер бачить скарги, заявки та користувачів тільки зі свого міста чи області. Будь-яка спроба подивитися дані за межами його регіону повертає порожній результат.

Таблиця 2.2

Варіанти використання системи

Варіант використання	Актор	Передумова	Результат
Реєстрація	Користувач	Користувач не автентифікований у системі	Створено обліковий запис користувача, надіслано email із посиланням для підтвердження
Вхід через логін/пароль	Користувач	Обліковий запис активований, email підтверджено	Користувачу видано access та refresh JWT-токени
Вхід через Google	Користувач	Наявність чинного Firebase ID token від сервісу Google	Сервер валідує token та видає відповідні JWT-токени
Створення оголошення	Користувач або організація	Користувач автентифікований, введені дані є валідними	Створено об'єкт моделі Report та, за наявності зображень, об'єкти ReportImage
Підписка на геозону	Користувач	Користувач автентифікований, обрано географічну точку та радіус	У базі даних створено запис Subscription із просторовим полем PointField
Отримання сповіщення	Користувач	Створення нового оголошення у радіусі оформленої підписки	Створено об'єкт Notification, подію доставлено через асинхронний WebSocket або email

Випр оголошення	Автор	Оголошення має статус активного, прапорець bump_used = False	Оновлено поле last_activity_at, термін дії оголошення автоматично продовжено
Скарга	Користувач	Наявність активного оголошення, автором якого є інший користувач	Створено об'єкт Complaint; у разі досягнення 10 скарг — автоматична архівація поста
Заявка організації	Користувач	Введені реєстраційні дані та документи організації є валідними	Створено об'єкт OrganizationRequest із початковим статусом pending
Схвалення організації	Менеджер або адміністратор	Наявність активної заявки від організації у дозволеному регіоні	Створено об'єкт моделі Organization, прапорець користувача user.is_organization змінено на True
Відновлення пароля	Користувач	Наявність доступу до електронної пошти, вказаної при реєстрації	Надіслано reset link, пароль успішно змінено після перевірки згенерованого токена
Модерація регіону	Менеджер	Користувач успішно авторизований та доданий до системної групи Managers	Доступ модератора обмежено об'єктами, що містяться у межах manager_region



Рис. 2.1 – Діаграма варіантів використання FurryHelp

Окремо під час проєктування були визначені нефункціональні вимоги до якості платформи. Для FurryHelp вони є важливими, оскільки система працює з терміновими оголошеннями, геолокаційними даними та сповіщеннями, які мають надходити користувачам без суттєвої затримки.

Таблиця 2.3

Нефункціональні вимоги до платформи FurryHelp

Крите- рій	Вимога	Очікуваний результат
Сприй- няття	Інтерфейс зрозумілий для всіх категорій користувачів без додаткового навчання	Освоєння до 15 хв для звичайного ко- ристувача, до 30 хв для організації або менеджера
Надій- ність	Стабільна обробка оголо- шень, скарг і сповіщень протягом тривалого часу	Безвідмовна робота 20 днів; не більше 1 помилки на 1000 запитів
Продук- тивність	Пошук на карті, фільтрація і доставка сповіщень без по- мітного сповільнення	Стабільна робота при 100+ одночасних користувачах
Експлуа- тація	Підтримка розширення: мі- ста, користувачі, організації, геозони	Масштабування через PostgreSQL/PostGIS, Redis, ASGI і Docker

Таке подання вимог дозволяє пов'язати технічні рішення з практичними умовами використання: користувач повинен швидко створити або знайти оголошення, а організація чи менеджер має отримати інструменти для перевірки інформації та реагування на звернення.

2.2. Розробка бази даних

База даних FurryHelp побудована на PostgreSQL з розширенням PostGIS. Вибір саме цієї СУБД продиктований не лише потребою у надійному реляційному сховищі, а й тим, що частина функціональності платформи підписки на геозони і відображення оголошень на карті принципово вимагає роботи з координатами на рівні бази даних. Без PostGIS перевірку того, чи потрапляє нове оголошення у радіус підписки, довелося б виконувати у Python, перебираючи всі записи вручну. З PostGIS цей запит виконується однією SQL-операцією через Distance() і повертає лише релевантні рядки [5].

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

Усі моделі системи розподілені між п'ятьма Django-додатками: users, reports, comments, organizations і notifications. Такий поділ не є формальним кожен додаток відповідає за власну область і не знає деталей реалізації сусіднього. Наприклад, логіка архівування повністю зосереджена в reports, тоді як механізм підписок і сповіщень у notifications. Це спрощує підтримку і дозволяє змінювати поведінку однієї частини системи, не зачіпаючи інші.

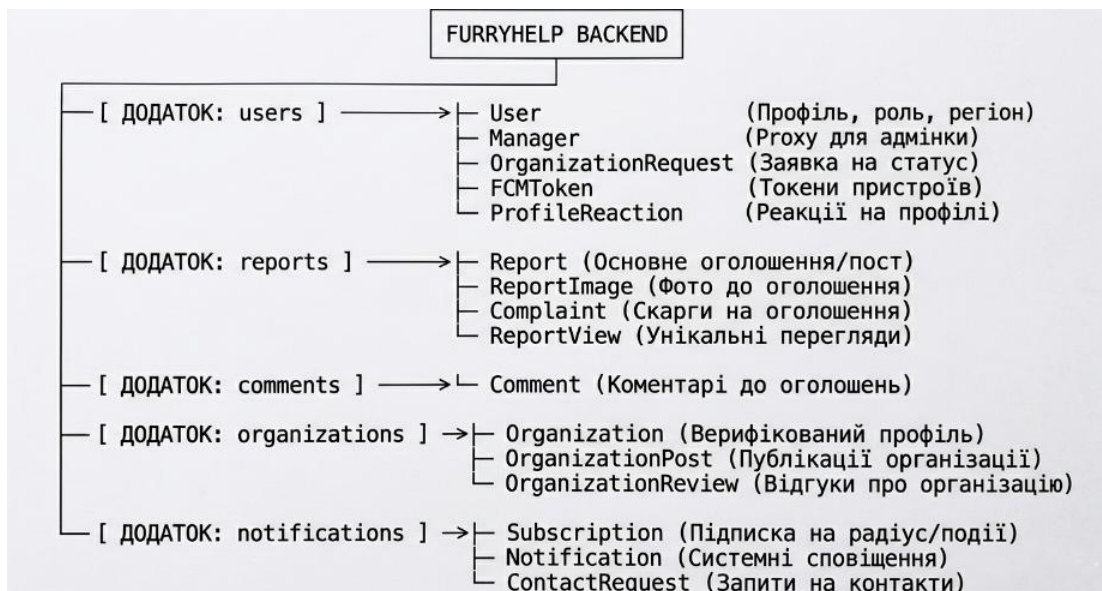


Рис. 2.2 – Структура бекенду

Зв'язки між моделями побудовані через ForeignKey і OneToOneField. Відносини "один до одного" використані там, де дублювання принципово неприпустиме: Organization і OrganizationRequest мають OneToOneField до User, що унеможлиблює ситуацію, коли один акаунт подає кілька заявок або має кілька профілів організації. Для захисту від дублювання на рівні бази даних також застосовані UniqueConstraint: один користувач не може подати більше однієї скарги на те саме оголошення, залишити більше одного відгуку про організацію або зафіксувати більше одного перегляду конкретного запису. На рисунку 2.3 наведено ER-діаграму основних таблиць і зв'язків між ними.

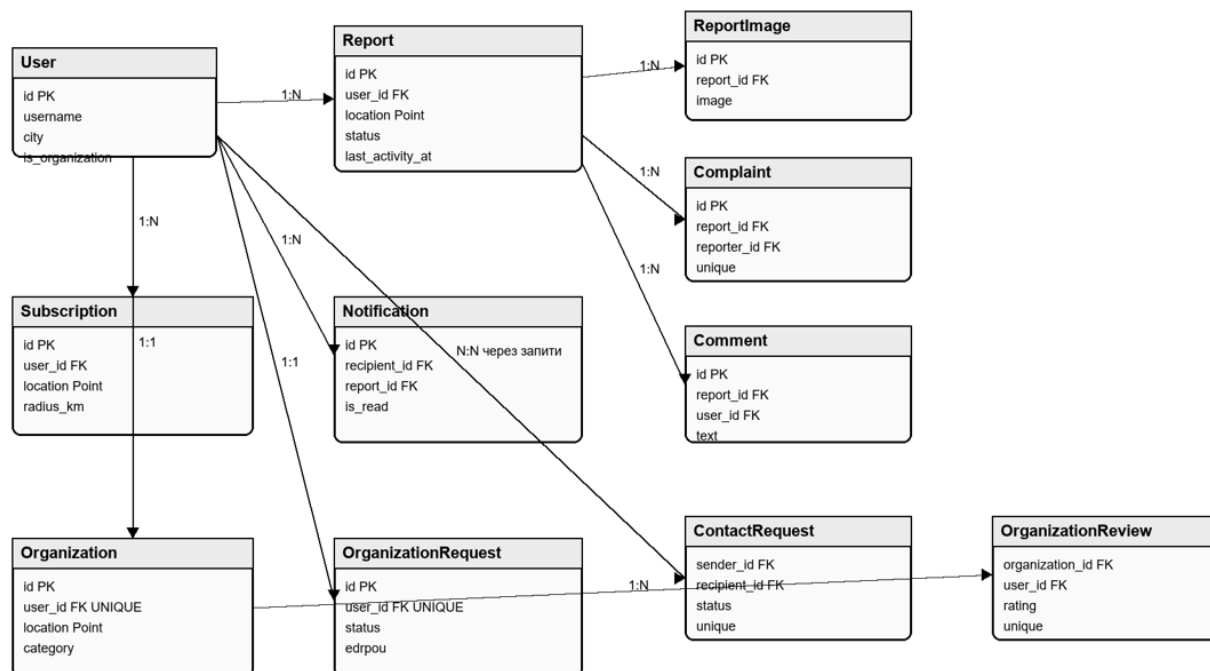


Рис. 2.3 – ER-діаграма основних таблиць і зв'язків

Модель користувача є головною в системі, тому більшість інших моделей пов'язані з нею через ForeignKey або OneToOneField. Вона створена на основі стандартного AbstractUser з Django, але має додаткові поля для потреб платформи.

Поле city потрібне для двох різних завдань. Спочатку воно працює як звичайний фільтр у стрічці оголошень, а потім використовується для розсилки push-сповіщень через Firebase. Коли хтось додає нове оголошення, система надсилає push-повідомлення всім користувачам із цього міста.

Поле manager_region потрібне тільки для менеджерів, воно визначає межі їхніх прав в адмінці Django.

Поле is_organization це звичайний прапорець, який система ставить сама, коли адміністратор схвалює заявку. Цей прапорець відкриває доступ до функцій організації, тому міняти сам тип акаунту в системі не доводиться.

Таблиця 2.4

Структура моделі User

Поле	Тип	Призначення
id	BigAutoField (PK)	Ідентифікатор користувача
username	CharField (unique)	Відображуване ім'я та логін

email	EmailField (unique)	Адреса для входу, підтвердження і відновлення пароля
is_organization	BooleanField	Прапор верифікованого представника організації
phone	CharField	Телефон після нормалізації формату
bio	TextField	Короткий опис профілю
city	CharField	Місто для фільтрів і push-сповіщень
manager_region	CharField	Регіон модерації для менеджерів
show_contact_info	BooleanField	Чи показувати контакти публічно
custom_contact	CharField	Додатковий спосіб зв'язку
resolved_reports_count	PositiveIntegerField	Лічильник вирішених оголошень
email_verified	BooleanField	Стан підтвердження email
email_notifications	BooleanField	Дозвіл на email-сповіщення

Модель оголошення є найбільш функціональною та складною в усій системі. Крім базових даних про саму тварину, вона має кілька спеціальних полів для керування життєвим циклом запису.

Поле `last_activity_at` оновлюється не тільки тоді коли автор щось змінює вручну, воно автоматично оновлюється з кожним новим коментарем під постом. Завдяки цьому популярні оголошення, де йде активне обговорення, довше залишаються видимими в системі. Поле `expiry_warned` потрібне для того, щоб система не надсилала автору повторні попередження про швидку архівацію. А прапорець `bump_used` обмежує використання функції підняття оголошення, дозволяючи зробити це лише один раз за поточний цикл.

Поле `location` з типом `PointField` є необов'язковим, адже далеко не кожен користувач може вказати точне місце на карті. Проте, якщо координати є, система використовує їх одразу у двох сценаріях: для виведення маркера на інтерактивній карті та для автоматичної перевірки геопідписок через PostGIS відразу після збереження оголошення [5].

Структура моделі Report

Поле	Тип	Призначення
id	BigAutoField (PK)	Ідентифікатор оголошення
user	ForeignKey(User)	Автор оголошення
title	CharField(100)	Заголовок
description	TextField(1000)	Опис ситуації та прикмет тварини
city	CharField(100, index)	Місто оголошення
views_count	PositiveIntegerField	Лічильник переглядів
animal_type	CharField	dog, cat або other
animal_gender	CharField	male, female або unknown
report_type	CharField (index)	lost, found, adoption або info
status	CharField (index)	active, resolved або archived
donation_link	URLField	Посилання на збір коштів, доступне лише організаціям
image	ImageField	Головне фото оголошення
location	PointField (GIS)	Координати події, необов'язково
created_at	DateTimeField (index)	Дата створення
last_activity_at	DateTimeField (index)	Остання активність для розрахунку архівації
expiry_warned	BooleanField	Чи надіслано попередження про наближення терміну

Модель підписки зберігає геозону, яку налаштував конкретний користувач. Центром цієї зони є поле PointField із чітко вказаним координаційним стандартом srid=4326, а сам радіус користувач задає в кілометрах.

