Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Хоптяр Іван Юрійович
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Мобільний застосунок для організації допомоги безпритульним тваринам
Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ. 2101093.01.19.ПЗ

Виконав студент IV курсу, група ІПЗ-21-1 _____ Іван ХОПТЯР
Підпис Ім'я, ПРІЗВИЩЕ

Керівник д-р фіз.-мат. наук, професор _____ Леонід БЕДРАТЮК
Науковий ступінь, звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. пед. наук, доцент _____ Наталія ПРАВОРСЬКА
Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

2 червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
2.01. 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Хоптяр Іван Юрійович

Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Мобільний застосунок для організації допомоги безпритульним тваринам

Керівник кваліфікаційної роботи Бедратюк Леонід Петрович, д-р фіз.-мат. наук, професор

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.01.2025 р. № 8

2. Строк подання студентом роботи на кафедру 01.06.2025 р.

3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, проєктування мобільного застосунку, програмна реалізація та тестування

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Презентаційні матеріали (слайди, 15 шт.), діаграма варіантів використання, діаграма архітектури вебзастосунку (модульна клієнт-серверна архітектура), ER-діаграма, блок-схема процесу реєстрації, блок-схема процесу додавання тварини, діаграма станів навігації додатку, інфраструктурна діаграма

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Н.І. Праворська, канд. пед. наук, доцент	5.05	5.05
Антиплагіат	Форкун Ю. В., канд. техн. наук, доцент	12.05	29.05

7. Дата видачі завдання « 02 » січня 2025р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою дипломного проєктування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12 – 31.12.2024	
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2025	
3 Проєктування програмного забезпечення	21.02 – 20.03.2025	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2025	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2025	
6 Попередній захист КвР	Травень 2025	
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2025	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2025	

Студент _____ Підпис Хоптяр І. Ю. Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Підпис Бедратюк Л. П. Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Мобільний застосунок для організації допомоги безпритульним тваринам».

Автор роботи: Хоптяр Іван Юрійович.

Керівник роботи: Бедратюк Леонід Петрович.

Пояснювальна записка: 117 с., 23 рис., 3 табл., 3 дод., 41 джерела.

Графічна частина: 15 слайдів.

МОБІЛЬНИЙ ЗАСТОСУНОК, ДОПОМОГА БЕЗПРИТУЛЬНИМ ТВАРИНАМ, ПОШУК ЗА ГЕОЛОКАЦІЄЮ, ASP.NET WEB API, .NET MAUI, Azure.

Метою цієї кваліфікаційної роботи є розробка мобільного застосунку PetCompas для фіксації безпритульних і загублених тварин. Застосунок дозволяє додавати оголошення з фото та геолокацією, переглядати тварин на мапі й координувати дії користувачів.

У роботі проаналізовано предметну область, вивчено існуючі сервіси, визначено вимоги до системи, обрано стек технологій та спроєктовано архітектуру.

Клієнтську частину реалізовано з використанням .NET MAUI, що забезпечує кросплатформенну підтримку для Android та iOS. Серверну частину побудовано на основі ASP.NET WebAPI з використанням REST-підходу, а для обробки запитів застосовано MediatR. Додаток також інтегрує нейромережу для розпізнавання тварин на фото.

Застосунок має простий інтерфейс, підтримує фільтрацію, обмін повідомленнями та надсилання сповіщень. Він може стати корисним інструментом для волонтерів, притулків, ветеринарів і небайдужих громадян, які прагнуть швидко реагувати на випадки безпритульності тварин, а також сприяє підвищенню рівня цифровізації соціально важливих процесів у сфері захисту тварин.

3.05.2025
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.2101093.01.19.ПЗ	Пояснювальна записка	117		
2	А4		Завдання на кваліфікаційну роботу	1		
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні матеріали	15		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ. 2101093.01.19.ПЗ		
Виконав		Хоптяр І.Ю.		30.05.25	Мобільний застосунок для організації допомоги безпритульним тваринам	Літ.	Арк. 5 / Аркушів 117
Керівник		Бедратюк Л. П.		30.05.25			
Рецензент		Павлюкова О.О.		30.05.25			
Н. Контр.		Праворська Н. І		28.05.25	Пояснювальна записка	ХНУ, ІПЗ-21-1	
Зав. Каф.		Бедратюк Л.П.		30.05.25			

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API	–	програмний інтерфейс прикладного програмування
ASP.NET	–	фреймворк для створення вебзастосунків на платформі .NET
CNN	–	Convolutional Neural Network – згорткова нейронна мережа
CRUD	–	Create, Read, Update, Delete – основні операції з базами даних
DTO	–	Data Transfer Object – об'єкт передачі даних
ER	–	Entity-Relationship – модель зв’язків сутностей
GPS	–	глобальна система позиціонування
HTTP	–	протокол передавання гіпертексту
JSON	–	JavaScript Object Notation – формат обміну даними
JWT	–	JSON Web Token – токен для передачі захищеної інформації
MAUI	–	фреймворк для кросплатформених застосунків
ORM	–	об'єктно-реляційне відображення
REST	–	архітектурний стиль взаємодії з вебсервісами
UI	–	User Interface – інтерфейс користувача
UML	–	Unified Modeling Language – уніфікована мова моделювання
YOLO	–	You Only Look Once – алгоритм розпізнавання об’єктів
XML	–	eXtensible Markup Language – розширювана мова розмітки

ВСТУП

У сучасному світі технології відіграють важливу роль у вирішенні соціально значущих проблем. Зокрема, стрімкий розвиток мобільних додатків, засобів штучного інтелекту, хмарних технологій та геолокаційних сервісів створює нові можливості для оперативної взаємодії, обміну інформацією та координації зусиль між різними групами користувачів. Усе більше прикладних цифрових рішень спрямовані не лише на комерційні потреби, а й на покращення якості життя суспільства, підтримку екологічних та гуманітарних ініціатив.

Однією з актуальних проблем, що набирає масштабів у багатьох країнах, є проблема безпритульних тварин. Щодня тисячі собак, котів та інших домашніх улюбленців опиняються на вулиці без належного нагляду, захисту чи шансів на новий дім. Основними причинами цього явища є безвідповідальність власників, низький рівень обізнаності населення, відсутність обов'язкової ідентифікації тварин, контрольованого розмноження та ефективної державної політики у сфері зоозахисту. В результаті — тварини часто гинуть від хвороб, голоду, травм або стають джерелом потенційної загрози для людей (через сказ, агресивну поведінку тощо).

Водночас існує багато небайдужих людей, волонтерів, працівників притулків та ветеринарів, які готові допомагати бездомним тваринам, але не мають достатньо зручних і надійних інструментів для координації своїх дій. Часто така допомога є спонтанною, незареєстрованою, обмеженою рамками особистих контактів або випадкових повідомлень у соціальних мережах, що унеможливлює створення централізованої системи реагування. У таких умовах інформація про знайдену тварину швидко втрачається, не доходить до потенційного власника або залишається без відповіді.

Саме тому важливо розробити ефективний, технологічно гнучкий та доступний інструмент, який дозволить оперативно реєструвати випадки

виявлення бездомних або загублених тварин, фіксувати їхнє місцезнаходження, додавати фотографії, описові характеристики та координати, а також забезпечити швидку передачу цієї інформації всім, хто може допомогти. Централізована база даних у вигляді мобільного застосунку з візуалізацією на карті та функцією автоматичного розпізнавання тварин може суттєво змінити підхід до розв'язання цієї проблеми.

У цьому контексті було обрано тему дипломного проєкту — розробка мобільного застосунку PetCompas, який орієнтований на створення цифрової платформи для швидкої реєстрації знайдених тварин, пошуку загублених улюбленців та об'єднання зусиль користувачів, волонтерів і притулків. Додаток дозволяє фотографувати знайдених тварин, додавати до них короткий опис, автоматично визначати місце зйомки та публікувати запис у спільній базі. Інші користувачі мають змогу переглядати мапу з такими позначками, знаходити тварин поблизу, контактувати з авторами оголошень і надавати необхідну допомогу.

PetCompas покликаний не лише полегшити процес комунікації між людьми, які хочуть допомогти тваринам, а й зробити це швидше, простіше та ефективніше. Завдяки простому інтерфейсу, підтримці геолокації, системі фільтрації, push-сповіщенням і можливості розпізнавання тварин за фото, додаток стане сучасним інструментом у руках небайдужих людей.

Основні завдання дипломного проєкту:

– дослідити предметну область та визначити ключові проблеми, які вирішує застосунок.

– проаналізувати існуючі аналоги та оцінити їхні сильні й слабкі сторони.

– визначити вимоги до функціоналу та продуктивності застосунку.

– спроєктувати архітектуру системи та взаємодію її компонентів.

– розробити структуру та модель бази даних.

– створити UML-діаграми для візуалізації роботи застосунку.

– спроєктувати та розробити інтерфейс користувача з урахуванням зручності використання.

– реалізувати серверну та клієнтську частини застосунку.

– провести тестування та усунути можливі помилки.

Результатом дипломного проєкту стане функціонуюча інформаційна система у вигляді мобільного застосунку, яка дозволяє швидко реєструвати знайдених або загублених тварин, додавати їх фото, описові характеристики та місцезнаходження, а також здійснювати пошук по інтерактивній карті. Застосунок підтримуватиме інтеграцію з нейронною мережею для автоматичного розпізнавання тварин на фото, що підвищить точність і швидкість ідентифікації. Завдяки зручному та зрозумілому інтерфейсу, система буде доступною широкому колу користувачів, включаючи волонтерів, співробітників притулків, ветеринарів та звичайних громадян.

Запропоноване рішення не тільки підвищить ефективність волонтерської діяльності та координації між учасниками процесу порятунку тварин, але й створить передумови для подальшого розвитку цифрових сервісів у сфері захисту тварин. PetCompas стане прикладом соціально відповідального програмного забезпечення, яке сприяє формуванню цифрової культури допомоги, підвищує обізнаність громадськості про проблему безпритульних тварин і стимулює об’єднання зусиль для її вирішення.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз та опис предметної області, її структурних та функціональних особливостей

Проблема безпритульних тварин [1, 7] є актуальною в багатьох країнах світу і має глибоке історичне коріння. За оцінками Всесвітньої організації охорони здоров'я, кількість бездомних собак у світі перевищує 200 мільйонів. У країнах з високим рівнем урбанізації, таких як США та країни Європи, проблема контролюється через стерилізацію та притулки, проте у багатьох державах, зокрема в країнах Азії, Африки та Латинської Америки, ситуація залишається критичною.

Однією з головних причин поширення безпритульних тварин [1, 10] є безвідповідальне ставлення власників, які відмовляються від своїх улюбленців через зміну життєвих обставин, відсутність можливостей для догляду або небажання контролювати розмноження тварин. Багато тварин опиняються на вулиці через недостатню систему реєстрації та відсутність державних програм контролю популяції. У країнах з жорсткими програмами контролю, таких як Німеччина або Нідерланди, діють ефективні системи ідентифікації та прилаштування безпритульних тварин, що дозволяє значно зменшити їхню кількість.

Такі тварини стикаються з численними загрозами: вони можуть загинути від голоду, холоду, хвороб або потрапити під транспортні засоби. Крім того, неконтрольоване розмноження бездомних тварин може призвести до поширення небезпечних захворювань, таких як сказ. Водночас у суспільстві є багато небайдужих людей, які готові допомагати безпритульним тваринам, але не завжди мають зручний інструмент для цього.

Останніми роками значного поширення набули мобільні технології та сервіси на основі геолокації, що дозволяють автоматизувати збір і обробку інформації про об'єкти, які потребують уваги. Використання таких технологій у сфері допомоги бездомним тваринам може значно підвищити ефективність їхнього пошуку, реєстрації та прилаштування.

Для ефективного вирішення проблеми безпритульних тварин необхідно забезпечити взаємодію між основними користувачами програми.

Звичайні користувачі – люди, які знаходять безпритульних тварин, бажають допомогти їм або шукають загублених домашніх улюбленців. Вони можуть додавати оголошення, переглядати інформацію про знайдених чи загублених тварин та зв’язуватися з іншими користувачами.

Адміністратори – особи, які забезпечують модерацію контенту, перевіряють скарги на фейкові або некоректні оголошення та слідкують за коректною роботою платформи.

Можливе розширення ролей, наприклад, додавання волонтерів – активних учасників, які допомагають з координацією та пошуком тварин, працюють із притулками та ветеринарними клініками.

Інформаційна система для вирішення цієї проблеми повинна мати зручний інтерфейс для введення даних про знайдених тварин, можливість автоматичного визначення місцезнаходження та інструменти для пошуку інформації за заданими критеріями.

Для реалізації програми, яка допомагає знайти загублених або бездомних тварин, необхідно передбачити кілька основних можливостей.

Потрібно забезпечити реєстрацію та авторизацію користувачів, щоб ідентифікувати осіб, які додають оголошення, і зберігати історію їхніх дій. Реєстрація може відбуватися через email, телефон або соціальні мережі (Google, Facebook), з обов’язковим погодженням з умовами використання.

Користувач повинен мати змогу додати інформацію про тварину: завантажити фото, зазначити характеристики (тип, розмір, особливі прикмети), описати обставини знахідки чи втрати та вказати контактні дані.

Необхідно передбачити функцію визначення місцезнаходження – автоматично через геолокацію або вибором точки на карті вручну, щоб точніше фіксувати місце знахідки чи втрати. Це дозволить спростити процес пошуку та забезпечити актуальність даних. Додатково можна реалізувати історію змін місцезнаходження для тварин, яких бачили в різних місцях, що допоможе волонтерам та власникам ефективніше координувати пошук.

Для зручного пошуку оголошень потрібен механізм фільтрації за типом тварини, місцезнаходженням, часом додавання та іншими характеристиками, такими як колір, порода або стан здоров'я. Дані мають відображатися у вигляді списку та інтерактивної карти з можливістю швидкого перегляду деталей.

Оголошення повинні містити фото, опис, локацію та контактні дані. Корисним буде можливість коментування, оцінки важливості інформації, а також зв’язку з автором через додаток. Додатково можна передбачити функцію підтвердження актуальності оголошень, щоб користувачі могли позначати знайдених або повернених тварин, що допоможе уникнути застарілих записів.

Потрібен механізм модерації контенту, щоб користувачі могли скаржитися на фейкові або некоректні записи, а адміністрація могла їх перевіряти. Автоматизовані алгоритми також можуть допомагати у виявленні потенційних порушень, таких як дублікати або оголошення, що містять неналежний контент.

Завдяки цим функціям мобільний застосунок PetCompas стане ефективним інструментом для допомоги бездомним тваринам та полегшить їхній пошук і прилаштування, забезпечуючи швидку взаємодію між людьми, які готові допомагати.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Першим кроком у проведенні аналізу програмно-технічного забезпечення в сфері допомоги безпритульним тваринам є дослідження існуючих рішень, які вже функціонують у цій галузі. На сьогодні існує низка мобільних додатків і веб-платформ, що спрямовані на допомогу безпритульним тваринам, і вони можуть бути розділені на кілька основних категорій: бази даних загублених і знайдених тварин, сервіси для пошуку тимчасового або постійного притулку, а також волонтерські платформи для збору допомоги.

PetFinder [7] (рисунок 1.1) – одна з найбільш відомих міжнародних платформ, що спеціалізується на пошуку домівки для тварин. Додаток дозволяє користувачам переглядати оголошення про тварин, які потребують нового дому, а також взаємодіяти з притулками та волонтерами. Крім того, PetFinder має функцію створення профілів тварин із фото, описом і контактною інформацією, що допомагає швидко знайти нових господарів для безпритульних тварин.

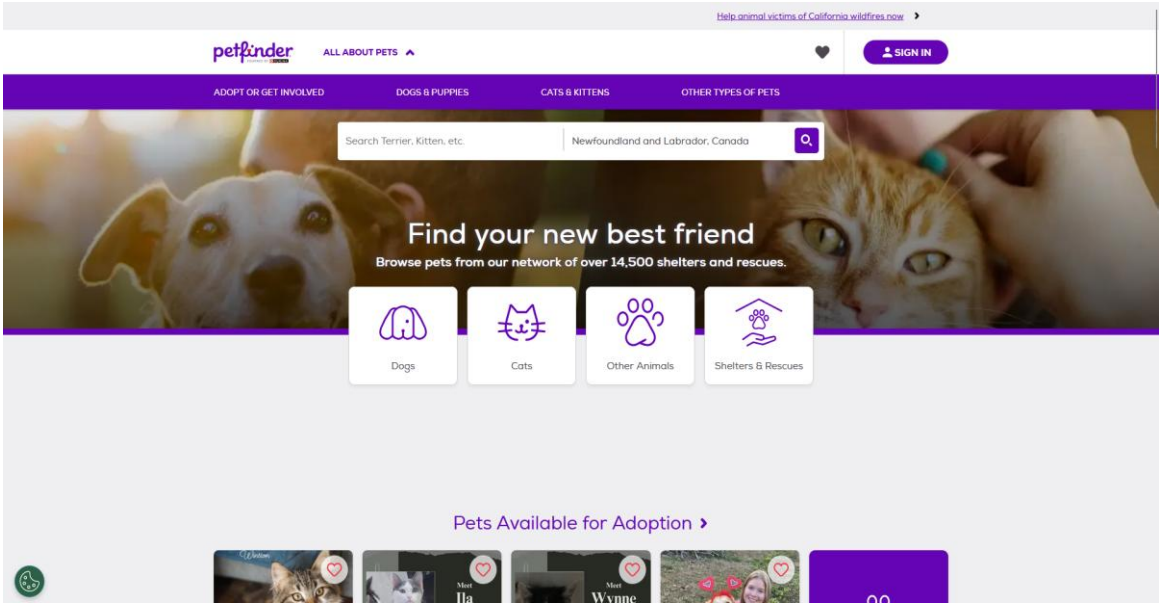


Рисунок 1.1 – PetFinder

FindPet [8] (рисунок 1.2) – мобільний додаток, який зосереджується на поверненні загублених тварин їхнім власникам. Він використовує штучний інтелект для розпізнавання облич тварин на фото, що дає змогу автоматично порівнювати їх із базою даних знайдених або загублених тварин. Користувачі можуть завантажити фото своєї зниклої тварини та отримати сповіщення, якщо буде знайдено схожий збіг.

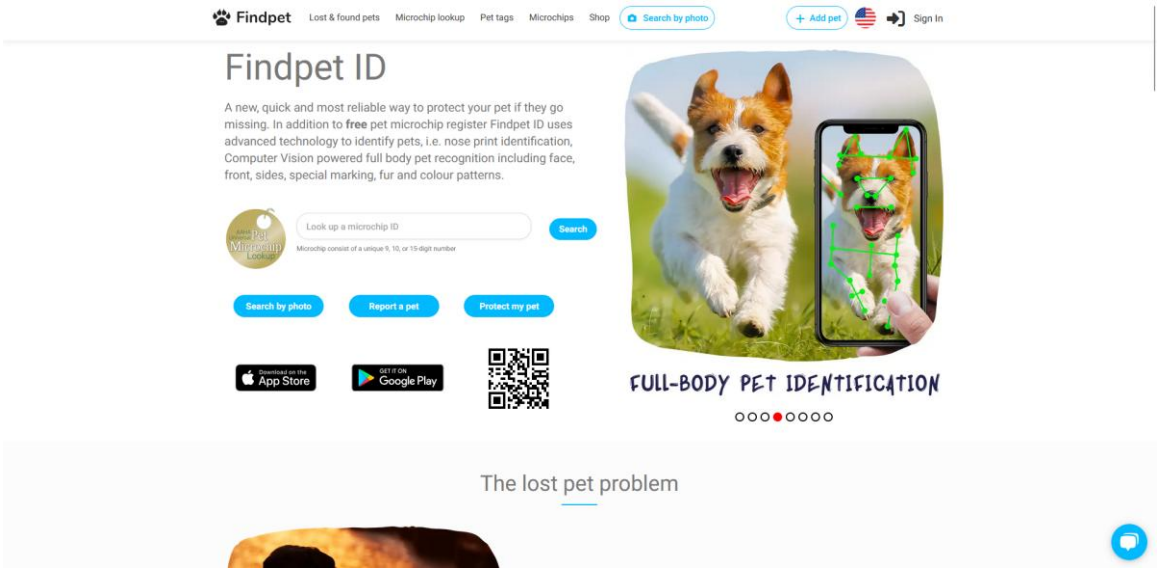


Рисунок 1.2 – PetFinder

PawBoost [9] (рисунок 1.3) – платформа, що спеціалізується на швидкому розповсюдженні інформації про загублених і знайдених тварин. Вона інтегрована з соціальними мережами, що дозволяє оперативно поширювати повідомлення серед місцевих спільнот. Додаток також підтримує функцію геолокації, завдяки якій користувачі можуть переглядати оголошення про знайдених тварин у своєму регіоні.

Adopt A Pet (рисунок 1.4) – сервіс, що об’єднує безліч притулків, волонтерських організацій і приватних осіб, які шукають нових власників для безпритульних тварин. Додаток має зручну систему фільтрації, що дозволяє обирати тварин за видом, віком, породою та іншими характеристиками.

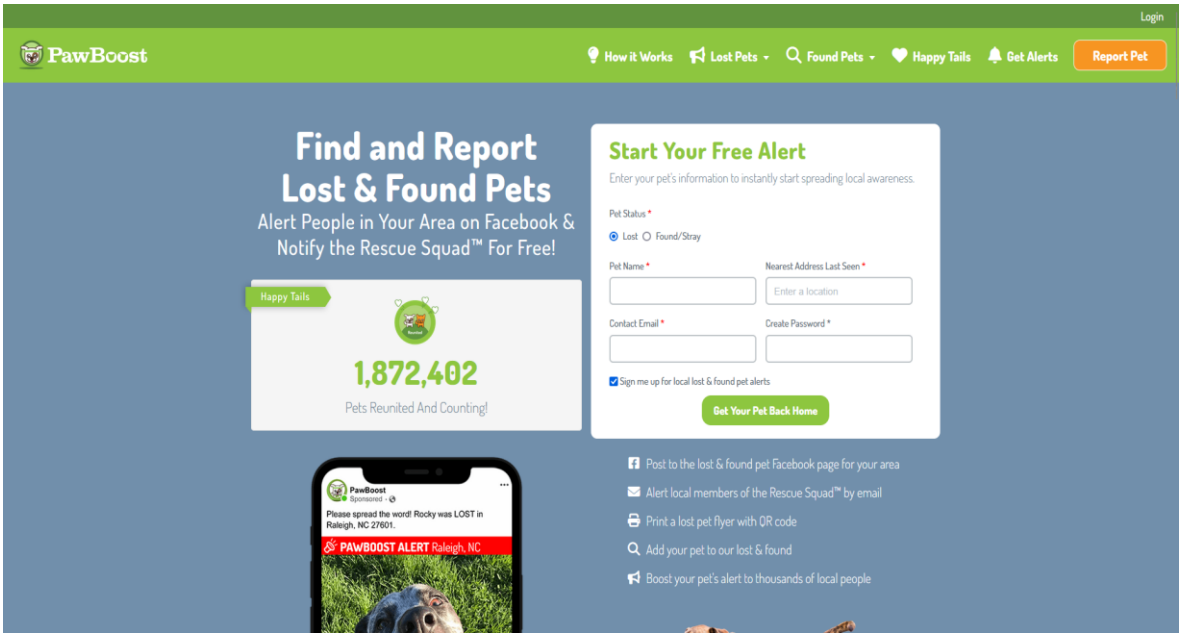


Рисунок 1.3 – PawBoost

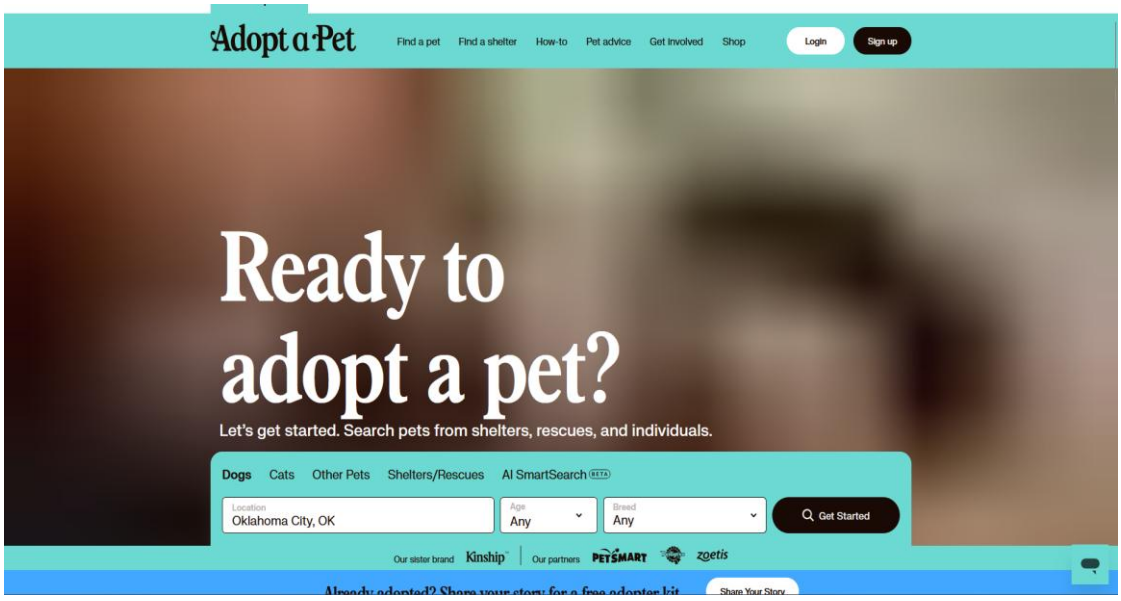


Рисунок 1.4 – Adopt A Pet

Таким чином, аналіз існуючих рішень у сфері допомоги безпритульним тваринам показує, що сучасні мобільні додатки здебільшого орієнтовані або на пошук зниклих тварин, або на прилаштування тварин у нові домівки. Розробка мобільного додатку, який поєднуватиме функції внесення знайденої тварини в базу даних для її подальшого пошуку та допомоги, може стати важливим

внеском у вирішення проблеми безпритульних тварин, оскільки дозволить оперативно об’єднувати зусилля волонтерів, притулків і звичайних користувачів для порятунку та прилаштування тварин.

1.3 Визначення функціональних та не функціональних вимог до програмного забезпечення

Для забезпечення повноцінної роботи мобільного застосунку PetCompas необхідно визначити функціональні вимоги, які відображають ключову поведінку системи з точки зору користувача. Ці вимоги визначають, які дії може виконувати користувач, та які функції повинні бути реалізовані для досягнення цілей проєкту. Вони є основою для подальшого проєктування архітектури, інтерфейсу та логіки взаємодії в системі. Нижче наведено перелік основних функціональних можливостей, реалізація яких забезпечить ефективну роботу додатку.

Функціональні вимоги:

– додавання інформації про знайдених тварин, включаючи фото, опис, стан здоров'я та особливі прикмети. Користувачі повинні мати змогу швидко завантажувати зображення тварини та додавати описові дані, які допоможуть іншим у її пошуку.

– автоматичне визначення геолокації під час завантаження фотографій. Це дозволить уникнути необхідності вручну вводити адресу та зробить систему більш зручною для користувачів.

– інтеграція карти для відображення місць перебування тварин у реальному часі. Всі додані записи повинні автоматично відображатися на інтерактивній карті, що допоможе швидко визначати місцезнаходження тварини.

– механізм реєстрації та авторизації користувачів. Реєстрація може бути реалізована через email або соціальні мережі, а система авторизації повинна забезпечити безпечний доступ до персонального кабінету.

– можливість оновлення або видалення власних записів. Користувачі зможуть редагувати деталі про тварину або видаляти інформацію, якщо тварину вже знайдено або їй надано допомогу.

– пошук тварин за критеріями: місце знаходження, час публікації, тип тварини. Це дозволить швидше знаходити потрібну інформацію та уникати перегляду нерелевантних записів.

