Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Юхимчук Марини Валеріївни
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Мобільний застосунок для догляду та контролю за домашніми тваринами
Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.2201120.01.25.ПЗ

Виконав студент IV курсу, група ІПЗ-22-1 _____ Марина ЮХИМЧУК
Підпис Ім'я, ПРІЗВИЩЕ

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

Нормоконтролер асистент _____ В'ячеслав БОЙКО
Підпис Ім'я, ПРІЗВИЩЕ

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

1 червня 2026 р.

Хмельницький 2026

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 ІПЗ
Л. П. Бедратюк
02. 01. 2026 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Юхимчук Марині Валеріївні
Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Мобільний застосунок для догляду та контролю за домашніми тваринами

Керівник кваліфікаційної роботи Бедратюк Леонід Петрович, д-р фіз.-мат. наук, проф.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 7
2. Строк подання студентом роботи на кафедру 01.06.2026 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 програмного забезпечення, програмна реалізація проєкту, тестування

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Три креслення:
1. Контекстна діаграма А-0,
2. Діаграма декомпозиції А-1,
3. Діаграма варіантів використання

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Бойко В. О., асистент		
Антиплагіат	Форкун Ю. В., канд. техн. наук, доцент	25.05.26	25.05.26

7. Дата видачі завдання «02» 01 2026р.

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 – 31.12.2025	
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 – 20.02.2026	
3 Проєктування програмного забезпечення	21.02 – 20.03.2026	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2026	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2026	
6 Попередній захист КвР	Травень 2026	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.06.2026	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2026	

Студент _____ Юхимчук М. В.
Підпис Ім'я, прізвище

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

АНОТАЦІЯ

Тема кваліфікаційної роботи «Мобільний застосунок для догляду та контролю за домашніми тваринами».

Автор роботи: Юхимчук Марина Валеріївна.

Керівник роботи: Бедратюк Леонід Петрович

Пояснювальна записка: 76 с., 30 рис., 7 табл., 3 дод., 40 джерел.

Графічна частина: 15 слайдів, 3 креслення ф. А3.

DART, FIREBASE, FLUTTER, МОБІЛЬНИЙ ЗАСТОСУНОК, НАГАДУВАННЯ, ДОМАШНІ ТВАРИНИ, МЕДИЧНІ ЗАПИСИ, SQLITE.

Мета кваліфікаційної роботи: розроблення мобільного застосунку для централізованого обліку та контролю догляду за домашніми тваринами.

У кваліфікаційній роботі виконано аналіз предметної області, огляд існуючих програмних рішень, проєктування структури застосунку, реалізацію основних функціональних модулів, створення користувацького інтерфейсу, організацію збереження даних та перевірку працездатності застосунку.

Для реалізації програмного продукту використано мову програмування Dart та фреймворк Flutter. Збереження та синхронізація даних реалізовані з використанням Firebase Firestore, локальне кешування – на основі SQLite.

Практичне значення результатів роботи полягає в тому, що розроблений застосунок дозволяє власникам домашніх тварин централізовано зберігати інформацію про тварин, вести медичні записи та отримувати своєчасні нагадування про заплановані процедури догляду.

28.05.26
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документу	Найменування документу	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.2201120.01.25.ПЗ	Пояснювальна	76		
			записка			
2	А4		Завдання на	1		
			кваліфікаційну			
			роботу			
3	А4		Анотація	1		
			Графічні документи	3		
4	А3	КвРІПЗ.2201120.01.25.ПЗ	Контекстна діаграма	1		
			А-0			
5	А3	КвРІПЗ.2201120.01.25.ПЗ	Діаграма	1		
			декомпозиції А-1			
6	А3	КвРІПЗ.2201120.01.25.ПЗ	Діаграма варіантів	1		
			використання			

Змн.	Арк	№ докум.	Підпис	Дата	КвРІПЗ. 2201120.01.25.ПЗ			
Виконав		Юхимчук М. В.		28.05.24	Мобільний застосунок для догляду та контролю за домашніми тваринами Відомість документів	Літ.	Арк.	Аркушів
Керівник		Бедратюк Л. П.		28.05.24			5	
Н. контр.		Бойко В. О.		28.05.24		ХНУ. ІПЗ-22-1		
Зав. каф.		Бедратюк Л. П.		28.05.24				

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API	–	Інтерфейс програмування застосунків, набір засобів взаємодії між компонентами
BaaS	–	Backend-as-a-Service, модель надання серверної частини як сервісу
CRUD	–	Базові операції роботи з даними: створення, читання, оновлення, видалення
DFD	–	Діаграма потоків даних, що відображає рух інформації в системі
IDEF0	–	Методологія функціонального моделювання процесів
JSON	–	Формат обміну структурованими даними
MVP	–	Мінімально життєздатний продукт із базовим функціоналом
NoSQL	–	Клас нереляційних баз даних
SQL	–	Мова запитів до реляційних баз даних
UML	–	Уніфікована мова моделювання систем
UI	–	Інтерфейс користувача
UX	–	Користувацький досвід взаємодії з системою
SDK	–	Software Development Kit, набір інструментів для розробки програмного забезпечення
ОС	–	операційна система
ОЗП	–	оперативний запам'ятовуючий пристрій
REST	–	Representational State Transfer, архітектурний стиль взаємодії компонентів розподіленої системи
URL	–	Uniform Resource Locator, уніфікований покажчик ресурсу

ВСТУП

Стрімкий розвиток мобільних технологій та поширення смартфонів зумовлюють активну цифровізацію повсякденних сфер діяльності людини. Однією з таких сфер є догляд за домашніми тваринами, що передбачає контроль стану здоров'я, дотримання графіків годування, виконання ветеринарних процедур та ведення медичної історії. У сучасних умовах значна частина власників тварин використовує розрізнені джерела зберігання інформації, зокрема паперові записи або нотатки у мобільних пристроях. Такий підхід ускладнює систематизацію даних та підвищує ризик пропуску важливих процедур.

У межах даної роботи предметна область розглядається в аспекті персонального догляду за домашніми тваринами, зокрема ведення профілів тварин, збереження медичної інформації, планування процедур догляду та формування нагадувань. Основними об'єктами обліку є користувач, профіль тварини, медичний запис і запланована подія. До типових процедур належать вакцинація, прийом ліків, відвідування ветеринара, годування та інші регулярні дії, що потребують контролю з боку власника.

Аналіз сучасного стану предметної області показує наявність мобільних застосунків для догляду за домашніми тваринами, проте більшість із них характеризуються перевантаженим інтерфейсом, обмеженим безкоштовним функціоналом або залежністю від сторонніх сервісів. Частина існуючих рішень орієнтована на інтеграцію з ветеринарними клініками, що знижує універсальність використання. Інші застосунки не забезпечують хмарної синхронізації даних або підтримки роботи в офлайн-режимі. Це свідчить про актуальність розроблення простого, автономного та функціонально достатнього програмного забезпечення для організації догляду за домашніми тваринами.

Актуальність теми кваліфікаційної роботи визначається потребою у створенні мобільного програмного забезпечення, яке забезпечує централізоване зберігання інформації про домашніх тварин, планування догляду та своєчасне

інформування користувача. Додатковим фактором актуальності є зростання кількості власників домашніх тварин та підвищення вимог до якості догляду за ними. Використання мобільного застосунку дозволяє автоматизувати процеси обліку, зменшити ризик пропуску важливих подій та підвищити зручність управління інформацією.

Галуззю застосування розробки є сфера персонального використання власниками домашніх тварин, насамперед користувачами, які утримують одну або кілька тварин і потребують зручного засобу для систематизації інформації. Основними сценаріями використання є створення профілю тварини, внесення медичних записів, планування подій догляду, отримання нагадувань та перегляд збереженої інформації.

Метою кваліфікаційної роботи є розроблення мобільного застосунку для централізованого обліку та контролю догляду за домашніми тваринами, який забезпечує створення профілів тварин, ведення медичної історії, планування процедур догляду та своєчасне інформування користувача з використанням механізмів локального і хмарного зберігання даних.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- аналіз предметної області догляду за домашніми тваринами;
- аналіз існуючих програмних рішень;
- визначення функціональних та нефункціональних вимог до ПЗ;
- розробка архітектури мобільного застосунку;
- проєктування структури даних та моделі зберігання інформації;
- розробка модулів управління профілями тварин;
- реалізація модуля ведення медичної картки;
- розробка системи планування догляду та нагадувань;
- реалізація механізму синхронізації даних;
- програмна реалізація мобільного застосунку;
- тестування та перевірка працездатності програмного забезпечення;
- апробація розробленого програмного продукту.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

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

Розвиток сфери догляду за домашніми тваринами має тривалу історію, яка еволюціонувала від базового утримання тварин до комплексного підходу, що включає контроль стану здоров'я, планування процедур та ведення відповідної документації. З появою інформаційних технологій відбувся поступовий перехід від паперових носіїв до електронних засобів обліку. Подальший розвиток мобільних технологій і поширення смартфонів сприяв формуванню нового класу програмних продуктів, орієнтованих на автоматизацію догляду за тваринами [1].

На сучасному етапі дана сфера характеризується високим рівнем актуальності, що обумовлено зростанням кількості власників домашніх тварин та підвищенням вимог до якості їх утримання. За даними досліджень ринку, кількість домашніх тварин у світі щорічно збільшується, а видатки власників на їх утримання та ветеринарне обслуговування демонструють стійку тенденцію до зростання. Це формує відповідний попит на засоби автоматизації процесів догляду, зокрема мобільні застосунки. Використання інформаційних технологій дозволяє оптимізувати процеси обліку, зменшити ймовірність помилок, пов'язаних із людським фактором, та забезпечити своєчасний доступ до необхідної інформації. Таким чином, розробка програмного забезпечення для підтримки процесів догляду за тваринами відповідає сучасним тенденціям цифровізації повсякденних процесів [2].

Ринок програмних рішень у даній предметній області представлений низкою мобільних застосунків, які реалізують різні підходи до організації функціоналу. Серед них можна виділити рішення, орієнтовані на комплексний облік інформації про тварин, застосунки з акцентом на взаємодію з ветеринарними установами, а також спрощені системи для ведення нотаток і нагадувань. Однак більшість таких продуктів характеризуються або надмірною

складністю інтерфейсу, або обмеженою функціональністю, що знижує їх ефективність у повсякденному використанні.

З погляду предметної структури, процеси догляду за домашніми тваринами можна поділити на кілька ключових категорій. По-перше, це регулярний догляд, який включає годування, гігієнічні процедури та фізичну активність. По-друге, медичне обслуговування, що охоплює профілактичні огляди, вакцинацію, дегельмінтизацію та лікування захворювань. По-третє, адміністративний облік – збереження ветеринарних документів, страхових полісів, родоводів та інших даних. Саме ця категорія найменш автоматизована серед широкого кола власників тварин і найчастіше ведеться в неструктурованому вигляді – у паперових записах, фотографіях документів або загальних нотатках на смартфоні. Таке фрагментарне зберігання інформації знижує її доступність у критичних ситуаціях, наприклад під час термінового звернення до ветеринара.

Процес створення програмних продуктів у цій галузі відповідає загальним принципам розробки програмного забезпечення та включає етапи аналізу предметної області, формування вимог, проєктування архітектури, реалізації функціоналу, тестування та подальшої підтримки. На етапі аналізу визначаються основні потреби користувачів та особливості предметної області. Далі формується структура застосунку, обираються технології реалізації та здійснюється програмна розробка. Завершальні етапи включають перевірку працездатності, виправлення помилок та оновлення функціоналу відповідно до змін вимог [8].

У процесі розробки зазвичай задіяні різні ролі, серед яких можна виділити розробників програмного забезпечення, дизайнерів інтерфейсу, тестувальників та, за необхідності, аналітиків. Для реалізації мобільних застосунків використовуються сучасні технології, зокрема кросплатформні фреймворки, хмарні сервіси для зберігання даних, а також системи контролю версій. Застосування таких інструментів дозволяє підвищити ефективність розробки та забезпечити гнучкість у подальшому розвитку продукту [4].

Серед сучасних програмних рішень можна виділити застосунки 11Pets, PetDesk та Pet Care Tracker, які реалізують функції ведення медичних записів, нагадувань і зберігання інформації про тварин. Водночас частина таких систем характеризується складною структурою інтерфейсу або орієнтацією на взаємодію з ветеринарними сервісами, що знижує зручність їх повсякденного використання.

Цільова аудиторія таких програмних продуктів представлена власниками домашніх тварин різного віку та рівня досвіду. Для молодших користувачів характерною є активна взаємодія з мобільними технологіями та очікування зручного інтерфейсу. Більш досвідчені користувачі орієнтовані на функціональність і надійність зберігання даних. Незалежно від вікової категорії, основними потребами залишаються простота використання, швидкий доступ до інформації та наявність системи нагадувань [3].

Структуру цільової аудиторії за поколіннями та рівень відповідних витрат наочно ілюструють рисунки 1.1.1 та 1.1.2.

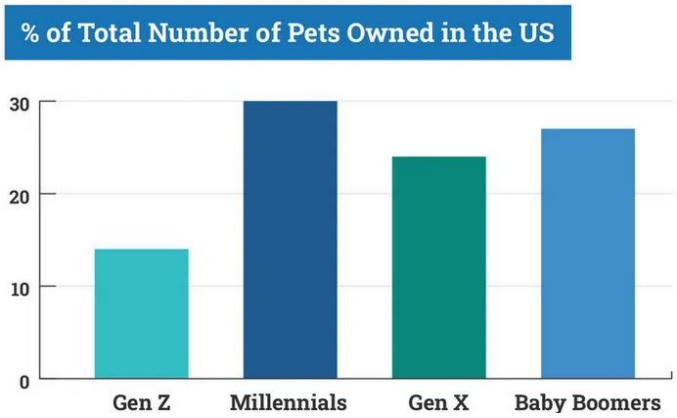


Рисунок 1.1.1 – Частка власників домашніх тварин у США за поколіннями

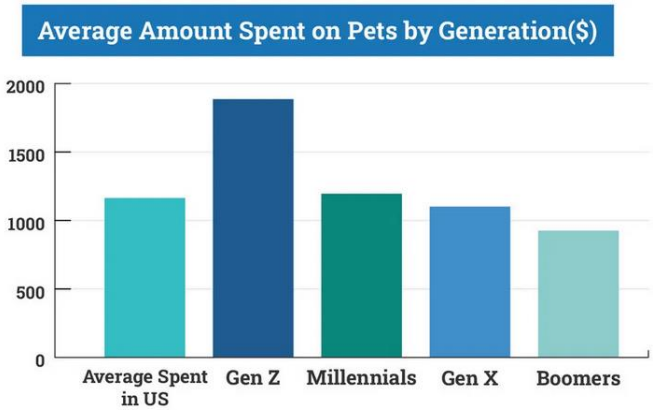


Рисунок 1.1.2 – Річні витрати на утримання тварин за поколіннями США

З точки зору класифікації потреб користувачів доцільно розрізняти первинні та вторинні функціональні потреби. До первинних належать реєстрація тварин та збереження базової інформації про них, ведення медичних записів і нагадування про заплановані події. Вторинними є аналітичні функції, статистика витрат, інтеграція з ветеринарними клініками та соціальні функції. Для MVP-версії застосунку достатньо реалізувати первинні потреби, оскільки саме вони визначають практичну цінність продукту для більшості користувачів. Вторинні потреби можуть бути реалізовані на наступних етапах розвитку системи без суттєвих змін її архітектури.

У сучасних умовах спостерігається зростання інтересу користувачів до цифрових інструментів, що спрощують повсякденні процеси. Як показують дослідження, більшість власників домашніх тварин надає перевагу мобільним застосункам як основному засобу організації інформації та планування регулярних процедур догляду. Це пов'язано з доступністю мобільних пристроїв та можливістю оперативного доступу до даних у будь-який момент часу. Наочним підтвердженням цієї тенденції є динаміка світового ринку мобільних застосунків для догляду за тваринами, представлена на рисунку 1.1.3.

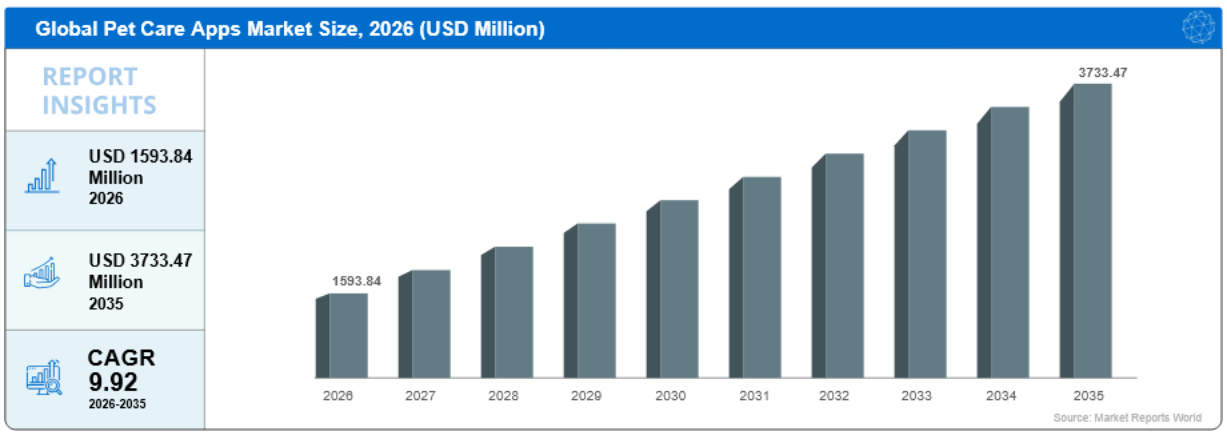


Рисунок 1.1.3 – Графік зростання ринку

Аналіз поведінки користувачів свідчить, що важливим фактором при виборі програмного забезпечення є простота інтерфейсу та швидкість виконання основних дій. Більшість користувачів уникає складних систем із перевантаженим функціоналом, віддаючи перевагу рішенням, які забезпечують мінімальну

кількість кроків для досягнення результату. Відзначається зростання вимог до надійності зберігання даних та можливості їх синхронізації між пристроями.

Крім того, значна частина користувачів очікує наявності системи нагадувань, яка дозволяє своєчасно виконувати заплановані дії без необхідності постійного контролю. Це особливо актуально у контексті догляду за тваринами, де пропуск процедур може мати негативні наслідки. Дослідження показують, що саме функція нагадувань є одним із ключових критеріїв при виборі застосунку для догляду за тваринами – її відзначають як важливу понад 70% респондентів серед цільової аудиторії.

Середовище використання таких застосунків визначається мобільним характером пристроїв. Використання здійснюється у повсякденних умовах, що передбачає необхідність забезпечення стабільної роботи як за наявності мережевого з'єднання, так і в автономному режимі. Це зумовлює вимоги до організації зберігання даних та реалізації механізмів синхронізації. Важливо, що застосунок повинен залишатися функціональним навіть при тимчасовій відсутності доступу до мережі інтернет, оскільки користувачі можуть звертатися до нього в умовах слабкого сигналу або повної відсутності з'єднання.

З точки зору візуального оформлення, більшість сучасних застосунків у даній сфері орієнтується на мінімалістичний стиль інтерфейсу, що забезпечує зручність навігації та швидкість виконання основних дій. Використання стандартних підходів до проєктування інтерфейсів сприяє підвищенню доступності продукту для широкого кола користувачів. Водночас дизайн не повинен бути надмірно спрощеним: відсутність візуальної ієрархії ускладнює орієнтацію в інтерфейсі і може призводити до помилкових дій.

Суттєвим фактором, що впливає на успішність застосунку на ринку, є модель доступу до функціоналу. Практика показує, що вільний базовий доступ із можливістю розширення через платну підписку є найбільш прийнятною моделлю для цільової аудиторії. Проте значна частина користувачів відмовляється від

застосунків, де ключові функції заблоковані без оплати, особливо якщо аналогічна функціональність пропонується безкоштовними конкурентами.

Узагальнюючи результати проведеного аналізу, можна зробити висновок про наявність потреби у створенні мобільного застосунку, який поєднує простоту використання, достатній функціонал та надійне зберігання даних. На основі цього формуються основні вимоги до майбутнього програмного забезпечення, що включають підтримку обліку інформації про тварин, ведення медичної історії, планування процедур догляду, реалізацію системи нагадувань та забезпечення синхронізації даних між пристроями.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Для визначення вимог до розроблюваного програмного забезпечення було проведено аналіз існуючих програмних рішень у сфері догляду за домашніми тваринами. Метою такого аналізу є виявлення основних підходів до реалізації функціоналу, визначення сильних і слабких сторін існуючих продуктів, а також формування обґрунтованих вимог до власного програмного рішення [9].

Одним із найбільш показових прикладів у даній сфері є застосунок 11Pets: Pet Care, який можна розглядати як один із базових орієнтирів для розвитку подібних систем. Даний продукт було створено з метою комплексного обліку інформації про домашніх тварин і він сформував підхід до інтеграції різних аспектів догляду в межах одного застосунку. Його значущість полягає у поєднанні функцій медичного обліку, планування процедур та системи нагадувань, що визначило стандартний набір функціональності для подібних рішень [5].

З функціональної точки зору застосунок забезпечує ведення профілів тварин, збереження медичної історії, організацію графіків догляду та автоматичне формування нагадувань. Архітектурно рішення орієнтоване на централізоване зберігання даних із підтримкою синхронізації між пристроями. Взаємодія з користувачем реалізована через багатовіконний інтерфейс із великою кількістю

елементів керування. Такий підхід дозволяє охопити широкий спектр задач, однак призводить до перевантаження інтерфейсу та ускладнення навігації. Серед ключових недоліків також слід відзначити часткову доступність функціоналу та залежність від платної підписки.

Поряд із комплексними рішеннями існують альтернативні підходи до реалізації функціоналу. Зокрема, можна виділити застосунки, орієнтовані на інтеграцію з ветеринарними сервісами, а також спрощені інструменти для ведення локального обліку без використання хмарних технологій. Такий поділ дозволяє класифікувати програмні продукти за ступенем складності та рівнем функціональної завершеності.

До сучасних представників першого підходу належить застосунок PetDesk, розроблений компанією PetDesk Inc. Даний продукт орієнтований на забезпечення взаємодії між власниками тварин і ветеринарними клініками. Основними функціями є онлайн-запис до ветеринара, отримання нагадувань про процедури та зберігання медичної інформації. Інтерфейс застосунку характеризується сучасним дизайном і зручною навігацією. Водночас суттєвим обмеженням є залежність від партнерських клінік, що звужує можливості використання застосунку як універсального інструменту обліку [6]. Для тих користувачів, чиї ветеринарні клініки не є партнерами PetDesk, значна частина функціоналу виявляється недоступною або суттєво обмеженою, що робить застосунок непридатним для загального використання.

Інший підхід реалізовано у застосунку PetNote – Pets Health Tracker, який орієнтований на мінімалістичну організацію даних. Основна функціональність включає ведення записів про події, створення нагадувань та збереження додаткової інформації у вигляді нотаток і зображень. Перевагою даного рішення є простота інтерфейсу та швидкість взаємодії з користувачем. Водночас відсутність хмарної синхронізації обмежує можливості збереження та перенесення даних між пристроями, що знижує надійність такого підходу [7].

Зокрема, у разі втрати або зміни мобільного пристрою всі дані користувача можуть бути безповоротно втрачені.

Для більш повного уявлення про ринок доцільно також розглянути загальні тенденції у розробці подібних застосунків. Аналіз показує, що сучасні рішення все частіше інтегрують механізми хмарного зберігання та синхронізації даних, що свідчить про усвідомлення розробниками важливості збереження інформації. Водночас ускладнення функціоналу нерідко призводить до погіршення користувацького досвіду, оскільки надлишкові можливості відволікають від основних задач. Окремою тенденцією є монетизація через підписку, яка дозволяє розробникам підтримувати актуальність продукту, проте знижує доступність для широкого кола користувачів.