Поля animal_type та report_type потрібні для того, щоб звузити фільтрацію. Наприклад, людина може підписатися на сповіщення тільки про загублених котів, ігноруючи при цьому повідомлення про знайдених собак чи оголошення про пошук нового дому. Якщо в обох цих полях стоїть значення any це

значить, що користувач хоче отримувати абсолютно всі сповіщення у своєму радіусі без всіляких додаткових фільтрів.

Таблиця 2.6

Структура моделі Subscription

Поле	Тип	Призначення
id	BigAutoField (PK)	Ідентифікатор підписки
user	ForeignKey(User)	Власник підписки
location	PointField(srid=4326)	Центр геозони
address_label	CharField	Довільна назва зони, наприклад "Мій дім"
radius_km	FloatField	Радіус у кілометрах
animal_type	CharField (index)	Фільтр за типом тварини або any
report_type	CharField (index)	Фільтр за типом оголошення або any
created_at	DateTimeField (index)	Дата створення підписки

Модель сповіщень ділить усі повідомлення на два типи. Перший тип `animal`, воно створюється автоматично, коли нове оголошення з'являється всередині геозони підписки. Другий тип `system`, його надсилає сама платформа під час архівації оголошень, після перевірки заявок або у разі блокування акаунту.

Поле `report` є необов'язковим, оскільки системні сповіщення можуть і не мати прив'язки до якогось конкретного оголошення. Поле `is_read` потрібне для роботи лічильника непрочитаних повідомлень в інтерфейсі застосунку, а його статус оновлюється через окремий `endpoint`.

Таблиця 2.7

Структура моделі Notification

Поле	Тип	Призначення
id	BigAutoField (PK)	Ідентифікатор сповіщення
recipient	ForeignKey(User)	Отримувач
report	ForeignKey(Report), nullable	Пов'язане оголошення, якщо є

notification_type	CharField (index)	animal або system
title	CharField (index)	Заголовок сповіщення
message	TextField	Текст сповіщення
is_read	BooleanField (index)	Стан прочитання
created_at	DateTimeField (index)	Дата створення

2.3. Проєктування та реалізація алгоритмів роботи

Логіка роботи FurryHelp не зібрана в одному великому модулі, а рознесена по тих місцях, де вона має бути за архітектурою. Життєвий цикл оголошень захищений у самі моделі та консольні команди, геосповіщення працюють через Django Signals, захист сокетів винесений у consumers, а антиспам налаштований прямо в методах ViewSet. Такий підхід дозволяє спокійно міняти код в одній частині системи і не боятися, що зламається щось інше. Крім того, це спрощує написання окремих тестів для кожного інструменту.

Головна проблема будь-якої дошки оголошень це застарілі записи, які висять місяцями без жодних оновлень. Для сервісу пошуку тварин це критично, адже пусті оголошення річної давнини просто дезінформують людей. Щоб вирішити цю проблему, в систему вбудували автоматичне керування статусами на основі трьох полів моделі Report. Це поля last_activity_at, expiry_warned та bump_used.

Час дії оголошення залежить від його категорії. Для статусів lost, found та info він становить 30 днів, а для adoption, тобто пошуку нового дому, цей термін дорівнює 60 дням, оскільки прилаштування тварин зазвичай триває довше. Таймер рахується не від дня створення поста, а від мітки last_activity_at. Ця дата автоматично оновлюється з кожним новим коментарем або після підняття оголошення в топ. Тому, якщо під постом триває активне обговорення, він не зникне сам по собі.

За два дні до закінчення терміну система надсилає власену попередження та виставляє прапорець `expiry_warned = True`, щоб не спамити повідомленнями знову. У цей момент автор може використати `bump`, тобто можливість один раз продовжити дію оголошення. Тоді мітка `last_activity_at` змінюється на поточний час, прапорці анулюються, і весь відлік починається заново.

Якщо ж реакції від автора немає і термін минув, оголошення автоматично переходить у статус `archived`. Повне видалення запису та завантажених фотографій з бази відбувається ще через 30 днів після архівації. Це залишає користувачеві шанс повернути оголошення назад, якщо він просто проґавив сповіщення.

За весь цей процес відповідає спеціальна команда `archive_stale_reports`, яка запускається щогодини в окремому Docker-контейнері. Робота цієї команди також покрита окремими тестами [10]. Повний цикл життя оголошення від його публікації до очищення бази показано на діаграмі активності на рисунку 2.4.

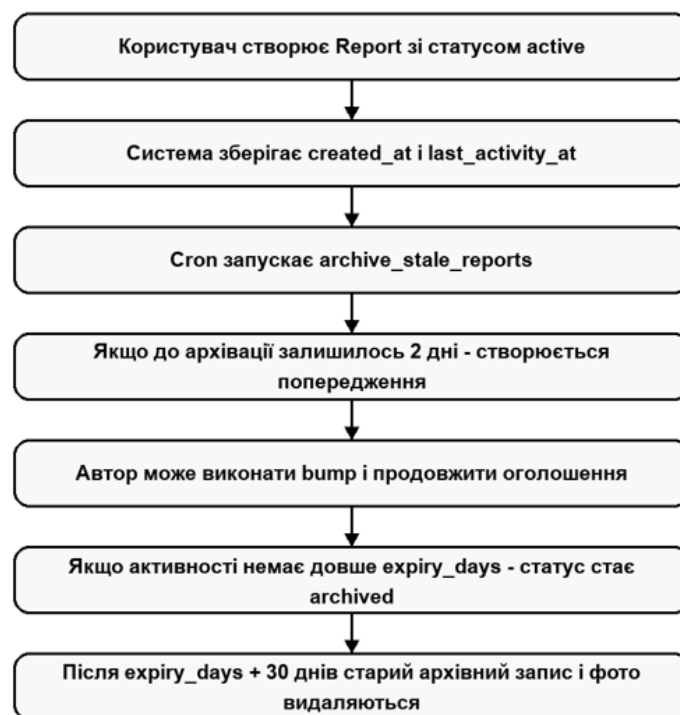


Рис. 2.4 – Схема життєвого циклу та архівації оголошень

Механізм сповіщень у системі має два незалежні рівні, які важливо не змішувати, оскільки вони вирішують різні задачі.

Перший рівень оновлення міської стрічки. Щойно створюється нове оголошення, Django Signal `send_report_notifications` надсилає подію у WebSocket-групу відповідного міста. Усі користувачі, які в цей момент переглядають стрічку, одразу бачать новий запис без перезавантаження сторінки. Паралельно цей же сигнал запускає Firebase push-сповіщення для всіх користувачів із відповідним містом у профілі, крім автора оголошення. Цей рівень працює за полем `city` і не знає нічого про координати чи радіуси

Другий рівень персональні підписки за геозоною. Сигнал `check_subscriptions_for_new_report` спрацьовує після збереження нового Report з ненульовими координатами. Система вибирає всі підписки, крім підписок автора, і через PostGIS функцію `Distance()` фільтрує ті, чия точка знаходиться у межах заданого радіуса від координат оголошення. Для тих підписників, які пройшли фільтр, через `bulk_create` одним запитом створюються записи Notification. Після цього кожному з них надсилається WebSocket-подія на персональний канал, а email відправляється у фоновому потоці, щоб не затримувати основний процес [5].

Таким чином, Firebase push і геопідписки є паралельними, але незалежними механізмами. Firebase орієнтований на місто і використовується для широкого охоплення, тоді як підписки дають точні персональні сповіщення на основі реальних координат і налаштованих фільтрів користувача.

На рисунку 2.5 наведено діаграму активності, що відображає обидва рівні механізму сповіщень і їхню взаємодію

					ІІЗ.КР.Б – 121 – 26 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		35

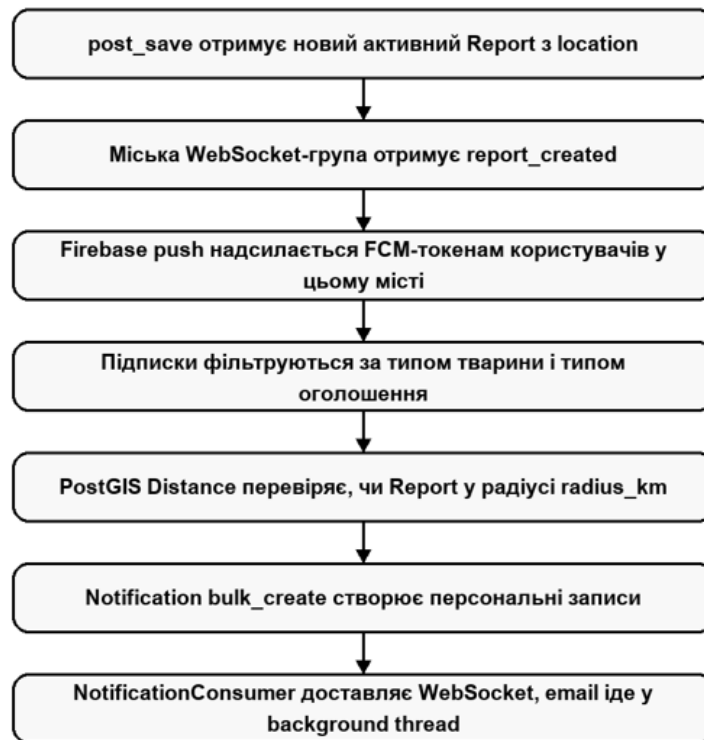


Рис. 2.5 – Схема механізму обробки та розсилки сповіщень

Система підтримує два способи входу, які ведуть до одного результату пари JWT-токенів.

Стандартний вхід через логін і пароль обробляється CustomLoginView. Перед видачею токенів перевіряються дві умови: email користувача підтверджений (email_verified = True) і акаунт активний (is_active = True). Якщо хоча б одна умова не виконана, відповідь містить відповідне повідомлення про причину відмови.

Вхід через Google реалізований через Firebase. Клієнт отримує Firebase ID token після успішної Google-авторизації і надсилає його на backend. Там firebase_admin.auth.verify_id_token() перевіряє підпис і витягує email. Якщо користувача з таким email ще не існує, система створює акаунт автоматично з email_verified = True. Після цього обидва шляхи сходяться: видаються однакові JWT access і refresh токени.

На рисунку 2.6 наведено діаграму активності процесу авторизації для обох сценаріїв.



Рис. 2.6 – Схема процесу реєстрації та автентифікації користувачів

Антиспам-логіка реалізована на двох рівнях обмеження частоти створення оголошень і система скарг від користувачів.

Перед збереженням нового оголошення метод `_check_spam_limits` рахує, скільки записів цей користувач створив за останні 15 хвилин. Для звичайного акаунту ліміт становить 3 оголошення, для організації 20.. Окремо перевіряється наявність посилання на збір коштів воно дозволене лише організаціям, і додатково валідується домен у серіалізаторі.

Система скарг працює незалежно від антиспаму. Будь-який авторизований користувач може поскаржитися на чуже активне оголошення, обравши причину такі як спам або непристойний контент на рівні бази даних не дозволяє подати більше однієї скарги від одного користувача на той самий запис. Коли кількість скарг досягає 10, оголошення автоматично переходить у статус `archived` і автор отримує системне сповіщення. Якщо протягом певного часу у конкретного користувача накопичується 20 заархівованих через скарги оголошень, система деактивує акаунт через `is_active = False`.

					ПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

На рисунку 2.6 наведено діаграму активності, що відображає процес перевірки антиспаму при створенні оголошення і механізм обробки скарг.

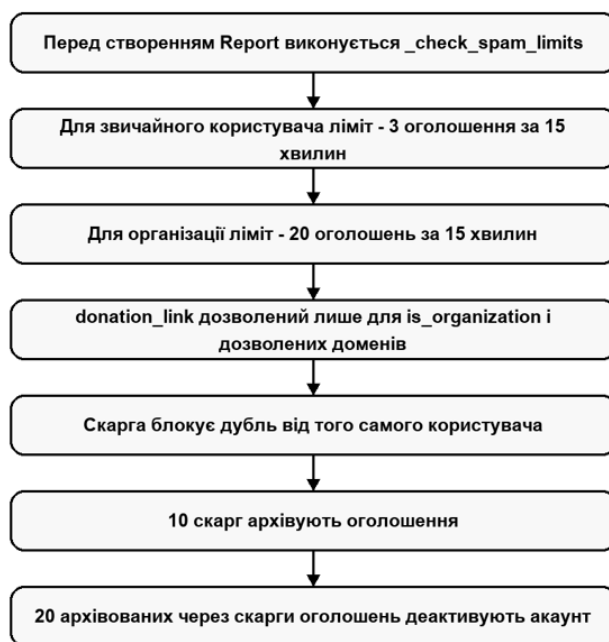


Рис. 2.7 – Схема логіки антиспаму та обробки скарг

2.4. Реалізація функціоналу

У цьому підрозділі розглянуто логіку роботи та особливості коду, який відповідає за головні функції сервера платформи FurryHelp. Повний код усього проєкту додається наприкінці роботи, тому тут описано лише найбільш важливі технічні рішення: налаштування автоматичного життєвого циклу оголошень, швидкий пошук радіусів через PostGIS та захист підключень WebSocket. Інші частини системи, такі як захист від спаму, робота черги повідомлень та скрипти архівації, детально розписані у Додатку А [5].

Під час створення моделі оголошення було вирішено об'єднати налаштування термінів дії з кодом обробки даних. Словник із лімітами часу для різних категорій постів розміщено прямо всередині класу моделі. Це дозволяє зберігати логіку перевірки оголошень в одному місці та не розпорошувати налаштування по різних файлах проєкту. Створений метод отримання терміну дії є універсальним: його використовує як консольна команда для перевірки застарілих постів, так і тести для перевірки коду, що значно спрощує

підтримку платформи та додавання нових типів оголошень у майбутньому, наприклад, окремих категорій для перетримки чи збору коштів на лікування.