– взаємодія користувачів через коментарі або оновлення статусу знайдених тварин. Це дасть змогу координувати зусилля та спрощувати процес надання допомоги.

– інтеграція нейронної мережі [18 - 19] для автоматичного визначення виду тварини за фото. Система повинна використовувати алгоритми машинного навчання, щоб розпізнавати породу або вид тварини, що підвищить ефективність платформи.

Функціональні вимоги спрямовані на забезпечення ефективної взаємодії користувачів із системою та створення інтуїтивно зрозумілого механізму для додавання, пошуку та обробки інформації про безпритульних тварин. Особлива увага приділяється зручності використання додатку, швидкому доступу до ключових функцій та забезпеченню актуальності інформації. Реалізація таких можливостей, як інтеграція карти, автоматичне визначення геолокації та підтримка розпізнавання тварин за допомогою нейронної мережі, значно підвищує корисність системи в реальних умовах.

Функціональні вимоги описують основні дії, які користувач може виконувати в системі, та служать основою для технічного проєктування інтерфейсу і логіки застосунку. Для наочного уявлення можливостей, які має надавати система, ці вимоги можуть бути подані у вигляді діаграми варіантів

використання, що демонструє взаємодію користувача з основними сценаріями використання додатку (рисунок 1.5).

Рисунок 1.5 – Діаграма варіантів використань

Нефункціональні вимоги:

– оптимізація продуктивності для швидкої обробки запитів і мінімального часу відгуку [20]. Серверна частина повинна бути розроблена з урахуванням високої швидкодії та використання кешування для оптимізації завантаження.

– стабільна робота при високих навантаженнях. Платформа повинна витримувати значну кількість одночасних запитів без зниження продуктивності.

– безпечна авторизація користувачів із використанням стандартних протоколів аутентифікації. Використання OAuth, JWT або інших сучасних методів захистить персональні дані користувачів.

– шифрування переданих даних для захисту персональної інформації. Усі передані дані повинні бути захищені за допомогою протоколів HTTPS та шифрування бази даних.

– гнучкість архітектури для можливості розширення функціоналу [11, 28]. Майбутні оновлення системи повинні легко інтегрувати нові можливості без серйозних змін у коді.

– адаптивний дизайн [32, 38] для роботи на різних пристроях та підтримка кількох мов. Інтерфейс повинен бути зручним як для користувачів ПК, так і для власників смартфонів, а також підтримувати кілька мов для охоплення більшої аудиторії.

– механізми моніторингу та логування для оперативного виявлення та усунення проблем. Адміністративна панель повинна включати інструменти для відстеження активності системи та усунення можливих збоїв.

Нефункціональні вимоги визначають якість роботи системи, її продуктивність, безпеку та масштабованість. Виконання цих вимог забезпечить надійну, безпечну та ефективну роботу платформи, що сприятиме кращій координації зусиль у допомозі безпритульним тваринам. Крім того, відповідність сучасним стандартам забезпечить зручність користування платформою та її здатність до подальшого розвитку й адаптації до нових викликів.

1.4 Аналіз сучасних рішень у сфері розпізнавання тварин на фото

Сучасні технології розпізнавання зображень активно використовують нейронні мережі для автоматичної ідентифікації об’єктів на фото, зокрема,

тварин. Аналіз основних методів та архітектур нейронних мереж, застосовуваних у цій сфері, дозволяє оцінити їх ефективність, швидкість роботи та обмеження, що визначає вибір оптимального рішення для мобільного застосунку PetCompas.

Однією з найбільш популярних архітектур є YOLO (You Only Look Once), яка забезпечує швидке та точне виявлення об'єктів у реальному часі [17]. Основна перевага YOLO полягає у тому, що вона обробляє зображення за один прохід через нейромережу, що значно скорочує час виконання порівняно з традиційними підходами, такими як R-CNN або Faster R-CNN. Ця технологія демонструє високу продуктивність у розпізнаванні загальних категорій об’єктів, але може мати труднощі з детальним класифікуванням схожих між собою видів тварин через обмежену здатність до розрізнення дрібних відмінностей у зовнішньому вигляді.

Альтернативним підходом є використання Convolutional Neural Networks (CNN), які глибоко аналізують особливості об’єктів. Мережі типу ResNet, DenseNet та EfficientNet дозволяють досягти високої точності у класифікації зображень [19], оскільки вони використовують глибокі рівні аналізу з детальним виокремленням особливостей. Однак, їх недоліком є значні вимоги до обчислювальних ресурсів, що може ускладнити їх застосування у мобільних пристроях без спеціалізованого апаратного прискорення.

Ще одним ефективним рішенням є нейронні мережі, створені на основі трансформерів, такі як Vision Transformers (ViTs) [18]. Вони використовують механізм самоуваги, що дозволяє краще аналізувати взаємозв’язки між пікселями на зображенні. ViTs демонструють високу точність у розпізнаванні складних зображень, проте поступаються YOLO у швидкості обробки, що може бути критичним фактором при роботі з великими потоками даних у реальному часі.

Окрім вибору архітектури, важливим аспектом є наявність і якість навчального датасету. Для ефективного навчання системи необхідно використовувати розмітку з великою кількістю зображень різних порід і типів тварин у різних умовах освітлення та ракурсах. У цьому контексті можливе використання відкритих датасетів, таких як Animal Faces-HQ (AFHQ), Stanford Dogs Dataset, або ж створення власного набору даних шляхом краудсорсингу через мобільний застосунок. Це дозволить адаптувати модель до реальних умов, у яких користувачі будуть фотографувати тварин.

Крім того, сучасні фреймворки, такі як TensorFlow Lite, ONNX та CoreML, дозволяють оптимізувати великі нейронні мережі для використання на мобільних пристроях, зменшуючи їх розмір і прискорюючи інференс без значної втрати точності. Завдяки цьому розпізнавання можна виконувати прямо на пристрої користувача, зменшуючи залежність від інтернет-з'єднання, зберігаючи конфіденційність зображень та покращуючи швидкодію.

Враховуючи особливості мобільного застосунку, важливо знайти баланс між швидкістю та точністю розпізнавання тварин. YOLO є привабливим варіантом завдяки високій продуктивності та можливості використання на мобільних пристроях із залученням оптимізованих версій, таких як YOLOv5, YOLOv8 чи YOLO-NAS. Проте, у випадку необхідності більш точної ідентифікації окремих видів тварин, доцільно розглянути гібридний підхід — використання YOLO для виявлення тварини на фото, а потім застосування CNN або ViT для уточнення її виду або породи.

Таким чином, сучасні нейронні мережі пропонують широкий спектр рішень для розпізнавання тварин на фото. Вибір конкретної архітектури залежить від вимог до швидкості, точності та доступних обчислювальних ресурсів. Оптимальним варіантом для мобільного застосунку є використання YOLO як базової моделі для швидкого виявлення тварин із можливістю інтеграції більш точних алгоритмів класифікації у разі потреби. Такий підхід

забезпечує не лише ефективність, але й масштабованість рішення у майбутньому.

1.5 Висновки і постановка задачі

Проблема, яка стоїть перед суспільством, полягає у відсутності зручного, надійного та технологічно розвиненого інструменту для фіксації та пошуку безпритульних або загублених тварин. Існуючі сервіси часто мають вузьку функціональність, не інтегрують дані в реальному часі, не підтримують геолокацію або автоматичне розпізнавання тварин, що значно знижує їхню ефективність. Через це процес взаємодії між тими, хто знаходить тварин, та тими, хто їх шукає або готовий допомогти, стає тривалим, неефективним і часто безрезультатним. Особливо складним є швидкий пошук інформації про тварину в конкретному районі, можливість зв'язку з іншими користувачами та підтвердження актуальності вже наявних оголошень.

Крім того, значна частина інформації про безпритульних тварин поширюється через соціальні мережі у вигляді дописів або історій, які швидко губляться в загальному потоці контенту. Такий підхід ускладнює систематизацію даних, обмежує можливості для фільтрації, аналітики та подальшого реагування. У результаті багато корисних ініціатив залишаються фрагментованими, не координуються між собою і не досягають максимального потенціалу у вирішенні проблеми. Наявність централізованої платформи, яка спеціалізується саме на цій сфері, могла б суттєво підвищити ефективність зусиль волонтерів, притулків та небайдужих громадян.

Завданням дипломного проєкту є створення мобільного застосунку PetCompas, який усуває ці недоліки та забезпечує єдиний цифровий простір для реєстрації, пошуку й допомоги безпритульним тваринам. Основна ідея полягає в тому, щоб користувач міг швидко зробити фото тварини, вказати її опис і

автоматично передати місце фіксації до загальної бази. Візуалізація даних на інтерактивній карті дозволяє оперативно знаходити найближчих тварин, переглядати деталі, контактувати з автором оголошення та об’єднувати зусилля. Застосунок повинен мати простий та інтуїтивно зрозумілий інтерфейс, бути доступним для широкої аудиторії та підтримувати кросплатформенність.

Важливим аспектом є соціальна складова застосунку. Окрім основної функціональності, платформа має сприяти формуванню відповідального ставлення до тварин, залученню нових учасників до волонтерської діяльності та налагодженню співпраці між громадськістю, притулками, зоозахисними організаціями та органами місцевого самоврядування. Інтеграція можливостей комунікації, обміну інформацією, оцінки важливості повідомлень і верифікації записів значно підвищить рівень довіри до сервісу й розширить його функціональність у майбутньому.

З технічної точки зору розробка включає реалізацію клієнт-серверної архітектури, інтеграцію картографічних сервісів та геолокації, використання нейронної мережі для розпізнавання виду тварини на фото, а також створення захищеного API для взаємодії з базою даних. Особлива увага приділяється швидкодії серверної частини, стабільності при високих навантаженнях, безпеці обміну даними та захисту персональної інформації. У перспективі можлива інтеграція з іншими платформами, такими як соціальні мережі, міські реєстри тварин або ветеринарні системи обліку.

Кінцевою метою проєкту є створення стабільного, ефективного та корисного мобільного застосунку, який зробить допомогу безпритульним тваринам простішою, швидшою та доступнішою для кожного небайдужого користувача. Такий застосунок сприятиме підвищенню ефективності взаємодії між волонтерами, притулками, ветеринарами та громадськістю, формуючи єдину цифрову екосистему для вирішення проблеми безпритульності тварин.

2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз та вибір архітектури системи

Для розробки мобільного застосунку для допомоги безпритульним тваринам необхідно вибрати відповідну архітектуру програмного забезпечення. Основним критерієм при виборі є необхідність ефективної взаємодії між клієнтською та серверною частинами, масштабованість та можливість розширення функціональності в майбутньому. Крім того, система повинна забезпечувати високу продуктивність, зручність використання та надійність у роботі. З огляду на ці вимоги, було обрано модульну клієнт-серверну архітектуру (рисунок 2.1), яка є стандартним рішенням для подібних систем [20].

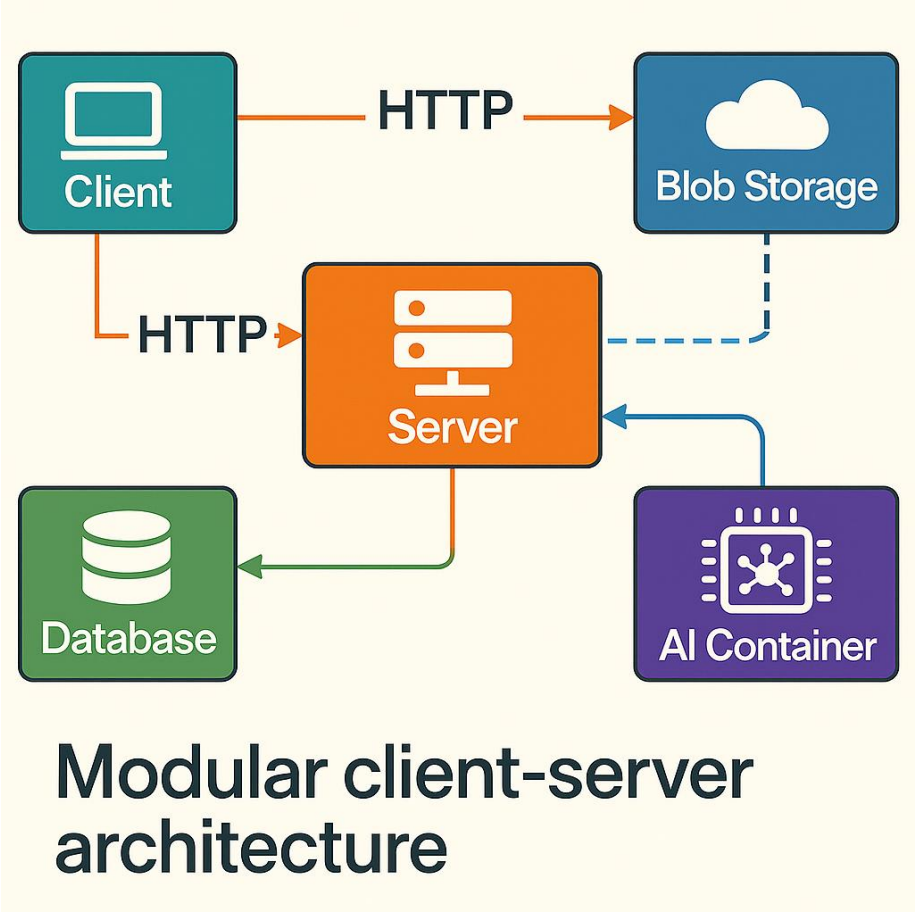


Рисунок 2.1 – Модульна клієнт-серверна архітектура

Клієнт-серверна архітектура передбачає розподіл обов'язків між двома основними компонентами: сервером (back-end) та клієнтом (front-end). Такий підхід дозволяє забезпечити централізоване управління даними, гнучкість у зміні користувацького інтерфейсу та підтримку різних платформ. Завдяки розподіленому підходу, система може легко адаптуватися до зростання навантаження та збільшення кількості користувачів.

Серверна частина (Back-end) – це головний компонент, який відповідає за обробку даних, бізнес-логіку та збереження інформації в базі даних. Він отримує запити від клієнтських пристроїв, обробляє їх та повертає відповідні результати. Сервер також забезпечує безпеку даних, обмежуючи доступ до інформації відповідно до прав користувачів. Крім того, серверна частина виконує автоматичну перевірку введених даних, обробку зображень та керування сповіщеннями для користувачів. Для взаємодії клієнта з сервером використовується API, що забезпечує стандартизований обмін даними у форматі JSON або XML [22].

Клієнтська частина (Front-end) – це мобільний застосунок, через який користувач взаємодіє із системою. Він надає зручний інтерфейс для внесення інформації про знайдених тварин, перегляду існуючих записів та взаємодії з картою, на якій відображаються знайдені тварини. Клієнтська частина надсилає запити на сервер та отримує від нього відповідь у вигляді даних, які відображаються у відповідному форматі. Крім того, застосунок може працювати в офлайн-режимі, зберігаючи тимчасові дані локально та синхронізуючи їх із сервером при підключенні до інтернету.

Вибір клієнт-серверної архітектури обумовлений наступними перевагами:

– централізоване зберігання даних. Всі дані зберігаються на сервері, що забезпечує їхню узгодженість, безпеку та можливість резервного копіювання.

– масштабованість. Архітектура дозволяє легко додавати нові функції або змінювати існуючі без значного втручання в роботу всієї системи.

– можливість доступу з різних пристроїв. Оскільки клієнт взаємодіє з сервером через API, можлива розробка не лише мобільного застосунку, але й веб-версії або інтеграція з іншими сервісами.

– безпека. Серверна частина може забезпечувати автентифікацію та авторизацію користувачів, контролювати доступ до даних та захищати їх від несанкціонованого доступу.

Для взаємодії між клієнтом та сервером використовується REST API [22], що забезпечує стандартизований обмін даними через HTTP. REST (Representational State Transfer) є популярним архітектурним стилем, який забезпечує гнучкість, масштабованість і зручність у використанні. Основні принципи REST API включають:

– використання стандартних HTTP-методів (GET, POST, PUT, DELETE) для обробки запитів.

– представлення ресурсів у вигляді унікальних URL-адрес.

– передачу даних у форматі JSON або XML для зручності інтеграції.

– слабке зв’язування між клієнтом і сервером, що забезпечує незалежний розвиток обох компонентів.

– підтримку кешування для підвищення продуктивності та зниження навантаження на сервер.

– щоб покращити структуру серверної частини, у розробці використовується Clean Architecture [11, 37]. Цей підхід дозволяє створити підтримуваний, розширюваний та тестований код. Основні принципи Clean Architecture включають:

– розділення логіки за рівнями: поділ додатка на шари (доменний, застосунковий, інфраструктурний, презентаційний).

– незалежність від фреймворків: система не прив’язана до конкретної бібліотеки або технології, що спрощує зміну залежностей.

– мінімальна залежність від бази даних: бізнес-логіка не має бути прив’язана до конкретної бази даних, що дозволяє легко змінювати джерела даних.

– тестованість: модульний підхід спрощує написання тестів для кожного рівня системи.

– гнучкість у розширенні: можливість легкого додавання нових функціональних модулів без значних змін у базовій структурі.

Завдяки використанню REST API та Clean Architecture, система набуває високої підтримуваності, масштабованості та гнучкості у розширенні функціональності в майбутньому [11, 22].

2.2 Опис структури даних та моделі бази даних

У системі буде використовуватися реляційна база даних, яка містить кілька основних сутностей [21], необхідних для функціонування додатка. Основними сутностями є User, Reports, Location, Animals, AnimalPhoto, Notification, Comment та Role. Взаємозв’язки між цими сутностями організовані таким чином, щоб забезпечити ефективне зберігання та обробку інформації [21, 20].

Сутність User представляє зареєстрованих користувачів додатка. Вона містить базові дані про користувача, такі як ім’я, електронна пошта та пароль. Користувач може створювати звіти про знайдених безпритульних тварин, які зберігаються в сутності Reports. Крім того, кожен користувач має певну роль у системі, що визначає його права та можливості взаємодії з додатком.

Сутність Role використовується для управління ролями користувачів у системі. Вона містить перелік можливих ролей, таких як "звичайний користувач", "волонтер" або "адміністратор". Зв’язок між User та Role дозволяє кожному користувачу мати певні дозволи, наприклад, адміністраторам може

бути надано право редагувати або видаляти звіти, а волонтерам — отримувати додаткові сповіщення про нові випадки.

Сутність Reports використовується для збереження інформації про знайдених тварин. Кожен запис у цій таблиці містить дані про тварину, її фотографії та місцезнаходження. Поле UserId забезпечує зв’язок між звітом і користувачем, який його створив. Додатково передбачено можливість оновлення статусу звіту, що дозволяє позначати знайдених тварин як прилаштованих або тих, що потребують допомоги.

Інформація про місце, де була знайдена тварина, зберігається в сутності Location. Ця сутність містить координати місцезнаходження та може бути пов’язана з кількома звітами. Такий підхід дозволяє уникнути дублювання даних у випадках, коли декілька користувачів повідомляють про одну й ту саму тварину в певному місці.

Сутність Animals описує характеристики знайдених тварин, включаючи їхній тип, розмір та інші атрибути. Вона має зв’язок із сутністю Reports, що дозволяє зберігати окремі описи для кожного випадку.

Для збереження фотографій тварин використовується сутність AnimalPhoto. Вона містить посилання на зображення та пов’язана із сутністю Reports, що дає можливість прикріплювати кілька фотографій до одного звіту. Це полегшує ідентифікацію тварин та підвищує ефективність роботи додатка.

Сутність Notification використовується для збереження push-сповіщень, які надсилаються користувачам при оновленні статусу їхніх звітів або появі нових повідомлень. Вона містить інформацію про тип сповіщення, користувача, якому воно адресоване, та статус прочитання. Це дозволяє оперативно інформувати користувачів про зміни у звітах та нові коментарі.

Сутність Comment дозволяє користувачам залишати коментарі до звітів. Вона містить текст повідомлення, ідентифікатор автора та дату створення. Ця

функціональність сприяє взаємодії між користувачами та допомагає швидше знаходити рішення для безпритульних тварин.

Для відображення структури бази даних та взаємозв'язків між сутностями використовується ER-діаграма (рисунок 2.2), яка візуально ілюструє зв'язки між основними таблицями, їхні атрибути, а також типи зв'язків (один-до-одного, один-до-багатьох тощо). Така модель дозволяє чітко визначити логіку зберігання даних, уникнути дублювання інформації та забезпечити цілісність даних. ER-діаграма є важливим інструментом [20, 21] як на етапі проєктування, так і при подальшій реалізації та супроводі системи, оскільки дозволяє зрозуміти, як саме взаємодіють об'єкти в базі.

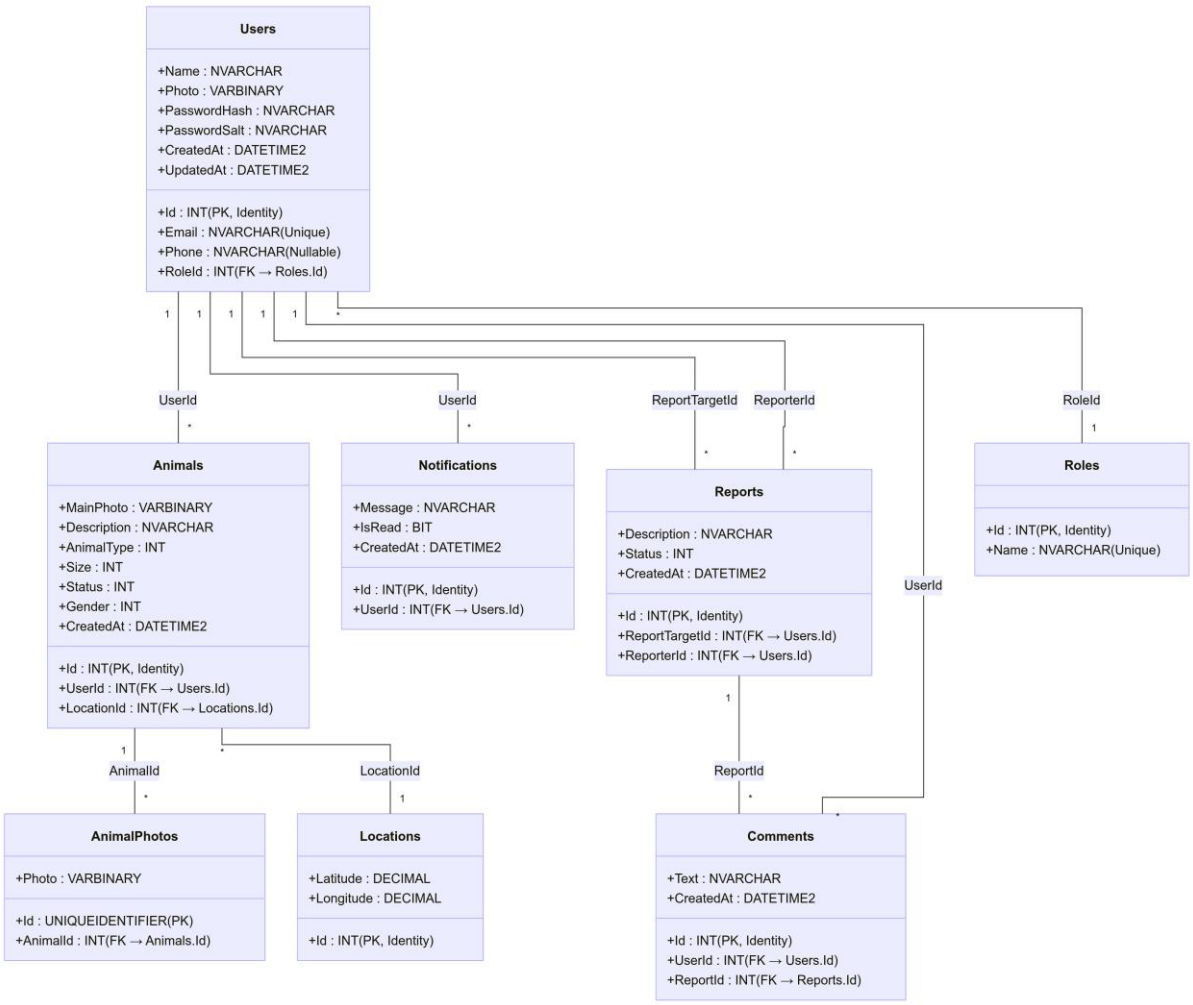


Рисунок 2.2 – ER діаграма

2.3 Проектування серверної частини системи

Серверна частина застосунку PetCompas буде реалізована на основі архітектури клієнт-сервер, що дозволить мобільному клієнту взаємодіяти з центральним сервером через API. Вона планується до побудови з використанням сучасних технологій для забезпечення масштабованості, безпеки та ефективності.

Серверна частина буде базуватись на ASP.NET WebAPI,[5] який є потужним фреймворком для створення веб-сервісів. Цей вибір зумовлений його високою продуктивністю, підтримкою RESTful API та гнучкістю у розробці. WebAPI дозволить створювати HTTP-контролери, що оброблятимуть запити від мобільного клієнта та повертатимуть необхідні дані у форматі JSON. Планується використання гнучкої маршрутизації ASP.NET WebAPI, що дасть змогу створювати як прості, так і складні ендпоінти з параметрами запиту. Крім того, передбачається інтеграція з іншими технологіями .NET, зокрема Entity Framework для роботи з базою даних, MediatR для організації запитів та AutoMapper для мапінгу об'єктів. Також передбачається використання Middleware для додаткової обробки запитів і відповідей, що забезпечить розширюваність і гнучку кастомізацію процесу обробки HTTP-запитів.

Для взаємодії з базою даних планується використання ORM Entity Framework [6], що забезпечить зручну роботу з об'єктами без необхідності написання складних SQL-запитів. Очікуваними перевагами є автоматичне відстеження змін у моделях, підтримка міграцій та можливість використання LINQ-запитів.

Для безпечної аутентифікації та авторизації користувачів передбачається використання JSON Web Token (JWT) [13]. Цей метод дозволить безпечно передавати дані між клієнтом і сервером, а також зберігати інформацію про користувача без необхідності використання сесій.

Для мапінгу моделей (DTO та сутностей бази даних) планується застосування бібліотеки AutoMapper [12], що дасть змогу зменшити кількість ручного коду та забезпечити автоматичне перетворення даних між різними рівнями програми.

У проєкті буде використовуватись бібліотека MediatR [4], яка реалізує патерн «Посередник» (Mediator). Цей підхід дозволить зменшити зв'язаність між компонентами системи, передаючи запити між об'єктами через посередника, а не напряму. Це покращить підтримуваність коду та дозволить легше додавати новий функціонал. У MediatR запити представлятимуться у вигляді команд або запитів (Queries, Commands), а їх обробка здійснюватиметься окремими хендлерами, що дозволить чітко розділяти відповідальності.

Для обробки результатів виконання операцій планується використання патерну Result, реалізованого через тип Either із бібліотеки LanguageExt [29, 30]. Він дозволить обробляти як успішні, так і помилкові результати в єдиній структурі, що підвищить надійність коду та спростить обробку помилок. Наприклад, повернення Either<Error, LanguageExt.Unit> при виконанні команди забезпечить обов’язкову обробку всіх можливих результатів.

Структура проєкту:

– проєкт PetCompas є головним проєктом, який міститиме точку входу в застосунок. У ньому передбачено розміщення контролерів, які оброблятимуть HTTP-запити, конфігураційних файлів (наприклад, appsettings.json), а також налаштувань WebAPI.

– проєкт PetCompas.Domain міститиме всі основні моделі, інтерфейси, DTO та репозиторії. Тут буде визначено структуру даних, що використовуватиметься в усій системі. Наприклад, у папці Models будуть розташовані класи, що представлятимуть об'єкти бази даних, а в Repositories — інтерфейси репозиторіїв.