З точки зору технічної реалізації, переважна більшість мобільних застосунків у даній сфері розробляється із застосуванням кросплатформних технологій, що дозволяє охопити аудиторію користувачів як платформи Android, так і iOS. Це скорочує строки розробки та витрати на підтримку окремих версій продукту. Для збереження даних зазвичай поєднуються локальні та хмарні механізми, що дозволяє забезпечити доступ до інформації незалежно від наявності мережевого з'єднання.

Порівняльний аналіз розглянутих рішень дозволяє виділити спільні елементи, характерні для програмних продуктів у даній предметній області:

- підтримка профілів тварин;
- ведення медичної історії;
- наявність системи нагадувань;
- використання мобільного інтерфейсу як основного способу взаємодії;
- орієнтація на персональне використання.

Водночас відмінності між рішеннями проявляються у підходах до організації зберігання даних, рівні складності інтерфейсу, ступені автономності та моделі доступу до функціоналу. Для узагальнення результатів аналізу та наочного

представлення відмінностей між розглянутими рішеннями сформовано порівняльну таблицю.

Таблиця 1.1 – Порівняльний аналіз аналогів

Критерій	11Pets	PetDesk	PetNote
Профіль тварини	Так	Так	Так
Медична картка	Так	Так	Так
Система нагадувань	Так	Так	Так
Хмарна синхронізація	Так	Ні	Ні
Безкоштовний доступ	Частково	Частково	Так
Простота інтерфейсу	Ні	Так	Так
Незалежність від стор. сервісів	Так	Ні	Так

На підставі аналізу таблиці можна зробити висновок, що жодне з розглянутих рішень не задовольняє одночасно всі ключові критерії. 11Pets забезпечує повний функціонал і хмарну синхронізацію, але поступається простотою інтерфейсу та обмежує доступ за моделлю підписки. PetDesk орієнтований на специфічну взаємодію з клініками та не підходить для широкого кола користувачів. PetNote відрізняється простотою, але суттєво програє у надійності зберігання даних через відсутність синхронізації. Саме цей «трикутний» розрив між функціональністю, зручністю та доступністю визначає практичну нішу для розроблюваного застосунку.

Проведений аналіз показує, що більшість існуючих рішень орієнтовані або на розширений функціонал із підвищеною складністю інтерфейсу, або на спрощені локальні системи без повноцінної синхронізації даних. У зв'язку з цим доцільним є розроблення мобільного застосунку, який поєднуватиме простоту використання, базовий набір функцій та підтримку локального і хмарного зберігання інформації.

З урахуванням виявлених особливостей доцільно використовувати у розроблюваному програмному забезпеченні такі підходи:

- забезпечення мінімалістичного та зрозумілого інтерфейсу;
- реалізація базового набору функцій без надлишкових елементів;
- підтримка синхронізації даних між пристроями;
- забезпечення автономності роботи без прив'язки до сторонніх сервісів;
- орієнтація на швидкість виконання основних дій.

Таким чином, результати аналізу існуючих програмних рішень дозволяють сформувати обґрунтовані вимоги до розроблюваного застосунку та визначити напрямки його подальшої реалізації.

1.3 Визначення функціональних та нефункціональних вимог

На основі проведеного аналізу предметної області та існуючих програмних рішень було сформовано вимоги до розроблюваного програмного забезпечення. Визначення вимог здійснюється з позиції функціональності системи, тобто описується, які задачі має виконувати програмний продукт, без деталізації способів їх реалізації [10].

Розроблюваний мобільний застосунок призначений для обліку інформації про домашніх тварин, ведення медичної картки та планування процедур догляду. Основний акцент робиться на забезпеченні простоти використання, цілісності зберігання даних та своєчасному інформуванні користувача. Програмний продукт реалізується у форматі MVP та орієнтований на персональне використання.

Взаємодія користувача із системою побудована за принципом мінімальної кількості дій для досягнення результату. Після проходження авторизації користувач отримує доступ до основного функціоналу застосунку, який включає перегляд списку тварин, управління профілями, ведення медичної історії та налаштування нагадувань. Логіка навігації є лінійною та передбачуваною, що забезпечує швидке освоєння застосунку без додаткового навчання.

Для формалізації взаємодії користувача із системою було побудовано UML-діаграму варіантів використання, яка відображає основні сценарії роботи із застосунком. Основним актором є користувач, який взаємодіє із системою через набір варіантів використання, що включають реєстрацію, авторизацію, перегляд інформації, управління даними та отримання сповіщень [11].

Користувач є єдиним зовнішнім актором системи та ініціює всі основні сценарії взаємодії. Система виступає як внутрішній актор, що обробляє вхідні запити та повертає відповідні результати. Відношення «include» використовується для обов'язкових підпроцесів, зокрема валідації даних при реєстрації та авторизації. Відношення «extend» описує необов'язкові розширення базових сценаріїв, наприклад відображення повідомлення про помилку при некоректному введенні даних.

Представлена на рисунку 1.3.1 діаграма відображає зв'язки між основними варіантами використання та допоміжними процесами, зокрема валідацією введених даних, синхронізацією інформації та надсиланням сповіщень. Використання відношень «include» та «extend» дозволяє деталізувати взаємодію між сценаріями та виділити спільні функціональні компоненти системи [12].

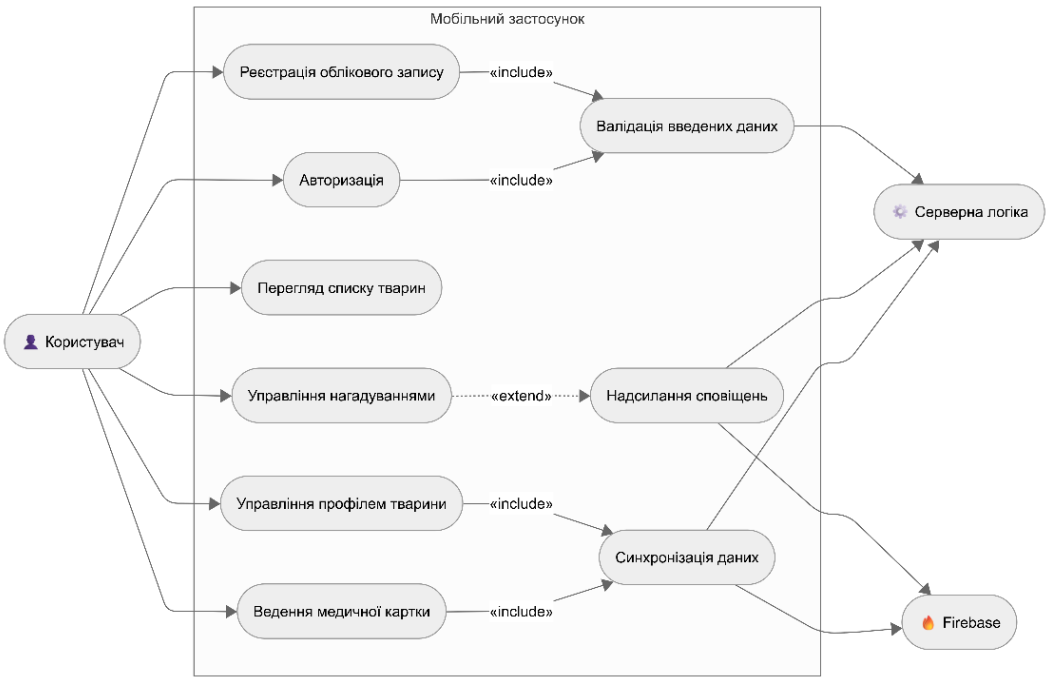


Рисунок 1.3.1 – Діаграма варіантів використання мобільного застосунку

Типовий сценарій використання передбачає авторизацію користувача, створення профілю тварини, внесення медичних записів та подальше налаштування нагадувань про заплановані події. Усі зміни повинні автоматично зберігатися та бути доступними під час наступного використання застосунку.

Для структурованого представлення вимог доцільно розглянути кожну функціональну вимогу у взаємозв'язку з відповідним сценарієм використання. Вимога щодо реєстрації та авторизації забезпечує базову ідентифікацію користувача та захист його персональних даних. Вимога щодо управління профілями тварин дозволяє зберігати структуровану інформацію про кожну тварину, включаючи кличку, вид, породу, дату народження, стать та інші характеристики. Вимога щодо ведення медичних записів забезпечує систематизацію інформації про стан здоров'я, проведені процедури та прийом ліків у хронологічному порядку. Вимога щодо нагадувань дозволяє планувати майбутні події та отримувати своєчасні сповіщення, що є ключовою практичною функцією системи.

Окремо слід розглянути вимогу щодо синхронізації даних між локальним та хмарним сховищами. Ця вимога є технічно складнішою за інші, оскільки передбачає узгодження стану даних між двома середовищами зберігання. Механізм синхронізації повинен забезпечувати актуальність інформації після відновлення мережевого з'єднання, а також виключати можливість втрати або дублювання записів.

Вимога щодо підтримки кількох профілів тварин у межах одного облікового запису відповідає реальним умовам використання, оскільки значна частина власників утримує більше однієї тварини. Система повинна дозволяти перемикатися між профілями без втрати контексту та забезпечувати ізоляцію даних між різними тваринами.

Основні функціональні вимоги:

- реєстрація та авторизація користувача;
- перевірка та валідація введених даних;

- створення, редагування та видалення профілів тварин;
- перегляд списку тварин;
- ведення медичних записів із прив'язкою до профілю тварини;
- створення та управління нагадуваннями із підтримкою повторення;
- синхронізація даних між локальним та хмарним сховищами;
- підтримка кількох профілів тварин у межах одного облікового запису.

Нефункціональні вимоги описують характеристики системи, що визначають якість її функціонування, але не пов'язані безпосередньо з реалізацією конкретних функцій. Вони охоплюють продуктивність, надійність, безпеку, зручність використання та масштабованість системи.

Вимога щодо сумісності з мобільними пристроями під управлінням Android 8.0 та вище обумовлена необхідністю охоплення достатньо широкої аудиторії користувачів. Вибір версії Android 8.0 як мінімальної обумовлений тим, що переважна більшість активних пристроїв функціонує на цій або новішій версії операційної системи, а підтримка більш ранніх версій потребувала б значних додаткових зусиль без суттєвого збільшення аудиторії.

Вимога щодо інтуїтивно зрозумілого інтерфейсу передбачає, що середній користувач повинен бути здатний виконати основні дії в застосунку без ознайомлення з інструкцією. Це досягається через послідовну навігаційну структуру, чітке візуальне оформлення елементів керування та узгоджену логіку взаємодії на всіх екранах.

Вимога щодо стабільної роботи системи сповіщень у фоновому режимі є критично важливою для практичної цінності застосунку. Нагадування повинні надходити користувачеві своєчасно незалежно від того, чи відкритий застосунок у момент спрацювання події. Це визначає необхідність використання системних механізмів планування сповіщень мобільної платформи.

Вимога щодо цілісності та узгодженості даних при синхронізації передбачає, що дані не повинні дублюватися, пошкоджуватися або втрачатися в

процесі передачі між локальним та хмарним сховищами. Це є особливо важливим у ситуаціях, коли синхронізація переривається через втрату мережевого з'єднання.

Вимога щодо можливості масштабування функціоналу забезпечує гнучкість системи для подальшого розвитку. Архітектура застосунку повинна дозволяти додавати нові функції або розширювати наявні без необхідності суттєвого перепроєктування базових компонентів.

Нефункціональні вимоги:

– забезпечення коректної роботи на мобільних пристроях під управлінням Android 8.0 та вище;

– забезпечення інтуїтивно зрозумілого інтерфейсу, що не потребує спеціального навчання;

– забезпечення стабільної роботи системи сповіщень у фоновому режимі;

– забезпечення цілісності та узгодженості даних при синхронізації;

– можливість масштабування функціоналу без значних змін базових компонентів системи [13].

Сукупність визначених функціональних та нефункціональних вимог формує вичерпну основу для переходу до етапу проєктування системи та є орієнтиром для прийняття архітектурних і технологічних рішень на наступних етапах розробки.

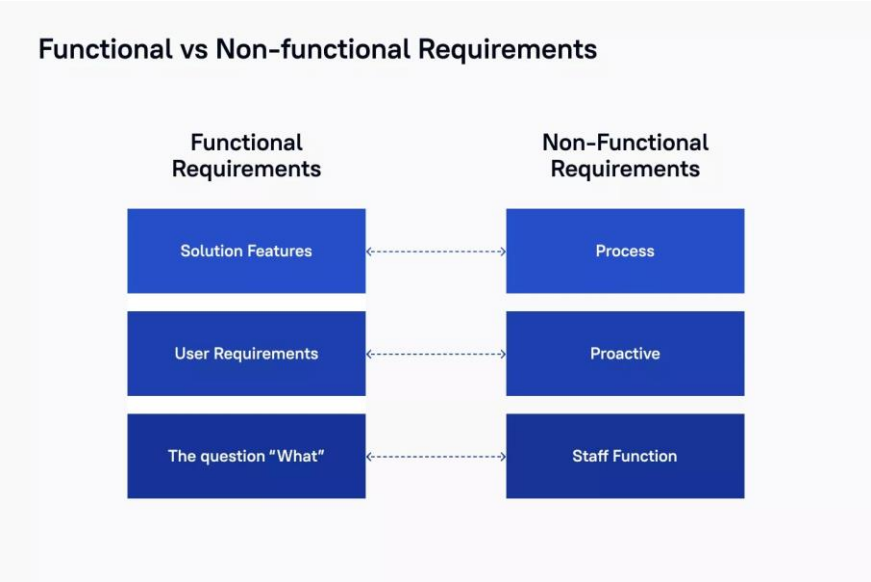


Рисунок 1.3.2 – Функціональні та нефункціональні вимоги

1.4 Висновки дослідження предметної області

У межах дослідження предметної області було проаналізовано особливості організації процесу догляду за домашніми тваринами, сучасні підходи до обліку відповідної інформації, а також роль інформаційних технологій у автоматизації таких процесів. Встановлено, що традиційні способи зберігання даних є недостатньо ефективними, оскільки не забезпечують належного рівня структурованості, доступності та контролю.

Проведений аналіз існуючих програмних рішень показав, що більшість застосунків у даній сфері реалізують базовий функціонал, який включає ведення профілів тварин, збереження медичної інформації та використання системи нагадувань. Водночас виявлено ряд недоліків, серед яких перевантаженість інтерфейсу, обмежена функціональність окремих рішень, а також залежність від сторонніх сервісів або платних моделей доступу. Жодне з проаналізованих рішень не поєднує одночасно простоту інтерфейсу, повноту базового функціоналу, хмарну синхронізацію та незалежність від сторонніх платформ, що підтверджує актуальність розробки власного рішення.

У процесі аналізу було визначено ключові фактори, що впливають на ефективність використання мобільних застосунків у даній предметній області. До них належать простота інтерфейсу, швидкість виконання основних дій, надійність зберігання даних та наявність системи нагадувань. Встановлено, що користувачі надають перевагу рішенням, які забезпечують мінімальну кількість дій для досягнення результату та не перевантажені зайвими функціями.

На основі дослідження було сформовано функціональні та нефункціональні вимоги до розроблюваного ПЗ та обґрунтовано необхідність забезпечення ведення обліку інформації про тварин, збереження медичних записів, планування процедур догляду та реалізацію механізму нагадувань. Окрему увагу приділено вимогам до стабільності роботи, зручності інтерфейсу та надійності механізмів синхронізації даних між пристроями.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

2.1 Проєктування архітектури та структури системи

Архітектура програмного забезпечення визначає структуру системи, взаємодію її компонентів та принципи організації даних і логіки. Від обраної архітектури залежать стабільність роботи застосунку, можливість його масштабування, підтримки та подальшого розвитку.

У сучасній розробці програмного забезпечення існує кілька основних архітектурних підходів, які можуть бути використані залежно від типу системи та її складності. До них належать клієнт-серверна архітектура, багаторівнева (шарова), монолітна та мікросервісна архітектури. Кожен із цих підходів має свої особливості, які варто враховувати при проєктуванні системи.

Клієнт-серверна архітектура передбачає розподіл системи на дві частини: клієнтську, яка взаємодіє з користувачем, і серверну, яка обробляє дані. Такий підхід дозволяє централізовано зберігати інформацію та забезпечує доступ до неї з різних пристроїв. Проте він залежить від наявності мережевого з’єднання, що може впливати на стабільність роботи.

Багаторівнева архітектура базується на поділі системи на логічні рівні, кожен із яких відповідає за окрему частину функціональності. Найчастіше виділяють рівень інтерфейсу, рівень логіки та рівень даних. Це дозволяє чітко розділити відповідальність між частинами системи, спрощує тестування та внесення змін. Недоліком є певне ускладнення структури та необхідність організації взаємодії між рівнями [16].

Монолітна архітектура передбачає створення системи як єдиного цілого, без чіткого поділу на незалежні компоненти. Вона проста у реалізації та добре підходить для невеликих проєктів, проте зі збільшенням функціональності стає складнішою у підтримці та масштабуванні [14].

Мікросервісна архітектура, навпаки, передбачає поділ системи на незалежні сервіси, які можуть розроблятися та розгортатися окремо. Це забезпечує високу

гнучкість і масштабованість, але значно ускладнює проєктування та підтримку. Для систем середнього рівня складності такий підхід є надлишковим [15].

Таблиця 2.1 – Порівняльний аналіз архітектурних підходів

Архітектура	Переваги	Недоліки	Доцільність
Клієнт-серверна	Централізоване зберігання даних, доступ з різних пристроїв	Залежність від мережевого з'єднання	Частково – як основа для хмарної синхронізації
Багаторівнева	Чіткий поділ відповідальності, простота тестування та внесення змін	Ускладнення структури, необхідність організації взаємодії між рівнями	Так – забезпечує розділення інтерфейсу, логіки та даних
Монолітна	Проста у реалізації, підходить для невеликих проєктів	Складна у підтримці та масштабуванні зі збільшенням функціоналу	Ні – обмежує подальший розвиток
Мікросервісна	Висока гнучкість і масштабованість, незалежне розгортання сервісів	Значно ускладнює проєктування та підтримку	Ні – надлишкова для системи середнього рівня складності

З огляду на специфіку розроблюваного застосунку комбінований підхід є найбільш доцільним. Клієнт-серверна складова реалізується через взаємодію мобільного клієнта з Firebase як хмарним бекендом, що забезпечує централізоване зберігання та синхронізацію даних. Багаторівнева складова відображається у внутрішній організації клієнтської частини, де чітко розмежовані рівень інтерфейсу, рівень бізнес-логіки та рівень роботи з даними. Такий розподіл

дозволяє змінювати окремі частини системи незалежно одна від одної, що є важливим для подальшого розвитку продукту.

Варто зазначити, що вибір архітектури значною мірою залежить від технологічного стеку, який використовується для реалізації системи. Різні інструменти можуть по-різному впливати на структуру та можливості архітектури. На даному етапі розробки технологічний стек ще не визначений остаточно, тому архітектура описується у загальному вигляді, без прив’язки до конкретних засобів реалізації. Це дозволяє зберегти гнучкість і при необхідності адаптувати рішення на наступних етапах.

Кожен із розглянутих підходів має свої переваги та недоліки. Прості архітектури забезпечують швидку реалізацію, але обмежують можливості розвитку. Більш складні підходи забезпечують гнучкість і масштабованість, проте потребують більших витрат ресурсів. Тому вибір архітектури є компромісом між складністю, продуктивністю та зручністю підтримки.

З урахуванням особливостей розроблюваного застосунку доцільно використовувати комбінований підхід, який поєднує елементи клієнт-серверної та багаторівневої архітектур. Це дозволяє розділити систему на логічні частини, зберігаючи при цьому відносну простоту реалізації.

Під час формування архітектури також враховувалися особливості мобільної платформи, зокрема необхідність підтримки роботи в умовах нестабільного мережевого з’єднання, обмеженість ресурсів мобільних пристроїв та потреба у швидкій взаємодії користувача із системою. Це вплинуло на вибір багаторівневої структури та використання локального зберігання даних.

Узагальнена структура системи передбачає наявність трьох основних рівнів. Перший рівень відповідає за взаємодію з користувачем і відображення інформації. Другий рівень реалізує основну логіку роботи застосунку, обробку даних і виконання сценаріїв. Третій рівень забезпечує зберігання і доступ до даних. Такий поділ дозволяє ізолювати окремі частини системи та спростити їх майбутні зміни.

Для більш детального опису структури системи виконано її декомпозицію на окремі функціональні модулі:

- модуль управління користувачами та профілями тварин;
- модуль ведення медичної інформації;
- модуль управління нагадуваннями;
- модуль збереження та доступу до даних.

Кожен із модулів виконує окрему функцію та взаємодіє з іншими через визначені інтерфейси. Взаємодія між модулями відбувається через рівень логіки, який забезпечує обробку запитів та передачу даних між компонентами системи. Модулі, що працюють із даними, використовують єдину модель доступу, що дозволяє забезпечити узгодженість інформації.

Під час проєктування системи використовувалися окремі принципи шаблонного проєктування, зокрема централізація доступу до даних та розділення відповідальності між компонентами. Це дозволяє зменшити залежність між модулями та спростити подальший розвиток системи.

Для наочного представлення структури системи та взаємодії між її основними компонентами побудовано узагальнену діаграму архітектури.

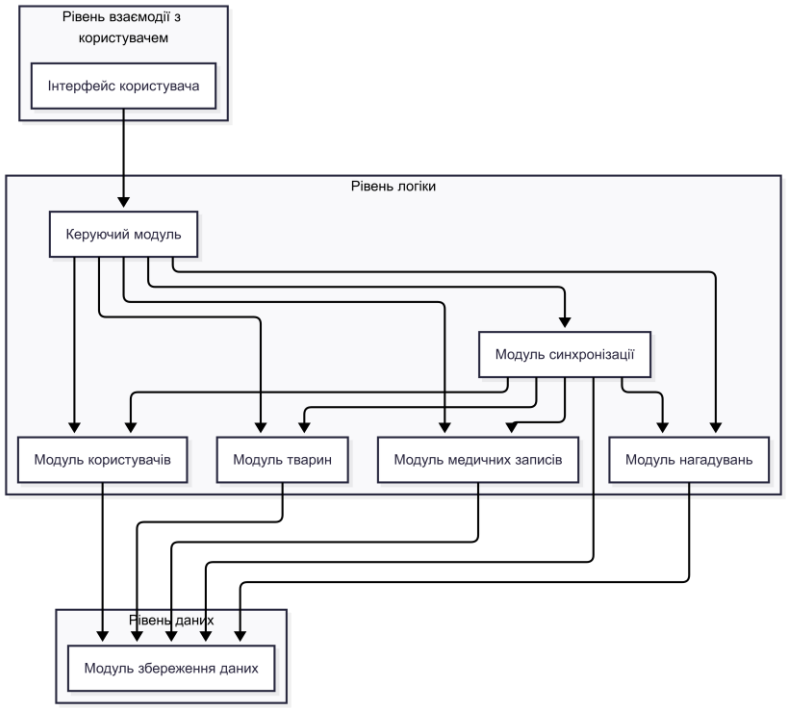


Рисунок 2.1.1 – Узагальнена структурна схема архітектури системи

З точки зору інформаційного наповнення система оперує такими основними типами даних:

- дані користувача;
- інформація про тварин;
- медичні записи;
- події та нагадування;
- службові налаштування.

Обробка цих даних здійснюється у межах відповідних модулів із подальшою передачею результатів до рівня взаємодії з користувачем.

У статичному представленні система розглядається як набір взаємопов’язаних модулів, що утворюють єдину структуру. У динамічному аспекті система функціонує як послідовність обробки запитів користувача, які проходять через рівень логіки та взаємодіють із підсистемами обробки даних.

Інтерфейсна модель визначає набір функцій, доступних для взаємодії між модулями, що забезпечує їх відносну незалежність.