Важливою частиною моделі є спільна робота полів, які відповідають за попередження користувача про архівацію та оновлення позиції оголошення в списку (функція `bump`). Коли користувач вирішує підняти своє оголошення, щоб воно знову стало помітним на карті, обидва ці прапорці автоматично скидаються до вихідного стану. Якщо цього не робити, то прапорець надісланого попередження залишився б активним назавжди. Через це в наступному циклі роботи системи автор оголошення просто не отримав би повідомлення про те, що його пост скоро закрийється, і тварина може залишитися без допомоги через застарілий статус публікації.

Головною особливістю системи надсилання геосповіщень є те, що всі складні математичні обчислення координат виконуються не на стороні мови Python, а безпосередньо у базі даних. Спеціальна функція `Distance` з розширення `PostGIS` рахує відстань між точкою підписки користувача та координатами нового оголошення під час виконання SQL-запиту. Програма переводить кілометри в метри на рівні бази даних за допомогою множення на тисячу та відсікає непотрібні записи ще до того, як вони потраплять у пам'ять сервера. Завдяки цьому Python-код отримує від бази даних уже готовий і відфільтрований список підписників, що значно економить оперативну пам'ять сервера і зменшує навантаження на мережу під час великої кількості запитів від користувачів [5].

Щоб отримати високу точність розрахунків на карті міста, координати підписок користувачів переводяться у спеціальний географічний тип даних за допомогою оператора явного приведення типів `Cast`. Це обов'язково для правильної роботи системи, тому що звичайні поля точок за замовчуванням рахують відстань на площині, не враховуючи форму Землі. У такій системі координати обчислюються в градусах, що дає великі помилки при спробі перевести їх у метри на різних широтах. Приведення до географічного типу забезпечує

					ІІЗ.КР.Б – 121 – 26 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		39

точний розрахунок відстані в метрах з урахуванням кривизни земної поверхні, що дозволяє чітко визначати межі радіуса пошуку.

З метою захисту особистих даних було налаштовано безпеку індивідуальних WebSocket-з'єднань. Вони працюють за динамічними адресами, які містять унікальний ID користувача в самому URL-шляху підключення. Якби такого захисту не було передбачено, будь-який зареєстрований користувач міг би читати приватні сповіщення та коментарі іншої особи, просто підставивши сторонній ID у рядок адреси під час ініціалізації з'єднання [24].

Для захисту від такої вразливості було створено асинхронний проміжний шар (Middleware). Він перехоплює запит на підключення на етапі рукоштовування, дістає безпечний JWT-токен, перевіряє його валідність і додає об'єкт перевіреного користувача у контекст запиту. Оскільки валідація токена відбувається до остаточного підключення до каналу, підробити дані у шляху стає неможливо. Якщо ID користувача в токені та ID у надісланому URL-шляху не збігаються, сервер одразу відхиляє запит та закриває з'єднання зі спеціальним кодом 4003. Такий підхід дозволяє клієнтському додатку на React чітко зрозуміти, що відбулася спроба зламу або підміни даних, а не звичайна втрата зв'язку через проблеми з мобільним інтернетом, і правильно показати відповідне попередження на екрані смартфона.

Висновки до другого розділу

У межах другого розділу було повністю спроектовано структуру бази даних, визначено основні варіанти використання платформи та реалізовано головні алгоритми сервісу.

База даних складається з 16 моделей, які розподілені між п'ятьма Django-додатками. Базовими сутностями виступають User, Report, Subscription та Notification. Геопросторові поля PointField з параметром SRID 4326 інтегровані у три моделі, серед яких оголошення, підписки та організації, що є основою для відображення карти та роботи механізму геопідписок..

					ПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		40

РОЗДІЛ 3. Інтерфейс та порядок роботи з системою

3.1. Порядок встановлення та налаштування системи

Розгортання FurryHelp побудоване на Docker Compose, і це не просто зручність без нього запустити платформу значно складніше. Справа в тому, що проєкт залежить не від одного процесу, а від чотирьох, які мають працювати одночасно і знати про існування одне одного: база даних з PostGIS, Redis, основний веб-сервер і окремий процес для автоматичної архівації. Docker Compose дозволяє описати всі ці залежності в одному файлі і підняти їх однією командою [5].

Сервіс db використовує образ `postgis/postgis:15-3.3`, а не звичайний `postgres`. Це принципово: стандартний PostgreSQL без розширення PostGIS просто не має геопросторових функцій, і геозапити в системі не запрацюють. Усі дані бази зберігаються у `volume postgres_data`, тому при перезапуску контейнера вони не зникають [5].

Сервіс `redis` на образі `redis:7-alpine` відповідає за `channel layer` без нього Django Channels не може передавати повідомлення між процесами, а значить WebSocket не працює взагалі [6].

Сервіс `web` запускає Daphne ASGI-сервер, який замінює класичний Gunicorn саме тому, що вміє тримати довготривалі WebSocket-з'єднання. Він слухає порт 8000 і обробляє одночасно і звичайні HTTP-запити до API, і WebSocket-з'єднання від клієнтів [21].

Сервіс `stop` використовує той самий Docker-образ, що і `web`, але замість сервера запускає `management command archive_stale_reports` у циклі з інтервалом, який задається через змінну середовища. За замовчуванням це 86400 секунд, тобто раз на добу. Саме цей процес відповідає за попередження авторів, архівування прострочених оголошень і видалення старих записів разом із файлами фотографій [10].

На рисунку 3.1 наведено діаграму розгортання, яка відображає зв'язки між усіма чотирма сервісами.

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		41

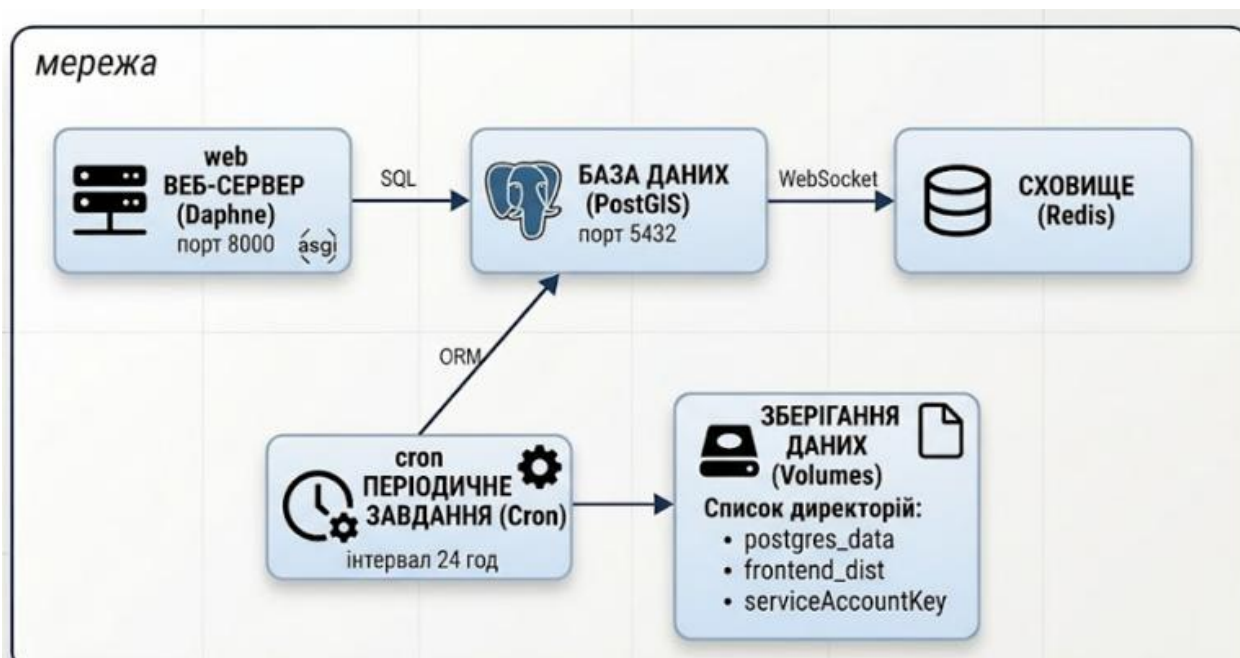


Рис. 3.1 – Діаграма розгортання Docker Compose

Розгортання застосунку застосунку FurryHelp було зроблено на хмарному сервері DigitalOcean. Сервер функціонує під керуванням операційної системи Ubuntu, а запуск усіх компонентів застосунку здійснюється за допомогою Docker Compose.

Компоненти production-середовища складаються з чотирьох Docker-контейнерів, кожен з яких виконує окрему функціональну роль. Характеристику компонентів наведено в таблиці 3.1.

Таблиця 3.1

Компоненти production-середовища FurryHelp

Контейнер	Технологія	Призначення
web	Django / Daphne (ASGI)	Обробка API-запитів, WebSocket-з'єднань, авторизації, оголошень, організацій, коментарів та сповіщень
db	PostgreSQL + PostGIS	Зберігання даних; геолокація, координати, карта та зони підписок

redis	Redis	Django Channels та WebSocket-логіка, зокрема real-time сповіщення
cron	Django Management Command	Виконання планових задач (архівація застарілих оголошень)

Контейнер web містить Django backend і запускається через Daphne як ASGI-застосунок. Він забезпечує обробку API-запитів, WebSocket-з'єднань, авторизацію, а також роботу з оголошеннями, організаціями, коментарями та сповіщеннями.

Контейнер db розгортає СУБД PostgreSQL з розширенням PostGIS, що застосовується для роботи з геолокаційними даними: координатами, картою та зонами підписок.

Контейнер redis використовується для забезпечення роботи Django Channels та реалізації WebSocket-логіки, зокрема real-time сповіщень користувачів.

Контейнер cron виконує планові фонові задачі. Зокрема, він запускає команду архівації застарілих або неактуальних оголошень.

Збирання та розгортання frontend:

Frontend-частина застосунку розроблена на базі бібліотеки React з використанням збирача Vite. Перед розгортанням виконується production-збірка командою: `npm run build`

У результаті виконання команди формується директорія `furry-client/dist`, яка містить статичні файли готового клієнтського застосунку. Ця директорія копіюється на сервер, після чого Django або Caddy роздає її як готовий frontend.

Процес розгортання автоматизовано засобами GitHub Actions. Загальна схема CI/CD-процесу реалізована у вигляді послідовного циклу, який почи-

нається на локальній машині розробника. Після внесення змін до коду та створення коміту виконується push до гілки main віддаленого репозиторію. Це є тригером для GitHub, який автоматично запускає CI pipeline для валідації та перевірки коректності внесених змін.

Наступний етап (CD) передбачає контрольований перехід до деплою. За умови успішного проходження всіх CI-перевірок, розробник вручну активує workflow «Deploy production». У межах цього процесу GitHub Actions ініціює захищене SSH-з'єднання з production-сервером.

Безпосереднє розгортання на сервері складається з трьох паралельних або послідовних кроків: синхронізації backend-коду за допомогою утиліти rsync, копіювання готової виробничої збірки frontend-частини, а також автоматичного виконання міграцій бази даних для актуалізації її структури.

Для підключення GitHub Actions до production-сервера використовуються repository secrets, що зберігаються виключно в налаштуваннях репозиторію GitHub. Жодні секретні дані не зберігаються безпосередньо у коді репозиторію.

Перелік використовуваних secrets:

- PROD_HOST — IP-адреса або доменне ім'я production-сервера;
- PROD_USER — ім'я користувача для SSH-підключення;
- PROD_SSH_KEY — закритий SSH-ключ для автентифікації;
- PROD_APP_DIR — шлях до директорії застосунку на сервері.

Таблиця 3.2

Змінні середовища та їхнє призначення

Змінна	Призначення	Примітка
DJANGO_SECRET_KEY	Секретний ключ Django для підписів і шифрування	У production має бути довгим випадковим рядком
DEBUG	Режим налагодження	False у production обов'язково

ALLOWED_HOSTS	Дозволені домени для запитів	Наприклад: furryhelp.com, www.furryhelp.com
DB_NAME / DB_USER / DB_PASSWORD	Параметри підключення до PostgreSQL	Передаються одночасно у сервіси db і web
DB_HOST / DB_PORT	Адреса бази даних з контейнера web	У Docker-мережі це db:5432
FRONTEND_URL	Базова URL для посилань у листах	Використовується у email verification та reset пароля
FIREBASE_CREDENTIALS_PATH	Шлях до service account JSON у контейнері	Наприклад: /app/config/serviceAccountKey.json
EMAIL_HOST / EMAIL_HOST_USER / EMAIL_HOST_PASSWORD	Параметри SMTP для відправки листів	У режимі розробки можна використовувати console backend
CORS_ALLOWED_ORIGINS	Дозволені origins для CORS	localhost у розробці, production-домени на сервері
ARCHIVE_STALE_REPORTS_INTERVAL_SECONDS	Інтервал між запусками архівації у сервісі cron	За замовчуванням 86400 секунд

Після підготовки конфігурації система розгортається покроково. Порядок важливий: спочатку потрібно зібрати frontend, потім підняти контейнери, і лише після цього застосовувати міграції бо без запущеної бази migrate просто завершиться з помилкою підключення.

