Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Паламарчука Дениса Васильовича
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Мобільний застосунок для запису до ветеринара та організації догляду за тваринами
Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.2101082.01.10.ПЗ

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

Керівник канд. пед. наук, доцент _____ Наталія ПРАВОРСЬКА
Науковий ступінь, звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер ст. викладач _____ Ганна БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

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

3 червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Паламарчуку Денису Васильовичу
Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Мобільний застосунок для запису до ветеринара та організації догляду за тваринами

Керівник кваліфікаційної роботи Праворська Наталія Іванівна, канд. пед. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.02.2025 р. № 23
2. Строк подання студентом роботи на кафедру 06.06.2025 р.
3. Вихідні дані до роботи Матеріали переддипломної практики

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

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди, 19 шт.), діаграма варіантів використання, ER-діаграма, діаграма послідовностей, діаграма класів, діаграма діяльності, діаграма архітектури застосунки, схема екранів застосунку

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Бедратюк Г.І, ст. викладач		
Антиплагіат	Форкун Ю.В., к.т.н., доцент	12.05.2025	12.05.2025

7. Дата видачі завдання «2» 01 2025р.

КАЛЕНДАРНИЙ ПЛАН

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

Студент _____ Підпис Паламарчук Д. В Ініціали, прізвище

Керівник роботи _____ Підпис Проворська Н. І Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Мобільний застосунок для запису до ветеринара та організації догляду за тваринами».

Автор роботи: Паламарчук Денис Васильович.

Керівник роботи: Праворська Наталія Іванівна.

Пояснювальна записка: 76 с., 24 рис., 6 табл., 4 дод., 42 джерела.

Графічна частина: 2 креслення.

МОБІЛЬНИЙ ЗАСТОСУНОК, ДОГЛЯД ЗА ТВАРИНАМИ, FLUTTER, FIREBASE, BLoC.

Метою роботи є розробка мобільного застосунку для запису до ветеринара та організації догляду за домашніми тваринами, який дозволяє зручно керувати профілями тварин, фіксувати медичні записи та планувати візити до клініки.

У кваліфікаційній роботі проведено аналіз предметної області, досліджено аналогічні сервіси, визначено функціональні та нефункціональні вимоги, спроєктовано архітектуру системи, обґрунтовано вибір стеку реалізовано основні модулі мобільного застосунку та проведено тестування.

Для реалізації клієнтської частини використано фреймворк Flutter, що забезпечує кросплатформенну підтримку. У якості бекенду інтегровано хмарні сервіси Firebase. Для керування станом застосунку використано BLoC.

Розроблений мобільний застосунок надає користувачам зручний інтерфейс для створення профілів тварин, запису на прийом до ветеринара, ведення медичних історій, додавання нагадувань про вакцинації та процедури, збереження файлів та перегляду їх безпосередньо у застосунку.

Застосунок може бути корисним власникам домашніх тварин, які прагнуть цифрово організувати догляд, медичну документацію та візити до ветеринарів в одному інтегрованому середовищі.

02.06.25
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.2101082.01.10.ПЗ	Пояснювальна записка	76		
2	А4		Завдання на кваліфікаційну роботу	1		
3	А4		Анотація	1		
			Графічні документи			
4	А4	КвРІПЗ.2101082.01.10.Е8	Логічна модель бази	1		
			даних			
5	А4	КвРІПЗ.2101082.01.10.Е8	Схема екранів	1		
			застосунку			

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ. 2101082.01.10.ВД		
Виконав		Паламарчук Д.В.		02.06	Мобільний застосунок для запису до ветеринара та організації догляду за тваринами	Літ.	Арк. 1 / Аркушів 1
Керівник		Праворська Н. І.		02.06			
Рецензент		Калуський М.В.		02.06			
Н. Контр.		Бедратюк Г.І		02.06		ХНУ, ІПЗ-21-1	
Зав. Каф.		Бедратюк Л.П.		02.06	Пояснювальна записка		

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

СКБД	–	Система керування базами даних
API	–	Application Programming Interface
AWS	–	Amazon Web Services
BLoC	–	Business Logic of Component
CI/CD	–	Continuous Integration / Continuous Delivery
DI	–	Dependency Injection
ER	–	Entity-Relationship
FCM	–	Firebase Cloud Messaging
FK	–	Foreign Key
GPS	–	Global Positioning System
HTTP	–	Hypertext Transfer Protocol
PK	–	Primary Key
iOS	–	iPhone Operating System
MVC	–	Model-View-Controller
MVP	–	Model-View-Presenter
MVVM	–	Model-View-ViewModel
QR	–	Quick Response
SHA-256	–	Support Vector Classifier
SM	–	State Management
SQL	–	Structured Query Language
UX	–	User Experience
NoSQL	–	Not only SQL

ВСТУП

Зараз мобільні застосунки є дуже важливим аспектом нашого життя. Кожна людина використовує мобільні технології для тієї чи іншою мети, наприклад використання месенджерів дозволяє людям залишатись на зв'язку постійно, підтримуючи комунікацію з родичами, друзями чи колегами. В свою чергу, використання застосунків для розваг, такі як ігри, музикальні або стрімінгові сервіси допомагають людям розслабитися та відпочити у будь-який вільний та зручний для них час, що робить мобільні технології важливим інструментом для проведення дозвілля.

В загальному, мобільних застосунків які вирішують ті чи інші проблеми є дуже і дуже багато, і згідно з статистикою [1] відсоток завантажень застосунків зростає порівняно з минулими роками. Медична сфера не стала виключенням. Однією з важливих складових медичної сфери є надання ветеринарних послуг, адже турбота про самопочуття домашніх улюбленців є головним та ключовим аспектом для усіх власників тварин, тому зручна та ефективна організація запису до спеціаліста ветеринарної медицини, а також введення інформації про тварин набувають особливої актуальності.

Вебсайти та мобільні застосунки для ветеринарного догляду можуть значно полегшити та покращити процес запису на прийом до потрібного ветлікаря, отримати рекомендації по догляду або ж просто допомогти у введені історії здоров'я тієї чи іншої тварини.

Проблема, на розв'язання якої спрямована дана кваліфікаційна робота, полягає у створенні зручного і персоналізованого рішення для запису на прийом до ветеринара та зручного моніторингу стану домашніх улюбленців. На цей час більшість вже існуючих рішень мають обмежені функції, по типу календар запису чи нагадування, але вони не враховують такі потреби тварин, як рекомендації або медичний анамнез. У зв'язку з цими причинами і виникає потреба у створені подібного мобільного застосунку.

Актуальність цієї теми визначається потребою створення мобільного застосунку, який спростить процес запису до ветлікаря, введення важливої інформації про тварину, а також покращить рівень обслуговування у медичній сфері. До того ж мобільні застосунки показують комерційну привабливість, що робить їх вигідними як і для простих користувачів, так і для власників бізнесу.

Мета роботи полягає в розробці інноваційного мобільного застосунок для запису на прийом до ветеринара та догляду за тваринами, який допоможе забезпечити ефективне управління процесами догляду за домашніми тваринами та спростить процес взаємодії між ветлікарями та власниками тварин.

Отже, для досягнення мети потрібно:

– виконати аналіз вже існуючих мобільних застосунків для ветеринарних послуг та догляду за тваринами;

– встановити основні вимоги до розроблюваного застосунку на основі аналізу ринку та потреб користувачів;

– вибрати та розробити архітектуру мобільного застосунку відповідно до визначених вимог;

– виконати програмну реалізацію застосунку, включаючи клієнтську та серверну частини;

– провести тестування усього мобільного застосунку, починаючи від UI-частини, закінчуючи серверною частиною.

Розроблений мобільний застосунок призначений саме для власників домашніх тварин, які хочуть зручно записуватися на прийом до ветлікаря. Він допоможе спростити процес взаємодії з ветеринарними клініками, забезпечуючи швидкий доступ до важливої інформації, нагадування про візити та лікування, а також можливість зберігати медичні записи в одному місці.

Отже, дана кваліфікаційна робота спрямована на створення програмного продукту, який підвищить рівень організації ветеринарного догляду, надасть інструменти для запису на прийом та ведення медичних записів, а також сприятиме покращенню якості життя домашніх тварин та їх власників.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Мобільний застосунок [2] (англ. mobile application) – це програмне забезпечення, яке призначене для використання на телефонах, планшетах або інших мобільних пристроях. В цілому, мобільні застосунки можна розділити на три типи [3]:

– нативні застосунки: вони розробляються під конкретну платформу. Такий тип застосунків дозволяє використовувати максимум можливостей операційної системи;

– мобільні вебзастосунки: цей тип застосунків являється мобільною версією вебсайту, але і мають доступ до можливостей операційної системи, що дозволяє інтегрувати деякі функції пристрою;

– кросплатформені(гібридні) застосунки: розробляються такими технологіями як Flutter, React Native, Xamarin та інші. Перевагами таких застосунків є те, що у них одна кодова база, яка працює може одразу працювати на декілька платформ.

Мобільні застосунки, не залежно від свого типу активно використовуються в різних сферах, наприклад:

– у медицині: мобільні застосунки допомагають пацієнтам та лікарям у веденні здорового способу життя, моніторингу стану здоров’я, нагадуванні про прийом ліків, бронюванні візитів, телемедичних консультаціях та доступі до медичних карток.

– в освіті: завдяки мобільним додаткам учні та студенти мають можливість дистанційно навчатися, переглядати навчальні матеріали, проходити тести, брати участь у відеоуроках, спілкуватися з викладачами та однокласниками.

– у сфері розваг: застосунки для перегляду фільмів, прослуховування музики, ігор, читання книг, соціальних мереж – все це забезпечує користувачам дозвілля в зручному форматі та з великою кількістю персоналізованих можливостей.

– у логістиці та транспорті: застосунки використовуються для замовлення таксі (наприклад, Uber, Bolt), відстеження доставки товарів, планування маршрутів, бронювання квитків, а також для управління автопарком і вантажними перевезеннями.

Сфер де використовуються мобільні застосунки набагато більше, адже вони стали невід'ємною частиною сучасного життя. Завдяки своїй зручності, мобільності та широкому функціоналу, застосунки допомагають вирішувати повсякденні завдання швидко та ефективно. Сьогодні майже кожна галузь діяльності має власні цифрові рішення у вигляді мобільних застосунків.

Тому, перед створенням нового мобільного застосунку, важливо чітко розуміти потреби користувачів та особливості галузі, в якій він буде застосовуватися у майбутньому.

Звичайно, що розробка будь-якого мобільного застосунку розпочинається з етапу дослідження предметної області, адже нам потрібно знати для кого буде розроблятися застосунок, тобто можна виділити переліки основних запитань, на які нам потрібно дати відповідь:

– хто є цільовою аудиторією застосунку;

– які ключові функції повинен мати застосунок;

– на яких платформах застосунок буде доступний;

– які конкурентні застосунки вже існують на ринку.

Для початку проводився аналіз існуючих платформ та їх частка у світі, адже правильний вибір операційної системи визначає охоплення цільової аудиторії та ефективність подальшої розробки застосунку.

Було визначено найпопулярніші операційні системи серед користувачів. У таблиці 1.1 розподіл по світу [4], а у таблиці 1.2 по Україні [5].

Таблиця 1.1 – Ринкова частка мобільних операційних систем у світі

Операційна система	Android	iOS	Samsung	Unknown	KaiOS	Windows
Ринкова частка (%)	72.24%	27.28%	0.31%	0.10%	0.03%	0.01%

З таблиці 1.1 видно, що найпопулярнішими операційними системами у світі на мобільні пристрої є Android та iOS, які в сумі займають 99.52% частки усього ринку і є основними платформами для сучасних смартфонів. Android залишається лідером із 72.24% ринку, це пояснюється тим, що є широкий вибір пристроїв від різних виробників по всьому світу та доступністю в різних цінових сегментах і відкритою екосистемою. iOS в свою чергу, маючи 27.28% ринку, займає значну частку завдяки преміальному сегменту, високій лояльності користувачів та інтеграції з екосистемою Apple.

Таблиця 1.2 – Ринкова частка мобільних операційних систем в Україні

Операційна система	Android	iOS	Samsung	Linux	Windows	Symbian
Ринкова частка (%)	66.48%	33.15%	0.26%	0.05%	0.04%	0.01%

Переглянувши таблицю 1.2, можна дійти висновку, що ситуація в Україні не сильно відрізняється від світової, проте спостерігаються невеликі відмінності. Android як і була, так і залишається домінуючою операційною системою, хоча його ринкова частка трішки нижча, ніж у світі, становлячи близько 66.48%. Це пояснюється доступністю великої кількості бюджетних і середньоцінових смартфонів на Android, які мають велику популярність серед українських користувачів.

В свою чергу iOS має вищу ринкову частку в Україні (33.15%) порівняно зі світовими показниками. Це може бути пов’язано з тим, що продукція Apple користується великою популярністю серед українських користувачів, особливо в великих містах-мільйонниках, де люди частіше надають перевагу пристрою преміумкласу.

Якщо говорити про такі операційні системи як: Samsung, Windows, KaiOS Symbian чи інші, то розробляти застосунок під них немає сенсу, оскільки витрачати час та ресурси на розробку, а в майбутньому і на підтримку на таких операційних системах недоцільно через маленьку кількість користувачів та дуже малу кількість фідбеку. До того ж нашому застосунку не потрібен доступ до спеціальних можливостей, та потрібно максимально охопити кількість користувачів, розробка проводилася під операційні системи, які мають найбільшу популярність, тому було вибрано саме гібридний тип застосунку.

Стосовно цільової аудиторії застосунку, то зрозуміло, що користувачами будуть саме ті люди які мають домашніх улюбленців (власники тварин), оскільки саме вони найбільш зацікавлені у догляді за своїми тваринами та потребують зручного інструменту для організації ветеринарного обслуговування. Проте, цільова аудиторія не обмежується лише власниками тварин. До потенційних користувачів також можна віднести ветеринарні клініки або притулки для тварин, які можуть використовувати застосунок для взаємодії з клієнтами та оптимізації своїх послуг.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Через те, що розробка застосунку саме під такі операційні системи як iOS та Android, то відповідно потрібно шукати уже існуючі аналоги на таких цифрових платформах як: AppStore для iOS та Play Market для Android. Ці платформи є основними каналами розповсюдження мобільних застосунків, де користувачі можуть знаходити, завантажувати та оновлювати застосунки, а також залишати відгуки та оцінки, що допомагає розробникам аналізувати потреби своєї цільової аудиторії та покращувати свій продукт.

Застосунки, які були зарекомендовані алгоритмами цих платформ були такі застосунки «11Pets», «AnimalID» та «DogCatApp».

11Pets [6] – мобільний застосунок, який призначений для догляду за домашніми тваринами та управління їх здоров’ям та доступний безплатно для усіх користувачів.

Застосунок підтримується на Android починаючи з версії 7.0 та на iOS починаючи з версії 15 включно та має 500 тис. + завантажень.

Судячи з опису у Play Market та інформації з офіційного вебсайту, данний застосунок має такі функції:

– введення медичних записів: застосунок дозволяє зберігати інформацію про ліки, процедури, відвідування та вакцинації;

– нагадування про догляд: можливість налаштовувати сповіщення про необхідність вакцинації, прийом вітамінів чи ліків;

– фотогалерея тварин: застосунок дозволяє завантажити декілька фотографій для певної тварини;

– моніторинг прийому їжі: можна стежити за дієтою та записувати, що та коли їсть тварина;

– можливість бронювання: застосунок дозволяє забронювати послуги у ветеринарів, тренерів, грумерів тощо.

З точки зору функціоналу, застосунок дуже багатофункціональний, тобто в якійсь мірі вирішує усі важливі проблеми людини, забезпечуючи інструменти для організації усієї важливої інформації щодо домашніх улюбленців.

Стосовно інших, другорядних функцій, то застосунок дозволяє змінювати інформацію про користувача, ділитися нею, та в разі чого змінювати або видаляти її(з можливістю повністю видалити свій профіль). Є можливість зміни таких речей як мови застосунку, формат часу та дати, грошові формати. Налаштування для тварин також присутнє, застосунок підтримує різні одиниці виміру, як для системи «SI» так і Американські та Британські формати.

Стосовно візуальної частини застосунку, то сторінки виконані в привабливому стилі. Дизайн основних сторінок застосунку можна подивитись на рисунку 1.1.

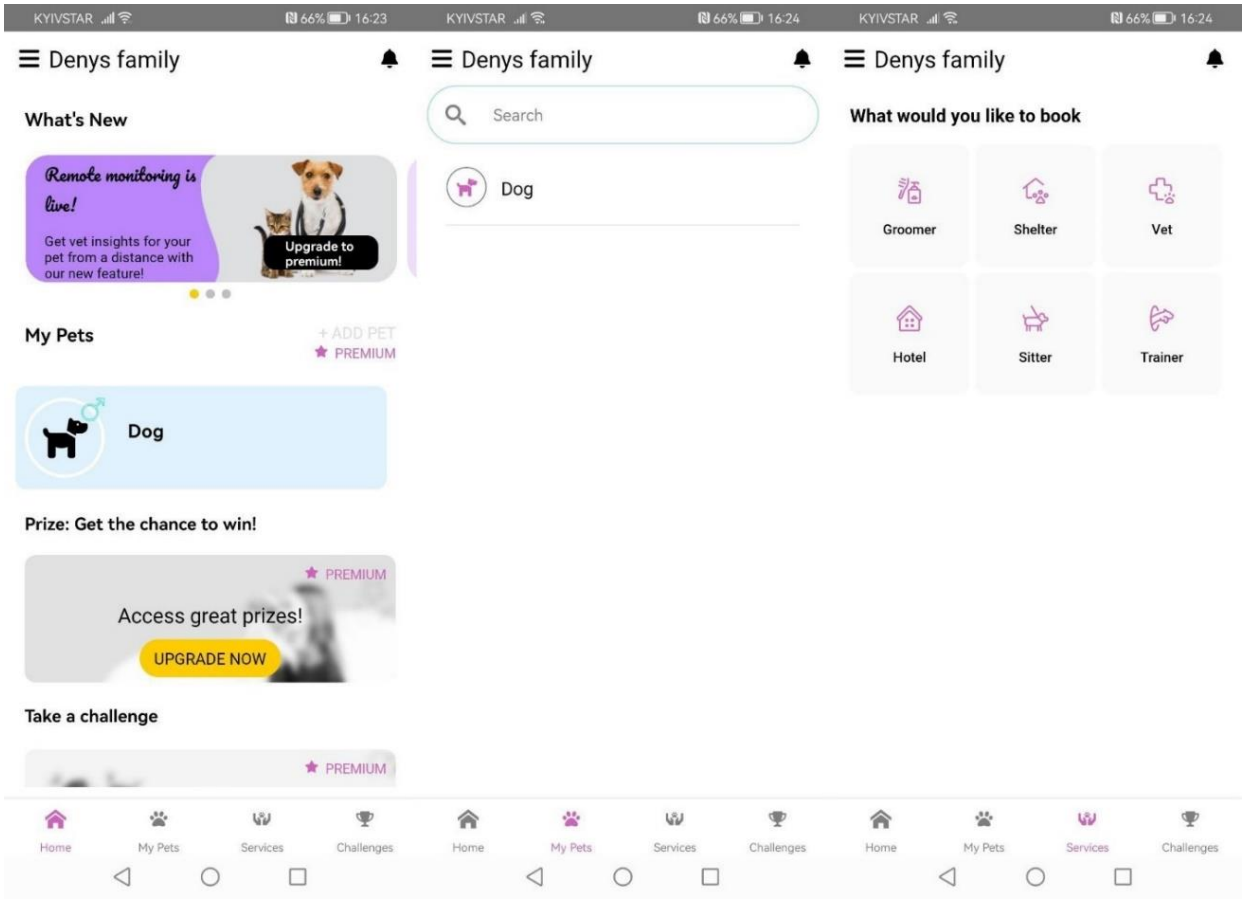


Рисунок 1.1 – Дизайн основних сторінок застосунку «11Pets»

Серед недоліків застосунку, то застосунок не надає можливості змінити тему, тобто доступна лише базова тема – світла, подібний недолік для деяких користувачів може виникати дискомфорт. Застосунок підтримує вісім мов, серед яких немає підтримки української версії. Також мінусом застосунку є те, що багато функціоналу недоступно без преміумпідписки. Прекрасним прикладом цього мінусу є те, що без преміумпідписки є те, що ви можете лише додати лише одного домашнього улюбленця, тобто, перед власниками декількох домашніх улюбленців постає вибір: купити преміумпідписку або використовувати інший застосунок.

Наступний мобільний застосунок це «AnimalID» [7], який допомагає власникам домашніх улюбленців вести інформацію про тварин, робити замовлення продуктів(і не тільки), записатись на прийом до ветлікарів і в разі чого знайти загубленого улюбленця за допомогою унікальних ідентифікаторів.

Застосунок підтримується на Android починаючи з версії 9.0 та на iOS починаючи з версії 16 включно. Має 100 тис. + завантажень.

Покористувавшись цим застосунком, можна виділити такі функції:

– персональний профіль тварини: зберігає всю необхідну інформацію про домашнього улюбленця, включаючи фотографії, медичні документи, контактні дані власника та опис тварини;

– реєстрація в міжнародній базі даних тварин: застосунок забезпечує безплатну реєстрацію тварини в міжнародній онлайн-базі даних, інтегрованій з Europetnet, яка є найбільшою європейською базою даних ідентифікованих тварин та їх власників;

– нагадування та нагороди: користувачі можуть отримувати нагадування про майбутні заходи по догляду за твариною, наприклад про вакцинацію, прийом вітамінів чи медичні прийоми у ветеринарів, а також отримувати за це винагороди від партнерів програми за активне використання застосунку;

– інтеграція з ветеринарними клініками: застосунок має можливість підключатися до ветеринарних клінік для запису на прийом, отримання консультацій або інших послуг;

– пошук тварин: застосунок дозволяє додавати певні ідентифікатори до профілю тварини, такі як QR-код або мікрочип, що допомагає швидко повернути улюбленця у разі втрати.

Щодо візуальної частини, то дизайн виконаний в мінімалістичному стилі. Застосунок має три основні сторінки, де знаходиться увесь потрібний користувачеві функціонал.

На початковій сторінці ми може перейти в свій профіль, та в разі чого змінити інформацію про користувача, а також додати, або переглянути вже існуючого улюбленця. Трішки нижче, на цій же сторінці можна перевірити захист тварини(наявність відповідних документів або мікрочипів), переглянути календар подій або статус замовлень. Ця сторінка зручна тим, що увесь функціонал знаходиться в одному місці.

На другій сторінці просто зображені діючі акції та промокоди які до того ж доступні лише преміумпідписникам, тому багато уваги на цю сторінку звертати не потрібно.

На останній сторінці просто знаходиться вся другорядна інформація, така як інформація про додаток, умови використання, поширенні запитання. Особливу увагу слід звернути на налаштування, оскільки вони дуже подібні до тих, що були у застосунку 11Pets: є налаштування форматів, а також можливість вибору мови, причому застосунок підтримує аж 14 мов серед яких є і українська. Дизайн сторінок можна переглянути на рисунку 1.2.

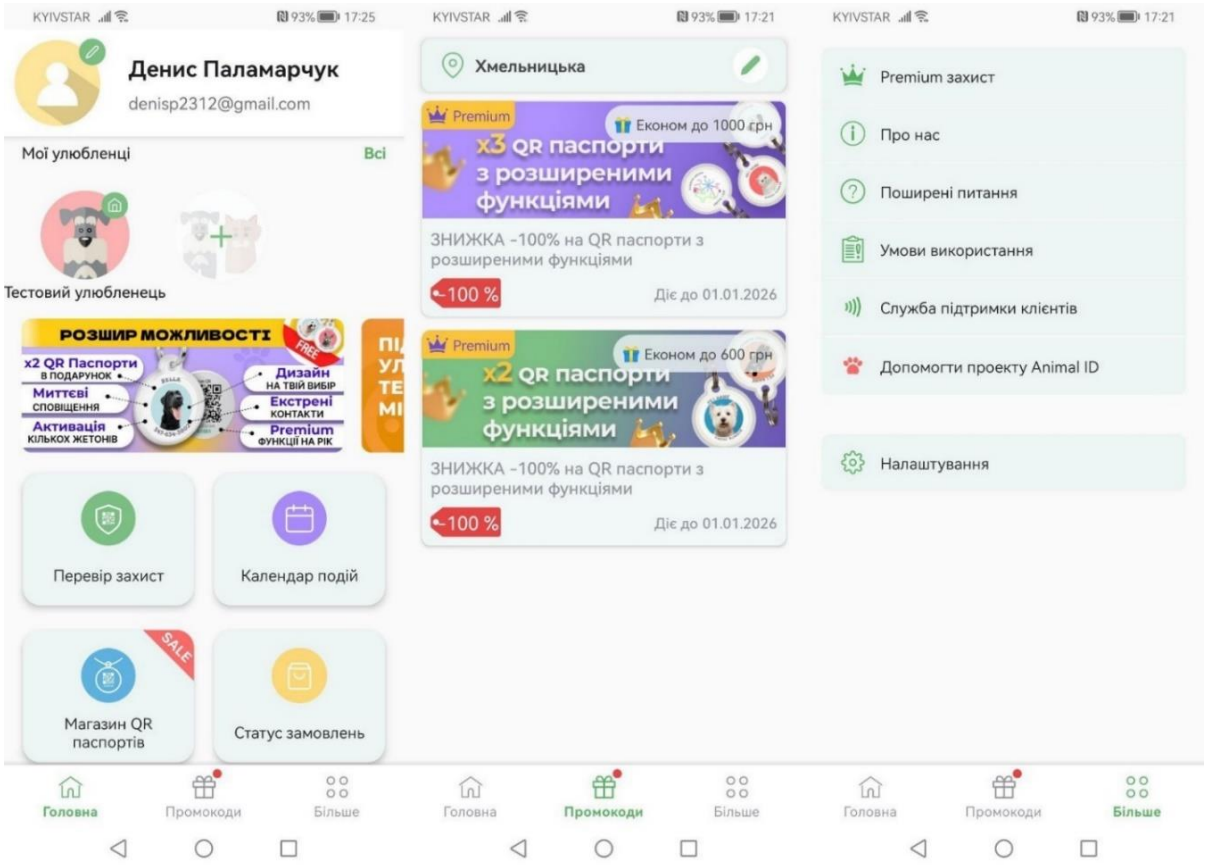


Рисунок 1.2 – Дизайн основних сторінок застосунку «Animal ID»

Серед недоліків, хотілось би виділити відсутність налаштування теми, як і у минулому застосунку присутня лише одна кольорова гамма – світла. Також, без преміумпідписки недоступний певний функціонал, але на відміну від

минулого застосунку відсутність преміумпідписки не сильно впливає на кількість функціоналу, який надає застосунок.

Останній застосунок – «DogCatApp» [8]. Цей застосунок як і минулі два, призначений для догляду за домашніми улюбленцями, але акцент зроблений саме на зберігання інформації про тварину. Застосунок був завантажений понад 50 тис. раз. Основні сторінки застосунку зображені на рисунку 1.3.