Архітектура системи також формується з урахуванням вимог до функціональності. Зокрема, необхідність роботи з кількома профілями тварин, ведення медичної історії, створення нагадувань і забезпечення доступу до даних у різних умовах використання впливають на структуру системи та взаємодію її компонентів.

Запропонований підхід дозволяє врахувати ці вимоги без значного ускладнення загальної архітектури.

Таким чином, сформована архітектура є достатньо гнучкою для подальшого розвитку системи та водночас не є надмірно складною. Вона створює основу для подальшого детального проєктування, у межах якого буде конкретизовано внутрішню структуру модулів, логіку їх роботи та організацію даних.

2.2 Проєктування модулів

Перед переходом до детального проєктування системи необхідно сформувати узагальнене уявлення про її функціонування, що дозволяє визначити основні процеси, їх послідовність та взаємозв’язки. Такий підхід забезпечує цілісність архітектурних рішень і дозволяє уникнути неузгодженостей між окремими компонентами системи на подальших етапах розробки. З цією метою доцільно застосувати функціональну декомпозицію.

Для опису функціональної структури системи використано підхід методології IDEF0, який дозволяє представити систему у вигляді ієрархії функцій із чітким визначенням вхідних і вихідних даних, керуючих впливів та механізмів виконання [17].

На верхньому рівні система розглядається як єдиний функціональний блок, що реалізує процес організації та підтримки догляду за домашніми тваринами. Користувач виступає основним джерелом вхідних даних, які включають інформацію про тварин, медичні записи, параметри подій та налаштування. Результатом роботи системи є структурована інформація, що відображається користувачу, а також повідомлення про заплановані дії [18].

Загальна функціональна модель системи представлена на рисунку 2.2.1.

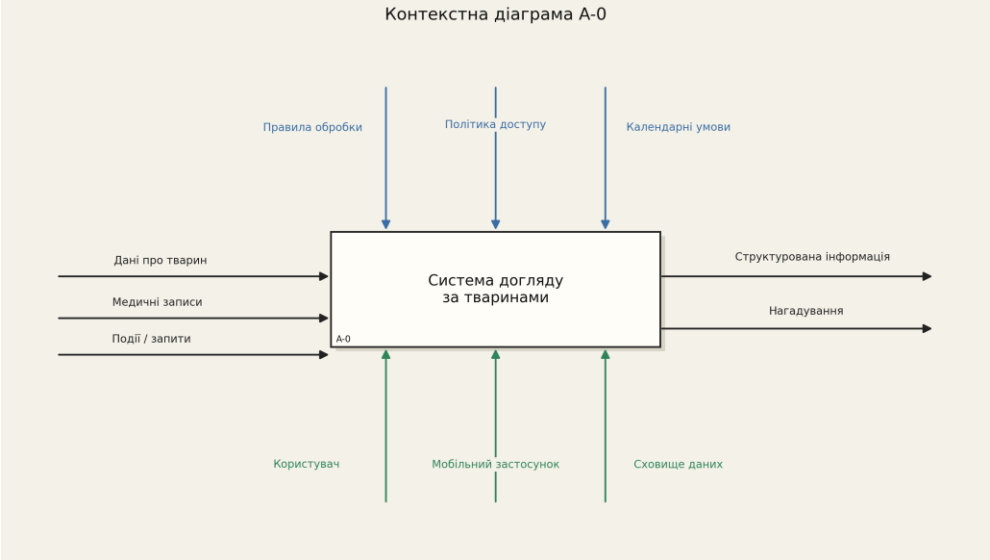


Рисунок 2.2.1 – Контекстна діаграма функціонування системи

Для більш детального аналізу система декомпонується на сукупність взаємопов’язаних процесів, кожен із яких відповідає за окрему частину функціональності. Основними функціональними компонентами є процеси управління користувачами, обробки інформації про тварин, ведення медичних записів, формування нагадувань, обробки та збереження даних, а також узгодження інформації. Кожен із цих процесів виконує визначену функцію та взаємодіє з іншими для забезпечення цілісності системи.

Взаємодія між процесами здійснюється шляхом передачі даних від одного функціонального блоку до іншого. Початковим етапом є введення даних користувачем, після чого вони передаються до відповідних процесів обробки. У разі необхідності виконуються додаткові запити до інших компонентів системи, що дозволяє отримати або уточнити інформацію. Зокрема, процес формування нагадувань використовує дані, отримані з процесів обробки інформації про тварин та медичних записів. Після завершення обробки результати передаються до підсистеми збереження, де фіксуються для подальшого використання.

Під час проєктування модулів важливим аспектом є узгодженість між компонентами системи. Кожен функціональний модуль виконує окрему частину логіки без дублювання функцій інших компонентів. Такий підхід спрощує структуру системи, зменшує залежність між процесами та полегшує підтримку програмного забезпечення.

Функціональні модулі формуються з урахуванням основних сценаріїв використання застосунку. Модуль управління профілями тварин відповідає за створення, редагування та відображення інформації про домашніх тварин. Цей компонент є одним із центральних у системі, оскільки медичні записи та нагадування використовують пов’язані з ним дані.

Модуль ведення медичних записів забезпечує роботу з інформацією про стан тварини, історію процедур та прийом ліків. Він відповідає за збереження, перегляд і оновлення медичних даних, забезпечуючи їх упорядковану організацію.

Модуль формування нагадувань реалізує створення подій та контроль часу їх виконання. Для роботи він використовує інформацію про тварин і медичні записи, формуючи повідомлення про заплановані дії користувача.

Модуль синхронізації та збереження забезпечує централізовану роботу з даними. Через нього здійснюється отримання, оновлення та передача інформації між компонентами системи. Такий підхід дозволяє підтримувати узгодженість даних та спрощує взаємодію між модулями.

Під час декомпозиції також враховувалася можливість подальшого розширення системи. Структура модулів дозволяє додавати нові функції без значного перепроєктування основних компонентів. Крім того, виділення окремих функціональних блоків зменшує дублювання логіки та забезпечує повторне використання спільних механізмів обробки даних.

Декомпозиція системи на функціональні компоненти представлена на рисунку 2.2.2.

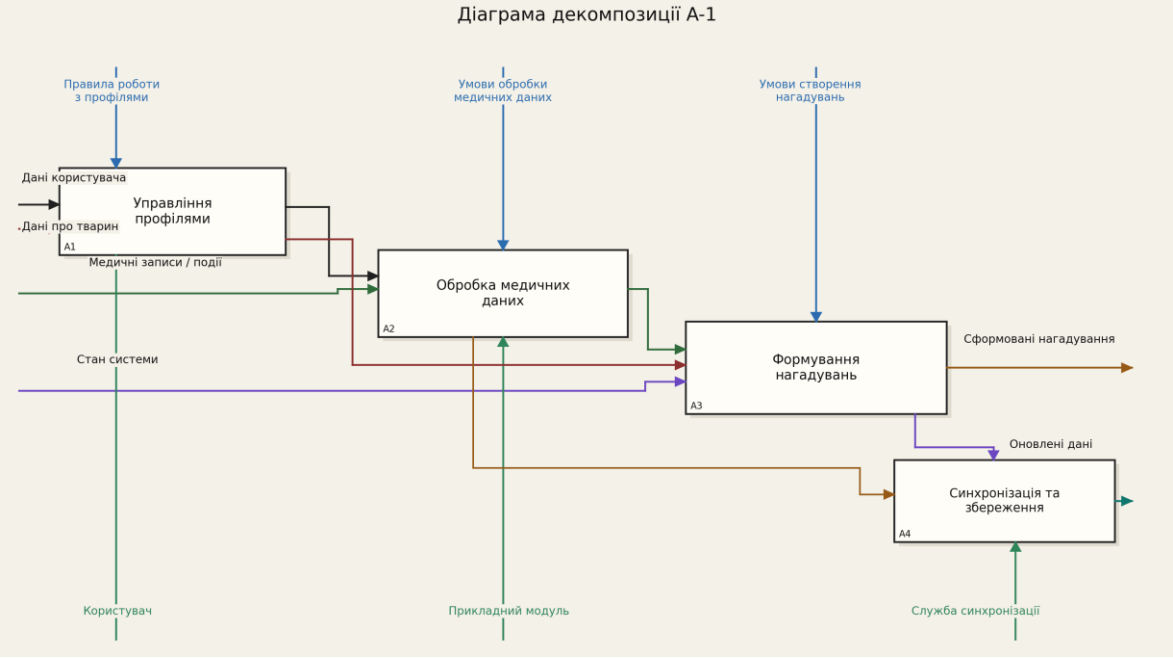


Рисунок 2.2.2 – Декомпозиція системи на функціональні модулі

Для доповнення функціональної декомпозиції розглядається рух даних між основними процесами системи. Діаграма потоків даних дозволяє визначити, які

саме дані передаються між користувачем, процесами обробки та сховищами, уточнюючи логіку взаємодії компонентів [19, 20].

Основними потоками є дані про тварин, медичні записи, параметри нагадувань і налаштування користувача. Вони надходять від користувача до процесів обробки, після чого зберігаються або використовуються для формування результатів. Частина оброблених даних повертається користувачу у вигляді структурованої інформації та повідомлень.

Діаграма потоків даних системи наведена на рисунку 2.2.3.

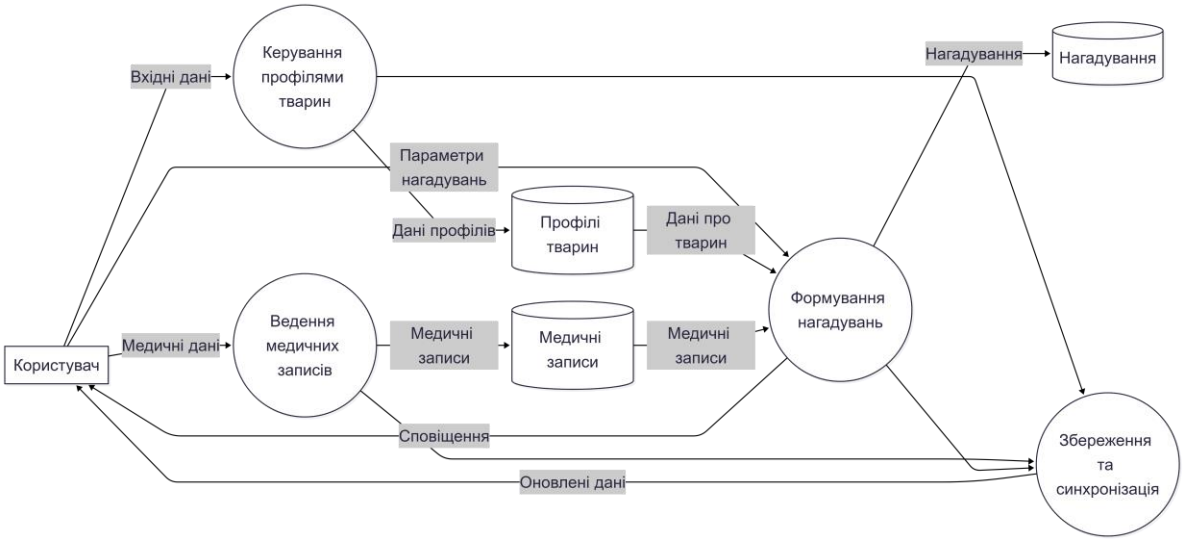


Рисунок 2.2.3 – Діаграма потоків даних системи

Таким чином, функціональна декомпозиція дозволяє чітко визначити основні процеси системи, їх взаємозв’язки та логіку обробки даних. Отримана модель створює основу для подальшого детального проєктування, у межах якого буде конкретизовано внутрішню структуру компонентів, їх поведінку та способи реалізації окремих функцій.

2.3 Детальне проєктування

Детальне проєктування спрямоване на уточнення структури та поведінки системи, визначеної на попередніх етапах аналізу та проєктування модулів. На цьому етапі формалізуються сценарії взаємодії, порядок виконання процесів і логіка обробки даних. Для цього використовується уніфікована мова моделювання UML, яка дозволяє представити систему у вигляді узгодженого набору моделей [21].

UML застосовується для опису структури, поведінки та взаємодії компонентів системи. У межах даного проєкту використовуються діаграми варіантів використання, діаграми послідовностей та блок-схеми (діаграми діяльності), що забезпечують перехід від загального опису функціональності до деталізації окремих процесів [23].

Основним актором системи є користувач, який взаємодіє із застосунком з метою управління інформацією про домашніх тварин. Користувач має обліковий запис та доступ до власних даних. До його функцій належать створення та редагування профілів тварин, ведення медичних записів, створення нагадувань і перегляд збереженої інформації.

Система обробляє введені дані, зберігає їх та використовує для формування результатів. У відповідь на дії користувача виконуються відповідні процеси обробки, які завершуються або відображенням результату, або ініціюванням подій. Поведінка системи визначається послідовністю дій, що виконуються залежно від стану даних і умов виконання.

До основних можливостей системи належать управління профілями тварин, ведення медичної інформації, створення нагадувань і доступ до даних. Взаємодія між користувачем і системою реалізується через набір сценаріїв, які визначають логіку виконання дій.

Діаграма варіантів використання відображає основні сценарії взаємодії користувача із системою.

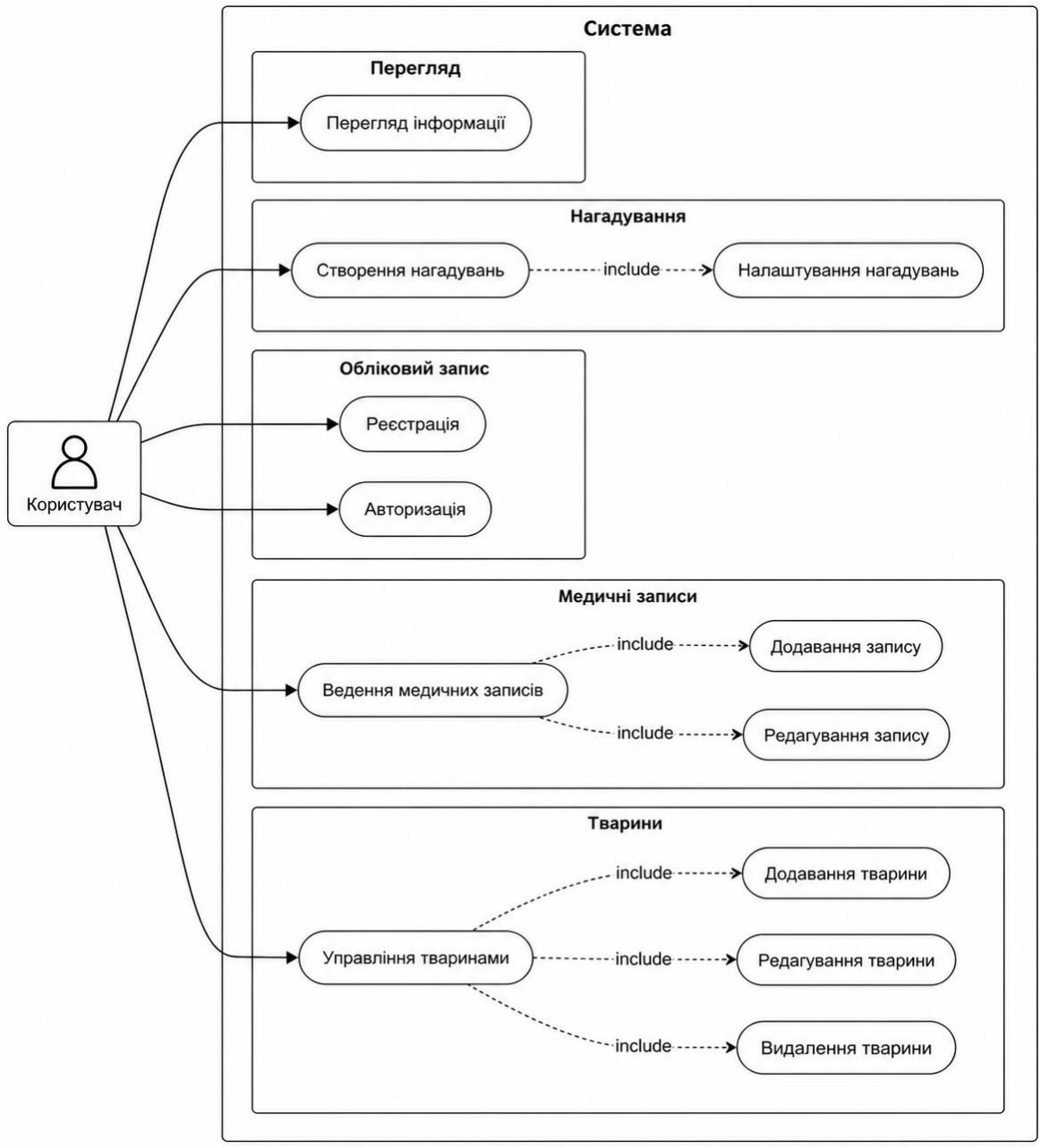


Рисунок 2.3.1 – Діаграма варіантів використання

Серед визначених сценаріїв ключовими є управління профілями тварин, ведення медичних записів і створення нагадувань, оскільки вони охоплюють основні процеси системи та визначають її призначення. Ці сценарії підлягають подальшій деталізації.

Для відображення взаємодії між об’єктами у межах окремих сценаріїв використовуються діаграми послідовностей, які показують порядок обміну повідомленнями [22].

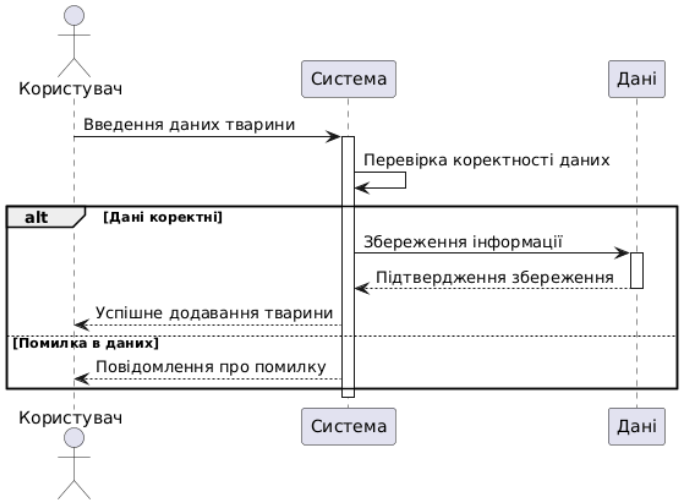


Рисунок 2.3.2 – Діаграма послідовності додавання тварини

Для деталізації логіки виконання процесів використовуються діаграми діяльності, які відображають послідовність дій та умови переходів.

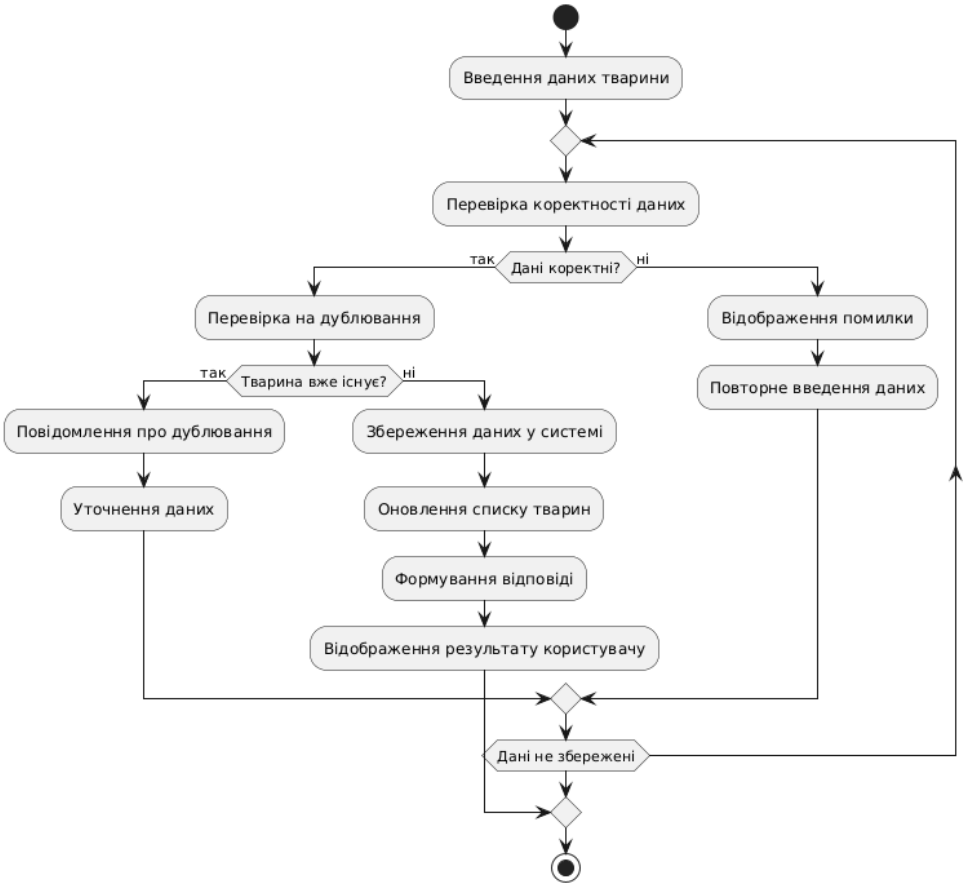


Рисунок 2.3.3 – Діаграма діяльності процесу обробки даних

Для відображення статичної структури системи побудовано діаграму класів, яка ілюструє основні сутності та зв'язки між ними і зображено на рисунку 2.3.4.

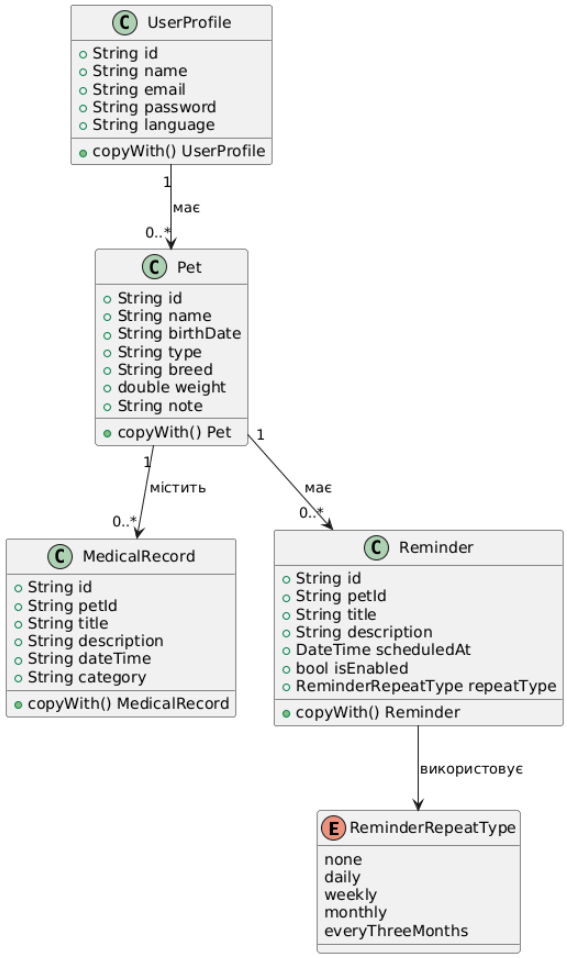


Рисунок 2.3.4 – Діаграма класів

Застосовані діаграми взаємодоповнюють одна одну: діаграма варіантів використання визначає основні сценарії, діаграми послідовностей уточнюють порядок взаємодії між компонентами, а діаграми діяльності деталізують внутрішню логіку процесів. Такий підхід дозволяє перейти від загального опису системи до конкретизації її поведінки.

Моделі адаптовані до специфіки застосунку, орієнтованого на обробку користувацьких даних і підтримку процесів догляду за домашніми тваринами. Основна увага приділяється послідовності дій користувача та передбачуваності реакції системи.