Порядок розгортання системи

Крок	Команда або дія	Очікуваний результат
1	Створити файл .env у папці FurryHelp	Django отримує секрети, параметри БД, SMTP і Firebase
2	Додати serviceAccountKey.json або задати FIREBASE_SERVICE_ACCOUNT_HOST_PATH	Firebase Admin SDK успішно ініціалізується при старті
3	За потреби задати VITE_API_URL у furry-client	React-клієнт знатиме адресу backend API
4	Виконати npm run build у папці furry-client	Зібрано dist, який монтується у /app/frontend_dist всередині контейнера
5	Виконати docker compose up --build	Підняті всі чотири сервіси: db, redis, web, cron
6	Виконати docker compose exec web python manage.py migrate	Створено всі таблиці, індекси та геопросторові розширення
7	Виконати docker compose exec web python manage.py createsuperuser	Створено обліковий запис адміністратора Django
8	Відкрити /admin/ і основний домен у браузері	Адмін-панель доступна, SPA завантажується і відпрацьовує маршрути

Після успішного виконання всіх кроків система повністю готова до роботи. Перевірити коректність геопросторових функцій можна безпосередньо через адмін-панель якщо при відображенні моделей із PointField не виникає помилок, PostGIS ініціалізований правильно. Коректність WebSocket-з'єднань перевіряється відкриттям стрічки оголошень у двох вкладках і публікацією нового запису він має з'явитися в обох вкладках без перезавантаження сторінки [5].

3.2. Структура інтерфейсу та порядок взаємодії з платформою

Клієнтська частина FurryHelp реалізована як Single Page Application на React 19 з Vite. Усі сторінки підключаються через lazy() і Suspense це означає, що при першому завантаженні браузер отримує лише мінімальний bundle, а код конкретної сторінки завантажується лише тоді, коли користувач туди переходить. Маршрутизація описана в App.jsx через React Router v7, навігація розділена між верхньою панеллю Navbar і нижньою BottomNav така структура природна для мобільного застосунку, де великий палець не дотягується до верху екрана [7].

Захищені сторінки обгорнуті у ProtectedRoute, який перевіряє наявність токена перед рендером. При цьому враховується офлайн-сценарій: якщо мережі немає, але токен ще не протух, користувач залишається на сторінці замість того, щоб отримати редирект на логін.

Нижче описано кожен екран платформи з поясненням його функціональності та технічних деталей реалізації.

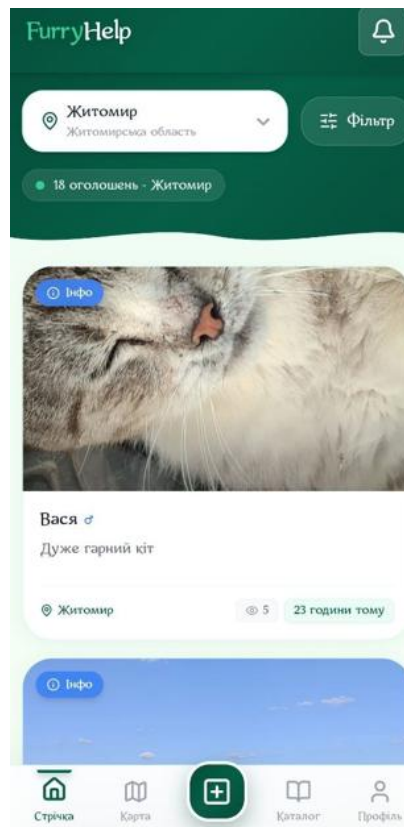


Рис. 3.2 – Головна стрічка оголошень

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		

Головна сторінка це стрічка активних оголошень із набором фільтрів: тип тварини, стать, тип оголошення і місто. Фільтри передаються як query-параметри до ReportViewSet, який обробляє їх через django-filter. Найцікавіша деталь тут WebSocket. Після завантаження сторінки клієнт відкриває з'єднання з CityConsumer для поточного міста. Коли хтось публікує нове оголошення, сигнал на backend надсилає подію у групу міста, і всі підключені клієнти отримують новий запис у стрічку без будь-якого перезавантаження або polling-запитів [24].

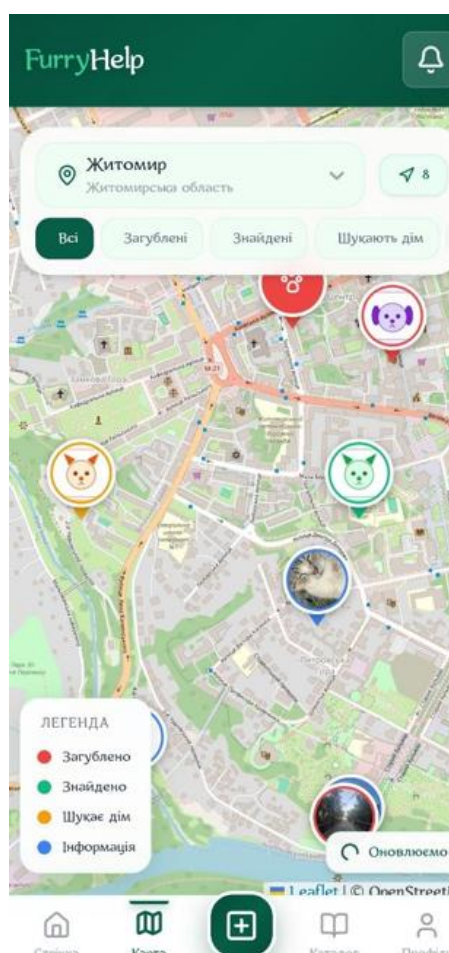


Рис. 3.3 – Інтерактивна карта оголошень

Карта побудована на Leaflet через бібліотеку react-leaflet. Вона не завантажує всі оголошення одразу натомість при кожному переміщенні або зумуванні клієнт надсилає поточний bbox (bounding box видимої області) на endpoint /api/reports/map-data/. Backend фільтрує лише ті записи з ненульовим

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						48
Змн.	Арк.	№ докум.	Підпис	Дата		

location, які потрапляють у ці межі, і повертає мінімальний набір даних для маркерів. Це дозволяє карті коректно працювати навіть при великій кількості оголошень у базі клієнт у будь-який момент бачить лише те, що у нього на екрані [7].

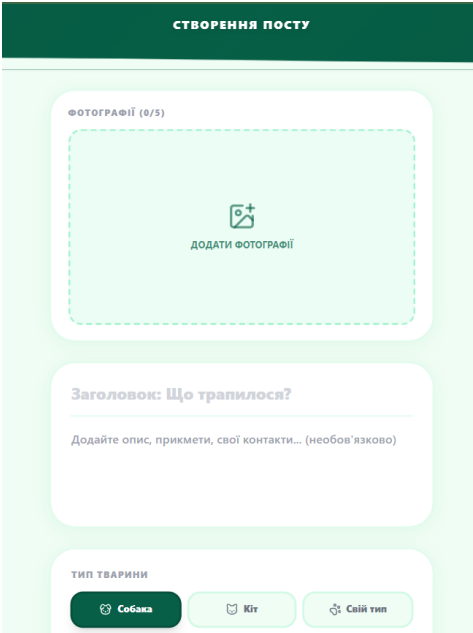


Рис. 3.4 – Заповнення полів основною інформацією про тварину

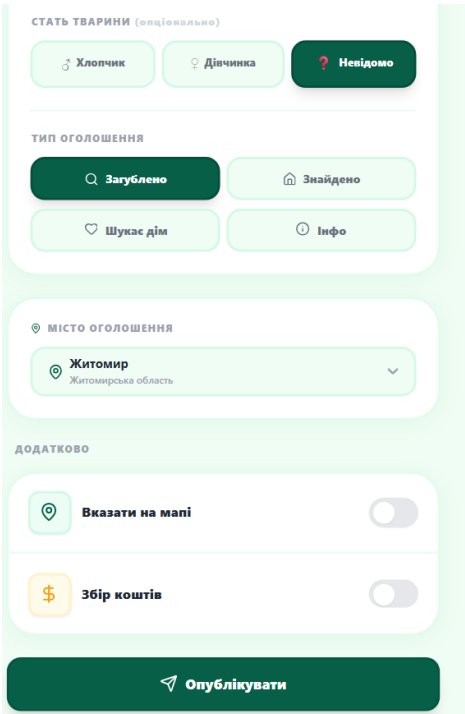


Рис. 3.5 – Додаткова інформація про тварину

Форма створення не розбита на кроки-wizard, а реалізована як єдина сторінка з логічними блоками: спочатку фотографії, потім опис і параметри тварини, далі місто і необов'язкова точка на карті, наприкінці додаткові опції типу посилання на збір коштів для організацій. Перед відправкою на сервер форма проходить валідацію на клієнті, а після антиспам-перевірку на backend.

Окремо реалізований офлайн-сценарій: якщо при спробі публікації мережа недоступна, дані форми разом із фотографіями у форматі base64 зберігаються в IndexedDB. Коли з'єднання відновлюється, offlineQueue.js автоматично формує FormData і відправляє збережені записи на сервер. Це особливо важливо для мобільного Android-додатку, де мережа може зникати прямо під час заповнення форми [23].

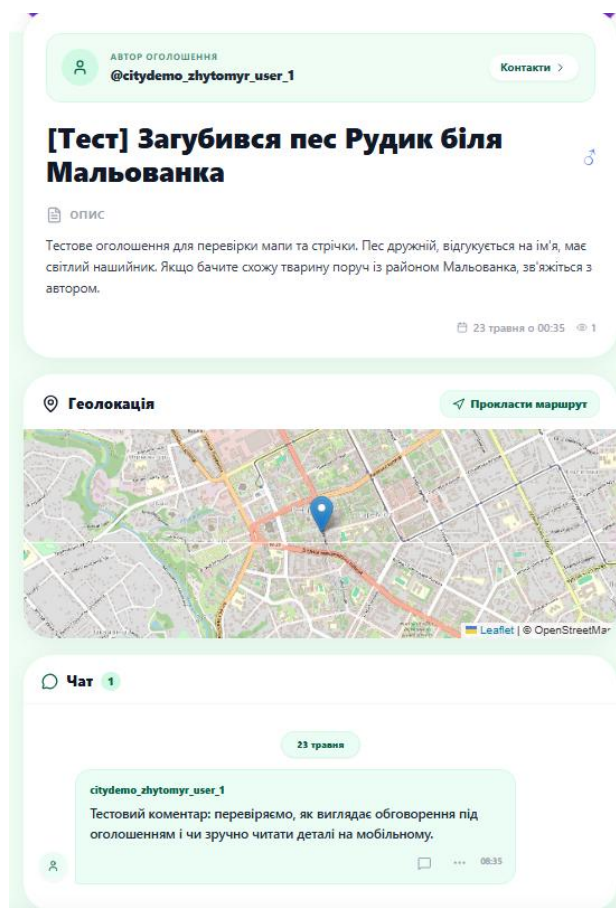


Рис. 3.6 – Деталі оголошення

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

Сторінка оголошення містить повну галерею фотографій, статус запису, міні-карту з маркером місця події і кнопку побудови маршруту через Google Maps. Нижче розташовані коментарі, які оновлюються в реальному часі через WebSocket-з'єднання з CommentConsumer новий коментар з'являється у всіх відкритих вкладках без перезавантаження [24].

Для авторизованих користувачів відображається кнопка скарги з вибором причини. Для автора можливість позначити оголошення як вирішене (resolved) і виконати bump, якщо прийшло попередження про наближення архівації. Якщо контакти власника приховані, замість номера телефону відображається кнопка надсилання ContactRequest.

Додати нову зону пошуку

1/10 зон використано

НАЗВА ЗОНИ

Мій дім

ТВАРИНА

Будь-яка

ТИП ОГЛОШЕННЯ

Всі типи

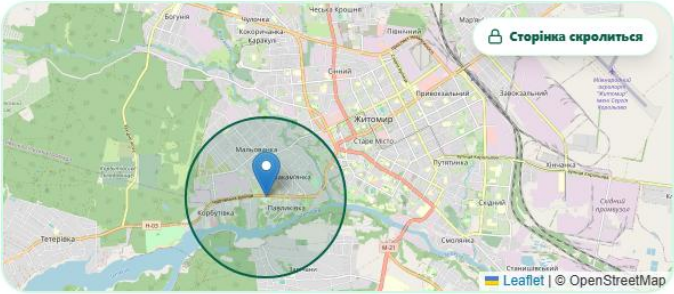
РАДІУС СПОВІЩЕНЬ

1.7 км

100 м

100 км

Сторінка скролиться



Створити підписку

Рис. 3.7 – Підписки на геозони

На цій сторінці користувач налаштовує персональні зони сповіщень. Інтерфейс дозволяє поставити маркер на карті або використати поточне місцезнаходження через Capacitor Geolocation, задати радіус у кілометрах і обрати

фільтри: тип тварини (dog, cat, other або any) і тип оголошення (lost, found або any). Після збереження на backend створюється запис Subscription з PointField(srid=4326). При появі нового оголошення PostGIS обчислює відстань між координатами оголошення і центром кожної підписки і якщо вона менша за заданий радіус, підписник отримує сповіщення [5].

