НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

комп'ютерних наук

_____ Голуб Б.Л., доц., к.т.н

(підпис) (ПІБ)

"___" _____2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Інформаційна система пошуку домашніх тварин»

Спеціальність 122 – «Комп'ютерні науки»

Гарант освітньої програми

Д. е. н., професор _____ Руденський Р. А.

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

к.т.н., доцент _____ Боярінова Ю.Є.

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконала _____ Олійник О.С.

(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук
_____ Голуб Б.Л., доц., к.т.н
(підпис) (ПІБ)
"___" _____2025 р.

ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи

студентці Олійник Олександрі Сергіївні

Спеціальність 122 – «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи «Інформаційна система пошуку домашніх тварин»

затверджена наказом ректора НУБіП України від "16" грудня 2024 р. № 2246 С

Термін подання завершеної роботи на кафедру 2025 . 06 . 02
рік, місяць, число

Вихідні дані до бакалаврської кваліфікаційної роботи: опис програмного забезпечення

Перелік питань, які потрібно розробити:

1. Аналіз проблемної області.
2. Вибір та обґрунтування засобів для розробки системи.
3. Проектування інформаційної системи.
4. Висновок

Дата видачі завдання "_____"_____20___р.

Керівник бакалаврської кваліфікаційної роботи_____ Боярінова Ю.Є.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Олійник О.С.
(підпис) (прізвище та ініціали студента)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання бакалаврської кваліфікаційної роботи	Строк виконання етапів бакалаврської кваліфікаційної роботи	Примітка
1	Формування мети, завдань та актуальності роботи	03.02.2025 – 09.02.2025 *(1 тиждень)*	Завершено
2	Аналіз предметної області, пошук та огляд аналогічних систем	10.02.2025 – 16.02.2025 *(1 тиждень)*	Завершено
3	Збір та аналіз інформаційних джерел, методів і технологій	17.02.2025 – 23.02.2025 *(1 тиждень)*	Завершено
4	Проєктування структури системи, розробка UML-діаграм	24.02.2025 – 02.03.2025 *(1 тиждень)*	Завершено
5	Розробка логічної та фізичної моделі бази даних	10.03.2025 – 16.03.2025 *(1 тиждень)*	Завершено
6	Розробка логічної та фізичної моделі бази даних	10.03.2025 – 16.03.2025 *(1 тиждень)*	Завершено
7	Вибір інструментів реалізації, налаштування середовища розробки	17.03.2025 – 23.03.2025 *(1 тиждень)*	Завершено
8	Реалізація архітектури додатку згідно з Clean Architecture	24.03.2025 – 06.04.2025 *(2 тижні)*	Завершено
9	Розробка функціоналу: автентифікація, база даних, Google Maps API	07.04.2025 – 20.04.2025 *(2 тижні)*	Завершено
10	Розробка інтерфейсів: екрани оголошень, фільтри, профіль	21.04.2025 – 04.05.2025 *(2 тижні)*	Завершено
11	Тестування програмного продукту: системне та приймальне тестування	05.05.2025 – 11.05.2025 *(1 тиждень)*	Завершено
12	Підготовка інсталяційного пакету	12.05.2025 – 14.05.2025 *(3 дні)*	Завершено
13	Оформлення пояснювальної записки та графічного матеріалу	15.05.2025 – 19.05.2025 *(5 днів)*	Завершено
14	Перевірка, редагування та остаточне оформлення роботи	20.05.2025 – 24.05.2025 *(5 днів)*	Завершено

АНОТАЦІЯ

до бакалаврської кваліфікаційної роботи
на тему: *«Інформаційна система пошуку домашніх тварин»*

Бакалаврська кваліфікаційна робота присвячена темі створення інформаційної системи для пошуку домашніх тварин. Темою роботи є розробка Android-додатку, який дозволяє користувачам публікувати оголошення про загублених або знайдених тварин. Об’єктом дослідження є процес пошуку та ідентифікації зниклих тварин. Предметом дослідження є інформаційна система для автоматизації цього процесу. У дослідженні використано методи проєктування програмного забезпечення, структурного аналізу даних та реалізації клієнт-серверної архітектури.

У роботі розглядаються існуючі інформаційні системи, які надають схожі послуги, та проведено порівняльний аналіз їх функціональності (Розділ 1). Описано архітектуру майбутнього програмного продукту, включаючи структурні, класові, послідовні діаграми та діаграми варіантів використання (Розділ 2). Розроблено фізичну структуру бази даних на основі потреб функціоналу додатку. Проведено реалізацію основних функцій у середовищі Android Studio із застосуванням Firebase для автентифікації, зберігання даних та інтеграції з Google Maps API (Розділ 3).

Результатом проведеного дослідження стало створення мобільного додатку з сучасним та зручним інтерфейсом, який дає змогу ефективно публікувати та переглядати оголошення про загублених або знайдених домашніх тварин, відфільтровувати результати за типом, породою, кольором та місцем, а також використовувати інтерактивну карту для визначення розташування.

ABSTRACT

of the Bachelor's Qualification Work
on the topic: "Information System for Pet Search

Bachelor's qualification work is devoted to the topic of creating an information system for searching for pets. The topic of the work is the development of an Android application that allows users to publish ads about lost or found animals. The object of the research is the process of searching for and identifying missing animals. The subject of the research is an information system for automating this process. The research uses methods of software design, structural data analysis and implementation of client-server architecture.

The work considers existing information systems that provide similar services and conducts a comparative analysis of their functionality (section 1). The architecture of the future software product is described, including structural, class, sequence diagrams and use case diagrams (section 2). The physical structure of the database is developed based on the needs of the application's functionality. The main functions are implemented in the Android Studio environment using Firebase for authentication, data storage and integration with the Google Maps API (section 3).

The result of the research was the creation of a mobile application with a modern and user-friendly interface that allows you to effectively publish and view ads about lost or found pets, filter results by type, breed, color, and location, and use an interactive map to determine the location

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 9

ВСТУП 10

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 13

1.1 Постановка завдання 13

1.2 Огляд інформаційних джерел та існуючих рішень 15

1.2.1 OLX (розділ «Тварини»). 16

1.2.2 Pet911 16

1.2.3 ZooPoshuk 17

1.3 Моделювання предметної області 18

1.3.1 Діаграма прецедентів 20

1.3.2 Діаграма діяльності 23

1.3.3 Діаграма послідовності 26

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ 30

2.1 Логічна модель даних 30

2.2 Побудова ER-діаграми 33

2.3 Вибір системи управління інформаційною базою 35

2.4 Створення інформаційної бази 38

2.4.1 Фізична реалізація бази даних 40

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ 44

3.1 Організаційна структура програмного забезпечення 44

3.2 Вибір інструментарію для створення ППЗ 47

3.3 Алгоритмізація та програмування програмних модулів 51

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ВИКОРИСТАННЯ СИСТЕМ 58
4.1 Тестування системи 58
4.1.1 Системне тестування 59
4.1.2 Приймальне тестування 61
4.2 Вимоги до апаратного та програмного забезпечення 63
4.2.1 Діаграма розміщення 63
4.2.2 Вимоги до апаратного забезпечення 64
4.2.3 Вимоги до програмного забезпечення 65
4.3 Склад інсталяційного пакету 66
ВИСНОВКИ 70
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 72
ДОДАТОК А ..... **Error! Bookmark not defined.**
ДОДАТОК Б 73

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

ППЗ – прикладне програмне забезпечення

СУБД – система управління базами даних

APK – Android Package Kit

CASE – Computer-Aided Software Engineering

IDE – Integrated Development Environment

ER – Entity-Relationship

MS – Microsoft

SDK – Software Development Kit

SQL – Structured Query Language

RDBMS – Relational Database Management System

UML – Unified Modeling Language

ВСТУП

У наш час, коли рівень урбанізації постійно зростає, проблема втрати домашніх тварин набуває все більшого поширення та соціального значення. Щороку тисячі власників стикаються з ситуаціями, коли їхні улюбленці губляться з різних причин: втеча через відчинені двері, страх від гучних звуків, неуважність під час прогулянки або неналежне маркування тварини. У результаті таких випадків власники переживають емоційний стрес, витрачають багато часу та зусиль на пошук тварини, а іноді – безрезультатно. Зазвичай пошук здійснюється за допомогою оголошень у соціальних мережах або паперових оголошень, що розклеюються на вулицях, однак ці методи є неструктурованими, неефективними і не гарантують швидкого результату.

Тоді як розвиток інформаційних технологій відкриває нові можливості для вирішення цієї проблеми. Автоматизація процесу пошуку тварин за допомогою сучасного програмного забезпечення дозволяє зменшити часові та інформаційні витрати, покращити комунікацію між людьми, які шукають своїх улюбленців, і тими, хто їх знаходить. Тому розробка інформаційної системи для автоматизації пошуку домашніх тварин є актуальним завданням, яке має реальне соціальне значення.

Така система повинна виконувати кілька важливих функцій: реєстрація користувачів, створення і перегляд оголошень, можливість фільтрації за критеріями (тип тварини, місце, породу, колір, відстань), швидкий зв’язок між користувачами. Крім того, важливо забезпечити зберігання інформації у зручній та захищеній формі, що дозволить не лише швидко знаходити потрібні записи, але й гарантує їхню безпеку та цілісність. Ураховуючи поширеність мобільних пристроїв, доцільно реалізувати функціональність у вигляді мобільного застосунку з інтуїтивно зрозумілим інтерфейсом.

Отже, мета бакалаврської кваліфікаційної роботи полягає в розробці інформаційної системи для пошуку домашніх тварин, яка дозволяє швидко публікувати, переглядати та фільтрувати оголошення про зникнення або знаходження тварин, а також швидко зв'язуватись між користувачами. Такий

застосунок покликаний мінімізувати втрати часу на пошук та підвищити ймовірність повернення тварини її власнику.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз предметної області, зокрема причин втрати тварин та поширених методів їхнього пошуку;
- проаналізувати існуючі програмні рішення аналогічного призначення, виявити їх функціональні можливості та обмеження;
- сформулювати функціональні та нефункціональні вимоги до майбутньої інформаційної системи;
- побудувати UML-діаграми для моделювання структури та логіки роботи системи;
- спроєктувати структуру бази даних та реалізувати її за допомогою хмарного сервісу Firebase;
- розробити клієнтську частину мобільного застосунку на платформі Android з використанням фреймворку Flutter;
- реалізувати серверну логіку та взаємодію з базою даних засобами Firebase (Cloud Functions та Firestore);
- здійснити тестування основних функціональних модулів системи;
- підготувати супровідну документацію для користувачів та описати процес встановлення і запуску програмного продукту.

Система має бути інтуїтивно зрозумілою, безпечною у використанні, адаптованою до мобільних пристроїв і розрахованою на широку аудиторію користувачів із різним рівнем цифрової грамотності.

Одним із ключових елементів розробки є побудова правильної структури бази даних, що забезпечить ефективне зберігання, пошук та оновлення оголошень у хмарному середовищі Firebase. Саме моделювання даних та інтеграція з Firestore є важливою технічною складовою розробки.

Пояснювальна записка містить 62 сторінок основного тексту, 11 використаних джерел, 1 додаток. Вона складається зі вступу, чотирьох основних розділів, висновків, списку використаних джерел та додатків. Зміст пояснювальної записки є логічно структурованим, охоплює повний цикл розробки інформаційної системи – від постановки проблеми до реалізації та впровадження.

Таким чином, результатом виконаної роботи є функціональний мобільний застосунок, що забезпечує ефективний інструмент для пошуку домашніх тварин, оптимізуючи взаємодію між користувачами, підвищуючи точність і швидкість обміну інформацією, а також зменшуючи ризики втрати тварин у сучасних урбанізованих умовах.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання

У межах даної кваліфікаційної роботи було прийнято рішення розробляти саме мобільний застосунок під операційну систему Android. Це зумовлено тим, що Android є найбільш поширеною мобільною платформою в Україні та у світі, що забезпечує максимальне охоплення цільової аудиторії.

Розробка мобільного застосунку для пошуку загублених або знайдених домашніх тварин вимагає створення інформаційної системи, яка забезпечуватиме ефективне збирання, зберігання, обробку, фільтрацію та представлення даних про такі випадки. Такий застосунок має слугувати посередником між людьми, які шукають своїх домашніх тварин, та тими, хто їх знайшов.

Основні цілі системи:

- надати користувачам простий спосіб публікації оголошень про зникнення або знаходження тварини;
- дозволити фільтрацію оголошень за ключовими ознаками (тип тварини, порода, колір, відстань);
- забезпечити зручний інтерфейс для перегляду та пошуку оголошень;
- створити зручну систему обліку особистих оголошень користувача;
- підвищити ймовірність повернення тварини її власнику

Програма повинна включати в себе набір функцій, які необхідні для реалізації основного процесу взаємодії між користувачами. Працюючи з програмою, користувач зможе:

- зареєструвати акаунт, увійти або відновити пароль (через email);
- створити оголошення про зниклу або знайдену тварину;
- переглядати оголошення, використовуючи фільтри;
- переглядати детальну інформацію про кожну тварину (фото, порода, колір, адреса події, опис, контактна особа);

- змінювати дані власного профілю та пароль;
- переглядати, редагувати та видаляти власні оголошення.

Основні проблеми, які має вирішити система:

1. Розпорошеність інформації. Оголошення про зниклих або знайдених тварин розміщуються в різних соціальних мережах чи у вигляді паперових листівок, що ускладнює швидкий пошук та зменшує шанси на повернення улюбленця власнику.

2. Відсутність зручного пошуку та фільтрації. Неможливість фільтрації оголошень за важливими параметрами змушує користувача переглядати велику кількість нерелевантної інформації.

3. Неефективна комунікація між сторонами. Власники та люди, які знайшли тварину, не мають безпечного й простого каналу зв’язку, що призводить до затримок, втрати контактів і, зрештою, знижує ймовірність успішного повернення тварини.

Звітність, яку формує система:

- можливість завантаження всіх особистих оголошень у вигляді таблиці (форматn MS Excel) для подальшого збереження, аналізу або експорту;
- відображення детальної інформації про кожне оголошення, включаючи дату створення, характеристики тварини, місце події та контактні дані.

Класифікація системи за критеріями:

1. За характером використання інформації – інформаційно-пошукова система, яка орієнтована на обробку, пошук та представлення структурованої інформації кінцевим користувачам.;

2. За масштабом – локальна, яка зосереджується на місті Київ та Київській області, однак має потенціал до масштабування.

3. За архітектурою мобільна клієнт – серверна система, побудована на базі сучасних хмарних технологій (Firebase, Google Maps API), що відповідає сучасним стандартам безпеки, продуктивності й комунікацій;

4. За ступенем автоматизації – напівавтоматизована, бо користувач самостійно вводить інформацію, але її обробка, фільтрація, зберігання й виведення на екран здійснюються автоматично.

5. За способом взаємодії – мобільний застосунок, зручний для широкого кола користувачів.

Розроблювана система повинна відповідати сучасним вимогам до мобільних інформаційних продуктів, бути функціональною, безпечною та зручною для користувача. Основні вимоги до системи:

- зручний та інтуїтивно зрозумілий інтерфейс для користувачів з різним рівнем цифрової грамотності.
- адаптація під мобільні пристрої з операційною системою Android.
- надійна система автентифікації та захисту персональних даних.
- можливість створення, редагування, фільтрації та перегляду оголошень про зниклих і знайдених тварин.
- підтримка геолокації та інтеграція з картою для вказування адреси події.
- стабільна робота з хмарною базою даних Firebase для зберігання та обробки інформації.

Такий застосунок має реальну соціальну значущість і сприяє вирішенню важливої проблеми повернення зниклих домашніх тварин їхнім власникам. Система поєднує функціональність, простоту та ефективність, що робить її корисним інструментом у повсякденному житті.

1.2 Огляд інформаційних джерел та існуючих рішень

Перш ніж розпочати розробку мобільного застосунку для пошуку домашніх тварин, важливо проаналізувати наявні інформаційні джерела, що стосуються обраної предметної області, а також здійснити огляд існуючих програмних рішень, які частково або повністю реалізують подібну функціональність. Це дозволяє не лише глибше зрозуміти вимоги до майбутньої

системи, а й виявити сильні та слабкі сторони конкурентних застосунків, щоб у подальшому врахувати їх при проєктуванні власного продукту.

Також було проведено аудит кількох популярних сервісів, які частково реалізують функціональність, подібну до тієї, що передбачена в даній роботі. Аналіз базувався на визначенні переваг і недоліків кожного рішення з метою врахування їх у власному проєкті.

1.2.1 OLX (розділ «Тварини»)

Частина онлайн-платформи OLX, яка призначена для розміщення оголошень, пов’язаних з тваринами. Тут користувачі можуть продавати або купувати домашніх улюбленців, пропонувати або шукати товари та послуги для них, а також публікувати інформацію про зниклих або знайдених тварин. Розділ охоплює широкий спектр оголошень і дозволяє зручно шукати потрібну інформацію за видом тварини, породою, ціною чи місцем розташування.

Переваги:

- велика кількість активних користувачів;
- проста та знайома більшості користувачів форма подання оголошення;
- підтримка як веб-, так і мобільної версії платформи;
- можливість додавання фотографій, контактної інформації та опису.