Таким чином, виконано детальне проєктування поведінки системи, визначено основні сценарії взаємодії та логіку обробки даних. Отримані результати створюють основу для переходу до етапу реалізації програмного забезпечення.

2.4 Проєктування користувацького інтерфейсу

Користувацький інтерфейс є важливою складовою програмного забезпечення, оскільки забезпечує взаємодію користувача із системою та визначає зручність її використання [24].

На етапі проєктування інтерфейсу визначаються структура екранів, логіка навігації та принципи розміщення елементів керування. Основною метою є створення інтуїтивно зрозумілого та простого у використанні середовища, що відповідає функціональним можливостям системи [25].

Проєктування інтерфейсу здійснюється на основі визначених сценаріїв використання. Основна увага приділяється мінімізації кількості дій, необхідних для виконання операцій, а також забезпеченню логічної послідовності переходів між екранами. Інтерфейс організовано таким чином, щоб користувач мав швидкий доступ до ключових функцій без необхідності складної навігації.

Основний функціонал застосунку реалізується через набір взаємопов’язаних екранів, які відображають різні аспекти роботи системи. До них належать головний екран із переліком тварин, екран профілю тварини, екран медичних записів та екран нагадувань.

Кожен із цих екранів виконує окрему функцію та забезпечує доступ до відповідних даних і операцій. При цьому дії користувача прив’язані до контексту конкретного екрана, що дозволяє уникнути дублювання елементів керування та зменшити навантаження на інтерфейс.

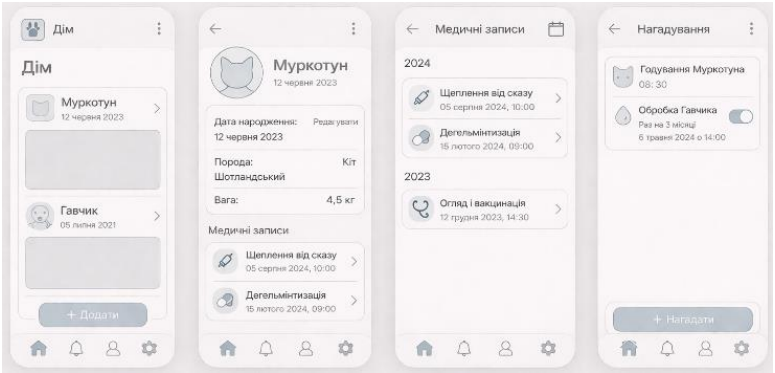


Рисунок 2.4.1 – Макет основного функціоналу застосунку

Головний екран відображає список тварин та виступає центральною точкою взаємодії. Саме на цьому екрані реалізовано можливість додавання нових профілів, що відповідає логіці використання системи. Екран профілю тварини надає доступ до детальної інформації та дозволяє виконувати її редагування. Екран медичних записів забезпечує перегляд історії подій у структурованому вигляді, а екран нагадувань призначений для управління запланованими діями.

Окрему групу становлять екрани авторизації, які забезпечують доступ користувача до системи. До них належать екрани реєстрації та входу. Вони реалізують введення облікових даних та перевірку їх коректності. Особливістю цих екранів є наявність механізмів валідації, що дозволяють виявляти помилки введення та відображати відповідні повідомлення користувачу.

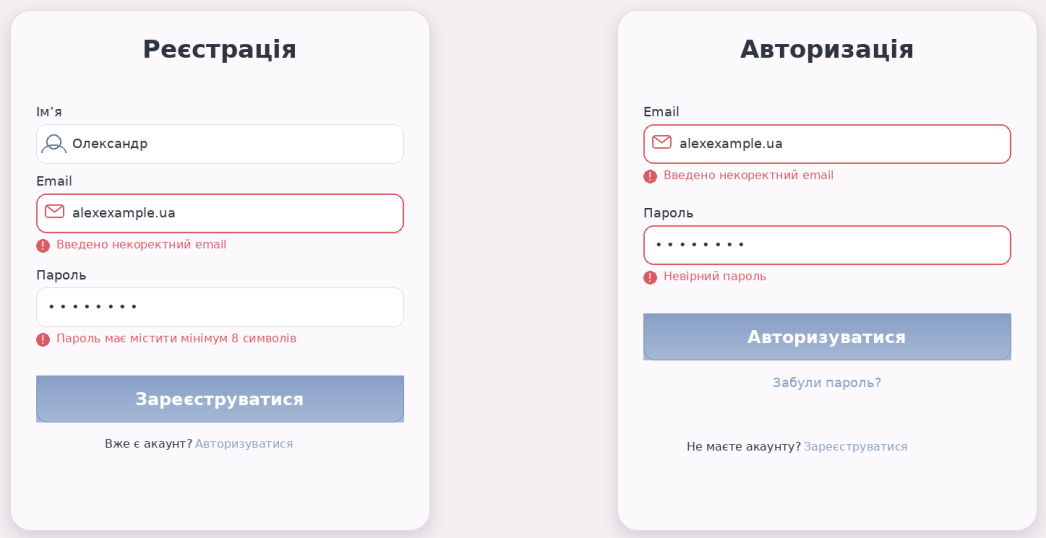


Рисунок 2.4.2 – Макет функціоналу авторизації застосунку

Навігація між основними розділами застосунку реалізована за допомогою нижньої панелі, яка містить доступ до головного екрана, нагадувань, профілю користувача та налаштувань. Такий підхід забезпечує швидке перемикання між функціональними частинами системи та підвищує зручність використання.

Під час проєктування інтерфейсу було прийнято рішення використовувати мінімалістичний підхід до оформлення, що передбачає обмежену кількість візуальних елементів і акцент на функціональності. Це дозволяє спростити реалізацію інтерфейсу та забезпечити його гнучкість для подальших змін [26].

Інтерфейс застосунку проєктується з урахуванням можливості використання на мобільних пристроях із різними розмірами екранів. Для цього застосовуються адаптивні принципи розміщення елементів інтерфейсу, що дозволяє зберігати цілісність структури та зручність взаємодії незалежно від параметрів пристрою.

Таким чином, спроєктований користувацький інтерфейс відповідає визначеним вимогам, забезпечує логічну структуру взаємодії та створює основу для реалізації програмного забезпечення.

2.5 Аналіз та вибір технологій для реалізації

Після завершення етапів аналізу та проєктування постає необхідність вибору технологій, за допомогою яких буде реалізовано розроблювану систему. Саме на цьому етапі визначається набір засобів, що забезпечуватимуть практичне втілення закладеної архітектури, функціональних вимог та сценаріїв взаємодії.

Вибір технологічного стеку безпосередньо впливає на продуктивність системи, можливість її подальшого масштабування, зручність супроводу, а також швидкість розробки й тестування. Тому використані інструменти мають відповідати не лише загальним тенденціям сучасної розробки, а й специфічним вимогам конкретного проєкту.

Формування вимог до технологічного стеку здійснюється на основі функціональних і нефункціональних вимог системи. Оскільки розроблюваний застосунок орієнтований на мобільні пристрої, однією з ключових вимог є кросплатформеність, що дозволяє охопити різні категорії користувачів без необхідності створення окремих реалізацій для кожної платформи.

Важливим фактором також є швидкість розробки та прототипування, оскільки проєкт реалізується у форматі MVP і повинен забезпечувати отримання працездатного результату в обмежені строки. Додатково система повинна підтримувати інтеграцію з хмарними сервісами, роботу з даними в реальному часі,

можливість локального збереження інформації для офлайн-доступу, а також механізми сповіщень, що є критичними для реалізації нагадувань про заплановані події. Сукупність цих вимог визначає критерії вибору інструментів реалізації.

Вибір мови програмування та клієнтської платформи пов’язаний із необхідністю забезпечення достатньої продуктивності, підтримки мобільних інтерфейсів, кросплатформеності та прийнятної швидкості розробки. У межах даного проєкту доцільно розглянути три основні підходи: нативну розробку окремо для кожної мобільної платформи, гібридну веб-орієнтовану модель та сучасний кросплатформений UI-підхід із єдиною кодовою базою.

Таблиця 2.2 – Порівняння підходів до реалізації клієнтської частини

Критерій	Нативний підхід	Гібридний веб-підхід	Кросплатформений UI-підхід
Продуктивність	Висока	Середня	Висока
Кросплатформеність	Низька	Висока	Висока
Швидкість розробки	Середня	Висока	Висока
Єдина кодова база	Ні	Так	Так
Якість мобільного UI	Висока	Середня	Висока
Зручність підтримки	Середня	Висока	Висока

Аналіз наведених підходів показує, що нативна модель забезпечує високий рівень продуктивності, однак потребує подвійної реалізації клієнтської логіки та інтерфейсу. Гібридний веб-підхід спрощує створення єдиної версії застосунку, проте може поступатися за якістю взаємодії та продуктивністю. Найбільш збалансованим для даного проєкту є використання кросплатформеного UI-фреймворку з єдиною мовою для всієї клієнтської логіки, оскільки такий підхід поєднує швидкість розробки, достатню продуктивність і можливість формування повноцінного мобільного інтерфейсу [27, 28].

Для реалізації клієнтської частини обрано Flutter, який забезпечує кросплатформеність, підтримку єдиної кодової бази та достатню продуктивність для мобільних застосунків.

Окремим питанням є вибір підходу до серверної частини. Наявність backend-компонента є необхідною, оскільки система повинна забезпечувати автентифікацію користувачів, збереження та синхронізацію даних, а також централізований доступ до інформації. У загальному випадку можна виділити два основні варіанти: використання власного серверного рішення або застосування моделі Backend-as-a-Service.

Таблиця 2.3 – Порівняння підходів до реалізації серверної частини

Критерій	Власний сервер	Backend-as-a-Service
Гнучкість реалізації	Висока	Середня
Швидкість розгортання	Низька	Висока
Необхідність адміністрування	Висока	Низька
Готові механізми автентифікації	Ні	Так
Інтеграція з БД	Потребує окремої реалізації	Вбудована
Доцільність для MVP	Середня	Висока

Як видно з таблиці, власний серверний підхід надає більше контролю над логікою системи, проте вимагає більших витрат часу на налаштування, підтримку та супровід. Для проєкту прикладного рівня, орієнтованого на швидку реалізацію базової функціональності, доцільнішим є використання BaaS-підходу. Така модель дозволяє швидко розгорнути серверну частину, використати готові сервіси авторизації та організувати роботу з базою даних [29, 30].

Взаємодія з хмарним середовищем здійснюється через API Firebase, що забезпечує передачу та синхронізацію даних між клієнтською частиною та віддаленим сховищем.

Вибір підходу до збереження даних визначається специфікою застосунку. Оскільки система повинна забезпечувати як централізований доступ до інформації, так і можливість роботи в умовах нестабільного з'єднання, доцільно поєднати хмарне та локальне зберігання. Хмарне сховище необхідне для синхронізації та доступу до даних з різних пристроїв.

Локальне зберігання використовується для підтримки офлайн-доступу, збереження проміжної інформації та зменшення залежності від мережі. Окремо доцільно виділити легке сховище для збереження налаштувань користувача, яке не потребує складної структури даних. З огляду на характер інформації, що використовується у застосунку, доцільним є використання NoSQL-моделі у хмарному середовищі, локальної бази даних для оперативного доступу та окремого легкого механізму для параметрів конфігурації.

Не менш важливим є підхід до управління станом застосунку. У мобільних системах, що мають кілька функціональних модулів та працюють із змінними даними, питання синхронізації стану між інтерфейсом і логікою є одним із ключових. Відсутність централізованого підходу ускладнює підтримку застосунку, призводить до дублювання логіки та знижує передбачуваність поведінки системи. З цієї причини доцільно використовувати просте та легке рішення для управління станом, яке не створює надмірної складності, але забезпечує централізовану передачу даних між компонентами.

Важливу роль у межах даної системи відіграє механізм сповіщень, оскільки нагадування про події догляду є однією з основних функцій застосунку. У загальному випадку можна використовувати серверні push-сповіщення або локальні сповіщення. Серверний підхід забезпечує ширші можливості керування подіями з боку віддаленого середовища, однак вимагає складнішої організації інфраструктури. Для базової версії системи доцільніше використати локальні

сповіщення, оскільки вони безпосередньо відповідають призначенню застосунку, простіші у впровадженні та не потребують складної серверної логіки.

Під час вибору технологій також враховувалися вимоги до продуктивності та ефективного використання ресурсів мобільного пристрою. Використання локального кешування даних дозволяє зменшити кількість звернень до хмарного середовища, а застосування асинхронної обробки запитів забезпечує плавність роботи інтерфейсу без блокування основного потоку виконання.

Окрім основних технологій реалізації, необхідно визначити й додаткові інструменти, що підтримують процес розробки. Насамперед це стосується системи контролю версій, яка є обов'язковою для фіксації змін, контролю історії розробки та забезпечення безпечної роботи з кодовою базою. Використання хмарного репозиторію дозволяє зберігати резервні копії проєкту та організовувати спільну роботу.

Для управління процесом розробки доцільно застосовувати таск-трекер, який дозволяє структурувати завдання, контролювати їх виконання та підтримувати послідовність реалізації функціональності.

У частині інтерфейсу важливим є використання готової дизайн-системи, що прискорює створення узгодженого UI та зменшує витрати часу на проєктування стандартних елементів.

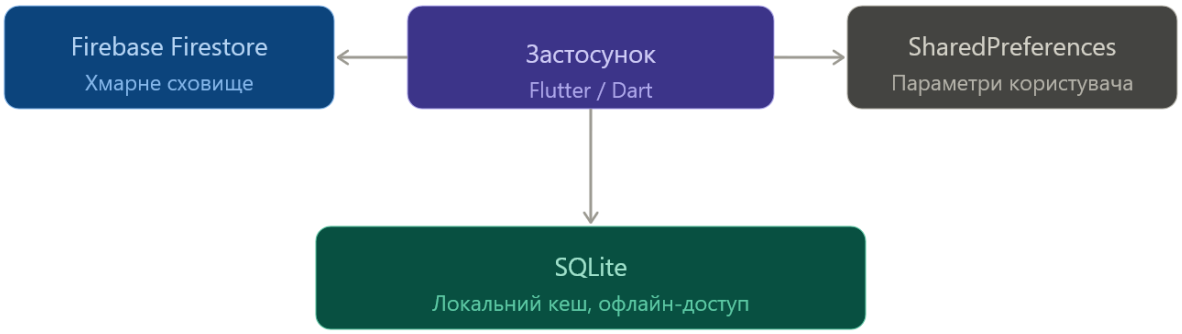


Рисунок 2.5.1 – Організація зберігання даних

На основі проведеного аналізу може бути сформовано остаточний технологічний стек проєкту [31].

Таблиця 2.4 – Підібраний інструментарій

Категорія	Технологія	Призначення
Мова програмування	Dart	Реалізація клієнтської логіки застосунку
UI-фреймворк	Flutter	Побудова кросплатформеного мобільного інтерфейсу
Хмарне середовище	Firebase Firestore	Зберігання та синхронізація даних
Автентифікація	Firebase Authentication	Реалізація авторизації користувачів
Локальне сховище	SQLite	Кешування та офлайн-зберігання даних
Налаштування	SharedPreferences	Збереження параметрів застосунку
Сповіщення	Flutter Local Notifications	Формування локальних нагадувань
Контроль версій	Git	Відстеження змін у проєкті
Хмарний репозиторій	GitHub	Зберігання та резервування кодової бази

Обраний стек є доцільним для реалізації проєкту, оскільки забезпечує кросплатформеність, підтримку локального та хмарного зберігання даних, а також реалізацію системи нагадувань.

Використання єдиної клієнтської логіки спрощує розробку та подальшу підтримку застосунку без надмірного ускладнення архітектури.

Таким чином, сформований технологічний стек відповідає вимогам системи, узгоджується з її архітектурою та створює основу для реалізації програмного забезпечення і впровадження розроблених проєктних рішень.

2.6 Висновки проєктування програмного продукту

У результаті проєктування сформовано цілісну модель системи, що охоплює її архітектуру, функціональну структуру, поведінку та користувацький інтерфейс. Визначено доцільність використання комбінованого підходу, який поєднує клієнт-серверну та багаторівневу архітектури.

Виконано функціональну декомпозицію системи із застосуванням IDEF0 та діаграм потоків даних, що дозволило визначити основні модулі, їх взаємозв'язки та логіку обробки інформації.

Детальне проєктування з використанням UML уточнило сценарії взаємодії та послідовність виконання процесів.

Побудовані діаграми варіантів використання, послідовностей та діяльності забезпечують повноту опису поведінки системи та охоплюють усі ключові сценарії взаємодії користувача із застосунком

Спроєктовано користувацький інтерфейс, орієнтований на простоту використання та мінімізацію дій користувача, що відповідає вимогам системи та обмеженням формату MVP.

Обрано технологічний стек, який забезпечує кросплатформеність, швидкість розробки та підтримку як онлайн-, так і офлайн-режиму роботи.

Запропоновані проєктні рішення враховують функціональні та нефункціональні вимоги системи, забезпечують узгодженість між її компонентами та створюють передумови для стабільної роботи застосунку в умовах змінних даних і різних сценаріїв використання.

Таким чином, етап проєктування програмного продукту завершено, а отримані рішення є узгодженими, відповідають поставленим вимогам та створюють основу для переходу до етапу реалізації системи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОДУКТУ

3.1 Реалізація програмної частини продукту

У даному підрозділі розглянуто реалізацію програмної логіки мобільного застосунку. Логіка системи побудована на обробці дій користувача, перевірці введених даних та подальшому оновленні стану застосунку. У межах MVP-проєкту реалізовано базові функціональні сценарії, пов’язані з авторизацією користувача, роботою з даними та системою нагадувань. Взаємодія з даними виконується асинхронно, що дозволяє уникнути блокування інтерфейсу під час роботи із запитами та локальними сповіщеннями [33].

Програмна логіка реалізована з урахуванням розділення відповідальності між окремими компонентами системи. Інтерфейсні екрани відповідають за взаємодію з користувачем та отримання введених даних, тоді як окремі сервіси та моделі забезпечують їх обробку, збереження та подальше використання. Такий підхід спрощує підтримку програмного коду та дозволяє ізолювати окремі функціональні частини застосунку.

Як приклад реалізації програмної частини розглянуто модуль нагадувань, оскільки він поєднує роботу з даними, обробку подій, взаємодію з інтерфейсом користувача та використання системних механізмів мобільної платформи.

Для представлення нагадування використовується окрема модель даних, яка містить основні параметри: ідентифікатор, прив’язку до тварини, текстові поля, дату виконання, стан активності та тип повторення. Для опису періодичності використовується перелік типів повторення.

Код:

```
enum ReminderRepeatType {
  none,
  daily,
  weekly,
  monthly,
  everyThreeMonths,
}
```

```
class Reminder {
  final String id;
  final String petId;
  final String title;
  final String description;
  final DateTime scheduledAt;
  final bool isEnabled;
  final ReminderRepeatType repeatType;
}
```

Такий підхід дозволяє працювати з нагадуванням як із цілісним об’єктом та спрощує передачу даних між компонентами застосунку. Крім того, використання окремої моделі забезпечує узгодженість структури даних під час створення, редагування та збереження нагадувань.

Для роботи зі сповіщеннями реалізовано окремий сервіс NotificationService, який відповідає за ініціалізацію механізму локальних повідомлень, планування сповіщень та їх скасування. Для реалізації використовується бібліотека flutter_local_notifications [34].

Код:

```
Future<void> scheduleReminderNotification({
  required Reminder reminder,
}) async {
  final nextScheduledAt = getNextValidScheduledAt(
    scheduledAt: reminder.scheduledAt,
    repeatType: reminder.repeatType,
  );

  if (nextScheduledAt == null) {
    throw Exception(
      'Неможливо запланувати нагадування: дата і час уже минули.',
    );
  }
  final scheduledTz = tz.TZDateTime.from(nextScheduledAt, tz.local);
  await flutterLocalNotificationsPlugin.zonedSchedule(
    id: reminder.id.hashCode,
    title: reminder.title,
    body: reminder.description.isEmpty
        ? 'Нагадування для вашої тварини'
        : reminder.description,
    scheduledDate: scheduledTz,
```

```
      notificationDetails: _notificationDetails(),
      androidScheduleMode: AndroidScheduleMode.exactAllowWhileIdle,
    );
  }
```

Наведений фрагмент демонструє процес створення локального сповіщення. Перед плануванням виконується перевірка дати нагадування та визначення найближчого допустимого моменту виконання. Після цього формується локальне повідомлення, яке передається операційній системі.

Окрему увагу приділено підтримці повторюваних нагадувань. Для цього реалізовано механізм автоматичного визначення наступної дати виконання залежно від типу повторення.

Код:

```
DateTime? getNextValidScheduledAt({
  required DateTime scheduledAt,
  required ReminderRepeatType repeatType,
}) {
  final now = DateTime.now();
  if (repeatType == ReminderRepeatType.none) {
    return scheduledAt.isAfter(now) ? scheduledAt : null;
  }
  var next = scheduledAt;
  while (!next.isAfter(now)) {
    switch (repeatType) {
      case ReminderRepeatType.daily:
        next = next.add(const Duration(days: 1));
        break;
      case ReminderRepeatType.weekly:
        next = next.add(const Duration(days: 7));
        break;
      case ReminderRepeatType.monthly:
        next = DateTime(
          next.year,
          next.month + 1,
          next.day,
          next.hour,
          next.minute,
        );
        break;
      case ReminderRepeatType.everyThreeMonths:
        next = DateTime(
          next.year,
```

```
            next.month + 3,
            next.day,
            next.hour,
            next.minute,
          );
          break;
        case ReminderRepeatType.none:
          return null;
      }
    }
    return next;
  }
```

Такий підхід дозволяє автоматично підтримувати актуальність повторюваних нагадувань без необхідності повторного введення дати користувачем. Додатково це забезпечує коректну роботу нагадувань навіть у випадках, коли користувач тимчасово не відкривав застосунок.

Створення нагадування реалізовано через окремий екран введення даних. Під час збереження виконується перевірка введених параметрів, після чого формується об’єкт нагадування та викликається сервіс сповіщень.

Код:

```
Future<void> _saveReminder() async {
  if (selectedPetId == null) {
    _showError('Оберіть тварину');
    return;
  }

  final title = titleController.text.trim();
  final description = descriptionController.text.trim();
  final scheduledAt = _buildScheduledAt();
  if (title.isEmpty) {
    _showError('Введіть назву нагадування');
    return;
  }
  if (scheduledAt == null) {
    _showError('Оберіть дату і час');
    return;
  }
  final reminder = Reminder(
    id: DateTime.now().millisecondsSinceEpoch.toString(),
    petId: selectedPetId!,
    title: title,
```

```
      description: description,
      scheduledAt: scheduledAt,
      isEnabled: true,
      repeatType: selectedRepeatType,
    );
    await NotificationService.instance.scheduleReminderNotification(
      reminder: reminder,
    );
  }
```

У даному випадку логіка побудована на попередній перевірці введених даних. Після створення об’єкта нагадування виконується його збереження та планування локального сповіщення. Перевірка параметрів до виконання основної операції дозволяє уникнути створення некоректних або неповних записів.

Для редагування та керування станом нагадувань реалізовано окремі механізми оновлення й перемикання активності. Якщо нагадування вимикається, відповідне сповіщення скасовується. При повторному ввімкненні система знову виконує його планування [35].

Код:

```
Future<void> _toggleReminder(Reminder reminder, bool value) async {
  final updatedReminder = reminder.copyWith(isEnabled: value);
  if (value) {
    await NotificationService.instance.scheduleReminderNotification(
      reminder: updatedReminder,
    );
  } else {
    await NotificationService.instance.cancelNotification(
      reminder.id.hashCode,
    );
  }
}
```

Такий підхід дозволяє забезпечити узгодженість між внутрішнім станом нагадування та системними сповіщеннями мобільного пристрою. Крім того, користувач отримує можливість швидко керувати активністю нагадувань без необхідності повторного створення або редагування запису.