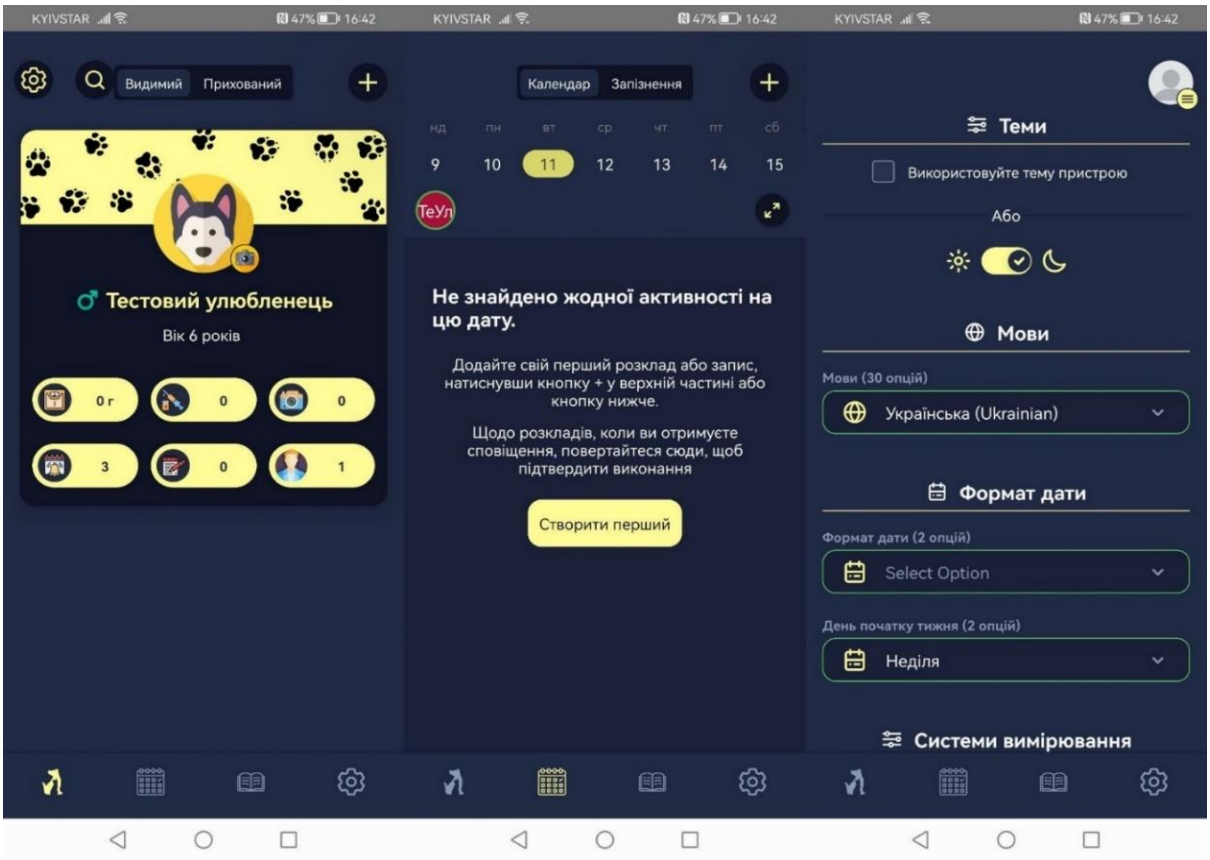


Рисунок 1.3 – Дизайн основних сторінок застосунку «DogCatApp»

Застосунок має такі функції:

– зберігання інформації: застосунок дозволяє зберігати багато різної інформації про тварину, вагу, вакцинацію, документи тощо;

– вибір ролей користувачів: користувач може обрати для себе роль, що зручно для інших, оскільки під час взаємодії вони будуть знати, яку саме роль ви відіграєте у відношенні до тварини (власник, доглядач або той, хто вигулює);

– планування подій: є можливість створити нагадування для різних подій, таких як прийом ліків, вакцинації, годування тощо;

– запис до ветеринара: застосунок має функцію запису до ветеринара, але має набагато менший перелік ветеринарних клінік, ніж минулі два застосунки;

– підтримка кастомізації інтерфейсу: застосунок, не тільки підтримує дві кольорові теми – чорну та світлу, але й надає можливість змінювати певні елементи, їх порядок.

Серед недоліків застосунку було виділено:

– погана адаптація інтерфейсу: телефон, на якому проводився аналіз застосунку має розмір 6.59 дюймів, і якщо подивитись на рисунок 1.3 , то видно, що певні елементи погано відцентровані, або наїжджають один на одного;

– проблеми локалізації: хоч і застосунок підтримує доволі велику кількість мов – понад 30, якість перекладу не найвищому рівні, і як видно на рисунку 1.3, деякий текст має граматичні помилки.

У таблиці 1.3 нижче зображено порівняння проаналізованих застосунків.

Таблиця 1.3 – Порівняння функцій застосунків

Функціонал	11Pets	AnimalID	DogCatApp
Реєстрація	+	+	+
Введення медичних записів(документів)	+	+	+
Нагадування	-	+	+
Фотогалерея тварин	+	-	+
Можливість бронювання послуг	+	-	-
Інтеграція з клініками	+/-	+	-
Пошук загублених тварин	-	+	-
Зміна теми або елементів інтерфейсу	-	-	+
Зміна мови, формату часу	+	+	+
Підтримка української мови	-	+	+
Наявність преміумаккаунт для усіх функцій	+	+	+

1.3 Визначення функціональних та нефункціональних вимог до програмного забезпечення

Функціональні вимоги [9-10] – це функції або дії, які система або програмне забезпечення повинна виконувати. Саме ці вимоги визначають, що розроблений застосунок повинен виконувати, тобто описують поведінку системи з користувачем або іншим програмним забезпеченням.

Після проведеного аналізу вже існуючих застосунків, розроблюваний застосунок повинен містити такі функціональні вимоги:

– наявність реєстрації: процес реєстрації має бути швидкою і простою, з можливістю використання різних способів аутентифікації (email, Google або з використанням соцмереж). Реєстрація в першу чергу потрібна для того, щоб зберегти дані користувачів у хмарі;

– введення інформації про тварин: користувач повинен мати можливість, створювати домашнього улюбленця, а також оновлювати або видаляти інформацію про нього;

– наявність нагадувань: застосунок повинен дозволяти користувачам встановлювати нагадування для подій, пов’язаних із їхніми домашніми тваринами. Це можуть бути нагадування про прийом ліків, візити до ветлікаря, прогулянки чи годування. Користувачі повинні мати можливість налаштовувати частоту та час нагадувань, а також отримувати сповіщення в обраний ними час;

– темна та світла тема: можливість вибору між темною та світлою темою забезпечить комфорт користувачів;

– наявність локалізації: застосунок повинен підтримувати українську та англійську мови, з майбутнім розширенням переліку мов.

Нефункціональні вимоги [9-10] – це вимоги, які визначають, як саме система повинна працювати, але не описують її конкретні функції.

Для мобільного застосунку з догляду за тваринами можна визначити такі нефункціональні вимоги:

– сумісність: мобільний додаток повинен підтримувати операційну систему iOS, версія 14+ та операційну систему Android, версія 8+;

– безпека: паролі повинні зберігати у хешованому вигляді, з використанням алгоритму SHA-256 [11]. Всі запити до Firestore повинні перевірятися через Firebase Security Rules;

– масштабованість: система повинна підтримувати одночасну роботу не менше ніж 10 000 активних користувачів без деградації продуктивності;

– зручність використання: затосунок повинен підтримувати відповідні рекомендації для операційних систем, Material Design [12] для Android та Human Interface Guidelines [13] для iOS.

UML (Unified Modeling Language) [14] – це уніфікована мова моделювання, яка використовується для візуального представлення структури та поведінки програмних систем. UML допомагає проєктувати, аналізувати та документувати програмне забезпечення, забезпечуючи спільну мову для розробників, аналітиків і замовників. UML містить різні типи діаграм, які поділяються на структурні (наприклад, діаграма класів) та поведінкові (наприклад, діаграма варіантів використання, діаграма послідовності).

Діаграма варіантів використання [15] (Use Case Diagram) – це одна із UML-діаграм, яка моделює взаємодію користувачів із системою. Вона наочно показує, які функції (в даному випадку варіанти використання) доступні користувачам та як ці функції взаємодіють між собою. Основі елементи діаграми варіантів використання:

Актор – це користувач або система, яка взаємодіє з програмою. На діаграмі зображується як людиноподібна фігура. Наприклад: «Користувач»;

Варіант використання (Use Case) – це конкретна функція або можливість системи, яку може виконувати актор. Позначається овалом із назвою дії. Наприклад: «Авторизація», «Запис на прийом».

Include (<<include>>) [16] – це відношення між варіантами використання, яке вказує, що один сценарій обов’язково містить інший. Використовується,

коли певна дія завжди передбачає виконання додаткової піддії. Наприклад, процес «Реєстрація» включає «Валідацію», оскільки перед створенням облікового запису необхідно перевірити коректність введених даних. Графічно позначається стрілкою з міткою <<include>>.

Extend (<<extend>>) [17] – показує опціональну або розширену поведінку. Використовується, коли певна дія може відбутися, але в цей час вона не є обов'язковою. Наприклад, "Видалити улюбленця" є розширенням "Переглянути улюбленця", оскільки переглядати улюбленця можна і без видалення (наприклад, безкоштовний прийом). Позначається штрихованою стрілкою з надписом <<extend>>.

Діаграма варіантів використання для користувача застосунку зображена на рисунку 1.4.

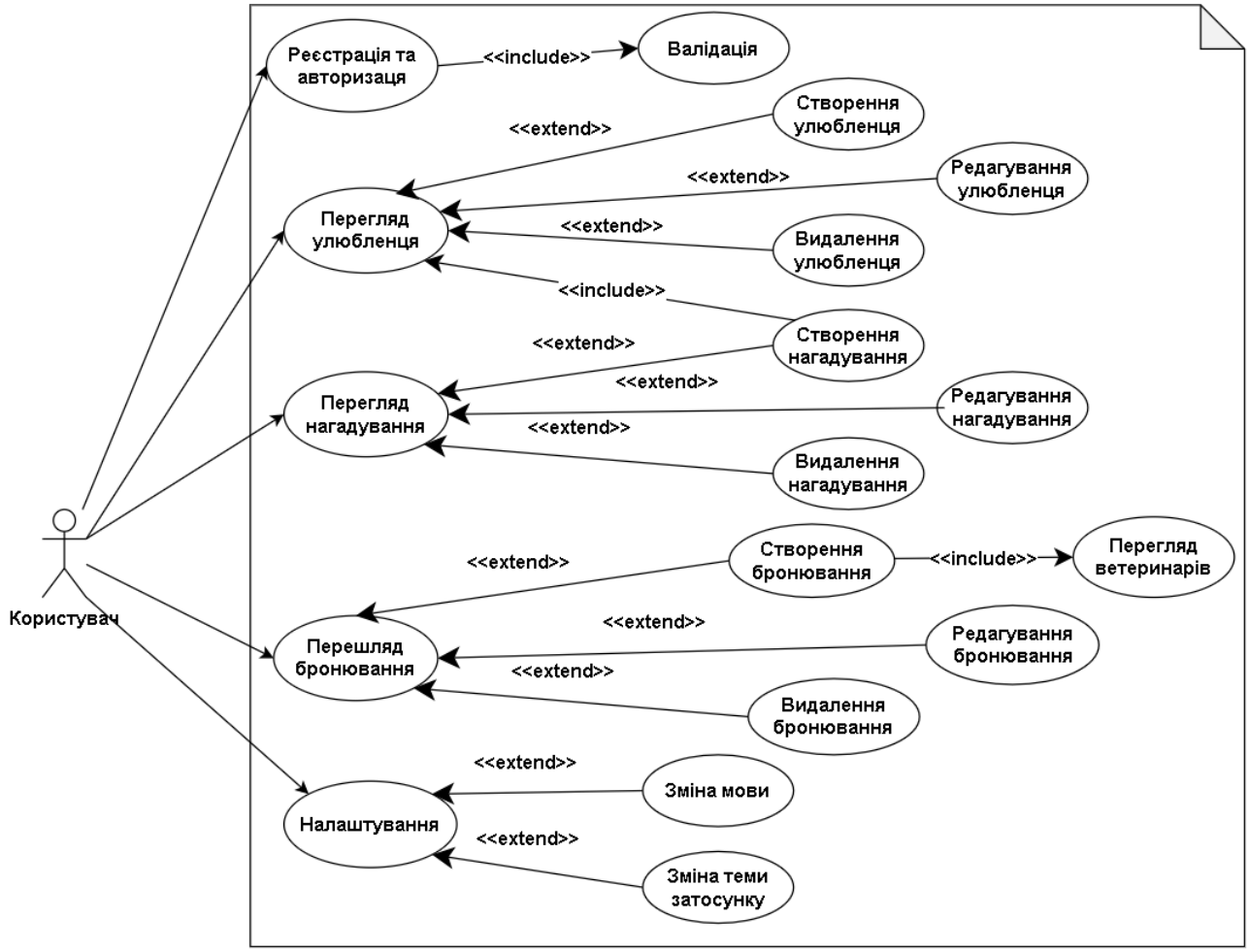


Рисунок 1.4 – Діаграма варіантів використання

Діаграма послідовності[15] – це візуальне представлення того, як певні об’єкти або компоненти у системі взаємодіють між собою протягом сценарію у часі. Вона показує, хто ініціює взаємодію, які повідомлення надсилаються, у якому порядку це відбувається та врешті-решт як реагують інші компоненти. Основна ідея полягає в тому, щоб відобразити хронологію обміну повідомленнями між компонентами процесу – від користувача до сервісів.

На рисунку 1.5 зображена діаграма послідовності де показаний повний сценарій взаємодії користувача з мобільним застосунком для догляду за тваринами. Спочатку користувач входить у застосунок, де проходить аутентифікацію через Firebase Auth. Далі користувач створює запис, вибираючи тварину, ветлікаря і дату – ці дані зберігаються у Firestore, а для нагадування викликається хмарна функція, яка планує push-сповіщення через FCM. Окремо зображено процес додавання нагадування про догляд за твариною, наприклад, вакцинацію чи годування. Інформація знову зберігається у Firestore, а Cloud Function планує повторюване нагадування через FCM.

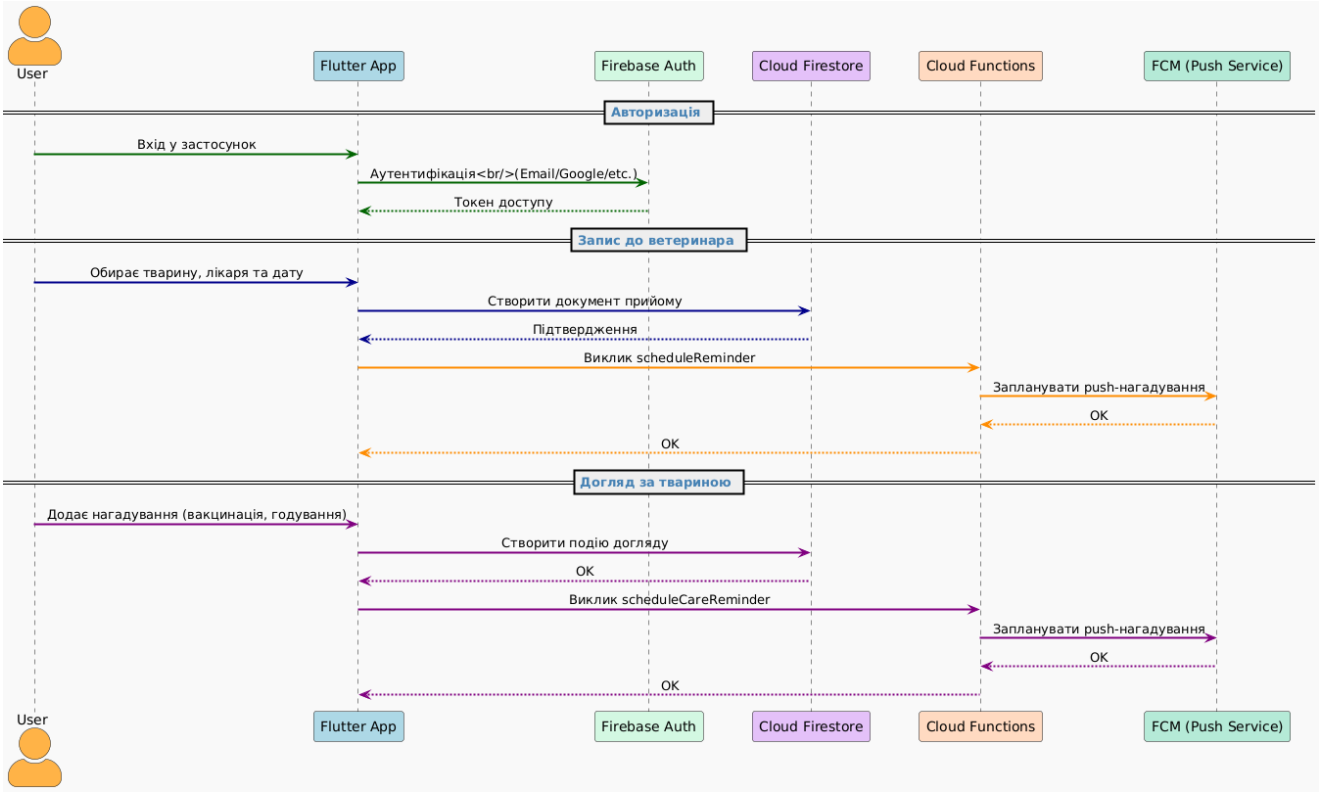


Рисунок 1.5 – Діаграма послідовності

Діаграма класів – це вид діаграми в UML (Unified Modeling Language), який використовується для візуалізації структури системи через класи, їхні властивості, методи та зв’язки між ними. Вона допомагає розробникам зрозуміти, як об’єкти в системі пов’язані між собою, які мають атрибути і яку поведінку реалізують. Така діаграма часто використовується на етапі проєктування програмного забезпечення для моделювання об’єктно-орієнтованої архітектури.

На рисунку 1.6 зображена діаграма класів яка показує структуру ветеринарного застосунку. Вона містить моделі користувача, домашніх тварин, нагадувань, записів на прийом, лікарів і допоміжних зв’язків між ними. Користувач може мати кількох тварин, кожна тварина може мати нагадування і бути записаною на прийом. Прийом пов’язаний як з користувачем, так і з лікарем. Окремі сервіси містять, які відповідають за аутентифікацію, керування тваринами, нагадуваннями, записами та лікарями, і взаємодіють із відповідними класами моделі.

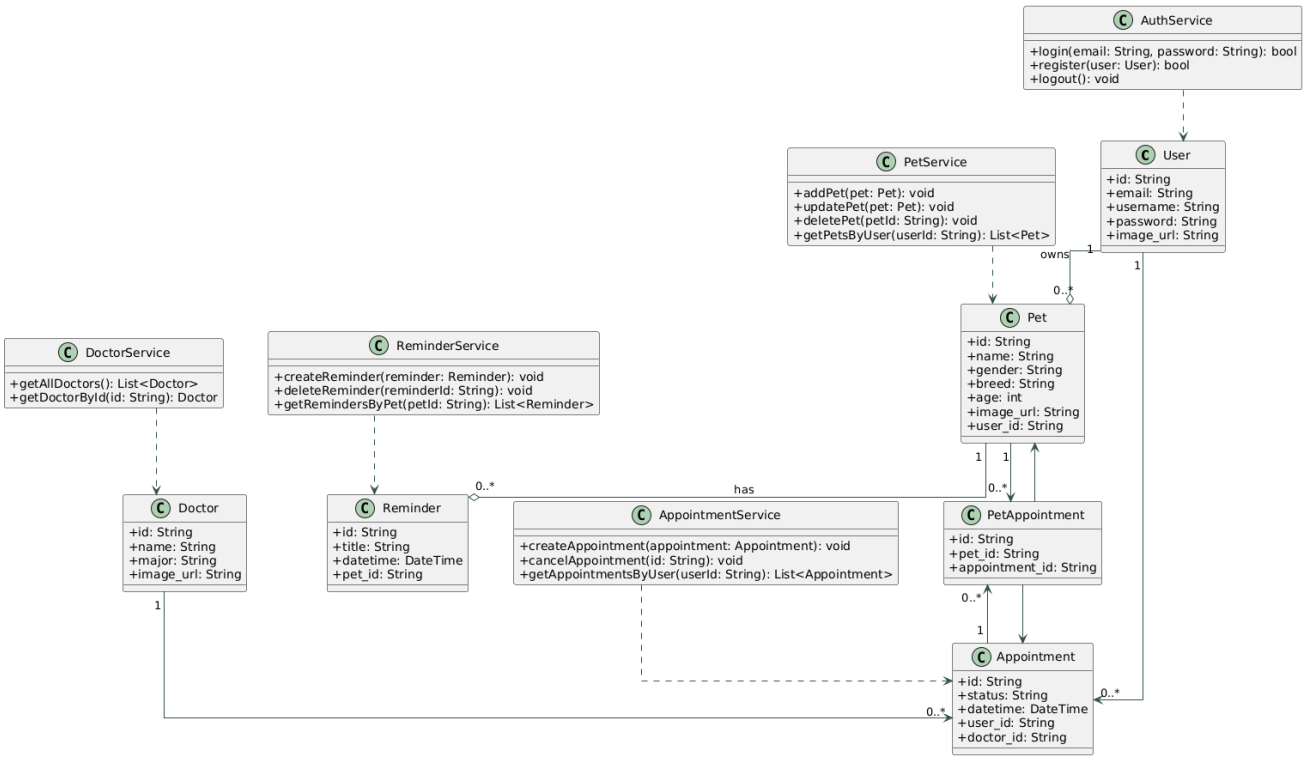


Рисунок 1.6 – Діаграма класів

1.4 Висновки дослідження предметної області та постановки задачі.

У даному розділі кваліфікаційної роботи, для початку було проведено аналіз предметної області, де було визначено, що мобільний застосунок для догляду за домашніми тваринами є актуальним рішенням, яке може значно спростити процес організації ветеринарного обслуговування, нагадувань про вакцинацію, збереження медичних записів тощо.

Аналіз ринку мобільних операційних систем показав, що найбільш доцільною є розробка застосунку під Android та iOS, оскільки вони займають понад 99% світового ринку мобільних пристроїв. Це забезпечує максимальну доступність для користувачів. Для забезпечення кросплатфомності було обрано гібридний підхід, що дозволяє підтримувати кілька операційних систем із мінімальними витратами на розробку та підтримку. Це рішення також дозволяє скоротити час розробки та знижує витрати на подальші оновлення.

Проведено аналіз конкурентних застосунків, таких як 11Pets, AnimalID, DogCatApp, що дозволило визначити основні функції, які повинні бути реалізовані в новому застосунку, та виявити слабкі сторони конкурентів. Це включає покращення інтерфейсу, інтеграцію з іншими системами та можливість персоналізації для користувачів. Визначено, що важливою є зручність використання та ефективне планування догляду за тваринами.

Далі були визначені функціональні та нефункціональні вимоги. Функціональні вимоги включають реєстрацію, ведення профілю тварини, нагадування про вакцинацію, медичні записи та планування прийомів. Нефункціональні вимоги стосуються безпеки, швидкості та зручності використання. Для відображення основних сценаріїв взаємодії користувача з системою було створено UML-діаграму варіантів використання, а також розроблені послідовності та класові діаграми. На основі цього було складене технічне завдання, яке міститься у додатку А.

2 ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

2.1 Проєктування архітектури та структури системи

Перед розробкою мобільного застосунку, спочатку потрібно вирішити, який саме тип архітектури буде мати застосунок.

Архітектура програмного забезпечення [18] – це розроблена структура застосунку, яка включає визначення взаємодії компонентів інтерфейсу з внутрішніми процесами системи. Іншими словами, це такий своєрідний підхід, який визначає які функції за що будуть відповідати та як вони можуть взаємодіяти один з одним.

Вибір архітектури є важливим етапом, адже саме тут вирішується який з типів архітектури найбільш доцільний в конкретному мобільному застосунку. Серед існуючих та найпопулярніших архітектур можна виділити [19-20]:

– монолітна архітектура: вона передбачає, що всі компоненти системи об’єднуються в єдину кодову базу, тобто вони працюють один з одним як одне ціле. Зазвичай, такий підхід спрощує розробку застосунку, його тестування та врешті-решт розгортання, адже вся логіка знаходиться в одному місці, що сильно полегшує управління. Але, у міру зростання проєкту ця архітектура стає складною для підтримки, оскільки будь-які зміни в тій чи іншій частині можуть впливати на всю систему;

– клієнт-серверна архітектура: цей підхід розділяє програмне забезпечення на дві основні частини: клієнтську, яка відповідає за інтерфейс користувача (UI) та взаємодію з ним, і серверну, яка обробляє всю бізнес-логіку, а також керує базами даних та відповідає на запити клієнта. У результаті, це дозволяє легко та ефективно розподіляти навантаження на застосунок, при цьому забезпечувати доступ до даних із різних пристроїв і централізовано керувати ними. Такий підхід є основою для багатьох веб-застосунків та мобільних систем, дозволяючи легко оновлювати серверну частину без необхідності змін у клієнтській частині програми;

– мікросервісна архітектура: ця архітектура складається з незалежних один від одного невеликих сервісів, кожен з яких виконує конкретну функцію та взаємодіє з іншими сервісами, зазвичай через API запити. Це дає змогу легко оновлювати окремі частини застосунку без впливу на всю логіку, що підвищує її надійність та масштабованість. Однак керування мікросервісами вимагає додаткових інструментів, адже управління взаємодією між сервісами та моніторинг роботи окремих сервісів ускладнює розробку та розгортання;

– MVC (Model-View-Controller) та MVVM (Model-View-ViewModel) [21]: ці два підходи, в більшості свому не являється архітектурами, їх більше доцільніше називати патернами, але в деяких застосунках ці патерни можна використовувати як повноцінну архітектуру. Вони організовують взаємодію між даними, бізнес-логікою та інтерфейсом користувача, спрощуючи підтримку й масштабованість застосунку. MVC розділяє програму на три основні частини: модель, що відповідає за дані та основну бізнес-логіку, представлення, яке відображає інформацію користувачеві, і контролер, що обробляє запити та координує роботу інших компонентів. MVVM, у свою чергу, додає ще один проміжний рівень ViewModel, який абстрагує бізнес-логіку від представлення, що особливо корисно для мобільних і десктопних додатків, де важливо мінімізувати безпосередній зв’язок між інтерфейсом та логікою;

– багаторівнева архітектура: вона забезпечує чітке розділення функціональних частин застосунку, що підвищує його модульність і полегшує підтримку. Зазвичай вона включає себе в кілька рівнів, серед яких презентаційний рівень, що відповідає за інтерфейс користувача, рівень бізнес-логіки, який містить основні функції для обробки даних, і останній – рівень доступу до даних, який працює з базою даних та іншими джерелами для отримання інформації. В загальному, такий підхід спрощує тестування, дозволяє незалежно змінювати або замінювати окремі рівні застосунку без впливу на всю систему і робить його зручним для корпоративних застосунків та великих інформаційних систем;

– функціональна архітектура: вона базується на функціональному програмуванні, де основними блоками є функції, що не міняють стан. Всі дані є незмінними, і операції над ними створюють нові структури, замість того щоб змінювати вже існуючі. Це підвищує стабільність програми. Основною перевагою є легкість тестування. Така архітектура також дуже добре масштабуються, оскільки функції можна обробляти паралельно одна одній. Однак вона може бути складною для впровадження та іноді має проблеми з продуктивністю через створення нових копій даних.

Як було зазначено, кожна з перелічених архітектур має свої переваги та недоліки і вибір підходящої архітектури залежить від багатьох факторів, таких як вимоги до продуктивності, гнучкість, масштабованість, витрати на розробку та підтримку застосунку в майбутньому.

Але, ці найпопулярніші архітектури не завжди варто використовувати. Наприклад зараз набирає популярності безсерверна архітектура.

Serverless-архітектура [22-23] – це такий підхід розробки, який дозволяє запускати застосунок або сервіси без необхідності керування фізичними або віртуальними серверами. Хоча це не означає, що цій архітектурі не має серверів. Їхнє адміністрування, масштабування та управління повністю передаються хмарному провайдеру, такому як AWS, Google Cloud або Microsoft Azure. Це дозволяє розробникам максимально зосередитися на бізнес-логіці та функціональності застосунку, а не на підтримці серверної частини.

Основою Serverless-архітектури є функції як сервіс (FaaS) [24]. У цьому підході застосунок складається з невеликих автономних функцій, які виконуються у відповідь на певні події, такі як HTTP-запити, зміни в базі даних або завантаження файлів. Функції запускаються автоматично, виконуються лише тоді, коли це потрібно, і зупиняються після завершення обробки, що робить цей підхід високоефективним у плані використання ресурсів.