Недоліки:

- відсутність спеціалізації на пошуку зниклих тварин;
- немає прив’язки оголошень до карти або GPS-навігації;
- фільтрація за параметрами тварини є обмеженою;
- відсутня система особистих кабінетів із керуванням власними оголошеннями.

1.2.2 Pet911

Всеукраїнська онлайн-платформа, створена для допомоги у пошуку зниклих домашніх тварин. Користувачі можуть безкоштовно розміщувати оголошення про зникнення або знаходження тварин, а також використовувати додаткові сервіси для пришвидшення пошуку.

Переваги:

- орієнтація саме на тему загублених і знайдених домашніх тварин;
- можливість фільтрації оголошень за містом, типом тварини та статусом;
- публічна база даних із вільним доступом до оголошень.

Недоліки:

- відсутній мобільний застосунок, що обмежує зручність доступу;
- застарілий інтерфейс, який ускладнює взаємодію з сайтом;
- обмежена функціональність у частині комунікації між користувачами.

1.2.3 ZooPoshuk

Безкоштовна українська платформа для пошуку зниклих і знайдених тварин. Користувачі можуть швидко розміщувати оголошення, які поширюються через соцмережі, листівки та сповіщення волонтерам. Сервіс працює по всій Україні з 2021 року.

Переваги:

- зручний пошук загублених/знайдених тварин;
- фільтри за породою, кольором, типом тварини;
- інтеграція з Google Maps для точного визначення місця;
- простий інтерфейс користувача;
- можливість додавання фото та контактів.

Недоліки

- обмеження по місту (лише Київська область);
- потребує стабільного інтернет-з’єднання;
- відсутність підтримки інших мов, крім української;
- залежність від точності даних користувачів;
- можливі проблеми з модерацією оголошень;

На основі аналізу можна зробити висновок, що існуючі сервіси частково покривають запити користувачів, однак жоден з них не надає універсального мобільного рішення, яке б одночасно:

- було доступним у вигляді застосунку для Android;
- дозволяло легко публікувати та фільтрувати оголошення;

- мало зручний особистий профіль;
- працювало у реальному часі на базі хмарних технологій.

Саме ці недоліки й стали підґрунтям для розробки нового мобільного застосунку, що поєднує кращі функціональні ідеї з актуальних рішень та усуває їхні слабкі сторони за рахунок сучасної реалізації на платформі Android з використанням Firebase.

1.3 Моделювання предметної області

Моделювання предметної області – це один з найважливіших етапів у проєктуванні інформаційної системи. Саме тут формуються логічні уявлення про структуру майбутнього застосунку, визначаються ключові об’єкти, сценарії використання, а також взаємозв’язки між елементами і поведінка системи. Якість створених моделей безпосередньо впливає на точність реалізації функціоналу, ефективність архітектурних рішень і зручність користування програмним продуктом.

У сучасній практиці розробки програмного забезпечення активно використовуються CASE-засоби (Computer-Aided Software Engineering), що забезпечують візуальне представлення архітектури системи у вигляді моделей. Такі інструменти дають змогу розробникам, аналітикам і замовникам спільно працювати над проєктом, погоджуючи всі важливі етапи ще до написання коду.

Серед найпоширеніших CASE-засобів для графічного моделювання можна виділити StarUML, Draw.io, Visual Paradigm, Lucidchart, Enterprise Architect, а також Microsoft Visio.

Одним з основних стандартів побудови моделей програмних систем є Unified Modeling Language (UML) – уніфікована мова моделювання. Розшифруємо: *modeling* передбачає створення моделі, що описує об'єкт. *Unified* (універсальний, єдиний) – підходить для широкого класу проектованих програмних систем, різних областей додатків, типів організацій, рівнів компетентності, розмірів проектів. UML описує об'єкт в єдиному заданому

синтаксисі, тому де б ви не намалювали діаграму, її правила будуть зрозумілими для всіх, хто знайомий з цією графічною мовою – навіть в іншій країні.

- Побудова моделей дозволяє:
- скоротити час на пошук помилок у логіці роботи системи;
- побачити взаємозв’язки між компонентами ще до написання коду;
- структурувати функціональні вимоги;
- забезпечити цілісне бачення проєкту на ранніх етапах розробки.

Під час створення моделей слід дотримуватись принципу абстрагування – включати лише ті аспекти системи, які мають практичне значення для її функціонування. Необов’язкові або другорядні деталі можуть ускладнити аналіз і сприйняття моделей.

Іншим важливим принципом є багатомодельність, що означає необхідність створення декількох різних діаграм для відображення різних аспектів поведінки або структури системи. Жодна окрема діаграма не може повністю описати складну програмну систему, тому для повноти картини доцільно використовувати кілька типів моделей.

UML забезпечує стандартну нотацію для багатьох типів діаграм, які можна умовно розділити на 3 основні групи: діаграми поведінки, діаграми взаємодії та структурні діаграми.

Діаграми UML класифікуються на:

• структурні – містять діаграми класів (відображає структуру класів у системі та їхні взаємозв’язки), компонентів (дає змогу уявити структуру високорівневих компонентів системи та їхні залежності) і об’єктів (фокусується на конкретних об’єктах у системі та їхніх взаємодіях). Призначені для візуалізації статичної структури системи, тобто її складових елементів та їхніх взаємозв’язків.

• поведінкові – включають діаграми випадків використання (описує взаємодію між системою та її користувачем), активностей (моделює потоки управління або процесів у системі).Поведінкові діаграми UML призначені для

опису динамічної поведінки системи, тобто її взаємодії з навколишнім середовищем і внутрішніх процесів.

- діаграми взаємодії – охоплюють діаграми послідовності (показує взаємодію між різними об’єктами в системі в певній послідовності подій), комунікації (зображує зв’язки та повідомлення між об’єктами), а також взаємодії загального вигляду (поєднує елементи активностей і послідовностей). Призначені для моделювання обміну повідомленнями та координації дій між об'єктами.

Загальна кількість UML-діаграм не є критично важливою, адже для більшості проєктів цілком достатньо зосередитися на тих діаграмах, які безпосередньо відображають ключові аспекти системи. Тому ми розглянемо лише ті діаграми, які справді потрібні для розробки цього проєкту.

1.3.1 Діаграма прецедентів

Одним із найважливіших інструментів для аналізу вимог до програмного забезпечення є діаграма прецедентів (англ. Use Case Diagram). Вона допомагає візуалізувати функціональні можливості системи з точки зору користувача. Завдяки цій діаграмі можна чітко визначити межі системи, а також зрозуміти, які дії можуть виконувати користувачі і як система реагує на ці дії.

Цей тип діаграми належить до поведінкових діаграм UML і використовується переважно на ранніх етапах проєктування, коли необхідно зібрати, систематизувати та представити функціональні вимоги у зручній візуальній формі. Основною перевагою діаграми прецедентів є її простота й зрозумілість – навіть для тих, хто не має технічної підготовки.

Основні елементи діаграми прецедентів:

1. Актори (Actors) – це зовнішні суб’єкти, які взаємодіють із системою. Це можуть бути люди (користувачі, адміністратори), інші системи або зовнішні пристрої. В UML актори зображуються у вигляді стилізованих «людинок». Актори ініціюють певні дії (прецеденти), отримують результати роботи системи або беруть участь у процесах всередині системи.

2. Прецеденти (Use Cases) – це конкретні функції або сервіси, які система надає акторам. Кожен прецедент описує послідовність дій, які система виконує для досягнення певної мети користувача. Прецеденти зображуються у вигляді овалів, всередині яких прописується їх назва. Прецеденти мають бути логічно завершеними функціями або процесами.

3. Зв'язки (Associations) – це лінії, що з'єднують акторів із прецедентами, показуючи, хто і які дії виконує. Існують різні типи зв'язків:

- асоціація – базовий зв'язок між актором і прецедентом, що означає взаємодію.
- include (включення) – показує, що один прецедент обов'язково включає виконання іншого. Це дозволяє уникнути дублювання спільної поведінки в кількох прецедентах.
- extend (розширення) – позначає необов'язкове додаткове поведінкове розширення прецедента, яке виконується за певних умов.
- generalization (узагальнення) – використовується для наслідування між прецедентами або акторами, коли одна роль чи функція є спеціалізацією іншої.

Процес побудови діаграми прецедентів починається з визначення акторів. На цьому етапі детально уточнюють ролі кожного актора та функції, які він виконуватиме у рамках системи. У межах розробки інформаційної системи були визначені два основні актори:

- користувач – особа, яка використовує систему для створення, перегляду та пошуку оголошень про зниклих або знайдених тварин.
- модератор – це адміністратор, який здійснює перевірку змісту оголошень перед їх публікацією.

Наступним кроком є формування прецедентів, тобто визначення всіх можливих функцій або випадків використання, які система має реалізувати для задоволення потреб цих акторів.

Для користувача передбачено такі функціональні можливості, як:

- пошук оголошень, з можливістю вибору фільтрів;

- перегляд оголошення, що включає перегляд контактної інформації власника;
- створення оголошення з введенням параметрів, описом тварини, фотографією;
- управління оголошеннями (редагування, видалення).

Модератор, у свою чергу, взаємодіє з прецедентами, що відповідають за перевірку та модерацію вмісту: контроль за змістом оголошення, підтвердження оголошення або відхилення оголошення. Це дозволяє забезпечити якість інформації в системі та запобігти публікації некоректних чи фейкових даних.

Далі, на основі ретельного аналізу вимог, встановлюють зв’язки між акторами та прецедентами, що показує, хто саме та якими функціями користується. Наприклад, користувач ініціює пошук оголошень, який за потреби може розширюватися вибором фільтрів (extend). Перегляд оголошення обов’язково включає перегляд контактної інформації власника (include). Аналогічно, створення оголошення включає завантаження фотографії та введення опису й параметрів тварини Це дає змогу побудувати цілісну картину функціональної взаємодії.

Таким чином, діаграма прецедентів (рис. 1.1) чітко відображає взаємодію користувачів із системою пошуку домашніх тварин, дозволяючи чітко окреслити функціональні вимоги до майбутнього програмного забезпечення.

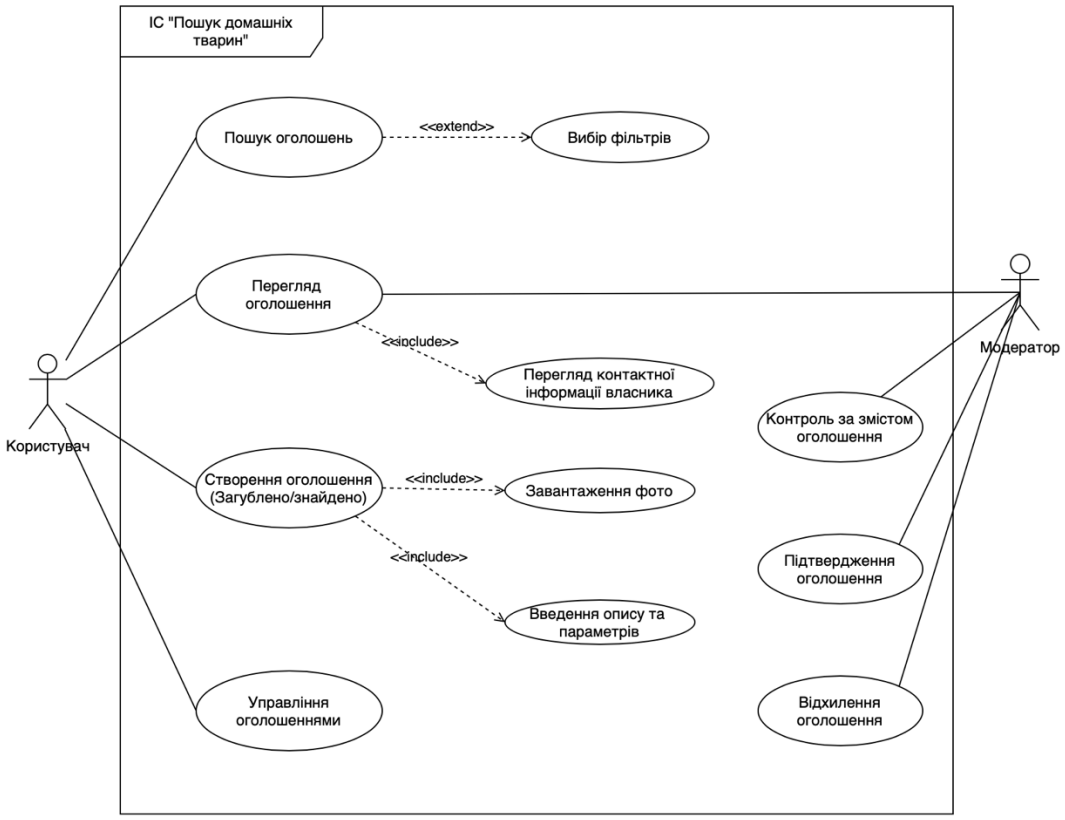

Рис. 1.1 Діаграма прецедентів

1.3.2 Діаграма діяльності

Діаграма діяльності (англ. Activity Diagram) є одним із видів поведінкових діаграм у нотації UML. Вона використовується для моделювання алгоритмів, послідовностей дій або бізнес-процесів у системі. Ця діаграма ілюструє, як керування переходить від однієї дії до іншої, підкреслюючи логіку виконання процесів. Така діаграма добре підходить для показу сценаріїв використання, умовних переходів, циклів і паралельних гілок.

Головною метою побудови діаграми діяльності є графічне представлення потоку управління або потоку даних, які реалізуються у межах конкретного процесу. Такі діаграми дозволяють розробнику або аналітику побачити загальну картину функціонування певної частини системи, визначити, які дії виконуються, в якій послідовності, які можливі альтернативні сценарії, як відбувається обробка умов та розгалуження логіки.

Для повного розуміння суті діаграми діяльності потрібно розглянути її основні складові, які використовуються для побудови:

1. Початковий вузол (Initial Node) – позначає початок процесу, зазвичай зображується як чорне коло. Його наявність є обов’язковою умовою для діаграми, адже саме з нього починається весь потік виконання.

2. Дія або активність (Action / Activity) – конкретна операція, яка виконується у межах процесу, позначається прямокутником зі скругленими кутами. Це можуть бути як елементарні дії (наприклад, "зберегти дані"), так і складні підпроцеси.

3. Потік керування (Control Flow) – напрямок руху потоку управління. Візуально представлений стрілкою, що з’єднує дві дії або вузли.

4. Умовний вузол (Decision Node) – елемент, що дозволяє реалізувати розгалуження логіки залежно від певних умов. Його зображують як ромб з кількома вихідними стрілками, кожна з яких має відповідну умову.

5. Злиття (Merge Node) – елемент, що дозволяє об'єднати кілька альтернативних гілок у спільний потік.

6. Розгалуження (Fork Node) – використовується для створення паралельного виконання кількох дій.

7. Об’єднання (Join Node) – синхронізує паралельні гілки та забезпечує продовження процесу після завершення усіх паралельних дій.

8. Кінцевий вузол (Final Node) – вказує на завершення виконання процесу. Візуально це чорне коло з білим обідком.

У межах даного проєкту було розроблено діаграму діяльності (рис. **1.**2), яка моделює поведінку користувача під час перегляду оголошень у системі пошуку домашніх тварин.

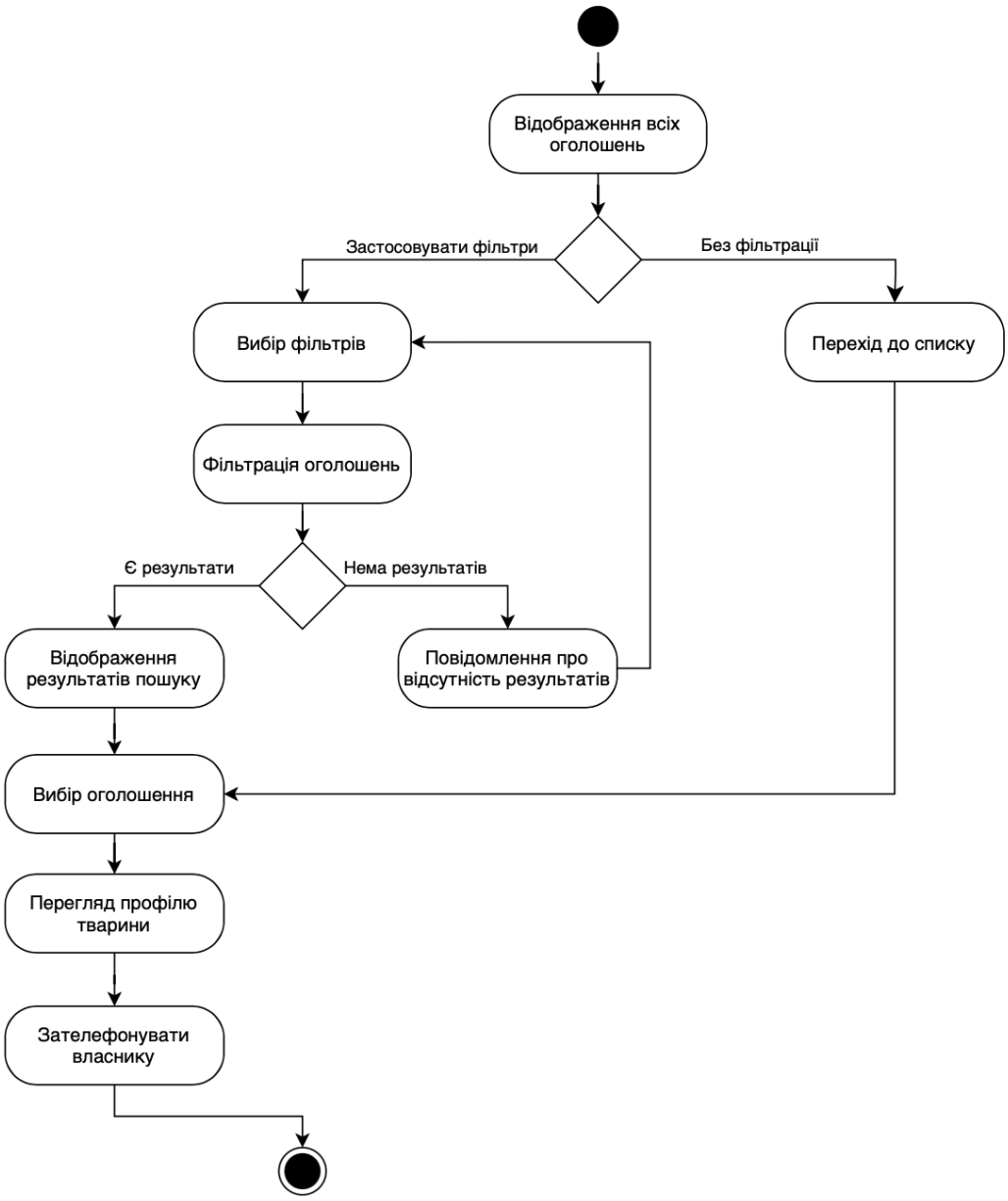

Рис. 1.2 Діаграма діяльності (Пошук загубленої тварини)

Особливістю реалізації є те, що всі оголошення відображаються одразу після запуску відповідного екрану, а фільтри використовуються для подальшого уточнення результатів. Тобто система не здійснює повноцінний пошук за запитом, а лише застосовує обрані параметри фільтрації до наявного списку. Користувач має змогу одразу ознайомитися з усіма публікаціями, що вже розміщені в системі.