Отже, у межах програмної реалізації було створено базові механізми роботи мобільного застосунку, пов’язані з обробкою даних, взаємодією з користувачем та використанням системних можливостей мобільної платформи. Реалізований модуль нагадувань демонструє використання моделей даних, асинхронної обробки, локальних сповіщень та керування станом застосунку. Запропонована структура є достатньою для реалізації MVP-версії продукту та може бути розширена в подальших версіях системи.

3.2 Реалізація розмітки мобільного застосунку

У даному підрозділі розглянуто реалізацію користувацького інтерфейсу мобільного застосунку, який побудовано на основі ієрархії віджетів. Такий підхід дозволяє формувати гнучку структуру екранів, де кожен елемент інтерфейсу є окремим компонентом і може бути перевикористаний у застосунку [32].

Основою розмітки виступає контейнер верхнього рівня, який задає загальну структуру екрана, включаючи верхню панель, основний вміст та нижню навігацію. Для побудови інтерфейсу використовується вкладена структура контейнерів, що дозволяє організувати розташування елементів за допомогою вертикальних і горизонтальних груп [36, 37].

Головний екран (“Дім”) реалізовано у вигляді вертикального списку, що містить інформаційний блок та перелік тварин. Інформаційний блок оформлено як окремий контейнер із візуальним виділенням, у якому відображаються узагальнені дані. Кожен елемент списку тварин представлений у вигляді картки з базовою інформацією.

Код:

```
Scaffold(
  appBar: AppBar(title: const Text('Дім')),
  body: ListView(
    padding: const EdgeInsets.all(16),
    children: [
      SummaryCard(),
```

```
        const SizedBox(height: 16),
        ...pets.map((pet) => PetCard(pet: pet)),
      ],
    ),
  )
```

Такий підхід дозволяє динамічно формувати список елементів та легко масштабувати інтерфейс при зміні кількості даних. Для відображення списків використовується прокручуваний контейнер ListView, який автоматично адаптує область вмісту до розміру екрана та дозволяє коректно працювати із великою кількістю елементів без переповнення інтерфейсу.

Екран нагадувань побудований за аналогічним принципом, однак кожен елемент списку додатково містить перемикач стану, що дозволяє вмикати або вимикати нагадування без переходу до редагування. Для цього використовується компонент перемикача, інтегрований у картку.

Код:

```
Switch(
  value: reminder.isEnabled,
  onChanged: (value) {
    onToggle(reminder, value);
  },
)
```

Це забезпечує швидку взаємодію з функціоналом без ускладнення інтерфейсу.

Екран профілю реалізовано у вигляді централізованого контейнера, який містить аватар користувача, основну інформацію та кнопку виходу. Вирівнювання елементів виконано по центру, що забезпечує простоту сприйняття.

Екран налаштувань містить окремі блоки з параметрами, які оформлені у вигляді карток. Для вибору мови використовується випадаючий список, що дозволяє змінювати значення без переходу на інші екрани.

Код:

```
DropdownButton<String>(
  value: selectedLanguage,
  items: languages.map((lang) {
    return DropdownMenuItem(
      value: lang,
      child: Text(lang),
    );
  }).toList(),
  onChanged: (value) {
    setState(() {
      selectedLanguage = value!;
    });
  },
)
```

Екрани авторизації та реєстрації побудовані з використанням форми, що містить поля введення та кнопки дій. Для кожного поля використовується окремий компонент введення з контролером, що дозволяє зчитувати та перевіряти введені значення.

Код:

```
TextField(
  controller: emailController,
  decoration: const InputDecoration(
    labelText: 'Email',
  ),
)
```

Нижня навігаційна панель реалізована як окремий компонент, що забезпечує швидкий доступ до основних розділів застосунку. Вона містить фіксований набір вкладок: головний екран, нагадування, профіль та налаштування.

Код:

```
BottomNavigationBar(
  currentIndex: currentIndex,
  onTap: onTap,
  items: const [
    BottomNavigationBarItem(icon: Icon(Icons.home), label: 'Дім'),
    BottomNavigationBarItem(icon:     Icon(Icons.notifications),     label:
'Нагадування'),
    BottomNavigationBarItem(icon: Icon(Icons.person), label: 'Профіль'),
```

```
          BottomNavigationBarItem(icon:       Icon(Icons.settings),       label:
'Налаштування'),
        ],
      )
```

У цілому розмітка мобільного застосунку реалізована з урахуванням простоти, зрозумілості та можливості подальшого розширення. Використання компонентного підходу дозволяє уникнути дублювання коду та забезпечує зручність підтримки інтерфейсу [38]. Побудова інтерфейсу на базі адаптивних компонентів Flutter також забезпечує коректне відображення застосунку на мобільних пристроях із різними розмірами екранів.

3.3 Розроблення бази даних

У межах мобільного застосунку реалізовано систему зберігання даних, що поєднує хмарну NoSQL базу даних Firebase Firestore та локальне сховище на базі SQLite. Така організація дозволяє забезпечити централізоване зберігання інформації з можливістю її кешування для підвищення продуктивності [39].

Проєктування бази даних виконано з урахуванням основних сутностей предметної області та зв’язків між ними. Структура побудована навколо користувача як центрального елемента системи, до якого прив’язані профілі тварин, медичні записи та нагадування. Це дозволяє логічно згрупувати дані та забезпечити їх узгоджене використання в межах застосунку.

У Firestore дані організовані у вигляді колекцій users, pets, medical_records та reminders. Кожна колекція відповідає окремій сутності та містить документи з відповідними атрибутами.

У межах поточної реалізації механізм доступу користувача реалізовано у спрощеному вигляді відповідно до формату MVP. При цьому архітектура системи передбачає інтеграцію з сервісами автентифікації Firebase, що дозволить у подальшому відокремити обробку облікових даних від основної логіки застосунку та підвищити рівень безпеки. Взаємодія застосунку з віддаленим сховищем даних

реалізується через Firebase SDK, який забезпечує доступ до Firestore без необхідності створення окремого REST API. Для виконання операцій використовуються методи collection(), doc(), set(), update(), delete() та get(). Дані передаються у форматі структур типу «ключ–значення», що за своєю організацією відповідає JSON-об’єктам.

Колекція pets містить інформацію про тварин і включає поле userId, що забезпечує зв’язок із користувачем. Медичні записи та нагадування, у свою чергу, прив’язуються до тварини через поле petId. Такий підхід дозволяє організувати вибірку даних без використання складних операцій об’єднання, характерних для реляційних моделей, та спрощує структуру зберігання.

Таким чином, один користувач може мати декілька профілів тварин, а кожна тварина може містити множину медичних записів і нагадувань. Для отримання пов’язаних даних застосовуються запити до відповідних колекцій із використанням ідентифікаторів зв’язку.

Код:

```
final reminders = await FirebaseFirestore.instance
    .collection('reminders')
    .where('petId', isEqualTo: petId)
    .get();
```

Наведений приклад демонструє отримання списку нагадувань, пов’язаних із конкретною твариною. Такий підхід дозволяє формувати вибірки даних без використання складних операцій об’єднання таблиць.

Наступний фрагмент ілюструє збереження даних у вигляді документа, де всі атрибути нагадування зберігаються у межах однієї структури.

Код:

```
await _firestore.collection('reminders').doc(reminder.id).set({
  'petId': reminder.petId,
  'title': reminder.title,
  'description': reminder.description,
  'scheduledAt': reminder.scheduledAt,
  'isEnabled': reminder.isEnabled,
  'repeatType': ReminderRepeatTypeHelper.toCode(reminder.repeatType),
});
```

Основним джерелом для зберігання та синхронізації інформації у застосунку виступає Firebase Firestore, тоді як локальне сховище використовується як допоміжний механізм кешування даних та підтримки часткової працездатності системи без активного мережевого підключення [40].

Для забезпечення стабільної роботи застосунку в умовах нестабільного інтернет-з’єднання передбачено використання локального зберігання даних. SQLite застосовується як локальна база даних для кешування основних сутностей застосунку: профілів тварин, медичних записів та нагадувань. SharedPreferences використовується для збереження простих параметрів користувача, зокрема вибраної мови інтерфейсу та службових налаштувань. Такий підхід дозволяє зменшити кількість звернень до хмарної бази даних, підвищити швидкість відображення інформації та забезпечити часткову працездатність застосунку без активного мережевого підключення.

Також у межах MVP структура бази даних є спрощеною та не включає складних механізмів міграції або версіонування. Для наочного відображення структури даних та зв’язків між сутностями доцільно використати ER-діаграму, яка демонструє взаємозв’язки між користувачами, тваринами, медичними записами та нагадуваннями.

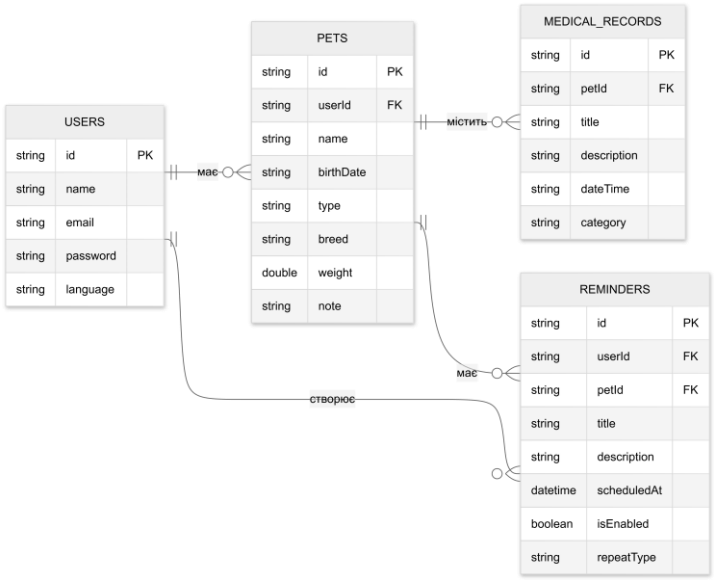


Рисунок 3.3.1 – ER-діаграма бази даних мобільного застосунку

3.4 Керівництво користувача

У даному підрозділі наведено довідкову інформацію щодо використання мобільного застосунку для обліку догляду за домашніми тваринами, а також описано основні сценарії взаємодії користувача із системою. Метою керівництва є забезпечення швидкого освоєння функціоналу застосунку без необхідності додаткового навчання або спеціальної підготовки.

Після встановлення та запуску застосунку користувач потрапляє на екран авторизації. Для початку роботи необхідно створити обліковий запис або виконати вхід за вже існуючими даними.

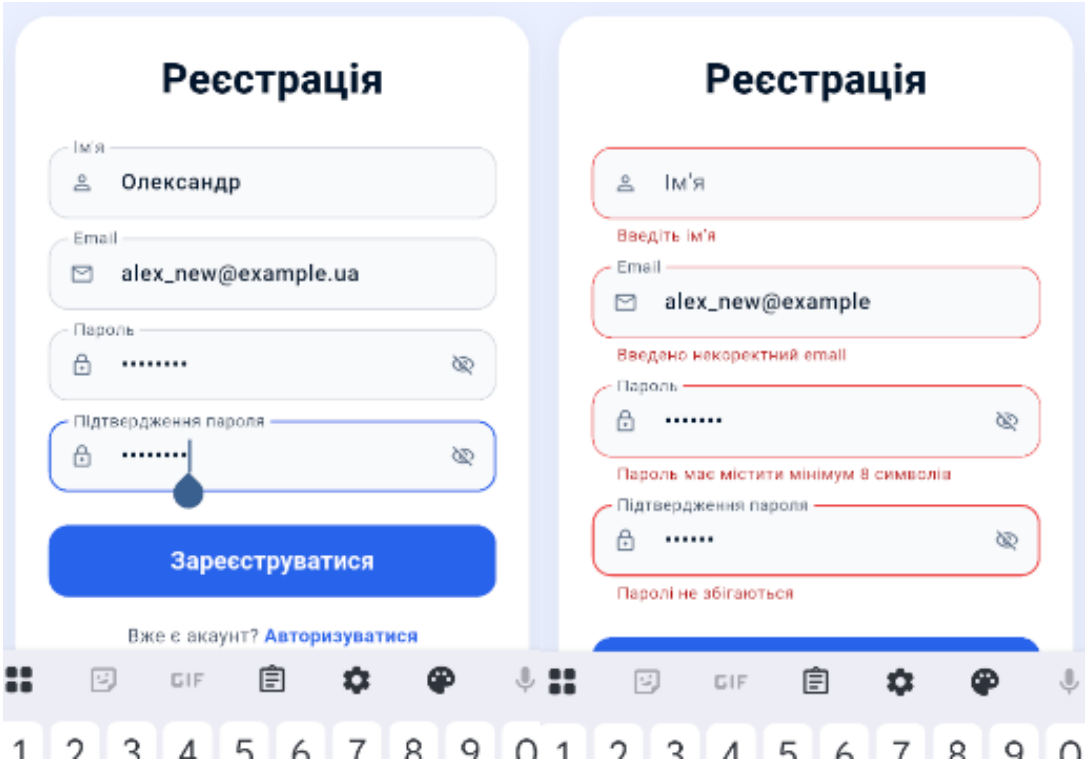


Рисунок 3.4.1 – Форма реєстрації

У процесі реєстрації користувач вводить ім’я, електронну пошту та пароль. У разі некоректного введення даних система відображає відповідні повідомлення, що дозволяє швидко виявити та виправити помилки. Після успішної авторизації відкривається головний екран застосунку.

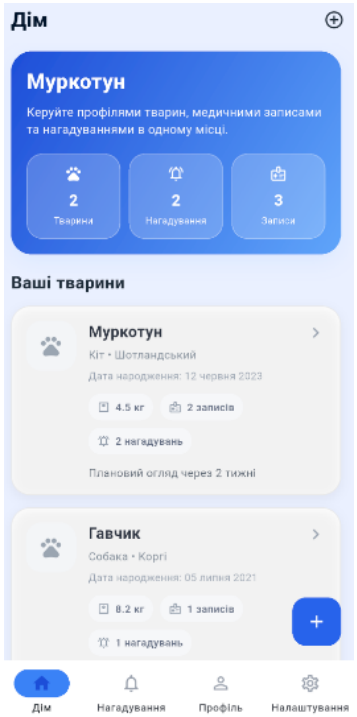


Рисунок 3.4.2 – Основна сторінка «Дім»

Головний екран призначений для відображення списку тварин, доданих користувачем. Кожен елемент списку містить базову інформацію, включаючи ім’я, тип, породу та інші характеристики. Для перегляду детальної інформації достатньо натиснути на відповідний елемент, після чого відкривається екран профілю тварини.

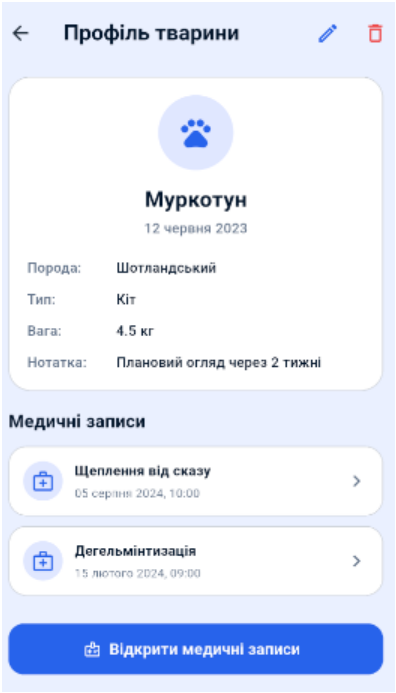


Рисунок 3.4.3 – Сторінка профілю тварини

Додавання нової тварини здійснюється за допомогою кнопки створення, розміщеної на головному екрані. Після переходу до форми введення користувач заповнює необхідні поля та зберігає дані.

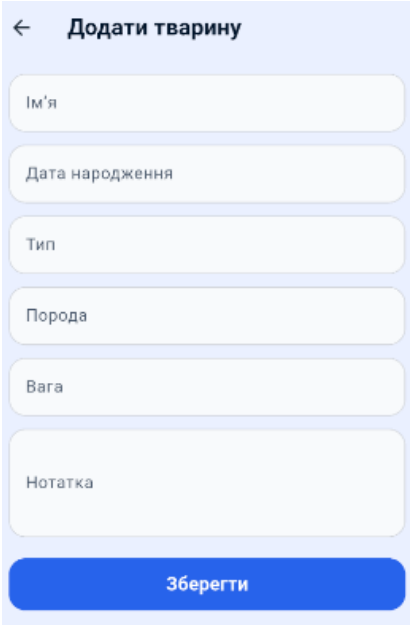


Рисунок 3.4.4 – Форма додавання тварини

Екран профілю тварини містить розширену інформацію, включаючи основні характеристики та короткий перелік медичних записів. З цього екрану користувач може перейти до редагування інформації або видалення профілю. Також передбачено перехід до повного списку медичних записів, що дозволяє детально ознайомитися з історією подій.

Розділ медичних записів призначений для ведення історії догляду та стану здоров’я тварини.

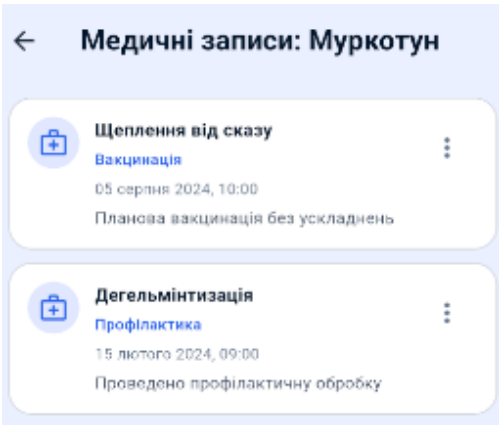


Рисунок 3.4.5 – Вміст сторінки «Медичні записи»

Користувач може створювати нові записи, редагувати або переглядати існуючі. Для додавання запису необхідно заповнити форму, вказавши назву, опис, дату та категорію події. Після збереження запис автоматично додається до списку та відображається у профілі тварини.

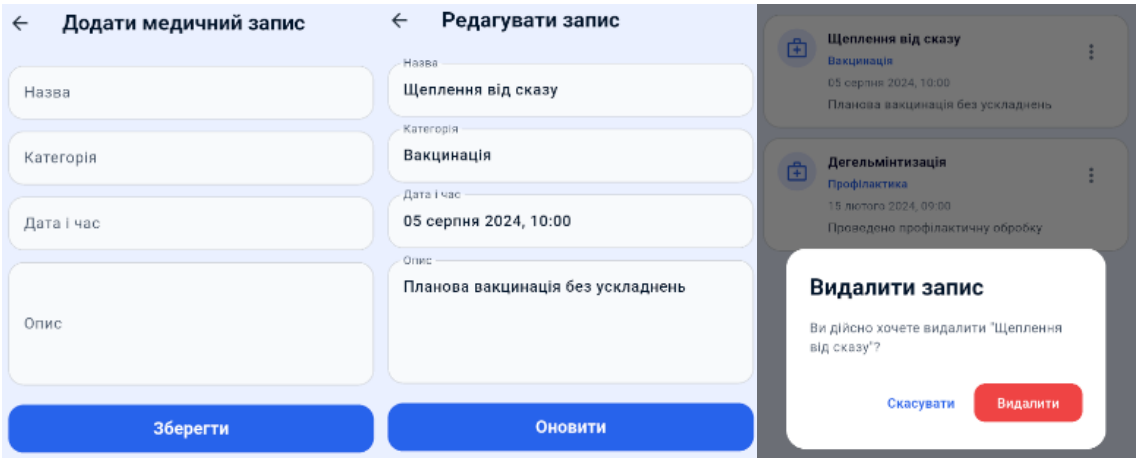


Рисунок 3.4.6 – Функціонал ведення медичних записів

Окрему функціональність становить система нагадувань. Вона дозволяє користувачу планувати події, пов’язані з доглядом за тваринами, наприклад годування, вакцинацію або прийом ліків.

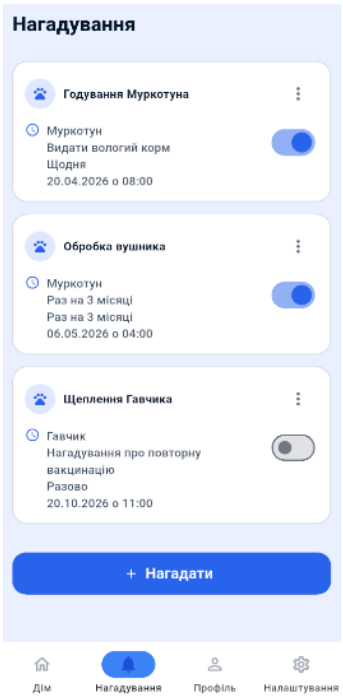


Рисунок 3.4.7 – Сторінка «Нагадування»

Для створення нагадування необхідно обрати тварину, задати назву події, встановити дату та час, а також визначити тип повторення. Після збереження нагадування система автоматично планує сповіщення, яке буде відображене у відповідний момент часу.

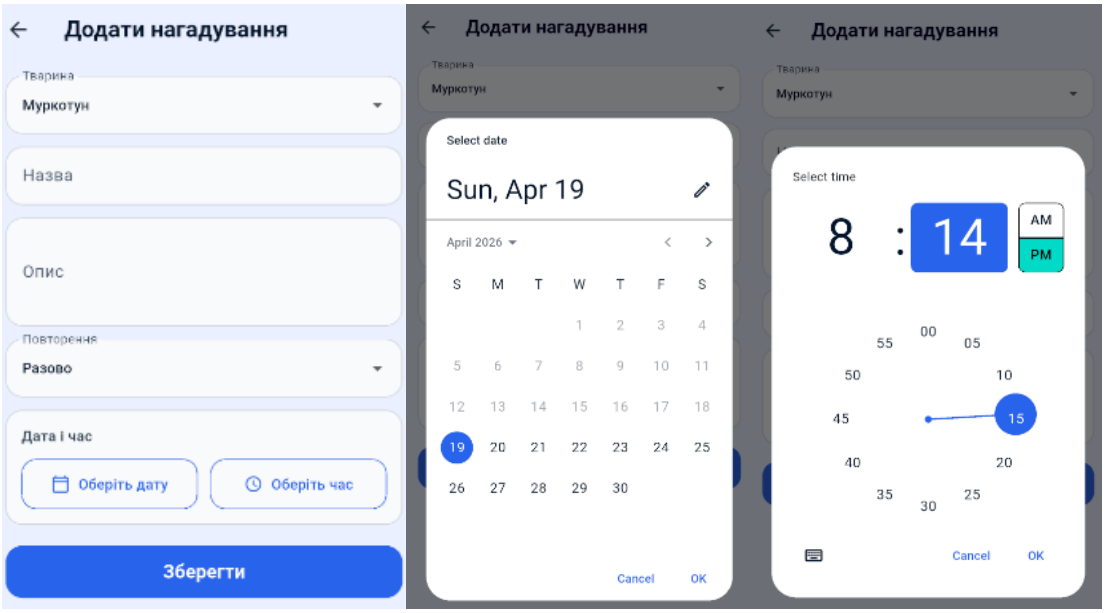


Рисунок 3.4.8 – Функціонал ведення нагадувань

Навігація між основними розділами застосунку здійснюється за допомогою нижньої панелі, яка забезпечує швидкий доступ до головного екрану, розділу нагадувань, профілю користувача та налаштувань. Такий підхід дозволяє користувачу швидко перемикатися між функціями без необхідності складної навігації.