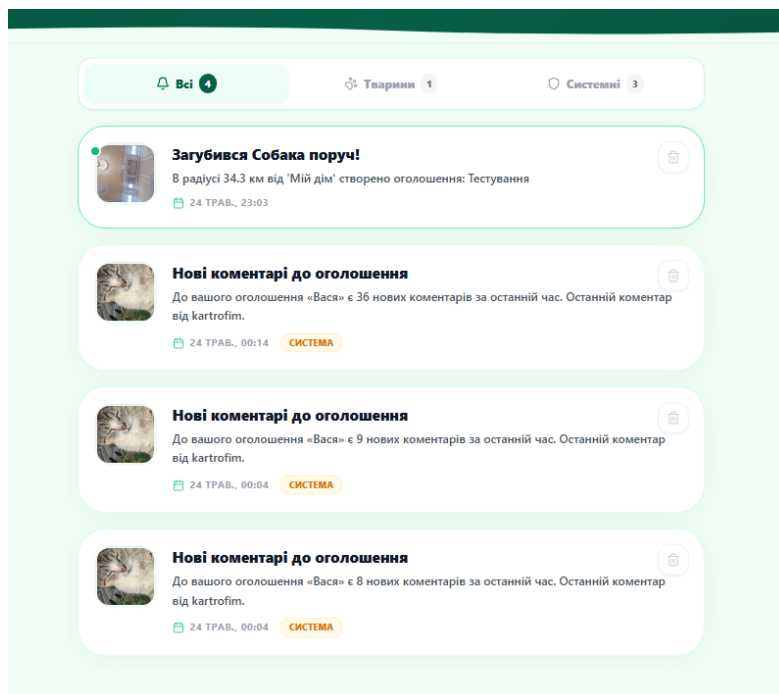


Рис. 3.8 – Центр повідомлень

Сторінка повідомлень агрегує всі події двох типів. Сповіщення типу animal створюються автоматично при спрацюванні геопідписки вони містять назву оголошення і відстань до нього. Сповіщення типу system надходять від платформи: архівація оголошення, результат заявки організації, блокування акаунту через накопичені скарги. Кожне можна позначити прочитаним або видалити. Для оголошень із попередженням про наближення архівації прямо з картки повідомлення доступна кнопка bump без необхідності переходити на сторінку оголошення.

Нові сповіщення доставляються через персональний WebSocket-канал NotificationConsumer, захищений перевіркою JWT. Лічильник непрочитаних оновлюється в реальному часі [24].

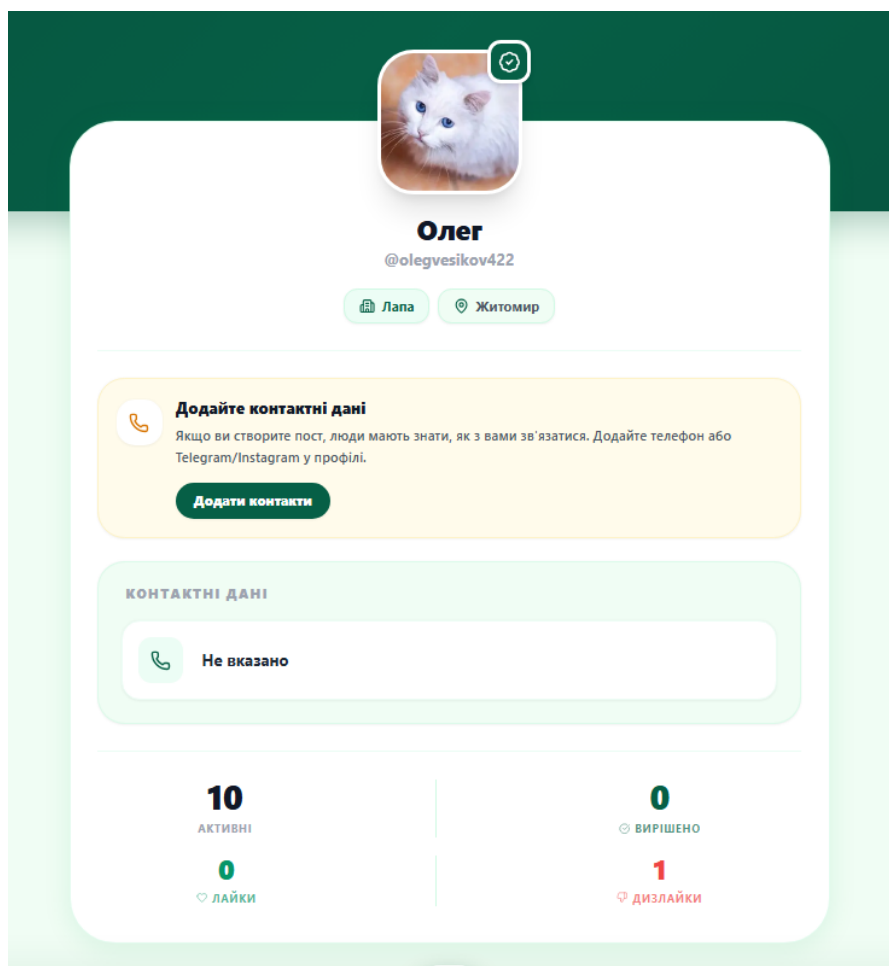


Рис. 3.9 – Контактна інформація користувача

Розумне нагадування: Якщо людина забула вказати свій телефон або нік в Інстаграмі/Телеграмі, додаток показує помітне жовте віконце: «Додайте контактні дані». Це зроблено для того, щоб коли користувач створить пост про загублену тварину, інші волонтери могли миттєво з ним зв'язатися і допомогти.

Статистика та репутація: Знизу видно чотири цифри. Вони показують активність людини: скільки оголошень на карті вона зараз тримає (10 активних), скільком тваринам уже реально допомогла (вирішено), а також лайки та

дизлайки від інших людей. Це допомагає одразу зрозуміти, чи можна довіряти цьому користувачу.

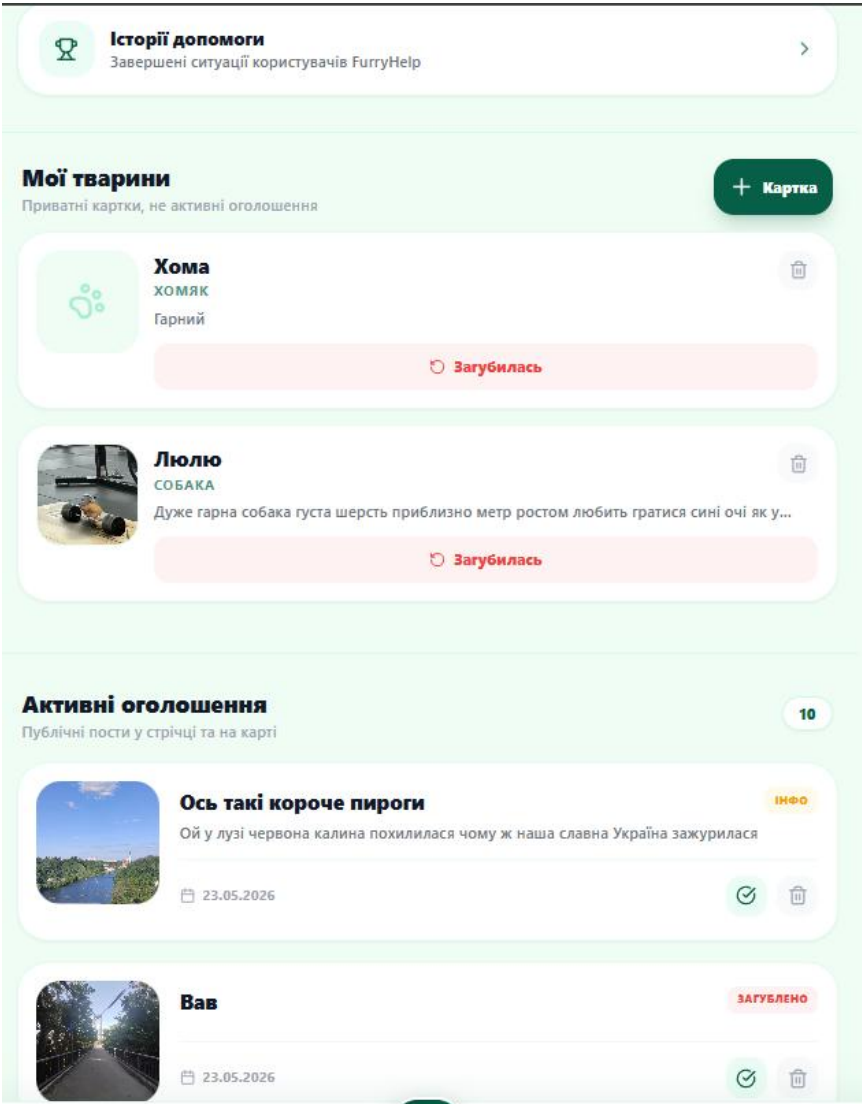


Рис. 3.10 – Додавання власних тварин в профіль та активні оголошення користувача

Блок "Мої тварини" Сюди користувач може просто додати своїх домашніх улюбленців, завантажити їхні фото та описи. Ці картки бачить тільки сам власник. Але якщо, тварина втече, не треба в паніці заново створювати оголошення і завантажувати фото. Можна просто натиснути одну червону кнопку «Загубилась» і додаток сам миттєво створить публічний пост на карті міста.

Блок "Активні оголошення" Тут зібрані всі пости, які користувач уже виставив на загальну карту для інших. Прямо звідси користувач може в один клік видалити пост або натиснути галочку «Вирішено», якщо тваринку вже знайшли, щоб мітка зникла з карти і не плутала інших волонтерів.

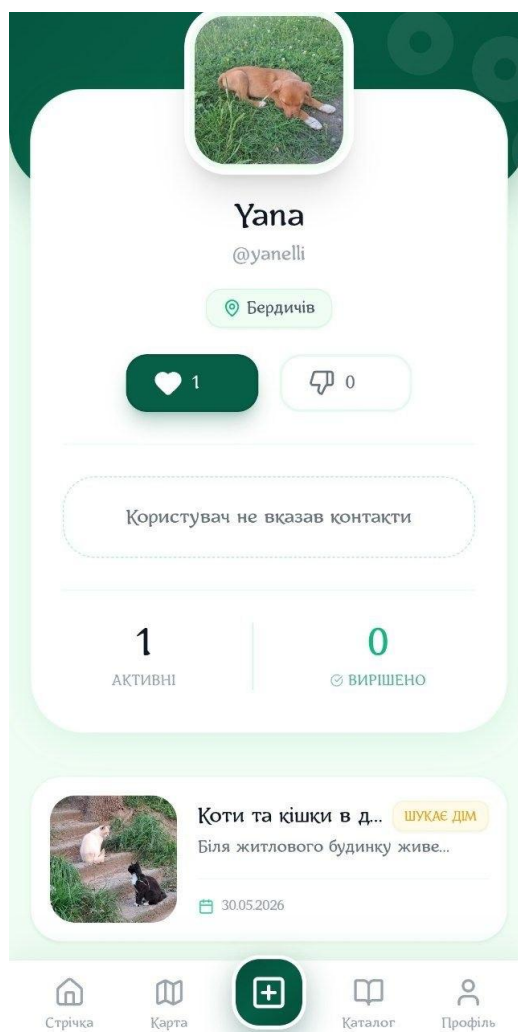


Рис. 3.11 – Публічний профіль

Публічна сторінка іншого користувача показує ім'я, аватар, місто і активні оголошення, також публічний профіль має систему довіри користувачі можуть ставити лайки або дизлайки якщо їм ця людина не сподобалося і таким чином інші користувачі будуть знати з ким вони мають справу, і чи довіряти тварину.

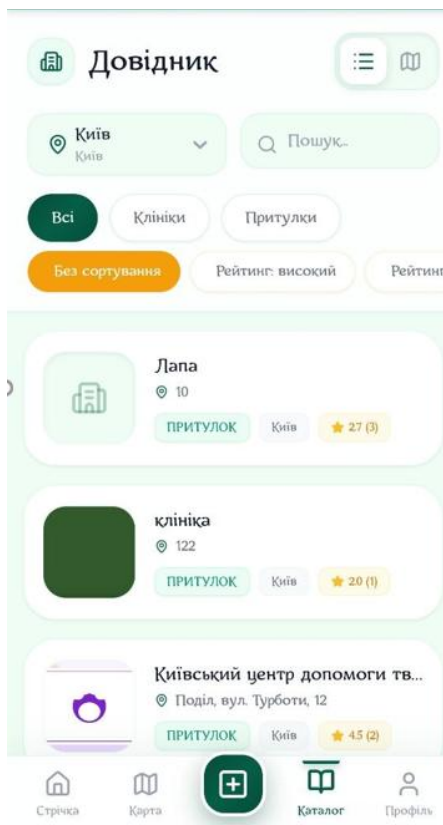


Рис. 3.12 – Каталог організацій.

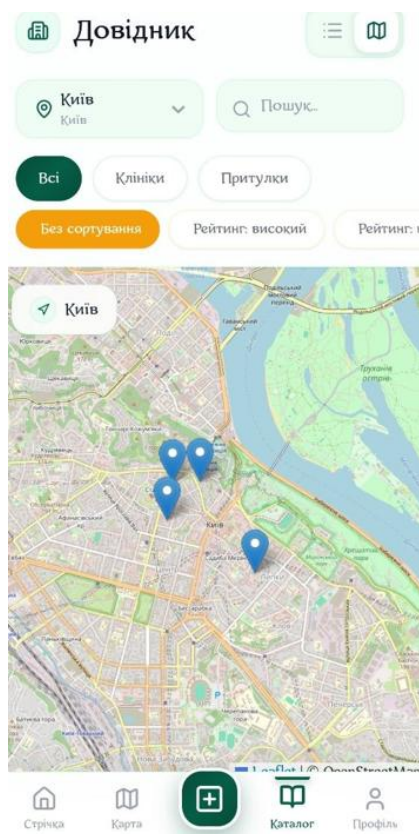


Рис. 3.13 – Місцезнаходження організації на карті

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						56
Змн.	Арк.	№ докум.	Підпис	Дата		

Каталог верифікованих організацій відкривається у двох режимах: список і карта. У режимі списку організації фільтруються за містом і категорією (shelter, clinic) та рейтингом залишеними людьми. У режимі карти маркери беруть координати з поля Organization.location і відображаються через Leaflet так само як і оголошення, але через окремий endpoint OrganizationViewSet [8].