Крім FaaS, Serverless-архітектура також використовує сервіси, які беруть на себе відповідальність за збереження даних, тобто виступають в ролі баз

даних, допомагають в управлінні чергами повідомлень, аутентифікацію користувачів, логування та інші задачі. Наприклад, у хмарних провайдерів є такі сервіси, як AWS DynamoDB (безсерверна база даних), Google Firestore (NoSQL-сховище), AWS S3 (хмарне сховище для файлів) та інші рішення, які допомагають будувати повноцінні системи без необхідності керувати інфраструктурою вручну.

Такий підхід має низку переваг. Він забезпечує автоматичне масштабування, оскільки хмарна платформа динамічно збільшує або зменшує ресурси залежно від навантаження на застосунок. Це дозволяє ефективно використовувати ресурси та оптимізувати витрати, оскільки компанії платять лише за фактичний час виконання функцій, а не за постійно запущені сервери. Також Serverless-архітектура значно прискорює розробку та розгортання застосунків, оскільки відсутня необхідність у налаштуванні серверів та управлінні ними.

Проте безсерверна архітектура має і свої певні недоліки. Вона підходить не для всіх типів застосунків, особливо тих, що вимагають постійної роботи процесів або низької затримки у виконанні. Функції FaaS мають обмежений час виконання (зазвичай від кількох секунд до 5 хвилин), що може створювати труднощі для довготривалих обчислень. Крім того, оскільки код виконується в середовищі, повністю контрольованому хмарним провайдером, виникає проблема так званого vendor lock-in – залежності від конкретного постачальника послуг. Тобто у тому випадку, якщо по якійсь причині, хмарний провайдер вийде з ладу або змінить умови використання, розробникам доведеться або повністю адаптувати свій код під нового постачальника, або переписувати значну частину коду. Як результат, це може спричинити значні витрати часу, ресурсів і коштів.

Але подібні ситуації виникають дуже рідко. Хмарний сервіс має резервні сервери, що зменшує шанс на відключення, а також постачальник завжди попереджає своїх користувачів заздалегідь про зміну тарифів чи інших послуг.

Ще одним викликом є складність у налагодженні та моніторингу, оскільки функції працюють у розподіленому середовищі, що ускладнює їхній аналіз та діагностику помилок.

Але, не зважаючи на це все Serverless-архітектура стає все більш популярною завдяки своїй ефективності, простоті розгортання та низьким витратам. Вона широко використовується у створенні API, мобільних бекендів, чат-ботів, обробці подій у реальному часі, автоматизації бізнес-процесів та навіть у машинному навчанні.

На рисунку 2.1 зображено безсерверну архітектуру.

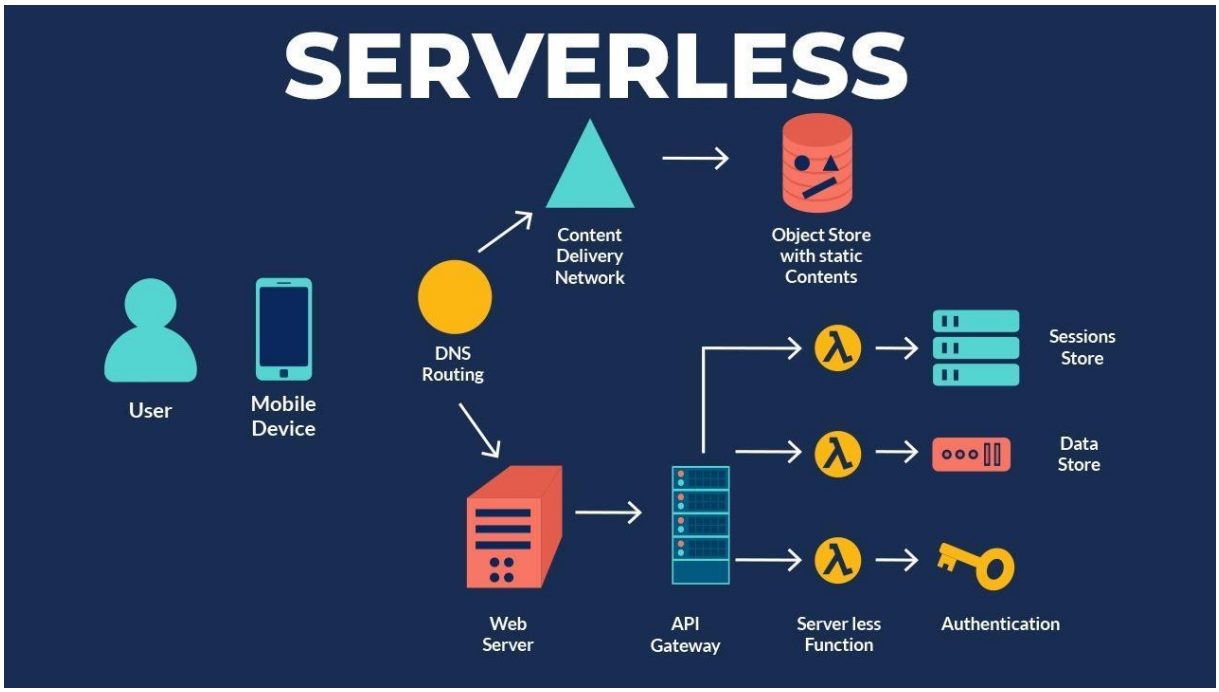


Рисунок 2.1 – Структура безсерверної архітектури.

В даному контексті варто ще згадати принцип «чистої архітектури».

Clean Architecture [37] – це підхід до організації коду всередині самої програми. Його основна ідея полягає у відокремленні логіки програми від деталей реалізації, таких як, наприклад, база даних, API, фреймворки чи інші системи що можуть надавати інформацію. Основна суть підходу – це створення зрозумілої та чіткої структури коду, де всі залежності спрямовані в бік логіки, що робить програму більш незалежною від зовнішніх факторів, які можуть

вплинути на роботу програми. Завдяки цьому, застосунок стає гнучкішим та маштабованішим. Тобто, якщо буде ситуація коли потрібно змінити базу даних або зовнішній сервіс з даними, це можна зробити без переписування основної логіки застосунку, оскільки логіка залишається ізольованою від реалізації. На рисунку 2.2 зображена діаграма чистої архітектури.

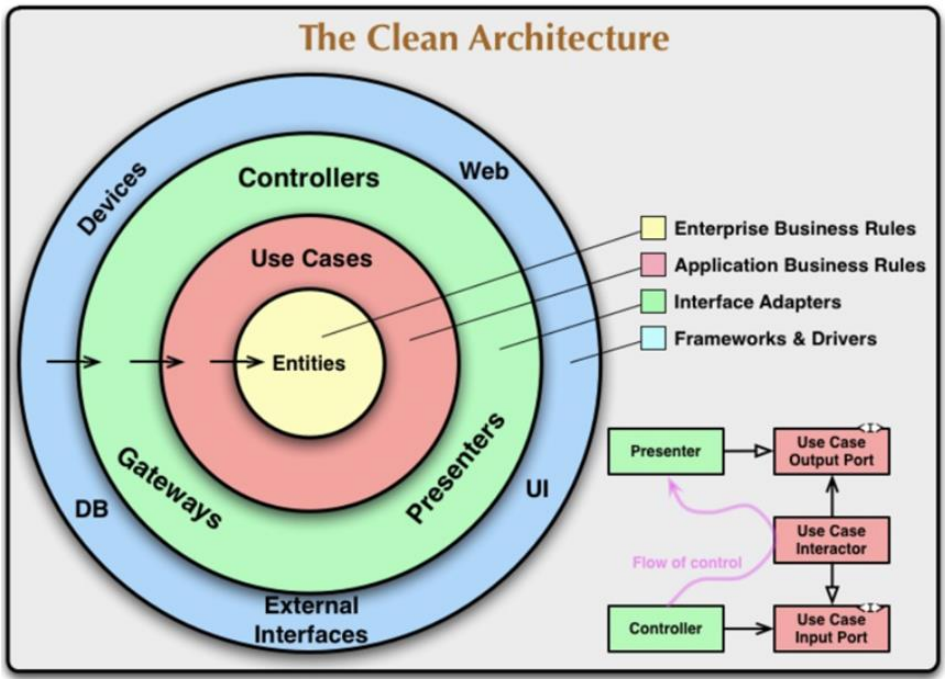


Рисунок 2.2 – Clean Architecture

Тут важливо розуміти одну річ, хоть і було обрано Serverless-архітектуру, але вона має певний зв’язок з Clean Architecture саме контексті структурування коду та ізоляції компонентів.

Clean Architecture визначає, як організувати код у застосунку, незалежно від системи, а Serverless-архітектура визначає, де і як цей код повинен виконуватися. Наприклад, у даному випадку можна побудувати застосунок із Clean Architecture і розгорнути його в serverless-середовищі – у такому випадку бізнес-логіка та структура коду залишаться чистими, а виконання відбуватиметься у безсерверному середовищі.

Окрім самої архітектури, ще варто згадати таку річ як управління станом(state management) [25].

Управління станом – це контролю та організації даних у застосунку, що забезпечує його коректну роботу. Стан включає всю інформацію, яка впливає на вигляд або поведінку інтерфейсу, наприклад, змінні значення, отримані дані або дії користувача. У звичайних випадках стан змінюється лише всередині окремих компонентів, але в більш складних застосунках його потрібно передавати між різними частинами. Це вимагає структурованого підходу, який допомагає уникнути хаосу, полегшує оновлення даних і підтримку коду. Слід зауважити, що це само по собі не є архітектурою, але його можна вважати архітектурним підходом, який тісно пов'язаний з архітектурою та працює в парі з нею.

Технологій які допомагають в управлінні станом застосунку є доволі багато, але для кросплатформених застосунків найпопулярнішими та найзручнішими є Riverpod та BLoC.

Riverpod [26] – це підхід для управління станом. Основна ідея технології полягає в тому, що вона для роботи використовує провайдери. Провайдери [27] – це такі спеціальні об'єкти, які визначають, як повинен створюватися та зберігатися стан. Завдяки цьому автоматично вивільняється пам'ять, зручніше вести спостереження за змінами у застосунку та тестувати його. Riverpod дозволяє працювати як із простими, так і зі складними об'єктами, включаючи асинхронні запити. Він підходить для керування глобальним і локальним станом, що робить його корисним інструментом для різних типів застосунків.

На рисунку 2.3 зображено архітектурний підхід «Riverpod»

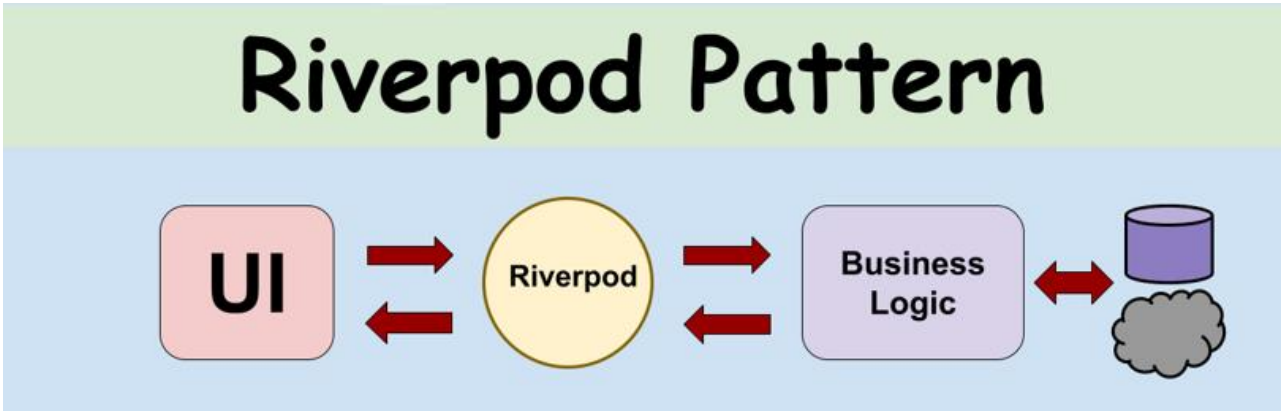


Рисунок 2.3 – Архітектурний підхід Riverpod

BLoC (Business Logic Component) [28] - це підхід для управління станом, який чітко розділяє бізнес-логіку та інтерфейс користувача. Він працює на потоках і реактивному підході, що дозволяє керувати станом через події та відповідні зміни стану.

Основними елементами цього підходу є події, стани, клас BLoC і потоки. Подія (Event) це така дія, яка надсилається у BLoC для запуску певної логіки, наприклад, натискання певної кнопки, введення тексту або отримання відповіді з сервера. Стан (State) в свою чергу відображає поточний статус застосунку або окремого екрану та може змінюватися у відповідь на певну подію. Клас BLoC приймає події, виконує бізнес-логіку та повертає новий стан. Потік (Stream) використовуються для надсилання нових станів до інтерфейсу, що дозволяє оновлювати UI залежно від поточного стану. Можна вважати, що сама концепція цього підходу будується на потоках

Цей підхід робить застосунки більш контрольованими та тестованими, оскільки дозволяє чітко визначати, які стани можливі та як вони можуть змінитися. BLoC особливо добре себе показує у великих застосунках з багатьма рівнями взаємодії, де необхідно забезпечити керованість і масштабування.

На рисунку 2.4 зображено архітектурний підхід «BLoC»

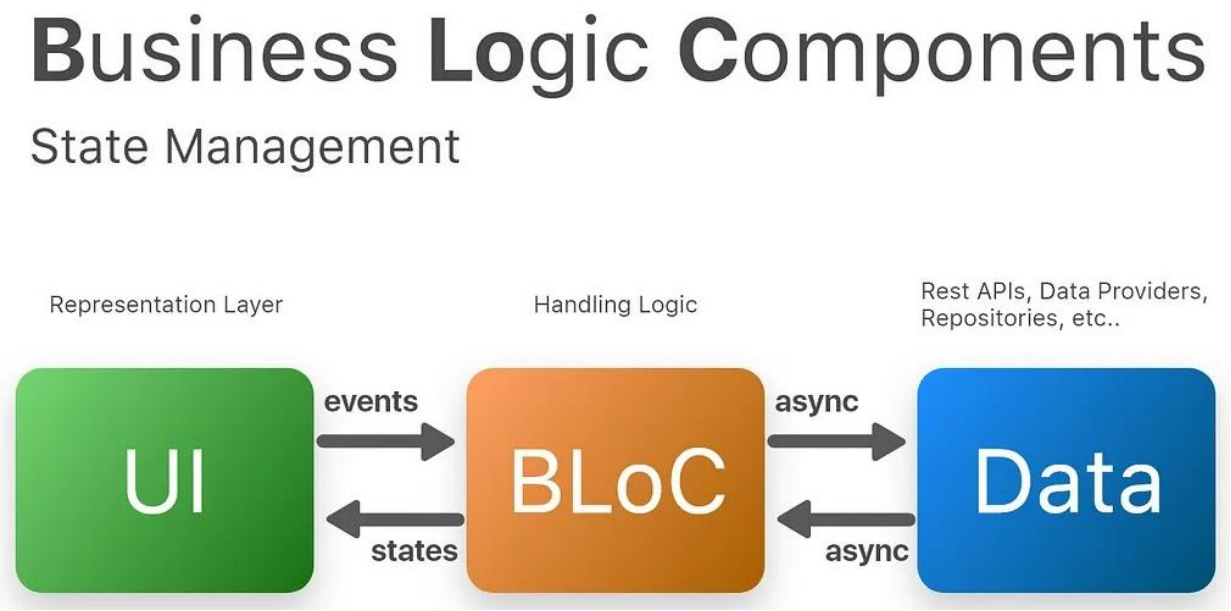


Рисунок 2.4 – Архітектурний підхід BLoC

2.2 Проєктування структури даних та логічної моделі бази даних

Логічна модель бази даних [29] – це представлення структури даних, яке показує, як організовані та взаємопов’язані дані в межах певної бази даних. Ця модель описує структуру даних, їхні властивості та правила взаємодії між сутностями. Важливим нюансом, є те що ця модель даних не має прив’язки до конкретної системи керуваня базами даних (СКБД), тобто вона підходить як і для SQL так і NoSQL баз даних. Ця модель допомагає зрозуміти, як дані будуть зберігатися та оброблятися, а також визначає зв’язки між сутностями, забезпечуючи цілісність і узгодженість інформації.

Основними елементами логічної моделі є сутності, атрибути та зв’язки. Сутність це такий об’єкт, про який потрібно зберігати дані (наприклад, користувач, домашній улюбленець, запис до ветеринара). Кожна сутність має свій набір властивостей, які називаються атрибутами (наприклад, для сутності «Користувач» атрибутами можуть бути ім’я, електронна пошта, номер мобільного телефону).

Ще дуже важливим поняттям є первинний ключ. Це такий унікальний ідентифікатор кожної сутності, який дозволяє однозначно ідентифікувати кожен запис у таблиці. Окрім первинного ключа є ще зовнішній ключ. Це атрибут, який встановлює зв’язок між сутностями, посилаючись на первинний ключ іншої сутності.

Зв’язки між сутностями оскільки вони визначають, як об'єкти взаємодіють між собою. Є три основні типи зв'язків:

– один до одного (1:1): цей той випадок, коли одному запису в одній таблиці відповідає тільки один запис у другій таблиці. Наприклад, один користувач може мати тільки один профіль.

– один до багатьох (1:N): коли одному запису в першій таблиці відповідає два або більше записів у другій таблиці. Наприклад, один користувач може мати домашніх улюбленців.

– багато до багатьох (N:N): ситуація, коли одному запису в першій таблиці відповідає кілька записів у другій таблиці, і навпаки. Наприклад, один запис до ветеринара може містити декілька тварин, і кожна тварина може мати декілька записів до ветеринара. Зазвичай для реалізації таких зв'язків створюється додаткова таблиця, яка містить зовнішні ключі для обох сутностей.

У загальному, така модель також визначає обмеження, які забезпечують правильність, узгодженість та надійність даних у базі. Одним із ключових аспектів є цілісність сутностей, яка гарантує, що кожен запис має унікальний ідентифікатор – первинний ключ (PK). Завдяки цьому можна однозначно ідентифікувати будь-який об'єкт або запис у системі, уникнути дублювання інформації та забезпечити швидкий доступ до конкретних даних.

Іншим важливим аспектом є цілісність посилань. Вона забезпечує, щоб значення зовнішніх ключів (FK) відповідали наявним первинним ключам у пов'язаних сутностях. Наприклад, запис у таблиці «Домашні улюбленці» (Pet), що містить зовнішній ключ до користувача, можливий лише за умови, що відповідний користувач уже існує в таблиці «Користувачі» (User).

На рисунку 2.5 зображено логічну модель «сутність зв'язок» (ER) мобільного застосунку.

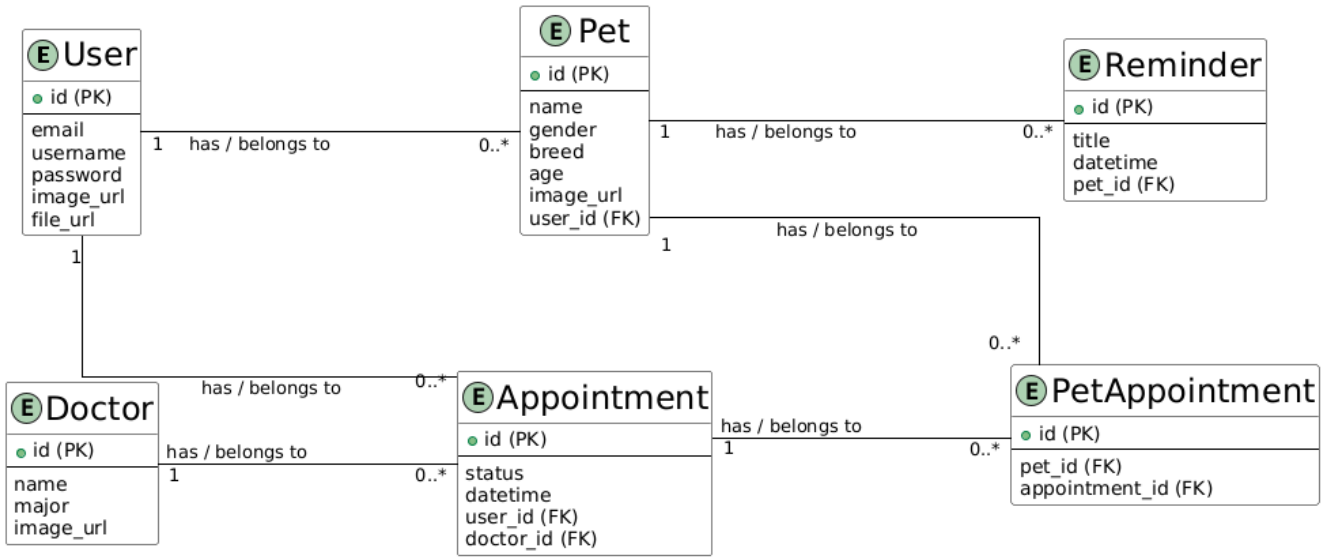


Рисунок 2.5 – ER-діаграма

2.3 Проєктування інтерфейсу користувача

Перед початком створення дизайну застосунку потрібно було для початку обрати основні кольори. У нашому випадку буде підтримуватися одразу дві кольорові теми – чорна та біла, відповідно для кожної з тем потрібно обрати відповідні кольори. Для білої теми це темно-блакитний, чорний, сірий та білий, для чорної – персиковий, сірий, білий, чорний.

Після того як було обрано кольори, було створено інтерфейс для перших двох сторінок застосунку, а саме сторінка «Реєстрація» та сторінка «Логін». Сторінки мають зверху логотип застосунку, далі ідуть декілька відповідних полей вводу, кнопку дії і окрему іконку для авторизації через gmail. Інтерфейс можна побачити на рисунку 2.6.

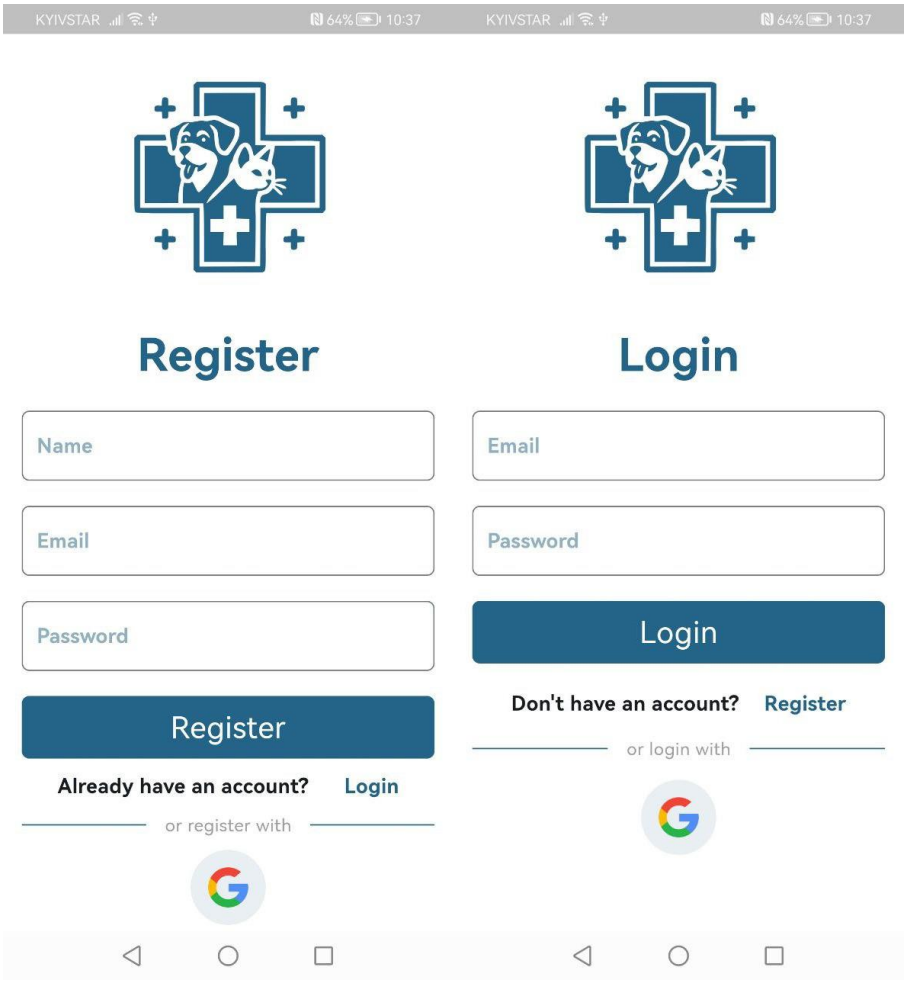


Рисунок 2.6 – Сторінки авторизації

Пройшовши авторизацію, користувач потрапляє на початкову сторінку застосунку, де також може бачити як саме проводиться навігація по застосунку – через нижній бар. Основні сторінки зображені на рисунку 2.7.

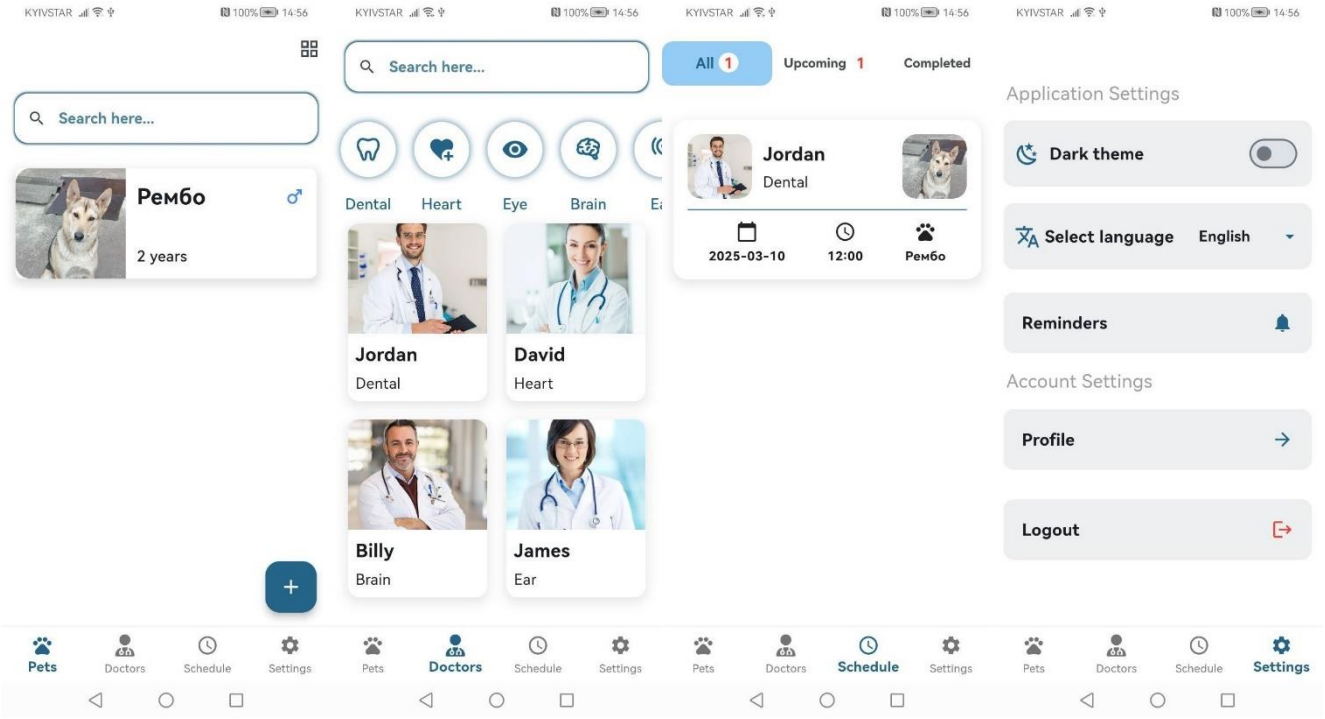


Рисунок 2.7 – Основні сторінки застосунку

З рисунку 2.7 видно, що у нас є чотири основних сторінки:

– Pets (Тварини);

– Doctors (Доктори);

– Schedule (Розклад);

– Settings (Налаштування).