– проєкт PetCompas.Handlers включатиме всі обробники команд і запитів, що застосовуватимуться у MediatR. Вони будуть згруповані у відповідні папки (наприклад, Animal, User), що дозволить зручно організувати логіку обробки запитів.

– проєкт PetCompas.Infrastructure міститиме всі сервіси, репозиторії та утиліти. Тут передбачається розміщення реалізацій інтерфейсів із Domain. У цьому проєкті будуть класи для роботи з базою даних (Repositories), сервіси (Services), JWT-токени (JwtTokenService.cs), а також утиліти (Utils).

Серверна частина застосунку PetCompas проєктується відповідно до принципів багаторівневої архітектури [11, 37], що забезпечить чітке розділення логіки, зручність у розширенні та підтримці. Такий підхід дозволить розмежувати відповідальність між різними частинами системи, що, своєю чергою, підвищить гнучкість у розробці та полегшить внесення змін у майбутньому. Кожен рівень системи відповідатиме за свій набір завдань, що сприятиме кращій структурованості та повторному використанню коду. Передбачене використання сучасного інструментарію, такого як ASP.NET WebAPI для створення RESTful API, Entity Framework як об'єктно-реляційного мапера для роботи з базою даних, MediatR для впровадження патерну "Посередник", а також JWT для забезпечення безпечної аутентифікації та авторизації. Сукупність цих технологій дозволить досягти високого рівня надійності, масштабованості та безпеки системи, а також забезпечить легкість у тестуванні, обслуговуванні й розширенні функціоналу.

Для кращого розуміння роботи серверної частини застосунку PetCompas розглянемо два ключові процеси: реєстрацію користувача та додавання тварини. Це дозволить продемонструвати повний потік даних, який проходить через усі рівні архітектури — від моменту отримання запиту до його обробки, перевірки, трансформації та збереження в базі даних, з наступним формуванням відповіді для клієнта.

Реєстрація нового користувача (рисунок 2.4) починається з надсилання HTTP-запиту на відповідний ендпоінт контролера, який приймає та обробляє вхідні дані. Після успішної валідації контролер передає запит у вигляді команди до MediatR. MediatR перенаправляє команду до відповідного хендлера, в якому реалізовано логіку обробки запиту. Хендлер викликає метод сервісу UserService, що відповідає за перевірку існування користувача з таким email у базі даних. Якщо користувач не знайдений, сервіс створює хеш паролю за допомогою спеціального механізму PasswordHasher, після чого на основі отриманих даних формується новий об'єкт користувача. Цей об'єкт передається до відповідного репозиторію, де здійснюється його збереження до бази даних через механізми Entity Framework. Після успішного збереження інформації система формує відповідь, яка повертається клієнту, сповіщаючи про успішну реєстрацію.

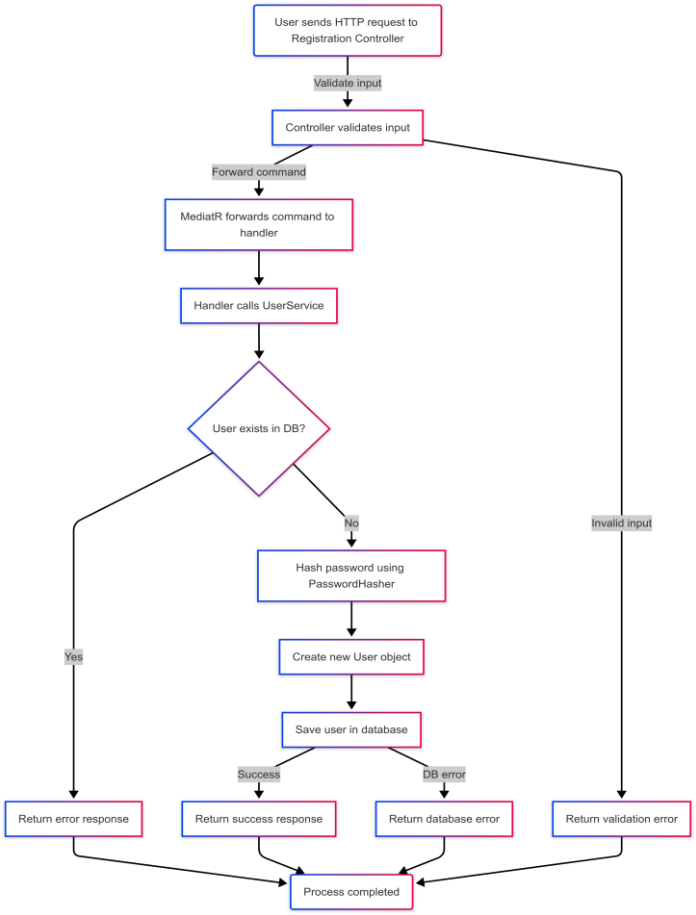


Рисунок 2.4 – Блок-схема процесу реєстрації

Коли користувач хоче додати інформацію про знайдену тварину (рисунок 2.5), він надсилає запит з фото, описом та місцеположенням до відповідного контролера. Контролер отримує запит і передає його у вигляді команди через MediatR до хендлера AddAnimalHandler. Хендлер звертається до AnimalService, який перед тим, як створювати новий запис у базі даних, використовує нейронну мережу для перевірки, чи дійсно на фото присутня тварина [17]. Якщо перевірка успішна, створюється новий об'єкт тварини, який передається до репозиторію та додається в базу даних. У разі успішного завершення операції користувач отримує підтвердження про додавання тварини.

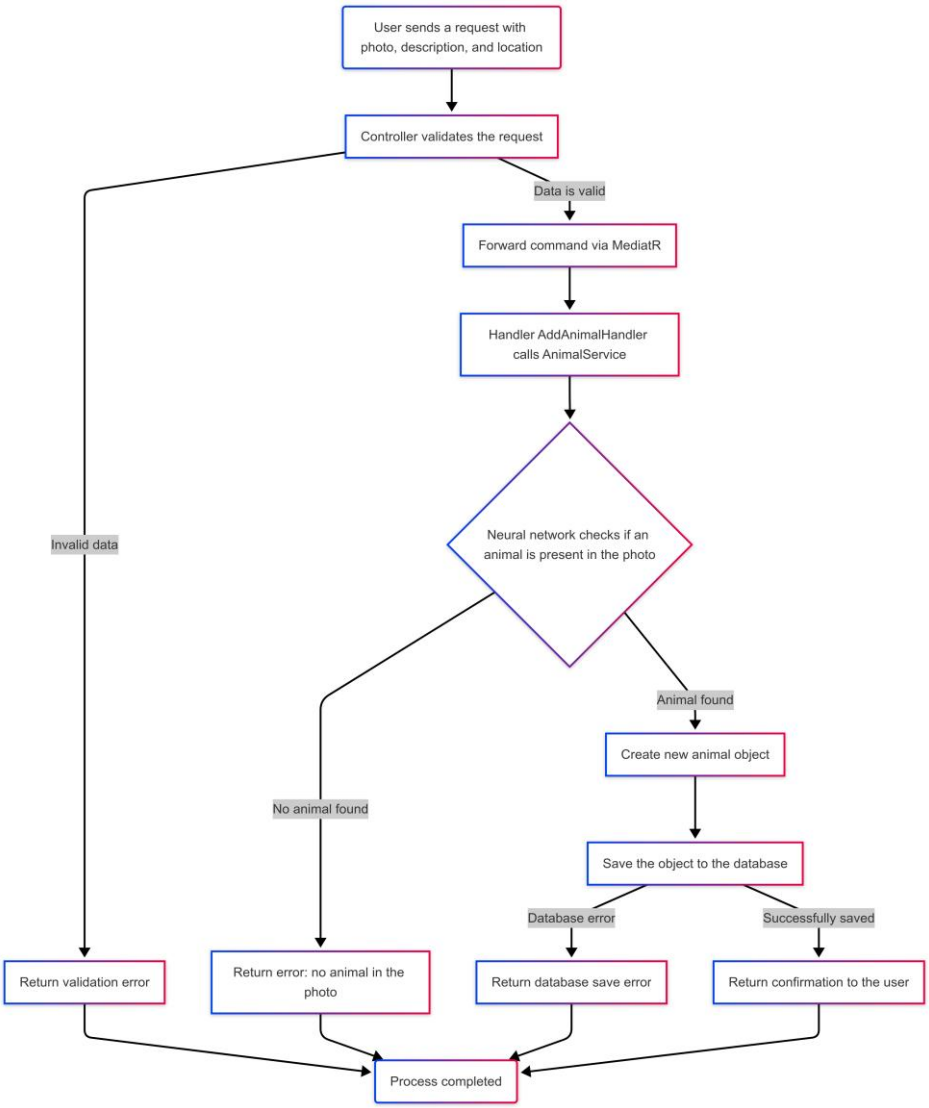


Рисунок 2.5 – Блок-схема процесу додавання тварини

Описані процеси ілюструють, як різні компоненти серверної частини взаємодіють між собою, забезпечуючи обробку запитів та збереження даних. Використання MediatR допомагає чітко розмежувати логіку запитів та їх обробку, а інтеграція нейронної мережі у процес додавання тварини покращує якість даних, що зберігаються у системі.

2.4 Проектування клієнтської частини системи

Клієнтська частина застосунку PetCompass буде реалізована за допомогою .NET MAUI [3], що дозволяє створювати кросплатформені мобільні додатки для Android та iOS на основі єдиного коду. Використання MAUI забезпечує гнучкість у розробці, високу продуктивність та зручну інтеграцію з іншими компонентами .NET екосистеми.

Проєкт клієнтської частини організовано відповідно до принципів MVVM (Model-View-ViewModel) [16], що розділяє логіку представлення (View), модель даних (Model) та логіку взаємодії (ViewModel). Це дозволяє зберігати чистоту коду, полегшує тестування та підтримку додатка.

У структурі проєкту обов’язково будуть наступні основні папки: Converters, Core, Platforms, Services, Views і Shared.

Папка Converters містить клас перетворювачів даних, що допомагає адаптувати інформацію для відображення у UI. Core містить основні утиліти, такі як Result.cs для обробки успіхів і помилок, а також ViewModelBase.cs, який є базовим класом для всіх ViewModel у застосунку. Platforms містить специфічні конфігурації для різних операційних систем, забезпечуючи належну роботу застосунку на різних пристроях.

Services включає в себе сервіси для роботи з геолокацією, навігацією, мережею та збереженням налаштувань користувача. GeolocationService відповідає за отримання поточного місця користувача [15], що критично

важливо для визначення координат знайдених тварин. NavigationService використовується для переміщення між сторінками застосунку. NetworkService забезпечує взаємодію з серверною частиною через HTTP-запити. SettingServices зберігає налаштування додатка.

Основна частина інтерфейсу користувача зосереджена у папці Views, яка містить дві ключові групи: Account і Master. У папці Account знаходяться екрани для управління обліковим записом, зокрема SignIn (вхід у систему) та SignUp (реєстрація нового користувача). Папка Master включає основні функціональні вікна програми, серед яких:

– AddAnimal – дозволяє користувачеві додати нову тварину, зробивши фото та завантаживши його разом з координатами місця знаходження;

– AddAnimalForm – форма для введення інформації про знайдену тварину;

– PhotoCapture – екран для фотографування тварини перед завантаженням;

– PhotoPreview – попередній перегляд зробленого фото перед надсиланням.

Також у Views містяться інші секції, такі як Details (перегляд детальної інформації про знайдену тварину), Main (головний екран додатка), Map (екран з інтегрованою картою для пошуку тварин), Shell (базова оболонка застосунку) та User (профіль користувача).

Файл App.xaml та його код App.xaml.cs відповідають за початкову конфігурацію застосунку, зокрема налаштування глобальної навігації та стилів. Файл MauiProgram.cs містить реєстрацію сервісів, необхідних для коректної роботи програми, включаючи залежності для використання DI (Dependency Injection) [24].

У проєкті буде використаний патерн Result, який є ефективним підходом до обробки результатів [29] виконання операцій у застосунку. Його основна ідея полягає в тому, щоб уникати використання винятків (exceptions) як засобу управління потоком логіки, натомість повертати результат операції у вигляді об’єкта, який містить або успішне значення, або інформацію про помилку.

Такий підхід дозволяє зробити код більш передбачуваним, читабельним та контрольованим, особливо в складних сценаріях взаємодії з мережею або обробки зображень.

Застосування Result також сприяє зниженню кількості неочікуваних збоїв у роботі застосунку та покращує UX (користувацький досвід), адже замість краху програми користувач отримає зрозуміле повідомлення з описом проблеми та варіантом її вирішення. Крім того, це дозволяє централізовано обробляти помилки на рівні UI без дублювання коду перевірок у кожному окремому сценарії.

Щодо навігації, вона реалізується через спеціалізований сервіс NavigationService, який базується на стековому підході керування екранами. У застосунку використовується enum ViewType, який містить перелік усіх доступних екранів (View), таких як головна сторінка, мапа, форма додавання тварини тощо. Під час переходу на новий екран передається значення типу ViewType, після чого сервіс створює та ініціалізує відповідне представлення. Це дозволяє централізовано керувати переходами між екранами, спрощує підтримку коду та полегшує модульне тестування окремих елементів інтерфейсу.

Важливою архітектурною особливістю навігації є реалізація оболонок Shell як ContentPage [23], а всіх екранів — як ContentView. У системі передбачено дві основні оболонки: AccountShell, яка відповідає за навігацію, пов’язану з автентифікацією та профілем користувача, та MasterShell, що охоплює функціонал основної частини застосунку. Такий підхід дозволяє відокремити логіку входу в систему від основного функціоналу, зберігаючи при цьому єдину навігаційну структуру. ContentView динамічно вставляються в ContentPage, залежно від поточної дії користувача, що дає змогу гнучко управляти відображенням вмісту без дублювання сторінок. Це також значно

спрощує реалізацію адаптивного дизайну та повторне використання компонентів між різними частинами інтерфейсу.

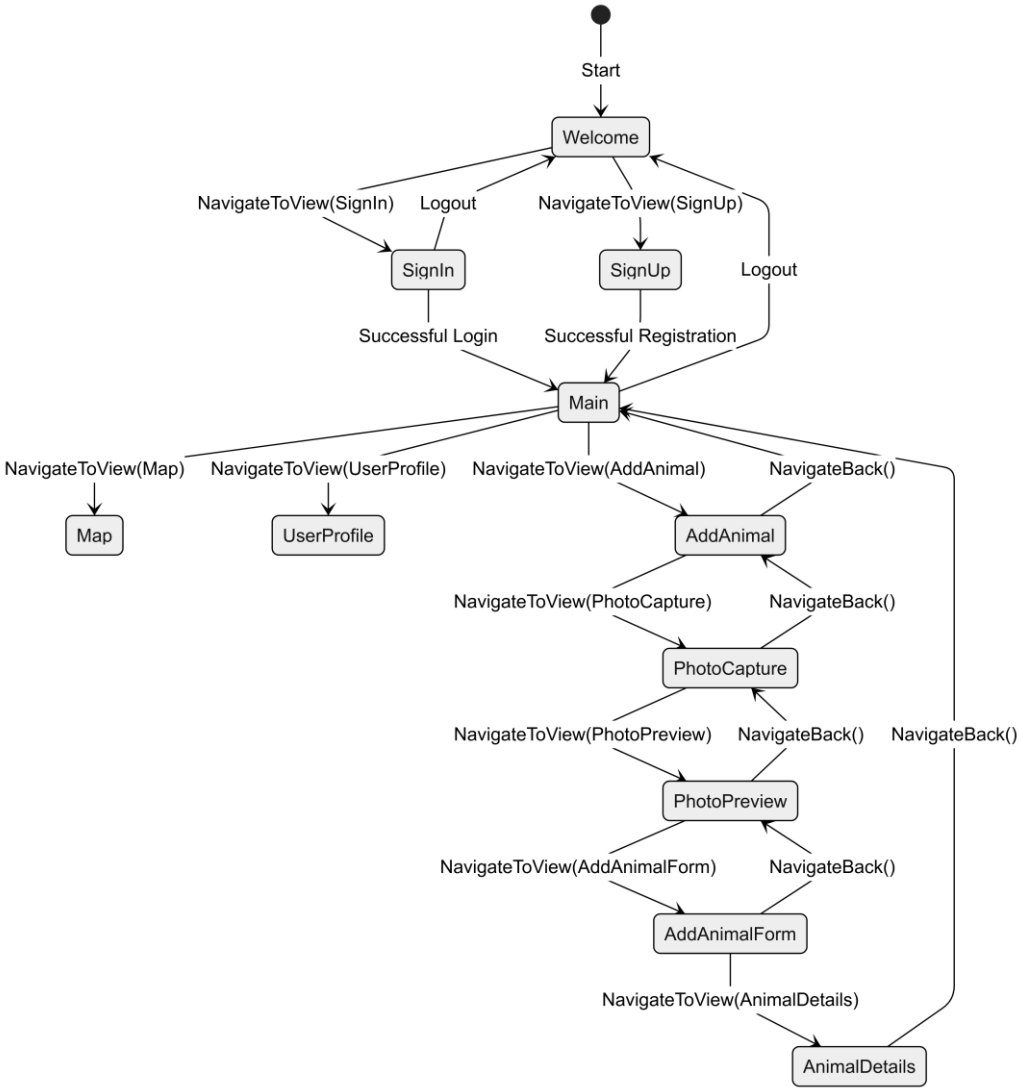


Рисунок 2.7 – Діаграма станів навігації додатку

Для додавання нової тварини в застосунку передбачено можливість роботи з камерою та галереєю. Основне фото є обов’язковим, тоді як додаткові зображення – опціональні. Для роботи з камерою використовується бібліотека Camera.MAUI [35], що дозволяє робити знімки безпосередньо в додатку. Також передбачена можливість вибору фото з галереї. Завантажені зображення

проходять перевірку на якість і відповідність вимогам перед їхнім збереженням у системі.

Розробка клієнтської частини застосунку PetCompass на MAUI дозволяє досягти високої ефективності, зручності використання та кросплатформеності. Завдяки розподілу функціональності між різними модулями забезпечується зручна підтримка та масштабованість системи, що дозволяє легко додавати нові можливості в майбутньому.

2.5 Створення макету мобільного застосунка та дизайн

Дизайн мобільного застосунку Pet Compass орієнтований на зручність та естетику, поєднуючи теплі природні кольори з сучасним мінімалістичним підходом [38]. Основний фон виконаний у світлому кремовому відтінку (#FEFAE0), що створює комфортне сприйняття інтерфейсу. Для підкреслення важливих елементів використовується теплий помаранчевий колір (#FFDEAF) у поєднанні з глибоким коричневим (#BC6C25), який додає контрасту. Кнопки та інші інтерактивні елементи оформлені у темно-зелених тонах (#606C38, #283618), що асоціюються з природою та гармонійно поєднуються з загальним стилем. Текстові елементи представлені у темно-сірому кольорі (#2C2C2C), що забезпечує гарну читабельність на світлому фоні. Всю кольорову схему можна побачити на рисунку 2.8.

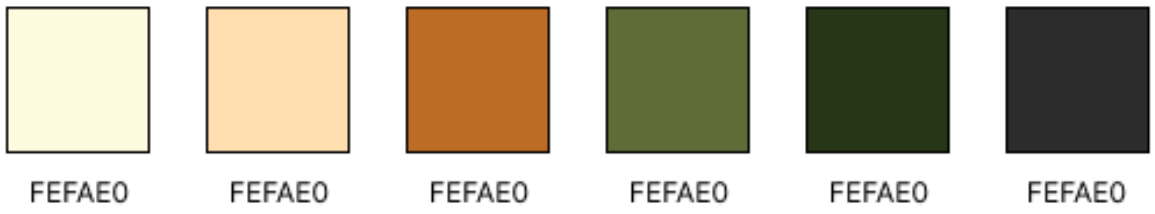
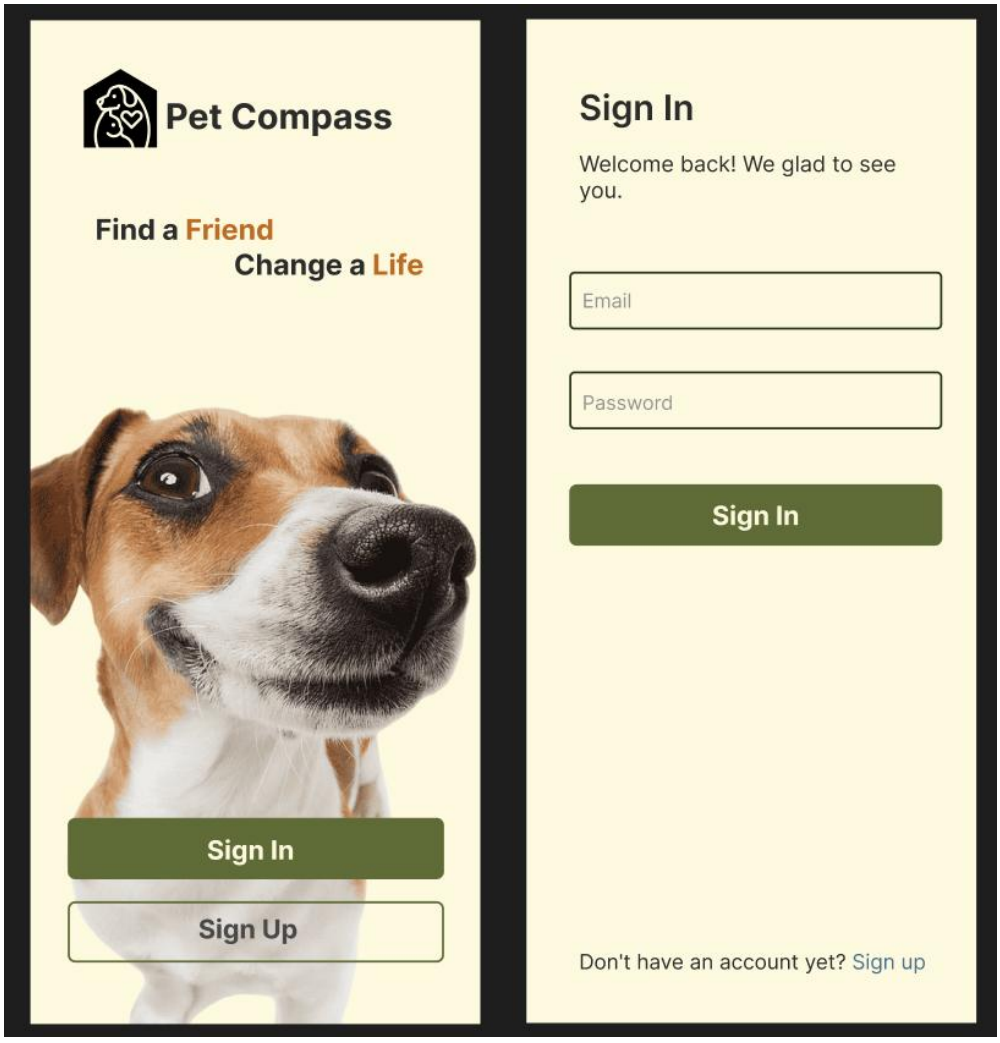


Рисунок 2.8 – Кольорова схема додатку PetCompas

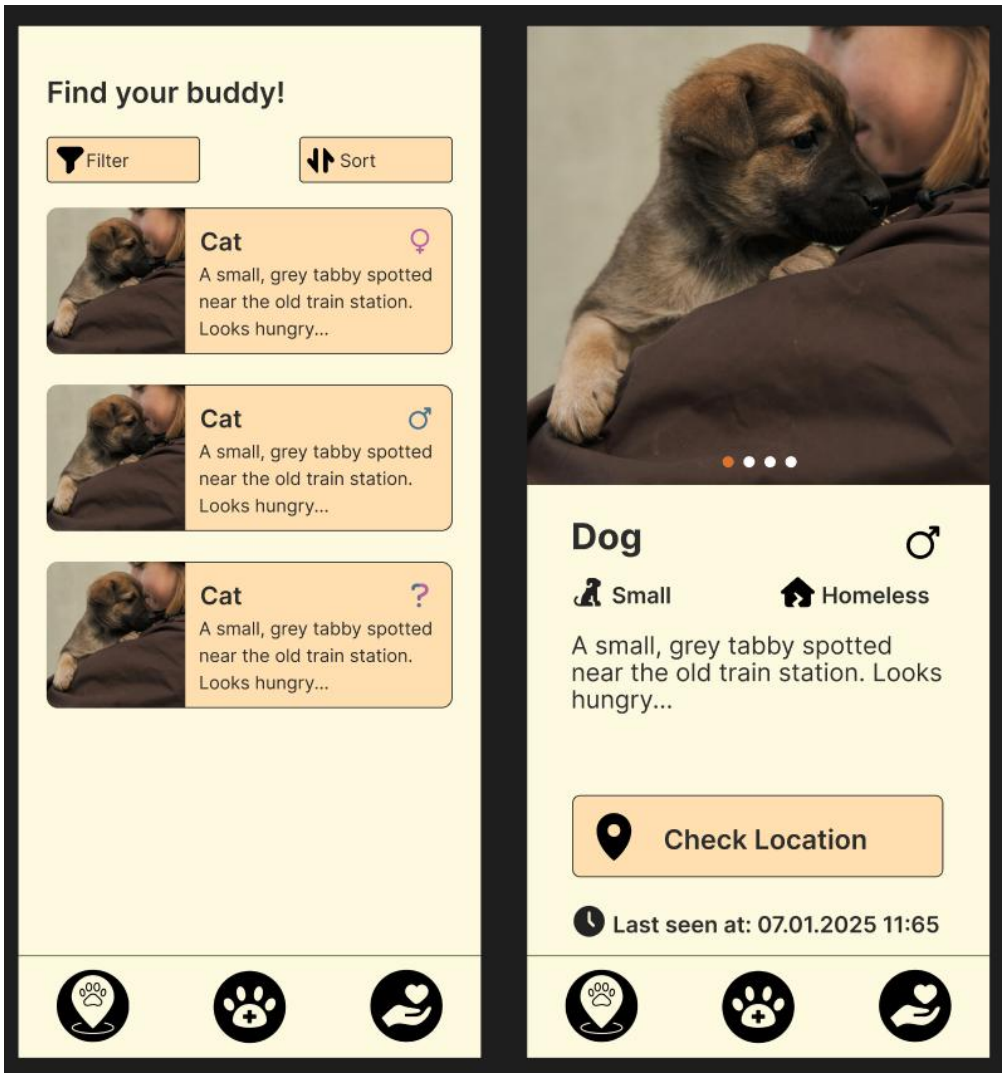
Головний екран містить простий і зрозумілий інтерфейс, що дозволяє користувачеві швидко зорієнтуватися у функціоналі [32]. Процес реєстрації та входу в систему виконаний у мінімалістичному стилі [38]: на екранах передбачено лише необхідні поля для введення даних, а основний акцент зроблено на кнопках дій. Дизайн декількох сторінок авторизації представлений на рисунку 2.9.



Риснок 2.9 – Дизайн екранів авторизації

Основний функціонал додатку реалізований через головну сторінку, де користувачі можуть переглядати список знайдених тварин. Картки тварин містять фото, короткий опис та кнопку для перегляду детальної інформації. Для

зручності навігації додано можливість фільтрації та сортування записів [23]. Натискання на конкретну картку відкриває детальну сторінку, де розміщена вся доступна інформація про тварину, включаючи останнє місцеперебування, опис та додаткові зображення. Вигляд цих екранів зображено на рисунку 2.9.



Риснок 2.10 – Дизайн екранів Main і AnimalDetails

Окремо передбачено інтерфейс для додавання нових тварин. Він містить поле для завантаження фото, автоматичне визначення місцезнаходження, а також можливість додати опис, вказати розмір та інші характеристики. Дизайн

цього розділу витриманий у загальному стилі застосунку, що забезпечує візуальну цілісність.

Загалом, макет мобільного застосунку PetCompass орієнтований на простоту взаємодії, інтуїтивне керування та ефективне вирішення основного завдання — пошуку та допомоги безпритульним тваринам.