Далі користувач може прийняти рішення – чи потрібно застосувати фільтри для звуження результатів. У разі потреби відбувається вибір параметрів фільтрації (тип тварини, розмір, колір, відстань), після чого система фільтрує список оголошень відповідно до обраних критеріїв.

Якщо результати відсутні, користувач отримує повідомлення про відсутність результатів та має можливість змінити фільтри, повернувшись до початкового кроку.

Якщо ж результати знайдені, відбувається відображення результатів пошуку. Далі користувач може вибрати конкретне оголошення, після чого відкривається профіль тварини з детальною інформацією: фото, опис, параметри та контактні дані. Останнім кроком у цьому процесі є дзвінок власнику – функція, яка дозволяє напряму зв’язатися з особою, що створила оголошення.

Ця діаграма діяльності точно відображає логіку взаємодії з оголошеннями у системі, де відсутній класичний пошук, але реалізована зручна фільтрація.

1.3.3 Діаграма послідовності

Діаграма послідовності (Sequence Diagram) – ще один із ключових інструментів мов моделювання UML (Unified Modeling Language), що використовується для графічного зображення послідовності взаємодій між об'єктами в рамках певного сценарію.

Діаграма послідовності (Sequence Diagram) – ще один із ключових інструментів мов моделювання UML, що використовується для графічного зображення послідовності взаємодій між об'єктами в рамках певного сценарію. Вона належить до діаграм поведінки, оскільки моделює динамічну поведінку системи у часі. Діаграма дозволяє описати, як об'єкти системи співпрацюють у часовій послідовності для досягнення визначеної мети або виконання конкретного функціонального процесу. Такі діаграми особливо важливі на етапах аналізу вимог, проєктування програмного забезпечення та взаємодії між компонентами системи.

Основною метою діаграми послідовності є моделювання часового порядку повідомлень, що передаються між учасниками системи. Кожне повідомлення на діаграмі представляє виклик методу або запит, який ініціює один об'єкт і обробляє інший. Таким чином, діаграма дозволяє чітко побачити, які саме об’єкти беруть участь у процесі, в якій послідовності відбувається обмін

інформацією між ними, хто ініціює дії, та які залежності існують між різними елементами системи.

На діаграмі послідовності по горизонталі розміщуються учасники взаємодії: це можуть бути актори (зовнішні користувачі чи системи), компоненти програми (інтерфейс, контролери, база даних тощо), а також зовнішні ресурси. Вертикально для кожного з них будується життєва лінія (lifeline) — пунктирна вертикальна лінія, яка показує період активності цього об’єкта у часовій шкалі. Взаємодії між об'єктами представляються стрілками, які символізують повідомлення: виклики функцій, запити, відповіді на запити тощо.

Типи повідомлень можуть бути різними. Найбільш поширеним є синхронний виклик — це повідомлення, при якому об'єкт, що ініціює виклик, чекає на результат до завершення операції. Такі повідомлення позначаються суцільною стрілкою з повною головкою.

Іншим видом є асинхронне повідомлення — коли об’єкт надсилає запит і не чекає на відповідь, а продовжує роботу паралельно. Асинхронні повідомлення зображаються стрілками з відкритою головкою. Результати обробки, як правило, повертаються у вигляді штрихованих стрілок, що позначають відповідь або результат дії.

На діаграмі також часто використовуються активаційні смужки — вертикальні прямокутники на життєвій лінії, які вказують на період часу, протягом якого об’єкт виконує операцію або є активним. Це дозволяє краще візуалізувати розподіл навантаження між компонентами та часові рамки їх активності.

У складніших сценаріях використовуються конструкції умовного виконання, циклів або альтернатив. Наприклад, умовні блоки (alt) дають змогу показати розгалуження сценарію залежно від певної умови. Аналогічно, блоки "loop" застосовуються для позначення повторюваних дій, таких як обробка списку елементів або багаторазове надсилання запитів.

Застосування діаграм послідовності є корисним у таких ситуаціях:

- при аналізі вимог до системи – допомагає зрозуміти, як користувач взаємодіє із системою.
- для документування сценаріїв використання (use cases) – особливо в рамках складних функцій із кількома учасниками.
- у процесі розробки архітектури – для візуалізації логіки взаємодії між підсистемами.
- під час комунікації між розробниками та аналітиками – як засіб узгодження бачення роботи системи.
- для тестування – тестувальники можуть використовувати діаграми для побудови тест-кейсів, що відповідають послідовності подій.

Діаграма послідовності є не лише технічним інструментом, а й засобом комунікації між усіма учасниками розробки. Вона дає змогу уникнути непорозумінь, підвищити якість проєктування та забезпечити точність реалізації логіки системи відповідно до вимог замовника.

На прикладі наведеної діаграми послідовності (рис. 1.3) зображено сценарій створення оголошення про загублену або знайдену тварину в мобільному застосунку. Учасниками процесу є користувач, модератор та об'єкт «Оголошення».

Спочатку користувач ініціює створення оголошення, після чого система надсилає запит на введення даних. Користувач заповнює необхідні деталі, такі як фото, опис та локація. Після заповнення інформації, оголошення надсилається модератору на перевірку.

Модератор проводить перевірку контенту і, залежно від результату, приймає одне з двох рішень: підтвердити публікацію або відхилити її з вказанням причини. Цей процес реалізовано за допомогою умовного блоку (alt), що дозволяє показати альтернативні сценарії розвитку подій: [Оголошення підтверджено] — оголошення публікується, або [Оголошення відхилено] — користувач отримує повідомлення з причиною відмови.

Дана діаграма демонструє, як у реальному часі відбувається взаємодія між користувачем і модератором, ілюструючи прозорий процес модерації для забезпечення достовірності публікацій.

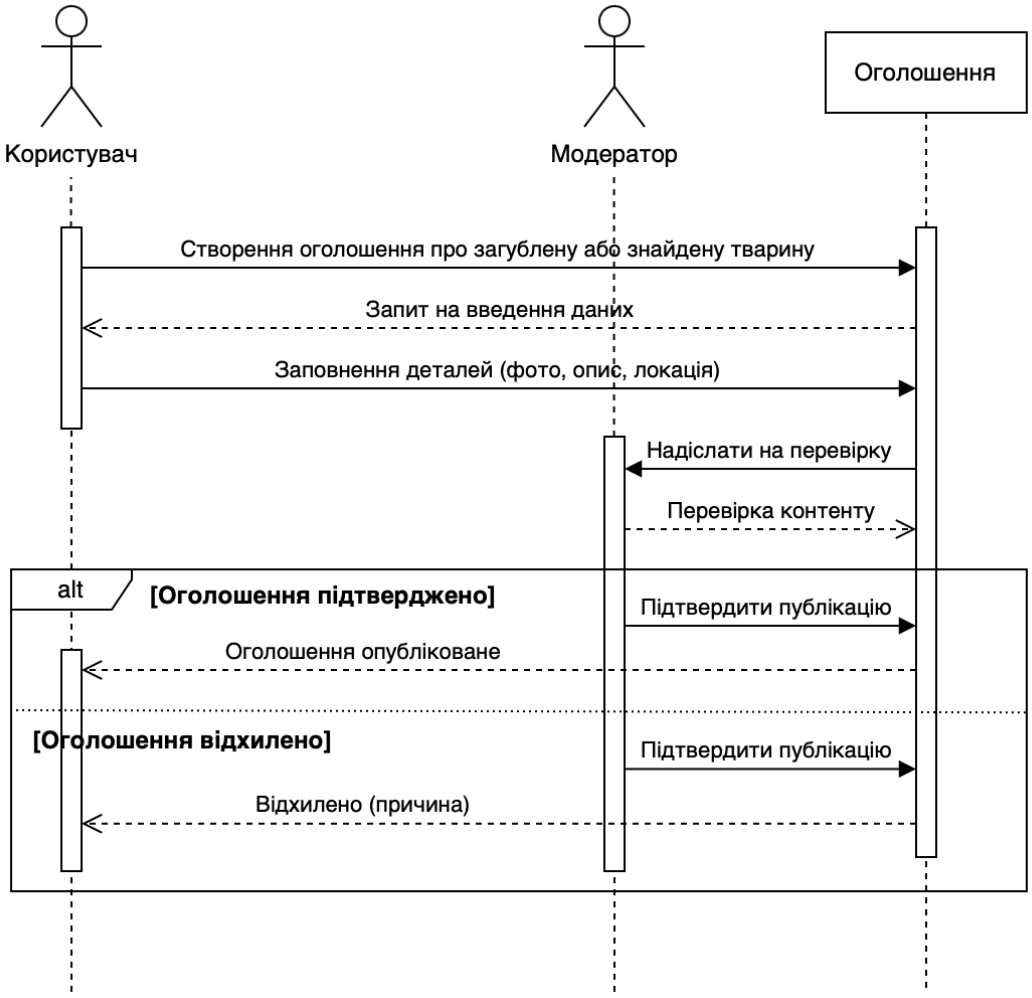

Рис. 1.3 Діаграма послідовності

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

Інформаційне забезпечення є однією з ключових складових будь-якої автоматизованої інформаційної системи. Воно включає в себе єдину систему класифікації, уніфіковану систему документації та інформаційну базу. Основною функцією інформаційного забезпечення є підтримка процесів збору, зберігання, обробки та надання інформації, необхідної для функціонування системи.

Інформаційна база – це організована сукупність даних, які зберігаються в пам’яті обчислювальної системи та підлягають систематизованій обробці за допомогою відповідного програмного забезпечення. У процесі проектування інформаційного забезпечення особливу увагу приділяють формуванню інформаційної бази, яка будується на основі логічної моделі даних. Логічна модель дозволяє абстрагуватися від конкретних особливостей реалізації в СУБД і сконцентруватися на структурі даних, їх взаємозв’язках, залежностях та обмеженнях цілісності.

Процес формування логічної моделі передбачає дотримання вимог реляційної моделі даних, зокрема нормалізації до третьої нормальної форми. Це дозволяє уникнути надлишковості, а також підвищує цілісність і несуперечність даних. На підставі логічної моделі будується ER-діаграма, яка візуалізує основні сутності системи, їх атрибути та взаємозв’язки між ними.

2.1 Логічна модель даних

Логічна модель даних — це модель даних, для побудови інформаційної бази будь-якої інформаційної системи. Вона слугує абстрактним представленням структури даних, що дозволяє формалізовано описати сутності предметної області, їх характеристики та взаємозв’язки без прив’язки до конкретної реалізації в СУБД. Такий підхід дозволяє зосередитися на аналізі логічної структури інформації, яку буде опрацьовувати система, та закладає основи для подальшого проектування та реалізації фізичної бази даних.

При побудові логічної моделі важливо враховувати специфіку функціонування системи, її основні завдання, потреби користувачів, а також обсяг та типи даних, які обробляються.

Для інформаційної системи пошуку домашніх тварин логічна модель базується на ключових об’єктах предметної області, важливих для реалізації основних функцій – створення, збереження та перегляду оголошень про зниклих або знайдених тварин. У результаті проведеного аналізу було виокремлено п’ять основних сутностей, що утворюють структуру логічної моделі:

1. Користувач (User) – є центральною сутністю системи, оскільки саме користувачі створюють оголошення, шукають або знаходять тварин. Кожен користувач зареєстрований через Firebase Authentication і отримує унікальний ідентифікатор (UID), який використовується як ключ для доступу до його персональних даних, що зберігаються у Firebase Firestore. Зберігається така інформація, як ім’я, прізвище, номер телефону та пошта, що дозволяє здійснювати ідентифікацію особи та забезпечувати ефективну комунікацію між учасниками процесу пошуку тварин. Такий підхід гарантує надійне зберігання й керування профілями користувачів.

2. Оголошення (Ad) – сутність, яка акумулює основну інформацію про факт зникнення або знаходження тварини. Оголошення створюється користувачем і містить усю необхідну інформацію: текстовий опис, дату, прикріплене зображення, категорію (загублена/знайдена тварина), а також допоміжні характеристики тварини, які дозволяють швидше ідентифікувати її.

3. Місто (City) – сутність, призначена для забезпечення простої та однозначної географічної ідентифікації користувачів у межах заданого регіону. Її введення дозволяє стандартизувати та уніфікувати географічні дані в профілях, що сприяє точнішому локальному пошуку та формуванню релевантних результатів. У межах системи передбачено підтримку лише одного регіону — Києва та Київської області, що дає змогу обмежити набір допустимих значень, спростити інтерфейс та зменшити імовірність введення некоректної інформації.

4. Порода (Breed) – сутність, що описує породи домашніх тварин, з якими працює система. На даному етапі проєктування передбачається підтримка лише найпоширеніших видів тварин – собак та котів. Кожен запис у таблиці порід містить унікальний ідентифікатор, назву породи та належність до певного виду тварини.

5. Колір (Color) – сутність для візуальної характеристики тварини, яка відіграє значну роль у її ідентифікації. Часто саме колір є тією особливістю, яка дозволяє швидко впізнати тварину серед багатьох схожих.

Для забезпечення цілісності та унікальності даних кожна сутність має первинний ключ, що однозначно ідентифікує запис. Атрибути є атомарними, тобто неподільними, що відповідає вимогам нормалізації. Такий підхід дозволяє ефективно структурувати дані, забезпечити швидкий доступ і надійне збереження.

У логічній моделі між сутностями встановлено чіткі зв’язки. Один користувач може створити кілька оголошень (зв’язок один-до-багатьох), тоді як кожне оголошення пов’язане з одним містом, породою та кольором тварини (зв’язки багато-до-одного). Це знижує дублювання даних і підвищує ефективність системи.

При проектуванні логічної моделі дотримано принципів нормалізації даних, зокрема досягнуто третю нормальну форму (3НФ), що означає:

- атрибути кожної сутності є атомарними, не подільними на більш дрібні частини;
- кожен неключовий атрибут залежить лише від первинного ключа;
- відсутні транзитивні залежності, які могли б призводити до надлишковості даних.

Відповідність цим принципам гарантує, що модель не містить надлишкових даних, забезпечується логічна цілісність і однозначність зберігання інформації. Така модель легко масштабується, модифікується і забезпечує простоту при реалізації запитів до бази даних.

2.2 Побудова ER-діаграми

Для візуалізації логічної моделі даних було створено ER-діаграму (рис. 2.1) (діаграму «сутність–зв’язок»), яка є важливим інструментом у процесі проектування бази даних. Побудова ER-діаграми дозволяє наочно представити взаємозв’язки між сутностями, уточнити структуру даних і спростити процес переходу до фізичної моделі.

ER-діаграма містить усі основні сутності, описані вище, а також їх атрибути та зв’язки. Кожна сутність представлена у вигляді прямокутника, в якому вказані атрибути. Ключові поля (первинні ключі) виділені відповідним чином. Зв’язки між сутностями позначено лініями, які містять уточнення щодо типу зв’язку (один-до-одного, один-до-багатьох тощо).