Сторінка «Pets», у тому випадку якщо користувач додав у застосунок тварину, буде відображати список тварин. Окрім цього є ще поле пошуку, де якщо ввівши ім’я користувач бистро знайде потрібну йому тварину. Окрім цього є кнопка, яка відкриває спеціальний інтерфейс для додавання тварин. Також, для кращого досвіду користувача була добавлена кнопка, яке перемикає відображення карточок тварин, доступно два види відображень: списком та стовпцем. У тому ж випадку, якщо у користувача немає ніяких доданих

домашнніх улюбленців він буде бачити відповідний надпис по центру сторінки, та зможе додати улюбленця будь-коли.

Наступна сторінка, це сторінка «Doctors». На сторінці просто відображений список ветеринарів. З допомогою поля пошуку та списку категорій можна без проблем найти потрібного вет-лікаря. Пошук можна виконувати як за іменем, так і за спеціальністю.

Сторінка «Schedule» відображає записи користувача у вигляді карточок, де є уся потрібна іфнформація про запис до лікаря. Записи в свою чергу можна відобразити як за станом «Майбутні» та «Виконані» так і вивести просто усі записи без прив’яхки до конкретного стану запису.

Сторінка «Settings» містить просто налаштування застосунку, а саме тему застосунку,мову, налаштування нагадувань. Є можливість перейти у профіль користувача та в раці чого вийти з опублікового запису.

Наступні сторінки це сторінка профілю та його зміни. Інтерфейс сторінок зображено на рисунку 2.8.

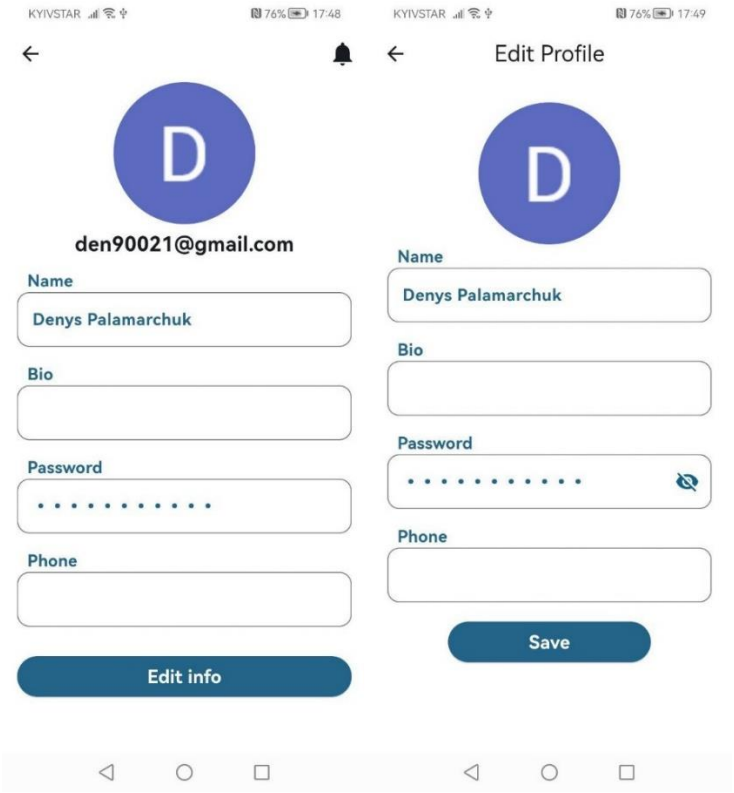


Рисунок 2.8 – Сторінки профілю та його зміни

На рисунку 2.8 видно, що ці дві сторінки мають подібний дизайн. На першій сторінці ми просто бачимо інформацію про користувача, тобто його фото, електрону почту, ім’я, пароль (який на даному етапі переглянути не можна), опис профілю та номер телефону, але цю інформацію користувач може і не вносити. Натиснувши на кнопку «Edit info», ми потрапляємо на наступну сторінку – зміни профілю. Візуально вона подібна до минулої, але різниця є. На відміну від минулої сторінки тут все клікабельне, тобто наприклад натиснувши на фото профілю відкриється галерея, де користувач зможе обрати нове фото для профілю. Аналогічно і поля вводу стали клікабельними, користувач в будь-який момент може змінити інформацію про себе (ім’я, пароль, опис, телефон). Єдине, що користувач не може змінити це електрону пошту, адже електрона пошта є логіном для входу.

На рисунку 2.9 зображені основні сторінки для роботи з домашнім улюбленцем.

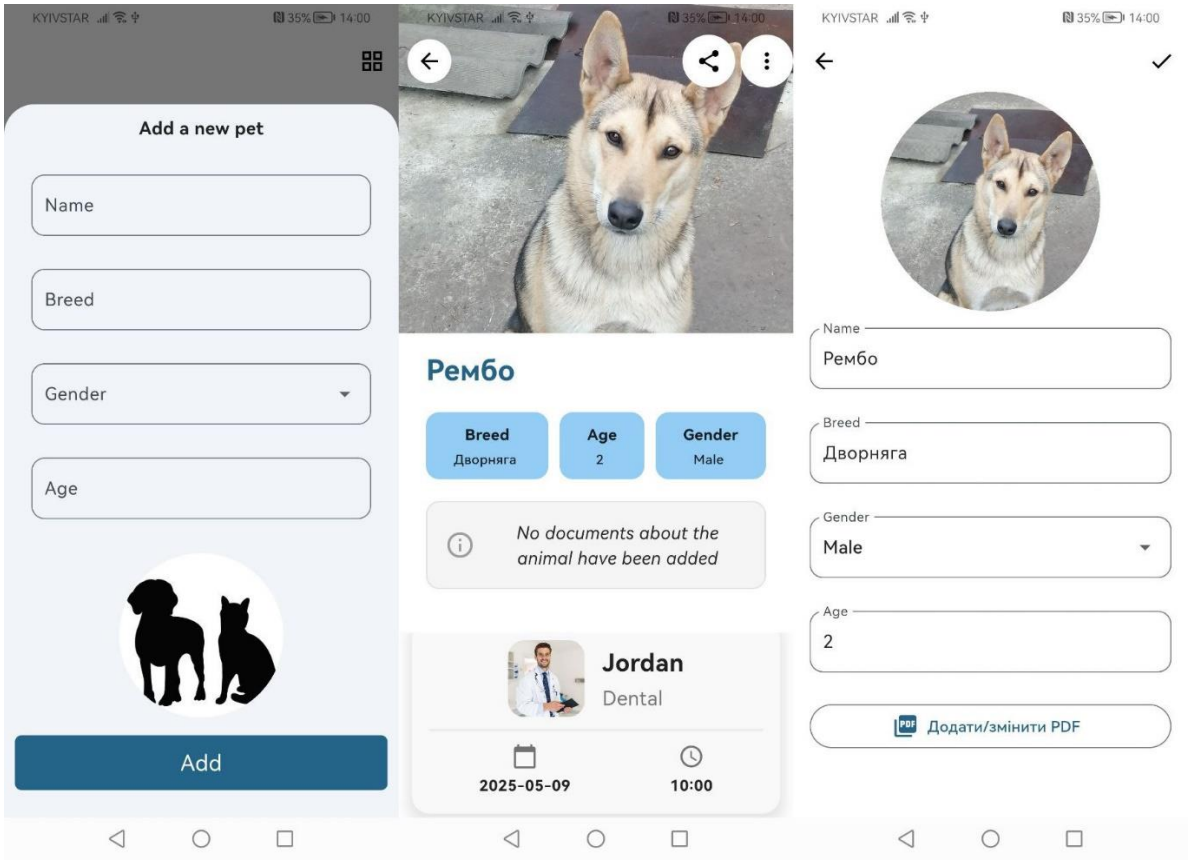


Рисунок 2.9 – Сторінки для роботи з твариною

Як можна бачити з рисунку 2.9 все починається з додавання тварини. Для того, щоб мати можливість додати тварину, потрібно на сторінці «Pets» нажати на кнопку з іконкою «+». Після чого відкриється BottomSheet (такий віджет який з’являється знизу екрана поверх поточного вмісту). Після цього можна побачити чотири клікабельних поля вводу, клікабельне фото та кнопку з надписом «Add». Для створення тварини достатньо лише вести ім’я улюбленця та натиснути кнопку додавання.

Наступна сторінка – це сторінка інформації про тварину. На сторінці є верхній бар з трьома кнопками:

– кнопка повернення на попередню сторінку;

– кнопка поділитися твариною в соцмережах;

– кнопка яка відкриває «попап меню», де можна перейти на сторінку з оновленням інформації про тварину, додати нагадування, або видалити її.

Під верхнім баром зображено фото тварини, а нижче базова інформація, а саме ім’я тварини, породу, вік та стать. В кінці сторінки знаходиться документи і список записів у ветеринара конкретно для цієї тварини.

Сторінка зміни даних в свою чергу містить клікабельне фото та чотири поля вводу, де користувач може змінити зображення або текстову інформацію про тварину та зберегти, натиснувши на кнопку у верхньому правому кутку.

Останніми сторінками для розгляду буде сторінка для створення запису до ветеринара та сторінка для створення нагадування. Їх інтерфейс можна побачити на рисунку 2.10.

Щоб перейти на сторінку для створення запису вам спочатку потрібно обрати лікаря, після чого користувач потрапляє на сторінку яка містить календар, де користувач може обрати дату, а після чого і час запису. Після цього у користувача з’являється можливість обрати потрібну тварину. Як користувач обере дату, час та улюбленця тільки тоді він зможе створити запис.

Сторінка для створення нагадувань має подібний інтерфейс, користувачу також потрібно обрати дату та час, а також вести надпис для нагадування.

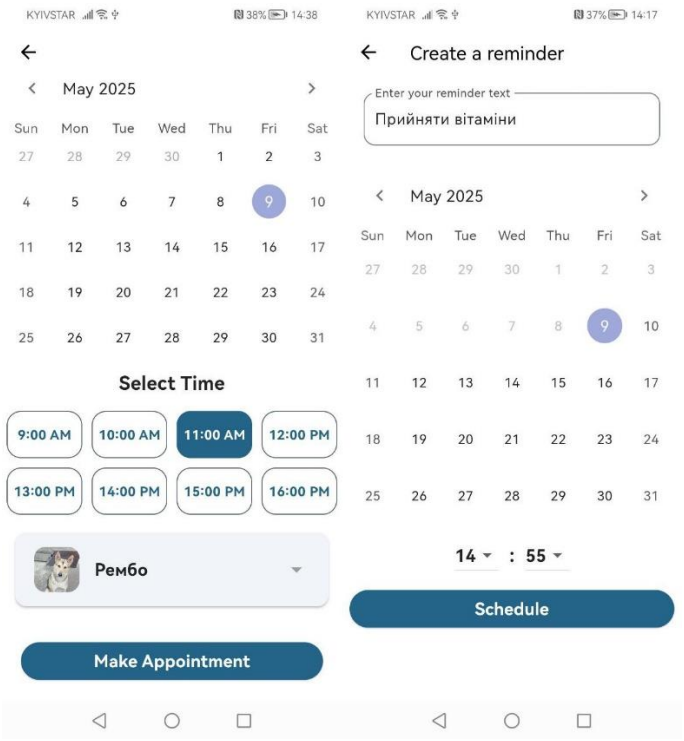


Рисунок 2.10 – Сторінки створення запису та нагадування

На рисунку 2.11 зображено усі сторінки застосунку, та як вони пов’язані один з одним.

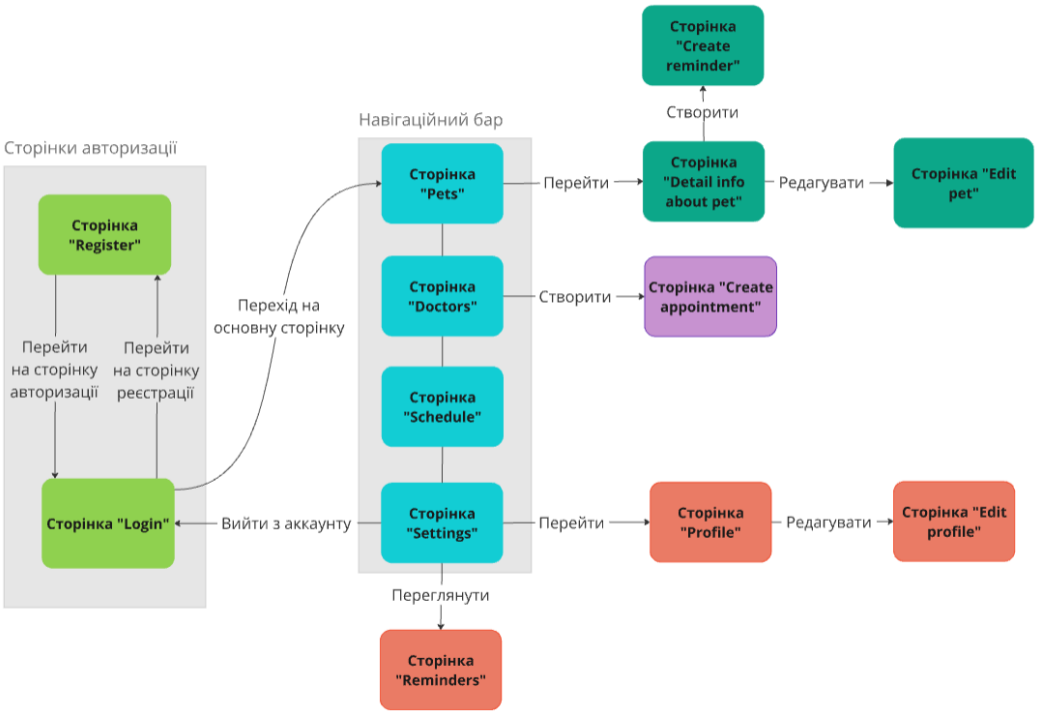


Рисунок 2.11 – Усі сторінки у застосунку

2.4 Аналіз та вибір технологій і методів реалізації застосунку

Після аналізу предметної області, було вирішено саме гібридний тип застосунку, адже потрібно розробити мобільний застосунок під дві найпопулярніші операційні системи для смартфонів – Android і IOS. Для цього було обрано фреймворк Flutter, розроблений на мові програмування Dart.

Dart [30] – це мова програмування, створена компанією Google. Вона створювалась як аналог JavaScript, адже в той час ця мова набирала популярності. Dart добре оптимізований та використовується переважно для створення мобільних, веб і серверних застосунків. Також ця мова підтримує об’єктно-орієнтоване програмування, та має статичну типізацію. Починаючи з версії 2.12 мова програмування стала null safety, допомагаючи уникнути помилок з null. Зараз мова програмування переважно використовується у фреймворку Flutter.

Flutter [31] – це фреймворк від компанії Google, який дозволяє створювати мобільні, веб та десктопні застосунки з єдиною кодовою базою. Він використовує мову програмування Dart як основну та має власний графічний движок, що забезпечує швидку та плавну роботу інтерфейсу на різних операційних системах та з різними розмірами екранами. Однією з головних переваг Flutter є його система віджетів.

Віджет (widget) [32] – це основний об’єкт інтерфейсу у Flutter. Тобто у фреймворку абсолютно усе, що бачить користувач є віджетом (кнопки, текст, поля вводу, зображення). Вони дозволяють створювати гнучкий і кастомізований дизайн без використання нативних компонентів певної операційної системи (але також можна використовувати й нативні компоненти для кращого досвіду користувача).

Цей фреймворк активно використовується багатьма компаніями для створення комерційних застосунків, оскільки він дозволяє зекономити ресурси за рахунок своєї кросплатформності. Проте через свою архітектуру він має і

певні недоліки, наприклад, збільшений розмір застосунку та необхідність взаємодії з нативним кодом для деяких функцій.

Щодо бази даних, то було обрано Firebase.

Firebase [33] – це платформа від компанії Google, яка надає набір інструментів для створення, управління та масштабування веб і мобільних застосунків без необхідності налаштовувати серверну частину. Вона включає в себе хмарну базу даних, автентифікацію, зберігання файлів(будь-якого формату), аналітику для користувачів, push-сповіщення, веб-хостинг застосунку і багато інших корисних функцій, що дозволяють розробникам зосередитися на логіці застосунку, а не на інфраструктурі.

Одним із головних компонентів Firebase є Cloud Firestore – NoSQL база даних, що підтримує зберігання та синхронізацію даних у реальному часі. Також є Realtime Database, яка працює за схожим принципом, але більше підходить для простих задач. Для автентифікації Firebase надає готові рішення для входу через Google, Facebook, Apple, електронну пошту, телефонний номер.

Ще одна функція – Cloud Storage [34], яка дозволяє зберігати файли, відео зображення, інтегруючись з системою доступу Firebase Authentication. Також є Cloud Functions, які дозволяють виконувати серверний код у відповідь на певні події, що спрощує роботу з бекенд-логікою. Firebase ідеально підходить для мобільних застосунків, особливо ті, які розробляються на Flutter, оскільки існують офіційні пакети для простої інтеграції сервісу в застосунок. Це дозволяє швидко запускати проєкти та масштабувати у майбутньому.

Щодо бібліотек, то фреймворк підтримує велику кількість бібліотек, які значно розширюють функціональність застосунків та спрощують розробку. Ці бібліотеки охоплюють різні аспекти створення мобільного інтерфейсу – від UI-компонентів і анімацій до роботи з базами даних, API, хмарними сервісами, геолокацією та інтеграцією з платформними функціями Android і iOS. Це пришвидшує розробку, покращує якість коду та дозволяє зосередитись на бізнес-логіці замість технічних дрібниць. Серед бібліотек можна виділити:

– бібліотека flutter_svg: використовується для відображення SVG-графіки у Flutter-застосунках. Вона дозволяє працювати з векторними зображеннями, які не втрачають якість при масштабуванні, що особливо зручно використовувати для іконок і логотипів.

– бібліотека font_awesome_flutter: використовується для доступу до великого набору іконок з колекції Font Awesome. Це зручно для створення інтуїтивно зрозумілого інтерфейсу з використанням популярних і візуально впізнаваних іконок.

– бібліотека material_design_icons_flutter: використовується для інтеграції розширеного набору Material Design іконок, який виходить за межі стандартного пакета Flutter. Вона дозволяє гнучко оформлювати інтерфейс згідно з принципами Material Design.

– бібліотека table_calendar: використовується для створення повнофункціонального календарного віджета з підтримкою подій, вибору діапазону дат та кастомного оформлення. Підходить для застосунків, де потрібна інтерактивна взаємодія з датами — наприклад, планувальників.

– бібліотека cupertino_icons: використовується для доступу до іконок у стилі iOS. Вона дозволяє зберігати єдиний візуальний стиль застосунку на iOS-платформі, що особливо важливо для кросплатформенних інтерфейсів.

– бібліотека flutter_bloc: використовується для впровадження архітектурного шаблону BLoC (Business Logic Component) у Flutter. Вона забезпечує розділення бізнес-логіки та інтерфейсу, спрощує тестування і робить код більш підтримуваним.

– бібліотека bloc: використовується як ядро для реалізації BLoC-підходу незалежно від Flutter. Вона містить механізми для керування потоками станів і подій, що дозволяє будувати чисту архітектуру навіть у Dart-консолях або серверних застосунках.

– бібліотека timezone: використовується для точної роботи з часовими поясами. Вона дозволяє враховувати локальні часові зони при обчисленнях, що

особливо важливо для глобальних застосунків із користувачами з різних світових регіонів.

– бібліотека intl: використовується для локалізації інтерфейсу користувача, а також для форматування дат, чисел і тексту згідно з мовними стандартами. Це ключовий інструмент для створення багатомовних застосунків.

– бібліотека shared_preferences: використовується для зберігання простих даних (наприклад, налаштувань, токенів) у вигляді пар ключ-значення. Це просте та легке рішення для збереження локального стану без використання повноцінної бази даних.

– бібліотека sqflite: використовується для локального зберігання структурованих даних у SQLite-базі. Вона надає можливості для створення таблиць, виконання запитів та збереження даних офлайн.

– бібліотека path: використовується для роботи з файловими шляхами у файловій системі пристрою. Вона дозволяє безпечно з'єднувати, розбирати і змінювати шляхи до файлів і папок.

– бібліотека path_provider: використовується для доступу до системних директорій, таких як тимчасові файли або документи користувача. Вона необхідна для роботи з файлами, які потрібно зберігати на пристрої.

– бібліотека firebase_auth: використовується для реалізації аутентифікації користувачів через email, Google, Apple або інші провайдери. Вона є основою для забезпечення безпеки доступу до функціоналу застосунку.

– бібліотека cloud_firestore: використовується для зберігання та синхронізації даних у хмарній NoSQL базі Firebase. Вона забезпечує швидкий доступ до даних у реальному часі та допомагає автоматично оновити інтерфейс при їх зміні.

– бібліотека firebase_storage: використовується для завантаження, збереження і отримання файлів у Firebase Storage. Вона підходить для зберігання мультимедійних ресурсів, документів, зображень або будь-яких інших файлів у хмарі.

2.5 Висновки проєктування архітектури та структури програмного забезпечення

У цьому розділі кваліфікаційної роботи було проведено аналіз існуючих архітектурних рішень для мобільних застосунків. Визначено їх переваги та недоліки, а також оцінено ефективність кожної архітектури в конкретних умовах. На аналізу було обрано архітектуру, яка найкраще відповідає вимогам і функціональності мобільного застосунку.

Далі під час проєктування логічної моделі даних було визначено основні сутності, атрибути та визначено зв’язки між ними. Окрім цього, було визначено первині і зовнішні ключі для чіткого і ефективнного зв’язку між сутностями. У результаті було створено ER-діаграму візуалізації структури бази даних і взаємозв’язків між сутностями.

На етапі проєктування інтерфейсу було обрано кольори для двох тем – білої та чорної. Біла тема використовує світлі відтінки для створення чистого та мінімалістичного вигляду, тоді як чорна – темні тони для елегантності, з яскравими акцентами для кращої видимості. Після вибору кольорової палітри було детально описано інтерфейс кожної сторінки та її призначення, включаючи характеристики кнопок, полів введення, списків та інших елементів.

У результаті аналізу було обрано гібридний тип застосунку для роботи на Android та iOS. Для цього використано Flutter на мові Dart, що забезпечує високу продуктивність і кросплатформність завдяки системі віджетів. Як базу даних обрано Firebase, який надає інструменти для зберігання даних, автентифікації та синхронізації у реальному часі. Поєднання Flutter та Firebase дозволяє створити ефективний та масштабований застосунок зі зручним інтерфейсом і стабільною роботою.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Реалізація мобільного застосунку

У розділі 2 було вказано, що при розробці мобільних застосунків на фреймворку Flutter основними елементами є віджети. Застосунок повністю складається з віджетів. В свою чергу віджети бувають різними та кожен має своє призначення та функції. Серед основних віджетів [35], які використовують у розробці на Flutter можна виділити:

– віджети для компонування: Container, SizedBox, Padding, Align, Center;

– віджети для розташування елементів: Column, Row, Stack, Wrap, Expanded, Flexible;

– віджети для оформлення та стилю: Text, Icon, Image, Theme;

– інтерактивні віджети: GestureDetector, InkWell, ElevatedButton, TextButton, IconButton, Switch, Checkbox, Slider;

– віджети для списків та колекцій: ListView, GridView, ReorderableListView, CustomScrollView;

– віджети для діалогів та сповіщень: AlertDialog, SimpleDialog, SnackBar;

– віджети для навігації та маршрутів: Navigator, PageView, BottomNavigationBar, Drawer, AppBar;

– віджети для анімацій та переходів: AnimatedContainer, AnimatedOpacity, AnimatedPositioned, Hero.

Вище перелічені віджети лише невелика частина тих, які надає фреймворк. Загальна ж кількість налічує більше сотні, це ще не враховуючи десятки тисяч віджетів які зробили розробники у вигляді біблотек, і які в будь-який момент можна встановити собі у проєкт. До того ж завжди можна створити власний кастомний віджет з уже існуючих віджетів. Така можливість кастомізації віджетів дуже зручна, і дуже добре показує як саме організована їх структура, а зокрема їх ієрархію, Тобто, фактично віджети можна поділити на батьківські та дочірні.

Ще важливим фактором при розробці є те, що віджети поділяють на StatelessWidget та StatefulWidget.

StatelessWidget [36] – це віджет, який не змінюється після створення. Він створюється лише один раз і не оновлюється навіть в тому випадку, коли програма змінює свій стан. Якщо потрібно якось змінити віджет, то Flutter просто створить його заново. Цей віджет потрібно використувати в тому випадку, коли у застосунку є щось незмінне, наприклад, текст із назвою сторінки. Такі віджети потрібні в першу чергу для оптимізації, адже програма просто запише його у пам'ять та буде брати з кешу.

StatefulWidget [36] – це віджет, який може змінюватися під час виконання програми. Він має об'єкт стану (State), який дозволяє змінювати вигляд або поведінку віджета без перестворення. Тобто, коли стан змінився, Flutter перебудовує лише цей віджет, ігноруючи інші частини інтерфейсу. В загальному це дозволяє створювати динамічні елементи в застосунку. Для зміни стану використовується функція setState(). Вона у деяких віджетах вбудована одразу, але цю функцію також можна використати в будь-якій частині програми просто її викликавши. Порівняння цих двох віджетів знаходиться у таблиці 3.1

Таблиця 3.1 – Порівняння двох видів віджетів у Flutter

Параметр	StatelessWidget	StatefulWidget
Зміна стану	Не змінюється після створення	Може змінюватися під час виконання програми
Метод оновлення	Віджет будується один раз	Використовує setState() для оновлення
Приклад використання	Текст, іконки, кнопки без змін	Поля вводу, форми, анімації
Продуктивність	Швидший, оскільки не потребує оновлення стану	Може споживати більше ресурсів через оновлення стану

Після розгляду основних блоків, з яких складається Flutter-застосунок, доцільно також звернути увагу на його загальну структуру. Зазвичай для організації архітектури застосунку використовується підхід Clean Architecture, який був обраний в минулому підрозділі.

На рисунку 3.1 представлено структуру папок застосунку, сформовану відповідно до принципів чистої архітектури, яка сприяє кращій організації коду, його масштабованості, модульності та легкості в підтримці. Така структура дозволяє чітко розділити відповідальність між різними шарами застосунку (presentation, domain, data), що особливо важливо у великих проєктах.

У кореневій директорії проєкту розміщено окремі папки, які містять конфігураційні файли та налаштування для кожної з підтримуваних платформ: iOS, Android, Linux, Web та Windows. Кожна з цих папок містить специфічні параметри для компіляції, залежностей, ресурсів і налаштувань, які потрібні для коректної роботи застосунку на відповідній платформі. Наприклад, папка android/ включає Gradle-скрипти та AndroidManifest.xml, а ios/ — Xcode-проєкт і Plist-файли.