2.6 Висновки проєктування програмного забезпечення

Проєктування програмного забезпечення є критично важливою фазою розробки мобільного застосунку PetCompas, яка визначає архітектуру системи, її технічну основу, логічну структуру та взаємозв'язки між компонентами. На цьому етапі були сформульовані ключові технічні рішення, що забезпечують стабільну, масштабовану й гнучку систему, здатну ефективно виконувати свої функції в умовах реального використання.

У процесі проєктування було обґрунтовано вибір клієнт-серверної архітектури, яка дозволяє досягти високої гнучкості в управлінні функціональністю як клієнтської, так і серверної частин застосунку. Такий підхід забезпечує централізовану обробку даних, розмежування обов'язків між фронтендом і бекендом, а також відкриває можливість масштабування системи в майбутньому. Реалізація RESTful API стала основою взаємодії між компонентами [22], забезпечуючи стандартизований спосіб обміну інформацією між клієнтом і сервером, що особливо важливо в умовах багатоплатформного середовища.

Особливу увагу було приділено структурі даних та моделі бази даних, яка побудована з урахуванням специфіки предметної області. Продумана система сутностей, таких як користувачі, тварини, звіти, фото, геолокаційні дані, ролі, коментарі та повідомлення, дозволяє ефективно зберігати інформацію, підтримувати її цілісність і забезпечувати швидкий доступ до потрібних даних.

Завдяки цьому система легко адаптується до змін у логіці використання й дозволяє розширювати функціонал без значних зусиль.

Проєктування серверної частини базувалося на використанні сучасного технологічного стеку [4, 6, 12, 13], що включає ASP.NET WebAPI, Entity Framework, MediatR, AutoMapper, JWT та інші компоненти, які дозволяють досягти високої продуктивності, безпеки та підтримуваності. Впровадження принципів Clean Architecture [11] сприяло чіткому поділу відповідальностей, мінімізації залежностей та спрощенню тестування, що у результаті забезпечує стабільну роботу системи та її гнучкість для майбутніх змін.

Клієнтська частина, створена з використанням .NET MAUI [3], продемонструвала ефективність платформи для побудови кросплатформених мобільних застосунків. Застосування патерну MVVM [16] дозволило забезпечити чистоту коду, спростити його підтримку й масштабування, а також поліпшити взаємодію користувача з інтерфейсом.

Важливо також відзначити, що під час проєктування було передбачено низку нефункціональних вимог, таких як безпека даних, адаптивність до змін навантаження, продуктивність, зручність використання та багатомовність. Їх врахування дозволило створити систему, яка відповідає сучасним стандартам якості програмного забезпечення та може бути успішно застосована в умовах реального середовища.

У підсумку, етап проєктування став фундаментом для успішної реалізації програмного продукту. Всі архітектурні, технологічні та логічні рішення були прийняті з урахуванням вимог до функціональності, масштабованості, зручності та надійності. Це забезпечило ефективну підготовку до наступного етапу — програмної реалізації, і створило передумови для подальшого вдосконалення та розвитку системи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Розробка бази даних

Одним із ключових етапів під час створення серверної частини системи стала розробка бази даних, яка відповідає всім функціональним потребам застосунку. Основна задача полягала в тому, щоб забезпечити надійне зберігання даних про користувачів, знайдених тварин, їхні фотографії, місцезнаходження, коментарі та інші пов’язані об’єкти, які формують логіку роботи додатку.

В якості системи керування базами даних було обрано SQL Server — потужний та стабільний інструмент [21], який чудово інтегрується з платформою .NET. Цей вибір зумовлений не лише зручністю використання під час розробки, а й високою продуктивністю та можливістю масштабування у майбутньому.

Для взаємодії із базою даних був застосований Entity Framework (EF Core) — сучасний ORM-фреймворк [6] (Object-Relational Mapping), який дозволяє працювати з базою даних через C#-класи, не використовуючи сирі SQL-запити. За допомогою EF було створено сутності, які відображають таблиці бази даних, а також налаштовано зв’язки між ними. Це значно спрощує розробку та дозволяє уникнути багатьох помилок, пов’язаних із вручну написаними SQL-інструкціями.

Структура бази даних побудована на основі 8 основних таблиць, кожна з яких виконує певну роль у загальній архітектурі системи:

1. User — зберігає облікові дані та контактну інформацію зареєстрованих користувачів. Кожен запис містить логін, пароль (у хешованому вигляді), email та дату реєстрації.

2. Roles — визначає, яку роль має користувач у системі (наприклад, звичайний користувач, модератор чи адміністратор). Це дозволяє обмежувати доступ до певного функціоналу.

3. Animals — таблиця для зберігання інформації про знайдених тварин, включаючи вид тварини, опис, дату знаходження, стать та інші важливі характеристики.

4. Location — містить координати (широта та довгота), які відповідають географічному положенню, де користувач зробив фото тварини. Ці дані використовуються для відображення тварин на карті.

5. AnimalPhotos — таблиця, яка містить посилання на завантажені фотографії. Кожне фото прив’язується до певної тварини.

6. Reports — фіксує інформацію про події, які залишили користувачі, наприклад, знахідки, втрати або оновлення щодо тварин. Також тут можна зберігати історію змін.

7. Notification — призначена для зберігання push- або email-сповіщень, які надсилаються користувачам (наприклад, про появу нової тварини поблизу).

8. Comments — забезпечує можливість додавати текстові повідомлення до тварин чи звітів. Це дозволяє іншим користувачам ділитися додатковою інформацією або уточненнями.

Всі таблиці мають чіткі зв’язки. Наприклад, кожен користувач може мати декілька доданих тварин, фотографій, коментарів та звітів. Кожна тварина може бути пов’язана з однією або кількома фотографіями, а також з точним місцезнаходженням. Для реалізації цих зв’язків використовувалися ключі — первинні та зовнішні, що забезпечують логічну цілісність даних.

Всі зв’язки між таблицями були реалізовані через відношення "один-до-багатьох" (наприклад, один користувач — багато тварин) або "багато-до-одного" (наприклад, багато фотографій — одна тварина). Це дозволяє легко

будувати запити, відображати пов’язані об'єкти та ефективно працювати з даними у застосунку.

Щоб уникнути проблем із дублюванням інформації, було встановлено відповідні обмеження: наприклад, унікальні значення для email, обов’язковість заповнення певних полів, обмеження довжини тексту тощо. Це все дозволяє підтримувати чистоту та правильність даних.

Важливою частиною розробки структури бази даних стало створення міграцій за допомогою можливостей Entity Framework [6]. Міграції дозволяють автоматично генерувати SQL-команди для створення таблиць, ключів та індексів, а також зберігати історію змін у структурі. Якщо в майбутньому потрібно буде додати нову таблицю чи змінити вже існуючу — достатньо створити нову міграцію, яка буде застосована до бази даних автоматично. Це значно полегшує обслуговування проєкту.

У системі передбачено каскадне видалення пов’язаних записів для збереження цілісності даних. Наприклад, при видаленні користувача автоматично видаляються всі його тварини, фотографії, коментарі, звіти та сповіщення. Це дозволяє уникнути появи «висячих» записів у базі даних і спрощує її обслуговування. Каскадне видалення було реалізовано за допомогою відповідних налаштувань у моделях Entity Framework.

Для захисту даних від втрат унаслідок технічних збоїв, помилок користувача або інших непередбачуваних ситуацій у системі реалізовано механізм регулярного резервного копіювання бази даних [21]. Це важливий елемент загальної стратегії забезпечення надійності та відновлюваності даних. Процес резервного копіювання було автоматизовано за допомогою SQL Server Agent — спеціального сервісу, який дозволяє створювати завдання (jobs) з розкладом запуску, моніторингом виконання та логуванням результатів. Усі резервні копії зберігаються у захищеному каталозі на сервері, а також регулярно

копіюються на зовнішнє хмарне сховище (наприклад, через скрипт з використанням Azure Blob Storage), що додатково підвищує надійність системи.

Також реалізовано механізм оповіщення адміністратора у разі помилок під час створення бекапу — наприклад, через email-сповіщення або лог-файли. Це дозволяє оперативно реагувати на потенційні проблеми та не допустити втрати важливих даних.

Уся стратегія резервного копіювання розроблена з урахуванням майбутнього масштабування, тому при збільшенні обсягів даних чи додаткових вузлів системи її можна легко адаптувати без перебудови основної логіки.

Особлива увага під час розробки була приділена захисту персональних даних користувачів, зокрема, облікової інформації. Одним із ключових аспектів безпеки стало надійне зберігання паролів. Для цього використовується механізм PasswordHasher, який забезпечує хешування паролів перед їхнім збереженням у базі даних. Це означає, що паролі зберігаються не у відкритому вигляді, а у вигляді криптографічно захищеного хешу, що унеможливлює їхнє безпосереднє прочитання навіть у разі витоку даних.

Крім того, в процесі хешування застосовується техніка HashSalt — додавання випадкових значень до кожного паролю перед обчисленням хешу. Це дозволяє уникнути атак типу «rainbow tables», у яких використовуються попередньо згенеровані таблиці хешів для популярних паролів. Навіть якщо два користувачі мають однаковий пароль, їх хеші в базі даних будуть різними завдяки унікальному значенню солі.

Такий підхід забезпечує високий рівень захисту навіть у випадку, якщо база даних потрапить до рук зловмисників. Реалізація PasswordHasher також враховує актуальні криптографічні алгоритми [27], які постійно оновлюються у відповідності до сучасних вимог безпеки.

Окрім захисту паролів, в системі передбачено базові заходи захисту від несанкціонованого доступу, а також можливість розширення безпекових

механізмів у майбутньому — наприклад, шляхом додавання двофакторної автентифікації або шифрування додаткових чутливих даних користувача.

Також було проведено індексацію полів, за якими найчастіше відбувається фільтрація [21] — наприклад, місцезнаходження тварини або дата додавання. Це дозволяє пришвидшити виконання запитів до бази та покращити загальну продуктивність застосунку, особливо при великій кількості записів.

Під час створення таблиць та їх структури було враховано майбутній розвиток проєкту — база даних побудована з урахуванням гнучкості, тобто за потреби її можна безболісно розширити, додавши нові поля чи таблиці без порушення існуючих даних.

Таким чином, створена база даних повністю відповідає вимогам застосунку, підтримує всю необхідну функціональність та дозволяє ефективно взаємодіяти з інформацією, забезпечуючи швидкий пошук, фільтрацію, оновлення та збереження важливих записів. Поєднання SQL Server та Entity Framework дозволило створити стабільну основу для подальшого розвитку мобільного додатку та веб-сервісу.

3.2 Розробка програмних модулів

3.2.1 Розробка сервера

Розробка серверної частини додатку для допомоги безпритульним тваринам здійснювалася з використанням технологій ASP.NET Core, що забезпечують високу продуктивність, масштабованість та гнучкість при реалізації сучасних RESTful API [5]. Архітектура серверного застосунку базується на багаторівневому підході, де кожен логічний рівень винесено в окремий проєкт, що значно спрощує підтримку, тестування та розвиток системи.

Основу серверу складає проєкт PetCompas (рисунок 3.1), який виступає точкою входу та відповідає за конфігурацію веб-серверу, реєстрацію служб,

налаштування маршрутизації та взаємодію з клієнтською частиною. Саме тут визначаються контролери, які приймають HTTP-запити, викликають відповідні обробники та повертають результати у форматі JSON. Для структурування запитів і відповідей використовуються DTO-класи, що забезпечує чітке розмежування між внутрішніми сутностями та зовнішнім API. Також цей проект містить ШІ файл yolo11n.pt, який використовується для пошуку тварин на зображеннях.

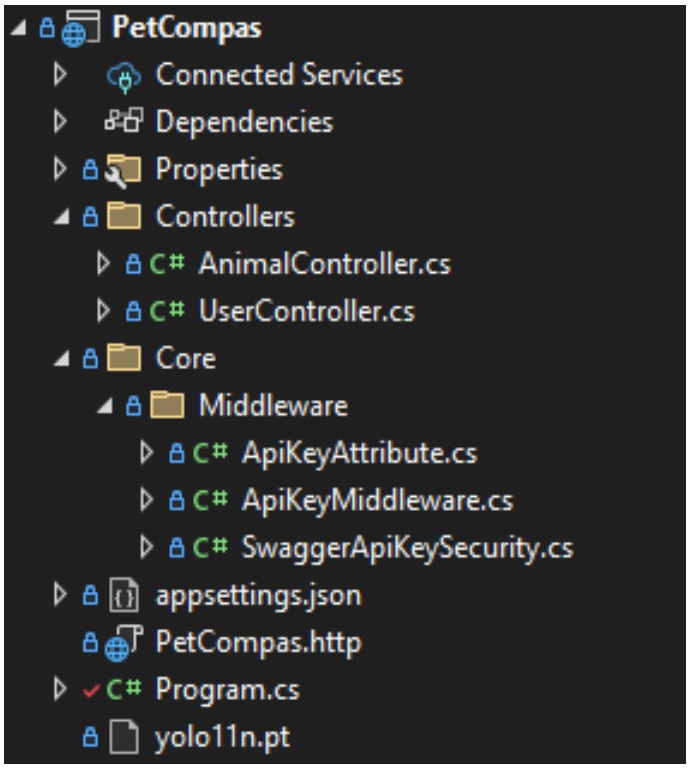


Рисунок 3.1 – Структура проекту PetCompas

Проєкт PetCompas.Domain (рисунок 3.2) містить усі основні сутності, які відображають модель предметної області. До прикладу, сутність User містить інформацію про користувачів додатку, зокрема ім’я, електронну адресу, роль (звичайний користувач або волонтер), а також список створених ними повідомлень. Сутність Report представляє повідомлення про знайдену або загублену тварину, включаючи опис, фотографії, координати, дату створення та

статус (активне, вирішене). Також у проєкті Domain визначено сутності Animal, AnimalPhoto, Location та допоміжні переліки — такі як AnimalType, Gender, Size, що описують характеристики тварин. Між сутностями реалізовані зв’язки один-до-багатьох і багато-до-одного за допомогою навігаційних властивостей, що дозволяє зручно будувати складні запити.

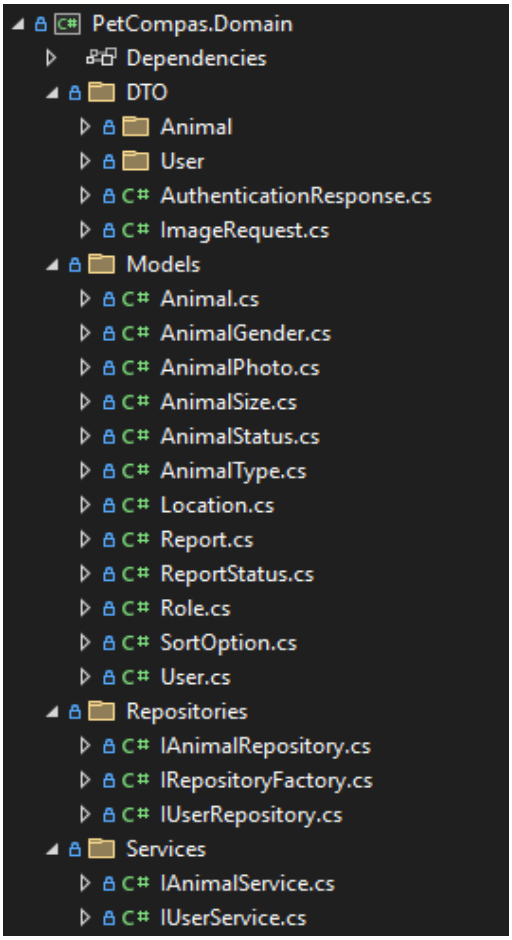


Рисунок 3.2 – Структура проекту PetComaps.Domain

Всі бізнес-правила та логіка обробки запитів реалізовані в проєкті PetCompas.Handlers (рисунок 3.3). Тут розміщені обробники запитів, які відповідають за створення, редагування, пошук, фільтрацію та видалення записів. Цей підхід базується на шаблоні CQRS (Command and Query Responsibility Segregation) [4], який дозволяє чітко розділити операції читання і

запису, спрощуючи масштабування та підтримку коду. Кожен обробник реалізує окрему дію — наприклад, CreateReportHandler, UpdateAnimalHandler, GetAllReportsHandler тощо. Для обміну даними між рівнями використовується шаблон Request/Response, що підвищує прозорість виконання операцій, Request також поділяється на Command і Query (Command використовується в основному для Post, Put та Delete запитів, а Query для Get).

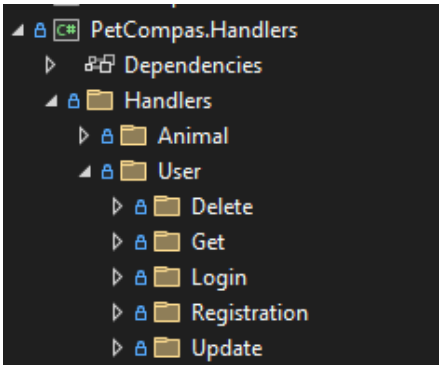


Рисунок 3.3 – Структура проекту PetComaps.Handlers

Проєкт PetCompas.Infrastructure (рисунок 3.4) виконує ключову роль у забезпеченні збереження даних та організації взаємодії із базою даних. В його основі реалізовано контекст бази даних за допомогою технології Entity Framework Core [6], що дозволяє ефективно працювати з реляційною структурою з використанням об’єктно-орієнтованого підходу.

Особливу увагу в межах цього модуля було приділено роботі з геолокаційною інформацією [15], оскільки одна з центральних функцій застосунку — це фіксація місцезнаходження знайдених тварин. У відповідних сутностях передбачено збереження координат широти та довготи, що дозволяє точно ідентифікувати місце події. Надалі ці координати використовуються в клієнтській частині для побудови інтерактивної карти з маркерами, які відображають розташування тварин. Завдяки цьому користувачі можуть швидко зорієнтуватися у просторі та прийняти рішення про допомогу тим тваринам, які

знаходяться поруч. Такий механізм геоприв'язки суттєво підвищує ефективність системи та забезпечує її практичну користь у реальних умовах.

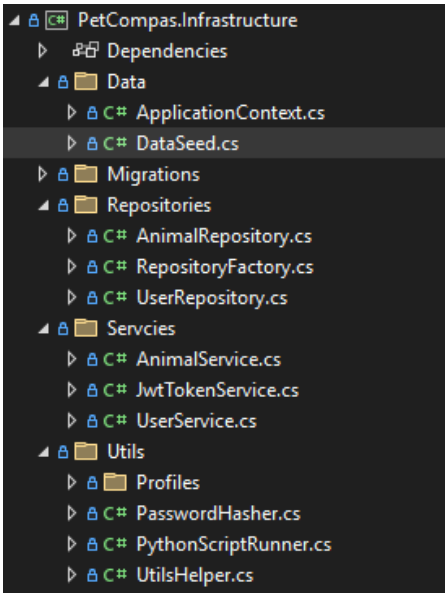


Рисунок 3.4 – Структура проекту PetComaps.Infrastructure

Додаткову функціональність винесено в паппку Utils, що містить універсальні утиліти, розширення та сервіси. Серед них — логування, робота з зображеннями, генерація унікальних ідентифікаторів, надсилання електронних листів тощо. Це дозволяє уникати дублювання коду та централізовано керувати допоміжною логікою.

Уся система побудована з урахуванням принципів SOLID [39], що забезпечує гнучкість, масштабованість і модульність. Для управління залежностями використовується вбудована система DI (Dependency Injection), яка дозволяє легко впроваджувати залежності між компонентами, тестувати їх ізоляційно та уникати жорстких зв'язків.

У вебзастосунку, створеному за допомогою ASP.NET Core, ключовим компонентом взаємодії між клієнтом і сервером є контролери. Контролери обробляють HTTP-запити, викликають необхідні сервіси або обробники, і повертають відповідь у форматі JSON або інший відповідний формат. У проєкті

реалізовано UserController, який відповідає за обробку запитів, пов’язаних із користувачами: реєстрація, авторизація, оновлення, отримання інформації та видалення.

Контролер позначений атрибутом [ApiController], що дозволяє автоматично здійснювати валідацію моделі та спрощує обробку помилок. Атрибут [Route("api/users")] вказує, що всі маршрути контролера починаються з api/users.

Клас використовує підхід CQRS (Command Query Responsibility Segregation), у якому запити (queries) і команди (commands) обробляються окремо за допомогою бібліотеки MediatR [4]. Це дозволяє спростити логіку контролера, передаючи всю бізнес-логіку в окремі обробники.

Крім того, контролер захищений кастомним фільтром ApiKeyAttribute, що додає додатковий рівень авторизації.

Для забезпечення безпечної аутентифікації користувачів у проєкті реалізовано механізм генерації та валідації JWT-токенів [13] (JSON Web Token). Цей підхід дозволяє авторизувати користувача після входу в систему без збереження стану сесії на сервері. Токен формується при успішному вході користувача та містить у собі унікальний ідентифікатор, email, а також ролі, які необхідні для контролю доступу до ресурсів.

Генерація токена здійснюється у сервісі JwtTokenService, де використовується симетричний ключ підпису, що зберігається у конфігурації застосунку. Термін дії токена за замовчуванням встановлено на 3 дні. Цей функціонал реалізовується в методі GenerateToken в середині JwtTokenService.

Метод генерації JWT токена:

```
        public string GenerateToken(int id, string email, IEnumerable<string>
roles)
        {
            var claims = new List<Claim>
            {
                new Claim(ClaimTypes.NameIdentifier, id.ToString()),
```

```
                new Claim(ClaimTypes.Email, email),
                new Claim(ClaimTypes.Role, string.Join(",", roles)),
                new Claim(JwtRegisteredClaimNames.Jti,
Guid.NewGuid().ToString())
            };

            var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["JwtSettings:Secret"]
));
            var creds = new SigningCredentials(key,
SecurityAlgorithms.HmacSha256);

            var token = new JwtSecurityToken(
                issuer: _configuration["JwtSettings:Issuer"],
                audience: _configuration["JwtSettings:Audience"],
                claims: claims,
                expires: DateTime.UtcNow.AddDays(3),
                signingCredentials: creds);

            return new JwtSecurityTokenHandler().WriteToken(token);
        }
```

У свою чергу, метод ValidateToken дозволяє перевірити достовірність токена, зокрема його підпис, термін дії, відповідність вказаним issuer та audience. У разі недійсного токена метод повертає null, що гарантує, що доступ не буде надано.

Ще одним важливим аспектом безпеки є збереження паролів [27]. Замість зберігання паролів у відкритому вигляді реалізовано механізм хешування з використанням HMACSHA256 та криптографічно безпечної солі. При реєстрації генерується сіль, і пароль хешується разом із нею. Результат зберігається у базі даних. При вході до системи хеш введеного пароля порівнюється з тим, що збережений у базі, використовуючи ту ж саму сіль. Це забезпечує захист навіть у разі витоку бази даних.

Метод хешування пароля:

```
private static string HashPassword(string password, byte[] salt)
{
    using (var hmac = new HMACSHA256(salt))
    {
        var passwordBytes = Encoding.UTF8.GetBytes(password);
        var hashBytes = hmac.ComputeHash(passwordBytes);
        return Convert.ToBase64String(hashBytes);
    }
}
```

Для реалізації функціоналу автоматичного розпізнавання тварин за фотографією в проєкті використано попередньо натреновану модель нейромережі YOLOv8 від бібліотеки ultralytics [17], яка є однією з найсучасніших та найефективніших моделей об'єктного детектування. Метою даного етапу є визначення, чи присутній на зображенні собака або кішка, що дозволяє автоматично класифікувати фотографії, які користувачі завантажують у застосунок.

Реалізація взаємодії з нейромережею відбувається через окремий Python-скрипт, який викликається з бекенду .NET за допомогою класу PythonScriptRunner. Зображення у форматі Base64 зберігається у тимчасовий файл, передається до Python-скрипту як аргумент, обробляється моделлю, після чого результат записується у вихідний файл і зчитується назад у C# код.

Нижче наведений приклад коду виклику Python скрипта на локальній машині:

```
public static async Task<Either<Error, string>> RunPythonScriptWithBase64(string
base64Image)
{
    string tempInputPath = Path.Combine(Path.GetTempPath(), "input_base64.txt");
    string tempOutputPath = Path.Combine(Path.GetTempPath(),
"output_base64.txt");
    try
    {
        await File.WriteAllTextAsync(tempInputPath, base64Image);
        ProcessStartInfo psi = new ProcessStartInfo
        {
            FileName = PythonPath,
            Arguments = $"\"{PythonScriptPath}\" \"{tempInputPath}\"
\"{tempOutputPath}\"",
            RedirectStandardError = true,
            RedirectStandardOutput = true,
            UseShellExecute = false,
            CreateNoWindow = true
        };
        using (Process process = new Process { StartInfo = psi })
        {
            var tcs = new TaskCompletionSource<bool>();
            process.Start();
            string result = (await
File.ReadAllTextAsync(tempOutputPath)).Trim();

            if (result == "dog" || result == "cat")
            {
                return Either<Error, string>.Right(result);
            }
```

```
            else
            {
                return Either<Error,
string>.Left(Error.New(AnimalNotIdentified));
            }
    }
}
```

Сам Python-скрипт завантажує модель YOLO, декодує зображення з Base64 у формат, що підтримується OpenCV, і передає його на обробку. Модель аналізує зображення та повертає мітку "dog", "cat" або "unknown", якщо відповідного об'єкта не виявлено. Ключова перевага такого підходу — можливість швидко змінювати модель чи розширювати її функціональність без змін у C# коді.

Завдяки використанню нейромережі вдалось автоматизувати частину аналізу фотографій, підвищивши зручність і точність ідентифікації тварин, що робить застосунок значно зручнішим.

Для реалізації функціоналу пошуку найближчих тварин [31] відносно місцезнаходження користувача використовується формула гаверсинуса (haversine formula). Ця формула дозволяє обчислити відстань між двома точками на поверхні Землі, заданими їхніми широтою та довготою, враховуючи кривизну планети. Такий підхід забезпечує високу точність при пошуку тварин у межах певного радіусу або для сортування за відстанню.

Використання формули реалізовано у відповідному обробнику запиту GetClosest, який отримує координати користувача та повертає список тварин, відсортованих за відстанню. Визначення відстані здійснюється методом GetDistance, який обчислює відстань у кілометрах за формулою 3.1.

$$d = 2r \arcsin\left(\sqrt{sin^2\left(\frac{\varphi_2-\varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\, sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right) \qquad (3.1)$$

Такий механізм дає змогу користувачеві миттєво дізнатися про тварин, які знаходяться поруч із ним, що є критично важливою функцією в контексті швидкого реагування на повідомлення про знайдених чи загублених тварин. Система не лише враховує координати, збережені в базі, але й ефективно працює з великою кількістю записів, завдяки попередньому отриманню усіх об'єктів та сортуванню їх на основі розрахованої відстані.

Для забезпечення високої доступності, масштабованості та зручності в обслуговуванні, серверна частина застосунку була розгорнута у хмарному середовищі Microsoft Azure. Основним хостингом для API-сервісу виступає Azure App Service — керована платформа для хостингу вебзастосунків, яка дозволяє автоматично масштабувати ресурси, управляти середовищами (наприклад, staging/production) та здійснювати безперервне розгортання з репозиторію. Завдяки інтеграції з системою контролю версій GitHub, оновлення сервера можуть виконуватись автоматично після злиття змін у головну гілку.

Збереження та обробка даних реалізується через Azure Database, яка надає керовану, безпечну та високо доступну реляційну СУБД яка дозволяє розгортати SQL Server базу даних. Для зберігання користувацьких зображень застосовується Azure Blob Storage — об'єктне сховище, що підтримує великі обсяги даних і забезпечує швидкий доступ до файлів. Кожне зображення, завантажене користувачем, спочатку передається на сервер, де проходить попередню перевірку, після чого зберігається у blob-контейнері, в свою чергу blob-контейнер повертає URL посилання на зображення.