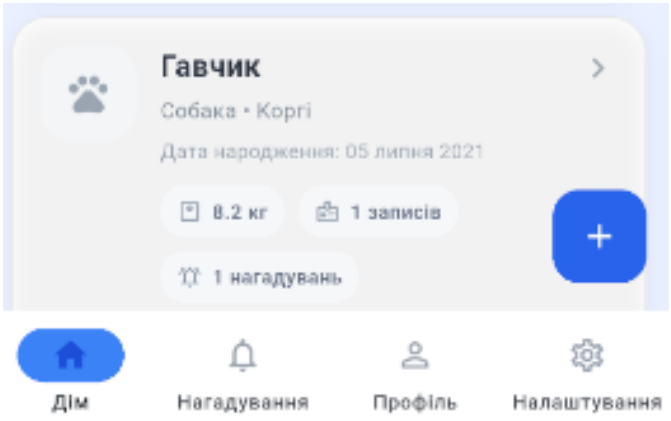


Рисунок 3.4.9 – Нижня панель

Також передбачено можливість виходу з облікового запису.

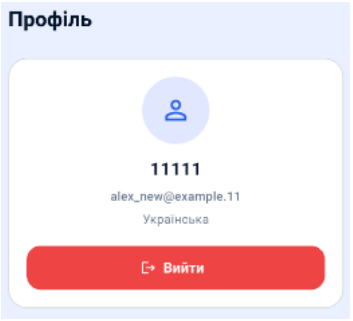


Рисунок 3.4.10 – Профіль користувача

У розділі налаштувань також є можливість для користувача змінювати параметри застосунку, зокрема мову інтерфейсу на даний момент, та інші персональні параметри, які будуть додані в майбутньому для надання користувачу можливостей налаштування інтерфейсу під себе (розмір тексту, тощо).

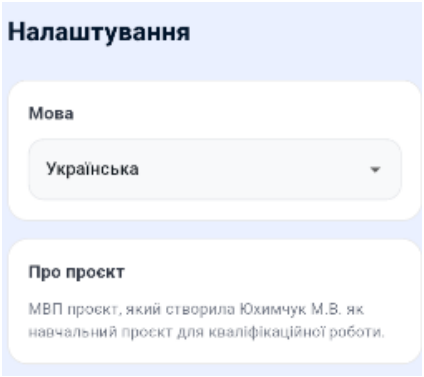


Рисунок 3.4.11 – Налаштування застосунку

Під час роботи застосунку всі дії користувача супроводжуються відповідною реакцією системи. У разі помилок введення або відсутності необхідних даних відображаються повідомлення, що дозволяє уникнути некоректних операцій. Це забезпечує стабільність роботи та підвищує зручність використання.

Таким чином, застосунок забезпечує інтуїтивно зрозумілий інтерфейс та повний набір функцій для організації догляду за домашніми тваринами. Використання простих сценаріїв взаємодії та логічної структури дозволяє користувачу ефективно працювати із системою без додаткових інструкцій.

3.5 Вимоги до технічних та програмних засобів

У ході розроблення мобільного застосунку доцільно враховувати обмеження апаратного та програмного середовища, які можуть впливати на стабільність, швидкодію та коректність виконання функцій. Розроблений програмний продукт призначений для використання на мобільних пристроях і не потребує значних обчислювальних ресурсів, оскільки основні операції пов’язані з обробкою користувацьких даних, їх збереженням та відображенням, а також формуванням і виконанням нагадувань.

Застосунок орієнтований на роботу в середовищах мобільних операційних систем, де важливу роль відіграє оптимізація використання ресурсів пристрою, зокрема оперативної пам’яті та енергоспоживання. Окрім цього, для забезпечення повноцінного функціонування передбачено використання мережевого з’єднання, яке необхідне для синхронізації даних та їх збереження у віддаленому середовищі. При цьому частина функціональності зберігає працездатність навіть за відсутності активного підключення до мережі.

Розроблення мобільного застосунку виконувалося з використанням Flutter SDK та мови програмування Dart. Як середовище розробки використовувалися Visual Studio Code, Android SDK та засоби емуляції мобільних пристроїв. У межах проєкту передбачено використання бібліотек firebase_core і cloud_firestore для інтеграції з Firebase Firestore, provider для керування станом застосунку, sqflite для локального кешування даних, shared_preferences для збереження локальних налаштувань, а також flutter_local_notifications для реалізації системи сповіщень.

Кожна з наведених бібліотек виконує окрему роль: firebase_core забезпечує ініціалізацію Firebase, cloud_firestore – роботу з хмарною базою даних, provider – керування станом, sqflite – локальне кешування, shared_preferences – збереження налаштувань, flutter_local_notifications – формування сповіщень.

З огляду на призначення застосунку, вимоги до пристрою залишаються помірними, що дозволяє використовувати його на широкому спектрі мобільних

платформ без необхідності спеціалізованого обладнання. Нижче наведено мінімальні та рекомендовані характеристики пристроїв, необхідні для стабільної роботи застосунку.

Таблиця 3.1 – Технічні вимоги

Характеристика	Мінімальні	Рекомендовані
Операційна система	Android 8.0 / iOS 12	Android 10+ / iOS 14+
Процесор	Багатоядерний,>1.8 ГГц	Багатоядерний,>2.2 ГГц
ОЗП (RAM)	2 GB	4 GB і більше
Вільне місце на пристрої	від 100 MB	від 200 MB
Підключення до мережі	3G / Wi-Fi	4G / Wi-Fi

Інтерфейс застосунку реалізовано з урахуванням підтримки різних розмірів екранів мобільних пристроїв. Для цього використовуються адаптивні компоненти Flutter, прокручувані області вмісту та гнучкі елементи розмітки, що дозволяє забезпечити коректне відображення інтерфейсу як на компактних смартфонах, так і на пристроях із більшою діагоналлю екрана.

3.6 Тестування програмного забезпечення

Для перевірки працездатності застосунку було обрано поєднання ручного функціонального тестування та юніт-тестування внутрішньої логіки. Ручне тестування дозволяє оцінити коректність роботи інтерфейсу, навігації та взаємодії користувача із системою в умовах, наближених до реального використання. Юніт-тестування використовується для перевірки окремих компонентів програмної логіки незалежно від графічного інтерфейсу. Такий підхід є доцільним для MVP-версії застосунку, оскільки дозволяє перевірити як зовнішню поведінку системи, так і правильність роботи її внутрішніх механізмів. Поєднання цих методів є

оптимальним для MVP, оскільки дозволяє охопити ключові аспекти якості без надмірних витрат на інфраструктуру автоматизованого тестування.

Перевірка працездатності мобільного застосунку виконувалася шляхом послідовного відтворення основних сценаріїв використання в умовах, наближених до реальних. Застосунок було розгорнуто в середовищі розроблення та протестовано за допомогою емулятора, а також на фізичному мобільному пристрої. У процесі перевірки здійснювалося проходження ключових дій користувача, зокрема реєстрації та авторизації, створення і редагування профілів тварин, додавання медичних записів та формування нагадувань. Окрема увага приділялася коректності переходів між екранами, роботі форм введення та відображенню актуальних даних після виконання операцій.

Критерієм успішності ручного тестування вважалася відповідність поведінки застосунку очікуваному результату для кожного сценарію – коректне відображення даних, відсутність помилок інтерфейсу та збереження інформації після завершення операції. У разі виявлення відхилень від очікуваної поведінки відповідні компоненти підлягали перевірці та корекції до досягнення стабільного результату. Для наочного підтвердження працездатності застосунку наведено приклади його роботи під час виконання основних сценаріїв.

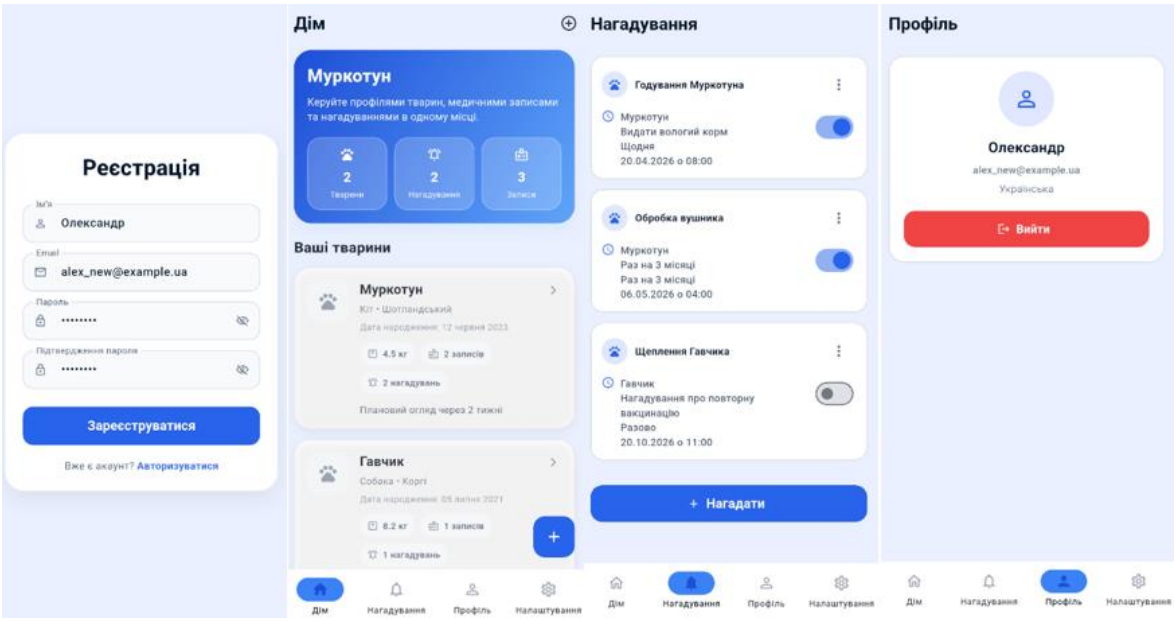


Рисунок 3.6.1 – Робота мобільного застосунку

За результатами тестування встановлено, що основні функції застосунку працюють стабільно, а логіка взаємодії з користувачем реалізована коректно.

Було перевірено ключові сценарії роботи системи, зокрема авторизацію користувача, створення профілю тварини, додавання та перегляд медичних записів, а також формування нагадувань.

У всіх наведених випадках отримано очікуваний результат, що підтверджує коректність реалізації основного функціоналу.

Додатково для перевірки внутрішньої логіки застосунку застосовано автоматизоване тестування окремих функціональних компонентів. У межах тестування перевірялися базові сценарії обробки даних, зокрема валідація введених значень, формування об’єктів та логіка роботи нагадувань. Запуск тестів виконувався у середовищі розроблення із використанням стандартних засобів Flutter, зокрема пакету flutter_test, що дозволило отримати підтвердження коректності реалізованої логіки.

У межах автоматизованого тестування було перевірено одинадцять тест-кейсів, що охоплюють ключові моделі та сервіси застосунку. Зокрема, перевірялася коректність роботи методу copyWith для моделей Pet, MedicalRecord, UserProfile та Reminder – тобто правильність часткового оновлення об'єктів із збереженням незмінних полів.

Окремо тестувалися допоміжні функції класу ReminderRepeatTypeHelper, а саме перетворення типу повторення у код і назву. Найбільш критичною частиною тестування стала перевірка логіки сервісу NotificationService, зокрема коректність визначення наступної дати спрацювання для нагадувань у різних часових ситуаціях – коли запланована дата є майбутньою, вже минулою або потребує зсуву на наступний цикл.

Результати виконання автоматизованих тестів наведено на рисунку 3.6.2. Усі виконані перевірки завершилися успішно, що підтверджує коректність реалізації основних логічних компонентів застосунку.

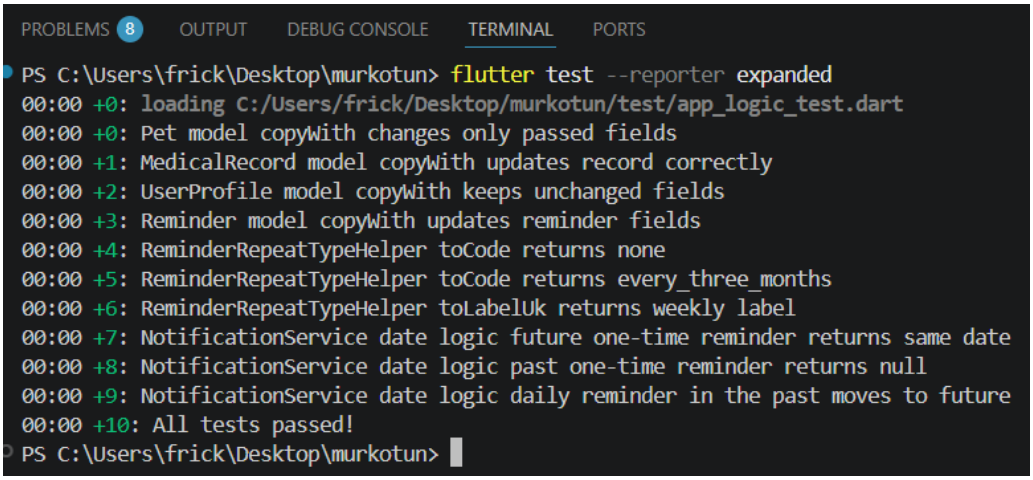


Рисунок 3.6.2 – Результати тестування

Результати виконання тестів свідчать про відсутність критичних помилок у ключових частинах застосунку та узгодженість роботи його основних компонентів. Таким чином, проведене тестування підтверджує відповідність реалізованого програмного продукту основним вимогам технічного завдання.

3.7 Висновки програмної реалізації та тестування продукту

У межах розділу реалізовано основні функціональні компоненти мобільного застосунку відповідно до визначених вимог, зокрема роботу з профілями користувачів і тварин, ведення медичних записів, формування нагадувань та базові механізми доступу до системи.

Розроблено користувацький інтерфейс, орієнтований на простоту та зручність використання, а також реалізовано збереження даних із використанням Firebase Firestore з можливістю локального кешування.

Проведене тестування підтвердило коректність роботи основних функцій. Ручне функціональне тестування охопило всі ключові сценарії взаємодії користувача із системою, а автоматизоване юніт-тестування підтвердило правильність роботи внутрішньої логіки моделей та сервісів – усі одинадцять тест-кейсів завершилися успішно.

Таким чином, створений застосунок є працездатною MVP-версією, що відповідає поставленим вимогам і може бути основою для подальшого розвитку.

ВИСНОВКИ

Розроблений мобільний застосунок орієнтований на впорядкування процесів догляду за домашніми тваринами шляхом об’єднання функцій обліку, збереження та нагадування в межах єдиного інформаційного середовища. Досягнення поставленої мети підтверджується створенням програмного продукту, який забезпечує реалізацію основних сценаріїв взаємодії користувача з даними та дозволяє систематизувати інформацію, що раніше зберігалася фрагментарно або неструктуровано.

Проведений аналіз предметної області дозволив виявити ключові проблеми, пов’язані з відсутністю зручних інструментів для централізованого ведення інформації про тварин, а також із необхідністю постійного контролю за подіями, пов’язаними з їх доглядом. Вивчення існуючих програмних рішень дало змогу визначити їх сильні та слабкі сторони, що стало основою для формування власного підходу до розробки із акцентом на достатність функціоналу, зрозумілість інтерфейсу та відсутність надлишкових елементів.

Вибір технологій обґрунтовано вимогами до мобільного застосунку, зокрема необхідністю швидкої реалізації, підтримки роботи з даними та забезпечення стабільності. Використання хмарної бази даних дозволило організувати централізоване зберігання інформації, тоді як застосування локального сховища сприяло підвищенню продуктивності та зменшенню залежності від якості мережевого з’єднання. Така комбінація забезпечує баланс між функціональністю та простотою реалізації.

Архітектура застосунку сформована з урахуванням принципів розподілу відповідальності між окремими рівнями, що дозволяє відокремити інтерфейсну частину від логіки обробки та роботи з даними. Це сприяє узгодженості компонентів системи, спрощує її підтримку та створює передумови для подальшого розвитку без суттєвого ускладнення структури.

Реалізована функціональність охоплює основні сценарії роботи: авторизацію користувача, управління профілями тварин, ведення медичних записів та створення нагадувань. Логіка застосунку побудована на послідовній обробці дій користувача з подальшим збереженням результатів у базі даних. Важливе місце займає система нагадувань, яка забезпечує своєчасне інформування про заплановані події та формує практичну цінність застосунку.

Отриманий програмний продукт характеризується достатнім рівнем функціональності для виконання поставлених задач у межах концепції мінімально життєздатного продукту. Його робота є стабільною при виконанні основних операцій, а застосування асинхронної обробки дозволяє уникати блокування інтерфейсу під час взаємодії з базою даних. Продуктивність застосунку відповідає вимогам мобільних пристроїв і не створює додаткового навантаження на ресурси.

Практична цінність розробки полягає у можливості її використання для організації догляду за домашніми тваринами в повсякденних умовах. Застосунок дозволяє зберігати важливу інформацію, планувати дії та отримувати нагадування, що знижує ймовірність пропуску важливих подій і підвищує зручність взаємодії з даними.

Отриманий результат відповідає поставленим завданням, оскільки реалізовано всі ключові функції, визначені у технічному завданні та на етапі проєктування. Обрана концепція MVP дозволила зосередитися на основному функціоналі, уникнувши надмірного ускладнення системи, що є доцільним для даного етапу розробки.

Подальший розвиток проєкту може бути спрямований на розширення функціональності та підвищення зручності використання. До перспективних напрямів належать удосконалення системи нагадувань, розширення можливостей роботи з даними, а також інтеграція додаткових сервісів. Окрему увагу може бути приділено підвищенню рівня безпеки та покращенню користувацького досвіду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Pet Care Market Size, Share & Trends Analysis Report. Grand View Research. URL: https://www.grandviewresearch.com/industry-analysis/pet-care-market (дата звернення: 12.03.2026).

2. Pet Care Apps Market Size & Trends Analysis Report. Business Research Insights. URL: https://www.businessresearchinsights.com/market-reports/pet-care-apps-market-123827 (дата звернення: 12.03.2026).

3. Number of smartphone users worldwide 2014–2029. Statista. URL: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide (дата звернення: 15.03.2026).

4. Verhoef P. C. et al. Digital transformation: A multidisciplinary reflection and research agenda. *Journal of Business Research*. 2021. Vol. 122. P. 889–901. URL: https://doi.org/10.1016/j.jbusres.2019.09.022 (дата звернення: 16.03.2026).

5. 11Pets: Pet Care – офіційний сайт розробника. URL: https://www.11pets.com (дата звернення: 18.03.2026).

6. PetDesk – Pet Health App. PetDesk Inc. URL: https://www.petdesk.com (дата звернення: 18.03.2026).

7. Dog and cat care – PetNote+. Google Play. URL: https://play.google.com/store/apps/details?id=com.lancerdog.petnote_plus (дата звернення: 19.03.2026).

8. Wasserman A. I. Software engineering issues for mobile application development. *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*. 2010. P. 397–400. URL: https://doi.org/10.1145/1882362.1882443 (дата звернення: 19.03.2026).

9. Nuseibeh B., Easterbrook S. Requirements engineering: a roadmap. *Proceedings of the Conference on the Future of Software Engineering (ICSE 2000)*. 2000. P. 35–46. URL: https://dl.acm.org/doi/10.1145/336512.336523 (дата звернення: 20.03.2026).

10. Wiegers K. Software Requirements: 10 Requirements Traps to Avoid. *Process Impact*. URL: https://medium.com/analysts-corner/10-requirements-traps-to-avoid-fb103bfeaaacaaac (дата звернення: 20.03.2026).

11. OMG Unified Modeling Language Specification. Version 2.5.1. Object Management Group, 2017. URL: https://www.omg.org/spec/UML/2.5.1 (дата звернення: 21.03.2026).

12. Siau K., Lee L. Are use case and class diagrams complementary in requirements analysis? An experimental study on use case and class diagrams in UML. *Requirements Engineering*. 2004. Vol. 9. P. 229–237. URL: https://www.researchgate.net/publication/220428080 (дата звернення: 21.03.2026).

13. Functional vs. Non-Functional Requirements. Visure Solutions. URL: https://visuresolutions.com/alm-guide/functional-vs-non-functional-requirements/ (дата звернення: 02.05.2026).

14. Microservices vs. Monolithic Architecture: What Is the Difference? Atlassian. URL: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith (дата звернення: 25.03.2026).

15. Microservices vs Monolithic Architecture: Which Is Best for Scaling Mobile Applications? Etima Blog. URL: https://www.etimablog.com/2026/03/microservices-vs-monolithic.html (дата звернення: 25.03.2026).

16. Chen Y. Research on the Application of Layered Architecture in Computer Software Development. *ResearchGate*. 2023. URL: https://www.researchgate.net/publication/376605859_Research_on_the_Application_of_Layered_Architecture_in_Computer_Software_Development (дата звернення: 25.03.2026).

17. IDEF0 – Function Modeling Method. IDEF Official Site. URL: https://www.idef.com/idefo-function_modeling_method (дата звернення: 26.03.2026).

18. Šerifi V. et al. Functional and information modeling of production using IDEF methods. *Strojniški vestnik – Journal of Mechanical Engineering*. 2009. Vol. 55, No. 2. URL:

https://www.researchgate.net/publication/235636672_Functional_and_information_modeling_of_production_using_IDEF_methods (дата звернення: 26.03.2026).

19. What Is a Data Flow Diagram (DFD)? IBM. URL: https://www.ibm.com/think/topics/data-flow-diagram (дата звернення: 26.03.2026).

20. What is a Data Flow Diagram? Lucidchart. URL: https://www.lucidchart.com/pages/data-flow-diagram (дата звернення: 26.03.2026).

21. Unified Modeling Language (UML) Diagrams. GeeksforGeeks. URL: https://www.geeksforgeeks.org/system-design/unified-modeling-language-uml-introduction (дата звернення: 27.03.2026).

22. UML Sequence Diagram Tutorial. Lucidchart. URL: https://www.lucidchart.com/pages/uml-sequence-diagram (дата звернення: 27.03.2026).

23. UML Diagram Types. Creately Blog. URL: https://creately.com/blog/diagrams/uml-diagram-types-examples (дата звернення: 27.03.2026).

24. Durgekar S. R. et al. A Review Paper on Design and Experience of Mobile Applications. *ResearchGate*. 2024. URL: https://www.researchgate.net/publication/377798147_A_Review_Paper_on_Design_and_Experience_of_Mobile_Applications (дата звернення: 28.03.2026).

25. Features of New Design Principles for Mobile Applications UI/UX for Smartphones. *ResearchGate*. 2021. URL: https://www.researchgate.net/publication/357406873_Features_of_New_Design_Principles_for_Mobile_Applications_UIUX_for_Smartphones (дата звернення: 28.03.2026).

26. The Allure and Impact of Minimalist UX Design. Toptal. URL: https://www.toptal.com/designers/ux/minimalist-ux-design-strategies (дата звернення: 28.03.2026).

27. Why Flutter is the most popular cross-platform mobile SDK. Stack Overflow Blog. 2022. URL: https://stackoverflow.blog/2022/02/21/why-flutter-is-the-most-popular-cross-platform-mobile-sdk (дата звернення: 29.03.2026).

28. Flutter Reigns Supreme: The Most Popular Cross-Platform Mobile Framework in 2023. HyperSense Software. URL: https://hypersense-software.com/blog/2023/05/22/flutter-most-popular-cross-platform-framework (дата звернення: 29.03.2026).

29. A Review on Firebase (Backend as a Service) for Mobile Application Development. IJRASET. URL: https://www.ijraset.com/research-paper/firebase-backend-as-a-service-for-mobile-application-development (дата звернення: 30.03.2026).

30. A Review on Firebase (Backend as a Service) for Mobile Application Development. *ResearchGate*. 2022. URL: https://www.researchgate.net/publication/358244355_A_Review_on_Firebase_Backend_as_A_Service_for_Mobile_Application_Development (дата звернення: 30.03.2026). I

31. Serverless in action: building a simple backend with Cloud Firestore and Cloud Functions. Google Cloud Blog. URL: https://cloud.google.com/blog/products/application-development/serverless-in-action-building-a-simple-backend-with-cloud-firestore-and-cloud-functions (дата звернення: 30.03.2026).