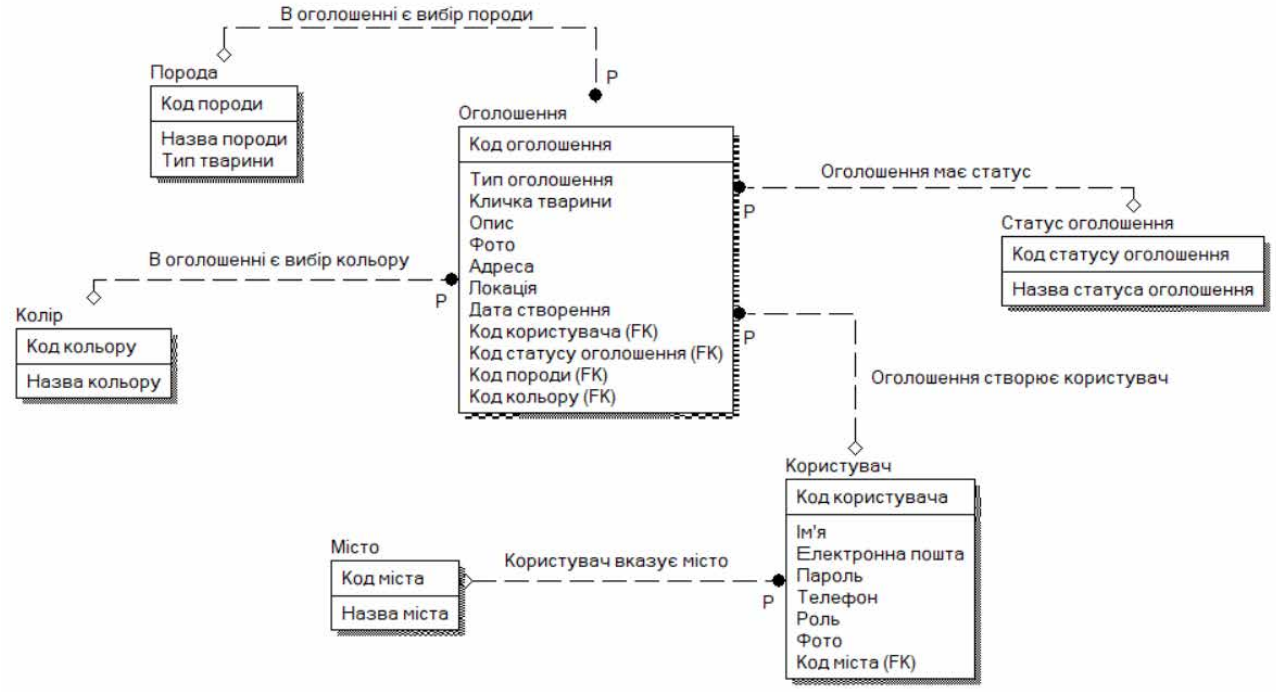

Рис. 2.1 ER-діаграма

Для коректного функціонування інформаційної системи важливо визначити взаємозв’язки між основними сутностями. Нижче описано всі ключові зв’язки логічної моделі даних:

1. Користувач – Оголошення: тип зв’язку: «один-до-багатьох» (1:P). Один користувач може створити кілька оголошень. Кожне оголошення, в свою чергу,

обов’язково пов’язане лише з одним автором. Це забезпечує відстеження відповідальності за кожну публікацію.

2. Оголошення – Порода: тип зв’язку: «багато-до-одного» (P:1). Кожне оголошення містить інформацію про одну породу тварини. Одна порода, у свою чергу, може зустрічатися у великій кількості оголошень, що дозволяє фільтрацію за породою.

3. Оголошення – Колір: тип зв’язку: «багато-до-одного» (P:1). В оголошенні фіксується один основний колір тварини. Один і той самий колір може бути вказаний у численних оголошеннях, що дозволяє здійснювати пошук за кольором.

4. Місто – Користувач: тип зв’язку: «один-до-багатьох» (1:P). Кожен користувач вказує лише одне місто. Одне й те саме місто може бути прив’язане до багатьох користувачів. Це дозволяє визначити, у якому населеному пункті перебуває автор оголошення.

5. Оголошення – Статус оголошення: тип зв’язку: «багато-до-одного» (P:1). Кожне оголошення має статус (наприклад, «створено», «очікує перевірки», «опубліковано»). Один статус може застосовуватися до багатьох оголошень.

Побудова ER-діаграми здійснювалася з використанням спеціалізованого програмного середовища ERWin. Такий підхід дозволяє досягти чіткої візуалізації логічної структури даних, що полегшує її подальше перенесення до Firebase та сприяє правильній організації колекцій і документів у хмарній базі.

Наявність ER-діаграми є не лише зручною для розробника, а й обов’язковою складовою технічної документації, що підтверджує відповідність логічної моделі основним принципам реляційного моделювання.

2.3 Вибір системи управління інформаційною базою

Система управління базами даних (СУБД) – це програмне забезпечення, яке забезпечує створення, зберігання, керування, модифікацію та отримання даних у базі даних. Вона виступає посередником між користувачами або додатками і самою базою даних, надаючи інтерфейс для роботи з інформацією та забезпечуючи її цілісність, безпеку й ефективність обробки.

Системи управління базами даних можна класифікувати за різними критеріями, зокрема:

1. Моделі даних**:** визначають, як дані будуть організовані та зберігатися в базі даних.

- Реляційні СУБД (RDBMS): дані зберігаються у вигляді таблиць, що взаємодіють між собою через відношення. Це одна з найпоширеніших моделей. Відомі реляційні СУБД включають MySQL, PostgreSQL, Oracle. Вони використовують мову SQL для доступу до даних. Підходять для структурованих даних з чітко визначеними зв'язками та обмеженнями.

- Об'єктно-орієнтовані СУБД: дані організовані у вигляді об'єктів, подібно до того, як це відбувається в об'єктно-орієнтованих мовах програмування (наприклад, Java або C++). Такі СУБД підтримують зберігання складних структур даних, що може бути корисним для специфічних задач. Прикладом об'єктно-орієнтованої СУБД є db4o.

- NoSQL СУБД: використовують інші моделі, окрім реляційної. Вони призначені для роботи з великими обсягами неструктурованих або частково структурованих даних. NoSQL включає такі підтипи, як документоорієнтовані (MongoDB, CouchDB), колонкові (Cassandra, HBase), графові (Neo4j, ArangoDB) та ключ-значення (Firebase Realtime Database, DynamoDB). Ці СУБД добре підходять для динамічних веб-додатків, мобільних застосунків та інших проектів, де структура даних може змінюватися.

- Файлові СУБД: це бази даних, що зберігають дані у вигляді файлів, які можуть мати різні формати. Ці системи зазвичай використовуються для

зберігання великих обсягів файлів без складних зв’язків між ними, наприклад, у документах, зображеннях, відео.

- Клієнт-серверні СУБД: СУБД, в яких сервер зберігає базу даних, а клієнт може взаємодіяти з цією базою через мережу. Це дозволяє централізовано управляти даними і забезпечувати доступ з різних пристроїв.

- Вбудовані СУБД: це системи управління базами даних, які інтегруються безпосередньо в програму чи систему. Вони часто використовуються в мобільних або вбудованих додатках, де є обмеження по ресурсах і простору. Прикладом такої СУБД є SQLite.

2. Тип доступу до даних**:**

- Індексований доступ: дані доступні за допомогою індексів, що дозволяють швидко знаходити елементи за певними критеріями. Цей підхід широко використовується в реляційних та NoSQL СУБД.

- Послідовний доступ: дані зчитуються послідовно, що може бути ефективно для великих обсягів даних, де необхідна швидка обробка пакетів інформації.

- Доступ за запитом: інтерфейс доступу, при якому дані отримуються на основі запитів користувачів, часто використовуючи SQL чи подібні мови запитів.

3. Функціональні можливості**:**

- Транзакції: багато СУБД підтримують транзакції, що дозволяє здійснювати кілька операцій як єдине ціле. Транзакції гарантують цілісність даних, навіть при збої системи.

- Масштабованість: можливість СУБД ефективно працювати з великими обсягами даних, масштабуючись вгору (додавання потужніших серверів) чи вшир (додавання нових серверів).

- Резервне копіювання та відновлення: СУБД можуть забезпечувати можливість автоматичного створення резервних копій і відновлення даних у разі збою системи.

- Безпека: важливим аспектом є механізми доступу та авторизації, що дозволяють обмежувати доступ до чутливих даних.

Після аналізу різних типів систем управління базами даних – зокрема реляційних, об'єктно-орієнтованих, файлових, вбудованих та NoSQL — мною було обрано використання Firebase Firestore як основної СУБД для розробки інформаційної системи пошуку домашніх тварин. Такий вибір був обґрунтований відповідністю цієї технології сучасним вимогам до мобільних застосунків: гнучкості, масштабованості, інтерактивності та зручності роботи з динамічними структурами даних.

Firebase Firestore — це хмарна документоорієнтована база даних, яка входить до складу платформи Firebase від компанії Google. Вона належить до класу NoSQL СУБД і забезпечує зберігання даних у вигляді документів, що згруповані у колекції. Кожен документ має структуру "ключ-значення" з можливістю вкладеності, що надає великої гнучкості у формуванні ієрархії та обсягу даних. Такий формат особливо добре підходить для застосунків, у яких структура інформації може змінюватися, доповнюватися або мати різну глибину в залежності від конкретного контенту — наприклад, оголошень про знайдених чи загублених тварин.

Однією з ключових переваг Firestore є можливість синхронізації даних у реальному часі між усіма клієнтськими пристроями, що забезпечує оперативне оновлення інформації без додаткових дій з боку користувача. Це створює приємний досвід користування і дає впевненість у тому, що дані в системі завжди актуальні.

Firestore не вимагає суворої попередньої схеми, як це властиво реляційним базам даних, що дозволяє без труднощів адаптувати модель даних до змінних вимог. Крім того, хмарна природа Firestore забезпечує автоматичне масштабування і високу надійність без необхідності ручного адміністрування серверної інфраструктури, що суттєво спрощує розробку та підтримку застосунку.

Окрема перевага – тісна інтеграція Firestore з іншими сервісами Firebase, такими як автентифікація, зберігання файлів, хмарні функції, push-сповіщення та аналітика. Завдяки цьому можливо реалізувати всі ключові компоненти бекенду

без потреби у сторонніх рішеннях. До того ж, сумісність з фреймворком Flutter робить Firestore ідеальним вибором для створення Android-застосунку.

Інформаційна база у цій системі має централізовану організацію: всі дані фізично зберігаються на серверах Google, що забезпечує єдине джерело правди і спрощує керування. Користувачі, незалежно від їхньої географічної локації, отримують доступ до однієї централізованої бази даних через інтернет. Хоча Firebase забезпечує високу доступність, реплікацію і масштабованість, структура даних та управління відбуваються централізовано – у єдиній хмарній системі.

Водночас Firestore підтримує певні механізми кешування та локального зберігання даних на пристроях клієнтів для роботи в офлайн-режимі, але це не робить базу розподіленою у класичному розумінні, адже всі зміни синхронізуються з центральним сервером.

Отже, структура бази є централізованою з хмарним розміщенням, що оптимально відповідає вимогам проекту щодо швидкого та безпечного доступу до інформації.

Вибір Firebase Firestore є логічним та обґрунтованим вибором після попереднього аналізу типів СУБД. Це сучасне рішення, що повністю відповідає технічним потребам проєкту та дозволяє реалізувати функціональну, масштабовану та зручну інформаційну систему для ефективного пошуку домашніх тварин.

2.4 Створення інформаційної бази

Firebase Firestore використовує ієрархічну модель зберігання даних. Основними структурними одиницями в Firestore є колекції (collections) та документи (documents). Колекції містять документи, які, у свою чергу, можуть містити інші вкладені колекції, що дозволяє будувати складну багаторівневу структуру даних. Однак у даній структурі застосунку реалізовано плоску модель, тобто всі основні сутності представлені окремими колекціями найвищого рівня, без вкладених колекцій усередині документів.

Такий підхід спрощує зчитування та запис інформації, особливо при реалізації мобільного застосунку, оскільки дозволяє працювати з даними без необхідності глибокої навігації через вкладені рівні.

Загальна структура організована за принципом: користувач → оголошення → довідники (порода, колір, місто).

У Firestore це реалізується як набір основних колекцій без вкладених підрівнів, але з явними посиланнями між документами:

```
Users/
└── {uId} (документ користувача)
ads/
└── {adId} (документ оголошення)
colors/
└── {colorId} (документ з назвою кольору)
breeds/
└── {breedId} (документ з назвою породи)
cities/
└── {cityId} (документ з назвою міста)
```

Структура кожного документа передбачає наявність полів, які відповідають властивостям певної сутності. Наприклад:

- Users зберігає такі дані, як: display_name, email, photo_url, phone_number, city, created_time.
- Ads містить інформацію про оголошення: nickname, description, location, photo, created_at, is_lost, а також посилання на довідкові дані — breed_id, colors_id, uid.

Обрана структура бази даних у Firestore забезпечує багато важливих переваг, що позитивно впливають як на продуктивність, так і на масштабованість застосунку.

По-перше, висока швидкодія досягається завдяки автоматичній індексації та використанню прямих посилань (DocumentReference), що дозволяє миттєво отримувати необхідні дані без використання ресурсомістких *JOIN*-запитів, як у традиційних реляційних базах.

По-друге, гнучка структура Firestore дає змогу легко масштабувати систему – додавання нових сутностей або полів (наприклад, коментарів, відгуків,

статусів оголошень) не потребує зміни існуючої архітектури, що значно спрощує підтримку та розвиток проєкту.

Третім важливим аспектом є зручність у використанні з Flutter. Завдяки офіційним пакетам, таким як cloud_firestore та firebase_auth, розробка проходить швидко та стабільно, а інтеграція з інтерфейсом відбувається без зайвих ускладнень.

Особливу увагу варто приділити системі безпеки, яка базується на UID автентифікованого користувача. Firestore дозволяє гнучко налаштовувати доступ не лише до колекцій, а й до окремих документів. Наприклад:

```
match /ads/{document} {
  allow read: if true;
  allow create: if request.auth != null
                && request.resource.data.uid == request.auth.uid;
  allow update, delete: if request.auth != null
                        && resource.data.uid == request.auth.uid;
}
```

Ці правила гарантують, що лише автентифіковані користувачі, які вказують свій uid у документі мають право створювати, редагувати або видаляти свої оголошення. Це забезпечує конфіденційність і захист персональних даних у системі.

2.4.1 Фізична реалізація бази даних

Для фізичної реалізації інформаційної бази застосунку було використано мову програмування Dart та фреймворк Flutter, у поєднанні з хмарною базою даних Firebase Cloud Firestore. Такий вибір зумовлений високою швидкістю розробки, кросплатформеністю, простотою вбудовування Firebase-сервісів та великою підтримкою серед розробників.

Взаємодія з Firestore у Flutter реалізована через офіційний пакет cloud_firestore, який забезпечує високорівневий інтерфейс для всіх CRUD-операцій (створення, читання, оновлення, видалення) над документами в базі даних.

Для збереження чистої архітектури застосунку реалізацію розділено на такі рівні (layers):

1. Data Layer (Рівень доступу до даних). Цей рівень відповідає за безпосередню взаємодію з Firestore. Тут знаходяться сервіси або репозиторії, які приховують логіку доступу до колекцій та документів бази даних. Вони виконують операції зчитування, додавання, оновлення й видалення даних, приховуючи деталі реалізації від решти системи.

Створено окремі сервіси для кожної основної сутності проєкту — зокрема, сервіс для оголошень. Цей сервіс відповідає за всі операції, пов’язані з взаємодією з колекцією ads у Firestore. У ньому реалізовано методи для отримання списку оголошень, створення нових документів, оновлення існуючих даних, а також видалення за необхідності.

У процесі реалізації я організувала обробку винятків (exception handling) на стороні сервісу, що дозволяє виявляти та інформувати користувача про можливі помилки (наприклад, проблеми з мережею або доступом до бази). Для передачі й зберігання даних використовується формат Map<String, dynamic>, який легко конвертується в документи Firestore.

2. Model Layer (Рівень моделей даних). Цей рівень відповідає за створення структур даних, що описують сутності бази даних. Кожна колекція Firestore має відповідний Dart-клас-модель, який визначає поля, їх типи, конструктори.

Кожна модель містить методи для перетворення з та у формат Map<String, dynamic>, що дозволяє легко передавати об’єкти між Firestore та внутрішніми компонентами застосунку. Це також спрощує серіалізацію/десеріалізацію даних та підтримує чисту структуру коду.

3. Presentation Layer (Презентаційний рівень).

Цей рівень відповідає за візуалізацію даних у Flutter та взаємодію користувача з інтерфейсом. Він включає екрани, форми, списки, карти та інші компоненти, які відображають інформацію з бази даних. Для керування станом і логікою використано контролери або ViewModel, що пов’язують UI з Data Layer.

Інтерфейс реалізовано на основі реактивної архітектури з використанням стрімів (Streams) та асинхронних запитів, що дозволяє оновлювати дані в реальному часі. Наприклад, після створення оголошення воно одразу з'являється на екрані без перезавантаження.

UX-практики враховано: форма перевіряє заповнення обов’язкових полів, повідомляє про помилки, а в профілі відображаються лише оголошення, пов’язані з UID користувача. Таким чином, реалізовано зв’язок між усіма рівнями архітектури — від введення даних користувачем до збереження в Firestore та зворотного відображення на екрані.

На рис. 2.2 представлено узагальнену схему процесу зчитування даних із бази Firestore, що ілюструє послідовність дій — від ініціалізації запиту до виведення даних на інтерфейс користувача.