Важливим компонентом системи є модуль штучного інтелекту, реалізований у вигляді окремого Docker-контейнера, який працює незалежно від основного API. Цей контейнер розгорнутий на окремому віртуальному середовищі Azure, що забезпечує ізоляцію, гнучке управління ресурсами та можливість оновлення або заміни моделі без втручання в основний бекенд. API

взаємодіє з контейнером через HTTP-запити, надсилаючи зображення для обробки та отримуючи назад результати класифікації.

Загальна архітектура рішення зображена на рисунку 3.5, де показано основні компоненти системи та їх взаємодію. Вона відповідає підходу мікросервісної архітектури, оскільки кожен компонент системи (сервер, база даних, зберігання, нейромережа) може масштабуватись та оновлюватись окремо, без впливу на інші частини. Такий підхід підвищує надійність і дозволяє гнучко реагувати на зміну навантаження.

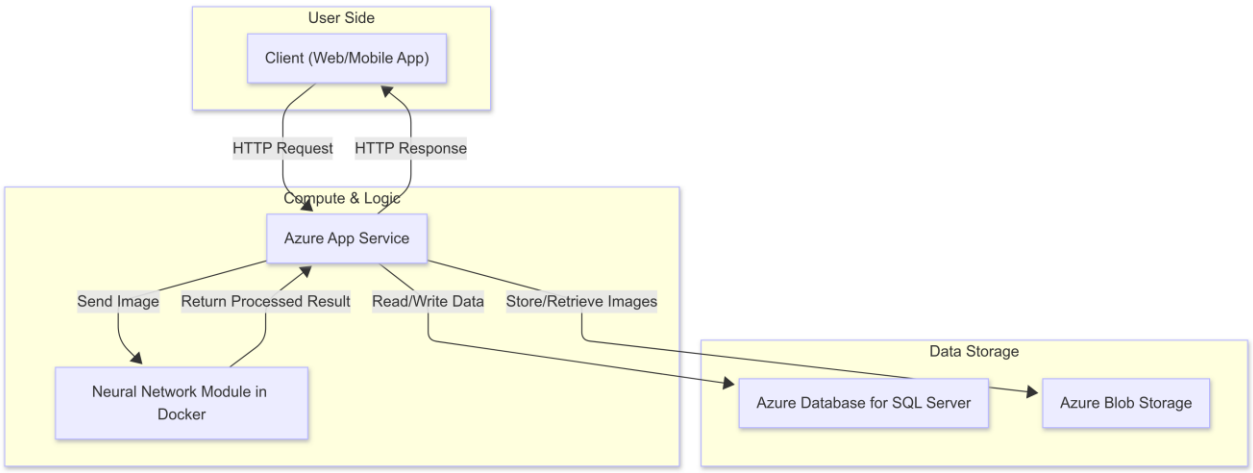


Рисунок 3.5 – Інфраструктурна діаграма

3.2.2 Розробка клієнтського додатка

Проєкт мобільного застосунку має назву PetCompassMobile і реалізований з використанням сучасної міжплатформної технології .NET MAUI [3], що забезпечує можливість розробки одного застосунку для кількох операційних систем, зокрема Android та iOS. Завдяки цьому підходу розробка стає швидшою, а підтримка застосунку — ефективнішою, оскільки значна частина коду є спільною для всіх платформ. Архітектура проєкту сформована з урахуванням принципів модульності, масштабованості та зручності в обслуговуванні, що є

особливо важливим для довготривалих проєктів із перспективою подальшого розвитку.

У структурі проєкту (рисунок 3.6) передбачено окремі каталоги, кожен з яких виконує чітко визначену роль. Каталог моделей, що позначений як Models, містить класи, які описують структуру даних, що використовуються в застосунку. Це можуть бути, наприклад, моделі тварин, користувачів, геолокації чи звітів. Такі об'єкти відповідають даним, які надходять із сервера або передаються на нього у процесі роботи програми.

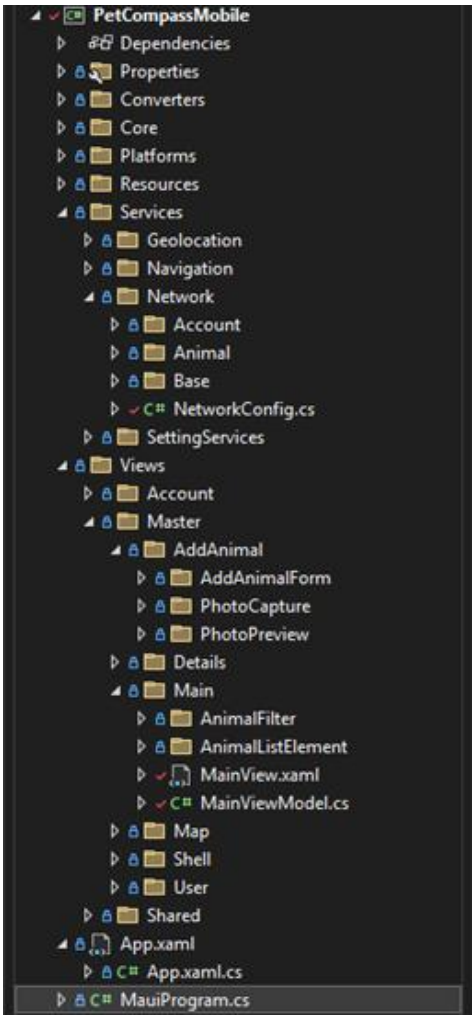


Рисунок 3.6 – Структура проекту PetComapsMobile

Інтерфейс користувача реалізується у XAML-файлах, що розміщені в каталозі Views. Кожна сторінка застосунку має власний XAML-файл, у якому

визначено візуальне оформлення, а також відповідний файл із логікою обробки подій та взаємодії з даними. Для прикладу, сторінки з головним вікном, переглядом деталей тварини чи формою створення звіту мають свої окремі представлення.

Каталог ViewModels відповідає за реалізацію логіки, яка поєднує дані з інтерфейсом користувача. Для цього застосовується архітектурний патерн MVVM (Model-View-ViewModel) [16], який забезпечує чітке розділення відповідальностей між представленням, моделями та логікою. Це дозволяє значно спростити тестування та супровід коду.

Окрему важливу роль відіграє каталог Services, де зберігаються класи для роботи з зовнішніми ресурсами. Сюди входять сервіси для обміну даними з сервером через HTTP-запити, сервіси для роботи з камерою та обробки фотографій, сервіси геолокації, а також взаємодії з локальним сховищем, наприклад, для збереження токена авторизації.

Візуальний стиль, кольорова гама, шрифти та зображення застосунку зібрані у каталозі Resources. Тут містяться XAML-файли зі спільними стилями для всього інтерфейсу, палітра кольорів, а також окремі папки з шрифтами та графічними елементами, що використовуються в інтерфейсі. Це дозволяє забезпечити єдиний візуальний стиль і швидко змінювати дизайн при потребі.

Допоміжні елементи, такі як конвертери значень або утиліти для обробки даних, розміщуються в каталозі Helpers. Вони використовуються як у XAML, так і у ViewModel-ах, і допомагають зробити код простішим і зручнішим у підтримці.

Платформозалежна функціональність реалізується у каталозі Platforms, де зберігаються частини коду, які залежать від конкретної операційної системи. Це можуть бути специфічні API, що доступні лише на Android або iOS, наприклад, для взаємодії з Bluetooth або системними налаштуваннями.

Базова конфігурація застосунку зосереджена у файлах App.xaml та AppShell.xaml [23]. Перший відповідає за глобальні ресурси, які застосовуються по всій програмі, а другий — за навігацію між сторінками, яка реалізується за допомогою Shell-навігації, стандартного механізму у .NET MAUI.

Загальна структура проєкту побудована так, щоб забезпечити гнучкість у розробці, простоту масштабування та зручність у підтримці. Завдяки застосуванню архітектури MVVM та модульного підходу, код залишається організованим та легко розширюваним. Застосунок інтегрується з сервером за допомогою REST API [22], що дозволяє надсилати дані про безпритульних тварин, отримувати інформацію про вже доданих тварин, а також фіксувати геолокацію користувача під час створення звіту. Крім цього, користувач має змогу переглядати карту з відмітками знайдених тварин, що допомагає краще координувати допомогу та підвищує ефективність пошуку.

Однією з ключових функцій застосунку є можливість автоматичного визначення поточного місцезнаходження користувача [15]. Це дозволяє точно фіксувати геолокацію знайденої тварини та зберігати її разом із рештою даних у створеному звіті. Для реалізації цієї функціональності у застосунку створено окремий сервіс, який відповідає за отримання координат пристрою — GeolocationService. Цей сервіс інкапсулює всю логіку, пов’язану з доступом до GPS, перевіркою прав доступу до геолокації, а також збереженням координат у локальному сховищі.

Отримання координат здійснюється асинхронно, із перевіркою дозволів доступу до місця розташування. Якщо дозволи не надано, система запитує їх у користувача. У разі успішного отримання координат вони зберігаються в локальне сховище у вигляді JSON-рядка для подальшого використання, наприклад, у випадках, коли немає доступу до GPS у момент створення звіту.

Основний метод сервісу GetCurrentLocation спочатку викликається на головному потоці, де відбувається перевірка дозволів та спроба отримати

координати. У разі успіху створюється об'єкт Location, який зберігається для подальшого використання.

```
 public async Task<PetCompassMobile.Core.DTO.Animals.Location>
GetCurrentLocation()
    {
        Core.DTO.Animals.Location resultLocation = null;

        await MainThread.InvokeOnMainThreadAsync(async () =>
        {
            if (!await CheckAndRequestLocationPermission()) return;

            Location? location = await GetLocation() ?? new Location();
            if (location != null)
            {
                resultLocation = new Core.DTO.Animals.Location
                {
                    Latitude = location.Latitude,
                    Longitude = location.Longitude
                };

                await SaveLocation(resultLocation);
            }
        });
```

Після отримання координат вони передаються в метод SaveLocation, який серіалізує об'єкт Location у JSON-формат та зберігає його через сервіс налаштувань (ISettingService). Це дозволяє повторно використовувати останнє відоме місцезнаходження навіть без доступу до GPS у певний момент.

Для підвищення стабільності роботи сервіс також має метод GetLastKnownLocation, який спочатку пробує витягнути останні координати з локального сховища. Якщо це не вдається, він ініціює отримання нових координат через GPS. Таким чином, застосунок забезпечує надійний механізм геопозиціонування навіть в умовах нестабільного сигналу або обмеженого доступу до системних ресурсів.

У мобільному застосунку PetCompassMobile інтеграція Google Maps реалізована за допомогою вбудованої MAUI-карти [31]. Основна логіка роботи з картою знаходиться у двох файлах — MainMapView.xaml (рисунок 3.7) та MainMapViewModel.cs. Перший відповідає за візуальну частину: тут визначено

компонент карти (<maps:Map>), який відображається на екрані. Другий — це модель представлення, яка керує логікою взаємодії з картою та завантаженням даних.

Після відкриття сторінки застосунок отримує список тварин із сервера (через сервіс IAnimalService) та додає їх на карту як мітки. Це відбувається у методі RefreshAnimalList файлу MainMapViewModel.cs. Далі кожна тварина виводиться у вигляді мітки (Pin) на карті в MainMapView.xaml.cs, а при натисканні на мітку користувач може переглянути деталі про тварину.

Карта також автоматично фокусується на збереженому або поточному місцезнаходженні користувача. За це відповідає властивість CurrentMapFocus у моделі представлення, яка оновлюється при отриманні геолокації та змінює центр карти відповідно.

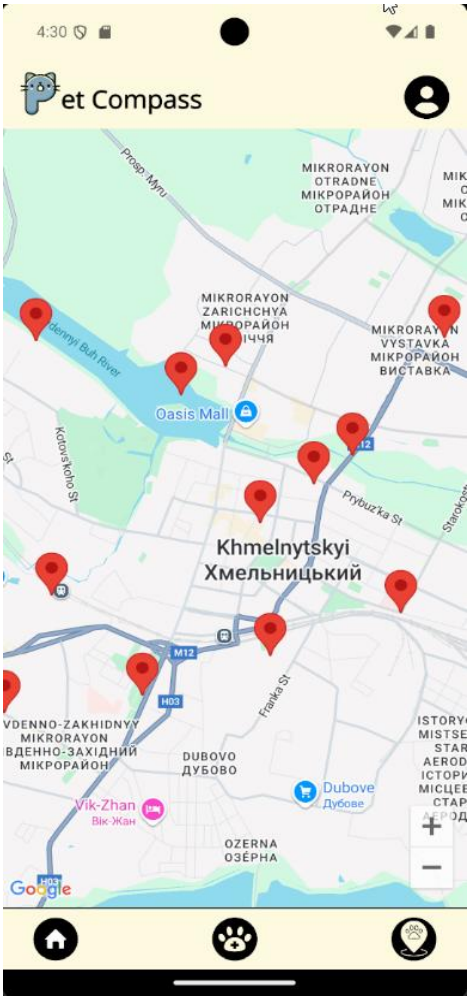


Рисунок 3.7 – Карта

Одним із ключових елементів користувацького інтерфейсу є форма додавання інформації про знайдену тварину — AddAnimalForm, яка дозволяє ввести опис, обрати тип, розмір, стать, а також додати фото. Цей екран поєднує текстові поля, випадаючі списки та візуальні елементи, що забезпечує зручність і швидкість внесення даних. У зв'язці з цим екраном працює PhotoCapture — інтерфейс з камерою пристрою [35], за допомогою якого користувач може зробити фотографію тварини прямо під час створення звіту. Вікно PhotoCapture включає кнопку зйомки, індикатор стану та перегляд зображення в реальному часі, а також дозволяє змінювати камеру. Ці екрани представлені на рисунку 3.8.

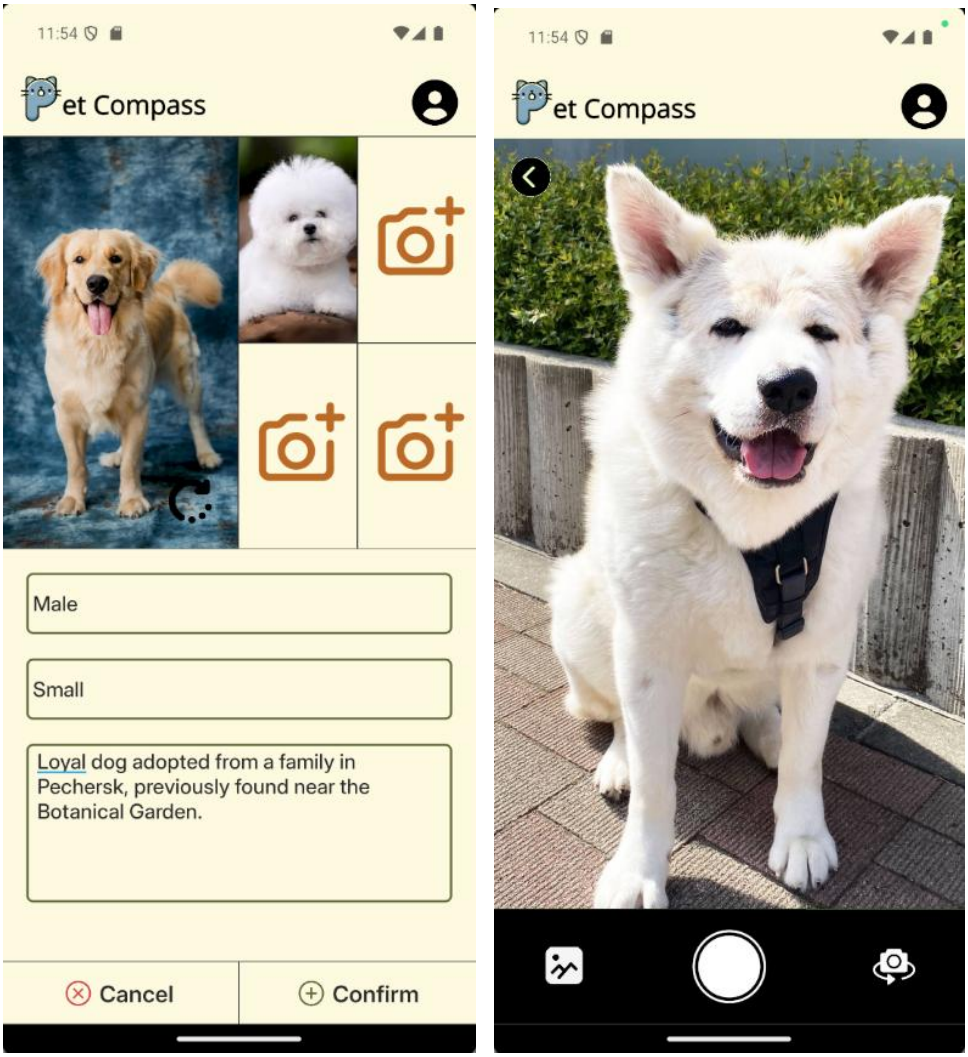


Рисунок 3.8 – Екрани AddAnimalForm і PhotoCapture

Ще один важливий екран — AnimalDetails, який відкривається після вибору конкретної мітки на карті або з переліку тварин. Він відображає детальну інформацію про тварину, включно з фото, описом, місцем знаходження, статтю та розміром. Тут також є можливість перегляду дати публікації та навігації до місця знаходження тварини. Основним вікном після входу у застосунок є MainView, яке виконує роль хабу — з нього користувач може перейти до карти, додати нову тварину, переглянути список звітів або змінити налаштування. Головна сторінка також показує короткий опис сервісу та мотивує користувача допомагати тваринам. Екрани AnimalDetails та MainView представлені на рисунку 3.9.

Рисунок 3.9 – Екрани MainView та AnimalDetails

Отже, проєкт PetCompassMobile демонструє високий рівень організації та сучасний підхід до розробки мобільного застосунку завдяки використанню технології .NET MAUI та архітектури MVVM. Завдяки модульній структурі з чітким розподілом функціональних компонентів забезпечено гнучкість, масштабованість і зручність підтримки коду. Реалізована функціональність, зокрема інтеграція з GPS, робота з картою, сервіси для збереження й обробки даних, дозволяє ефективно фіксувати випадки виявлення безпритульних тварин, що робить застосунок практичним інструментом для користувачів. Інтерфейс є інтуїтивно зрозумілим і добре структурованим, що сприяє зручності використання та підвищує шанси на широке залучення користувачів у процес допомоги тваринам.

3.3 Тестування

Для забезпечення надійності та стабільності програмного забезпечення було проведено повноцінне тестування розробленого застосунку, яке охоплювало клієнтську частину (мобільний застосунок на основі .NET MAUI) та серверну частину (ASP.NET Web API). У процесі тестування використовувалися різні підходи, включаючи функціональне, модульне (unit), інтеграційне та системне тестування. Це дозволило на різних рівнях перевірити правильність роботи компонентів, їх взаємодію, а також загальну відповідність очікуванням користувача.

Особлива увага приділялася модульному тестуванню серверної частини [26, 34], яке дозволило ізольовано перевірити функціональність кожного окремого методу контролера та сервісів. Наприклад, тестувалися методи створення нової тварини, отримання списку тварин, збереження геолокації, валідація даних, а також логіка взаємодії з базою даних. Для написання unit-

тестів використовувався фреймворк xUnit, що дозволяє організовувати гнучку структуру тестів та проводити незалежну перевірку функціоналу [25]. У ході модульного тестування були виявлені та своєчасно усунуті деякі помилки логіки, пов'язані з обробкою некоректних вхідних даних. На рисунку 3.11 представлений приклад результатів виконання модульних тестів, який демонструє успішне проходження усіх перевірок.

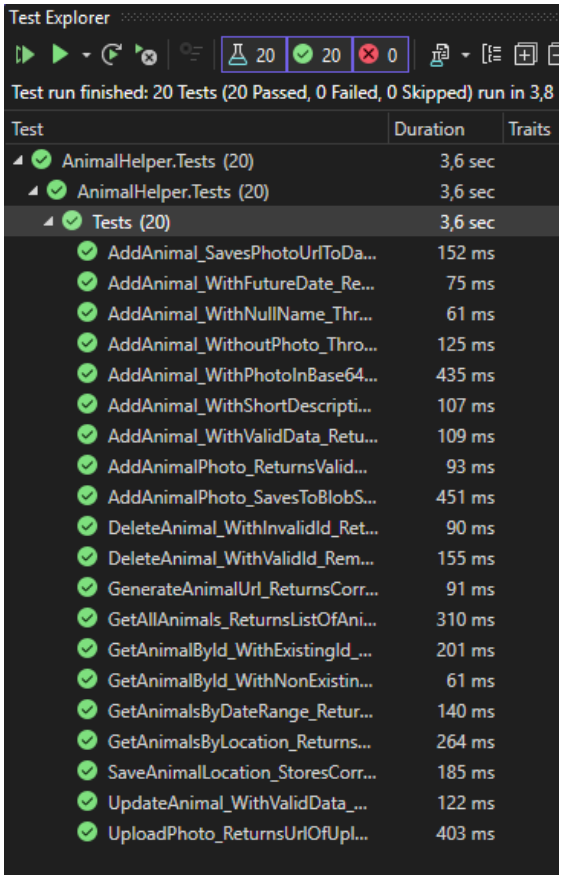


Рисунок 3.11 – Модульні тести

Інтеграційне тестування проводилося з метою перевірки коректної взаємодії між клієнтською та серверною частинами. Під час інтеграційного тестування, яке передбачало перевірку взаємодії між мобільним клієнтом та серверною частиною застосунка, було виявлено проблему з часом відгуку при додаванні нової інформації про знайдену тварину. Зокрема, після надсилання

фотографії на сервер, програма демонструвала значну затримку. Аналіз запитів показав, що основною причиною стало використання формату base64 для передачі зображень, що значно збільшувало розмір запиту. Це не тільки впливало на швидкодію, а й створювало ризик нестабільної роботи мобільного додатку за умов слабкого інтернет-з'єднання.

Для усунення цієї проблеми було ухвалено рішення змінити підхід до зберігання та обробки зображень. Замість передачі фото у вигляді рядка base64, було реалізовано механізм завантаження файлів безпосередньо до Azure Blob Storage. Це дозволило зменшити навантаження на сервер і скоротити час обробки запитів. Після успішного завантаження зображення до хмарного сховища, сервер формує URL-адресу до нього, яка й зберігається у базі даних.

Таким чином, при додаванні нової тварини користувач завантажує фото на blob storage, а сервер оперує вже лише посиланням на це зображення. Такий підхід не лише значно покращив продуктивність застосунка, а й зробив передачу даних ефективнішою та стабільнішою. Під час повторного тестування система показала стабільну роботу навіть при повільному інтернет-з’єднанні, що підтвердило доцільність впроваджених змін.

Системне тестування включало перевірку усього застосунку як єдиного цілого в умовах, наближених до реального використання. Було протестовано основні сценарії взаємодії користувача з системою, включаючи створення нової публікації, перегляд тварин на мапі, сортування за датою або місцем, а також перегляд детальної інформації про знайдену тварину. Під час тестування приділялася увага стабільності роботи програми, зручності інтерфейсу, а також часу відповіді серверу на запити. Загалом, система показала високу продуктивність та правильну роботу при типових навантаженнях.

Окремо було проведено функціональне тестування ключових можливостей застосунку. Приклади функціональний тестів, проведених під час розробки та перевірки, наведені у таблиці 3.1.

Таблиця 3.1.

№	Тестована функція	Результат	Висновок
1	Завантаження зображення тварини	Зображення успішно передано на сервер	Задовільний
2	Визначення геолокації під час додавання фото	Геолокація точно визначена	Задовільний
3	Відображення тварин на інтегрованій карті	Усі тварини коректно відображені	Задовільний
4	Надсилання повідомлення через форму зв’язку	Повідомлення успішно збережене	Задовільний
5	Відображення детальної інформації про тварину	Інформація коректно відображена	Задовільний
6	Обробка ситуації без підключення до інтернету	Показано відповідне повідомлення	Задовільний
7	Валідація порожніх або некоректних полів	Користувач попереджений про помилку	Задовільний
8	Виведення повідомлення про успішне збереження	Повідомлення відображено	Задовільний

Таким чином, процес тестування охопив усі ключові аспекти функціонування розробленого застосунку — від перевірки базової логіки окремих методів (unit-тести) до комплексного аналізу взаємодії між модулями системи (інтеграційні тести). Це дозволило переконатися, що кожен компонент працює відповідно до очікувань, а також гарантувати стабільну роботу всієї системи в реальних умовах.

Особлива увага приділялася перевірці критично важливих сценаріїв: додавання нових тварин, обробки зображень, роботи з геолокацією, збереження даних у хмарних сервісах та взаємодії з нейромережею. Проведення тестів дало змогу виявити потенційні проблеми — зокрема, затримки, пов’язані з

передачею зображень у форматі base64. Після зміни архітектури передачі фото (перехід до використання Azure Blob Storage та зберігання лише URL у базі даних), продуктивність системи помітно зросла, а час обробки запитів зменшився.

Загалом, використання різних рівнів тестування не лише допомогло своєчасно виявити й усунути недоліки, а й стало надійною основою для подальшого масштабування та підтримки проєкту. Результати тестування підтверджують, що застосунок є стабільним, функціонально повноцінним і зручним у використанні для надання допомоги безпритульним тваринам.

3.4 Технічні характеристики системи

Для забезпечення стабільної роботи системи, що допомагає безпритульним тваринам, необхідно врахувати технічні вимоги до окремих її компонентів. Застосунок складається з клієнтської частини (таблиця 3.2), серверної частини (таблиця 3.3), бази даних, контейнера з нейронною мережею для виявлення тварин на фото, а також хмарного сховища для зображень.

Таблиця 3.2

Параметр	Значення
Операційна система	Android 8.0+ / iOS 12+ / Web-браузери (Chrome, Firefox, Edge)
Процесор	ARM або x86, з частотою не менше 1.4 ГГц
Оперативна пам’ять	Від 2 ГБ
Вільне місце на диску	Від 100 МБ
Підключення до Інтернету Мінімум 3G, бажано Wi-Fi або LTE	
Екран	Мінімальна роздільна здатність 720p

Таблиця 3.3

Параметр	Значення
Операційна система	Windows Server 2019 / Ubuntu 20.04+
Процесор	2+ ядра, 2.0 ГГц або більше
Оперативна пам’ять	Від 4 ГБ
Вільне місце на диску	Від 2 ГБ
Мережеве з’єднання	Стабільне з’єднання з затримкою менше 100 мс
Платформа виконання	.NET 8.0
Хостинг	Azure App Service (Standard план або вище)
Бекенд	REST API з JSON, автентифікація через JWT

База даних системи реалізована за допомогою Azure Database for PostgreSQL. Цей сервіс забезпечує стабільне зберігання даних з можливістю масштабування, автоматичним резервним копіюванням і високою доступністю. Для роботи системи достатньо інстансу з конфігурацією: 1 vCore, 2 ГБ оперативної пам’яті, SSD-сховище від 5 ГБ. Підключення відбувається через захищене з’єднання SSL.

Нейромережева модель для виявлення тварин реалізована у вигляді окремого Docker-контейнера, що використовує YOLOv8. Контейнер запускається на окремому віртуальному середовищі або в Azure Container Instances. Для обробки зображень потрібна наявність GPU або, принаймні, процесора з хорошою підтримкою векторних обчислень, 2+ vCPU і щонайменше 4 ГБ ОЗП.

Для зберігання зображень використовується Azure Blob Storage, який забезпечує високу швидкість доступу до файлів, надійне резервне копіювання і масштабованість. Зображення зберігаються у форматі JPEG/PNG, а в базі даних фіксується лише URL для доступу. Це дозволяє значно зменшити навантаження

на сервер і покращити продуктивність клієнтського застосунку при отриманні даних.

Загалом архітектура побудована з урахуванням принципів масштабованості, відмовостійкості та продуктивності, що дозволяє ефективно використовувати систему навіть за зростання кількості користувачів і збільшення обсягів даних.