32. Understanding Flutter Widgets: A Beginner's Guide to Building User Interfaces. Medium. URL: https://medium.com/@kavyamistry0612/understanding-flutter-widgets-a-beginners-guide-to-building-user-interfaces-eed1acf17070 (дата звернення: 01.04.2026).

33. Using Firebase in Mobile App Development: A Practical Guide. CodingCops. URL: https://codingcops.com/firebase-in-mobile-app-development-a-practical-guide (дата звернення: 01.04.2026).

34. Implementing local notifications in Flutter. LogRocket Blog. URL: https://blog.logrocket.com/implementing-local-notifications-in-flutter (дата звернення: 02.04.2026).

35. How to implement Flutter local notifications to boost user engagement. Codemagic Blog. URL: https://blog.codemagic.io/flutter-local-notifications (дата звернення: 02.04.2026).

36. Building user interfaces with Flutter. Flutter Official Docs. URL: https://docs.flutter.dev/ui (дата звернення: 03.04.2026).

37. Build a Flutter layout. Flutter Official Docs. URL: https://docs.flutter.dev/ui/layout/tutorial (дата звернення: 03.04.2026).

38. Guidelines for Creating App Interfaces with Flutter UI Design. DhiWise. URL: https://www.dhiwise.com/post/guidelines-for-creating-app-interfaces-using-flutter-ui-design (дата звернення: 03.04.2026).

39. Persist data with SQLite. Flutter Official Docs. URL: https://docs.flutter.dev/cookbook/persistence/sqlite (дата звернення: 04.04.2026).

40. Mastering SQLite in Flutter: A Comprehensive Guide to Sqflite. Medium / YavarTechWorks. URL: https://medium.com/yavar/mastering-sqlite-in-flutter-a-comprehensive-guide-to-sqflite-c59d5ef56b62 (дата звернення: 04.04.2026).

Додаток А
(Презентація)

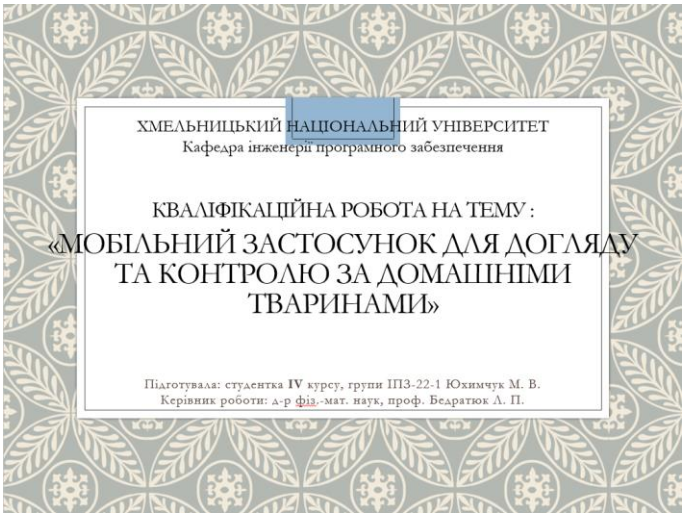


Рисунок А.1

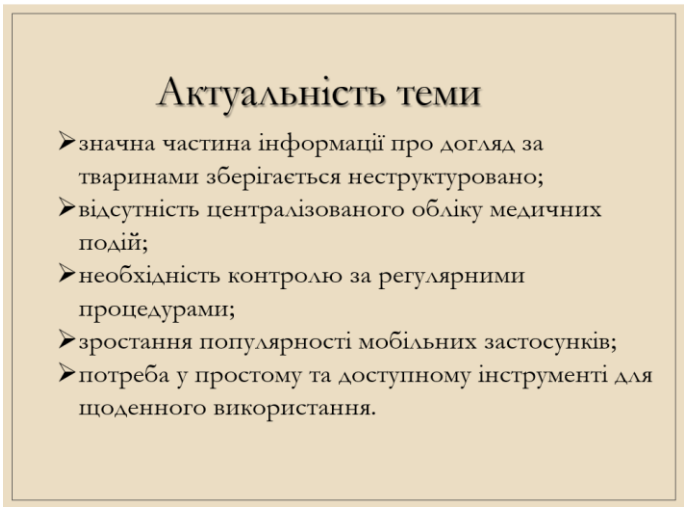


Рисунок А.2

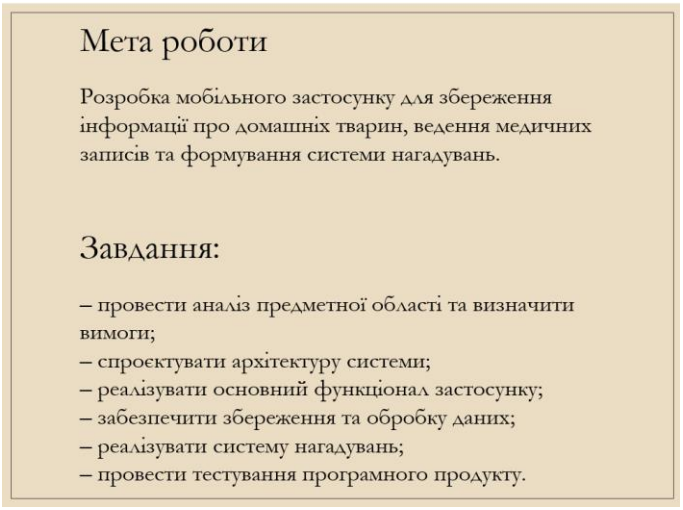
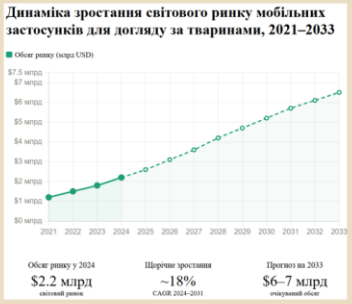


Рисунок А.3

Змістовний аналіз предметної області, її структурних та функціональних особливостей

Сфера догляду за домашніми тваринами активно розвивається разом із поширенням мобільних технологій. Проте значна частина власників зберігає інформацію про тварин неструктуровано або не фіксує її взагалі, що ускладнює контроль за станом здоров'я вихованця.



Лише ~4–7% власників тварин використовують спеціалізовані мобільні застосунки для обліку догляду, попри зростання ринку

Типові проблеми власників:

- Інформація в нотатках або на папері
- Пропуск щеплень і планових процедур
- Відсутність зручного нагадування

Рисунок А.4

Аналіз наявного програмно-технічного забезпечення

Система	Переваги	Недоліки
11Pets	широкий функціонал	перевантажений інтерфейс
PetDesk	медичні нагадування	обмежений безкоштовний доступ
PetNote+	простота використання	обмежені можливості

Рисунок А.5

Визначення функціональних та нефункціональних вимог до ПЗ

Функціональні:
– реєстрація та авторизація;
– управління профілями тварин;
– ведення медичних записів;
– створення нагадувань.

Нефункціональні:
– стабільна робота;
– простий інтерфейс;
– підтримка Android;
– можливість масштабування.

Рисунок А.6

Рисунок А.7

Рисунок А.8

Рисунок А.9

Аналіз та вибір технологій

Технологія	Призначення	Причина вибору
Flutter, Dart	Розробка мобільного застосунку	Забезпечує кросплатформеність та швидку розробку
Firebase	Хмарне зберігання даних	Дозволяє організувати централізоване зберігання та синхронізацію даних
SQLite	Локальне зберігання (кешування)	Забезпечує доступ до даних без підключення до мережі та підвищує продуктивність
Provider	Керування станом застосунку	Спрощує управління станом та взаємодію між компонентами
flutter_local_notifications	Реалізація системи нагадувань	Дозволяє реалізувати локальні сповіщення без використання зовнішніх сервісів

Рисунок А.10

Реалізація модулів і база даних

Реалізація модулів і база даних
- ✓ створено інтерфейс користувача;
- ✓ реалізовано CRUD для даних;
- ✓ інтегровано систему нагадувань;
- ✓ забезпечено збереження інформації.

Рисунок А.11

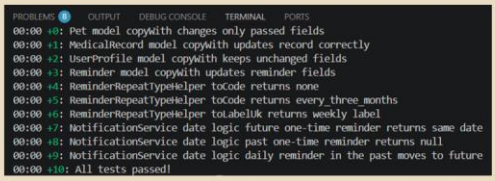
Вимоги до технічного та програмного забезпечення

Характеристика	Мінімальні	Рекомендовані
Операційна система	Android 8.0 / iOS 12	Android 10+ / iOS 14+
Процесор	Багатоядерний,>1.8 ГГц	Багатоядерний,>2.2 ГГц
ОЗП (RAM)	2 GB	4 GB і більше
Вільне місце на пристрої	від 100 MB	від 200 MB
Підключення до мережі	3G / Wi-Fi	4G / Wi-Fi

Рисунок А.12

Тестування ПЗ

У межах тестування перевірялися базові сценарії обробки даних, зокрема валідація введених значень, формування об'єктів та логіка роботи нагадувань.



За результатами виконання тестів встановлено, що всі перевірені сценарії завершилися успішно, що підтверджується відсутністю помилок під час виконання.

Рисунок А.13

Висновки:

- ✓ розроблено мобільний застосунок для догляду за тваринами;
- ✓ реалізовано основні функції: профілі, записи, нагадування;
- ✓ система відповідає вимогам і працює стабільно;
- ✓ проєкт має потенціал подальшого розвитку.

Рисунок А.14

ДЯКУЮ ЗА УВАГУ

Рисунок А.15

Додаток Б
(Програмний код)

AddMedicalRecordScreen:

```
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import '../../../models/medical_record.dart';
import '../../../models/pet.dart';

class AddMedicalRecordScreen extends StatefulWidget {
  final Pet pet;

  const AddMedicalRecordScreen({
    super.key,
    required this.pet,
  });

  @override
  State<AddMedicalRecordScreen> createState() =>
_AddMedicalRecordScreenState();
}

class _AddMedicalRecordScreenState extends State<AddMedicalRecordScreen> {
  final _formKey = GlobalKey<FormState>();

  final titleController = TextEditingController();
  final descriptionController = TextEditingController();
  final dateTimeController = TextEditingController();
  final categoryController = TextEditingController();

  @override
  void dispose() {
    titleController.dispose();
    descriptionController.dispose();
    dateTimeController.dispose();
    categoryController.dispose();
    super.dispose();
  }

  Future<void> _saveRecord() async {
    final isValid = _formKey.currentState?.validate() ?? false;
    if (!isValid) return;

    final record = MedicalRecord(
      id: DateTime.now().millisecondsSinceEpoch.toString(),
      petId: widget.pet.id,
      title: titleController.text.trim(),
      description: descriptionController.text.trim(),
      dateTime: dateTimeController.text.trim(),
      category: categoryController.text.trim(),
    );

    await FirebaseFirestore.instance
        .collection('medical_records')
```

```
      .doc(record.id)
      .set({
    'id': record.id,
    'petId': record.petId,
    'title': record.title,
    'description': record.description,
    'dateTime': record.dateTime,
    'category': record.category,
  });

  Navigator.of(context).pop(true);
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: const Color(0xFFEAF0FF),
    appBar: AppBar(
      backgroundColor: const Color(0xFFEAF0FF),
      surfaceTintColor: Colors.transparent,
      elevation: 0,
      title: const Text(
        'Додати медичний запис',
        style: TextStyle(
          color: Color(0xFF0F172A),
          fontWeight: FontWeight.w700,
        ),
      ),
      iconTheme: const IconThemeData(color: Color(0xFF1F2937)),
    ),
    body: Form(
      key: _formKey,
      autovalidateMode: AutovalidateMode.onUserInteraction,
      child: ListView(
        padding: const EdgeInsets.all(16),
        children: [
          _buildField(
            'Назва',
            titleController,
            validator: (value) {
              final text = value?.trim() ?? '';
              if (text.isEmpty) return 'Введіть назву запису';
              if (text.length < 3) return 'Мінімум 3 символи';
              return null;
            },
          ),
          const SizedBox(height: 12),
          _buildField(
            'Категорія',
            categoryController,
            validator: (value) {
              final text = value?.trim() ?? '';
              if (text.isEmpty) return 'Введіть категорію';
              return null;
            },
          ),
```

```
          const SizedBox(height: 12),
          _buildField(
            'Дата і час',
            dateTimeController,
            validator: (value) {
              final text = value?.trim() ?? '';
              if (text.isEmpty) return 'Введіть дату і час';
              if (text.length < 5) return 'Некоректний формат';
              return null;
            },
          ),
          const SizedBox(height: 12),
          _buildField(
            'Опис',
            descriptionController,
            maxLines: 4,
            validator: (value) {
              final text = value?.trim() ?? '';
              if (text.isEmpty) return 'Введіть опис';
              if (text.length < 5) return 'Мінімум 5 символів';
              if (text.length > 500) return 'Максимум 500 символів';
              return null;
            },
          ),
          const SizedBox(height: 20),
          SizedBox(
            height: 50,
            child: ElevatedButton(
              onPressed: _saveRecord,
              style: ElevatedButton.styleFrom(
                backgroundColor: const Color(0xFF2563EB),
                foregroundColor: Colors.white,
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(16),
                ),
                elevation: 0,
              ),
              child: const Text(
                'Зберегти',
                style: TextStyle(
                  fontSize: 18,
                  fontWeight: FontWeight.w700,
                ),
              ),
            ),
          ),
        ],
      ),
    ),
  );
}

Widget _buildField(
  String label,
  TextEditingController controller, {
  int maxLines = 1,
```

```
    String? Function(String?)? validator,
  }) {
    return TextFormField(
      controller: controller,
      maxLines: maxLines,
      validator: validator,
      style: const TextStyle(
        color: Color(0xFF0F172A),
        fontWeight: FontWeight.w500,
      ),
      decoration: InputDecoration(
        labelText: label,
        labelStyle: const TextStyle(color: Color(0xFF475569)),
        filled: true,
        fillColor: const Color(0xFFF8FAFC),
        errorMaxLines: 2,
        border: OutlineInputBorder(
          borderRadius: BorderRadius.circular(16),
          borderSide: const BorderSide(color: Color(0xFFD1D5DB)),
        ),
        enabledBorder: OutlineInputBorder(
          borderRadius: BorderRadius.circular(16),
          borderSide: const BorderSide(color: Color(0xFFD1D5DB)),
        ),
        focusedBorder: OutlineInputBorder(
          borderRadius: BorderRadius.circular(16),
          borderSide: const BorderSide(
            color: Color(0xFF2563EB),
            width: 1.6,
          ),
        ),
        errorBorder: OutlineInputBorder(
          borderRadius: BorderRadius.circular(16),
          borderSide: const BorderSide(color: Color(0xFFEF4444)),
        ),
        focusedErrorBorder: OutlineInputBorder(
          borderRadius: BorderRadius.circular(16),
          borderSide: const BorderSide(
            color: Color(0xFFEF4444),
            width: 1.6,
          ),
        ),
      ),
    );
  }
}
```

AddPetScreen

```
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import '../../models/pet.dart';

class AddPetScreen extends StatefulWidget {
  const AddPetScreen({super.key});

  @override
```

```
  State<AddPetScreen> createState() => _AddPetScreenState();
}

class _AddPetScreenState extends State<AddPetScreen> {
  final _formKey = GlobalKey<FormState>();

  final nameController = TextEditingController();
  final birthDateController = TextEditingController();
  final typeController = TextEditingController();
  final breedController = TextEditingController();
  final weightController = TextEditingController();
  final noteController = TextEditingController();

  @override
  void dispose() {
    nameController.dispose();
    birthDateController.dispose();
    typeController.dispose();
    breedController.dispose();
    weightController.dispose();
    noteController.dispose();
    super.dispose();
  }

  double? _parseWeight(String value) {
    final normalized = value.trim().replaceAll(',', '.');
    return double.tryParse(normalized);
  }

  Future<void> _savePet() async {
    final isValid = _formKey.currentState?.validate() ?? false;
    if (!isValid) return;

    final newPet = Pet(
      id: DateTime.now().millisecondsSinceEpoch.toString(),
      name: nameController.text.trim(),
      birthDate: birthDateController.text.trim(),
      type: typeController.text.trim(),
      breed: breedController.text.trim(),
      weight: _parseWeight(weightController.text)!,
      note: noteController.text.trim(),
    );

    await FirebaseFirestore.instance
        .collection('pets')
        .doc(newPet.id)
        .set({
      'id': newPet.id,
      'name': newPet.name,
      'birthDate': newPet.birthDate,
      'type': newPet.type,
      'breed': newPet.breed,
      'weight': newPet.weight,
      'note': newPet.note,
    });
```

```
  Navigator.of(context).pop(true);
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: const Color(0xFFEAF0FF),
    appBar: AppBar(
      backgroundColor: const Color(0xFFEAF0FF),
      surfaceTintColor: Colors.transparent,
      elevation: 0,
      title: const Text(
        'Додати тварину',
        style: TextStyle(
          color: Color(0xFF0F172A),
          fontWeight: FontWeight.w700,
        ),
      ),
      iconTheme: const IconThemeData(color: Color(0xFF1F2937)),
    ),
    body: Form(
      key: _formKey,
      autovalidateMode: AutovalidateMode.onUserInteraction,
      child: ListView(
        padding: const EdgeInsets.all(16),
        children: [
          _buildField(
            'Ім’я',
            nameController,
            validator: (value) {
              final text = value?.trim() ?? '';
              if (text.isEmpty) return 'Введіть ім’я';
              if (text.length < 2) return 'Мінімум 2 символи';
              return null;
            },
          ),
          const SizedBox(height: 12),
          _buildField(
            'Дата народження',
            birthDateController,
            validator: (value) {
              final text = value?.trim() ?? '';
              if (text.isEmpty) return 'Введіть дату народження';
              if (text.length < 4) return 'Некоректний формат';
              return null;
            },
          ),
          const SizedBox(height: 12),
          _buildField(
            'Тип',
            typeController,
            validator: (value) {
              final text = value?.trim() ?? '';
              if (text.isEmpty) return 'Введіть тип тварини';
              return null;
            },
```

```
            ),
            const SizedBox(height: 12),
            _buildField(
              'Порода',
              breedController,
              validator: (value) {
                final text = value?.trim() ?? '';
                if (text.isEmpty) return 'Введіть породу';
                return null;
              },
            ),
            const SizedBox(height: 12),
            _buildField(
              'Вага',
              weightController,
              keyboardType: const TextInputType.numberWithOptions(decimal:
true),
              validator: (value) {
                final text = value?.trim() ?? '';
                if (text.isEmpty) return 'Введіть вагу';
                final weight = _parseWeight(text);
                if (weight == null) return 'Вага має бути числом';
                if (weight <= 0) return 'Вага має бути більшою за 0';
                return null;
              },
            ),
            const SizedBox(height: 12),
            _buildField(
              'Нотатка',
              noteController,
              maxLines: 3,
              validator: (value) {
                final text = value?.trim() ?? '';
                if (text.length > 250) {
                  return 'Максимум 250 символів';
                }
                return null;
              },
            ),
            const SizedBox(height: 20),
            SizedBox(
              height: 50,
              child: ElevatedButton(
                onPressed: _savePet,
                style: ElevatedButton.styleFrom(
                  backgroundColor: const Color(0xFF2563EB),
                  foregroundColor: Colors.white,
                  shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(16),
                  ),
                  elevation: 0,
                ),
                child: const Text(
                  'Зберегти',
                  style: TextStyle(
                    fontSize: 18,
```

```
                  fontWeight: FontWeight.w700,
                ),
              ),
            ),
          ),
        ],
      ),
    ),
  );
}

Widget _buildField(
  String label,
  TextEditingController controller, {
  TextInputType keyboardType = TextInputType.text,
  int maxLines = 1,
  String? Function(String?)? validator,
}) {
  return TextFormField(
    controller: controller,
    keyboardType: keyboardType,
    maxLines: maxLines,
    validator: validator,
    style: const TextStyle(
      color: Color(0xFF0F172A),
      fontWeight: FontWeight.w500,
    ),
    decoration: InputDecoration(
      labelText: label,
      labelStyle: const TextStyle(color: Color(0xFF475569)),
      filled: true,
      fillColor: const Color(0xFFF8FAFC),
      errorMaxLines: 2,
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(16),
        borderSide: const BorderSide(color: Color(0xFFD1D5DB)),
      ),
      enabledBorder: OutlineInputBorder(
        borderRadius: BorderRadius.circular(16),
        borderSide: const BorderSide(color: Color(0xFFD1D5DB)),
      ),
      focusedBorder: OutlineInputBorder(
        borderRadius: BorderRadius.circular(16),
        borderSide: const BorderSide(
          color: Color(0xFF2563EB),
          width: 1.6,
        ),
      ),
      errorBorder: OutlineInputBorder(
        borderRadius: BorderRadius.circular(16),
        borderSide: const BorderSide(color: Color(0xFFEF4444)),
      ),
      focusedErrorBorder: OutlineInputBorder(
        borderRadius: BorderRadius.circular(16),
        borderSide: const BorderSide(
          color: Color(0xFFEF4444),
```

```
            width: 1.6,
          ),
        ),
      ),
    );
  }
}
```

EditMedicalRecordScreen

```
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import '../../models/medical_record.dart';

class EditMedicalRecordScreen extends StatefulWidget {
  final MedicalRecord record;

  const EditMedicalRecordScreen({
    super.key,
    required this.record,
  });

  @override
  State<EditMedicalRecordScreen> createState() =>
_EditMedicalRecordScreenState();
}

class _EditMedicalRecordScreenState extends State<EditMedicalRecordScreen> {
  final _formKey = GlobalKey<FormState>();

  late final TextEditingController titleController;
  late final TextEditingController descriptionController;
  late final TextEditingController dateTimeController;
  late final TextEditingController categoryController;

  @override
  void initState() {
    super.initState();
    titleController = TextEditingController(text: widget.record.title);
    descriptionController = TextEditingController(text:
widget.record.description);
    dateTimeController = TextEditingController(text: widget.record.dateTime);
    categoryController = TextEditingController(text: widget.record.category);
  }

  @override
  void dispose() {
    titleController.dispose();
    descriptionController.dispose();
    dateTimeController.dispose();
    categoryController.dispose();
    super.dispose();
  }

  Future<void> _updateRecord() async {
    final isValid = _formKey.currentState?.validate() ?? false;
    if (!isValid) return;
```

```
  final updated = widget.record.copyWith(
    title: titleController.text.trim(),
    description: descriptionController.text.trim(),
    dateTime: dateTimeController.text.trim(),
    category: categoryController.text.trim(),
  );

  await FirebaseFirestore.instance
      .collection('medical_records')
      .doc(updated.id)
      .update({
    'title': updated.title,
    'description': updated.description,
    'dateTime': updated.dateTime,
    'category': updated.category,
  });

  Navigator.of(context).pop(true);
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: const Color(0xFFEAF0FF),
    appBar: AppBar(
      backgroundColor: const Color(0xFFEAF0FF),
      surfaceTintColor: Colors.transparent,
      elevation: 0,
      title: const Text(
        'Редагувати запис',
        style: TextStyle(
          color: Color(0xFF0F172A),
          fontWeight: FontWeight.w700,
        ),
      ),
      iconTheme: const IconThemeData(color: Color(0xFF1F2937)),
    ),
    body: Form(
      key: _formKey,
      autovalidateMode: AutovalidateMode.onUserInteraction,
      child: ListView(
        padding: const EdgeInsets.all(16),
        children: [
          _buildField(
            'Назва',
            titleController,
            validator: (value) {
              final text = value?.trim() ?? '';
              if (text.isEmpty) return 'Введіть назву запису';
              if (text.length < 3) return 'Мінімум 3 символи';
              return null;
            },
          ),
          const SizedBox(height: 12),
          _buildField(
```

Додаток В
(Технічне завдання)

1. Загальна характеристика системи

Назва проєкту: «Murkotun».