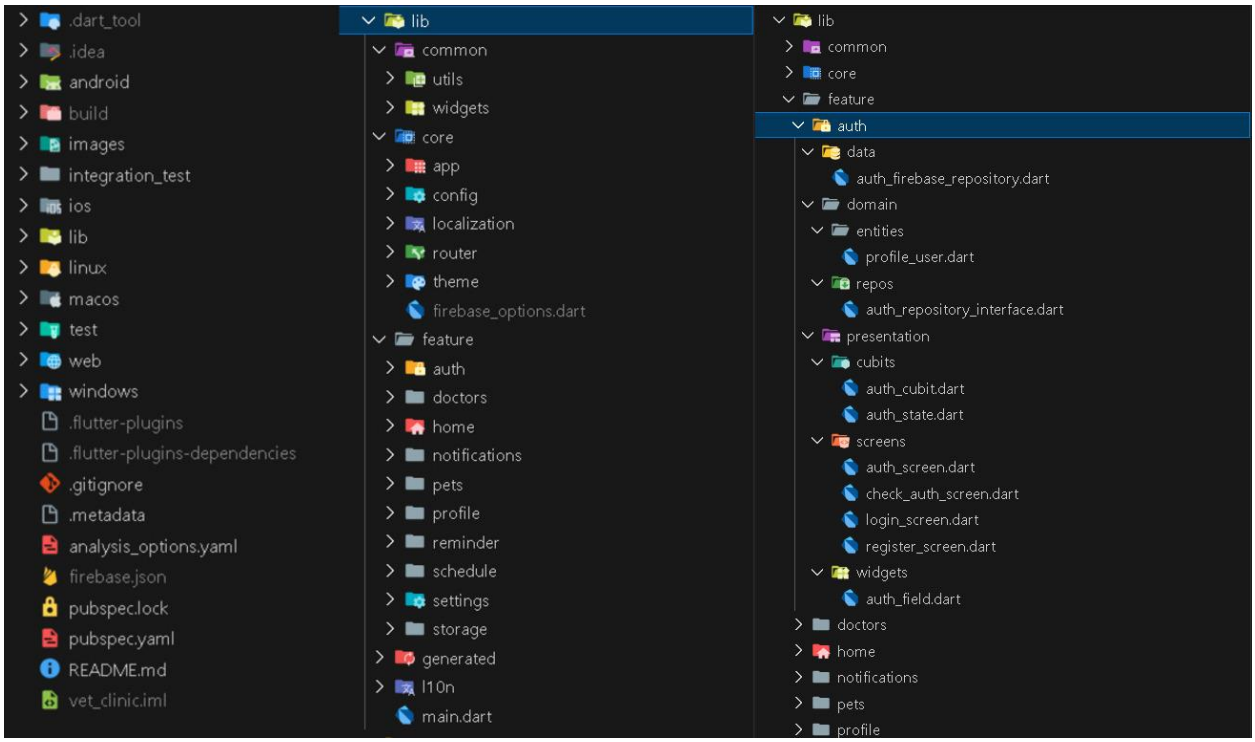


Рисунок 3.1 – Структура папок

Також там знаходяться і інші важливі файли для конфігурації застосунку, такі наприклад як pubspec.yaml, де міститься назва проєкту, його опис та версія, список залежностей(бібліотек), а також налаштування шрифтів та зображень.

Головною папкою у Flutter застосунках принято вважати lib, адже саме у цій папці буде находитися увесь код застосунку. В свою чергу, ця папка може мати дочірні папки, кожна з яких буде відповідати за якийсь функціонал чи налаштування. Наприклад у папці «common», зазвичай знаходяться кастомні віджети, які використовуються на усіх рівнях застосунку (індикатор завантаження чи спливаюче вікно з повідомленням) а також папка «utils», яка в свою чергу містить функції які використовуються у всьому застосунку, це наприклад функція переведення формату часу, або функція для вибору зображення з галереї телефону, тощо. Ну і відповідно інші папки мають свою зону відповідальності, так каталог «core» зазвичай містить папки з налаштуваннями застосунку, а також окремі папки для кольорів і теми застосунку, локалізації та навігації.

Але більше всього варто зробити акцент на папці «feature», адже саме ця папка містить функціональні модулі застосунку, як от наприклад «auth». В свою чергу модуль ділиться на 3 основні папки:

– папка «data»: тут знаходяться дані та їх джерела;

– папка «domain»: це рівень бізнес-логіки;

– папка «presentation»: тут міститься UI та керування станом.

Також варто уточнити, що саме у контексті фреймворку Flutter, чиста архітектура дуже сильно пов’язана з Dependency Injection.

Dependency Injection (DI)[38] – це один з патернів проєктування, який використовується для управління залежностями між об’єктами у застосунку. Його основна ідея полягає в тому, що об’єкт не створює залежності сам, а отримує їх із зовнішнього джерела, зазвичай він їх отримує під час ініціалізації застосунку в одному із кореневих віджетів. Це дозволяє зробити код більш гнучким, масштабованим і легким у тестуванні.

Використання цього патерну також дозволяє реалізувати принципи SOLID[39], зокрема принципу інверсії залежностей (Dependency Inversion Principle). Відповідно до цього принципу, високорівневі модулі не повинні залежати від низькорівневих модулів, а повинні залежати від абстрактних класів. Завдяки DI залежності впроваджуються через інтерфейси або абстракції, що дозволяє легко змінювати реалізації залежностей без зміни основного коду.

Окрім підвищення гнучкості коду, патерн спрощує масштабування проєкту. У тому випадку, якщо потрібно додати новий функціонал або замінити реалізацію уже існуючого, достатньо змінити спосіб ін'єкції залежності без модифікації основного коду. Це дозволяє розділяти логіку на незалежні модулі та підключати їх у різних частинах програми без порушення загальної архітектури. Особливо корисність підходу можна побачити під час тестування.

У Flutter точкою входу у програму є файл main.dart, який відповідає за ініціалізацію застосунку та викликає функцію runApp(). Ця функція приймає кореневий віджет застосунку, який у більшості випадків являється StatelessWidget або StatefulWidget. Важливо, щоб метод build() цього класу повертав віджет MaterialApp (або CupertinoApp для інтерфейсу на iOS), оскільки віджет встановлює базові параметри програми, такі як тема, локалізація, маршрути та головний екран. Саме у цьому файлі перед запуском йде ініціалізація DI-залежностей та BlocProvider, щоб забезпечити керування станом, репозиторіями та сервісами у застосунку.

Окрім цього, у main.dart можна виконувати додаткові налаштування застосунку, наприклад:

– ініціалізація бази даних (наприклад, Firebase Firestore або SQLite) перед початком роботи програми;

– налаштування логування для відстеження помилок;

– зміна орієнтації екрану;

– глобальна обробка винятків та помилок, щоб запобігти аварійному завершенню роботи.

Щодо програмних модулів, то слід виділити репозиторій для роботи з домашніми улюбленцями. Нижче наведений код інтерфейсу репозиторію для домашніх улюбленців.

```
abstract interface class PetRepositoryInterface {
  Future<Pet?> createPet(String uid, Pet pet);
  Future<List<Pet>> fetchAllPets(String uid);
  Future<void> updatePet(Pet updatePet, String uid);
  Future<Pet> fetchPet(String uid,String petId);
  Future<void> deletePet(String uid,String petId);}
```

Нижче наведений код для реалізації інтерфейсу.

```
class FirebasePetRepository implements PetRepositoryInterface {
  final FirebaseAuth firebaseAuth = FirebaseAuth.instance;
  final FirebaseFirestore firebaseFirestore = FirebaseFirestore.instance;
  final FirebaseStorage firebaseStorage = FirebaseStorage.instance;

  @override
  Future<Pet> createPet(String uid, Pet pet) async {
    try {
      final petId = const Uuid().v4();
      final newPet = Pet(petId: petId, uid: uid, name: pet.name, breed:
pet.breed, gender: pet.gender, age: pet.age, petImageUrl: pet.petImageUrl ??
ApplicationImages.petDefaultImage, petPdfUrl: pet.petPdfUrl ?? "");
      await
firebaseFirestore.collection('Users').doc(uid).collection('Pets').doc(petId).set
(newPet.toJson());
      return newPet;} catch (e) {
      throw Exception('Не вдалося створити тварину: $e');}}

  @override
  Future<void> deletePet(String uid, String petId) async {
    try {await
firebaseFirestore.collection('Users').doc(uid).collection('Pets').doc(petId).del
ete();
      final appointmentsSnapshot = await
firebaseFirestore.collection('Users').doc(uid).collection('Appointments').where(
'petId', isEqualTo: petId).get();
      for (var doc in appointmentsSnapshot.docs) {
        await doc.reference.delete();}
    } catch (e) {
      throw Exception(e.toString());}}

  @override
  Future<List<Pet>> fetchAllPets(String uid) async {
    try {final snapshot = await
firebaseFirestore.collection("Users").doc(uid).collection("Pets").get();
      if (snapshot.docs.isEmpty) return [];
      return snapshot.docs.map((doc) => Pet.fromJson(doc.data())).toList();
    } catch (e) {
      throw Exception(e.toString());}}

  @override
  Future<void> updatePet(Pet updatePet, String uid) async {
    try {
```

```
      await
firebaseFirestore.collection("Users").doc(uid).collection("Pets").doc(updatePet.
petId).update({
        'name': updatePet.name,
        'breed': updatePet.breed,
        'gender': updatePet.gender,
        'age': updatePet.age,
        'petImageUrl': updatePet.petImageUrl,
        'petPdfUrl': updatePet.petPdfUrl,
      });
    } catch (e) {
      throw Exception(e.toString());
    }
  }

  @override
  Future<Pet> fetchPet(String uid, String petId) async {
    try {
      final snapshot = await
firebaseFirestore.collection("Users").doc(uid).collection("Pets").doc(petId).get
();
      if (!snapshot.exists) throw Exception("Pet not found");
      final petData = snapshot.data();
      if (petData == null) throw Exception("Pet data is null");
      return Pet(petId: petId, uid: uid, name: petData['name'], breed:
petData['breed'], gender: petData['gender'], age: petData['age'], petImageUrl:
petData['petImageUrl'], petPdfUrl: petData['petPdfUrl']);
    } catch (e) {
      throw Exception("Failed to fetch pet: ${e.toString()}");}}}
```

Імплементація (реалізація) класу-інтерфейсу відбувається через ключове слово «implements». Це означає, що клас зобов’язаний реалізувати всі методи, визначені в інтерфейсі, за допомогою. Методи реалізується за допомогою ключового слова «override» Така реалізація логіки потрібна для того, щоб відокремити опис того, що повинен робити репозиторій, від того, як саме він це реалізує. Це дозволяє будувати гнучку архітектуру, де залежність від конкретної технології (наприклад, Firebase) не прив’язана жорстко до всієї логіки мобільного застосунку. Такий підхід спрощує тестування, оскільки можна легко замінити справжню реалізацію на фейкову або іншими словами – мок-версію. Це також забезпечує масштабованість, тобто якщо в майбутньому з певних причин потрібно буде змінити платформу Firebase на іншу платформу зберігання, це можна буде зробити без змін у коді, що використовує інтерфейс. Таким чином, імплементація інтерфейсу допомагає дотримуватися принципів чистої архітектури, роблячи код більш зрозумілим, підтримуваним, читабельним і розширюваним.

3.2 Розробка бази даних

У попередньому підрозділі серед баз даних було саме Firebase Firestore – NoSQL базу даних. На відміну від SQL баз даних які зберігають інформацію у вигляді таблиць, у яких є поля, то базі даних дані зображені у вигляді колекцій і документів, тобто, firestore є документно-орієнтованою NoSQL базою даних, у якій дані організовані у вкладену структуру:

– колекції містять документи;

– документи містять поля, які працюють за принципом ключ-значення або мають вкладені під-колекції.

В свою чергу для зберігання файлів було використано Firebase Storage, оскільки він надає хмарне сховище.

Файли користувачів, такі як аватарки, зберігаються в бакеті user_images/{id}, а фотографії тварин – у відповідній директорії pet_images /{petId}. Для збереження файлів використовується унікальний ідентифікатор (наприклад, Uuid), це допомагає уникнути конфлікту імен. При завантаженні файлів є певне обмеження за розміром і типом даних. Доступ до файлів контролюється через правила безпеки Firebase, які надає хмарний сервіс, вони дозволяють перегляд та зміну даних лише авторизованим користувачам та лише над своїми даними. При видалені файлів передбачено каскадне видалення, адже коли користувач видаляє свій профіль або тварину, потрібно і видаляти відповідні файли із Firebase Storage. На рисунку 3.2 показаний Firebase Storage.

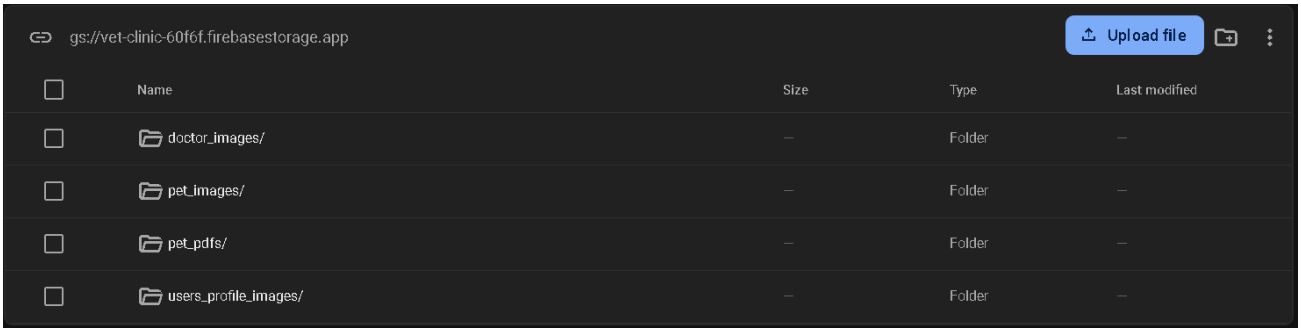


Рисунок 3.2 – Firebase Storage

База даних розроблялася за логічною моделю «сутність-зв’язок», яка зображена на рисунку 2.4.

Відповідно, перша колекція розробленої бази даних це «users», яка в свою чергу буде містити такі поля:

– id: поле, яке містить унікальний ідентифікатор користувача, конкретно у випадку Firebase це поле не є обов’язковим, адже Firebase надає зручні інструменти які дозволяють робити дії з юзерами (видалення, оновлення тощо) без ідентифікатора;

– email: поле, яке зберігає електрону почту користувача;

– username: поле для зберігання ім’я користувача;

– password: поле, яке зберігає пароль користувача;

– image_url: поле, яке містить посилання на фото користувача, яке зберігається у Firestore Storage.

Приклад колекції «user» з тестовим записом зображено на рисунку 3.3

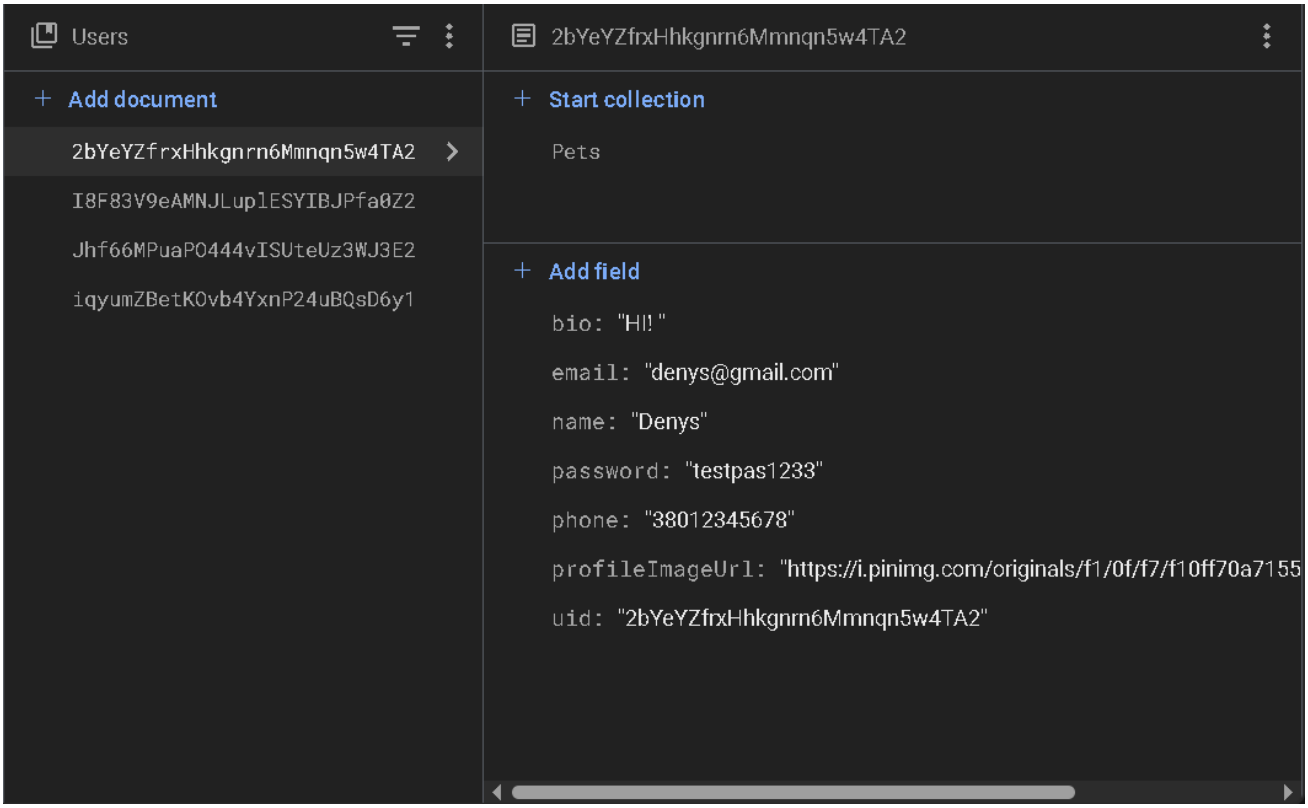


Рисунок 3.3 – Колекція «users»

Наступна колекція це «pets», але в даній ситуації, цю колекцію варто зробити під-колекцією, адже вона повністю залежить від користувача. У Firebase Firestore вкладена колекція – це просто колекція, яка зберігається всередині документа іншої колекції. Отож, у результаті під-колекція «pets» буде містити такі поля:

– id: поле, для зберігання унікального ідентифікатора для кожного домашнього улюбленця;

– name: поле, яке зберігає ім’я тварини;

– gender: поле, яке зберігає стать тварини;

– breed: поле, яке зберігає породу;

– age: поле, яке зберігає вік;

– petImageUrl: поле, яке містить посилання на фото тварини, яке зберігається у Storage;

– petPdfUrl: поле, яке містить посилання на документ(файл) тварини, яке зберігається у Storage;

– user_id: ідентифікатор користувача, якому належить цей улюбленець.

Приклад під-колекції «pets» з тестовим записом зображено на рисунку 3.4

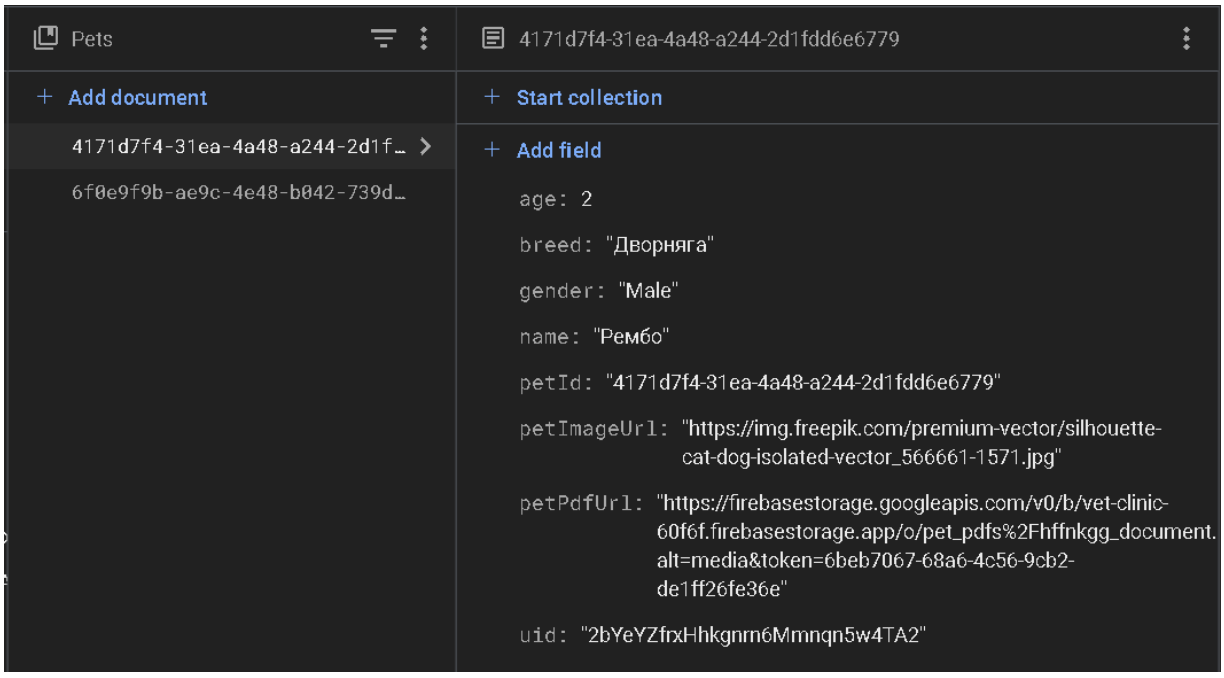


Рисунок 3.4 – Під-колекція «pets»

Ну і також варто згадати ще про колекцію «appointments». Вона в свою чергу буде вважатися під-колекцією «pets» та буде містити з такими поля:

– id: поле, для зберігання унікального ідентифікатора для кожного запису;

– status: поле для статусу запису (scheduled, canceled, completed тощо);

– datetime: дата та час запису, підтримуються часові зони;

– pet_id: ідентифікатор тварини, яка записана на прийомі;

– doctor_id: ідентифікатор доктора, який буде на прийомі.

Приклад під-колекції «appointments» з тестовим записом зображено на рисунку 3.5.

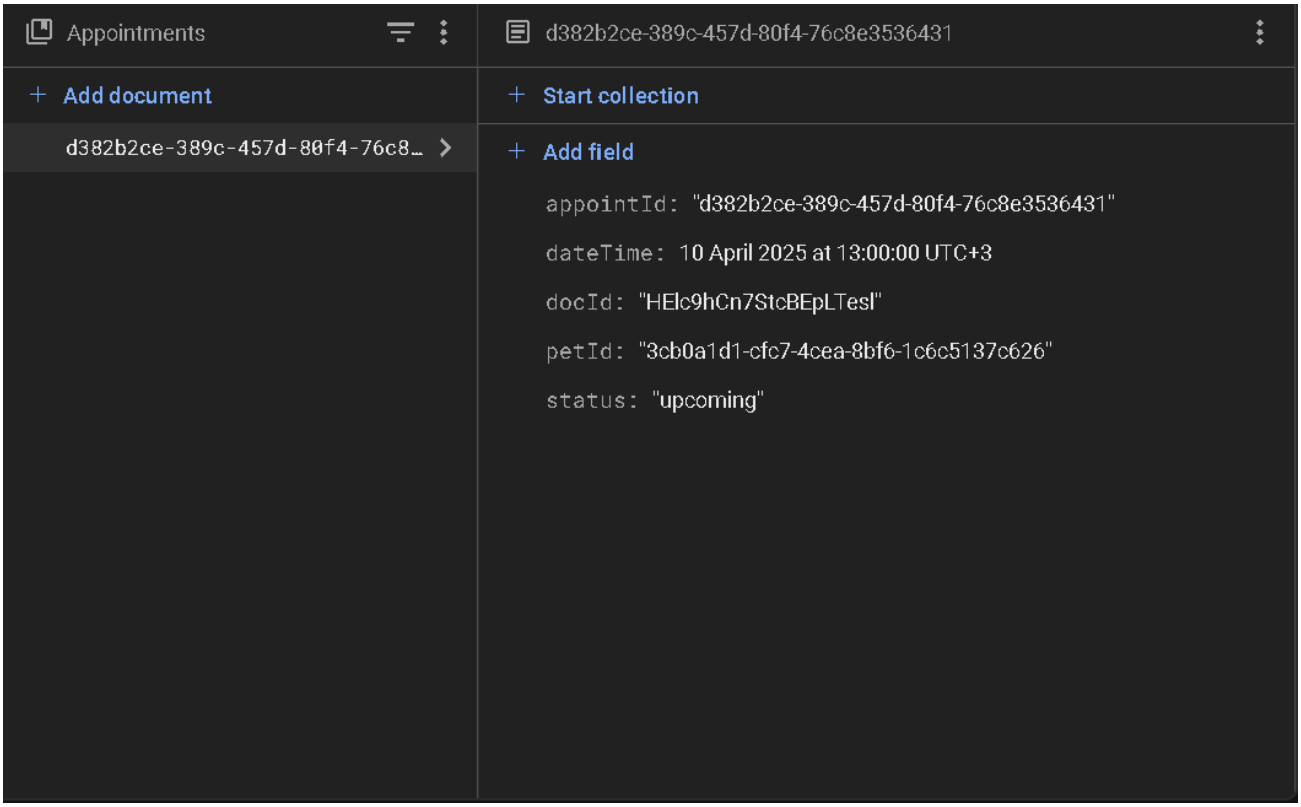


Рисунок 3.5 – Під-колекція «appointments»

По аналогії була розроблена і колекція «doctors». Вона по ієрархії знаходиться на рівні з «users», адже вона також незалежна ні від кого.

Стосовно таблиці «reminders», то було прийнято рішення створити локальну таблицю в базі даних SQLite, щоб простіше було взаємодіяти з локальними нагадуваннями через push-сповіщення.

3.3 Вимоги до технічних та програмних засобів

Розроблений мобільний застосунок призначений для використання на сучасних мобільних пристроях з операційною системою Android або iOS. Для забезпечення стабільної та коректної роботи програмного забезпечення необхідно дотримуватись наведених нижче мінімальних вимог до апаратного та програмного забезпечення.

Мінімальні апаратні вимоги:

– операційна система мобільного пристрою: Android 8.0 (API 26) або новіша / iOS 13.0 або новіша;

– процесор: 64-бітний, мінімум 4 ядра;

– оперативна пам’ять: не менше 2 ГБ;

– вільна пам’ять для встановлення: щонайменше 150 МБ;

– інтернет-з’єднання: стабільне підключення до Wi-Fi або мобільного інтернету (2G і вище).

Програмне забезпечення:

– системне ПЗ: Android OS / iOS з підтримкою Google Play Services або Apple Services;

– драйвери та утиліти: драйвери для роботи з камерою, сховищем та мережею (встановлюються системою автоматично);

Додаткові вимоги:

– доступ до ресурсів пристрою: дозвіл на доступ до камери, локального сховища та мережі;

– безперервне інтернет-з’єднання: для повноцінної роботи застосунку потрібне інтернет з’єднання для обміну даними з сервером та отримання сповіщень від застосунку.

Застосунок розроблений із врахуванням вимог енергоефективності, оптимізований для стабільної роботи на широкому спектрі пристроїв середнього та високого класу.

3.4 Тестування системи

3.4.1 Аналіз методів тестування мобільного застосунку

Тестування, як один з етапів життєвого циклу є невід'ємною складовою розробки мобільного застосунку. Воно забезпечує якість продукту та гарантує його стабільну і коректну роботу як на різних пристроях так і на різних операційних системах. Зазвичай у Flutter виконують такі типи тестування [40]:

– unit-тести: ці тести в першу чергу фокусуються на перевірці окремих класів, методів чи функцій без залежностей від інтерфейсу користувача чи зовнішніх сервісів. Такі тести дозволяють швидко і точно перевірити логіку бізнес-процесів. Наприклад, можна протестувати функцію обробки даних, валідацію форм, обчислення тощо;

– widget-тести: ці тести (інколи їх називають компонентними тестами) дозволяють перевірити, як конкретні елементи інтерфейсу реагують на взаємодію користувача, наприклад, зміна стану або параметрів. Вони ізольовані та працюють швидко, що дозволяє ефективно перевіряти інтерфейс користувача без повного запуску застосунку;

– integration-тести: вони охоплюють абсолютно весь застосунок або його окремі великі частини й перевіряють, як усі компоненти працюють разом. Фактично, вони емулюють дії користувача, такі як, натискання кнопок, введення даних, навігацію між екранами, взаємодію з базами даних F або REST API. Вони проводять сценарій тестування «від початку до кінця».

– golden тести: це тип тестування для інтерфейсу користувача, який дозволяє перевіряти, чи не змінився зовнішній вигляд UI. Під час тесту фреймворк рендерить віджет і зберігає його як зображення – в так званий «золотий файл». Потім, при наступному запуску тесту, це зображення порівнюється з актуальним результатом. Якщо є будь-які відмінності, тест не проходить. Такий підхід дозволяє виявляти навіть дрібні небажані зміни в дизайні застосунку.