3.5 Висновки програмної реалізації та тестування

Етап програмної реалізації став кульмінаційним моментом усього циклу розробки мобільного застосунку PetCompas, оскільки саме на цьому етапі було перетворено попередньо спроєктовану архітектуру, логіку та структуру даних у функціональний програмний продукт. Особливістю реалізації даного застосунку стала побудова клієнтської частини за допомогою сучасної кросплатформеної технології .NET MAUI, що забезпечило можливість підтримки одразу кількох мобільних платформ, зокрема Android та iOS, без дублювання коду. Це дозволило зосередитись на зручності користувача, продуктивності інтерфейсу та стабільності взаємодії з серверною частиною. Архітектурно клієнт реалізовано відповідно до патерну MVVM, що забезпечує чітке розділення обов’язків між логікою представлення, бізнес-логікою та даними. Завдяки такому підходу розробка інтерфейсу стала гнучкою, масштабованою та придатною до розширення в майбутньому.

Розробка серверної частини відбувалась з дотриманням принципів Clean Architecture, що дозволило досягти високої модульності та ізоляції відповідальностей у коді. Вся бізнес-логіка була винесена у відповідні рівні, що забезпечило незалежність від конкретних технологій доступу до даних, зовнішніх API або інтерфейсів користувача. Це стало особливо важливим у контексті реалізації таких складних функцій, як реєстрація користувачів,

додавання тварин, перевірка зображень за допомогою нейронної мережі та геолокаційне позиціонування. Сервер побудовано на базі ASP.NET WebAPI з використанням MediatR, що забезпечує реалізацію патерну "Посередник", мінімізуючи зв’язність між компонентами системи.

У рамках реалізації також було побудовано інфраструктурний рівень, який відповідає за зберігання даних та взаємодію з базою. Тут використано ORM Entity Framework Core, що забезпечує ефективне управління міграціями, збереженням сутностей та побудову запитів. Значна увага була приділена підтримці геолокаційного функціоналу, оскільки застосунок передбачає не просто збереження інформації про тварин, а й точну фіксацію їхнього розташування для подальшого відображення на карті в інтерфейсі користувача.

Окреме місце в процесі розробки посідає тестування, яке дозволило перевірити стабільність роботи основних функцій та виявити потенційні помилки ще до впровадження системи у практичне використання. Було проведено ручне функціональне тестування, що охопило ключові сценарії взаємодії користувача із застосунком, зокрема реєстрацію та авторизацію, додавання тварини з фото, визначення місця знахідки, перегляд картки тварини, комунікацію між користувачами, а також отримання сповіщень. У результаті тестування було підтверджено коректність обробки запитів на сервері, відповідність валідації даних очікуваним параметрам, стабільну роботу картографічного компонента та точність передачі координат.

Отже, результати етапу програмної реалізації та тестування свідчать про успішність розробленого програмного продукту та його готовність до реального використання. Усі заплановані функціональні блоки реалізовано повною мірою, архітектура системи виявилася ефективною та придатною для розширення, а якість реалізації підтверджена результатами тестування.

ВИСНОВКИ

У межах даної кваліфікаційної роботи було реалізовано повноцінний цикл створення сучасного мобільного застосунку PetCompas, призначеного для організації допомоги безпритульним і загубленим тваринам. Розробка охопила всі етапи життєвого циклу програмного забезпечення — від детального аналізу предметної області, виявлення соціально значущої проблеми, проєктування архітектури, структури даних і вибору технологічного стеку, до безпосередньої реалізації, тестування та документального оформлення.

Метою дослідження було створення інструменту, що дозволить з одного боку — швидко фіксувати місцезнаходження знайдених тварин із фотографіями, з іншого — забезпечити зручну взаємодію між користувачами, волонтерами, працівниками притулків і власниками загублених тварин. Застосунок покликаний вирішувати реальну проблему безпритульності тварин шляхом створення спільного інформаційного простору, в якому кожен небайдужий може додати повідомлення, знайти загублену тварину або допомогти їй знайти нову домівку.

На першому етапі роботи було проведено всебічний аналіз предметної області. Було досліджено основні чинники, що спричиняють зростання кількості бездомних тварин, а також розглянуто існуючі цифрові сервіси у цій сфері. Аналіз ринку продемонстрував фрагментованість підходів та брак комплексного рішення, яке б поєднувало функції геолокації, візуального розпізнавання, інтерактивної мапи та зворотного зв’язку. Це дозволило сформувати обґрунтовані функціональні та нефункціональні вимоги, що лягли в основу технічного завдання.

Архітектурно система була спроєктована на базі клієнт-серверної моделі з чітким поділом відповідальностей. На сервері реалізовано REST API на основі ASP.NET WebAPI з використанням патерну «Посередник» (MediatR), що дозволило досягти низького зв'язування між компонентами та забезпечити

легкість масштабування. В основі серверної логіки лежить принцип Clean Architecture, що забезпечив ізоляцію бізнес-логіки, незалежність від фреймворків і гнучкість при зміні джерел даних. Модуль інфраструктури реалізовано з використанням ORM Entity Framework Core, що дозволило побудувати логічно структуровану модель бази даних, зокрема для зберігання користувачів, звітів про тварин, фото, геолокаційної інформації та коментарів.

Клієнтська частина реалізована на платформі .NET MAUI, що забезпечило одночасну підтримку Android та iOS при мінімізації зусиль на розробку. У застосунку було реалізовано модульну архітектуру з використанням патерну MVVM, що сприяє зручному розділенню коду, його тестованості та масштабованості. Значну увагу було приділено користувацькому інтерфейсу: інтуїтивна навігація, підтримка камер і карти, інтерактивні елементи та адаптивний дизайн роблять взаємодію із застосунком простою і доступною. Користувач може за кілька кроків зробити фото тварини, додати опис, автоматично зафіксувати координати та опублікувати запис, що відображається на інтерактивній мапі в режимі реального часу.

Особливу роль у застосунку відіграє геолокаційний функціонал, реалізований як у клієнтській, так і у серверній частині. Координати широти й довготи зберігаються для кожного повідомлення, що дозволяє фільтрувати записи за місцем, будувати карти пошуку, визначати зону поширення проблеми або маршрути пересування тварин. У проєкті також було враховано питання безпеки — автентифікація реалізована через JWT-токени, що гарантує захищену роботу з персональними даними.

Після завершення розробки було проведено комплексне тестування всіх функціональних модулів. Ретельно перевірено основні сценарії використання, такі як реєстрація, створення звіту, робота з фото та картою, пошук і перегляд інформації, сповіщення та комунікація між користувачами. Результати тестування підтвердили відповідність реалізації заявленим вимогам і стабільну

роботу системи у штатному режимі. Також перевірено реакцію системи на помилкові запити, некоректні введення та інші граничні ситуації, що підтвердило її надійність.

Загалом, розроблений мобільний застосунок PetCompas є повноцінним програмним рішенням, яке може бути використане в реальних умовах, зокрема волонтерами, притулками для тварин, ветеринарними службами та небайдужими громадянами. Його функціональність охоплює всі основні потреби у сфері фіксації, пошуку та комунікації щодо безпритульних тварин. Гнучка архітектура, чистий і підтримуваний код, сучасний технічний стек, адаптивний інтерфейс та високий рівень безпеки забезпечують не лише ефективну поточну роботу, а й потенціал до подальшого розвитку.

У перспективі розробку можна доповнити новими модулями: реалізувати механізм верифікації записів через спільноту, інтегрувати нейромережі для точнішого розпізнавання породи тварини, додати розсилки або чат між користувачами, а також розробити веб-інтерфейс для організацій. Також можливе масштабування платформи для використання у різних містах, регіонах або країнах.

Таким чином, виконання цієї дипломної роботи дозволило здобути цілісний досвід з проєктування, реалізації та тестування складного програмного продукту соціального призначення, що демонструє як технічні навички розробника, так і розуміння важливості цифрових інструментів у вирішенні актуальних суспільних проблем. Результат є практично значущим, технологічно обґрунтованим та має потенціал подальшого масштабування і впровадження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Безпритульні тварини: проблема та можливі шляхи її вирішення. URL: https://novynar.city/articles/381585/bezpritulni-tvarini-problema-ta-mozhlivi-shlyahi-ii-virishennya (дата звернення 03.01.2025)

2. Домашні тварини та штучний інтелект: Як технології допомагають нашими улюбленцями URL: https://zoohub.ua/blog/domashni-tvarini-ta-shtuchnij-intelekt (дата звернення 03.01.2025)

3. Що таке .NET MAUI? URL: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui (дата звернення 05.01.2025)

4. Deciding Whether to Use MediatR in Your .NET Project. URL: https://www.featbit.co/developers/dn-deciding-whether-to-use-mediatr-in-net-project (дата звернення 08.01.2025)

5. ASP.NET Web API overview. URL: https://learn.microsoft.com/en-us/aspnet/web-api/overview/getting-started-with-aspnet-web-api (дата звернення 10.01.2025)

6. What is Entity Framework Core? URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення 11.01.2025)

7. Досвід Європейських країн у ідентифікації та реєстрації домашніх тварин. URL: https://www.agro-id.gov.ua/dosvid-yevropejskix-kra%D1%97n-u-identifikaci%D1%97-ta-reyestraci%D1%97-domashnix-tvarin/ (дата звернення 11.01.2025).

8. PetFinder – About the project. URL: https://www.petfinder.com/about/ (дата звернення 12.01.2025)

9. FindPet – AI for pet recovery. URL: https://findpet.com/ (дата звернення 12.01.2025)

10. PawBoost – Lost & Found pet alerts. URL: https://www.pawboost.com/ (дата звернення 12.01.2025)

11. Animal Shelters and Rescues in Ukraine – Directory. URL: https://happypaw.ua/ua (дата звернення 12.01.2025)

12. Clean Architecture – Robert C. Martin. URL: https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html (дата звернення 12.01.2025)

13. AutoMapper in .NET – Official Documentation. URL: https://automapper.org/ (дата звернення 14.01.2025)

14. JSON Web Tokens – Introduction. URL: https://jwt.io/introduction (дата звернення 16.01.2025)

15. .NET MAUI Platform Features: https://stfalcon.com/uk/blog/post/How-to-Write-a-Software-Requirements-Specification (дата звернення 18.01.2025)

16. How to use Geolocation in .NET MAUI. URL: https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/device/geolocation (дата звернення 19.01.2025)

17. What is MVVM pattern? URL: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm (дата звернення 22.01.2025)

18. YOLO – You Only Look Once: Real-Time Object Detection. URL: https://pjreddie.com/darknet/yolo/ (дата звернення 23.01.2025)

19. Vision Transformers (ViT): A Survey. URL: https://arxiv.org/abs/2010.11929 (дата звернення 24.01.2025)

20. EfficientNet: Rethinking Model Scaling. URL: https://arxiv.org/abs/1905.11946 (дата звернення 25.01.2025)

21. Розбираємо ази архітектури й нефункціональні вимоги. URL: https://dou.ua/forums/topic/49766/ (дата звернення 25.01.2025)

22. Жученко, А. І. Основи проектування баз даних [Електронний ресурс] : навчальний посібник для студентів спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» / А. І. Жученко, Л. Д. Ярощук ; НТУУ

«КПІ». – Електронні текстові дані (1 файл: 2, 50 Мбайт). – Київ : НТУУ «КПІ», 2015. – 158 с. – Назва з екрана. (дата звернення 26.01.2025)

23. REST API: що це, як працює, переваги і приклади. URL: https://goit.global/ua/articles/rest-api-shcho-tse-iak-pratsiuie-perevahy-i-pryklady/ (дата звернення 28.01.2025)

24. MAUI Navigation – Shell, Pages, and Views. URL: https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/shell/ (дата звернення 29.01.2025)

25. Dependency Injection in .NET. URL: https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection (дата звернення 02.02.2025)

26. Test automation in .NET – unit & integration testing. URL: https://learn.microsoft.com/en-us/dotnet/core/testing/ (дата звернення 03.02.2025)

27. Особливості тестування додатків на мобільних пристроях. URL: https://training.qatestlab.com/blog/technical-articles/testing-mobile-devices/ (дата звернення 04.02.2025)

28. C# Best Practices. URL: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions (дата звернення 05.02.2025)

29. Програмування в стилі Domain-Driven Design. URL: https://martinfowler.com/bliki/DomainDrivenDesign.html (дата звернення 12.02.2025)

30. The Result Pattern in C#: A comprehensive guide URL: https://www.linkedin.com/pulse/result-pattern-c-comprehensive-guide-andre-baltieri-wieuf/ (дата звернення 14.02.2025)

31. LanguageExt – Functional programming in C#. URL: https://github.com/louthy/language-ext (дата звернення 16.02.2025)

32. Google Maps API Documentation. URL: https://developers.google.com/maps/documentation (дата звернення 16.02.2025)

33. Android UI/UX: Інструменти, принципи та рекомендації. URL: https://foxminded.ua/android-ui-ux-guidelines/ (дата звернення 18.02.2025)

34. Що таке Unit тести і як їх писати. URL: https://foxminded.ua/yunit-testy/ (дата звернення 14.03.2025)

35. Поширені патерни у C#. URL: https://refactoring.guru/design-patterns/csharp (дата звернення 19.03.2025)

36. Камера в .NET MAUI – офіційна документація. URL: https://learn.microsoft.com/en-us/dotnet/communitytoolkit/maui/views/camera-view?tabs=android (дата звернення 22.03.2025)

37. Використання карт в мобільних застосунках. URL: https://developer.android.com/training/maps (дата звернення 22.03.2025)

38. The CleanArchitecture. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (дата звернення 22.03.2025)

39. Основи UI/UX: дизайн-процес для початківців https://dou.ua/forums/topic/42880/ (дата звернення 12.04.2025)

40. Принципи SOLID. Навіщо вони потрібні програмістам та як з ними працювати. URL: https://campus.epam.ua/ua/blog/602 (дата звернення 13.04.2025)

41. Досвід Європейських країн у ідентифікації та реєстрації домашніх тварин. URL: https://www.agro-id.gov.ua/dosvid-yevropejskix-kra%D1%97n-u-identifikaci%D1%97-ta-reyestraci%D1%97-domashnix-tvarin/ (дата звернення 15.04.2025).

ДОДАТОК А

(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Назва проєкту:

Мобільний застосунок для організації допомоги безпритульним тваринам

Мета розробки:

Створення мультиплатформного застосунку, який забезпечить взаємодію між волонтерами, притулками та звичайними користувачами з метою надання допомоги безпритульним тваринам, оптимізації волонтерської діяльності та підвищення обізнаності про проблему.

Обґрунтування розробки застосунку

Протягом останніх років зростає проблема безпритульних тварин. Існуючі цифрові рішення не охоплюють весь спектр потреб для організації допомоги таким тваринам. Розробка універсального інструменту дозволить:

- ефективно координувати зусилля волонтерів;
- надавати інформацію про тварин у реальному часі;
- залучати небайдужих громадян;
- структурувати роботу притулків.

Призначення розробки

Застосунок надає можливість бачити на мапі тварин, які потребують допомоги, фільтрувати їх за видом, статусом, місцем перебування. Користувач може переглядати інформацію про конкретну тварину — її фото, опис, стан здоров’я, контактні дані відповідального волонтера чи притулку. Також доступна

можливість створення власних оголошень про знайдених або загублених тварин. Для волонтерів і працівників притулків реалізовано окремі функції — додавання тварин до бази, ведення звітів, публікація запитів на допомогу. У кожного користувача є особистий кабінет з історією його активностей, а також доступні базові функції профілю — налаштування, мова, фото, контактні дані.

Цільова аудиторія

Цей застосунок орієнтований на активних волонтерів, співробітників притулків, а також на звичайних громадян, які знаходять тварин і бажають повідомити про це або допомогти. Водночас застосунок є корисним для людей, які загубили власну тварину і шукають її через карту або список оголошень.

Технічна реалізація

Проєкт реалізується на базі .NET MAUI з використанням шаблону MVVM і принципів чистої архітектури. Презентаційна логіка розділена від бізнес-логіки, що дозволяє легко підтримувати та масштабувати застосунок. В якості хмарного бекенду використовується Azure DB, а для відображення тварин на мапі — Google Maps API. Для реалізації реактивної логіки використовується бібліотека ReactiveUI, а для зручної ін’єкції залежностей — .NET Community Toolkit. Код пишеться мовою C# з використанням XAML для UI, а також застосовується бібліотека LanguageExt для функціонального підходу в обробці даних.

3 Вимоги до затосунку

3.1 Функціональні вимоги

– додавання інформації про знайдених тварин, включаючи фото, опис, стан здоров'я та особливі прикмети. Користувачі повинні мати змогу швидко завантажувати зображення тварини та додавати описові дані, які допоможуть іншим у її пошуку.

– автоматичне визначення геолокації під час завантаження фотографій. Це дозволить уникнути необхідності вручну вводити адресу та зробить систему більш зручною для користувачів.

– інтеграція карти для відображення місць перебування тварин у реальному часі. Всі додані записи повинні автоматично відображатися на інтерактивній карті, що допоможе швидко визначати місцезнаходження тварини.

– механізм реєстрації та авторизації користувачів. Реєстрація може бути реалізована через email або соціальні мережі, а система авторизації повинна забезпечити безпечний доступ до персонального кабінету.

– можливість оновлення або видалення власних записів. Користувачі зможуть редагувати деталі про тварину або видаляти інформацію, якщо тварину вже знайдено або їй надано допомогу.

– пошук тварин за критеріями: місце знаходження, час публікації, тип тварини. Це дозволить швидше знаходити потрібну інформацію та уникати перегляду нерелевантних записів.

– взаємодія користувачів через коментарі або оновлення статусу знайдених тварин. Це дасть змогу координувати зусилля та спрощувати процес надання допомоги.

– інтеграція нейронної мережі [18 - 19] для автоматичного визначення виду тварини за фото. Система повинна використовувати алгоритми машинного навчання, щоб розпізнавати породу або вид тварини, що підвищить ефективність платформи.

3.2 Нефункціональні вимоги

– оптимізація продуктивності для швидкої обробки запитів і мінімального часу відгуку [20]. Серверна частина повинна бути розроблена з урахуванням високої швидкодії та використання кешування для оптимізації завантаження.

– стабільна робота при високих навантаженнях. Платформа повинна витримувати значну кількість одночасних запитів без зниження продуктивності.

– безпечна авторизація користувачів із використанням стандартних протоколів аутентифікації. Використання OAuth, JWT або інших сучасних методів захистить персональні дані користувачів.

– шифрування переданих даних для захисту персональної інформації. Усі передані дані повинні бути захищені за допомогою протоколів HTTPS та шифрування бази даних.

– гнучкість архітектури для можливості розширення функціоналу [11, 28]. Майбутні оновлення системи повинні легко інтегрувати нові можливості без серйозних змін у коді.

– адаптивний дизайн [32, 38] для роботи на різних пристроях та підтримка кількох мов. Інтерфейс повинен бути зручним як для користувачів ПК, так і для власників смартфонів, а також підтримувати кілька мов для охоплення більшої аудиторії.

– механізми моніторингу та логування для оперативного виявлення та усунення проблем. Адміністративна панель повинна включати інструменти для відстеження активності системи та усунення можливих збоїв.

3.3 Вимоги до середовища виконання

Мобільний застосунок призначений для використання на сучасних смартфонах та планшетах під управлінням операційних систем Android та iOS. Він забезпечує повноцінну функціональність у кросплатформному середовищі завдяки використанню .NET MAUI.

Застосунок підтримує встановлення лише однієї копії на пристрої. Використання сторонніх засобів для клонування або модифікації програми може призвести до нестабільної роботи, некоректного збереження даних або втрати доступу до функцій.

Для коректної роботи основних можливостей необхідне стабільне з’єднання з Інтернетом. Однак деякі функції, як-от перегляд профілів тварин або історії взаємодій, доступні в офлайн-режимі, з подальшою синхронізацією після відновлення підключення.

Мобільний застосунок оптимізований під сучасні пристрої та операційні системи. Проте на момент першого випуску можлива часткова різниця у функціональності між Android та iOS через особливості реалізації та обмеження платформи.

Мінімальні системні вимоги для Android:

- Операційна система: Android 8.0 (Oreo) або вище
- Процесор: від 1.4 ГГц, архітектура ARMv8 або вище
- Оперативна пам’ять (RAM): не менше 2 ГБ
- Вільне місце на пристрої: не менше 300 МБ
- Додатково: доступ до камери, геолокації, Інтернету

Мінімальні системні вимоги для iOS:

- Операційна система: iOS 13 або вище
- Підтримувані пристрої: iPhone 8 / iPad (6-го покоління) або новіші
- Оперативна пам’ять (RAM): не менше 2 ГБ
- Вільне місце на пристрої: не менше 300 МБ
- Додатково: доступ до камери, геолокації, Інтернету

Застосунок використовує Google Maps API, Azure, MSSQL та інші сучасні інструменти, які вимагають сумісності з відповідними версіями ОС та доступу до служб локації. На старих пристроях можливе зниження продуктивності, довший час запуску або обмежена підтримка деяких функцій.

3.4 Вимоги до інформаційної та програмної сумісності

Мобільний застосунок для допомоги безпритульним тваринам розроблено на основі кросплатформного фреймворку .NET MAUI, що забезпечує підтримку

декількох операційних систем із єдиної кодової бази. Завдяки використанню .NET MAUI та архітектури MVVM, застосунок підтримує сучасні вимоги до адаптивного інтерфейсу, забезпечуючи сумісність із різними типами пристроїв, включаючи смартфони, планшети та настільні системи.

Підтримувані платформи та версії ОС:

- Android: підтримка Android 8.0 (API 26) і вище.
- iOS: підтримка iOS 13.0 і вище (оптимізовано для iPhone та iPad).
- Windows: підтримка Windows 10 версії 1809 і вище.
- macOS: підтримка macOS 10.15 (Catalina) і вище (залежно від конфігурації .NET MAUI).

Інформаційна сумісність та зберігання даних:

Інші сервіси та API:

- Google Maps API інтегровано для відображення місця знаходження тварин на карті, а також для геолокації оголошень.
- LanguageExt застосовується для підвищення безпеки та стабільності логіки за допомогою функціонального підходу.

Завдяки цим технологіям забезпечується:

- Повноцінна підтримка хмарної синхронізації.
- Надійна передача та зберігання мультимедійних даних.
- Висока продуктивність і стабільність незалежно від типу платформи.

4 Вимоги до програмної документації

Під час передачі проєкту замовнику надається наступний комплект документації:

– вихідний код програми з детальними коментарями;

– опис функціональних можливостей застосунку;

– технічне завдання (цей документ);

– інструкція для користувача.

5 Стадії та етапи розробки

№	Стадія	Етап	Опис	Результат
1	Підготовча	Аналіз проблеми	Вивчення потреб користувачів і проблеми безпритульних тварин	Визначені цілі та завдання проєкту
		Формування вимог	Збір та систематизація функціональних та нефункціональних вимог	Технічне завдання
		Вибір технологій	Обґрунтування використання .NET MAUI, MSSQL, Google Maps API, LanguageExt, RxUI	Визначений технологічний стек
2	Проєктування	Архітектурне проєктування	Створення структури застосунку згідно з принципами Clean Architecture та MVVM	Схема архітектури та залежностей між модулями
		Проєктування інтерфейсу користувача (UI/UX)	Створення мокапів, прототипів, макетів інтерфейсу	Інтерактивні прототипи екранів застосунку
3	Реалізація (розробка)	Розробка доменного шару	Створення сутностей,	Реалізований Domain Layer

			інтерфейсів репозиторіїв та use case-ів	
		Розробка шару даних	Інтеграція MSSQL, реалізація репозиторіїв	Реалізований Data Layer
		Розробка шару презентації	Створення XAML-інтерфейсів, ViewModel-ів, підключення Google Maps API	Реалізований Presentation Layer
		Налаштування інфраструктури	Інтеграція бібліотек, створення служб, організація DI	Готовий Infrastructure Layer
4	Тестування	Модульне тестування	Розробка unit-тестів для бізнес-логіки	Перевірена правильність окремих модулів
		Інтеграційне тестування	Тестування взаємодії між шарами	Виявлені та виправлені помилки взаємодії компонентів
		Тестування інтерфейсу користувача	Перевірка юзабіліті, адаптивності, логіки відображення	Оптимізований та зручний UI

5	Розгортання та публікація	Збірка застосунку	Компіляція застосунку під Android та iOS	APK та IPA-файли
		Публікація	Завантаження в Google Play і App Store	Застосунок доступний користувачам
6	Підтримка та розвиток	Збір відгуків користувачів	Аналіз фідбеку для вдосконалення функціональності	Перелік покращень і нових фіч
		Виправлення помилок та оновлення	Реліз патчів, додавання нових можливостей	Актуальна та стабільна версія застосунку

Таблиця А.1 – Стадії та етапи розробки проєкту «PetCompass»

6 Порядок контролю та приймання:

Види випробувань:

– Юніт-тестування: перевірка окремих елементів бізнес-логіки, таких як use case-и, сервіси та функціональні модулі, реалізовані у доменному шарі з використанням LanguageExt для функціонального підходу.

– Модульне тестування (component testing): перевірка роботи окремих компонентів інтерфейсу, ViewModel-ів та їх взаємодії з користувачем у межах MVVM-архітектури.

– Інтеграційне тестування: оцінка взаємодії між шарами (presentation, domain, data)

– Тестування інтерфейсу користувача (UI testing): перевірка коректності відображення екранів, навігації між сторінками, взаємодії з Google Maps API та реактивного оновлення інтерфейсу.

– Тестування з кінця в кінець (end-to-end testing): перевірка повного користувацького сценарію, включаючи реєстрацію, вхід, додавання тварини, перегляд на мапі, створення оголошення та взаємодію з волонтерами.

– Тестування стабільності та відмовостійкості: перевірка поведінки застосунку у складних умовах, зокрема при нестабільному інтернет-з'єднанні.

Вимоги до приймання:

– відповідність вимогам: застосунок повинен відповідати функціональним і нефункціональним вимогам, зазначеним у технічному завданні.

– відмовостійкість та стабільність: застосунок має бути стійким до відмов, коректно працювати на різних пристроях та в різних мережевих умовах.

– функціональне тестування: перевірка коректності основних функцій, включаючи реєстрацію та авторизацію користувачів, управління профілями тварин, запис на прийом до ветеринара, налаштування нагадувань.

– відповідність документації: усі документи, інструкції та довідкові матеріали повинні бути актуальними, зрозумілими та повністю відповідати функціоналу застосунку.