Об’єктом розроблення є мобільний застосунок для організації догляду за домашніми тваринами. Система призначена для збереження інформації про тварин, ведення медичних записів та планування процедур із використанням нагадувань.

Застосунок орієнтований на персональне використання та працює на мобільних пристроях. Основна взаємодія з системою відбувається нерегулярно, тому важливо забезпечити швидкий доступ до інформації та можливість виконання основних дій за мінімальний час.

У сфері догляду за тваринами значна частина інформації зберігається неструктуровано або взагалі не фіксується. Це ускладнює контроль за станом тварини та може призводити до пропуску важливих процедур. Запропонована система має забезпечити централізоване зберігання та обробку даних.

2. Мета та призначення розробки

Метою розроблення є створення мобільного застосунку, який дозволяє зберігати інформацію про домашніх тварин, вести медичні записи та створювати нагадування про необхідні дії.

Призначення системи полягає у спрощенні процесу догляду за тваринами за рахунок автоматизації обліку даних та своєчасного інформування про заплановані події. Результатом має бути програмний продукт у форматі мінімально життєздатного продукту (MVP), що реалізує базовий функціонал і може бути розширений у майбутньому.

3. Аналіз цільової аудиторії

Цільова аудиторія мобільного застосунку «Murkotun» охоплює власників домашніх тварин різного віку та досвіду.

Таблиця 3.2 – Цільова аудиторія

Категорія	Вік	Основні характеристики
Молоді власники	18-30 років	Вперше завели тварину, потребують структурованого інструменту для формування звичок догляду. Активно користуються мобільними застосунками.
Досвідчені власники	31-45 років	Мають одну або кілька тварин, цінують зручність синхронізації даних між пристроями та надійність зберігання медичної інформації.
Старше покоління	46-60 років	Потребують максимально простого інтерфейсу без зайвих функцій. Важлива інтуїтивність та зрозумілість кожної дії.

4.1.Функціональні вимоги до системи. Система повинна забезпечувати виконання таких функцій:

- реєстрація користувача;
- авторизація з перевіркою введених даних;
- створення, перегляд, редагування та видалення профілів тварин;
- збереження основної інформації про тварину;
- ведення медичних записів для кожної тварини;
- створення, редагування та видалення медичних записів;
- перегляд історії медичних подій;
- створення нагадувань із прив’язкою до тварини;
- встановлення дати, часу та типу повторення нагадувань;
- підтримка різних типів повторення;
- відображення списку нагадувань;
- увімкнення та вимкнення нагадувань;
- формування сповіщень у заданий час;
- збереження даних у централізованому сховищі.

4.2.Нефункціональні вимоги. Система повинна відповідати таким вказаним вимогам:

– коректна робота на мобільних пристроях із ОС Android 8.0 і вище;
– зрозумілий користувацький інтерфейс;
– мінімальна кількість дій для виконання основних операцій;
– стабільна робота системи сповіщень у фоновому режимі;
– можливість роботи без постійного підключення до мережі;
– забезпечення цілісності даних;
– можливість подальшого розширення функціоналу;
– обмежене використання ресурсів пристрою;
– передбачувана реакція системи на дії користувача.

5. Основні сценарії використання

Реєстрація та авторизація передбачає введення облікових даних і отримання доступу до функціоналу системи.

Робота з профілями тварин включає створення профілю, перегляд інформації, редагування або видалення записів.

Ведення медичних записів передбачає додавання нових записів, перегляд історії та внесення змін.

Робота з нагадуваннями включає створення події, встановлення часу виконання та отримання сповіщення.

6. Структура інтерфейсу

Інтерфейс застосунку повинен включати такі основні екрани:

– головний екран зі списком тварин;
– екран профілю тварини;
– екран медичних записів;
– екран нагадувань;
– екран авторизації та реєстрації;
– екран налаштувань.

7. Організація даних

Система повинна працювати з такими типами даних:

– дані користувача;
– дані про тварин;
– медичні записи;
– нагадування.

Дані повинні бути пов’язані між собою. Профілі тварин прив’язані до користувача, а медичні записи та нагадування – до конкретної тварини.

8. Обмеження

У межах розробки приймаються такі обмеження:

– реалізація системи у форматі мінімально життєздатного продукту;
– базовий рівень забезпечення безпеки;
– використання готових хмарних сервісів;
– відсутність інтеграції зі сторонніми системами;
– обмежений набір функцій.

9. Очікуваний результат

Результатом виконання технічного завдання є мобільний застосунок, що забезпечує збереження інформації про домашніх тварин, ведення медичних записів та формування системи нагадувань.

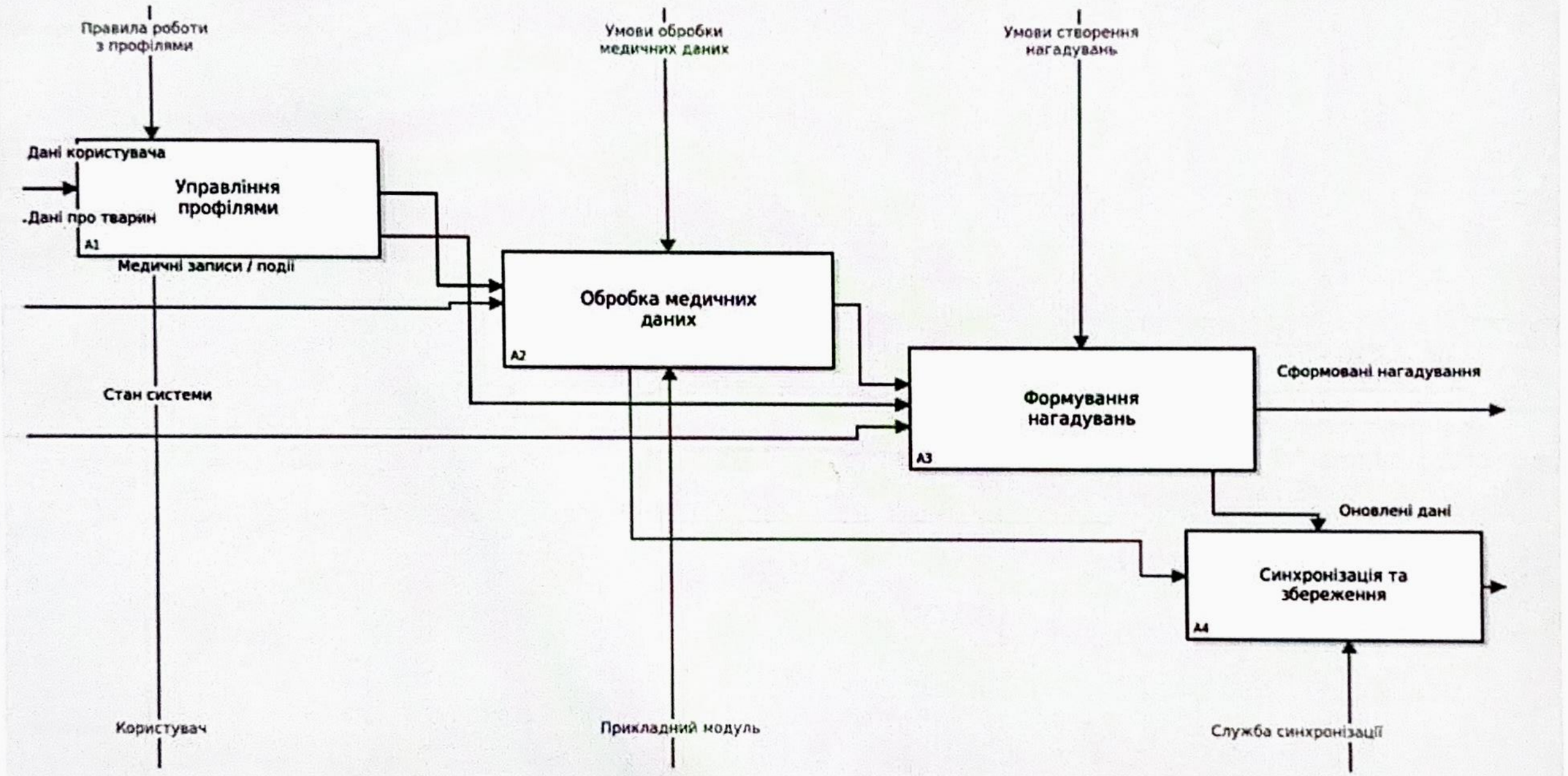
Система повинна стабільно працювати на мобільних пристроях, відповідати визначеним вимогам та бути придатною для подальшого розвитку.

ГРАФІЧНА ЧАСТИНА

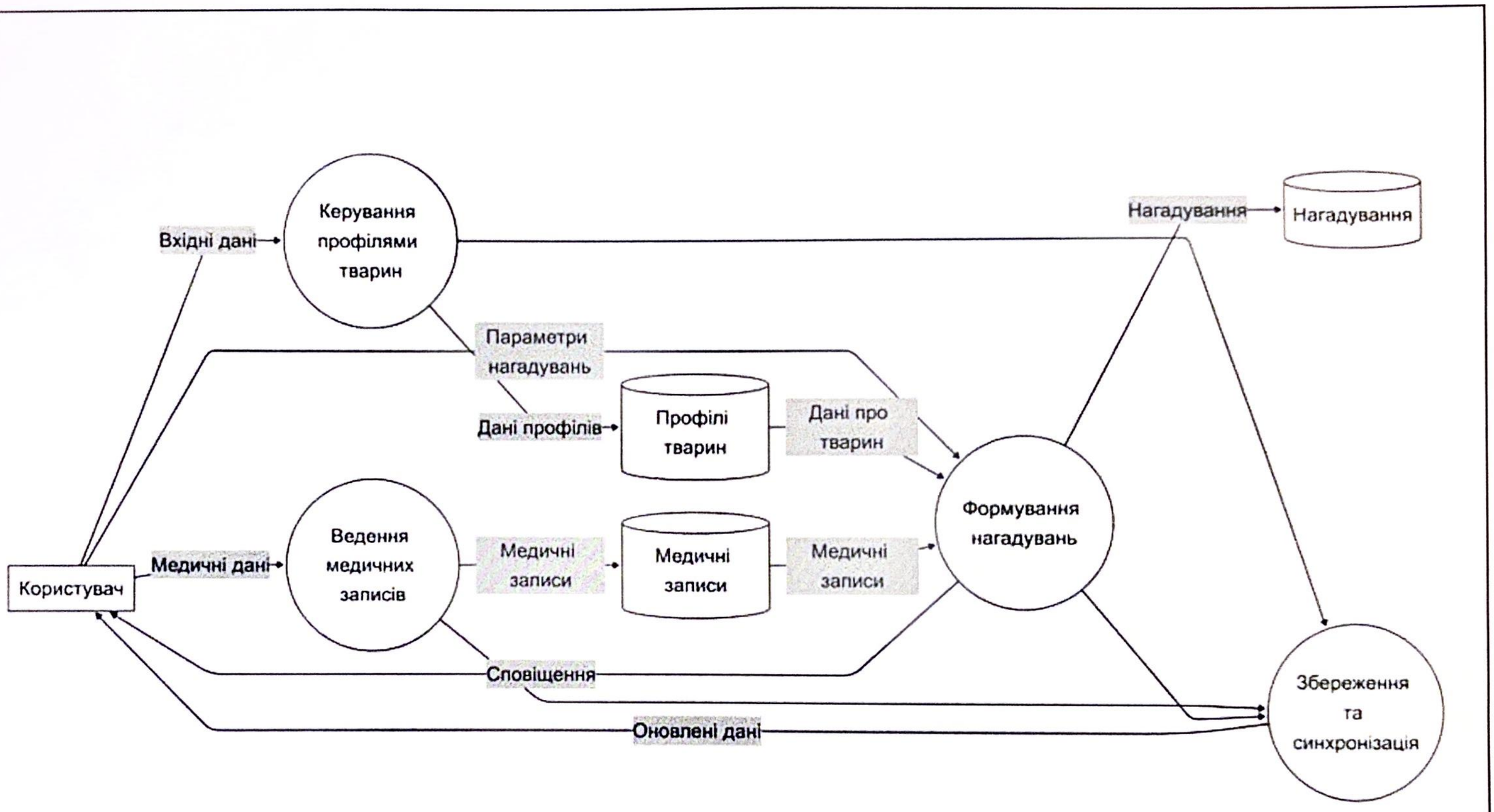
Контекстна діаграма A-0

					КвРІПЗ.2201120.01.25.ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Контекстна діаграма A-0	Літера	Маса	Масштаб
Розробила		Юхимчук М. В						
Керівник		Бедратюк Л. П.				Аркуш 1	Аркушів 3	
Консульт								
Н. Контр.		Бойко В. О.				ХНУ, ІПЗ-22-1		
Зав. каф.		Бедратюк Л. П.						

Діаграма декомпозиції А-1



					КвРІПЗ.2201120.01.25.ПЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробила		Юхимчук М. В.		28.05.25	Діаграма декомпозиції А-1	Літера	Маса / Масштаб
Керівник		Бедратюк Л. П.		28.05.25			
Консульт.						Аркуш 2	Аркушів 3
Н. Контр.		Бойко В. О.		28.05.25	ХНУ, ІПЗ-22-1		
Зав. каф.		Бедратюк Л. П.		28.05.25			



					КвРІПЗ.2201120.01.25.ПЗ			
Зм	Арк.	№ докум.	Підпис	Дата	Діаграма варіантів використання	Літера	Маса	Масштаб
Розробила		Юхимчук М. В.						
Керівник		Бедратюк Л. П.				Аркуш 3	Аркушів 3	
Консульт								
Н. Контр.		Бойко В. О.				ХНУ, ІПЗ-22-1		
Зав. каф.		Бедратюк Л. П.						

СУПРОВІДНІ ДОКУМЕНТИ

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Юхимчук Марина Валеріївна,
студент IV курсу спеціальності 121 – Інженерія програмного забезпечення,
група ІПЗ-22-1

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

підтверджую, що ознайомилась 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.

02 вересня 2025 р. _____
Підпис

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: Марина ЮХИМЧУК

Співавтор:

Назва: Мобільний застосунок для догляду та контролю за домашніми тваринами

Науковий керівник: д-р фіз.-мат. наук, проф. Леонід БЕДРАТЮК

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:5.3%

Коефіцієнт подібності 2:2.38%

Мікропробіли: 21

Заміна букв: 0

Інтервали: 0

Білі знаки: 79

Дата створення звіту: 2026-05-21 10:00:49.0

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 експерт

25.05.26

☰

Thu May 21 11:25:13 EEST 2026, Форкун Юрій Вікторович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (http://ap.km.ua) v-16.718

Максимальне співпадіння з одним документом 2.0%

Словники перевірки: UA, US, RU. **Помилок в документах: 12%**

	Документ		Сумарний збіг по Базі Даних	
ID: 271869 Назва: БКР Мобільний застосунок для догляду та контролю за домашніми тваринами Додано в БД: 2026-05-21 Автора: Марина ЮХИМЧУК Керівники: д-р фіз.-мат. наук, проф. Леонід БЕДРАТЮК Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	96547	768	4048 (4%)	53 (7%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «Бакалавр»

Дипломник Юхимчук Марина Валеріївна
Тема Мобільний застосунок для догляду та контролю за домашніми тваринами
Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 ; кількість сторінок записки 76 .
1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі розглянуто створення мобільного застосунку для догляду та контролю за домашніми тваринами. Проведено аналіз предметної області, сформульовано вимоги, обґрунтовано архітектурні та технологічні рішення. Реалізовано продукт з можливостями ведення профілів тварин, медичних записів та системою нагадувань. Проведено тестування, що підтвердило коректну роботу розробленого програмного забезпечення.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи Розробка проєкту розпочалася з обґрунтування доцільності створення мобільного застосунку для догляду за домашніми тваринами. Було визначено основні технічні орієнтири, сформульовано функціональні та нефункціональні вимоги, проведено аналіз існуючих програмних рішень і виявлено їх недоліки. На наступному етапі виконано проєктування архітектури застосунку з використанням комбінованого підходу, що поєднує клієнт-серверну та багаторівневу моделі. Реалізовано ключові функціональні модулі, зокрема управління профілями тварин, ведення медичних записів, систему нагадувань із підтримкою повторення та механізм синхронізації даних між локальним сховищем SQLite і хмарною базою Firebase Firestore. Завершальним етапом стало тестування застосунку, під час якого перевірено стабільність роботи функціональних модулів та відповідність реалізованих рішень заданим вимогам. Проведено ручне функціональне тестування основних сценаріїв взаємодії користувача із системою, а також автоматизоване юніт-тестування внутрішньої логіки. Результати засвідчили працездатність системи та відповідність розробленого продукту поставленим вимогам.

4. Позитивні сторони роботи Проєкт демонструє сучасний підхід до розроблення мобільного застосунку з використанням актуального технологічного стеку. Архітектурні рішення виконано з урахуванням вимог до масштабованості та зручності підтримки. Реалізовано повний набір основних функціональних модулів, що забезпечують практичну цінність продукту. У роботі грамотно застосовано можливості Flutter та Firebase на належному технічному рівні

5. Негативні сторони роботи Обмеженість функціоналу: реалізовано лише базові модулі без поглибленої системи аналітики чи статистики догляду. Не передбачено інтеграції з ветеринарними сервісами або можливості експорту медичних записів. Механізм синхронізації даних реалізовано у спрощеному вигляді без повноцінної обробки конфліктів при одночасному редагуванні з різних пристроїв.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення відповідає тематиці роботи та містить діаграми архітектури, варіантів використання, послідовностей, потоків даних, класів, ER-діаграму бази даних тощо. Пояснювальна записка оформлена згідно вимог і має логічну структуру викладення матеріалу.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота є завершеною, цілісною та технічно грамотною. У пояснювальній записці чітко прослідковується логіка побудови рішення – від аналізу предметної області та існуючих програмних рішень до практичної реалізації мобільного застосунку. Застосовані сучасні технології та архітектурні підходи забезпечили створення функціонального та практично цінного продукту. Робота добре структурована та наповнена ілюстративним матеріалом.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ к.т.н., доцент кафедри КІК, ХНУ Капустян Марія Вікторівна

"28" 05 2026 р. _____
(підпис)

SemanticAI
for Education

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

першого освітнього рівня «Бакалавр»

Студента: Юхимчук Марини Валеріївни
Група: ІПЗ-22-1

Тема: *«Мобільний застосунок для догляду та контролю за домашніми тваринами»*

Спеціальність: 121 – Інженерія програмного забезпечення

Короткий зміст пояснювальної записки

У вступі кваліфікаційної роботи окреслено мету, актуальність та завдання, пов'язані з розробкою мобільного застосунку для догляду за домашніми тваринами. Мета роботи полягає в створенні програмного забезпечення, яке забезпечить централізоване зберігання інформації про тварин, планування догляду та своєчасне інформування користувача. Актуальність теми підкреслюється зростанням кількості власників тварин та підвищенням вимог до якості догляду. Завдання включають аналіз предметної області, визначення вимог до програмного забезпечення, розробку архітектури та реалізацію мобільного застосунку.

Відповідність отриманих результатів роботи поставленим завданням

На основі аналізу тексту:

- **Завдання**: Аналіз предметної області догляду за домашніми тваринами.

- **Результат**: Описано еволюцію догляду, проблеми, які вирішує ПЗ, та структуру процесів.

Оцінка відповідності: **Повністю відповідають**.

- **Завдання**: Аналіз існуючих програмних рішень.
- **Результат**: Наведено перелік аналогів, їх характеристики та порівняння.

Оцінка відповідності: **В цілому відповідають**.

- **Завдання**: Визначення функціональних та нефункціональних вимог до ПЗ.
- **Результат**: Описано функціональні та нефункціональні вимоги, але без конкретизації.

Оцінка відповідності: **В основному відповідають**.

Оцінка розділів

Розділ 1

У розділі 1 проведено змістовний аналіз предметної області, її структурних та функціональних особливостей, а також аналіз наявного програмно-технічного забезпечення. Описано проблеми, які вирішує ПЗ, структуру процесів догляду, цільову аудиторію та їх інформаційні потреби. Визначено функціональні та нефункціональні вимоги.

Позитивні сторони: Описано еволюцію догляду за тваринами, актуальність теми, структуру процесів, цільову аудиторію та їх потреби. Визначено функціональні та нефункціональні вимоги.

Негативні сторони: Відсутні конкретні приклади застосунків, чітке формулювання проблем, деталі функцій, специфікації сценаріїв використання, а також інформація про безпеку даних.

Розділ 2

У розділі 2 розглянуто проєктування архітектури та структури системи, декомпозицію для мобільного застосунку, залежності та взаємодію, а також проєкт інтерфейсу користувача. Описано вибір технологій реалізації.

Позитивні сторони: Розглянуто кілька архітектур, враховано специфіку мобільної платформи, описано декомпозицію на модулі, враховано адаптивність UI, обрано Flutter як технологію.

Негативні сторони: Відсутні конкретні приклади реалізації архітектур, діаграми, деталі про шаблони проєктування, а також інформація про обробку станів.

Розділ 3

У розділі 3 описано реалізацію програмної частини продукту, локального та віддаленого зберігання даних, технічні та програмні вимоги, а також тестування

мобільного застосунку.

Позитивні сторони: Описано реалізацію основних екранів, узгодженість з архітектурою, використання локальної БД, інтеграцію з API, обґрунтовано вибір методів тестування.

Негативні сторони: Відсутні деталі про репозиторії, структуру таблиць, приклади тест-кейсів, а також візуалізація результатів тестування.

Позитивні сторони

Кваліфікаційна робота демонструє оригінальність у підході до розробки мобільного застосунку, що відповідає сучасним вимогам. Описані рішення є функціональними та зручними для користувача. Розробка враховує специфіку мобільних платформ, що підвищує її практичну значущість.

Недоліки

Робота потребує уточнень у деталях реалізації, зокрема в описі архітектури, функцій, сценаріїв використання та тестування. Відсутність конкретних прикладів та візуалізацій ускладнює сприйняття матеріалу.

Відгук в цілому

Кваліфікаційна робота є актуальною та має практичну значущість, оскільки відповідає потребам власників домашніх тварин у зручному та функціональному програмному забезпеченні. Зміст роботи відповідає темі та завданню, але потребує доопрацювання в частині деталізації та конкретизації.

Оцінка кваліфікаційної роботи

Оцінка: **добре**. Кваліфікаційна робота виконана в повному обсязі з дотриманням основних вимог, але має певні недоліки, які потребують виправлення.

Рекомендації

Рекомендується доопрацювати роботу, зокрема уточнити інформацію про архітектуру, функціональні можливості, сценарії використання, а також візуалізувати результати тестування. Це підвищить якість роботи та її готовність до захисту.

OpenAI API-асистент
Session ID: c9a5f374-f148-481e-b630-255e3baa4684
Підписано автоматично, модель gpt-4o-mini
Дата: 20.05.2026

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.
Назва кваліфікаційної роботи: «Мобільний застосунок для догляду та контролю за домашніми тваринами»
Автор: Юхимчук Марина Валеріївна
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Бедратюк Леонід Петрович, д-р фіз.-мат. наук, професор
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту.	**відповідає**
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. Робота приймається до захисту, але має бути відкоригована.	
3	Виявлені запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Виявлені запозичення частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
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.	
5	Інше:	

Підтвердження:
Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:
1) у тексті кваліфікаційної роботи системою перевірки на плагіат Anti-Plagiarism виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках, у структурі змісту, назвах назвах розділів/підрозділів, рамках форм, у назвах та URL-адресах публікацій переліку джерел посилання;
2) запозичення, виявлені у тексті роботи, є фрагментарними.
Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 2.0% з одного джерела. Загальна сумарна подібність у базі даних складає 4% за символами та 7% за лексемами. Не виявлено мікропробілів, зайвих білих знаків або маніпуляцій з інтервалами. З урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 27.01.26

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Леонід БЕДРАТЮК