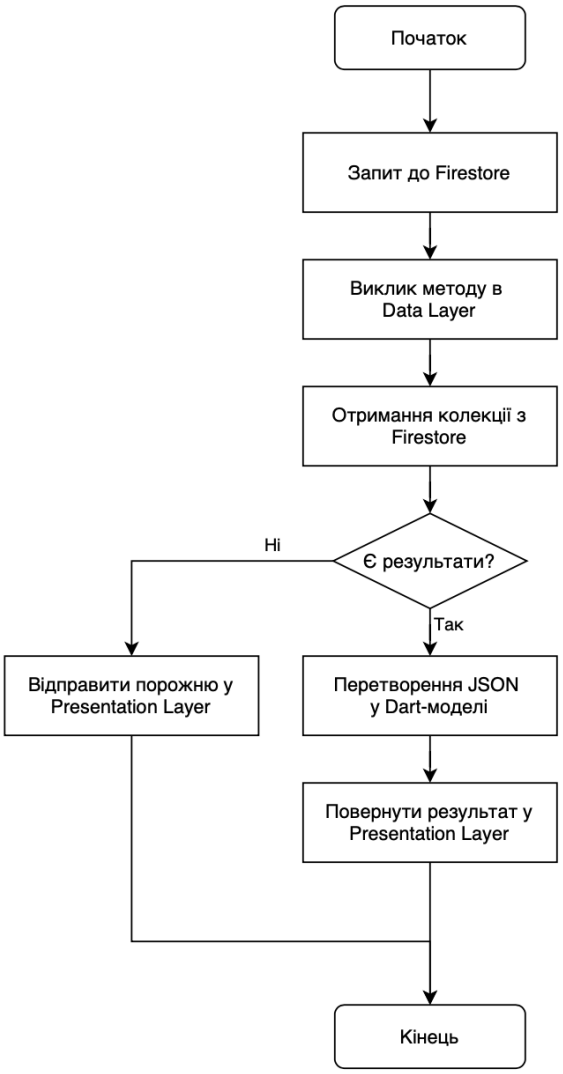

Рис. 2.2 Схема зчитування даних

На рис. 2.3 показано схему додавання даних у Firestore. Дані передаються в Data Layer, звідки виконується запит на створення нового документа в базі. У разі успіху дані зберігаються, при помилці — повідомлення передається у Presentation Layer.

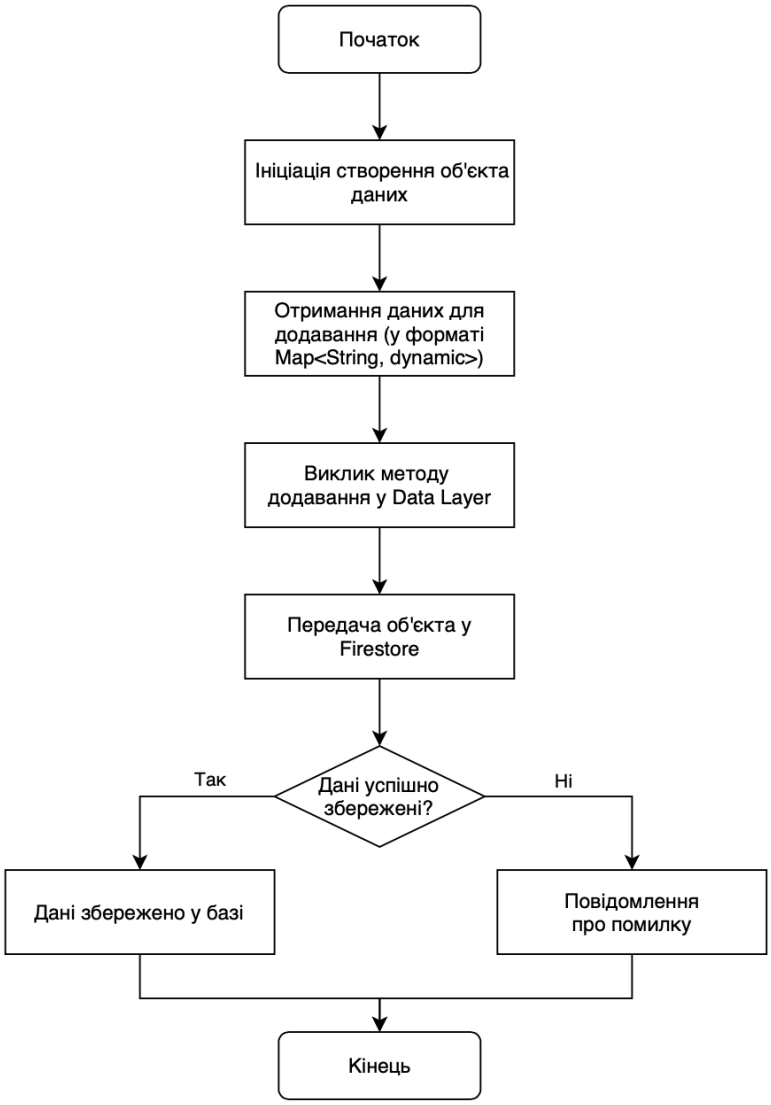

Рис. 2.3 Схема додавання даних

Фото тварини зазвичай зберігається в Firebase Storage, а в Firestore зберігається лише посилання на файл. Це дозволяє зменшити навантаження на базу даних та прискорити зчитування текстової інформації.

Усі операції з Firestore супроводжуються обробкою винятків. Це гарантує, що в разі відсутності інтернету, проблем з авторизацією або перевищення лімітів користувач отримає відповідне повідомлення про помилку, а дані не буде втрачено.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Прикладне програмне забезпечення (ППЗ) – це програми, створені для виконання конкретних прикладних задач користувача в межах певної предметної області. На відміну від системного ПЗ, яке забезпечує базову роботу комп’ютера або мобільного пристрою, прикладне програмне забезпечення є інструментом безпосередньої взаємодії з користувачем і реалізує функціональність, що відповідає його потребам. Для інформаційної системи пошуку домашніх тварин прикладне ПЗ включає реалізацію всіх основних сервісів додатку — від авторизації до відображення та фільтрації оголошень про домашніх тварин.

Під час розробки сучасних мобільних застосунків ППЗ має чітко організовану структуру, яка забезпечує зручну підтримку, масштабованість і розділення відповідальностей між логічними частинами проєкту.

3.1 Організаційна структура програмного забезпечення

У межах розробки інформаційної системи пошуку домашніх тварин було реалізовано багаторівневу організаційну структуру програмного забезпечення. Вона визначає логічний поділ коду на окремі частини або модулі, кожен з яких відповідає за конкретну функціональність або підсистему в межах загального програмного рішення. Такий підхід забезпечує зрозумілу архітектуру, спрощує обслуговування, тестування, масштабування та розвиток проєкту, а також дозволяє уникати дублювання коду і зменшити ризик виникнення помилок при зміні однієї з частин системи.

У розробці мобільного застосунку Petify було застосовано принципи Clean Architecture, що дозволило реалізувати чітке розділення відповідальностей між шарами програми. Така структура дозволяє кожному модулю зосередитися лише на своїх завданнях: інтерфейс користувача відповідає за відображення, бізнес-логіка – за обробку даних, а модулі доступу до даних – за взаємодію із зовнішніми джерелами (зокрема Firebase).

Узагальнено прикладне програмне забезпечення проєкту Petify поділяється на такі функціональні рівні:

1. Презентаційний рівень (Presentation Layer, UI)

Цей рівень відповідає за все, що бачить і з чим взаємодіє користувач. До його складу входять візуальні компоненти, екрани, кнопки, поля введення, а також логіка керування станом інтерфейсу. Саме тут реалізується механізм обробки введених даних, повідомлень, відображення помилок та відповідей системи.

Використання Flutter дозволяє будувати гнучкий інтерфейс за допомогою віджетів (Widgets), які легко адаптуються під різні розміри екранів Android-пристроїв. Flutter також підтримує гаряче оновлення (hot reload), що значно прискорює процес розробки та тестування UI.

Основні компоненти рівня:

- екрани входу, реєстрації, створення оголошення, перегляду тварин, профілю користувача;
- сторінка з фільтрацією, картою;
- обробка подій взаємодії (натискання, введення тексту тощо).

2. Доменний рівень (Domain Layer)

На цьому рівні зосереджена бізнес-логіка застосунку, тобто логіку обробки даних, прийняття рішень, валідації введених даних, а також сценарії користувача (use cases). Особливістю рівня є його незалежність від UI та зовнішніх джерел даних, що забезпечує повторне використання логіки в межах інших проєктів або платформ.

Рівень містить сутності, які описують основні об’єкти системи, а також інтерфейси та сервіси, що реалізують основні функції.

Основні елементи доменного рівня:

- Моделі оголошень, користувачів.
- Валідація введених користувачем даних перед відправкою.
- Сценарії: створення оголошення, оновлення профілю, фільтрація за параметрами.

Така структура дозволяє легко адаптувати логіку під нову платформу, змінюючи лише зовнішні шари.

3. Рівень доступу до даних (Data Layer)

Цей рівень відповідає за взаємодію з Firebase (Firestore, Firebase Storage, Firebase Authentication), Google Maps API та іншими зовнішніми сервісами. Саме тут реалізовані репозиторії, які отримують або зберігають дані, після чого передають їх у доменну логіку.

Firebase виступає основним хмарним бекендом, що забезпечує:

- реєстрацію та авторизацію користувачів (email + password);
- зберігання та отримання оголошень, фотографій, метаданих;
- хостинг зображень (Firebase Storage).
- роботу в реальному часі завдяки Firestore.

Також сюди входить інтеграція з Google Maps API, яка використовується для встановлення адреси оголошення через карту.

Ключові компоненти:

- FirebaseRepository, який реалізує інтерфейси репозиторіїв з доменного рівня;
- Data source-обгортки над Firestore, Storage, Authentication;
- API-клієнт для Google Maps.

Для кращого розуміння архітектури системи наведено діаграму вузлів (рис. 3.1), яка ілюструє взаємодію основних компонентів. На цій діаграмі відображено Android-пристрій користувача, де встановлений мобільний застосунок Petify, який реалізує основний функціонал: авторизацію, створення і перегляд оголошень, відображення міток на карті для адреси певного оголошення, роботу з фільтрами та управління профілем. Пристрій комунікує через захищений протокол HTTPS з серверами зовнішніх сервісів.

Сервер Firebase, який виконує роль основного бекенду, зберігає базу даних оголошень, користувачів та допоміжних даних (кольори, породи, міста) у Firestore. Він також забезпечує аутентифікацію користувачів, зберігання медіафайлів через Firebase Storage та підтримку реального часу. Інша важлива складова — сервер Google Cloud, що надає Google Maps API для інтеграції

картографічних функцій: геокодування адрес, відображення геоміток, фільтрації за відстанню.

Діаграма вузлів показує, як усі ці компоненти взаємодіють між собою, формуючи цілісну багаторівневу систему з чітко розмежованими зонами відповідальності. Такий підхід сприяє гнучкості, розширюваності та зручності підтримки програмного продукту, що є важливим аспектом при подальшому розвитку та масштабуванні інформаційної системи.

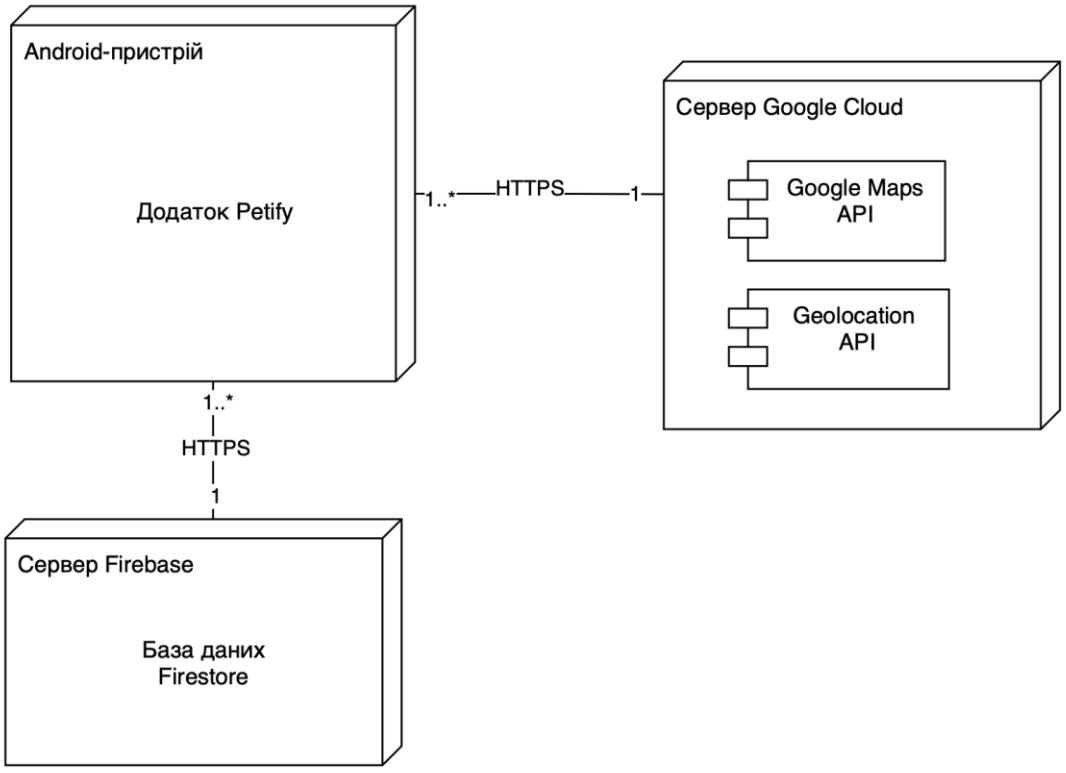

Рис. 3.2 Діаграма вузлів

3.2 Вибір інструментарію для створення ППЗ

На початку розробки прикладного програмного забезпечення дуже важливо правильно обрати інструменти, за допомогою яких буде реалізовано проєкт. Від цього вибору залежить не лише швидкість та зручність створення додатку, а й його стабільність, підтримка у майбутньому, можливість масштабування, оновлення та додавання нового функціоналу. Такими інструментами є мова програмування, середовище розробки (IDE), фреймворки, сторонні бібліотеки, сервіси для зберігання даних і аналітики тощо.

Буде детально розглянуто, які саме засоби були обрані для реалізації мобільного додатку для пошуку домашніх тварин, а також обґрунтовано, чому саме ці інструменти виявилися найбільш доцільними.

Для розробки інтерфейсу та основної логіки програми було обрано фреймворк **Flutter**. Це набір засобів розробки від Google, який дозволяє фахівцям створювати сучасні кросплатформні мобільні програми для IOS та Android з мінімальними витратами часу. Програмістам достатньо створити єдину кодову базу, яка потім окремо компілюватиметься на кожній ОС без будь-яких доопрацювань ПЗ.

Програми, створені на Flutter, мало чим поступаються нативним у плані продуктивності, при цьому їх розробка обходиться бізнесу значно дешевше. Це дає технології практично безмежні можливості розвитку, завдяки чому вона, незважаючи на відносну «молодість», успішно відвойовує частку ринку React Native.

На Flutter було створено багато програмних продуктів великих корпорацій, таких як Google Ads, Alibaba, Ebay і Cryptomaniac, що говорить про спроможність і поширеність технології.

Попри поширене переконання, сам по собі Flutter не є мовою програмування, а для створення програмного забезпечення він використовує **Dart**. Це мова, що інтерпретується, яка також як і SDK була створена корпорацією Google.

Реліз Dart відбувся ще восени 2011 року, проте суттєва популярність прийшла до нього дещо пізніше — у 2017 році, коли було анонсовано бета-версію Flutter. Вже в 2021 році в Google повідомили про майбутній вихід Flutter 2, через що інтерес до Dart піднявся на новий рівень.

Сьогодні Dart активно використовується для розробки різних програмних продуктів, включаючи серверні, десктопні, мобільні та веб-програми. Відмінною особливістю мови є те, що написаний на ньому код компілюється в байт-код JAVA, і на машинному рівні практично не відрізняється від нативного, що відкриває безмежні можливості для розробників.

Dart має досить багато важливих переваг, завдяки яким вона ідеально підходить для реалізації даного проєкту. Розглянемо основні з них докладніше:

1. Висока продуктивність та підтримка компіляції AOT/JIT. Dart використовує як JIT-компіляцію (для швидкої розробки з «гарячим перезавантаженням»), так і AOT-компіляцію (для максимальної продуктивності у фінальній збірці). Це дозволяє поєднувати швидку розробку з високою швидкістю роботи застосунку на пристрої користувача.

2. Простота у вивченні та читабельність коду. Dart має зрозумілий синтаксис, близький до JavaScript та Java, тому вона є доступною навіть для початківців. Завдяки цьому значно зменшується поріг входу та прискорюється командна розробка.

3. Гарна документація та підтримка спільноти. Як продукт Google, Dart має детально описану документацію, велику кількість прикладів та навчальних матеріалів. Активна спільнота постійно ділиться новими рішеннями та оновленнями.

4. Підтримка модульного тестування. Dart «з коробки» має вбудовані інструменти для юніт-тестування, що дозволяє розробникам перевіряти стабільність коду ще до запуску додатку, без потреби підключати сторонні фреймворки.