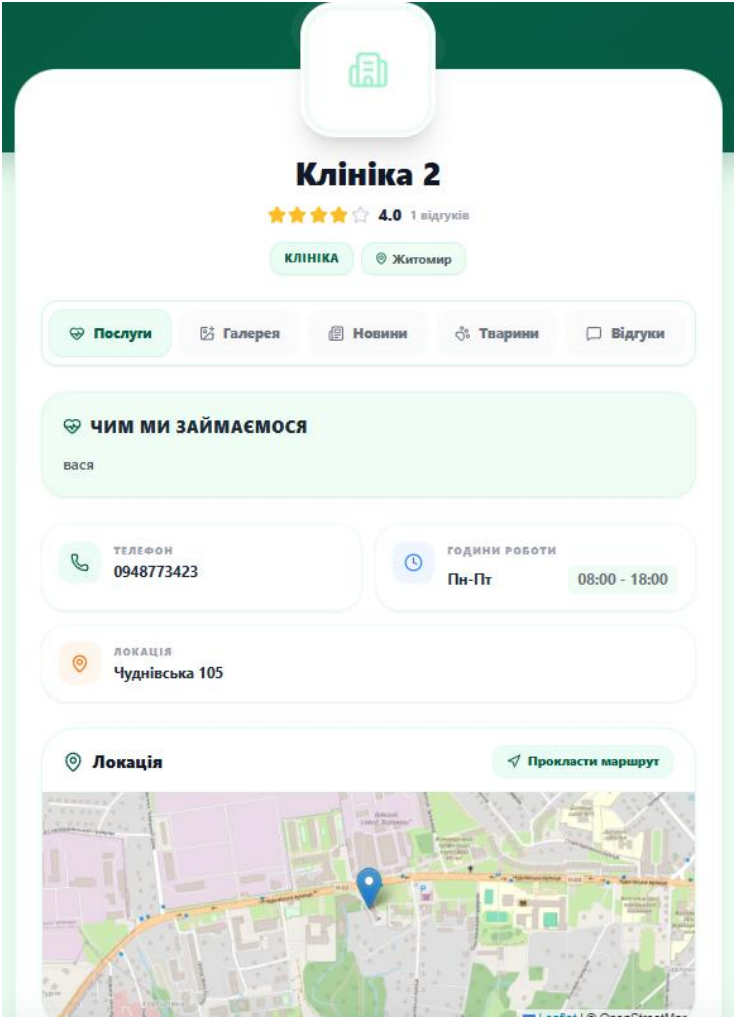


Рис. 3.14 – Сторінка організації

Профіль організації містить повний опис, контакти, адресу, графік роботи і міні-карту з розташуванням. Нижче стрічка публікацій організації (OrganizationPost), список пов'язаних оголошень і відгуки користувачів із середнім рейтингом, галерею де організація може публікувати фото, новини публікувати які можуть бути важливі для користувача та тварини які можуть бути в цій організації для адопції. Кожен користувач може залишити лише

один відгук це забезпечено UniqueConstraint на рівні бази. Власник організації не може відгукуватися про власний профіль.

Анкета організації

Заповніть дані, щоб отримати статус верифікованої організації.

УВАГА: Всі дані будуть ретельно перевірені адміністратором. Створення фейкових організацій з метою введення в оману користувачів або шахрайства може тягнути за собою кримінальну відповідальність.

Анкета призначена ВИКЛЮЧНО для офіційних притулків та клінік.

ОСНОВНІ ДАНІ

Назва (напр. ГО "Щаслива Лапа")

Код ЄДРПОУ (обов)

Притулок

Офіційний сайт (необов'язково)

Офіційні соцмережі (Instagram/Facebook) - обс

Офіційний телефон (для перевірки)

Короткий опис вашої діяльності...
(необов'язково)

0/1000

Рис. 3.15 – Заявка на отримання статусу організації

Форма заявки збирає всю необхідну інформацію: назву, код ЄДРПОУ, посилання на соціальну мережу і сайт, телефон, місто, адресу, опис діяльності, категорію і графік роботи. Координати організації вказуються через інтерактивну карту або вводяться вручну. Після подання створюється запис OrganizationRequest зі статусом pending. Один акаунт може подати лише одну заявку це гарантовано OneToOneField між OrganizationRequest і User.

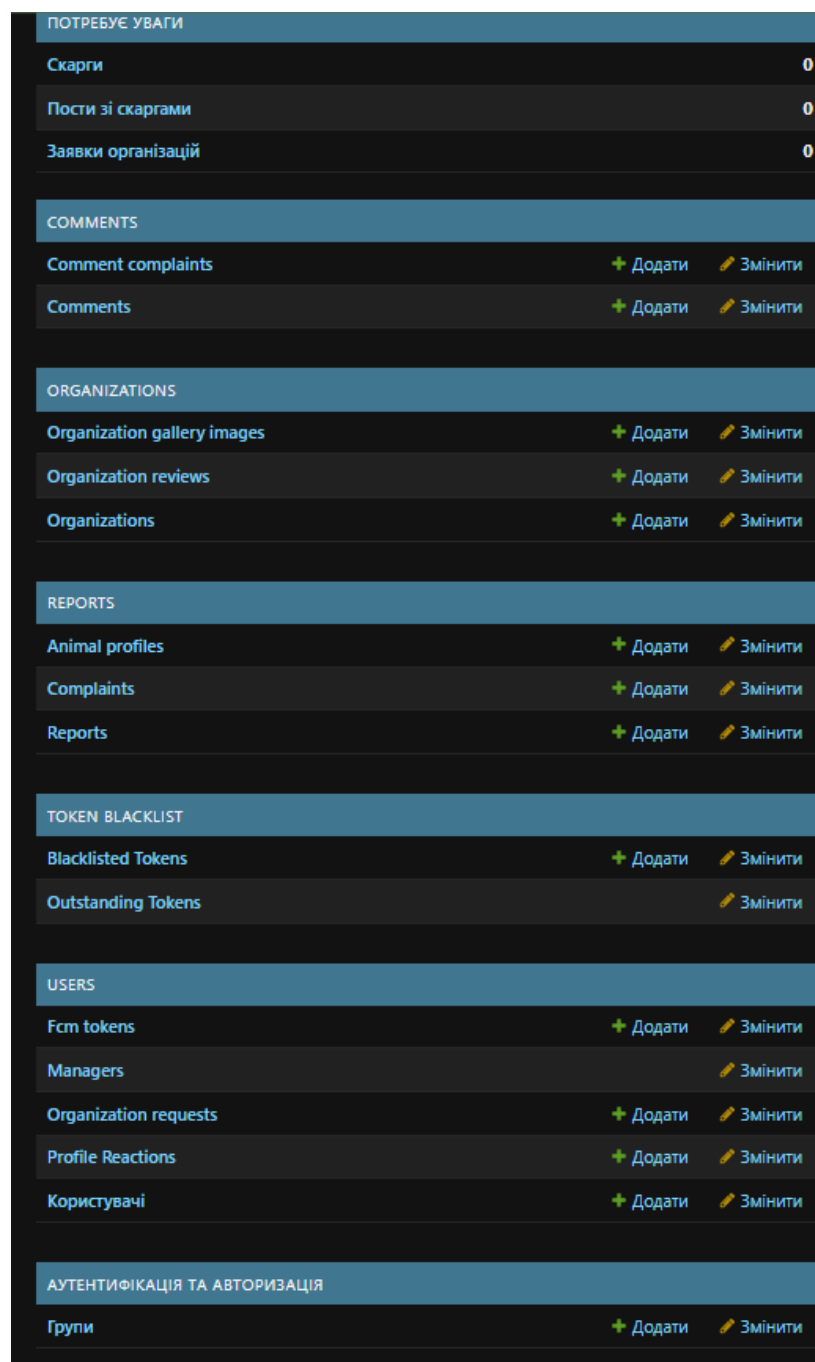


Рис. 3.17 – Адміністративна панель Django

Django admin використовується для операцій, які не мають інтерфейсу у публічній частині: управління користувачами, призначення ролі менеджера, розгляд заявок організацій, обробка скарг і блокування акаунтів. Для менеджерів queryset у кожному ModelAdmin автоматично фільтрується через метод `manager_city_q` менеджер фізично не може побачити або змінити об'єкти за межами свого `manager_region`, навіть якщо введе URL вручну.

3.3. Тестування роботи програмного продукту

Тестування FurryHelp розділене на два рівні, які доповнюють один одного. Автоматизовані тести перевіряють бізнес-правила на рівні backend ті місця, де одна неточна зміна в коді може непомітно зламати логіку архівації, скарг або bump. Ручне тестування охоплює повні користувацькі сценарії, де потрібно оцінити не лише відповідь API, а й реакцію інтерфейсу, поведінку WebSocket і роботу push-сповіщень [24].

Загалом у проєкті реалізовано 37 автоматизованих тестів: 33 у файлі reports/test_lifecycle.py і 4 у organizations/test_reviews.py. Для їхнього запуску необхідне середовище з встановленими GeoDjango-залежностями зокрема GDAL і PostGIS. У чистому локальному середовищі без цих бібліотек Django зупиняється ще на етапі ініціалізації GIS backend, навіть не доходячи до запуску тестів. Команда для запуску всього набору: [4].

```
python manage.py test reports.test_lifecycle organizations.test_reviews --
noinput
```

У правильно налаштованому середовищі очікується проходження всіх 37 тестів без помилок.

Таблиця 3.4

Групи автоматизованих тестів

К-сть	Група тестів	Що перевіряє
6	ArchiveReportsTest	Автоархівацію прострочених постів (різні терміни для info та adoption), захист свіжих записів та створення системного сповіщення.
4	ExpiryWarningTest	Надсилання попередження за 2 дні до архівації, блокування повторних нотифікацій та ігнорування свіжих оголошень.
8	BumpApiTest	Оновлення дати активності після підняття (bump), доступність фічі лише автору, захист від дабл-бампу та скидання ліміту в новому циклі.

4	CommentActivityTest	Оновлення last_activity_at після коментаря та захист активного оголошення від передчасної архівації за рахунок спілкування в ньому.
6	ComplaintTest	Автоархівацію при ≥ 10 скаргах (і стабільність при 9), блокування повторних скарг від одного юзера та заборону скаржитися на власні пости.
3	DryRunTest	Повну відсутність змін у базі даних та відсутність генерації сповіщень при запуску cron-команди з прапором --dry-run.
2	ResolveTest	Успішне закриття оголошення автором (переведення в статус resolved) та суворе блокування цієї дії для сторонніх користувачів.
4	OrganizationReviewAPITests	Обмеження «один відгук від одного юзера», розрахунок середнього рейтингу організації, заборону самовідгуків та права на редагування.

Варто окремо зазначити, що тести перевіряють не лише позитивні сценарії, де все йде за планом, а й граничні умови та заборони. Саме ці випадки найчастіше ламаються при рефакторингу: не власник не може виконати bump або resolve, дублікати скарг відхиляються з коректним HTTP-статусом, власник не може скаржитися на власне оголошення, а організація не може залишати відгук у своєму ж профілі.

Поряд з автоматизованими тестами проводилося ручне тестування повних користувацьких сценаріїв. Воно охоплює ті частини системи, які залежать від зовнішніх сервісів або вимагають перевірки поведінки інтерфейсу в реальних умовах.

Таблиця 3.5

Сценарії ручного тестування

Сценарій	Кроки	Очікуваний результат
Реєстрація	Заповнити username, email, пароль, місто; перейти за посиланням підтвердження	Акаунт створено, email_verified = True, вхід дозволено
Вхід через Google	Натиснути "Увійти через Google" у web або Android-оболонці	Firebase token перевірено на backend, видано JWT
Створення оголошення	Додати фото, опис, місто, тип тварини, точку на карті; натиснути публікацію	Створено Report, фото стиснуто у WebP, запис з'явився у стрічці
Офлайн-створення	Вимкнути мережу, заповнити і надіслати форму, увімкнути мережу	Запис збережено в IndexedDB, після відновлення мережі синхронізовано автоматично
Карта оголошень	Відкрити /map, змінити місто, пересунути карту	API отримує bbox, маркери відповідають поточному viewport і фільтру міста
Геопідписка	Створити Subscription з радіусом, потім іншим акаунтом опублікувати Report у цій зоні	Отримано Notification і WebSocket-подію на персональному каналі
Коментарі в реальному часі	Відкрити деталі оголошення у двох вкладках, додати коментар в одній	Коментар з'являється в обох вкладках без перезавантаження
Bump оголошення	Змодельовати попередження про архівацію, натиснути кнопку продовження	last_activity_at оновлено, сповіщення про попередження прибрано
Система скарг	Подати 10 скарг на одне оголошення з різних акаунтів	Оголошення переходить у archived, автор отримує системне сповіщення
Заявка організації	Подати заявку, схвалити її в Django admin	Створено Organization, user.is_organization = True, користувач отримав сповіщення
Контактний запит	Приховати контакти, надіслати ContactRequest з іншого акаунта, прийняти	Після прийняття відправник бачить контактні дані

Результати тестування підтверджують, що критичні бізнес-правила системи захищені на рівні backend і не залежать від поведінки клієнта. Навіть якщо frontend з якоїсь причини надішле некоректний запит наприклад, спробує виконати bump від імені не автора або подати повторну скаргу сервер відхилить його з відповідним HTTP-статусом.

Для повноцінної перевірки в умовах, наближених до production, доцільно додатково провести інтеграційне тестування WebSocket-з'єднань під навантаженням, Firebase push-сповіщень через реальні FCM-токени, відправки email через справжній SMTP і поведінки Android-оболонки на фізичному пристрої.

Висновки до третього розділу

У третьому розділі описано порядок розгортання системи, структуру інтерфейсу і результати тестування.