Flutter надає доволі зручні і ефективні інструменти для написання різних типів тестів, серед яких можна виділити[41]:

– бібліотека flutter_test: це офіційний інструмент для написання unit або widget-тестів у Flutter. Вона надає всі необхідні засоби для перевірки логіки, взаємодії з інтерфейсом, роботи з віджетами та імітації дій користувача. За допомогою цієї бібліотеки можна тестувати зміну станів, відображення елементів інтерфейсу, введення тексту в полі вводу та натискання кнопок без повного запуску застосунку. Вона є базовим компонентом для швидкого та ефективного тестування більшості частин Flutter-застосунку;

– бібліотека patrol: ще один офіційний інструмент для тестування Flutter-застосунків, який поєднує в собі як інтеграційні тести, так і інші їх види. Вона дозволяє проводити тестування сценаріїв користувача: наприклад, відкриття екранів, навігацію, авторизацію, введення даних і взаємодію з API. Ці тести запускаються на реальних чи емуляторних пристроях і максимально наближені до реального використання застосунку користувачем.

– бібліотеки mockito та mocktail: обидві бібліотеки призначені для мокування – створення несправжній версій об’єктів та сервісів, які застосунок використовує у своїй роботі. Це дозволяє тестувати окремі компоненти в ізоляції від зовнішніх залежностей, наприклад, API, бази даних чи Firebase. Mockito – класична бібліотека з чітким підходом до створення моків, тоді як mocktail – сучасніший варіант, зручніший для Dart та підтримує null safety. Обидві бібліотеки широко використовуються в unit- та widget-тестах.

– bloc_test: це бібліотека для тестування BLoC у Flutter, яка дозволяє перевіряти, як BLoC реагує на події і які стани змінює. Вона працює за принципом build, потім act і в кінці expect, де створюється BLoC, подаються події і перевіряються очікувані стани. Це зручно для ізольованого тестування бізнес-логіки без залежності від UI або зовнішніх сервісів. bloc_test добре інтегрується з mocktail чи mockito і підходить як для простих, так і для складних сценаріїв, роблячи тести більш читабельними й надійними.

3.4.2 Тестування мобільного застосунку за допомогою емулятора

Тестування будь-якого мобільного застосунку зазвичай розпочинається з використання емуляторів, оскільки це значно спрощує й пришвидшує процес перевірки функціональності. Емулятори є невід'ємним інструментом у повсякденній роботі розробників і тестувальників, особливо на етапах активної розробки та налагодження. Серед основних переваг використання емуляторів можна виділити наступні:

– швидкість і зручність: емулятори дозволяють запускати застосунок без потреби підключати фізичні пристрої.

– можливість автоматизованого тестування: емулятори добре інтегруються з CI/CD системами.

– економія ресурсів: не потрібно купувати велику кількість пристроїв для тестування на різних версіях ОС чи з різними характеристиками.

– інструменти розробника: емулятори часто мають вбудовані інструменти для налагодження, логування, емуляції різних умов (наприклад, слабкий інтернет, GPS, дзвінки).

Однак, попри всі переваги, використання лише емуляторів не є достатнім для повноцінного тестування. Вони не завжди можуть точно імітувати реальні умови експлуатації. Наприклад, на фізичних пристроях може відрізнятись швидкодія, поведінка сенсорів, точність взаємодії з інтерфейсом, рівень споживання енергії, реакція на багатозадачність чи фонові процеси. Також варто враховувати особливості прошивок від різних виробників, які можуть впливати на роботу застосунку.

Узагальнюючи все вище сказане, найкращий підхід – це поєднання тестування на емуляторах із регулярною перевіркою на реальних пристроях, що дозволяє забезпечити високу якість та стабільність мобільного застосунку в реальних умовах. Такий підхід допомагає своєчасно виявляти потенційні проблеми, які не видно під час віртуального тестування.

Для цього було розроблено тести, які будуть реалізовуватися у застосунку. Повний список тестів для інтерфейсу користувача (включаючи окремі кастомні віджети) знаходиться у таблиці 3.2.

Таблиця 3.2 – Список тестів для інтерфейсу користувача

Назва тесту	Категорія	Пояснення
renders errorText when error is provided (CustomField)	Юніт-тест	Виводиться повідомлення про помилку
toggles visibility icon (CustomField)	Юніт-тест	Іконка перемикає видимість паролю
changes value on selection (CustomDropDownButtonFormField)	Юніт-тест	Змінюється значення при виборі іншого
renders loading spinner (Loader)	Юніт-тест	Присутній CircularProgressIndicator
triggers onPressed on tap (TileButton)	Юніт-тест	Викликається дія при натисканні
renders correctly (ThemeLogo)	Модульний тест	Логотип рендериться згідно з темою
renders logo + app name (ThemeLogo)	Модульний тест	Відображається SVG + текст назви
renders search input (SearchField)	Юніт-тест	Поле пошуку з’являється на екрані
filters list on input (SearchField)	Віджет-тест	Перевірка відображення фільтрованого списку
validates empty fields (AuthScreen)	Інтеграційний тест	Валідація для порожніх полів
login fails with wrong creds (AuthScreen)	Інтеграційний тест	Виводиться помилка при невірних даних

Продовження таблиці 3.2

navigates to register screen (LoginScreen)	Інтеграційний тест	Відбувається перехід на реєстрацію
emits Authenticated state (AuthCubit)	Юніт-тест	Cubit повертає авторизований стан
navigates to home after auth (CheckAuthScreen)	Інтеграційний тест	Успішна авторизація — перехід на головну
shows correct screen per route (VetClinicApp)	Інтеграційний тест	Тест маршрутизації: правильний екран для кожного шляху
sets correct theme (VetClinicApp)	Юніт-тест	Застосовується відповідна тема
golden test of AuthScreen	Golden тест	Візуальне тестування повного екрану
golden test of TileButton	Golden тест	Знімок рендеру окремого компонента
golden test of Loader	Golden тест	Перевірка стилів завантажувача

Аналогічно було розроблено тести і для бізнес-логіки застосунку. Повний список тестів для логіки знаходиться у таблиці 3.3.

Таблиця 3.3 – Список тестів для бізнес логіки

Назва тесту	Категорія	Пояснення
повертає список тварин (GetAnimalsUseCase)	Юніт-тест	Перевіряє, що use case повертає всі тварини користувача
додає тварину (CreateAnimalUseCase)	Юніт-тест	Тестує логіку створення нового улюбленця

Продовження таблиці 3.3

видаляє тварину за ID (DeleteAnimalUseCase)	Юніт-тест	UseCase видаляє тварину і повертає підтвердження
повертає список лікарів (DoctorRepositoryImpl)	Юніт-тест	Перевірка правильного отримання лікарів з Supabase
Створює візит (CreateAppointmentUseCase)	Юніт-тест	Тестує логіку створення нового запису до ветеринара
повертає візити з деталями (GetAppointmentsWithDetailsUseCase)	Юніт-тест	Перевіряє об'єднання візиту + тварина + лікар
повертає візити лише для поточного користувача (AppointmentRepository)	Юніт-тест	Тестує фільтрацію візитів по userId
кидає помилку при неправильній даті (CreateAppointmentUseCase)	Юніт-тест	Валідація вхідних даних на рівні use case
оновлює стан при додаванні тварини (AnimalBloc)	Юніт-тест	Bloc змінює стан після події створення тварини
повертає помилку при фейлі (AnimalBloc)	Юніт-тест	Bloc правильно оброблює винятки з репозиторію
оновлює список візитів (AppointmentBloc)	Юніт-тест	Bloc завантажує візити і віддає новий стан
автентифікація успішна (AuthCubit)	Юніт-тест	Cubit переходить у стан авторизованого користувача
обробляє стан завантаження (AppointmentBloc)	Юніт-тест	Bloc спочатку в стані loading, потім loaded
видаляє тварину за ID (DeleteAnimalUseCase)	Юніт-тест	UseCase видаляє тварину і повертає підтвердження

3.4.3 Аналіз результатів тестування мобільного застосунку

Нижче наведений код до інтеграційного тесту, який перевіряє базову аутентифікацію та навігацію у застосунку, який використовує Firebase. Спочатку виконується ініціалізація Firebase та створення об’єктів для роботи з аутентифікацією. Для тесту логіну генерується унікальний email, видаляється потенційно існуючий користувач з такими даними, після чого створюється новий користувач. Далі через інтерфейс користувача вводяться email і пароль, натискається кнопка логіну, і перевіряється, що відкривається головний екран, тобто вхід пройшов успішно.

Після цього запускається профіль користувача, імітується натискання кнопки переходу в налаштування, після чого виконується вихід із системи. Тест перевіряє, що після цього з’являється екран входу. Усе це дозволяє переконатися, що основні функції входу, виходу і навігації між відповідними екранами працюють коректно.

```
      void main() {
        IntegrationTestWidgetsFlutterBinding.ensureInitialized();
        late AuthFirebaseRepository authRepository;
        late FirebaseAuth firebaseAuth;
        setUpAll(() async
      {
          WidgetsFlutterBinding.ensureInitialized();
          await Firebase.initializeApp(options:
DefaultFirebaseOptions.currentPlatform);
          authRepository = AuthFirebaseRepository();
          firebaseAuth = FirebaseAuth.instance; });
        Future<void> deleteTestUser(String email, String password) async {
          try {
            final user = await authRepository.loginWithEmailAndPassword(email,
password);
            if (user != null) {
              await firebaseAuth.currentUser?.delete();}
          } catch (e)
      {
         print(e.toString());
      }}
        testWidgets('Успішний логін користувача', (tester) async {
          app.main();
          await tester.pumpAndSettle();
       final email =
'login_${DateTime.now().millisecondsSinceEpoch}@example.com';
          const password = 'password123';
          await deleteTestUser(email, password);
          await authRepository.registerWithEmailAndPassword(email, 'Test User',
password);
```

```
          await Future.delayed(const Duration(seconds: 3));
          await tester.pumpAndSettle();
          await tester.enterText(find.byKey(const Key('emailField')), email);
          await tester.enterText(find.byKey(const Key('passwordField')),
password);
          await tester.tap(find.byKey(const Key('loginButton')));
          await tester.pumpAndSettle();
          expect(find.byType(HomeScreen), findsOneWidget);});
        testWidgets('Navigate to settings and perform logout', (WidgetTester
tester) async {
          final profileRepository = FirebaseProfileRepository();
          final storageRepository = FirebaseStorageRepository();
          Future<SharedPreferences> initPrefs() async {
            final preferences = await SharedPreferences.getInstance();
            return preferences;}
          final preferences = await initPrefs();
          final settingsRepository = SettingsRepository(preferences:
preferences);
          final profileCubit = ProfileCubit(
            profileRepository: profileRepository,
            storageRepository: storageRepository,);
          await tester.pumpWidget(
            MultiBlocProvider(
              providers: [
                BlocProvider<ProfileCubit>.value(value: profileCubit),
                BlocProvider<ThemeCubit>(
                  create: (_) => ThemeCubit(settingsRepository:
settingsRepository),),
                BlocProvider<LanguageCubit>(
                  create: (_) => LanguageCubit(settingsRepository:
settingsRepository),),],
              child: const MaterialApp(
                localizationsDelegates: [
                  S.delegate,
                  GlobalMaterialLocalizations.delegate,
                  GlobalWidgetsLocalizations.delegate,],
                home: ProfileScreen(uid: 'testUid'),
      ),),);
          expect(find.byType(ProfileScreen), findsOneWidget);
          await tester.tap(find.byKey(const Key('settingsButton')));
          await tester.pumpAndSettle();
          expect(find.byType(SettingsScreen), findsOneWidget);
          await tester.tap(find.byKey(const Key('logoutButton')));
          await tester.pumpAndSettle();
          expect(find.byType(LoginScreen), findsOneWidget);
      });}
```

На рисунку 3.6 зображений результат тесту, коли дані були введені правильно та успішно оброблені системою.

```
PS C:\Users\admin\Desktop\vet_clinic> flutter test integration_test
00:00 +0: loading C:/Users/admin/Desktop/vet_clinic/integration_test/auth_firebase_repository_test.dart
00:50 +0: loading C:/Users/admin/Desktop/vet_clinic/integration_test/auth_firebase_repository_test.dart
√ Built build\app\outputs\flutter-apk\app-debug.apk
00:53 +0: loading C:/Users/admin/Desktop/vet_clinic/integration_test/auth_firebase_repository_test.dart
00:56 +0: loading C:/Users/admin/Desktop/vet_clinic/integration_test/auth_firebase_repository_test.dart
01:09 +3: All tests passed!
```

Рисунок 3.6 – Результат end-to-end тесту реєстрації

Цей результат свідчить про те, що логіка взаємодії з Firebase Authentication працює коректно: дані, передані у формі входу, були перевірені на стороні сервера, авторизація пройшла без помилок, і користувач був переадресований на головний екран без необхідності повторного введення пароля або інших втручань.

На рисунку 3.7 зображено результат виконання тесту з очікуваною помилкою при повторному вході в систему. У даному випадку тест перевіряє відповідність отриманого імені користувача очікуваному значенню. Як видно з логу помилки, очікуване значення – 'Wrong Name', тоді як фактичне – 'Test User', що призводить до помилки. Це дозволяє переконатися, що система коректно реагує на невідповідність даних і може бути використано для перевірки функціоналу до некоректних або змінених входів. Такий тест ілюструє, що механізм валідації працює, і система реагує згідно з очікуванням.

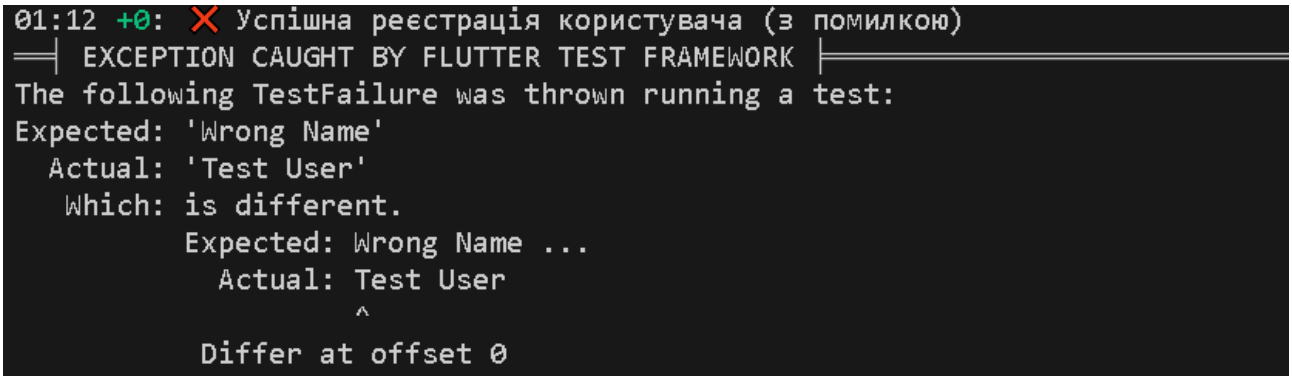


Рисунок 3.7 – Очікуваний результат помилки end-to-end тесту

Це лише невелика частина тих тестів, які можуть бути у нашому застосунку, реальна їх кількість значно більша, адже у застосунках таких розмірів існує багато взаємодій між компонентами, різні сценарії використання, обробка помилок, граничні випадки та особливості роботи на різних пристроях і версіях операційної системи. Кожен модуль або функція застосунку потребує окремих юніт-тестів, інтеграційних тестів, golden тестів, а також end-to-end тестів які імітують поведінку реального користувача.

3.5 Висновки програмної реалізації та тестування

У цьому підрозділі розглянуто ключові аспекти розробки мобільного застосунку за допомогою Flutter. Інтерфейс побудовано на віджетах, що забезпечують гнучкість і повторне використання компонентів. Розмежування на StatelessWidget та StatefulWidget допомагає ефективно керувати станом застосунку. Завдяки дотримання принципів Clean Architecture спростилась та стала зрозуміліша структура коду, а застосування Dependency Injection покращила гнучкість і керування залежностями. Застосунок відповідає сучасним архітектурним підходам, що сприяє його стабільності та розширюваності в майбутньому.

База даних побудована на Firebase Firestore, що відображає логічні зв'язки між користувачами, тваринами, записами на прийом та лікарями. Зберігання файлів через Firebase Storage забезпечує безпеку і унікальність імен, а для локальних нагадувань використано SQLite, що гарантує стабільну роботу без Інтернету. Така структура відповідає вимогам застосунку та забезпечує ефективну взаємодію компонентів.

Технічні та програмні вимоги гарантують стабільну роботу на більшості пристроїв Android та iOS, навіть середнього рівня. Забезпечено підтримку всіх необхідних служб для ефективної взаємодії з апаратними можливостями пристрою та якісного користувацького досвіду.

Окремо описано важливість тестування для забезпечення якості застосунку. Розглянуті типи тестів, зокрема unit, widget, integration та golden тести, а також інструменти для тестування у Flutter (flutter_test, patrol, mockito, bloc_test). У роботі також наведено список тестів, зокрема для перевірки реєстрації та логіну, що забезпечує коректну роботу застосунку на різних пристроях та системах.

ВИСНОВКИ

У межах кваліфікаційної роботи реалізовано повний цикл створення кросплатформеного мобільного застосунку для запису до ветеринара та догляду за домашніми тваринами. Робота охоплює дослідження предметної області, аналіз вимог, проєктування архітектури, програмну реалізацію, тестування та підготовку супровідної документації.

Основна мета полягала у розроблені сучасного, зручного і функціонального інструменту, що дає змогу власникам домашніх улюбленців ефективно організовувати ветеринарне обслуговування, вести облік медичної інформації, отримувати нагадування про важливі події та забезпечити повноцінне цифрове середовище для догляду за тваринами.

У першому розділі проводився всебічний аналіз предметної області та обґрунтовувалась необхідність створення подібного програмного забезпечення. Результати дослідження показали, що існує актуальний попит на централізовані рішення для зберігання інформації про стан здоров’я тварин, їхні щеплення, історію відвідувань ветеринарів тощо. Було досліджено наявні рішення на ринку (зокрема, 11Pets, AnimalID, DogCatApp), і на основі їх сильних сторін та недоліків сформульовано функціональні та нефункціональні вимоги до майбутнього застосунку. Також здійснено техніко-економічне обґрунтування вибору технологій: обрана кросплатформена технологія Flutter дозволила забезпечити одночасну підтримку Android та iOS, а хмарна платформа Firebase – реалізувати зберігання даних, автентифікацію користувачів і роботу в реальному часі.

У другому розділі увага приділялася архітектурному та серверному проєктуванню системи. Проаналізовано популярні архітектурні підходи до побудови мобільних застосунків (MVC, MVP, MVVM, Serverless Architecture). Як результат – вибір Serverless Architecture як оптимального варіанту, що дозволяє спростити інфраструктуру, зменшити витрати на серверне

обслуговування та забезпечити масштабованість застосунку. Сформовано логічну модель даних, визначено основні сутності системи (користувач, тварина, лікар, прийом), їх атрибути та зв'язки. На основі цієї моделі побудовано ER-діаграму, яка є основою для реалізації бази даних. У процесі проєктування інтерфейсу застосунку було враховано сучасні вимоги до UX/UI, а також б реалізовано підтримку світлої та темної теми, підібрано кольорову палітру, шрифти, компонування елементів, що забезпечує інтуїтивність та комфорт користування застосунком.

У третьому розділі відбувалася реалізація застосунку. Для побудови адаптивного, модульного інтерфейсу використовувалися основні інструменти Flutter SDK, включаючи StatelessWidget та StatefulWidget. Завдяки дотриманню принципів Clean Architecture вдалося реалізувати чітку розділеність відповідальності: шар UI, бізнес-логіка (use cases) та доступ до даних (репозиторії). Реалізовано ін'єкцію залежностей з використанням механізмів Dependency Injection, що підвищило гнучкість і масштабованість системи. Основним сховищем для зберігання даних про користувачів, тварин, записи та лікарів став Firebase Firestore, а для зображень – Firebase Storage, що забезпечило безпечну та швидку обробку медіаконтенту. Для реалізації локальних сповіщень було інтегровано SQLite, що дозволяє користувачам отримувати нагадування навіть за відсутності підключення до мережі Інтернет. Застосунок підтримує аутентифікацію, створення профілів, додавання тварин, запис на майбутні події та інші функціональні можливості.

Окрему увагу приділено тестуванню. Для перевірки працездатності реалізованих компонентів було реалізовано комплексний набір тестів: unit-тести перевіряли бізнес-логіку, widget-тести – правильність рендерингу інтерфейсу, integration-тести – взаємодію між компонентами, а golden-тести – відповідність зовнішнього вигляду заданому шаблону. Було використано такі інструменти, як flutter_test, mockito, bloc_test, patrol. Це дозволило перевірити стабільність застосунку на різних пристроях, з різними версіями ОС, та забезпечити високу якість кінцевого продукту.

Впровадження розробленого застосунку надає низку переваг для користувачів. Воно дозволяє скоротити витрати часу, уникати пропущених візитів, автоматизувати ведення медичної історії тварин, зберігати важливу інформацію у зручному форматі, що покращує якість догляду за улюбленцями. Також програмний продукт дозволяє оптимізувати комунікацію з ветеринарами, забезпечити зберігання історії консультацій, нагадувань, та швидко орієнтуватись у даних про кількох тварин одночасно. Система може бути легко масштабована та адаптована до інших сфер, зокрема для роботи у зоомагазинах, притулках для тварин, ветеринарних клініках, а також у сфері ведення обліку тварин у муніципальних або фермерських господарствах.

У майбутньому розробку можна розширити шляхом інтеграції з онлайн-консультаціями, чіпами або QR-ідентифікаторами для швидкої ідентифікації тварини, реалізацією веб чи десктоп версії платформи, підтримкою сповіщень через email або push-механізми, а також створенням аналітичного модуля для відстеження динаміки здоров’я тварин.

Розробка також була представлена у збірнику наукових праць за матеріалами XV Всеукраїнської науково-практичної конференції «Актуальні проблеми комп’ютерних наук АПКН-2024» [42].

Таким чином, виконання кваліфікаційної роботи дозволило реалізувати повноцінний життєвий цикл створення програмного забезпечення: від ідеї та аналізу до розробки, тестування та можливості впровадження. Результатом є сучасний, стабільний та зручний у користуванні мобільний застосунок, що відповідає поставленим завданням, є практично цінним і має значний потенціал для подальшого розвитку та масштабування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ринок мобільних додатків 2023 – звіт. Загальна кількість завантажень iOS та Android у 2023 році URL: https://asomobile.net/uk/blog/rinok-mobilnih-dodatkiv-2023-zvit/?utm_source=chatgpt.com# (дата звернення 02.01.2025)

2. What is a mobile app (mobile application)? URL: https://www.techtarget.com/whatis/definition/mobile-app (дата звернення 02.01.2025)

3. Які існують типи мобільних застосунків. URL: https://training.qatestlab.com/blog/technical-articles/types-of-mobile-applications/ (дата звернення 02.01.2025)

4. Mobile Operating System Market Share Worldwide. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-202401-202501 (дата звернення 05.01.2025)

5. Mobile Operating System Market Share Ukraine. URL: https://gs.statcounter.com/os-market-share/mobile/ukraine#monthly-202401-202501 (дата звернення 05.01.2025)

6. About 11Pets application. URL: https://www.11pets.com/en/about (дата звернення 10.01.2025)

7. Що таке Animal ID? URL: https://animal-id.net/ua/about/project (дата звернення 16.01.2025)

8. About DogCat application. URL: https://dogcat.app/ (дата звернення 20.01.2025)

9. Функціональні та нефункціональні вимоги. URL: https://www.guru99.com/uk/functional-vs-non-functional-requirements.html (дата звернення 24.01.2025)

10. Специфікація вимог до програмного забезпечення URL: https://stfalcon.com/uk/blog/post/How-to-Write-a-Software-Requirements-Specification (дата звернення 24.01.2025)

11. What is SHA- 256? Introduction of SHA-256 URL: https://www.encryptionconsulting.com/education-center/sha-256/ (дата звернення 26.01.2025)

12. Get to know Material 3 – from UX guidance and tools to reusable components and open-source code. URL: https://m3.material.io/get-started (дата звернення 28.01.2025)

13. Designing for iOS. URL: https://developer.apple.com/design/human-interface-guidelines/designing-for-ios (дата звернення 28.01.2025)

14. Для чого потрібні UML діаграми? URL: https://foxminded.ua/uml-diagramy/ (дата звернення 05.02.2025)

15. Types of a UML diagrams. URL: https://miro.com/diagramming/what-is-a-uml-diagram/ (дата звернення 05.02.2025)

16. Include relationships. URL: https://www.ibm.com/docs/en/dma?topic=diagrams-include-relationships (дата звернення 05.02.2025)

17. Extend relationships. : https://www.ibm.com/docs/en/dma?topic=diagrams-extend-relationships (дата звернення 05.02.2025)

18. Архітектура програмного забезпечення: все що треба знати. URL: https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya (дата звернення 18.02.2025)

19. Основні типи архітектури програмного забезпечення. URL: https://www.artofba.com/uk/post/main-types-of-software-architecture (дата звернення 19.02.2025)

20. Архітектура програмного забезпечення URL: https://javarush.com/ua/groups/posts/uk.2519.chastina-2-pogovorimo-trokhi-pro-arkhtekturu-pz (дата звернення 19.02.2025)

21. Патерни проєктування. URL: https://www.linkedin.com/posts/stivax_%D0%BF%D0%B0%D1%82%D0%B5%D1%80%D0%BD%D0%B8-%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D1%83%D0%B2%D0

%B0%D0%BD%D0%BD%D1%8F-mvvm-mvp-%D1%96-mvc-activity-7090375282259288067-X9fc/ (дата звернення 20.02.2025)

22. Overview of serverless architecture and its advantages. URL: https://elartu.tntu.edu.ua/bitstream/lib/41424/2/110-111.pdf (дата звернення 21.02.2025)

23. Огляд Serverless архітектури. URL: https://devzone.org.ua/post/ohliad-serverless-arkhitektury (дата звернення 21.02.2025)

24. What is function as a service (FaaS)? URL: https://www.ibm.com/think/topics/faas (дата звернення 24.02.2025)

25. What is state management? URL: https://www.techtarget.com/searchapparchitecture/definition/state-management (дата звернення 27.02.2025)

26. What is Riverpod? URL: https://riverpod.dev/docs/introduction/why_riverpod (дата звернення 01.03.2025)

27. Provider: What is it, what is it for, and how to use it? URL: https://medium.com/bancolombia-tech/flutter-provider-what-is-it-what-is-it-for-and-how-to-use-it-47d6941860d7 (дата звернення 01.03.2025)

28. Bloc Concepts. URL: https://bloclibrary.dev/bloc-concepts/ (дата звернення 01.03.2025)

29. Логічна модель даних. URL: https://data-life-ua.com/db/lohichna-model-danykh/ (дата звернення 04.03.2025)

30. Programming Language Dart. URL: https://www.ionos.com/digitalguide/websites/web-development/dart-programming-language/ (дата звернення 10.03.2025)

31. Flutter | An introduction to the open source SDK by Google. URL: https://www.geeksforgeeks.org/flutter-an-introduction-to-the-open-source-sdk-by-google/ (дата звернення 10.03.2025)

32. Widgets in Flutter. URL: https://docs.flutter.dev/get-started/fundamentals/widgets (дата звернення 10.03.2025)

33. Firebase – Introduction. URL: https://www.geeksforgeeks.org/firebase-introduction/ (дата звернення 10.03.2025)

34. Cloud Storage for Firebase. URL: https://firebase.google.com/docs/storage (дата звернення 10.03.2025)

35. Widget Catalog. URL: https://docs.flutter.dev/ui/widgets (дата звернення 11.03.2025)

36. Flutter – Stateful vs Stateless Widgets URL: https://www.geeksforgeeks.org/flutter-stateful-vs-stateless-widgets/ (дата звернення 12.03.2025)

37. The CleanArchitecture. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (дата звернення 12.03.2025)

38. The DependencyInjection Component https://symfony.com/doc/current/components/dependency_injection.html. (дата звернення 14.03.2025)

39. Принципи SOLID. Навіщо вони потрібні програмістам та як з ними працювати. URL: https://campus.epam.ua/ua/blog/602 (дата звернення 15.03.2025)

40. Тестування Flutter-додатків. URL: https://avada-media.ua/blog/testirovaniye-flutter-prilozheniy/ (дата звернення 20.03.2025).