5. Стабільність та підтримка об’єктно-орієнтованого підходу. Dart підтримує об'єктно-орієнтоване програмування, включаючи спадкування, типізацію та інтерфейси. Це забезпечує стабільну архітектуру, зручність у масштабуванні та рефакторингу коду.

Для реалізації програмного забезпечення було обрано **Android Studio** – офіційне середовище розробки від Google, яке повністю підтримує Flutter та мову Dart. Це потужна IDE, побудована на базі IntelliJ IDEA, яка забезпечує всі необхідні інструменти для повного циклу розробки мобільного застосунку.

Android Studio забезпечує високий рівень зручності для розробника завдяки інтелектуальному автодоповненню коду, підсвічуванню синтаксису та інтерактивним підказкам.

Вбудовані інструменти для налагодження, перегляду логів та тестування дозволяють швидко знаходити та виправляти помилки, а також проводити профілювання продуктивності застосунку. Завдяки підтримці емуляторів Android, розробник може перевіряти роботу програми на різних віртуальних пристроях без необхідності використовувати фізичний смартфон.

Окрім цього, Android Studio має потужну екосистему плагінів, яка розширює її можливості – наприклад, для роботи з системами контролю версій, базами даних, хмарними сервісами чи інтеграцією з Firebase. Завдяки поєднанню гнучкості, стабільності та глибокої інтеграції з інструментами Flutter, Android Studio є одним із найкращих середовищ для реалізації сучасних мобільних додатків.

Для зберігання та обробки даних у цьому проєкті використовується хмарна платформа Firebase. Вона надає великий набір інструментів для мобільної розробки:

- Firebase Authentication – безпечна аутентифікація користувачів (електронна пошта і пароль).
- Cloud Firestore – зберігання оголошень, профілів користувачів тощо.
- Firebase Storage – зберігання зображень тварин.
- Firebase Analytics – збір аналітики використання додатку.

Для структурування коду використовується умовна архітектура MVVM з елементами Clean Architecture. Це забезпечує зручне розділення логіки, незалежність компонентів і кращу підтримуваність. У додатку чітко розділені:

- логіка взаємодії з даними (Data),
- логіка поведінки додатку (Domain),
- логіка інтерфейсу (Presentation).

Для роботи з геоданими у межах реалізації застосунку було використано **Google Maps API** у поєднанні з Flutter-плагінами, зокрема:

- google_maps_flutter – для вбудовування інтерактивної карти в інтерфейс застосунку;
- geocoding – для перетворення координат у зрозумілі адреси (та навпаки);

- geolocator – для визначення геолокації користувача в реальному часі.

Цей стек інструментів дозволив ефективно реалізувати функціонал, пов’язаний із визначенням поточної геопозиції користувача, побудовою карти, відображенням точок (наприклад, де було втрачено або знайдено тварину), а також введенням адрес вручну з автоматичним визначенням координат. Таке рішення забезпечує високу точність, стабільну роботу з геоданими та зручність для користувача, що є критично важливим для застосунків, пов’язаних з географічним пошуком або розміщенням інформації.

Таким чином, обраний набір інструментів повністю відповідає технічним вимогам та масштабу проєкту. Він дозволяє швидко та якісно створити сучасний мобільний додаток з підтримкою Firebase, зручним UI та стабільною архітектурою, що легко масштабуватиметься у майбутньому.

3.3 Алгоритмізація та програмування програмних модулів

Розробка програмного забезпечення мобільного застосунку Petify здійснювалась за модульним підходом, що дозволяє розділити систему на логічно ізольовані частини, кожна з яких виконує окрему функцію в рамках цілісного ППЗ. Відповідно до організаційної структури, описаної в попередньому підрозділі, алгоритмізація та програмування модулів охоплює наступні ключові елементи модуль аутентифікації, модуль створення оголошень, модуль фільтрації й пошуку, модуль інтеграції з картою (Google Maps API) та модуль профілю користувача.

Модуль аутентифікації користувача забезпечує безпечний доступ користувачів до функціоналу застосунку шляхом реєстрації та авторизації. Для реалізації механізмів автентифікації обрано хмарну платформу Firebase Authentication, що дозволяє інтегрувати механізми входу за допомогою електронної пошти та пароля, з високим рівнем безпеки та зручністю для розробника.

Алгоритм реєстрації нового користувача (рис 3.3):

1. Введення ім’я, номеру телефону, електронної пошти та пароля.

2. Перевірка коректності введених даних (email-формат, довжина пароля).

3. Успішна перевірка – надсилання запиту на створення облікового запису до Firebase.

4. Створення документа користувача в базі даних Firestore.

5. Перенаправлення на головний екран застосунку.

6. Місто та фото користувач може редагувати в профілі.

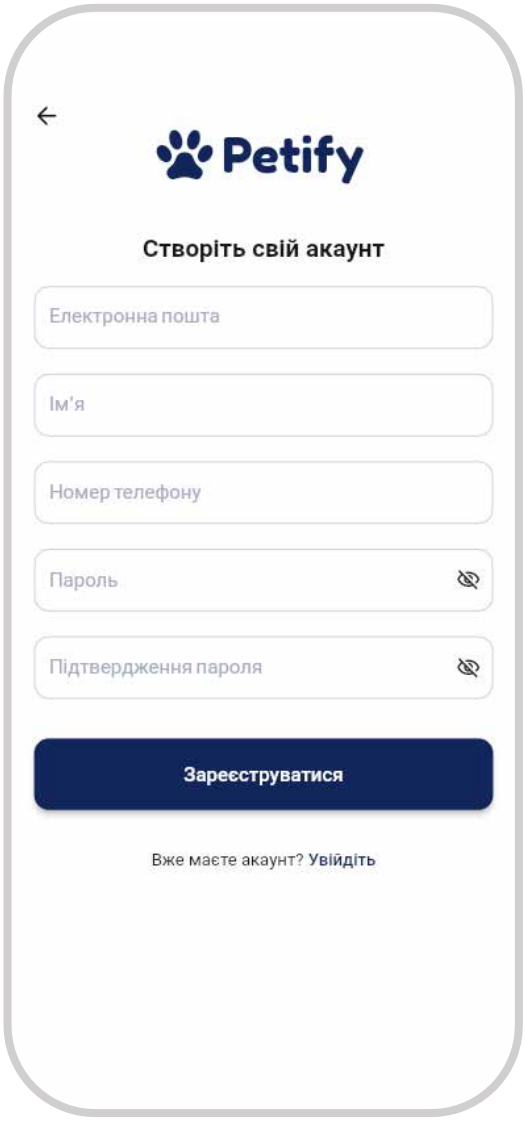

Рис. 3.3 Реєстрація користувача

Процедура авторизації відбувається аналогічно, з тією відмінністю, що замість створення нового облікового запису система отримує унікальний токен доступу та ініціалізує сесію користувача.

Модуль створення оголошень є одним з ключових у структурі застосунку, оскільки забезпечує можливість публікації оголошень про зникнення або

знаходження тварин. Кожне оголошення містить розширену інформацію про тварину, що дозволяє іншим користувачам оперативно реагувати на події.

Інтерфейс реалізовано у вигляді зручної форми з полями для заповнення (рис 3.4):

- тип оголошення (зникнення або знаходження);
- тип тварини (собака або кіт)
- кличка тварини (тільки для зниклих);
- порода, колір;
- фотографія тварини;
- опис (прикмети тварини, місце зникнення);
- місце події — через інтеграцію з Google Maps API;
- номер телефону (одразу видно в формі без змоги редагування).

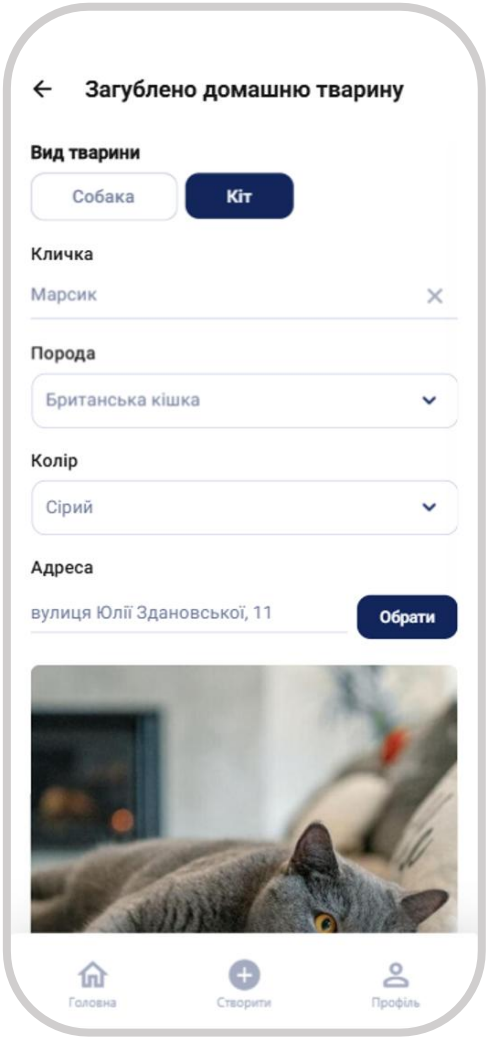

Рис. 3.4 Створення оголошення

Після заповнення усіх обов’язкових полів та підтвердження, зібрані дані формуються в об’єкт і надсилаються до бази даних Firestore, де зберігаються в окремій колекції ads. Після успішного створення оголошення користувач отримує підтвердження про публікацію.

Завдяки цьому процес створення оголошень стає не лише функціональним, але й зручним, що є важливим у емоційно складних ситуаціях, пов’язаних із втратою домашньої тварини.

Для зручного пошуку серед наявних записів реалізовано модуль фільтрації оголошень, що дозволяє здійснювати пошук за кількома параметрами: типом тварини (кіт або собака), породою, кольором, а також за відстанню від користувача. На рис. 3.5 представлено інтерфейс фільтру.

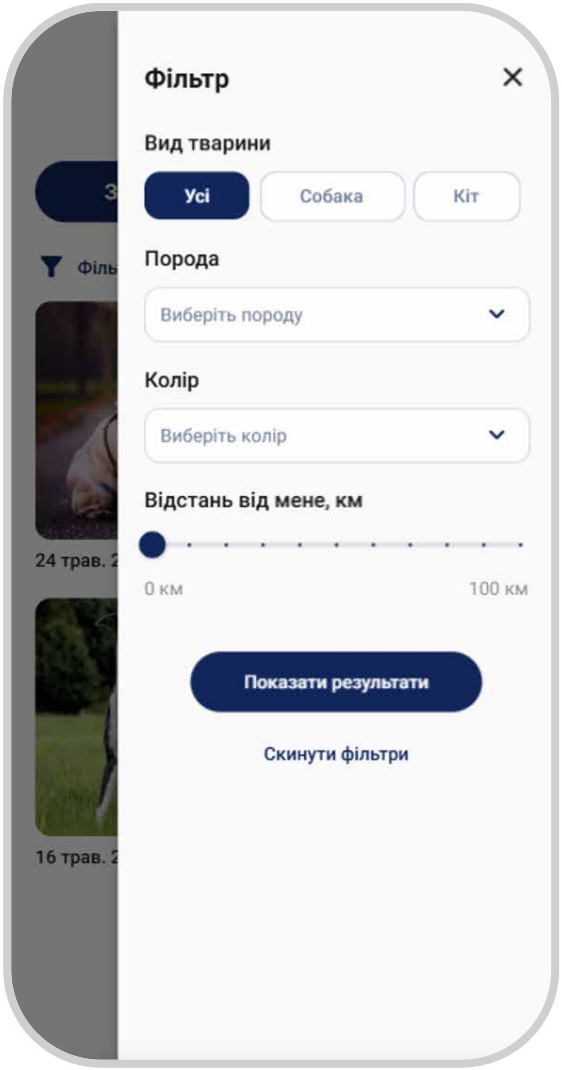

Рис. 3.4 Створення оголошення

Алгоритм функціонування фільтрації полягає у зчитуванні обраних критеріїв, формуванні відповідного запиту до бази даних Firestore та виведенні лише релевантних результатів. У випадку вибору фільтрації за відстанню, реалізується функція calculateDistance(). Код функції представлено у Додатку А. Вона обчислює відстань у кілометрах між двома географічними точками. Це дозволяє зосередитись на тих оголошеннях, які мають найвищий рівень актуальності для користувача.

Важливим елементом для точності введення даних є модуль інтеграції з Google Maps API, який дозволяє зручно взаємодіяти з картою під час створення оголошення. При створенні оголошення користувач має змогу обрати місце події безпосередньо на карті, скористатися автоматичним визначенням поточного розташування або ввести адресу вручну (рис 3.6). Це підвищує достовірність та географічну точність даних.

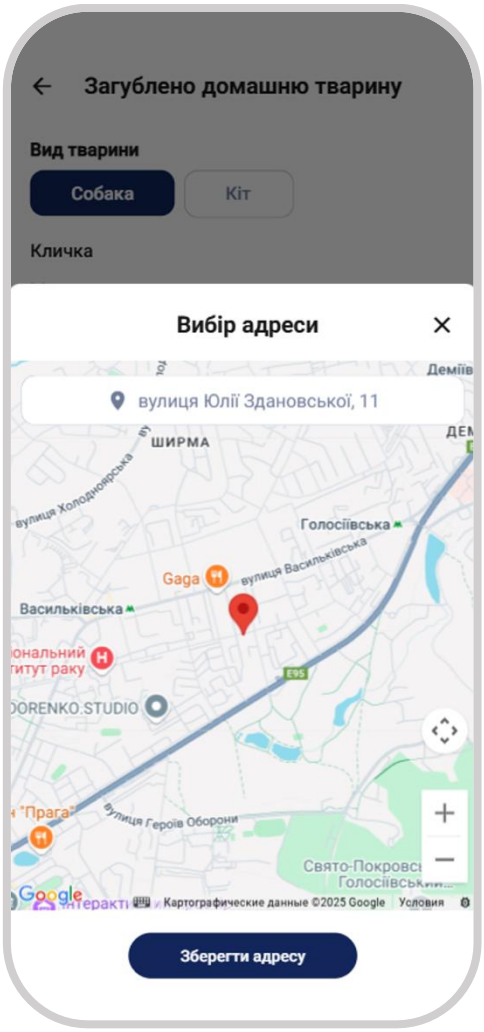

Рис. 3.6 Редагування профілю користувача

У цій послідовності дій, останнім є модуль профілю користувача, який виконує функцію персонального кабінету, де зосереджена ключова інформація про користувача та доступ до управління оголошеннями.

Цей модуль надає можливість переглядати, змінювати або доповнювати власну інформацію, зокрема ім’я, місто проживання, фотографію профілю та інші персональні параметри. Окрім того, саме тут можна змінити пароль для входу, що є важливим елементом підтримки безпеки облікового запису. Наведено інтерфейс редагування профілю користувача на рис. 3.6.

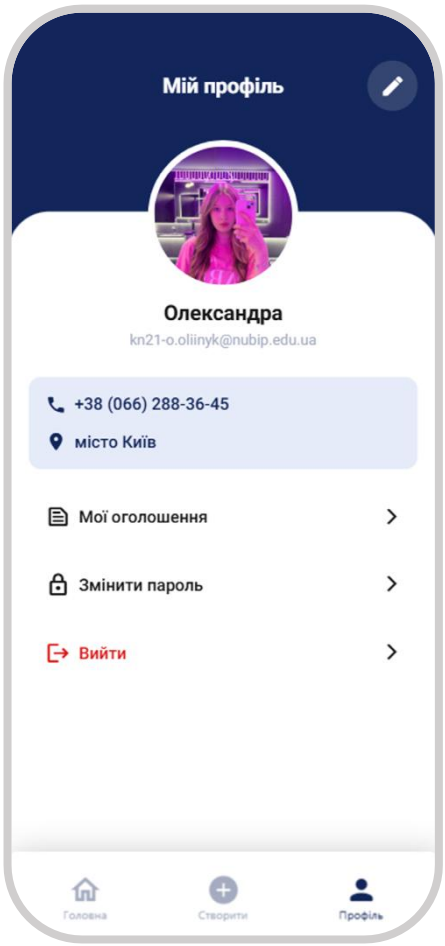

Рис. 3.6 Редагування профілю користувача

При відкритті профілю застосунок миттєво зчитує актуальні дані з бази Firestore та відображає їх у зручному інтерфейсі. Користувач може внести потрібні зміни, після чого натискає кнопку збереження. Система автоматично перевіряє правильність введеної інформації, наприклад, формат електронної пошти або номеру телефону. У разі успішної валідації зміни оновлюються в базі даних, що гарантує цілісність і точність інформації.

Окрім особистих даних, у профілі відображаються оголошення, які користувач публікував раніше. Вони доступні для редагування (рис. 3.7), перегляду та за потреби – повного видалення. Це дозволяє ефективно управляти своїми оголошеннями, підтримувати їхню актуальність та оновлювати у разі змін у ситуації.