Технологічний стек:

– Фреймворк розробки інтерфейсу: .NET MAUI (Multi-platform App UI)

– Архітектура: Clean Architecture, MVVM

– Стан управління:.NET Community Toolkit MVVM

– Функціональне програмування: LanguageExt

– База даних та автентифікація: MSSQL Authentication, JWT

– Зберігання медіа: Azure Blob Storage

– Карти та геолокація: Google Maps API

– Інструменти тестування: xUnit, Moq

ДОДАТОК Б

(обов'язковий)

КОД ПРОГРАМИ

Сервер:

```
AnimalController:
    [Route("api/animals")]
    [ApiController]
    public class AnimalController(IAnimalRepository animalRepository, IMediator
mediator) : ControllerBase
    {
        private readonly IMediator _mediator = mediator;


        [HttpPost]
        public async Task<IActionResult> Add(CreateAnimalDTO createAnimalDTO)
        {
            Either<Error, Unit> result = await _mediator.Send(new
AddAnimalCommand(createAnimalDTO));

            return result.Match<IActionResult>(
                animal => Created(),
                ex => BadRequest(ex.Message)
            );
        }


        [HttpGet("{id}")]
        public async Task<IActionResult> Get(int id)
        {
            Either<Error, Option<Animal>> result = await _mediator.Send(new
GetAnimalQuery(id));

            return result.Match(
                option => option.Match<IActionResult>(
                    animal => Ok(animal),
                    () => NotFound()
                ),
                err => BadRequest(err.Message)
            );
        }

        [HttpGet("sorted/{sortOption}")]
        public async Task<IActionResult> GetSorted([FromRoute] SortOption
sortOption)
        {
            Either<Error, IEnumerable<Animal>> result = await _mediator.Send(new
GetSortedAnimalsQuery(sortOption));

            return result.Match<IActionResult>(
                animalsEnumerable =>
                {
                    List<GetAnimalResponse> animals = new
List<GetAnimalResponse>();
```

```
                    foreach (Animal animal in animalsEnumerable)
                    {

animals.Add(AnimalToDTOMapper.AnimalToGetAnimalResponse(animal));
                    }

                    return Ok(animals);
                },
                ex => BadRequest(ex.Message)
            );
        }

        [HttpGet("sorted/closest")]
        public async Task<IActionResult> GetClosest([FromQuery] double latitude,
[FromQuery] double longitude)
        {
            if (latitude == 0 || longitude == 0)
            {
                return BadRequest("Empty coordinates.");
            }

            Either<Error, IEnumerable<Animal>> result = await _mediator.Send(new
GetClosestAnimalsQuery(new UserLocationDTO(longitude, latitude)));

            return result.Match<IActionResult>(
                animalsEnumerable =>
                {
                    List<GetAnimalResponse> animals = new
List<GetAnimalResponse>();

                    foreach (Animal animal in animalsEnumerable)
                    {

animals.Add(AnimalToDTOMapper.AnimalToGetAnimalResponse(animal));
                    }

                    return Ok(animals);
                },
                ex => BadRequest(ex.Message)
            );
        }

        [HttpGet("locations")]
        public async Task<IActionResult> GetAnimalLocations()
        {
            Either<Error, IEnumerable<MapAnimalDTO>> result = await
_mediator.Send(new GetAnimalLocationsQuery());

            return result.Match<IActionResult>(
                animalsEnumerable =>
                {
                    return Ok(animalsEnumerable);
                },
                ex => BadRequest(ex.Message)
            );
        }

        [HttpPut]
        public async Task<IActionResult> Update(Animal animal)
        {
```

```
            Either<Error, LanguageExt.Unit> result = await _mediator.Send(new
UpdateAnimalCommand(animal));

            return result.Match<IActionResult>(
                updatedAnimal => Ok(),
                ex => BadRequest(ex.Message));
        }

        [HttpDelete("{animalId}")]
        public async Task<IActionResult> Delete([FromRoute] int animalId)
        {
            Either<Error, LanguageExt.Unit> result = await _mediator.Send(new
DeleteAnimalQuery(animalId));

            return result.Match<IActionResult>(
                success => Ok(),
                ex => BadRequest(ex.Message));
        }

        [HttpGet]
        public async Task<IActionResult> All()
        {
            Either<Error, IEnumerable<Animal>> result = await _mediator.Send(new
GetAllAnimalQuery());

            return result.Match<IActionResult>(
                animalsEnumerable =>
                {
                    List<GetAnimalResponse> animals = new
List<GetAnimalResponse>();

                    foreach (Animal animal in animalsEnumerable)
                    {

animals.Add(AnimalToDTOMapper.AnimalToGetAnimalResponse(animal));
                    }

                    return Ok(animals);
                },
                ex => BadRequest(ex.Message)
            );
        }
    }
}
```

UserController:

```
namespace PetCompas.Controllers
{
    [Route("api/users")]
    [ApiController]
    [TypeFilter(typeof(ApiKeyAttribute))]
    public class UserController : ControllerBase
    {
        private readonly IConfiguration _configuration;
        private readonly IMediator _mediator;
```

```
        public UserController(IConfiguration configuration, IMediator mediator)
        {
            _mediator = mediator;
            _configuration = configuration;
        }

        [HttpPost("registration")]
        public async Task<IActionResult> Register(RegistrationRequest
registrationRequest)
        {
            Either<Error, User> result = await _mediator.Send(new
RegistrationCommand(registrationRequest));

            return result.Match<IActionResult>(
                user => Created("User created.", user.Id),
                ex => BadRequest(ex));
        }

        [HttpPost("login")]
        public async Task<IActionResult> Login(LoginRequest request)
        {
            Either<Error, string> result = await _mediator.Send(new
LoginCommand(request));

            string secretKey = _configuration["XApiKey"];

            return result.Match<IActionResult>(
                token => Ok(new AuthenticationResponse(token, secretKey)),
                ex => Unauthorized(new { ex.Message })
            );
        }

        [HttpPut]
        [Authorize]
        public async Task<IActionResult> Update(User user)
        {
            UpdateUserCommand command = new UpdateUserCommand(user);
            Either<Error, LanguageExt.Unit> result = await
_mediator.Send(command);

            return result.Match<IActionResult>(
                updatedUser => Ok(),
                ex => BadRequest(ex));
        }

        [HttpGet("{id}")]
        [Authorize]
        public async Task<IActionResult> Get(int id)
        {
            Either<Error, Option<User>> result = await _mediator.Send(new
GetUserQuery(id));

            return result.Match(
                option => option.Match<IActionResult>(
                    user => Ok(user),
                    () => NotFound()
                ),
                err => BadRequest(err.Message)
            );
        }
```

```
        [HttpDelete("{id}")]
        [Authorize]
        public async Task<IActionResult> Delete(int id)
        {
            Either<Error, LanguageExt.Unit> result = await _mediator.Send(new
DeleteUserCommand(id));

            return result.Match<IActionResult>(
                success => Ok(),
                ex => BadRequest(ex));
        }
    }
}
```

Модель Animal:

```
namespace PetCompas.Domain.Models
{
    public class Animal
    {
        public int Id { get; set; }

        [Required]
        public string MainPhoto { get; set; }

        public string Description { get; set; }

        public AnimalType AnimalType { get; set; }

        public AnimalSize Size { get; set; }

        public AnimalStatus Status { get; set; }

        public AnimalGender Gender { get; set; }

        [Required]
        public int UserId { get; set; }

        [Required]
        public int LocationId { get; set; }

        public DateTime CreatedAt { get; set; }

        [ForeignKey("LocationId")]
        public virtual Location? Location { get; set; }

        [ForeignKey("UserId")]
        public virtual User? User { get; set; }

        public virtual ICollection<AnimalPhoto>? Photos { get; set; }
    }
}
```

Клієнт:

AnimalService:

```
namespace PetCompassMobile.Services.Network.Animal;

public class AnimalService(IHttpClientFactory httpClientFactory) :
NetworkServiceBase(httpClientFactory, NetworkConfig.BaseUrl), IAnimalService
{
    public async Task<Result<string, string>> Create(CreateAnimalDTO animal,
CancellationToken cancellationToken = default)
    {
        Result<string, string> serverResponse = await
PerformCreateAnimal(animal, cancellationToken);
        return serverResponse.Match(
            content =>
            {
                return Result<string, string>.Success(content);
            },
            error =>
            {
                return Result<string, string>.Failure(error);
            }
        );
    }

    public async Task<Result<string, string>> ProcessImage(string base64Image,
CancellationToken cancellationToken = default)
    {
        ImageRequest imageRequest =
NetworkServiceHelper.CreateImageRequest(base64Image);
        Result<string, string> serverResponse = await
PerformProcessImage(imageRequest, cancellationToken);
        return serverResponse.Match(
            content =>
            {
                return Result<string, string>.Success(content);
            },
            error =>
            {
                return Result<string, string>.Failure(error);
            }
        );
    }

    public async Task<Result<string, string>> GetSorted(SortOption sortOption,
CancellationToken cancellationToken = default)
    {
        Result<string, string> serverResponse = await
PerformGetSorted(sortOption, cancellationToken);
        return serverResponse.Match(
            content =>
            {
                return Result<string, string>.Success(content);
            },
            error =>
            {
                return Result<string, string>.Failure(error);
            }
        );
    }

    public async Task<Result<string, string>> Get(int id, CancellationToken
cancellationToken = default)
    {
```

```
        Result<string, string> serverResponse = await PerformGetAnimal(id,
cancellationToken);
        return serverResponse.Match(
            content =>
            {
                return Result<string, string>.Success(content);
            },
            error =>
            {
                return Result<string, string>.Failure(error);
            }
        );
    }

    public async Task<Result<string, string>> GetAll(CancellationToken
cancellationToken = default)
    {
        Result<string, string> serverResponse = await
PerformGetAllAnimals(cancellationToken);
        return serverResponse.Match(
            content =>
            {
                return Result<string, string>.Success(content);
            },
            error =>
            {
                return Result<string, string>.Failure(error);
            }
        );
    }

    public async Task<Result<string, string>> GetLocations(CancellationToken
cancellationToken = default)
    {
        Result<string, string> serverResponse = await
PerformGetLocations(cancellationToken);
        return serverResponse.Match(
            content =>
            {
                return Result<string, string>.Success(content);
            },
            error =>
            {
                return Result<string, string>.Failure(error);
            }
        );
    }

    public async Task<Result<string, string>> Delete(int id, CancellationToken
cancellationToken = default)
    {
        Result<string, string> serverResponse = await PerformDelete(id,
cancellationToken);
        return serverResponse.Match(
            content =>
            {
                return Result<string, string>.Success(content);
            },
            error =>
            {
                return Result<string, string>.Failure(error);
```

```
            }
        );
    }

    public async Task<Result<string, string>> GetClosest(double longitude,
double latitude, CancellationToken cancellationToken = default)
    {
        Result<string, string> serverResponse = await
PerformGetClosest(longitude, latitude, cancellationToken);
        return serverResponse.Match(
            content =>
            {
                return Result<string, string>.Success(content);
            },
            error =>
            {
                return Result<string, string>.Failure(error);
            }
        );
    }


    private async Task<Result<string, string>> PerformGetAnimal(int id,
CancellationToken cancellationToken)
    {
        return await ResultExtensions.RunTryAsync(async () =>
            {
                using var request = new HttpRequestMessage(HttpMethod.Get,
NetworkConfig.GetAnimal(id));

                request.Headers.Add(nameof(ApiConnectionConst.XApiKey),
ApiConnectionConst.XApiKey);

                using HttpResponseMessage response = await
HttpClient.SendAsync(request, cancellationToken);
                response.EnsureSuccessStatusCode();

                return await
response.Content.ReadAsStringAsync(cancellationToken);
            })
            .MatchAsync(
                content => Task.FromResult(Result<string,
string>.Success(content)),
                ex =>
                {
                    string errorMessage =
ex.Contains(ErrorMessages.TimeOutException)
                        ? "The server is not responding. Please try again
later."
                        : $"Request failed: {ex}";

                    return Task.FromResult(Result<string,
string>.Failure(errorMessage));
                }
            );
    }

    private async Task<Result<string, string>>
PerformGetAllAnimals(CancellationToken cancellationToken)
    {
        return await ResultExtensions.RunTryAsync(async () =>
```

```
            {

                using var request = new HttpRequestMessage(HttpMethod.Get,
NetworkConfig.GetAllAnimals);

                request.Headers.Add(nameof(ApiConnectionConst.XApiKey),
ApiConnectionConst.XApiKey);

                using HttpResponseMessage response = await
HttpClient.SendAsync(request, cancellationToken);
                response.EnsureSuccessStatusCode();

                return await
response.Content.ReadAsStringAsync(cancellationToken);
            })
            .MatchAsync(
                result => Task.FromResult(Result<string,
string>.Success(result)),
                ex =>
                {
                    string errorMessage =
ex.Contains(ErrorMessages.TimeOutException)
                        ? "The server is not responding. Please try again
later."
                        : $"Request failed: {ex}";

                    return Task.FromResult(Result<string,
string>.Failure(errorMessage));
                }
            );
    }

    private async Task<Result<string, string>>
PerformGetLocations(CancellationToken cancellationToken)
    {
        return await ResultExtensions.RunTryAsync(async () =>
        {
            using var request = new HttpRequestMessage(HttpMethod.Get,
NetworkConfig.GetLocations);

            request.Headers.Add(nameof(ApiConnectionConst.XApiKey),
ApiConnectionConst.XApiKey);

            using HttpResponseMessage response = await
HttpClient.SendAsync(request, cancellationToken);
            response.EnsureSuccessStatusCode();

            return await response.Content.ReadAsStringAsync(cancellationToken);
        })
            .MatchAsync(
                result => Task.FromResult(Result<string,
string>.Success(result)),
                ex =>
                {
                    string errorMessage =
ex.Contains(ErrorMessages.TimeOutException)
                        ? "The server is not responding. Please try again
later."
                        : $"Request failed: {ex}";
```

```
                    return Task.FromResult(Result<string,
string>.Failure(errorMessage));
                }
            );
    }

    private async Task<Result<string, string>> PerformGetSorted(SortOption
sortOption,CancellationToken cancellationToken)
    {
        return await ResultExtensions.RunTryAsync(async () =>
            {
                using var request = new HttpRequestMessage(HttpMethod.Get,
NetworkConfig.GetSorted(sortOption));

                request.Headers.Add(nameof(ApiConnectionConst.XApiKey),
ApiConnectionConst.XApiKey);

                using HttpResponseMessage response = await
HttpClient.SendAsync(request, cancellationToken);
                response.EnsureSuccessStatusCode();

                return await
response.Content.ReadAsStringAsync(cancellationToken);
            })
            .MatchAsync(
                result => Task.FromResult(Result<string,
string>.Success(result)),
                ex =>
                {
                    string errorMessage =
ex.Contains(ErrorMessages.TimeOutException)
                        ? "The server is not responding. Please try again
later."
                        : $"Request failed: {ex}";

                    return Task.FromResult(Result<string,
string>.Failure(errorMessage));
                }
            );
    }

    private async Task<Result<string, string>>
PerformCreateAnimal(CreateAnimalDTO animal, CancellationToken cancellationToken)
    {
        return await ResultExtensions.RunTryAsync(async () =>
            {
                using var request = new HttpRequestMessage(HttpMethod.Post,
NetworkConfig.CreateAnimal);

                request.Headers.Add(nameof(ApiConnectionConst.XApiKey),
ApiConnectionConst.XApiKey);

                request.Content = JsonContent.Create(animal);

                using HttpResponseMessage response = await
HttpClient.SendAsync(request, cancellationToken);
                response.EnsureSuccessStatusCode();

                return await
response.Content.ReadAsStringAsync(cancellationToken);
            })
```

```
            .MatchAsync(
                result => Task.FromResult(Result<string,
string>.Success(result)),
                ex =>
                {
                    string errorMessage =
ex.Contains(ErrorMessages.TimeOutException)
                        ? "The server is not responding. Please try again
later."
                        : $"Request failed: {ex}";

                    return Task.FromResult(Result<string,
string>.Failure(errorMessage));
                }
            );
    }

    private async Task<Result<string, string>> PerformProcessImage(ImageRequest
imageRequest, CancellationToken cancellationToken)
    {
        return await ResultExtensions.RunTryAsync(async () =>
        {
            var requestBody = new
            {
                image = imageRequest.Base64Image
            };

            using var request = new HttpRequestMessage(HttpMethod.Post,
NetworkConfig.YoloApiUrl);
            request.Content = new StringContent(
                JsonSerializer.Serialize(requestBody),
                Encoding.UTF8,
                "application/json"
            );

            using HttpResponseMessage response = await
HttpClient.SendAsync(request, cancellationToken);
            string responseContent = await
response.Content.ReadAsStringAsync(cancellationToken);

            if (!response.IsSuccessStatusCode)
            {
                return Result<string,
string>.Failure(ParseErrorMessage(responseContent));
            }

            YoloResponse resultObj =
JsonSerializer.Deserialize<YoloResponse>(responseContent);


            if (string.IsNullOrWhiteSpace(resultObj?.result))
            {
                return Result<string, string>.Failure("The animal was not
identified in the photo.");
            }

            return Result<string, string>.Success(resultObj.result);
        })
            .MatchAsync(
                Task.FromResult,
                ex =>
```

```
                {
                    string errorMessage =
ex.Contains(ErrorMessages.TimeOutException)
                        ? "The server is not responding. Please try again
later."
                        : $"Request failed: {ex}";

                    return Task.FromResult(Result<string,
string>.Failure(errorMessage));
                }
            );
    }

    private async Task<Result<string, string>> PerformGetClosest(double
latitude, double longitude, CancellationToken cancellationToken)
    {
        return await ResultExtensions.RunTryAsync(async () =>
        {
            using var request = new HttpRequestMessage(HttpMethod.Get,
NetworkConfig.GetClosest(latitude, longitude));

            request.Headers.Add(nameof(ApiConnectionConst.XApiKey),
ApiConnectionConst.XApiKey);

            using HttpResponseMessage response = await
HttpClient.SendAsync(request, cancellationToken);
            response.EnsureSuccessStatusCode();

            return await response.Content.ReadAsStringAsync(cancellationToken);
        })
            .MatchAsync(
                result => Task.FromResult(Result<string,
string>.Success(result)),
                ex =>
                {
                    string errorMessage =
ex.Contains(ErrorMessages.TimeOutException)
                        ? "The server is not responding. Please try again
later."
                        : $"Request failed: {ex}";

                    return Task.FromResult(Result<string,
string>.Failure(errorMessage));
                }
            );
    }

    private async Task<Result<string, string>> PerformDelete(int animalId,
CancellationToken cancellationToken)
    {
        return await ResultExtensions.RunTryAsync(async () =>
            {
                using var request = new HttpRequestMessage(HttpMethod.Delete,
NetworkConfig.DeleteAnimal(animalId));

                request.Headers.Add(nameof(ApiConnectionConst.XApiKey),
ApiConnectionConst.XApiKey);

                using HttpResponseMessage response = await
HttpClient.SendAsync(request, cancellationToken);
                response.EnsureSuccessStatusCode();
```

```
                return await
response.Content.ReadAsStringAsync(cancellationToken);
            })
            .MatchAsync(
                result => Task.FromResult(Result<string,
string>.Success(result)),
                ex =>
                {
                    string errorMessage =
ex.Contains(ErrorMessages.TimeOutException)
                        ? "The server is not responding. Please try again
later."
                        : $"Request failed: {ex}";

                    return Task.FromResult(Result<string,
string>.Failure(errorMessage));
                }
            );
    }


    private string ParseErrorMessage(string responseContent)
    {
        try
        {
            var errorResponse =
JsonSerializer.Deserialize<ErrorResponse>(responseContent, new
JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });

            return errorResponse?.Message ?? "Unknown error";
        }
        catch
        {
            return responseContent;
        }
    }
}
```

Головне представлення:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="PetCompassMobile.Views.Master.Main.MainView"
             xmlns:vm="clr-namespace:PetCompassMobile.Views.Master.Main"
             xmlns:animalListElement="clr-
namespace:PetCompassMobile.Views.Master.Main.AnimalListElement"
             x:DataType="vm:MainViewModel">

    <ContentView.Content>
        <Grid Padding="15,0"
              BackgroundColor="{StaticResource LightCream}"
              RowDefinitions="Auto, Auto, *">

            <Grid
                Grid.Row="1"
                Margin="0,0,0,10"
                ColumnDefinitions="*, *"
```

```
                ColumnSpacing="90">

                <Border
                    BackgroundColor="#FFDEAF"
                    Grid.Column="0"
                    Stroke="{StaticResource CharcoalBlack}"
                    StrokeThickness="0.5"
                    StrokeShape="RoundRectangle 5">
                    <Grid
                        ColumnDefinitions="Auto, *"
                        Padding="7">

                        <Grid.GestureRecognizers>
                            <TapGestureRecognizer
Tapped="TapGestureRecognizer_OnTapped"></TapGestureRecognizer>
                        </Grid.GestureRecognizers>

                        <Image
                            Grid.Column="0"
                            HeightRequest="24"
                            Source="arrange.png"
                            HorizontalOptions="Start"
                            Margin="0,0,5,0" />

                        <Label
                            VerticalOptions="Center"
                            Grid.Column="1"
                            FontAttributes="Bold"
                            HorizontalOptions="Start"
                            Text="Sort" />

                        <Picker x:Name="SortPicker"
                                Title="Sort by"
                                ItemsSource="{Binding SortOptions}"
                                SelectedItem="{Binding SelectedSortOption,
Mode=TwoWay}"

SelectedIndexChanged="SortPicker_OnSelectedIndexChanged"
                                WidthRequest="1"
                                MaximumHeightRequest="1"
                                Opacity="0"
                                IsEnabled="True"/>
                    </Grid>

                </Border>
            </Grid>

            <RefreshView IsRefreshing="{Binding IsRefreshing}"
                         Command="{Binding RefreshCommand}"
                         Grid.Row="2">
                <CollectionView x:Name="AnimalCollectionView"
                    IsVisible="{Binding IsNotBusy}"
                    ItemsSource="{Binding Animals}"
                    VerticalOptions="FillAndExpand">
                    <CollectionView.ItemTemplate>
                        <DataTemplate>
                            <animalListElement:AnimalListElement>

<animalListElement:AnimalListElement.GestureRecognizers>
                                    <TapGestureRecognizer
Tapped="OnAnimalTapped"/>
```

```
</animalListElement:AnimalListElement.GestureRecognizers>
                            </animalListElement:AnimalListElement>
                        </DataTemplate>
                    </CollectionView.ItemTemplate>
                </CollectionView>
            </RefreshView>


            <ActivityIndicator Grid.Row="2"
                               IsVisible="{Binding IsBusy}"
                               IsRunning="{Binding IsBusy}"
                               VerticalOptions="Center"
                               Color="{StaticResource CharcoalBlack}"
                               HorizontalOptions="Center">

            </ActivityIndicator>
        </Grid>
    </ContentView.Content>
</ContentView>
```

MainViewModel:

```
namespace PetCompassMobile.Views.Master.Map;

public partial class MainMapViewModel : ViewModelBase,
IRecipient<GeolocationMessage>
{
    private readonly IAnimalService _animalService;
    private readonly IGeolocationService _geolocationService;

    public ObservableCollection<MapAnimalDTO> Animals { get; } = new();

    [ObservableProperty] private Core.DTO.Animals.Location _currentMapFocus =
null;

    public Action? OnAnimalsUpdated { get; set; }
    public Action? OnMapFocusUpdate { get; set; }

    public MainMapViewModel(IServiceProvider serviceProvider, IAnimalService
animalService, IGeolocationService geolocationService) : base(serviceProvider)
    {
        _geolocationService = geolocationService;
        _animalService = animalService;
        WeakReferenceMessenger.Default.Register<GeolocationMessage>(this);
    }

    public async override Task OnAppearingAsync()
    {
        await base.OnAppearingAsync();
        await SetDefaultLocation();
        await InitializeMap();
    }

    public void Receive(GeolocationMessage message)
    {
        if(!message.Location.Equals(CurrentMapFocus))
        {
            CurrentMapFocus = message.Location;
            OnMapFocusUpdate?.Invoke();
```

```
        }
    }

    private async Task InitializeMap()
    {
        await RefreshAnimalList();
    }

    public async void NavToAnimalDetails(int id)
    {
        Result<string, string> animalResult = await _animalService.Get(id);

        animalResult.Match(
            animalJson =>
            {
                Animal animal =
JsonConvert.DeserializeObject<Animal>(animalJson);

                WeakReferenceMessenger.Default.Send(new
ReplaceViewMessage(ViewTypes.AnimalDetails));
                WeakReferenceMessenger.Default.Send(new
AnimalInfoMessage(animal));
            },
            error =>
            { });
    }

    private async Task RefreshAnimalList()
    {
        try
        {
            IsBusy = true;
            var animalsJsonResult = await _animalService.GetLocations();

            animalsJsonResult.Match(
                onSuccess: animalsJson =>
                {
                    try
                    {
                        IEnumerable<MapAnimalDTO> animals =
JsonConvert.DeserializeObject<IEnumerable<MapAnimalDTO>>(animalsJson);

                        Animals.Clear();

                        foreach (MapAnimalDTO animal in animals)
                        {
                            Animals.Add(animal);
                        }

                        OnAnimalsUpdated?.Invoke();
                    }
                    catch (JsonException jsonEx)
                    {
                        Console.WriteLine($"JSON Deserialization Error:
{jsonEx.Message}");
                    }
                    finally
                    {
                        IsBusy = false;
                    }
                },
```

```
                onFailure: error =>
                {
                    IsBusy = false;
                });
        }
        catch (Exception e)
        {
            throw;
        }
    }
```

Модуль нейроної мережні:

```
import cv2
import base64
import sys
import numpy as np
from ultralytics import YOLO

model = YOLO('yolo11n.pt')


def decode_base64_to_image(base64_string):
    image_data = base64.b64decode(base64_string)
    np_arr = np.frombuffer(image_data, np.uint8)
    return cv2.imdecode(np_arr, cv2.IMREAD_COLOR)


def detect_animal(image):
    results = model(image)
    for result in results:
        for box in result.boxes:
            cls = int(box.cls[0].item())
            label = model.names[cls]
            if label in ['dog', 'cat']:
                return label
    return "unknown"
if __name__ == "__main__":
    input_path = sys.argv[1]
    output_path = sys.argv[2]
    with open(input_path, "r") as file:
        base64_image = file.read().strip()
    image = decode_base64_to_image(base64_image)
    result = detect_animal(image)

    with open(output_path, "w") as file:
        file.write(result)
```

ДОДАТОК В

(обов'язковий)

ПРЕЗЕНТАЦІЙНІ СЛАЙДИ

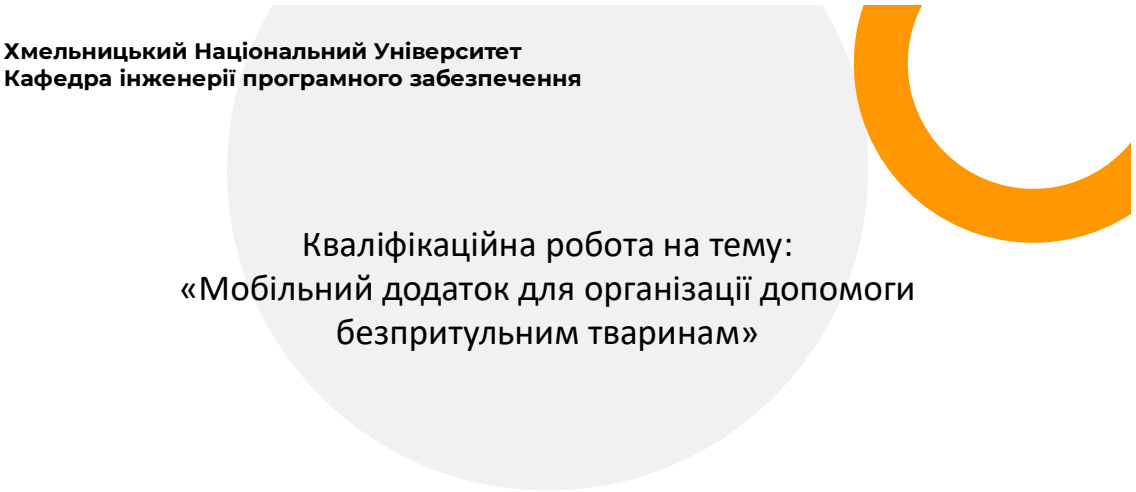


Рисунок В1 – Слайд 1

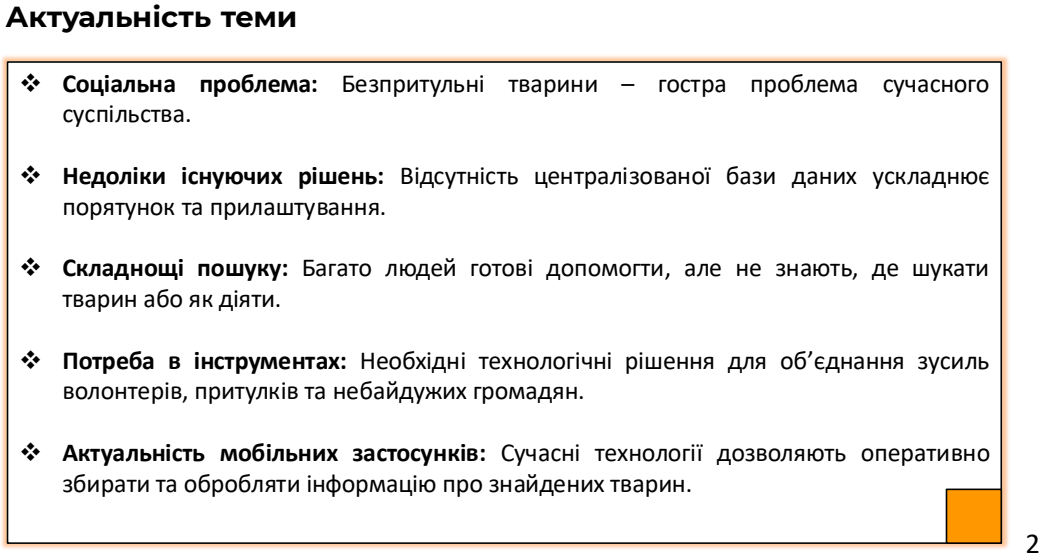


Рисунок В2 – Слайд 2

Мета та завдання

Мета: Розробити мобільний застосунок для фіксації безпритульних тварин, їх геолокації та оперативної взаємодії користувачів.

01 Дослідити предметну область та проаналізувати існуючі рішення.

02 Визначити вимоги до функціоналу та продуктивності застосунку.

03 Спроєктувати архітектуру системи та модель бази даних.

04 Реалізувати інтерфейс користувача та функціональні модулі.

05 Провести тестування та оптимізацію застосунку.

3

Рисунок В3 – Слайд 3

Аналіз предметної області

- У більшості країн немає зручних цифрових інструментів для координації допомоги безпритульним тваринам.
- Наявні сервіси часто обмежені функціоналом і не забезпечують інтегрованого підходу до фіксації, пошуку та прилаштування тварин.
- Для ефективної взаємодії між волонтерами, притулками та звичайними користувачами потрібна єдина платформа з геолокацією, фотофіксацією та обміном інформацією в реальному часі.

📱 Чому саме мобільний застосунок:

Мобільний додаток дозволяє оперативно фіксувати знайдених тварин безпосередньо на місці, можливість використання геолокації для визначення місця тварини на карті, швидкий доступ до бази з будь-якої точки, висока популярність смартфонів.

4

Рисунок В4 – Слайд 4

Аналіз наявного програмно-технічного забезпечення предметної області

Назва	Основна функція	Особливості
PetFinder	Пошук домівки для безпритульних тварин	Профілі з фото, описом та контактами; підтримка взаємодії з притулками
FindPet	Повернення загублених тварин власникам	Штучний інтелект для розпізнавання на фото; сповіщення про збіги
PawBoost	Швидке розповсюдження інформації про знайдених тварин	Інтеграція з соціальними мережами; геолокація для локального пошуку
Adopt A Pet	Прилаштування тварин через волонтерські організації	Фільтрація за видом, віком, породою; підтримка співпраці з притулками

5

Рисунок В5 – Слайд 5

Функціональні і нефункціональні вимоги

Функціональні вимоги	Нефункціональні вимоги
Додавання інформації: фото, опис, геолокація знайдених тварин	Швидкодія: швидка обробка запитів і мінімальний час відгуку
Пошук за параметрами: вид, місце, час, характеристики тварин	Стійкість до навантажень: підтримка великої кількості одночасних запитів
Інтерактивна карта з деталями тварин	Безпека даних: захист через HTTPS та шифрування бази даних
Реєстрація та авторизація: email, соцмережі	Кросплатформеність: підтримка Android та iOS через .NET MAUI
Взаємодія користувачів: коментарі, оновлення статусу, зв’язок	Адаптивний дизайн: зручність на різних пристроях, підтримка кількох мов

6

Рисунок В6 – Слайд 6

Функціональні і нефункціональні вимоги

Use case діаграма

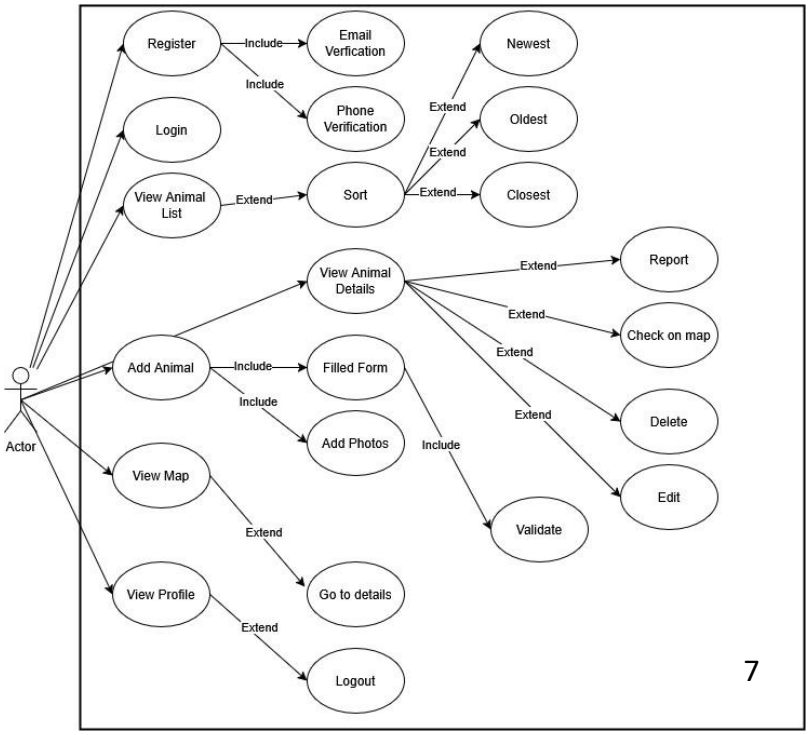
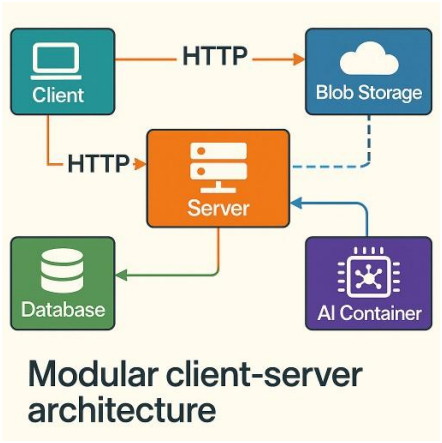


Рисунок В7 – Слайд 7

Проєктування архітектури та структури системи

- Чітке розділення відповідальностей
- Масштабованість і гнучкість у розширенні
- Зручність супроводу та тестування
- Інтеграція з нейромережею та хмарним сховищем

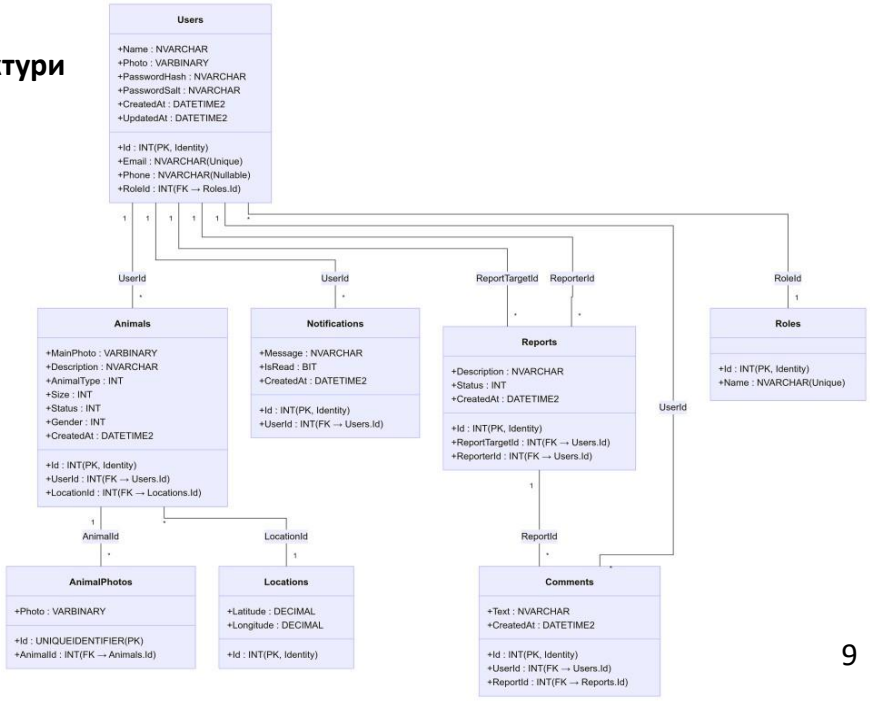


8

Рисунок В8 – Слайд 8

Проєктування архітектури та структури системи

ER діаграма



9

Рисунок В9 – Слайд 9

Проєктування архітектури та структури системи

10

Рисунок В10 – Слайд 10

Рисунок В11 – Слайд 11

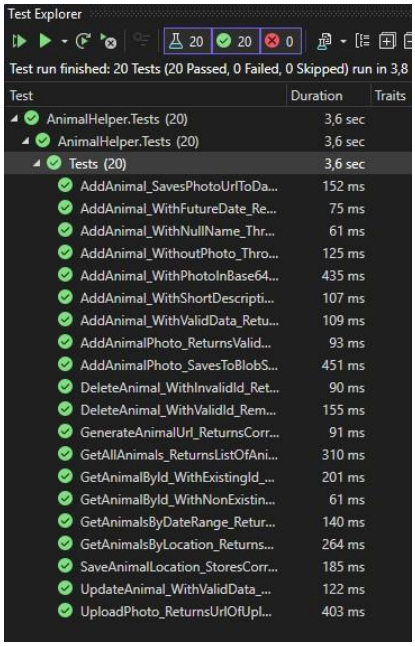
Рисунок В12 – Слайд 12

Тестування

Під час тестування були виявлені незначні помилки, зокрема затримки при завантаженні зображень.

Цю проблему було вирішено шляхом інтеграції Azure Blob Storage, що суттєво покращило швидкість збереження та доступу до фото.

Основний функціонал працює стабільно, серйозних збоїв не виявлено.



12

Рисунок В13 – Слайд 13

Висновки

Проведено аналіз предметної області

Досліджено проблему безпритульних тварин та існуючі сервіси (PetFinder, (PetFinder, FindPet тощо). тощо). Визначено ключові вимоги до функціональності системи.

Обґрунтовано архітектурне рішення

Обрано модульну клієнт-серверну архітектуру з Clean Architecture та REST API для забезпечення гнучкості й масштабованості.

Розроблено повноцінну систему

Реалізовано мобільний застосунок на .NET MAUI та сервер на ASP.NET Web API із використанням MediatR, EF Core, AutoMapper та JWT.

Інтегровано зовнішні сервіси

Додано підтримку розпізнавання тварин через AI контейнер (YOLO) та збереження зображень у хмару (Azure Blob Storage).

Проведено тестування функціоналу

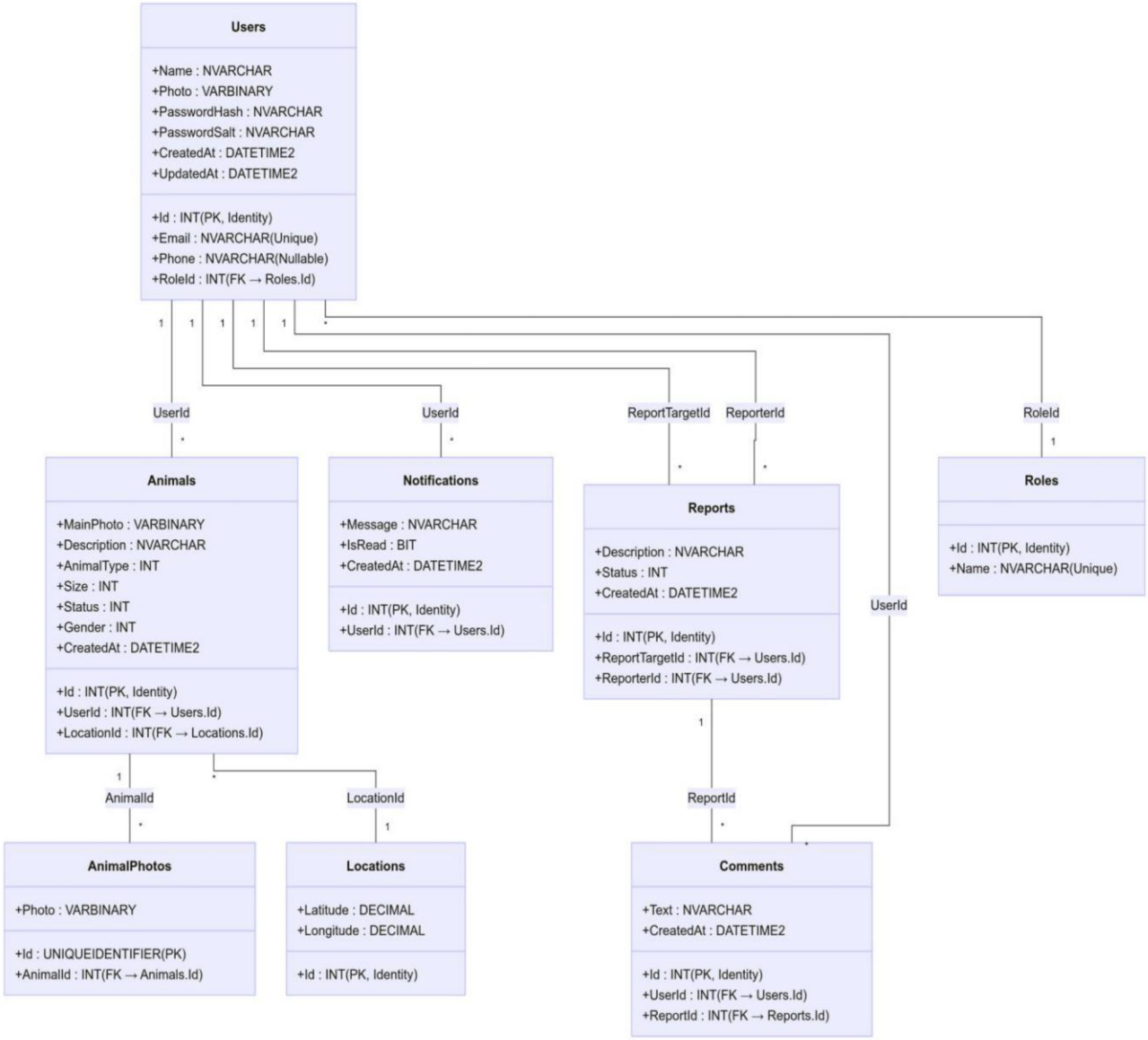
Перевірено ключові сценарії. Виявлені незначні помилки усунено — зокрема, покращено швидкість завантаження фото через Azure.

13

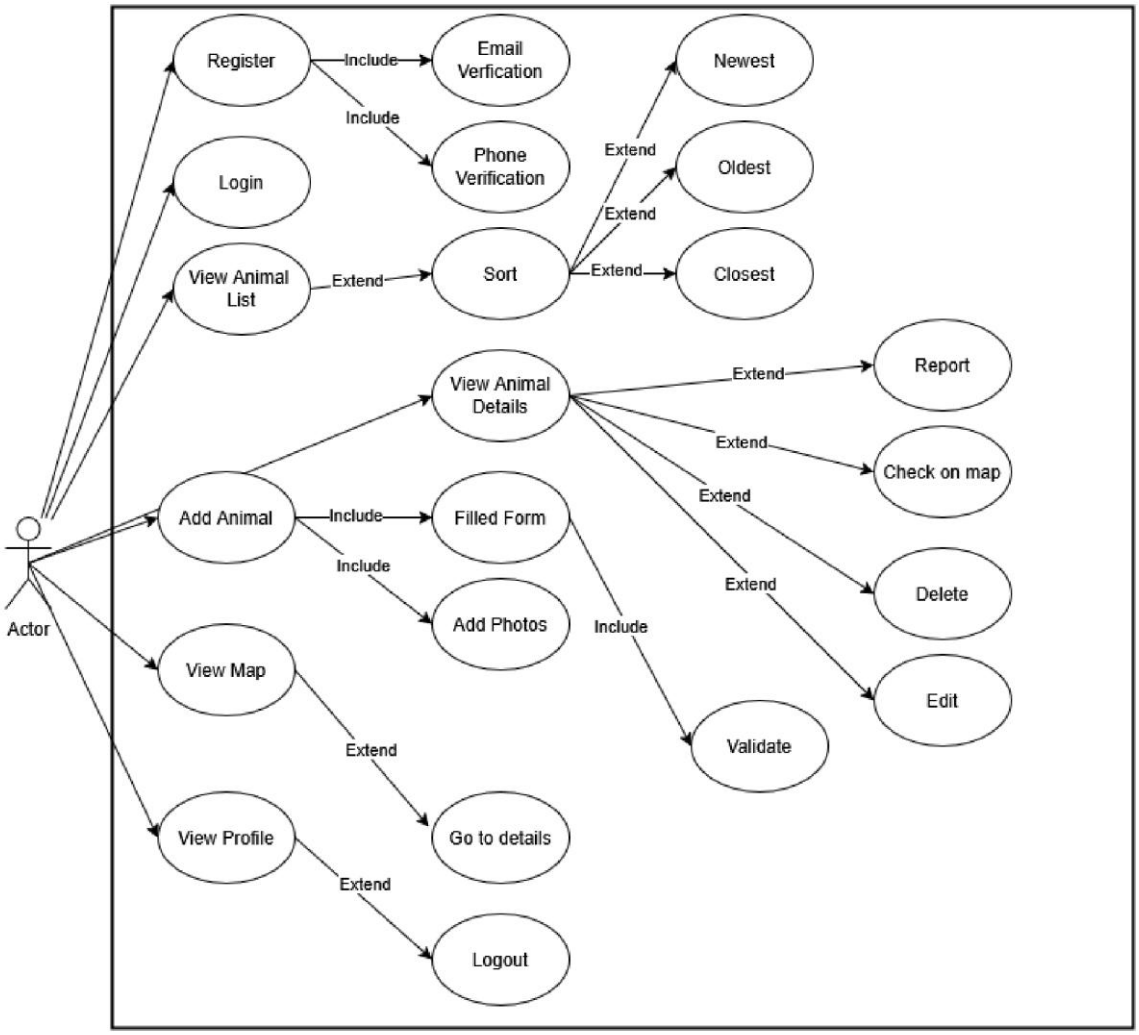
Рисунок В14 – Слайд 14

Рисунок В15 – Слайд 15

ГРАФІЧНІ МАТЕРІАЛИ



					КвРІПЗ.2101093.01.19		
					Мобільний застосунок для організації допомоги безпритульним тваринам ER діаграма	*Літ.*	*Маса.*	*Масштаб*
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розробив		Хоптяр І.Ю.				Аркуш 1	Аркушів 2	
Керівник		Бедратюк Л. П.						
Н. Контр.		Праворська Н.І.				ХНУ, ІПЗ-21-1		
Зав. каф.		Бедратюк Л. П.						



					КвРІПЗ.2101093.01.19			
					Мобільний застосунок для організації допомоги безпритульним тваринам Діаграма варіантів використань	*Літ.*	*Маса.*	*Масштаб*
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розробив		Хоптяр І.Ю.				Аркуш 2	Аркушів 2	
Керівник		Бедратюк Л. П.						
Н. Контр.		Праворська Н.І.				ХНУ, ІПЗ-21-1		
Зав. каф.		Бедратюк Л. П.						

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Хоптяр Іван Юрійович
факультет ІТ, IVкурс, група ІПЗ-21-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлена. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщена та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

2.01.2025
дата

підпис

Thu May 29 11:04:29 EEST 2025, Форкун Юрій Вікторович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 3.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 11%**

	Document		Sum coincidence on the DB	
ID: 242380 Title: Мобільний застосунок для організації допомоги безпритульним тваринами Added in a DB: 2025-05-29 Authors: Мобільний застосунок для організації допомоги безпритульним тваринам Heads: ХОПТЯР Іван Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	94389	1425	5216 (6%)	72 (5%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Хоптяр Іван

Співавтор:

Назва: БКР_Мобільний застосунок для організації допомоги безпритульним тваринам_

Науковий керівник:

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:4.2%

Коефіцієнт подібності 2:1.9%

Мікропробіли: 0

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-05-28 19:40:12.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата 29.05.2025 експерт

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Хоптяр Іван Юрійович

Тема: Мобільний застосунок для організації допомоги безпритульним тваринам

Спеціальність**:** 121 «Інженерія програмного забезпечення»

Обсяг кваліфікаційної роботи:

Кількість листів креслень _____ Кількість сторінок записки ___117___

1. Короткий зміст пояснювальної записки та прийнятих рішень: Пояснювальна записка містить опис актуальної соціальної проблеми — безпритульності тварин — і обґрунтування потреби в цифровому рішенні для її вирішення. Метою роботи є створення мобільного застосунку *PetCompas*, що дозволяє фіксувати факти знаходження загублених або безпритульних тварин із фото та геолокацією, а також координувати дії користувачів. У роботі розглянуто предметну область, визначено функціональні та нефункціональні вимоги, обґрунтовано вибір технологій (MAUI, ASP.NET WebAPI, Azure, YOLO), спроєктовано архітектуру клієнт-серверної системи, реалізовано мобільний клієнт і серверну частину, проведено тестування. Об’єкт дослідження — процес виявлення та допомоги безпритульним тваринам; предмет — програмні засоби для фіксації та пошуку таких тварин з використанням мобільних технологій і ШІ. Практична цінність полягає в потенційній користі застосунку для волонтерів, притулків і небайдужих громадян.

2. Висновок про відповідність роботи дипломному завданню Робота відповідає поставленим завданням: здійснено аналіз предметної області, вивчено існуючі рішення, обґрунтовано вибір стеку технологій, реалізовано клієнтську та серверну частини, включено функції розпізнавання тварин, побудовано архітектуру та проведено тестування. Висновки узгоджуються з поставленою метою. Реалізація демонструє повноцінний функціонал MVP-рівня, однак у роботі бракує окремих аспектів, зокрема розширеного тестування продуктивності та порівняльного аналізу алгоритмів ШІ.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: Розділ 1 містить глибоке

дослідження предметної області, аналіз аналогічних сервісів, постановку задачі та формулювання вимог. Варто відзначити ґрунтовність аналізу ринку та логічну аргументацію необхідності створення нового застосунку.

Розділ 2 описує архітектуру застосунку. Обґрунтовано використання модульної клієнт-серверної моделі та Clean Architecture. Визначено структуру бази даних, спроєктовано взаємодію між компонентами. Проте, було б доцільно розглянути альтернативні архітектурні рішення та більше уваги приділити обґрунтуванню вибору окремих технологій.

Розділ 3 демонструє реалізацію основних модулів: клієнт на MAUI, сервер на ASP.NET Web API, використання MediatR, реалізація JWT-аутентифікації, підключення до нейронної мережі YOLO через API. Опис реалізації структурований, проте деякі технічні деталі реалізації, а також сценарії навантаження та поведінка системи в екстремальних умовах, подані поверхнево.

4. Позитивні сторони роботи: Робота вирізняється високою актуальністю та соціальною значущістю, оскільки спрямована на вирішення проблеми безпритульних тварин за допомогою сучасних IT-засобів. Вдалим є поєднання теоретичних досліджень і практичної реалізації, що дозволяє продемонструвати повний цикл розробки. Архітектура системи побудована грамотно, із врахуванням принципів масштабованості та підтримуваності. У реалізації використано сучасні інструменти, зокрема .NET MAUI, ASP.NET Web API, MediatR, Azure та нейромережу YOLO, що свідчить про актуальність технологічного підходу. Інтерфейс застосунку створено з урахуванням принципів зручності для користувача, що позитивно впливає на загальний досвід взаємодії. Пояснювальна записка структурована логічно, містить необхідні діаграми та ілюстративний матеріал, що покращує сприйняття викладеної інформації.

5. Негативні сторони роботи: Попри значні переваги, робота має окремі недоліки. Зокрема, недостатньо розкрито технічні аспекти інтеграції нейронної мережі в систему, що обмежує розуміння механізму її роботи в контексті застосунку. Також відсутній детальний аналіз навантаження та стрес-тестування, що важливо для оцінки продуктивності та стійкості системи. Обґрунтування вибору окремих технологій, наприклад MediatR, подане поверхнево й не супроводжується порівнянням з альтернативами. Крім того, було б доцільно включити аналіз сильних і слабких сторін

подібних сервісів, таких як FindPet чи PawBoost, для чіткішого позиціонування запропонованого рішення.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Графічні матеріали подані у вигляді UML-діаграм, схем архітектури, блок-схем процесів. Оформлення відповідає вимогам до кваліфікаційних робіт. Пояснювальна записка добре структурована, містить ілюстрації, таблиці, посилання на джерела.

7. Відгук про роботу в цілому: Кваліфікаційна робота Хоптяра Івана Юрійовича є вдалою спробою вирішення актуальної соціальної проблеми засобами сучасних ІТ. Робота демонструє вміння автора працювати з актуальним стеком технологій, будувати клієнт-серверні системи, інтегрувати нейромережі та враховувати потреби кінцевих користувачів. Попри деякі недоліки в частині тестування та деталізації реалізації, проєкт має потенціал до подальшого розвитку і масштабування. Автор продемонстрував належний рівень самостійності, логічного мислення та вміння приймати інженерні рішення.

8. Інші зауваження: Доцільним було б вказати перспективи використання додатку з боку державних або муніципальних ініціатив у сфері зоозахисту, що підвищило б прикладну значущість роботи.

9. Оцінка дипломної роботи: Робота виконана в повному обсязі, відповідає вимогам, має практичну значущість та соціальну спрямованість. Незначні недоліки не знижують загальну якість. Робота заслуговує оцінки «добре».

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) Гавлітова Ольга Олександрівна, д.ф., доцент, зав. каф. КІС

"02" 06 2025 р. (підпис)

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ КАФЕДРИ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи «Мобільний застосунок для організації допомоги безпритульним тваринам»
Автор Хоптяр Іван Юрійович
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»
Рівень вищої освіти Бакалавр
Спеціальність 121 – Інженерія програмного забезпечення
Науковий керівник: Бедратюк Леонід Петрович, д-р фіз.-мат. наук, професор

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	відповідає
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих методів та технологій, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;

2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з 10-40 джерелами на один фрагмент речення;

4) система фіксувала технічні особливості (наприклад, поєднання латиниці й українських індексів), а не модифікацію тексту.

Сумарний обсяг запозичень — 4.2%, що відповідає науковим стандартам і не впливає на якість кваліфікаційної роботи.

Дата 1-06.25

Завідувач кафедри _____ Підпис Леонід Бедратюк Ім'я, ПРІЗВИЩЕ

Гарант освітньої програми _____ Підпис Леонід Бедратюк Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Підпис Леонід Бедратюк Ім'я, ПРІЗВИЩЕ

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Хоптяр Іван Юрійович,

студент IV курсу спеціальності 121 – Інженерія програмного забезпечення,
група ІПЗ-21-1

здобувач вищої освіти (шифр та назва спец-ті, курс, академічна група)

підтверджую, що ознайомився з Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті та Кодексом академічної доброчесності і **зобов'язуюсь** дотримуватися їх вимог під час освітнього процесу, проведення наукової діяльності, виконання організаційно-адміністративних функцій тощо.

Усвідомлюю, що у разі порушення мною принципів академічної доброчесності нестиму відповідальність перед академічною спільнотою ХНУ згідно з нормами, визначеними Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, законодавства України.

2 січня 2025 р. _____

Підпис

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ІПЗ-21-1
Хоптяр І. Ю.
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»: Мобільний застосунок для організації допомоги безпритульним тваринам

(керівник роботи – Бедратюк Леонід Петрович)
Прізвище, ім'я, по батькові

2.01.2025
Дата

Підпис студента