41. Top Flutter Testing packages. URL https://fluttergems.dev/testing/ (дата звернення 23.03.2025).

42. Паламарчук Д.В МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ЗАПИСУ ДО ВЕТЕРИНАРА ТА ДОГЛЯДУ ЗА ТВАРИНАМИ Актуальні проблеми комп'ютерних наук АПКН-2024 : зб. наук. пр., м. Хмельницький, 12 листоп. 2024 р. Хмельницький, 2024. С. 417–420. URL: https://kn.khmnu.edu.ua/wp-content/uploads/sites/18/apkn-2024-corpuspaper.pdf (дата звернення: 10.04.2024).

ДОДАТОК А

(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Загальні відомості

Робота виконується в рамках проєкту розробки мобільного застосунку, спрямованого на спрощення процесу догляду за домашніми улюбленцями, а також для запису до спеціаліста ветеринарної медицини та збереження всієї необхідної інформації про здоров’я улюбленців в одному місці.

Застосунок дозволяє власникам тварин вести електронний профіль своїх улюбленців, записуватися на прийоми до ветеринарів, отримувати нагадування про догляд (вакцинації, прийом ліків, грумінг тощо) та переглядати історію візитів і рекомендацій лікарів.

Умовне позначення системи: Мобільний застосунок «VetCare» – застосунок для запису до ветеринара та організації догляду за тваринами.

Обґрунтування розробки застосунку

Розробка здійснюється відповідно до «Завдання на кваліфікаційну роботу», затвердженого дипломним керівником та завідувачем кафедри інженерії програмного забезпечення.

Назва проєкту: «VetCare» – мобільний застосунок для запису до ветеринара та організації догляду за домашніми тваринами.

Призначення розробки

Мобільний застосунок **«VetCare»** призначений для надання власникам домашніх тварин зручного та простого інструменту для організації догляду за їхніми улюбленцями.

Застосунок дозволить користувачам записуватися на прийом до ветлікаря, переглядати інформацію про клініки та спеціалістів, вести історію відвідувань і

лікування, отримувати нагадування про вакцинацію, прийом ліків та інші важливі події, а також зберігати всі дані про здоров'я тварини в одному місці.

Крім того, застосунок матиме функцію рекомендацій щодо догляду, використовуючи штучний інтелект, що допоможе власникам краще піклуватися про своїх тварин.

Функціональне призначення мобільного застосунку «VetCare»

Мобільний застосунок **«VetCare»** надає користувачам такі можливості:

– створення персонального профілю власника;

– перегляд графіку та можливість записатися на прийом до ветеринара;

– ведення інформації про тварину;

– отримання та створення власних нагадувань про важливі події (щеплення, прийом ліків, грумінг тощо);

– зміна мови та теми інтерфейсу;

Користувачами застосунку є власники домашніх тварин, які користуються смартфонами на платформі Android та IOS.

Застосунок може використовуватися на будь-якому пристрої з ОС Android версії 7.0 або вище, а також з ОС iOS версії 14 або вище.

3 Вимоги до затосунку

3.1 Функціональні вимоги

– реєстрація та авторизація: застосунок повинен надавати можливість реєстрації нових користувачів та авторизації існуючих. Для реєстрації необхідно ввести ім'я, електронну пошту, пароль та підтвердження пароля. Після реєстрації користувач повинен мати можливість увійти в систему за допомогою своєї електронної пошти та пароля. Доступна також можливість реєстрації та авторизації через Google.

– збереження даних користувача: вся інформація повинна зберігатися в обліковому записі користувача у Firebase для забезпечення конфіденційності та безпеки. Дані про профілі тварин, записи на прийом та історія відвідувань мають бути доступні користувачеві з будь-якого пристрою після авторизації.

– зміна інформації про користувача: користувач повинен мати змогу, оновити або видалити інформацію про себе.

– створення домашніх тварин: користувач має можливість створювати домашніх улюбленців, вказуючи ім’я, вид тварини, породу, вік,

– запис на прийом до ветеринара: користувач повинен мати можливість записатися на прийом до ветеринарної клініки, вибравши дату, час та лікаря. Перед записом повинна бути можливість переглянути доступні слоти для бронювання запису.

– нагадування: користувачі повині мати можливість створити та отримувати створені нагадування.

3.2 Нефункціональні вимоги

Застосунок VetCare повинен відповідати наступним вимогам до надійності:

– наявність реєстрації: процес реєстрації має бути швидким і простим, забезпечуючи можливість створення облікового запису за допомогою електронної пошти та пароля, а також через сторонні сервіси (google, соціальні мережі). реєстрація необхідна для збереження даних користувачів у хмарному сховищі firebase, що дозволяє синхронізувати інформацію між пристроями.

– введення інформації про тварин: користувач повинен мати можливість створювати профілі своїх домашніх улюбленців, додавати інформацію про ім’я, вік, породу, вагу, медичні дані та фото. Також повинна бути доступна можливість редагування чи видалення інформації про тварину у разі змін або потреби оновлення даних.

– наявність нагадувань: застосунок повинен дозволяти користувачам створювати та налаштовувати нагадування для важливих подій, пов’язаних із доглядом за тваринами. Це можуть бути нагадування про прийом ліків, заплановані візити до ветлікаря, час годування, грумінг чи прогулянки. користувачі повинні мати можливість вибору частоти нагадувань (одноразові, щоденні, щотижневі тощо), встановлення конкретного часу для отримання сповіщень, а також їх редагування або скасування.

– темна та світла тема: для забезпечення комфорту користувачів у різних умовах освітлення застосунок має підтримувати можливість перемикання між темною та світлою темою інтерфейсу. Ця функція має налаштовуватись вручну або автоматично відповідно до системних налаштувань пристрою.

– наявність локалізації: застосунок повинен підтримувати українську та англійську мови, з можливістю додавання інших мов у майбутньому. Вибір мови має бути доступним у налаштуваннях застосунку або визначатися автоматично на основі мовних налаштувань пристрою.

– захист персональних даних: усі дані користувачів, включаючи особисту інформацію та дані про тварин, повинні безпечно зберігатися в firestore та firebase storage. Передбачена підтримка шифрування даних та дотримання політики конфіденційності.

– обробка помилок: у разі виникнення системних або мережевих помилок користувач повинен отримувати зрозумілі повідомлення про проблему з поясненням можливих дій для її виправлення.

– блокування некоректних дій: система повинна запобігати неправомірним діям, таким як подвійне бронювання одного часу, спроби доступу до чужих даних або некоректне введення інформації (наприклад, введення нереальних дат чи некоректних форматів номерів телефону).

– оптимізація продуктивності: застосунок повинен працювати плавно та швидко навіть на пристроях із середніми технічними характеристиками, мінімізуючи затримки та використання ресурсів пристрою.

– контактна інформація для підтримки: у застосунку повинен бути розділ із контактами служби підтримки, через який користувачі можуть звернутися за допомогою у разі технічних проблем або питань щодо роботи застосунку.

3.3 Умови експлуатації

Умови експлуатації повинні відповідати санітарним і технічним вимогам використання мобільного пристрою, з урахуванням температурного режиму та рівня вологості, встановлених виробником гаджета. Застосунок підтримується на смартфонах та планшетах.

3.4 Вимоги до середовища виконання

Мобільний застосунок VetCare призначений для використання на смартфонах та планшетах, що працюють під управлінням операційних систем Android та iOS. На одному пристрої може бути встановлена лише одна копія застосунку.

Використання сторонніх засобів для клонування програм може призвести до некоректної роботи застосунку, і в таких випадках стабільність його функціонування не гарантується.

Для виконання більшості функцій (авторизація, запис до ветеринара, збереження даних у хмарному сховищі, отримання нагадувань) необхідне стабільне з'єднання з Інтернетом. Деякі функції (наприклад, перегляд профілю тварини, історії прийомів) можуть працювати в офлайн-режимі, але всі зміни будуть синхронізовані після підключення до мережі.

Застосунок розробляється з повноцінною підтримкою Android та iOS. Проте на момент першого випуску можлива різниця у функціональності між платформами, оскільки адаптація для iOS може потребувати додаткових оптимізацій.

Мінімальні системні вимоги для Android:

– операційна система: android 7.0 (nougat) або вище;

– частота процесора: від 1 ггц;

– оперативна пам'ять (ram): не менше 2 гб;

– вільне місце на пристрої: не менше 250 мб.

Мінімальні системні вимоги для iOS:

– операційна система: ios 13 або вище;

– підтримувані пристрої: iphone 8 / ipad (6-го покоління) або новіші;

– оперативна пам'ять (ram): не менше 2 гб;

– вільне місце на пристрої: не менше 250 мб;

Застосунок оптимізований для роботи на сучасних пристроях, але на старих моделях можлива знижена продуктивність або обмежений функціонал.

3.5 Вимоги до інформаційної та програмної сумісності

Мобільний застосунок VetCare буде розроблений на платформі Flutter з використанням BLoC (Business Logic of Component) для ефективного управління станом. Завдяки цьому застосунок забезпечить сумісність зі смартфонами та планшетами, що працюють під управлінням операційних систем Android та iOS.

– Android: підтримка Android 7.0 (Nougat) і вище;

– iOS: підтримка iOS 13 і вище (оптимізовано для iPhone та iPad).

Firebase в даному застосунку відіграє роль бази даних, яка зберігає облікові записи користувачів, інформацію про домашніх тварин, історію прийомів до ветеринара. Крім того, Firebase забезпечує збереження фотографій тварин, списку ветеринарних клінік, а також синхронізацію даних між пристроями користувача для забезпечення безперервного доступу до інформації.

3.6 Спеціальні вимоги

Вимоги до інтерфейсу:

− доступність білої та чорної теми;

− застосунок повинен мати навігаційне меню внизу екрану для швидкого переключення між екранами та візуальне відображення глибини (глибина взаємодії або рівень меню) та назви екрану.

4 Вимоги до програмної документації

Під час передачі проєкту замовнику надається наступний комплект документації:

– вихідний код програми з детальними коментарями;

– опис функціональних можливостей застосунку;

– технічне завдання (цей документ);

– інструкція для користувача.

5 Стадії та етапи розробки

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

Стадія розробки	Етапи робіт	Зміст робіт
Технічне завдання (01.02.2025 – 10.02.2025)	Обґрунтування необхідності розробки застосунку	Аналіз ринку та потреб користувачів, визначення аудиторії, формування вимог до функціональності, UI/UX та безпеки, вибір технологічного стека (Flutter, Firebase), визначення стадій та етапів розробки.
Ескізний проєкт (10.02.2025 – 10.03.2025)	Розробка ескізного проєкту	Проєктування архітектури застосунку, структури даних у Firestore, схеми взаємодії з Firebase Authentication та Firebase Storage, створення UX-макетів головних екранів.
Технічний проєкт (11.03.2025 – 30.03.2025)	Розробка технічного проєкту	Деталізація архітектури, створення схем збереження та отримання даних, проєктування основних алгоритмів.
Робочий проєкт (1.04.2025 – 20.04.2025)	Розробка застосунку	Реалізація основних функцій (авторизація, управління профілями тварин, запис на прийом), інтеграція Firebase, Firestore, Firebase Storage, відлагодження та оптимізація.
Розробка документації (21.04.2025 – 05.05.2025)	Створення документації	Опис архітектури та структури коду, документація бази даних, опис тестових сценаріїв.
Тестування системи (06.05.2025 – 20.05.2025)	Перевірка та виправлення помилок	Проведення функціонального, інтеграційного та E2E-тестування, корекція знайдених помилок та доопрацювання.

6 Порядок контролю та приймання:

Види випробувань:

– юніт-тестування: перевірка окремих частин коду, таких як бізнес-логіка, сервіси та утиліти.

– тестування компонентів (модульне): оцінка роботи окремих компонентів застосунку, зокрема екранів, віджетів та їх взаємодії з користувацьким інтерфейсом.

– інтеграційне тестування: перевірка взаємодії різних частин застосунку між собою та з зовнішніми сервісами (firebase authentication, firestore, firebase storage).

– тестування за знімками (snapshot testing): виявлення візуальних змін у відображенні ui-компонентів після оновлень.

– тестування з кінця в кінець (end-to-end testing, e2e): перевірка повної взаємодії користувача з застосунком, включаючи реєстрацію, додавання тварин, запис на прийом та отримання сповіщень.

– відмовостійкість: тестування стабільності та надійності застосунку за умов навантаження, різних рівнів мережевого сигналу та використання на різних пристроях.

Вимоги до приймання:

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

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

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

– взаємодія з сервісами: перевірка коректної роботи застосунку з firebase authentication, firestore, firebase storage, api пошуку ветеринарних клінік.

– безпека та конфіденційність: тестування захисту персональних даних, авторизації користувачів та безпечного зберігання інформації у firebase.

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

Технологічний стек:

– фреймворк: Flutter

– стан управління: BloC

– база даних та автентифікація: Firebase Authentication, Firestore

– зберігання медіафайлів: Firebase Storage

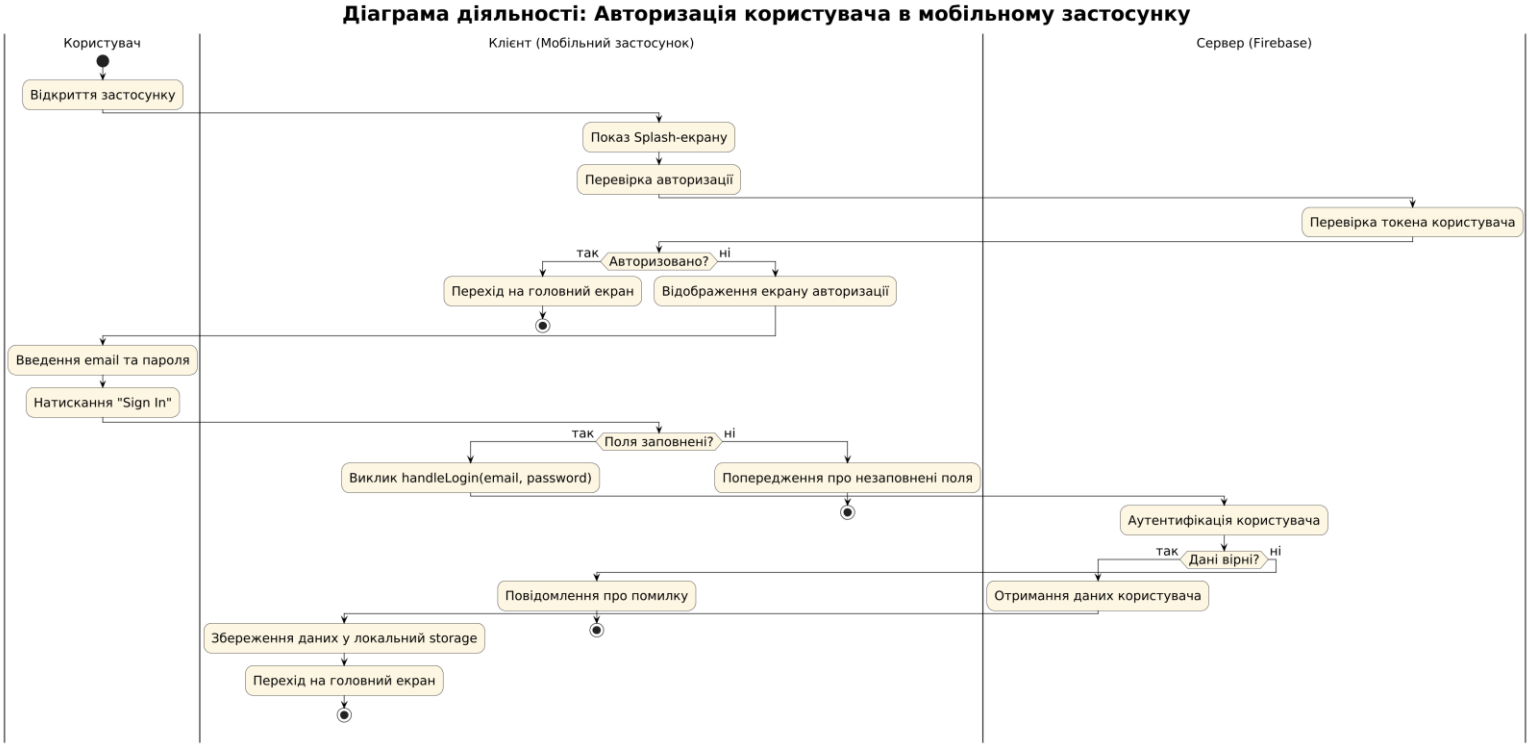
– тестування: flutter_test, patrol, mockito, bloc_test

– Аналітика та сповіщення: Firebase Analytics, Firebase Cloud Messaging (FCM), flutter_local_notiffications

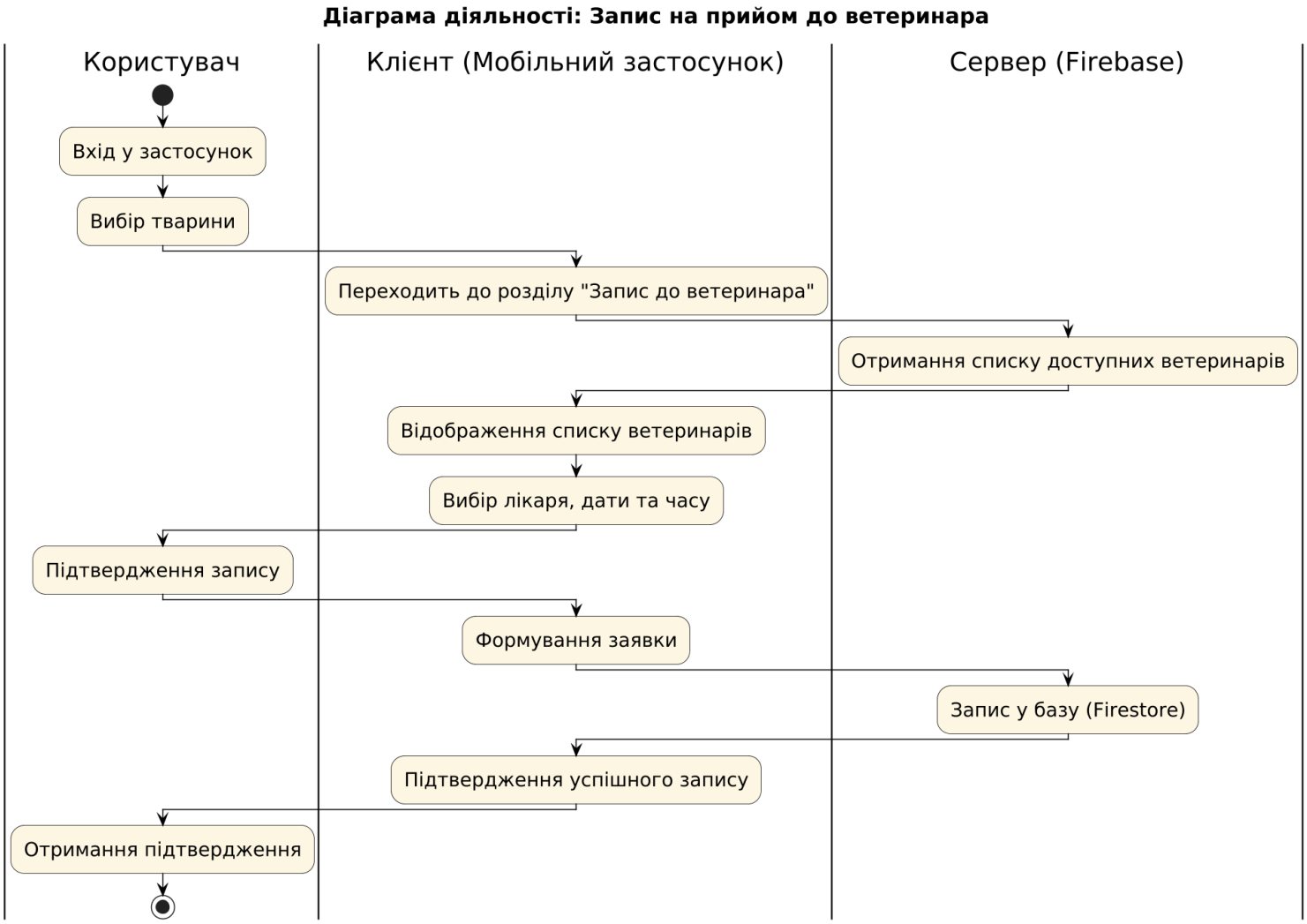
ДОДАТОК Б

(обов’язковий)

ДІАГРАМИ ДІЯЛЬНОСТІ



Додаток Б.1 – рисунок «Діаграма діяльності для процесу авторизації»



Додаток Б.2 – рисунок «Діаграма діяльності для запису на прийом до ветеринара»

ДОДАТОК В

(обов'язковий)

КОД ПРОГРАМИ

theme.dart

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
const _darkPrimaryColor = Color.fromARGB(255, 255, 30, 0);
const _lightPrimaryColor = Color.fromARGB(255, 19, 97, 135);
final darkTheme = ThemeData(
  useMaterial3: true,
  primaryColor: _darkPrimaryColor,
  scaffoldBackgroundColor: const Color.fromARGB(255, 17, 17, 20),
  colorScheme: ColorScheme.fromSeed(
    seedColor: _darkPrimaryColor,
    brightness: Brightness.dark,
    background: const Color.fromARGB(255, 17, 17, 20),
    surface: const Color.fromARGB(255, 28, 28, 32),
    onSurface: Colors.white,
    onBackground: Colors.white, ),
  cardColor: const Color.fromARGB(255, 28, 28, 32),
  dividerColor: Colors.white.withOpacity(0.05),
  iconTheme: const IconThemeData(color: Colors.white70),
  appBarTheme: const AppBarTheme(
    backgroundColor: Colors.transparent,
    scrolledUnderElevation: 0,
    elevation: 0,
    iconTheme: IconThemeData(color: Colors.white),
    titleTextStyle: TextStyle(color: Colors.white, fontSize: 20),
    systemOverlayStyle: SystemUiOverlayStyle(
      statusBarColor: Colors.transparent,
      statusBarIconBrightness: Brightness.light,
      statusBarBrightness: Brightness.light, ), ),);
final lightTheme = ThemeData(
  useMaterial3: true,
  primaryColor: _lightPrimaryColor,
  scaffoldBackgroundColor: Colors.white,
  colorScheme: ColorScheme.fromSeed(
    seedColor: _lightPrimaryColor,
    brightness: Brightness.light,
```

```
    background: Colors.white,
    surface: const Color(0xFFF9F9F9),
    onSurface: Colors.black87,
    onBackground: Colors.black87, ),
  cardColor: const Color(0xFFF9F9F9),
  dividerTheme: DividerThemeData(
    color: Colors.black.withOpacity(0.05), ),
  iconTheme: const IconThemeData(color: Colors.black54),
  snackBarTheme: const SnackBarThemeData(
    backgroundColor: _lightPrimaryColor,
    contentTextStyle: TextStyle(color: Colors.white), ),
  appBarTheme: const AppBarTheme(
    backgroundColor: Colors.transparent,
    scrolledUnderElevation: 0,
    elevation: 0,
    iconTheme: IconThemeData(color: Colors.black),
    titleTextStyle: TextStyle(color: Colors.black, fontSize: 20),
    systemOverlayStyle: SystemUiOverlayStyle(
      statusBarColor: Colors.transparent,
      statusBarIconBrightness: Brightness.dark,
      statusBarBrightness: Brightness.dark,), ),);
extension ThemeExtension on BuildContext {
  ColorScheme get theme => Theme.of(this).colorScheme;
}
```

app_initializer.dart

```
class AppInitializer extends StatelessWidget {
  const AppInitializer(
      {super.key,
      required this.child,
      required this.config,
      required this.repositoryContainer});
  final Widget child;
  final AppConfig config;
  final RepositoryContainer repositoryContainer;
  @override
  Widget build(BuildContext context) {
    return MultiRepositoryProvider(
        providers: [
          RepositoryProvider<AuthRepositoryInterface>(
              create: (context) => repositoryContainer.authRepository),
          RepositoryProvider<StorageRepositoryInterface>(
              create: (context) => repositoryContainer.storageRepository),
          RepositoryProvider<ProfileRepositoryInterface>(
```

```
            create: (context) => repositoryContainer.profileRepository),
        RepositoryProvider<PetRepositoryInterface>(
            create: (context) => repositoryContainer.petRepository),
        RepositoryProvider<DoctorRepositoryInterface>(
            create: (context) => repositoryContainer.doctorRepository),
        RepositoryProvider<AppointmentRepositoryInterface>(
            create: (context) => repositoryContainer.appointmentRepository),
        RepositoryProvider<SettingsRepositoryInterface>(
            create: (context) => repositoryContainer.settingsRepository),
        RepositoryProvider<NotificationsRepositoryInterface>(
            create: (context) => repositoryContainer.notificationsRepository),
        RepositoryProvider<ReminderRepositoryInterface>(
            create: (context) => repositoryContainer.reminderRepository), ],
      child: MultiBlocProvider(providers: [
        BlocProvider(
          create: (context) => AuthCubit(
              authRepository: context.read<AuthRepositoryInterface>()),),
        BlocProvider(
            create: (context) => ProfileCubit(
                profileRepository: context.read<ProfileRepositoryInterface>(),
                storageRepository:
                    context.read<StorageRepositoryInterface>())),
        BlocProvider(
            create: (context) => PetCubit(
                petRepository: context.read<PetRepositoryInterface>(),
                storageRepository:
                    context.read<StorageRepositoryInterface>())),
        BlocProvider(
            create: (context) => PetPdfCubit(
                petRepository: context.read<PetRepositoryInterface>(),
                storageRepository:
                    context.read<StorageRepositoryInterface>())),
        BlocProvider(
            create: (context) => DoctorCubit(
                  doctorRepository:
context.read<DoctorRepositoryInterface>(),  )),
        BlocProvider(
            create: (context) => AppointmentCubit(
                  appointmentRepository:
                      context.read<AppointmentRepositoryInterface>(),   )),
        BlocProvider(
            create: (context) => ThemeCubit(
                settingsRepository:
                    context.read<SettingsRepositoryInterface>())),
```

```
          BlocProvider(
              create: (context) => LanguageCubit(
                  settingsRepository:
                      context.read<SettingsRepositoryInterface>())),
          BlocProvider(
              create: (context) => ReminderCubit(
                  reminderRepository:
                      context.read<ReminderRepositoryInterface>())),
          BlocProvider(
            create: (context) => NotificationCubit(
              notificationRepository:
                  context.read<NotificationsRepositoryInterface>(), ),),
        ], child: child));}}
```

firebase_storage_repository.dart

```
import 'dart:io';
import 'package:firebase_storage/firebase_storage.dart';
import
'package:vet_clinic/core/storage/domain/storage_repository_interface.dart';
import 'package:http/http.dart' as http;
import 'package:permission_handler/permission_handler.dart';
import 'package:path_provider/path_provider.dart';
import 'package:path/path.dart' as p;
class FirebaseStorageRepository implements StorageRepositoryInterface {
  final FirebaseStorage storage = FirebaseStorage.instance;
  @override
  Future<String?> uploadProfileImage(String path, String fileName) {
    return _uploadFile(path, fileName, 'users_profile_images');}
  @override
  Future<String?> uploadPetImage(String path, String fileName) {
    return _uploadFile(path, fileName, 'pet_images');}
  @override
  Future<String?> uploadPetPdf(String path, String fileName) {
    return _uploadFile(path, fileName, 'pet_pdfs');}
  Future<String?> _uploadFile(
      String path, String fileName, String folder) async {
    try {
      final file = File(path);
      final storageRef = storage.ref().child('$folder/$fileName');
      final uploadTask = await storageRef.putFile(file);
      final downloadUrl = await uploadTask.ref.getDownloadURL();
      return downloadUrl;
    } catch (e) {
      return null;}}
```