Розгортання платформи виконується через Docker Compose і охоплює чотири сервіси: PostgreSQL з PostGIS, Redis, Daphne ASGI-сервер і окремий стоп-контейнер для архівації. Така конфігурація дозволяє відтворити повне робоче середовище однією командою незалежно від операційної системи розробника або сервера [5].

Інтерфейс реалізований як мобільно-орієнтований SPA з підтримкою офлайн-сценарію через IndexedDB. Описано інтерфейс платформи, кожен з яких відповідає окремому сценарію взаємодії користувача з системою [23].

За результатами тестування 37 автоматизованих тестів проходять успішно і покривають усі критичні бізнес-правила: архівацію, bump, скарги, коментарі і відгуки організацій, включно з граничними умовами і заборонами. Ручне тестування підтвердило коректну роботу WebSocket, геопідписок, офлайн-черги і Google-авторизації у реальних умовах [24].

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		64

ВИСНОВКИ

У ході виконання дипломної роботи розроблено повноцінну клієнт-серверну платформу FurryHelp, яка вирішує практичну задачу координації допомоги безпритульним і загубленим тваринам засобами сучасних веб-технологій.

На етапі аналізу предметної області встановлено, що існуючі рішення спеціалізовані lost-pet сервіси, каталоги адопції та неформальні Facebook-групи не забезпечують комплексного підходу: відсутня геопідписка за радіусом, немає структурованого life cycle оголошення, регіональна модерація і верифікація організацій або відсутні, або реалізовані частково

На етапі проєктування сформовано архітектуру системи на основі ASGI-стеку з Django Channels і Redis, що забезпечило підтримку WebSocket поряд зі звичайними HTTP-запитами. Вибір PostgreSQL з розширенням PostGIS дозволив реалізувати геопросторові запити на рівні СУБД зокрема фільтрацію підписок за радіусом через функцію Distance() без перебору даних у Python [5].

У процесі розробки реалізовано такі ключові компоненти: структуровані оголошення з автоматичним life cycle і системою bump; двохрівневий механізм геосповіщень через PostGIS Distance і Firebase Cloud Messaging; верифікацію організацій через заявку і Django Signal; регіональну модерацію через проху-модель менеджера з обмеженням queryset; офлайн-чергу на клієнті через IndexedDB; Android-оболонку на Capacitor з нативною геолокацією і Firebase Auth; розгортання через Docker Compose з чотирма ізольованими сервісами [5].

Написано 37 автоматизованих тестів, які покривають life cycle оголошення, систему скарг, механізм bump, коментарі та відгуки організацій включно з граничними умовами і заборонами.

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		65

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Діаграма діяльності UML (Activity Diagram) [Електронний ресурс]. – Режим доступу: <http://www.tsatu.edu.ua/kn/wp-content/uploads/sites/16/laboratorna-robota-78-diahrama-dijalnosti.pdf>
2. Django REST Framework: офіційна документація [Електронний ресурс]. – Режим доступу: <https://www.django-rest-framework.org/>
3. GeoDjango: геопросторові можливості Django [Електронний ресурс]. – Режим доступу: <https://docs.djangoproject.com/en/5.0/ref/contrib/gis/>
4. PostGIS: геопросторове розширення PostgreSQL [Електронний ресурс]. – Режим доступу: <https://postgis.net/documentation/>
5. Django Channels: асинхронна обробка WebSocket [Електронний ресурс]. – Режим доступу: <https://channels.readthedocs.io/en/stable/>
6. React: офіційна документація [Електронний ресурс]. – Режим доступу: <https://react.dev/>
7. Leaflet: відкрита бібліотека для інтерактивних карт [Електронний ресурс]. – Режим доступу: <https://leafletjs.com/>
8. Redis: система кешування та pub/sub [Електронний ресурс]. – Режим доступу: <https://redis.io/docs/>
9. Docker: контейнеризація додатків [Електронний ресурс]. – Режим доступу: <https://docs.docker.com/>
10. Швець О. Занурення в патерни проектування / Олександр Швець. – 2022. – 396 с.
11. Мартін Р. Чиста архітектура: мистецтво створення програмного забезпечення / Роберт Мартін. – Харків: Фабула, 2019. – 368 с.
12. Capacitor: крос-платформна розробка мобільних застосунків [Електронний ресурс]. – Режим доступу: <https://capacitorjs.com/docs>
13. Firebase Cloud Messaging: офіційна документація [Електронний ресурс]. – Режим доступу: <https://firebase.google.com/docs/cloud-messaging>
14. Firebase Authentication: офіційна документація [Електронний ресурс]. – Режим доступу: <https://firebase.google.com/docs/auth>

					ПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						66
Змн.	Арк.	№ докум.	Підпис	Дата		

- 15.Vite: інструмент збірки для сучасного веб-розробництва [Електронний ресурс]. – Режим доступу: <https://vitejs.dev/guide/>
- 16.React Router: документація маршрутизації [Електронний ресурс]. – Режим доступу: <https://reactrouter.com/en/main>
- 17.Simple JWT: JWT-автентифікація для Django REST Framework [Електронний ресурс]. – Режим доступу: <https://django-rest-framework-simplejwt.readthedocs.io/>
- 18.PostgreSQL: офіційна документація СУБД [Електронний ресурс]. – Режим доступу: <https://www.postgresql.org/docs/>
- 19.Django: офіційна документація веб-фреймворку [Електронний ресурс]. – Режим доступу: <https://docs.djangoproject.com/en/5.0/>
- 20.Daphne: ASGI-сервер для Django Channels [Електронний ресурс]. – Режим доступу: <https://github.com/django/daphne>
- 21.Pillow: бібліотека обробки зображень для Python [Електронний ресурс]. – Режим доступу: <https://pillow.readthedocs.io/en/stable/>
- 22.IndexedDB API: документація Mozilla Developer Network [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API
- 23.WebSocket API: документація Mozilla Developer Network [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
- 24.OWASP: рекомендації щодо безпеки веб-застосунків [Електронний ресурс]. – Режим доступу: <https://owasp.org/www-project-top-ten/>
- 25.Фаулер М. Рефакторинг: вдосконалення існуючого коду / Мартін Фаулер. – Київ: Видавництво «Наш формат», 2020. – 448 с.
- 26.Петфайндер: платформа пошуку тварин для адопції [Електронний ресурс]. – Режим доступу: <https://www.petfinder.com/>
- 27.Lost My Doggie: сервіс пошуку загублених домашніх тварин [Електронний ресурс]. – Режим доступу: <https://www.lostmydoggie.com/>

- 28.WhiteNoise: обслуговування статичних файлів у Python-застосунках [Електронний ресурс]. – Режим доступу: <https://whitenoise.readthedocs.io/>
- 29.react-leaflet: React-компоненти для Leaflet-карт [Електронний ресурс]. – Режим доступу: <https://react-leaflet.js.org/>
- 30.Brevo: Transactional Email API and SMTP Documentation [Електронний ресурс]. – Режим доступу: <https://developers.brevo.com/docs/getting-started>

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		68

ДОДАТКИ

		.			ІПЗ.КР.Б – 121 – 26 - ІЗ	Арк.
						69
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТОК А

У додатку наведено ключові фрагменти реалізації платформи FurryHelp. Вони показують, як у проєкті організовано життєвий цикл оголошення, анти-спам-обмеження, геолокаційні сповіщення, WebSocket-з'єднання та контейнеризоване розгортання.

Лістинг А.1 – Модель Report і поля життєвого циклу оголошення

```
class Report(models.Model):
    ANIMAL_TYPE_CHOICES = (
        ('dog', 'Собака'),
        ('cat', 'Кіт'),
        ('other', 'Інше'),
    )

    ANIMAL_GENDER_CHOICES = (
        ('male', 'Хлопчик ♂'),
        ('female', 'Дівчинка ♀'),
        ('unknown', 'Невідомо'),
    )

    REPORT_TYPE_CHOICES = (
        ('lost', 'Загубився'),
        ('found', 'Знайшовся'),
        ('adoption', 'Шукає дім'),
        ('info', 'Інформація'),
    )

    STATUS_CHOICES = (
        ('active', 'Активний ●'),
        ('resolved', 'Врятовано ●'),
        ('archived', 'Архів ●'),
    )

    EXPIRY_DAYS = {
        'lost': 30,
        'found': 30,
        'adoption': 60,
        'info': 30,
    }

    EXPIRY_WARNING_DAYS_BEFORE = 2

    user = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        related_name='reports'
    )

    animal_profile = models.ForeignKey(
        'AnimalProfile',
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        related_name='reports'
    )
```

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		70

```

title = models.CharField(
    max_length=100,
    verbose_name="Заголовок",
    help_text="Максимум 100 символів"
)
description = models.TextField(
    max_length=1000,
    blank=True,
    default='',
    verbose_name="Опис",
    help_text="Максимум 1000 символів"
)
city = models.CharField(max_length=100, default="Київ", verbose_name="Місто")
views_count = models.PositiveIntegerField(default=0, verbose_name="Перегляди")

animal_type = models.CharField(
    max_length=10,
    choices=ANIMAL_TYPE_CHOICES,
    default='dog'
)
custom_animal_type = models.CharField(
    max_length=40,
    blank=True,
    default='',
    verbose_name="Уточнення типу тварини"
)

animal_gender = models.CharField(
    max_length=10,
    choices=ANIMAL_GENDER_CHOICES,
    default='unknown',
    verbose_name='Стать тварини',
    help_text='Опціонально: хлопчик, дівчинка або невідомо'
)

report_type = models.CharField(
    max_length=20,
    choices=REPORT_TYPE_CHOICES,
    default='lost'
)

status = models.CharField(
    max_length=20,
    choices=STATUS_CHOICES,
    default='active'
)

donation_link = models.URLField(blank=True, null=True)

image = models.ImageField(
    upload_to='reports/',
    null=True,
    blank=True,
    help_text="Фото тварини"
)

location = models.PointField(null=True, blank=True)

created_at = models.DateTimeField(
    default=timezone.now,
    verbose_name="Створено",
)

# fields for the autoarchive system
last_activity_at = models.DateTimeField(
    default=timezone.now,
    verbose_name="Остання активність",
    help_text="Оновлюється при коментарі, редагуванні або bump"
)

expiry_warned = models.BooleanField(

```

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						71
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        default=False,
        verbose_name="Попередження надіслано",
        help_text="Чи вже надіслано попередження про скорє архівування"
    )
    bump_used = models.BooleanField(
        default=False,
        verbose_name="Bump використано",
        help_text="Чи було використано продовження в поточному циклі"
    )

    def get_expiry_days(self):
        """Returns the number of days until autoarchive depending on type"""
        return self.EXPIRY_DAYS.get(self.report_type, 30)

    @property
    def animal_type_label(self):
        if self.animal_type == 'other' and self.custom_animal_type:
            return self.custom_animal_type

```

Лістинг А.2 – Перевірка антиспам-обмежень і продовження оголошення

```

def _check_spam_limits(self, user, has_donation_link):
    """
    Checks anti-spam restrictions before creating a post.
    Returns (ok: bool, error_message: str | None).
    """
    now = timezone.now()
    max_allowed = MAX_ACTIVE_REPORTS_ORG if user.is_organization else
MAX_ACTIVE_REPORTS_USER

    # Using timedelta directly
    cooldown_start = now - timedelta(minutes=COOLDOWN_MINUTES)

    recent_count = Report.objects.filter(
        user=user,
        created_at__gte=cooldown_start,
    ).count()

    if recent_count >= max_allowed:
        oldest_recent = Report.objects.filter(
            user=user,
            created_at__gte=cooldown_start,
        ).order_by('created_at').first()

        if oldest_recent:
            # Using timedelta directly
            unlock_at = oldest_recent.created_at +
timedelta(minutes=COOLDOWN_MINUTES)
            remaining_seconds = (unlock_at - now).total_seconds()
            remaining_minutes = int(remaining_seconds / 60) + 1
            return False, (
                f"Ви вже створили {max_allowed} оголошень за останні
{COOLDOWN_MINUTES} хвилин. "
                f"Зачекайте ще {remaining_minutes} хв."
            )

    return True, None

def create(self, request, *args, **kwargs):
    user = request.user
    has_donation_link = bool(request.data.get('donation_link'))

    if has_donation_link and not user.is_organization:
        return Response(
            {
                "detail": (
                    "З міркувань безпеки збір коштів доступний лише верифікованим
"
                    "організаціям. Створіть інформаційне оголошення або зверніться
до "
                    "організації для платної допомоги."
                )
            }

```

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						72
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        )
        },
        status=status.HTTP_403_FORBIDDEN,
    )

    # Apply spam and limit checks
    ok, error_msg = self._check_spam_limits(user, has_donation_link)
    if not ok:
        return Response({"detail": error_msg},
            status=status.HTTP_429_TOO_MANY_REQUESTS)

    # Use city from form data, fallback to user profile city
    user_city = request.data.get('city') or user.city or "Київ"

    images = request.FILES.getlist('uploaded_images')
    if len(images) > settings.REPORT_MAX_IMAGES:
        return Response(
            {"detail": f"До одного оголошення можна додати максимум
{settings.REPORT_MAX_IMAGES} фото."},
            status=status.HTTP_400_BAD_REQUEST,
        )

    for img in images:
        try:
            validate_uploaded_image(img, settings.UPLOADED_IMAGE_MAX_BYTES)
        except Exception:
            return Response(
                {"detail": "Один із завантажених файлів не є підтримуваним зобра-
женням."},
                status=status.HTTP_400_BAD_REQUEST,
            )

    main_image = images[0] if images else None

    serializer = self.get_serializer(data=request.data)
    serializer.is_valid(raise_exception=True)
    animal_profile = serializer.validated_data.get('animal_profile')

    if not main_image and animal_profile and animal_profile.image:
        main_image = clone_image_file(animal_profile.image,
            f"animal_{animal_profile.pk}_report")

    # Save report
    report = serializer.save(user=user, city=user_city, image=main_image,
        bump_used=False)

    if animal_profile:
        animal_profile.last_used_at = timezone.now()
        animal_profile.save(update_fields=['last_used_at'])

    # If created offline, respect the original created_at time
    created_at_override = request.data.get('created_at')
    if created_at_override:
        try:
            from django.utils.dateparse import parse_datetime
            parsed = parse_datetime(created_at_override)
            if parsed:
                if timezone.is_naive(parsed):
                    parsed = timezone.make_aware(parsed,
                        timezone.get_current_timezone())

                earliest_allowed = timezone.now() - timedelta(days=30)
                latest_allowed = timezone.now() + timedelta(minutes=5)
                if not earliest_allowed <= parsed <= latest_allowed:
                    raise ValueError("created_at override is outside the allowed
offline sync window")

                report.created_at = parsed
                report.save(update_fields=['created_at'])
        except Exception:
            pass