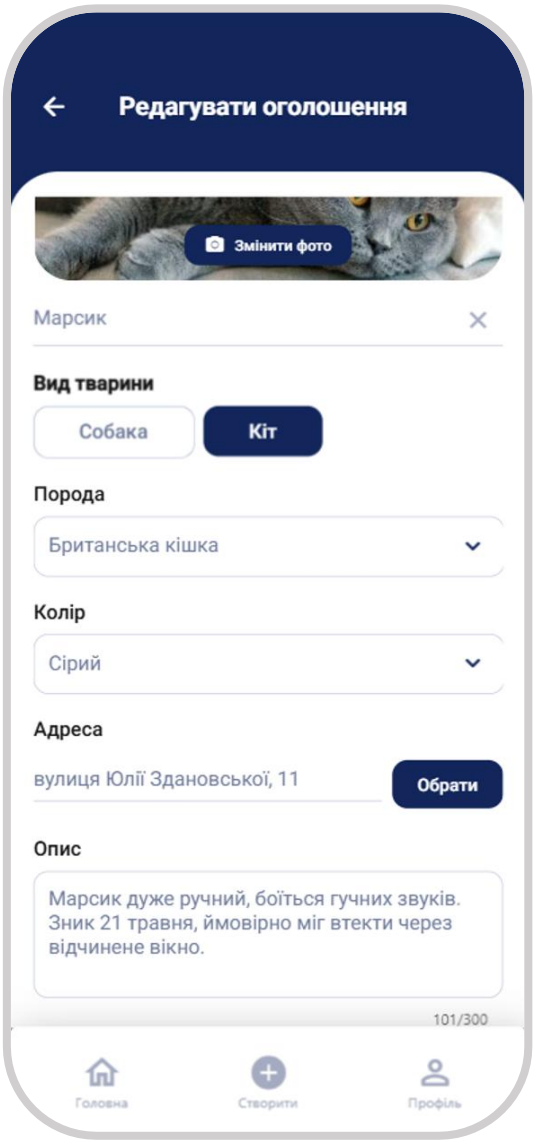

Рис. 3.7 Редагування власного оголошення користувачем

Усі описані модулі реалізовано відповідно до принципів Clean Architecture, що передбачає чітке розділення відповідальностей, інверсію залежностей, мінімізацію зв’язків між компонентами, що, у свою чергу, забезпечує легкість тестування, розширення та підтримки системи. Використання хмарної платформи Firebase спростило реалізацію процесів автентифікації, зберігання даних та їх синхронізацію.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ВИКОРИСТАННЯ СИСТЕМ

4.1 Тестування системи

Тестування програмного забезпечення є важливою складовою розробки, що дозволяє переконатися у правильності роботи системи, її стабільності та відповідності очікуванням користувачів для подальшої експлуатації.

Саме завдяки тестуванню можливо виявити та усунути помилки, забезпечити коректну роботу системи відповідно до технічного завдання, а також підтвердити відповідність функціоналу очікуванням користувачів. Тестування виконує роль своєрідного «фільтра якості», який гарантує стабільність, надійність і безпечність роботи застосунку у реальних умовах.

Тож, тестування – це процес виконання програми з метою виявлення дефектів. У цьому процесі здійснюється підбір тестових сценаріїв, що охоплюють усі основні функції системи, які мають бути перевірені на коректність реалізації. Також враховуються граничні умови, нештатні ситуації та поведінка системи при різних вхідних даних.

Процес тестування складається з кількох етапів:

1. Планування тестування. На цьому етапі визначаються цілі, типи тестування, ресурси, інструменти та терміни проведення. Для нашої системи тестування проводилось з урахуванням усіх функціональних можливостей, описаних у технічному завданні.

2. Розробка тестових сценаріїв (тест-кейсів). Тест-кейси створювались для кожного модуля системи: реєстрації та авторизації користувачів, створення та редагування оголошень, перегляду списку знайдених/загублених тварин, фільтрації, взаємодії з картою та ін.

3. Виконання тестів. Тестування проводилось вручну на різних пристроях та з використанням емуляторів у середовищі Android Studio. Під час цього етапу фіксувалися помилки, які надалі передавалися розробникам для усунення.

4. Аналіз результатів. За підсумками тестування формувався звіт, у якому вказувалися знайдені баги, ступінь їх критичності та статус виправлення.

Тестування програмного забезпечення можна класифікувати за різними критеріями. Найбільш поширеним є поділ на:

1. Функціональне тестування. Спрямоване на перевірку того, *що* система робить. Воно оцінює відповідність поведінки програми функціональним вимогам. Приклади:

- модульне тестування;
- системне тестування;
- інтеграційне тестування;
- приймальне тестування.

2. Нефункціональне тестування. Перевіряє *як* система виконує свої функції, тобто такі характеристики як продуктивність, надійність, безпека, зручність використання тощо. Сюди входить:

- тестування продуктивності;
- навантажувальне тестування;
- юзабіліті-тестування.

У даному випадку, оскільки мова йде про розроблену інформаційну систему для пошуку домашніх тварин, найдоцільніше буде провести функціональне **(системне та приймальне)** тестування. Вони дозволяють оцінити роботу системи в цілому та перевірити її відповідність очікуванням кінцевого користувача.

4.1.1 Системне тестування

Системне тестування – це комплексна перевірка всієї системи як єдиного цілого. Воно дозволяє виявити помилки у взаємодії компонентів, протестувати логіку виконання функцій, а також перевірити коректність взаємодії з базою даних та сторонніми сервісами.

У випадку Petify системне тестування охоплювало всі основні функції:

- реєстрацію та авторизацію користувачів;

- створення та редагування оголошень;
- використання фільтрів (тип, колір, порода, місце події);
- відображення карт Google Maps;
- інтеграцію з Firebase (автентифікація, база даних, збереження зображень);
- профіль користувача та зміну пароля.

Розроблений системний тест-кейс у табл. 1 описує набір умов, дій та очікуваних результатів, який розроблений для перевірки конкретного сценарію використання всієї системи або її значної частини.

Таблиця 1

Приклад системного тест-кейсу

№	Назва сценарію	Кроки	Очікуваний результат	Статус
1	Реєстрація нового користувача	1. Відкрити екран реєстрації 2. Ввести email і пароль	Користувача зареєстровано, перенаправлено на головний екран	Успішно
2	Авторизація з валідними даними	1. Ввести email і пароль 2. Натиснути «Увійти»	Користувач потрапляє на головний екран	Успішно
3	Фільтрація оголошень	1. Обрати «Кіт» 2. Натиснути «Застосувати фільтри»	Виводяться лише оголошення з типом «Кіт»	Успішно
4	Вибір адреси з карти	1. Натиснути кнопку «Обрати» 2. Вибрати місце на карті або написати частину адреси в пошуку	Відображається обрана адреса в полі форми	Успішно
5	Додавання оголошення	1. Заповнити всі поля 2. Натиснути «Опублікувати»	Оголошення збережено у базі, з’являється у списку	Успішно

Системне тестування проводилось як вручну (на реальному пристрої та в емуляторі Android Studio), так і за допомогою простих автоматизованих скриптів на основі Flutter Driver.

4.1.2 Приймальне тестування

Приймальне тестування є завершальним етапом у процесі перевірки програмного забезпечення та має вирішальне значення для ухвалення рішення про готовність системи до реального використання. Його головна мета – переконатися, що розроблений продукт повністю відповідає вимогам, викладеним у технічному завданні, а також задовольняє очікування кінцевих користувачів та зацікавлених сторін.

Цей тип тестування, як правило, здійснюється за участі представників замовника або майбутніх користувачів системи, які перевіряють програму з позиції її практичного використання у щоденних сценаріях. Приймальне тестування дає змогу побачити продукт очима реального користувача та виявити можливі недоліки, які могли бути непомітними на попередніх етапах тестування.

У процесі приймального тестування оцінювалися такі аспекти:

- Зручність реєстрації: наскільки просто новому користувачу створити обліковий запис без сторонньої допомоги.
- Інтуїтивність додавання оголошення: чи може користувач легко знайти відповідну функцію та успішно додати інформацію про загублену або знайдену тварину.
- Обробка помилкових ситуацій: чи система правильно реагує на типові проблеми — наприклад, некоректно введений email, відсутність підключення до інтернету тощо.

Приклади сценаріїв, за якими проводилося приймальне тестування:

1. Реєстрація нового користувача та створення оголошення.

Сценарій: Користувач уперше відкриває застосунок, проходить процес реєстрації, додає оголошення про загублену тварину, додає параметри, фото, адресу та опис.

Очікуваний результат: Усі дії виконуються без помилок, дані зберігаються, оголошення відображається у списку.

2. Перевірка працездатності елементів інтерфейсу

Сценарій: Користувач навігає застосунок, натискаючи різні кнопки та перемикачі.

Очікуваний результат: Кожна кнопка виконує свою функцію, переходи між екранами здійснюються коректно.

3. Повідомлення про помилки

Сценарій: Користувач вводить email у неправильному форматі під час реєстрації.

Очікуваний результат: Виводиться зрозуміле повідомлення про помилку, користувача не пропускає далі без виправлення.

4. Динамічне оновлення списків

Сценарій: Користувач публікує нове оголошення.

Очікуваний результат: Новий запис миттєво з’являється у відповідному списку, без потреби перезапуску застосунку.

Результати приймального тестування Petify були позитивними. Усі критично важливі бізнес-сценарії були пройдені без збоїв. Застосунок виявився зручним для нових користувачів, функціонал працював згідно з очікуваннями, а можливі нештатні ситуації (як-от неправильні дані або відсутній інтернет) оброблялися коректно.

Для зручного тестування були використані інструменти:

- Android Studio Emulator – для емуляції різних пристроїв і версій Android;
- Firebase Debug Console – для моніторингу запитів до бази даних та автентифікації;
- Logcat – для аналізу логів застосунку;
- Реальний пристрій (смартфон Samsung з Android 12) – для перевірки поведінки на фізичному рівні.

4.2 Вимоги до апаратного та програмного забезпечення

Важливим етапом під час розробки та впровадження інформаційної системи є визначення вимог до апаратного та програмного забезпечення. Це дозволяє не лише забезпечити коректну роботу системи, але й оптимізувати витрати на інфраструктуру та її обслуговування.

4.2.1 Діаграма розміщення

Для наочного представлення взаємозв’язків між основними компонентами системи було побудовано діаграму розміщення (рис 4.1). Це один із типів структурних діаграм в UML, який використовується для візуалізації фізичного розгортання компонентів системи на апаратному або віртуальному обладнанні.

Ця діаграма показує, на яких вузлах (сервери, пристрої, віртуальні машини, мобільні пристрої тощо) розміщуються програмні елементи, а також як між собою пов’язані ці вузли через канали зв’язку (наприклад, HTTPS, Bluetooth, локальна мережа).

Система складається з трьох основних вузлів:

1. Мобільний додаток (Android-пристрій)

Користувач встановлює Android-застосунок Petify.apk, який містить інтерфейс користувача та логіку взаємодії з хмарними сервісами. У додаток вбудований конфігураційний файл google-services.json, який забезпечує правильну інтеграцію з Firebase.

2. Хмарний сервер Firebase Cloud

Використовується як бекенд-сховище для даних користувачів. Включає:

- Firebase Authentication — для реєстрації та автентифікації користувачів;
- Cloud Firestore — для зберігання оголошень про тварин;
- Firebase Storage — для збереження фотографій тварин.

3. Хмарна інфраструктура Google Cloud

Застосовується для реалізації геолокаційної функціональності:

- Google Maps API — для відображення карт та визначення місць;
- Geolocation API — для отримання координат пристрою.

Усі компоненти взаємодіють між собою через захищені HTTPS-з’єднання, що забезпечує безпечну передачу даних між клієнтом і хмарними сервісами.

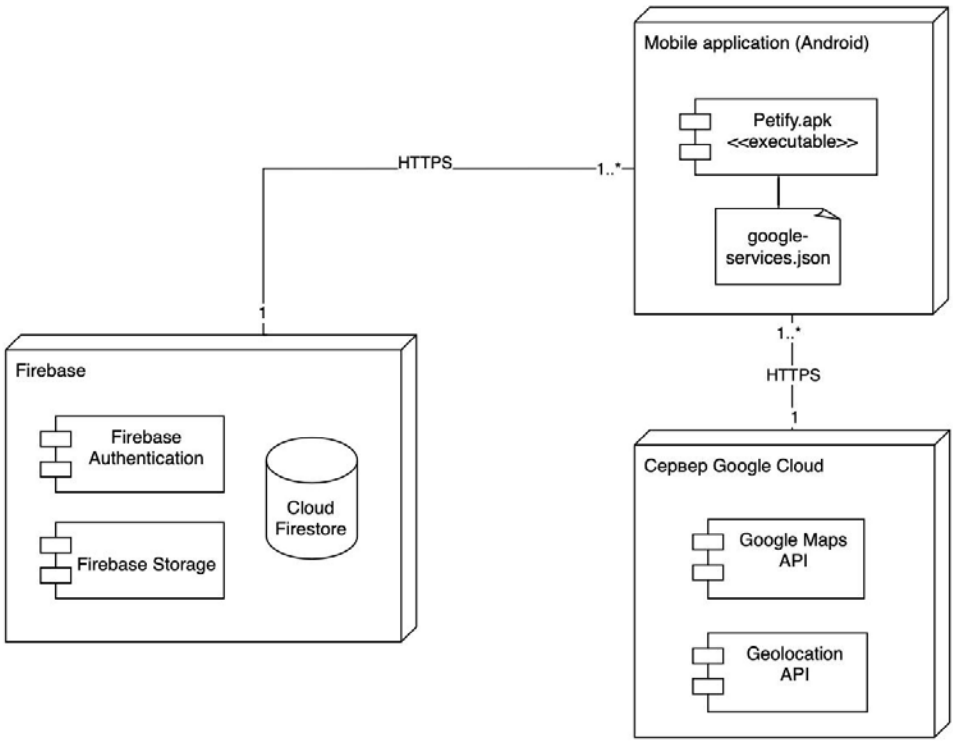

Рис. 1.1 Діаграма розміщення

4.2.2 Вимоги до апаратного забезпечення

Для забезпечення стабільної та комфортної роботи мобільного застосунку Petify достатньо базових технічних характеристик сучасних мобільних пристроїв. Оскільки основна взаємодія користувача з інформаційною системою здійснюється через Android-додаток, апаратні вимоги стосуються саме клієнтського середовища — тобто смартфона, на якому встановлено застосунок.

Щоб гарантувати коректну роботу інтерфейсу, швидке завантаження контенту, а також використання карт та геолокаційних функцій, мобільний пристрій користувача повинен відповідати таким мінімальним вимогам:

- Операційна система: Android 10.0 (API 29) або новіша версія, що гарантує підтримку сучасних бібліотек, API та сервісів Google.
- Процесор: з архітектурою ARMv8 (64-бітна), із тактовою частотою не менше ніж 1.8 ГГц. Бажано двоядерний або багатоядерний, для забезпечення плавної роботи інтерфейсу.

- Оперативна пам’ять (RAM): не менше 2 ГБ, що дозволяє стабільно працювати з динамічними елементами інтерфейсу.
- Вільна пам’ять: наявність принаймні 100 МБ вільного простору для встановлення додатку, кешування даних та зберігання тимчасових файлів (зображень тощо).
- Інтернет-з’єднання: стабільний доступ до мережі через Wi-Fi або мобільний зв’язок (3G/4G/5G) для здійснення запитів до бази даних та отримання контенту.
- GPS-модуль: наявність GPS-модуля або підтримка геолокації через мережу для автоматичного визначення поточного положення користувача, фільтрації оголошень за відстанню та зручного вибору адреси на карті.

4.2.3 Вимоги до програмного забезпечення

Належне функціонування мобільного застосунку Petify потребує відповідності програмного середовища сучасним вимогам платформи Android. Застосунок створено з використанням сучасних фреймворків та інтегрується з хмарними сервісами, тому ефективна робота системи залежить від коректного налаштування програмного забезпечення на стороні користувача.

Основні програмні вимоги до клієнтського середовища:

- Android SDK версії 29 або вище: набір інструментів для розробки й налагодження Android-додатку.
- Служби Google Play: обов’язкова наявність для роботи з Firebase, геолокацією, картами та сервісами автентифікації.
- Google Maps API: для відображення інтерактивної карти, вибору місця розташування та фільтрації оголошень за відстанню.
- Системні дозволи:
  - доступ до камери та/або галереї – для додавання фто домашньої тварини при створенні оголошення;
  - доступ до геолокації – для фільтрації оголошень за відстанню та попереднього визначення адреси при створенні оголошенн;

o доступ до мережі Інтернет – для взаємодії з хмарною базою даних Firebase та завантаження контенту.

- Сумісність із Flutter SDK: застосунок створено за допомогою кросплатформеного фреймворку Flutter, тому користувацький пристрій має підтримувати сучасні UI-компоненти, а також апаратне прискорення рендерингу.
- Безпечне з'єднання (HTTPS): усі запити до віддалених сервісів мають відбуватись через захищений протокол, що гарантує конфіденційність переданих даних.

Таким чином, чітке визначення вимог до апаратного та програмного забезпечення є ключовим фактором для стабільного функціонування мобільного застосунку Petify. Забезпечення сумісності з актуальними версіями Android, підтримка необхідних сервісів Google, коректна робота з геолокацією, картами та хмарними базами даних створюють надійне середовище для ефективної взаємодії користувача із системою. Збалансовані технічні вимоги дозволяють досягти високої продуктивності без перевантаження пристрою, що особливо важливо для повсякденного використання застосунку широким колом користувачів.