```

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						73
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        # Save additional images
        for img in images[1:]:
            ReportImage.objects.create(report=report, image=img)

        headers = self.get_success_headers(serializer.data)
        return Response(serializer.data, status=status.HTTP_201_CREATED,
headers=headers)

    @action(detail=True, methods=['post'],
permission_classes=[permissions.IsAuthenticated])
    def bump(self, request, pk=None):
        """The author confirms that the ad is still relevant (continues for a new
term)."""
        report = self.get_object()

        if report.user != request.user:
            return Response(
                {"detail": "Тільки автор може продовжити оголошення."},
                status=status.HTTP_403_FORBIDDEN
            )

        if report.status != 'active':
            return Response(
                {"detail": "Можна продовжити тільки активне оголошення."},
                status=status.HTTP_400_BAD_REQUEST
            )

        if report.bump_used:
            from notifications.models import Notification
            Notification.objects.filter(
                recipient=request.user,
                report=report,
                title='Оголошення скоро буде закрито'
            ).delete()
            return Response(
                {"detail": "Ви вже продовжили це оголошення. Наступне продовження
стане доступним після нового попередження."},
                status=status.HTTP_400_BAD_REQUEST
            )

        report.last_activity_at = timezone.now()
        report.expiry_warned = False
        report.bump_used = True
        report.save(update_fields=['last_activity_at', 'expiry_warned', 'bump_used'])

```

Лістинг А.3 – Геолокаційне сповіщення користувачів за підписками

```

# Checking subscriptions when creating a new ad
@receiver(post_save, sender=Report)
def check_subscriptions_for_new_report(sender, instance, created, **kwargs):
    if not created or instance.status != 'active' or not instance.location:
        return

    # We filter out the ad author and those who do not have a location
    candidates = (
        Subscription.objects
        .select_related('user')
        .exclude(user=instance.user)
        .exclude(location__isnull=True)
        .only('id', 'user_id', 'user__id', 'radius_km', 'address_label',
'animal_type', 'report_type', 'location')
    )

    # Filter by animal type
    if instance.animal_type in ['dog', 'cat']:
        candidates = candidates.filter(animal_type__in=[instance.animal_type, 'any'])
    else:
        candidates = candidates.filter(animal_type__in=['other', 'any'])

    # Filter by ad type

```

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						74
Змн.	Арк.	№ докум.	Підпис	Дата		

```

if instance.report_type in ['lost', 'found']:
    candidates = candidates.filter(report_type__in=[instance.report_type, 'any'])

# Calculation of distance within the database in meters
candidates = candidates.annotate(
    dist_meters=Distance(
        Cast('location', PointField(geography=True)),
        instance.location
    )
).filter(
    dist_meters__lte=F('radius_km') * 1000
)

notifications_to_create = []
notified_users = set()

for sub in candidates.iterator(chunk_size=500):
    if sub.user_id in notified_users:
        continue

    title = "Нове оголошення поруч!"
    if instance.report_type == 'lost':
        title = f"Загубився {instance.animal_type_label} поруч!"
    elif instance.report_type == 'found':
        title = f"Знайшовся {instance.animal_type_label} поруч!"

    message = f"В радіусі {sub.radius_km} км від '{sub.address_label}' створено
оголошення: {instance.title}"

    notif = Notification(
        recipient=sub.user,
        report=instance,
        title=title,
        message=message
    )
    notifications_to_create.append(notif)
    notified_users.add(sub.user_id)

if notifications_to_create:
    created_notifs = Notification.objects.bulk_create(notifications_to_create)
    for notif in created_notifs:
        _send_socket_message(notif)

    _send_subscription_push(instance, notified_users)

```

Лістинг А.4 – WebSocket-consumer для персональних сповіщень

```

class NotificationConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        user = self.scope.get("user")

        # Is this a registered person at all?
        if not user or user.is_anonymous:
            logger.debug("Anonymous notification websocket rejected.")
            await self.close(code=4001)
            return

        self.user_id = self.scope['url_route']['kwargs']['user_id']

        # Is this user trying to connect to channel
        if str(user.id) != str(self.user_id):
            logger.warning("User %s tried to open notification websocket for user
%s.", user.id, self.user_id)
            await self.close(code=4003)
            return

    async def disconnect(self, close_code):
        # Checking whether the group_name variable has even managed to be created
        if hasattr(self, 'group_name'):
            await self.channel_layer.group_discard(
                self.group_name,

```

					ПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        self.channel_name
    )
    logger.debug("Notification websocket disconnect: Group='%s' | Code=%s",
self.group_name, close_code)
    else:
        logger.debug("Notification websocket disconnected before group creation.
Code=%s", close_code)

    async def send_notification(self, event):
        notification = event['data']
        await self.send(text_data=json.dumps({
            'type': 'send_notification',
            'data': notification
        }))

```

Лістинг А.5 – Автоматична архівація неактивних оголошень

```

for report_type, expiry_days in Report.EXPIRY_DAYS.items():
    warning_days = expiry_days - Report.EXPIRY_WARNING_DAYS_BEFORE
    # Archiving expired posts
    archive_cutoff = now - timedelta(days=expiry_days)
    stale_reports = Report.objects.filter(
        status='active',
        report_type=report_type,
        last_activity_at__lt=archive_cutoff,
    ).select_related('user').only(
        'id', 'user_id', 'title', 'report_type', 'last_activity_at'
    )

    for report in stale_reports.iterator(chunk_size=500):
        if dry_run:
            self.stdout.write(
                f" [APXIB] ID={report.id} '{report.title}' "
                f"(тип={report_type}, остання активність:
{report.last_activity_at})"
            )
        else:
            report.status = 'archived'
            report.save(update_fields=['status'])

            Notification.objects.filter(
                recipient=report.user,
                report=report,
                notification_type='system',
                title="Оголошення скоро буде закрито",
            ).delete()

            # Notify the author
            Notification.objects.create(
                recipient=report.user,
                report=report,
                notification_type='system',
                title="Оголошення автоматично закрито",
                message=(
                    f"Ваше оголошення «{report.title}» було автоматично заар-
xivовано "
                    f"через відсутність активності протягом {expiry_days}
днів. "
                    f"Якщо потреба актуальна – створіть нове оголошення."
                ),
            )

            logger.debug(
                f"Архівовано пост ID={report.id} '{report.title}' "
                f"(тип={report_type}, без активності {expiry_days}+ днів)"
            )

            archived_total += 1

# Warning 2 days before archiving
warning_cutoff = now - timedelta(days=warning_days)

```

					ПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						76
Змн.	Арк.	№ докум.	Підпис	Дата		

```

warn_reports = Report.objects.filter(
    status='active',
    report_type=report_type,
    expiry_warned=False,
    last_activity_at__lt=warning_cutoff,
    # Do not warn those that should already be archived
    last_activity_at__gte=archive_cutoff,
).select_related('user').only(
    'id', 'user_id', 'title', 'report_type',
    'last_activity_at', 'expiry_warned', 'bump_used'
)

for report in warn_reports.iterator(chunk_size=500):
    remaining_days = Report.EXPIRY_WARNING_DAYS_BEFORE

    if dry_run:
        self.stdout.write(
            f" [ПОПЕРЕДЖЕННЯ] ID={report.id} '{report.title}' "
            f"(тип={report_type}, залишилось ~{remaining_days} днів)"
        )
    else:
        report.expiry_warned = True
        report.bump_used = False
        report.save(update_fields=['expiry_warned', 'bump_used'])

        Notification.objects.create(
            recipient=report.user,
            report=report,
            notification_type='system',
            title="Оголошення скоро буде закрито",
            message=(
                f"Ваше оголошення «{report.title}» буде автоматично заар-
                хивовано "
                f"через {remaining_days} дні через відсутність активності.
                "
                f"Натисніть «Продовжити» в оголошенні, якщо воно ще актуа-
                льне."
            ),
        ),

        logger.debug(
            f"Попередження надіслано для поста ID={report.id}
            '{report.title}'"
        )

        warned_total += 1

# Permanent deletion of old archived posts.
# Without archived_at, wait for the type expiry window plus 30 days.
# This prevents a post from being archived and deleted in the same run.
for report_type, expiry_days in Report.EXPIRY_DAYS.items():
    delete_cutoff = now - timedelta(days=expiry_days + 30)
    reports_to_delete = Report.objects.filter(
        status='archived',
        report_type=report_type,
        last_activity_at__lt=delete_cutoff
    ).prefetch_related(
        Prefetch('images', queryset=ReportImage.objects.only('id',
        'report_id', 'image'))
    ).only('id', 'title', 'last_activity_at', 'image')

```

Лістинг А.6 – Docker Compose-конфігурація сервісів платформи

```

services:
  db:
    image: postgres/postgis:15-3.3
    container_name: furry_help_db_container
    restart: always
    logging: *default-logging
    environment:
      - POSTGRES_DB=${DB_NAME}

```

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						77
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    - POSTGRES_USER=${DB_USER}
    - POSTGRES_PASSWORD=${DB_PASSWORD}
ports:
  - "127.0.0.1:5433:5432"
volumes:
  - postgres_data:/var/lib/postgresql/data
networks:
  - furry_network

redis:
  image: redis:7-alpine
  restart: always
  logging: *default-logging
  networks:
    - furry_network

web:
  build: .
  image: furryhelp-django:latest
  container_name: furry_help_web_container
  restart: always
  command: daphne -b 0.0.0.0 -p 8000 FurryHelp.asgi:application
  logging: *default-logging
  volumes:
    - ./app
    - ../furry-client/dist:/app/frontend_dist
    - ${FIREBASE_SERVICE_ACCOUNT_HOST_PATH:-
./serviceAccountKey.json}:/run/secrets/firebase-service-account.json:ro
  ports:
    - "127.0.0.1:8000:8000"
  env_file:
    - .env
  environment:
    - DB_HOST=db
    - DB_PORT=5432
    - ASGI_THREADS=${ASGI_THREADS:-8}
    - DB_CONN_MAX_AGE=${DB_CONN_MAX_AGE:-0}
    - FIREBASE_CREDENTIALS_PATH=/run/secrets/firebase-service-account.json
  depends_on:
    - db
    - redis
  networks:
    - furry_network

# Service for automatic archiving of inactive ads
cron:
  build: .
  image: furryhelp-django:latest
  container_name: furry_help_cron_container
  restart: always
  logging: *default-logging
  command: >
    sh -c "echo 'Cron-сервіс запущено. Архівация з налаштованим інтервалом.' &&
    while true; do
      python manage.py archive_stale_reports;
      sleep ${ARCHIVE_STALE_REPORTS_INTERVAL_SECONDS:-86400};
    done"
  volumes:
    - ./app
    - ${FIREBASE_SERVICE_ACCOUNT_HOST_PATH:-
./serviceAccountKey.json}:/run/secrets/firebase-service-account.json:ro
  env_file:
    - .env
  environment:
    - DB_HOST=db
    - DB_PORT=5432
    - FIREBASE_CREDENTIALS_PATH=/run/secrets/firebase-service-account.json
  depends_on:
    - db
    - redis

```

					ІПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						78
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТОК Б

API платформи побудовано за принципами REST. Основні маршрути згруповано за модулями, що спрощує супровід системи та дозволяє клієнтській частині працювати з оголошеннями, підписками, сповіщеннями, організаціями та профілем користувача через передбачувані HTTP-запити.

Таблиця Б.1

Основні API-маршрути платформи FurryHelp

Маршрут	Метод	Призначення	Роль користувача
/api/reports/	GET	Отримання списку оголошень із фільтрами	Усі користувачі
/api/reports/	POST	Створення нового оголошення з фото та координатами	Авторизований користувач
/api/reports/{id}/	GET	Перегляд детальної інформації про оголошення	Усі користувачі
/api/reports/{id}/bump/	POST	Продовження актуальності оголошення	Автор оголошення
/api/reports/{id}/complain/	POST	Надсилення скарги на оголошення	Авторизований користувач

продовж. табл. Б.1

/api/report-map/	GET	Отримання маркерів ого- лошень для карти	Усі користу- вачі
/api/subscriptions/	POST	Створення геозони підпи- ски	Авторизова- ний користу- вач
/api/my-notifications/	GET	Отримання персональних сповіщень	Авторизова- ний користу- вач
/api/organizations/	GET	Перегляд ка- талогу органі- зацій	Усі користу- вачі
/api/organization-requests/	POST	Подання зая- вки на верифі- кацію органі- зації	Авторизова- ний користу- вач
/api/users/me/	GET/PATCH	Перегляд і ре- дагування профілю	Авторизова- ний користу- вач