4.3 Склад інсталяційного пакету

Для забезпечення належного функціонування інформаційної системи користувачеві необхідно встановити мобільний застосунок на свій Android-пристрій. Інсталяційний пакет включає всі програмні компоненти, що необхідні для розгортання та функціонування мобільного застосунку на пристрої користувача.

Формування цього пакету відбувається з урахуванням архітектури системи, визначеної на діаграмі розміщення (розділ 4.2.1), де основними вузлами є мобільний пристрій, хмарні сервіси Firebase та інфраструктура Google Cloud. Нижче представлено повну структуру інсталяційного пакету та його складові.

4.3.1 Основні компоненти інсталяційного пакету

Інсталяційний пакет Petify постачається у вигляді APK-файлу (Android Package). Це стандартний формат файлів інсталяції для додатків, що працюють на операційній системі Android. Фактично, це архівний файл, який містить усі елементи, необхідні для успішної установки та запуску програми на пристрої Android. Якщо порівнювати з іншими операційними системами, то APK для Android є аналогом файлів .exe у Windows або .dmg у macOS.

До складу інсталяційного пакету входять такі основні елементи:

- Файл APK – основний файл Android-додатку Petify.apk, що містить зібраний код застосунку, ресурси інтерфейсу, стилі, шрифти та необхідні бібліотеки.
- Конфігураційний файл google-services.json – забезпечує налаштування зв’язку з Firebase, містить ідентифікатори проєкту, ключі автентифікації та інші параметри інтеграції.
- Модулі Flutter – скомпільовані Dart-файли з UI-компонентами, бізнес-логікою, сервісами взаємодії з базою даних, API геолокації та обробкою зображень.
- Зовнішні залежності (packages) – набір підключених пакетів з pubspec.yaml, таких як firebase_core, cloud_firestore, firebase_auth, firebase_storage, google_maps_flutter, geolocator та інші. Вони інтегровані до APK під час збирання та визначають загальну структуру пакету.
- Сервісні маніфести – AndroidManifest.xml, що визначає дозволи (доступ до камери, геолокації, Інтернету), а також обмеження й конфігурацію запуску додатку.
- Ресурси застосунку – локалізовані рядки, зображення, іконки, карти маршрутів, файли стилів та інші допоміжні ресурси, необхідні для відображення інтерфейсу.

Інсталяційний пакет створюється за допомогою офіційних інструментів Flutter SDK та Android Studio, підписується цифровим ключем і може бути розповсюджений через Google Play або безпосередньо за допомогою APK-файлу.

4.3.2 Післяінсталяційні вимоги

Після інсталяції додатку APK-файл взаємодіє з хмарними сервісами, тому інсталяційний пакет не є самодостатнім – він розгортається у зв’язці з хмарною інфраструктурою. Тому для забезпечення повної працездатності додатку необхідно дотримуватись певних умов:

1. Підключення до інтернету: додаток потребує стабільного інтернет-з’єднання, оскільки всі ключові функції (перегляд оголошень, завантаження фотографій, створення нових записів, оновлення профілю) реалізуються через мережеву взаємодію з Firebase та іншими API. Без доступу до мережі більшість функціональності буде недоступною.

2. Автентифікація користувача через Firebase: одразу після запуску додатку користувач має пройти авторизацію або реєстрацію. Для цього використовується Firebase Authentication, що підтримує логін за допомогою електронної пошти та пароля. Без успішної автентифікації користувач не отримає доступу до персоналізованого функціоналу (створення, перегляд, та редагування оголошень, а також редагування профілю).

3. Зв’язок із Firebase Cloud Firestore: додаток встановлює з’єднання з базою даних Firestore для:

- отримання списку оголошень про загублених або знайдених тварин;
- фільтрація за типом тварини, кольором, породою, відстанню;
- збереження нових оголошень, створених користувачами;
- перегляду особистих оголошень з профілю користувача.

4. Робота з Firebase Storage: для відображення фотографій у стрічці, а також для завантаження зображень під час створення оголошень додаток використовує хмарне сховище Firebase Storage. Завантажені фото автоматично асоціюються з відповідним записом у базі даних.

5. Використання Google Maps API: додаток інтегрований із Google Maps API, що дозволяє користувачам обирати місце, де було втрачено чи знайдено тварину, безпосередньо на карті. Це також дає змогу іншим користувачам бачити локацію у зручному форматі.

Отже, для ефективної роботи застосунку Petify необхідно, щоб після інсталяції було забезпечено підключення до інтернету, коректне функціонування Firebase-сервісів, а також наявність активних API-ключів для картографічних сервісів Google.

4.3.3 Середовище розробки та збірки інсталяційного пакету

Розробка мобільного застосунку та формування інсталяційного пакету, здійснюється в комплексному середовищі розробки з використанням сучасних інструментів, платформ та фреймворків. Далі детально описано основні компоненти середовища.

Flutter SDK – основна технологія, що використовується для створення користувацького інтерфейсу додатку. Це фреймворк з відкритим кодом, розроблений Google, який дозволяє створювати нативні мобільні додатки для Android та iOS з єдиною кодовою базою. Flutter SDK містить набір інструментів, бібліотек і команд для створення UI, логіки та доступу до системних ресурсів. Весь застосунок Petify створений саме на Flutter, що забезпечує високу продуктивність, зручність у розробці та швидке прототипування.

Android Studio є офіційним середовищем розробки для створення Android-додатків, у якому виконується безпосередня робота над проєктом. Середовище містить потужний редактор Dart-коду, інтеграцію з Flutter SDK та емулятор пристроїв Android для тестування. Саме тут виконується написання основного коду, інтеграція з Firebase та формування фінального APK-файлу.

Gradle – система автоматичної збірки, яка керує процесом компіляції, збору залежностей, обробки ресурсів і створення інсталяційного пакету. Вона працює у фоновому режимі Android Studio, виконує налаштування проєкту на основі конфігурацій у build.gradle, а також взаємодіє з pubspec.yaml для підключення зовнішніх Dart-бібліотек. Gradle відповідає за трансформацію вихідного Dart-коду у машинний код Android, інтеграцію з Firebase SDK, додавання необхідних permission та забезпечення узгодженості середовища розгортання.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи розроблено інформаційну систему для пошуку домашніх тварин – Android-додаток Petify, що дозволяє користувачам публікувати оголошення про загублених або знайдених тварин, переглядати оголошення інших користувачів, фільтрувати їх за параметрами (тип тварини, порода, колір, відстань) та взаємодіяти через контактні дані. Основою архітектури проєкту стало використання Flutter з принципами Clean Architecture, що забезпечило гнучкість, масштабованість та легку підтримку програмного коду.

У процесі розробки було реалізовано такі функціональні можливості:

- авторизація та реєстрація з використанням Firebase Authentication;
- збереження оголошень у базі даних Firebase Firestore;
- завантаження фотографій у Firebase Storage;
- інтеграція з Google Maps API для відображення та вибору адреси;
- створення профілю користувача з можливістю редагування даних та перегляду власних оголошень.

Станом на завершення роботи модерація оголошень не реалізована, проте її впровадження заплановане на наступних етапах розробки. Ця функціональність дозволить підвищити довіру до системи, зменшити кількість помилкових або некоректних оголошень і покращити загальну якість взаємодії користувачів.

Виконане проєктування охоплювало всі необхідні етапи: аналіз предметної області, розробку архітектури, моделювання структури бази даних, побудову діаграм UML, формування інтерфейсів користувача, створення і тестування додатку, підготовку інсталяційного пакету.

Система відповідає всім вимогам, що визначені в технічному завданні, і демонструє позитивні техніко-економічні характеристики. У порівнянні з сучасними аналогами, такими як вебсайти або додатки загального оголошувального характеру (наприклад, OLX чи Facebook-групи), Petify має

вузьку спеціалізацію, орієнтовану саме на пошук тварин, простий і швидкий інтерфейс для додавання оголошень, автоматизовану класифікацію та геолокацію, що дозволяє скоротити час пошуку та підвищити ймовірність повернення тварини власнику.

До нових технічних рішень, реалізованих у межах цієї роботи, можна віднести:

- використання Clean Architecture з чітким розділенням шарів даних, домену та представлення, що дозволяє легко масштабувати систему;
- розробку кастомізованих фільтрів оголошень з урахуванням типу, породи, кольору та відстані від користувача;
- інтеграцію Firebase як єдиної хмарної платформи для автентифікації, зберігання, синхронізації даних та роботи в реальному часі.

Запропоноване рішення може бути використане в межах зоозахисних організацій, волонтерських спільнот, ветеринарних клінік, притулків для тварин, а також приватними користувачами. Враховуючи його адаптивність і доступність, додаток має потенціал для масштабування в межах регіону, а згодом – і по всій території України.

Рекомендовано розширити функціональність додатку у майбутньому, зокрема, додати чат між користувачами, push-сповіщення про нові оголошення поблизу, систему підтвердження власності на тварину, а також реалізувати вебверсію з синхронізацією даних. Це дозволить ще більше підвищити ефективність пошуку та розширити аудиторію користувачів.

Таким чином, поставлені в роботі завдання повністю виконані, система створена з урахуванням сучасних технічних вимог, має конкурентні переваги, та може бути успішно застосована в реальних умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Pet911 – Сервіс пошуку загублених тварин в Україні [Електронний ресурс]. – Режим доступу: https://pet911.com.ua.

2. Zooposhuk – Онлайн-платформа для пошуку домашніх улюбленців [Електронний ресурс]. – Режим доступу: https://zooposhuk.com.ua.

3. Evergreen. UML-діаграми: короткий гід [Електронний ресурс]. – Режим доступу: https://evergreens.com.ua/ua/articles/uml-diagrams.html.

4. Зосім М. Уніфікована мова моделювання UML [Електронний ресурс] / Макс Зосім. Режим доступу: https://www.maxzosim.com/unifikovana-mova-modeluvannia/.

5. Foxminded. UML-діаграми: навіщо та як їх створювати [Електронний ресурс]. – Режим доступу: https://foxminded.ua/uml-diagramy/.

6. SpaceLab. Яку мову використовує Flutter і що в ній особливого [Електронний ресурс]. – Режим доступу: https://spacelab.ua/articles/kakoy-yazyk-ispolzuyet-flutter-i-chto-v-nem-osobennogo/.

7. Pressman R.S. Software Engineering: A Practitioner’s Approach. – 8th Edition. McGraw-Hill, 2014. — 816 p.

8. Sommerville I. Software Engineering. – 10th Edition. Pearson, 2015. – 640 p.

9. Flutter Documentation [Електронний ресурс]. – Режим доступу: https://flutter.dev/docs.

10. Google Developers. Material Design Guidelines [Електронний ресурс]. – Режим доступу: https://material.io/design.

11. IEEE Software Engineering Body of Knowledge (SWEBOK) [Електронний ресурс]. – Режим доступу: https://www.computer.org/education/bodies-of-knowledge/software-engineering.

ДОДАТОК А

Код програми

Сторінок 6

КИЇВ – 2025

А. 1 Реєстрація користувача

```
class CreateAccountModel extends FlutterFlowModel<CreateAccountWidget> {

  FocusNode? emailFocusNode;
  TextEditingController? emailController;
  String? Function(BuildContext, String?)? emailValidator;

  FocusNode? nameFocusNode;
  TextEditingController? nameController;
  String? Function(BuildContext, String?)? nameValidator;

  FocusNode? phoneFocusNode;
  TextEditingController? phoneController;
  final phoneMask = MaskTextInputFormatter(mask: '+38 (###) ###-##-##');
  String? Function(BuildContext, String?)? phoneValidator;

  FocusNode? passwordFocusNode;
  TextEditingController? passwordController;
  late bool isPasswordVisible;
  String? Function(BuildContext, String?)? passwordValidator;

  FocusNode? confirmPasswordFocusNode;
  TextEditingController? confirmPasswordController;
  late bool isConfirmPasswordVisible;
  String? Function(BuildContext, String?)? confirmPasswordValidator;

  @override
  void initState(BuildContext context) {
```

```
    isPasswordVisible = false;
    isConfirmPasswordVisible = false;
  }

  @override
  void dispose() {
    emailFocusNode?.dispose();
    emailController?.dispose();

    nameFocusNode?.dispose();
    nameController?.dispose();

    phoneFocusNode?.dispose();
    phoneController?.dispose();

    passwordFocusNode?.dispose();
    passwordController?.dispose();

    confirmPasswordFocusNode?.dispose();
    confirmPasswordController?.dispose();
  }
}
```

А. 2 Авторизація користувача

```
class LoginModel extends FlutterFlowModel<LoginWidget> {

  FocusNode? emailFocusNode;
  TextEditingController? emailController;
  String? Function(BuildContext, String?)? emailValidator;
```

```
  FocusNode? passwordFocusNode;
  TextEditingController? passwordController;
  late bool isPasswordVisible;
  String? Function(BuildContext, String?)? passwordValidator;

  @override
  void initState(BuildContext context) {
    isPasswordVisible = false;
  }

  @override
  void dispose() {
    emailFocusNode?.dispose();
    emailController?.dispose();
    passwordFocusNode?.dispose();
    passwordController?.dispose();
  }
}
```

А. 3 Функція обчислення відстані

```
double? calculateDistance(LatLng? point1, LatLng? point2) {

  if (point1 == null || point2 == null) {
    return null;
  }
  double _deg2rad(double deg) {
    return deg * (math.pi / 180);
  }
```

```
  const earthRadius = 6371; // км
  final dLat = _deg2rad(point2.latitude - point1.latitude);
  final dLon = _deg2rad(point2.longitude - point1.longitude);
  final a = math.sin(dLat / 2) * math.sin(dLat / 2) +
      math.cos(_deg2rad(point1.latitude)) *
          math.cos(_deg2rad(point2.latitude)) *
          math.sin(dLon / 2) *
          math.sin(dLon / 2);
  final c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a));
  return earthRadius * c;
}
```

А. 4 Зміна паролю

```
import 'package:firebase_auth/firebase_auth.dart';

Future<bool> changePassword(
    String currentPassword, String newPassword, String email) async {
  try {
    User? user = FirebaseAuth.instance.currentUser;

    // Authenticate the user.
    AuthCredential credential = EmailAuthProvider.credential(
      email: email,
      password: currentPassword,
    );
    await user?.reauthenticateWithCredential(credential);

    // If auth is successful, change the password.
```

```
    await user?.updatePassword(newPassword);

    // Password changed successfully.
    return true;
  } catch (e) {
    // Handle reauthentication errors and password change errors.
    return false;
  }
}
```

А. 5 Зчитування геолокації користувача

```
Future<LatLng> getCurrentLocation() async {
  LocationPermission permission = await Geolocator.checkPermission();
  if (permission == LocationPermission.denied) {
    permission = await Geolocator.requestPermission();
    if (permission == LocationPermission.denied) {
      // Permission denied — return default location
      return LatLng(0.0, 0.0);
    }
  }

  final position = await Geolocator.getCurrentPosition(
    desiredAccuracy: LocationAccuracy.high,
  );
  return LatLng(position.latitude, position.longitude);
}
```

А. 6 Фрагмент створення оголошення (Загублено домашню тварину)

```
class CreateAdLostModel extends FlutterFlowModel<CreateAdLostWidget> {
  String selectedType = 'Собака';

  DocumentReference? breedToRef;
  String breedToAd = 'Не вказано';

  DocumentReference? colorToRef;
  String colorToAd = 'Не вказано';

  FormFieldController<List<String>>? choiceChipsTypeValueController;
  String? get choiceChipsTypeValue =>
      choiceChipsTypeValueController?.value?.firstOrNull;
  set choiceChipsTypeValue(String? val) =>
      choiceChipsTypeValueController?.value = val != null ? [val] : [];

  FocusNode? textFieldNicknameFocusNode;
  TextEditingController? textFieldNicknameTextController;
  String? Function(BuildContext, String?)?
textFieldNicknameTextControllerValidator;

  String? dropDownBreedValue;
  FormFieldController<String>? dropDownBreedValueController;
  DocumentReference? docOutput;
  BreedsRecord? breedRefToText;

  String? dropDownClValue;
  FormFieldController<String>? dropDownClValueController;
```

```
  DocumentReference? docOutputCl;
  ColorsRecord? colorRefToText;

  bool isDataUploading = false;
  FFUploadedFile uploadedLocalFile = FFUploadedFile(bytes:
Uint8List.fromList([]));
  String uploadedFileUrl = '';

  FocusNode? descriptionFieldFocusNode;
  TextEditingController? descriptionFieldTextController;
  String? Function(BuildContext, String?)? descriptionFieldTextControllerValidator;

  FocusNode? phoneNumberFocusNode;
  TextEditingController? phoneNumberTextController;
  final phoneNumberMask = MaskTextInputFormatter(mask: '+38 (###) ###-##-##');
  String? Function(BuildContext, String?)? phoneNumberTextControllerValidator;
  @override
  void initState(BuildContext context) {}
  @override
  void dispose() {
    textFieldNicknameFocusNode?.dispose();
    textFieldNicknameTextController?.dispose();
    descriptionFieldFocusNode?.dispose();
    descriptionFieldTextController?.dispose();
    phoneNumberFocusNode?.dispose();
    phoneNumberTextController?.dispose();
  }
}
```