```
  @override
  Future<File?> downloadPdfWithFirebase(String url, String fileName) async {
    try {
      if (Platform.isAndroid &&
          (await Permission.manageExternalStorage.status.isDenied)) {
        final status = await Permission.manageExternalStorage.request();
        if (!status.isGranted) {
          await openAppSettings();
          return null; }}
      final response = await http.get(Uri.parse(url));
      if (response.statusCode != 200) {
        return null;}
      Directory? downloadsDir;
      if (Platform.isAndroid) {
        downloadsDir = Directory('/storage/emulated/0/Download');} else {
        downloadsDir = await getApplicationDocumentsDirectory();}
      final fullPath = p.join(downloadsDir.path, "$fileName.pdf");
      final file = File(fullPath);
      await file.writeAsBytes(response.bodyBytes);
      return file;
    } catch (e) {return null;}}
  @override
  Future<void> deletePetPdf(String filePath) async {
    try {
      print("Deleting from Firebase Storage: $filePath");
      final ref = storage.ref().child(filePath);
      await ref.delete();} catch (e) {
      print("Error deleting PDF: $e");}}}
```

auth_firebase_repository.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:google_sign_in/google_sign_in.dart';
import 'package:vet_clinic/core/config/applocation_images.dart';
import 'package:vet_clinic/feature/auth/domain/entities/profile_user.dart';
import
'package:vet_clinic/feature/auth/domain/repos/auth_repository_interface.dart';
class AuthFirebaseRepository implements AuthRepositoryInterface {
  final FirebaseAuth firebaseAuth = FirebaseAuth.instance;
  final FirebaseFirestore firebaseFirestore = FirebaseFirestore.instance;
  final GoogleSignIn googleSignIn = GoogleSignIn();
  @override
  Future<ProfileUser?> loginWithEmailAndPassword(String email, String password)
async {
```

```
    try {
      var cred = await firebaseAuth.signInWithEmailAndPassword(email: email,
password: password);
      return ProfileUser(uid: cred.user!.uid, email: email, name: '', password:
'', phone: '', bio: '', profileImageUrl: '');} catch (e) {
      throw Exception('Failed + $e'); }}
  @override
  Future<ProfileUser?> registerWithEmailAndPassword(String email, String name,
String password) async {try {
      var cred = await firebaseAuth.createUserWithEmailAndPassword(email: email,
password: password);
      var user = ProfileUser(uid: cred.user!.uid, email: email, name: name,
password: password, phone: '', bio: '', profileImageUrl:
ApplicationImages.defaultUserImage);
  await firebaseFirestore.collection("Users").doc(user.uid).set(user.toJson());
      return user; catch (e) {
      throw Exception('Failed + $e');}}
  @override
  String getCurrentUserId() => firebaseAuth.currentUser?.uid ?? '';
  @override
  Future<ProfileUser?> getCurrentUser() async {
    var u = firebaseAuth.currentUser;
    if (u == null) return null;
    return ProfileUser(uid: u.uid, email: u.email!, name: '', password: '',
phone: '', bio: '', profileImageUrl: '');}
  @override
  Future<void> logout() async => await firebaseAuth.signOut();
  @override
  Future<ProfileUser?> loginWithGoogle() async {
    try {var googleUser = await googleSignIn.signIn();
      if (googleUser == null) return null;
      var googleAuth = await googleUser.authentication;
      var cred = GoogleAuthProvider.credential(accessToken:
googleAuth.accessToken, idToken: googleAuth.idToken);
      var userCred = await firebaseAuth.signInWithCredential(cred);
      var u = userCred.user;
      if (u == null) return null;
      var doc = await firebaseFirestore.collection("Users").doc(u.uid).get();
      var user = ProfileUser(uid: u.uid, email: u.email!, name: u.displayName ??
u.email!, password: '', phone: '', bio: '', profileImageUrl: u.photoURL ??
ApplicationImages.defaultUserImage);if (!doc.exists) await
firebaseFirestore.collection("Users").doc(user.uid).set(user.toJson());
      return user;} catch (e) {
      throw Exception('Failed to sign in with Google: $e'); }}}
```

notifications_repositore.dart

```
import 'package:firebase_messaging/firebase_messaging.dart';
import 'package:flutter_local_notifications/flutter_local_notifications.dart';
import
'package:vet_clinic/feature/notifications/domain/entities/notification.dart';
import
'package:vet_clinic/feature/notifications/domain/repos/notification_repository_i
nterface.dart';
import 'package:vet_clinic/feature/reminder/model/reminder.dart';
import 'package:timezone/timezone.dart' as tz;
class NotificationsRepository implements NotificationsRepositoryInterface {
  NotificationsRepository({
    required FlutterLocalNotificationsPlugin localNotifications,
    required FirebaseMessaging firebaseMessaging,
  })  : _localNotifications = localNotifications,
        _firebaseMessaging = firebaseMessaging;
  final FlutterLocalNotificationsPlugin _localNotifications;
  final FirebaseMessaging _firebaseMessaging;
  static const _defaultChannel = AndroidNotificationChannel(
    'high_importance_channel',
    'High Importance Notifications',
    description: 'This channel is used for important notifications.',
    importance: Importance.high,);
  @override
  Future<String?> getToken() => _firebaseMessaging.getToken();
  @override
  Future<bool> requestPermisison() async {
    final settings = await _firebaseMessaging.requestPermission();
    final isAuthorized =
        settings.authorizationStatus == AuthorizationStatus.authorized;
    if (isAuthorized) {
      await FirebaseMessaging.instance
          .setForegroundNotificationPresentationOptions(
        alert: true,
        badge: true,
        sound: true,  ); }
    return isAuthorized;}
  @override
  Future<void> init() async {
    final androidPlugin =
        _localNotifications.resolvePlatformSpecificImplementation<
            AndroidFlutterLocalNotificationsPlugin>();
    if (androidPlugin != null) {
      await androidPlugin.createNotificationChannel(_defaultChannel);}
```

```
  _localNotifications.resolvePlatformSpecificImplementation<
          AndroidFlutterLocalNotificationsPlugin>()
      ?.requestNotificationsPermission();
  FirebaseMessaging.onMessage.listen((RemoteMessage message) {
    final notification = message.notification;
    final android = message.notification?.android;
    if (notification != null && android != null) {
      showLocalNotification( Notification(
          title: notification.title!,
          message: notification.body!, ),);}}); }
@override
Future<void> showLocalNotification(Notification notification) async {
  await _localNotifications.show(
    notification.hashCode,
    notification.title,
    notification.message,
    NotificationDetails(
      android: AndroidNotificationDetails(
        _defaultChannel.id,
        _defaultChannel.name,
        channelDescription: _defaultChannel.description,
        icon: 'ic_launcher',),),);}
@override
Future<void> scheduleNotification(Reminder reminder) async {
  final scheduledTime = tz.TZDateTime.from(reminder.dateTime, tz.local);
  await _localNotifications.zonedSchedule(
    reminder.reminderId.hashCode,
    reminder.title,
    'It\'s time for: ${reminder.title}',
    scheduledTime,
    NotificationDetails(
      android: AndroidNotificationDetails(
        _defaultChannel.id,_defaultChannel.name,
        channelDescription: _defaultChannel.description,
        icon: 'ic_launcher',
      ),),
    matchDateTimeComponents: DateTimeComponents.time,
    androidScheduleMode: AndroidScheduleMode.exactAllowWhileIdle,
    uiLocalNotificationDateInterpretation:
        UILocalNotificationDateInterpretation.absoluteTime,);}
@override
Future<void> cancelNotification(int reminderId) async {
  try {
    await _localNotifications.cancel(reminderId.hashCode);
```

```
    } catch (e) {}}}
```

firebase_pet_repository.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:uuid/uuid.dart';
import 'package:vet_clinic/core/config/applocation_images.dart';
import 'package:vet_clinic/feature/pets/domain/entities/pet.dart';
import
'package:vet_clinic/feature/pets/domain/repos/pet_repository_interface.dart';
class FirebasePetRepository implements PetRepositoryInterface {
  final FirebaseAuth firebaseAuth = FirebaseAuth.instance;
  final FirebaseFirestore firebaseFirestore = FirebaseFirestore.instance;
  final FirebaseStorage firebaseStorage = FirebaseStorage.instance;
  @override
  Future<Pet> createPet(String uid, Pet pet) async {
    try {final petId = const Uuid().v4();
      final newPet = Pet(
        petId: petId,
        uid: uid,
        name: pet.name,
        breed: pet.breed,
        gender: pet.gender,
        age: pet.age,
        petImageUrl: pet.petImageUrl ?? ApplicationImages.petDefaultImage,
        petPdfUrl: pet.petPdfUrl ?? "", );
      await firebaseFirestore
          .collection('Users')
          .doc(uid)
          .collection('Pets')
          .doc(petId)
          .set(newPet.toJson());return newPet;
    } catch (e) {throw Exception('Не вдалося створити тварину: $e'); } }
  @override
  Future<void> deletePet(String uid, String petId) async {
    try { await firebaseFirestore.collection('Users')
          .doc(uid)
          .collection('Pets')
          .doc(petId)
          .delete();
      final appointmentsSnapshot = await firebaseFirestore
          .collection('Users').doc(uid)
          .collection('Appointments').where('petId', isEqualTo: petId)
```

```
        .get();
    for (var doc in appointmentsSnapshot.docs) {
      await doc.reference.delete();}
  } catch (e) { throw Exception(e.toString());}}
@override
Future<List<Pet>> fetchAllPets(String uid) async {try {
    final snapshot = await firebaseFirestore
        .collection("Users")
        .doc(uid)
        .collection("Pets")
        .get();
    if (snapshot.docs.isEmpty) {   return [];}
    final pets = snapshot.docs.map((doc) {
      return Pet.fromJson(doc.data());}).toList();
    return pets; } catch (e) {
    throw Exception(e.toString());}}
@override
Future<void> updatePet(Pet updatePet, String uid) async { try {
    await firebaseFirestore
        .collection("Users")
        .doc(uid)
        .collection("Pets")
        .doc(updatePet.petId)
        .update({
      'name': updatePet.name,
      'breed': updatePet.breed,
      'gender': updatePet.gender,
      'age': updatePet.age,
      'petImageUrl': updatePet.petImageUrl,
      'petPdfUrl': updatePet.petPdfUrl,
    }); } catch (e) {
    throw Exception(e.toString());} }
@override
Future<Pet> fetchPet(String uid, String petId) async {try {
    final snapshot = await firebaseFirestore
        .collection("Users")
        .doc(uid)
        .collection("Pets")
        .doc(petId)
        .get();
    if (!snapshot.exists) { throw Exception("Pet not found"); }
    final petData = snapshot.data();
    if (petData == null) {
      throw Exception("Pet data is null");}
```

```
      final pet = Pet(
        petId: petId,
        uid: uid,
        name: petData['name'],
        breed: petData['breed'],
        gender: petData['gender'],
        age: petData['age'],
        petImageUrl: petData['petImageUrl'],
        petPdfUrl: petData['petPdfUrl'], );
      return pet;} catch (e) {throw Exception("Failed to fetch pet:
${e.toString()}");}}}
```

firebase_profile_repository.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:vet_clinic/feature/auth/domain/entities/profile_user.dart';
import
'package:vet_clinic/feature/profile/domain/repos/profile_repository_interface.da
rt';
class FirebaseProfileRepository implements ProfileRepositoryInterface {
  final FirebaseFirestore firebaseFirestore = FirebaseFirestore.instance;
  @override
  Future<ProfileUser?> fetchUserProfile(String uid) async {try {
      final userDoc =
          await firebaseFirestore.collection("Users").doc(uid).get();
      if (userDoc.exists) { final userData = userDoc.data();
        if (userData != null) {
          return ProfileUser(
              uid: uid,
              name: userData['name'],
              email: userData['email'],
              bio: userData['bio'] ?? '',
              phone: userData['phone'] ?? '',
              profileImageUrl: userData['profileImageUrl'].toString(),
              password: userData['password']);}}
      return null;} catch (e) {return null;}}
  @override
  Future<void> updateProfile(ProfileUser updatedProfile) async {try {
      await firebaseFirestore
          .collection("Users")
          .doc(updatedProfile.uid)
          .update({
        'name': updatedProfile.name,
        'bio': updatedProfile.bio,
```

```
        'phone': updatedProfile.phone,
        'password': updatedProfile.password,
        'profileImageUrl': updatedProfile.profileImageUrl,});
    } catch (e) { throw Exception();}}}
```

reminder_repository.dart

```
import 'package:vet_clinic/feature/reminder/model/reminder.dart';
import 'package:vet_clinic/feature/reminder/reminder_repository_interface.dart';
import
'package:vet_clinic/feature/reminder/sqflite%20Database%20helper/connection.dart
';
import
'package:vet_clinic/feature/reminder/sqflite%20Database%20helper/tables.dart';
class ReminderRepository implements ReminderRepositoryInterface {
  final DatabaseHelper databaseHelper;
  ReminderRepository({required this.databaseHelper});
  @override
  Future<Reminder> createReminder(Reminder reminder) async {
    final db = await databaseHelper.initDatabase();
    final id = await db.insert(Tables.remindersTableName, reminder.toJson());
    return reminder.copyWith(reminderId: id); }
  @override
  Future<void> deleteReminder(int reminderId) async {
    final db = await databaseHelper.initDatabase();
    await db.delete(
      Tables.remindersTableName,
      where: 'reminderId = ?',
      whereArgs: [reminderId], );}
  @override
  Future<List<Reminder>> fetchReminders() async {
    final db = await databaseHelper.initDatabase();
    final maps = await db.query(Tables.remindersTableName);
    return maps.map((map) => Reminder.fromJson(map)).toList();}
  @override
  Future<void> updateReminderTitleAndDateTime(
    int reminderId, String title, DateTime dateTime,) async {
    final db = await databaseHelper.initDatabase();
    await db.update(  Tables.remindersTableName,{
        'title': title,'dateTime': dateTime.toIso8601String(),},
      where: 'reminderId = ?', whereArgs: [reminderId], );}}
```

settings_repository.dart

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

```
import
'package:vet_clinic/feature/settings/domain/repos/settings_repository_interface.
dart';
class SettingsRepository implements SettingsRepositoryInterface {
  final SharedPreferences preferences;
  SettingsRepository({required this.preferences});
  static const _isDarkThemeSelectedKey = "darh_theme_selcted";
  static const _isUkrainianLanguageKey = "ukrainian_language_selected";
  @override
  bool isDarkThemeSelected() {
    final selectedTheme = preferences.getBool(_isDarkThemeSelectedKey);
    return selectedTheme ?? false;}
  @override
  Future<void> setDarkThemeSelected(bool selected) async {
    preferences.setBool(_isDarkThemeSelectedKey, selected);}
  @override
  bool isUkrainianLanguageSelected() {
    final selectedLanguage = preferences.getBool(_isUkrainianLanguageKey);
    return selectedLanguage ?? false;}
  @override
  Future<void> setUkrainianLanguage(bool selectedLanguage) async {
    preferences.setBool(_isUkrainianLanguageKey, selectedLanguage);}}
```

login_screen.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:vet_clinic/common/widgets/custom_square_elevated_button.dart';
import 'package:vet_clinic/common/widgets/theme_logo.dart';
import 'package:vet_clinic/core/theme/theme.dart';
import 'package:vet_clinic/feature/auth/presentation/cubits/auth_cubit.dart';
import 'package:vet_clinic/feature/auth/presentation/widgets/auth_field.dart';
import 'package:vet_clinic/generated/l10n.dart';

class LoginScreen extends StatefulWidget {
  const LoginScreen({super.key, required this.onTap});
  final void Function()? onTap;
  @override
  State<LoginScreen> createState() => _LoginScreenState();}
class _LoginScreenState extends State<LoginScreen> {
  final TextEditingController emailController = TextEditingController();
  final TextEditingController passwordController = TextEditingController();
  void login() {
    final String email = emailController.text;
    final String password = passwordController.text;
```

```
    final authCubit = context.read<AuthCubit>();
    if (email.isNotEmpty && password.isNotEmpty) {
      authCubit.login(email, password);} else {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(S.of(context).pleaseEnterBothEmailAndPassword)),
); } }
  void loginWithGoogle() {
    final authCubit = context.read<AuthCubit>();
    authCubit.loginWithGoogle();}
  @override
  void dispose() {
    emailController.dispose();
    passwordController.dispose();
    super.dispose(); }
  @override
  Widget build(BuildContext context) {
    final theme = context.theme;
    return Scaffold(
      body: SafeArea(
        child: SingleChildScrollView(
          child: Padding(
            padding: const EdgeInsets.symmetric(horizontal: 15),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                const ThemedLogo(),
                Text(
                  S.of(context).login,
                  style: TextStyle(
                      fontSize: 36,
                      fontWeight: FontWeight.bold,
                      color: theme.primary), ),
                const SizedBox(height: 20),
                AuthField(
                  key: const Key('emailField'),
                  hintText: S.of(context).email,
                  controller: emailController, ),
                const SizedBox(height: 20),
                AuthField(
                  key: const Key('passwordField'),
                  hintText: S.of(context).password,
                  obscureText: true,
                  controller: passwordController, ),
```

```
                const SizedBox(height: 20),
                CustomSquareElevatedButton(
                  text: S.of(context).login,
                  onTap: login,  ),
                const SizedBox(height: 20),
                _RegisterRow( widget: widget,theme: theme,),
                const SizedBox(height: 15),
                _Divider(theme: theme),
                const SizedBox(height: 15),
                socialRowLogin(theme),],),),),),),);}
  Row socialRowLogin(ColorScheme theme) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        GestureDetector(
          onTap: loginWithGoogle,
          child: CircleAvatar(
            radius: 30,
            backgroundColor: theme.primary.withOpacity(0.1),
            child: Image.asset(
              "images/google.png",
              width: 32,
              height: 32,
              fit: BoxFit.cover,),),),],);}}
class _Divider extends StatelessWidget {
  const _Divider({required this.theme,});
  final ColorScheme theme;
  @override
  Widget build(BuildContext context) {
    return Row(children: [Expanded(child: Divider(color: theme.primary,)),
        const Padding(
          padding: EdgeInsets.symmetric(horizontal: 15),
          child: Text("or login with",
            style: TextStyle(color: Colors.grey),),),
        Expanded(child: Divider(color: theme.primary)),],);}}
class _RegisterRow extends StatelessWidget {
  const _RegisterRow({
    required this.widget,
    required this.theme,});
  final LoginScreen widget;
  final ColorScheme theme;
  @override
  Widget build(BuildContext context) {
    return Row(
```

```
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Text(S.of(context).dontHaveAnAccount,
          style: const TextStyle(fontWeight: FontWeight.w700, fontSize: 16),),
        const SizedBox(width: 20),
        GestureDetector(
          onTap: widget.onTap,
          child: Text(
            S.of(context).register,
            style: TextStyle(
                fontSize: 16,
                fontWeight: FontWeight.bold,
                color: theme.primary),),),],);}}
```

register_screen.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:vet_clinic/common/widgets/custom_square_elevated_button.dart';
import 'package:vet_clinic/common/widgets/theme_logo.dart';
import 'package:vet_clinic/core/theme/theme.dart';
import 'package:vet_clinic/feature/auth/presentation/cubits/auth_cubit.dart';
import 'package:vet_clinic/feature/auth/presentation/widgets/auth_field.dart';
import 'package:vet_clinic/generated/l10n.dart';
class RegisterScreen extends StatefulWidget {const RegisterScreen({super.key,
required this.onTap});final void Function()? onTap;
  @override
  State<RegisterScreen> createState() => _RegisterScreenState();}
class _RegisterScreenState extends State<RegisterScreen> {
  final nameController = TextEditingController();
  final emailController = TextEditingController();
  final passwordController = TextEditingController();
  void register() async {
    final String email = emailController.text.trim();
    final String name = nameController.text.trim();
    final String password = passwordController.text.trim();
    final emailRegExp =
        RegExp(r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$");
    final passwordRegExp =
        RegExp(r"^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d@$!%*?&]{8,}$");
    if (email.isNotEmpty && name.isNotEmpty && password.isNotEmpty) {
      if (!emailRegExp.hasMatch(email)) {
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(content: Text(S.of(context).invalidEmailAddress)),);
        return;}
```

```
    if (!passwordRegExp.hasMatch(password)) {
      ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text(
   S.of(context).thePasswordMustBeAtLeast8CharactersLongAnd)),);return;}
    BlocProvider.of<AuthCubit>(context).register(email, name, password);
  } else {ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(content: Text(S.of(context).pleaseFillOutTheFields)),);}}
@override
void dispose() {
  emailController.dispose();
  nameController.dispose();
  passwordController.dispose();
  super.dispose();}
@override
Widget build(BuildContext context) {
  final theme = context.theme;
  return Scaffold(
    resizeToAvoidBottomInset: true,
    body: SafeArea(
      child: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal: 15),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              const ThemedLogo(),
              Text(
                S.of(context).register,
                style: TextStyle(
                  fontSize: 36,
                  fontWeight: FontWeight.bold,
                  color: theme.primary,),),
              const SizedBox(height: 20),
              AuthField(
                key: const Key('nameField'),
                hintText: S.of(context).name,
                controller: nameController,),
              const SizedBox(height: 20),
              AuthField(
                key: const Key('emailField'),
                hintText: S.of(context).email,
                controller: emailController,),
              const SizedBox(height: 20),
              AuthField(
                key: const Key('passwordField'),
```

```
                  hintText: S.of(context).password,
                  obscureText: true,
                  controller: passwordController,),
                const SizedBox(height: 20),
                CustomSquareElevatedButton(
                  text: S.of(context).register,
                  onTap: register,),
                const SizedBox(height: 10),
                _LoginRow(widget: widget, theme: theme),
                const SizedBox(height: 10),
                _Divider(theme: theme),
                const SizedBox(height: 10),
                socialRowRegister(theme),],),),),),);}
  Row socialRowRegister(ColorScheme theme) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        GestureDetector(
          onTap: register,
          child: CircleAvatar(
            radius: 30,
            backgroundColor: theme.primary.withOpacity(0.1),
            child: Image.asset(
              "images/google.png",
              width: 32,
              height: 32,
              fit: BoxFit.cover,),),),],);}}
class _Divider extends StatelessWidget {
  const _Divider({required this.theme});
  final ColorScheme theme;
  @override
  Widget build(BuildContext context) {
    return Row(
      children: [
        Expanded(child: Divider(color: theme.primary)),
        const Padding(
          padding: EdgeInsets.symmetric(horizontal: 15),
          child: Text(
            "or register with",
            style: TextStyle(color: Colors.grey),),),
        Expanded(child: Divider(color: theme.primary)),],);}}
class _LoginRow extends StatelessWidget {
  const _LoginRow({required this.widget, required this.theme});
  final ColorScheme theme;
```

```
  final RegisterScreen widget;
  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      children: [
        Text(
          S.of(context).alreadyHaveAnAccount,
          style: const TextStyle(fontWeight: FontWeight.w700, fontSize: 16),),
        GestureDetector(
          onTap: widget.onTap,
          child: Text(
            S.of(context).login,
            style: TextStyle(
              color: theme.primary,
              fontSize: 16,
              fontWeight: FontWeight.bold,),),),],);}}
```

doctors_screen.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import
'package:material_design_icons_flutter/material_design_icons_flutter.dart';
import 'package:vet_clinic/common/widgets/loader.dart';
import 'package:vet_clinic/common/widgets/search_field.dart';
import 'package:vet_clinic/feature/auth/presentation/cubits/auth_cubit.dart';
import 'package:vet_clinic/feature/doctors/domain/entities/doctor.dart';
import
'package:vet_clinic/feature/doctors/presentation/cubits/doctor_cubit.dart';
import
'package:vet_clinic/feature/doctors/presentation/cubits/doctor_state.dart';
import
'package:vet_clinic/feature/doctors/presentation/screens/doctor_details_screen.d
art';
import 'package:vet_clinic/feature/doctors/presentation/widgets/category.dart';
import
'package:vet_clinic/feature/doctors/presentation/widgets/doctor_card.dart';
import 'package:vet_clinic/generated/l10n.dart';
class DoctorsScreen extends StatefulWidget {
  const DoctorsScreen({super.key});
  @override
  State<DoctorsScreen> createState() => _DoctorsScreenState();}
class _DoctorsScreenState extends State<DoctorsScreen> {
  String? uid;
```

```
final TextEditingController searchController = TextEditingController();
String? selectedCategory;
late Map<String, String> categoryMap;
@override
void initState() {super.initState();
  uid = BlocProvider.of<AuthCubit>(context).getCurrentUid();_fetchDoctors();}
@override
void didChangeDependencies() {
  super.didChangeDependencies();
  _initializeCategoryMap();}
void _fetchDoctors() {
  BlocProvider.of<DoctorCubit>(context).fetchDoctors();}
void _initializeCategoryMap() {
  categoryMap = {
    S.of(context).dental: "dental",
    S.of(context).heart: "heart",
    S.of(context).eye: "eye",
    S.of(context).brain: "brain",
    S.of(context).ear: "ear",
    S.of(context).hand: "hand",};}
void _onSearch(String query) {
  String normalizedQuery = categoryMap[query] ?? query;
  BlocProvider.of<DoctorCubit>(context).searchDoctors(normalizedQuery);}
void _onCategorySelected(String category) {
  setState(() {
    selectedCategory = (selectedCategory == category) ? null : category;});
  String? categoryKey =
      selectedCategory != null ? categoryMap[selectedCategory] : null;
  BlocProvider.of<DoctorCubit>(context)
      .filterDoctorsByCategory(categoryKey ?? '');}
@override
Widget build(BuildContext context) {
  List<String> typeDoctor = categoryMap.keys.toList();
  List<IconData> typeIcons = [
    MdiIcons.toothOutline,
    MdiIcons.heartPlus,
    MdiIcons.eye,
    MdiIcons.brain,
    MdiIcons.earHearing,
    MdiIcons.handClap,];
  return Scaffold(      body: Column(
      children: [
        SearchField(onChanged: _onSearch),
        _BuildSymptoms(
```

```
            typeDoctor: typeDoctor,
            typeIcons: typeIcons,
            selectedCategory: selectedCategory,
            onCategorySelected: _onCategorySelected,),
          Expanded(
            child: BlocBuilder<DoctorCubit, DoctorState>(
              builder: (context, state) {
                if (state is DoctorLoaded) {
                  return _BuildDoctorGridView(
                      doctors: state.filteredDoctors, uid: uid!);
                } else if (state is DoctorLoading) {
                  return const Loader();
                } else if (state is DoctorEmpty) {
                  return Center(
                    child: Text(
                      state.text,style: const TextStyle(
                          fontSize: 18,
                          fontWeight: FontWeight.bold,
                          color: Colors.grey),
                      textAlign: TextAlign.center,),);} else {return const
Center(child: Text("Some error"));}},),),],),);}}
class _BuildDoctorGridView extends StatelessWidget {
  final List<Doctor> doctors;
  final String uid;
  const _BuildDoctorGridView({required this.doctors, required this.uid});
  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.symmetric(horizontal: 10),
      child: GridView.builder(
        padding: EdgeInsets.zero,
        itemCount: doctors.length,
        shrinkWrap: true,
        physics: const BouncingScrollPhysics(),
        gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: 2,
          mainAxisSpacing: 4,
          crossAxisSpacing: 5,
          childAspectRatio: 0.8,),
        itemBuilder: (context, index) {
          final doctor = doctors[index];
          return DoctorCard(
            onTap: () {
              Navigator.push(context,
```

```
     MaterialPageRoute(builder: (context) =>  DoctorDetailsScreen(doctor:
doctor,)),);},image: doctor.docImagePath,
            name: doctor.name,
            rating: "4.5",
            major: doctor.major,);},),);}}
class _BuildSymptoms extends StatelessWidget {
  final List<String> typeDoctor;
  final List<IconData> typeIcons;
  final String? selectedCategory;
  final Function(String) onCategorySelected;
  const _BuildSymptoms({
    required this.typeDoctor,
    required this.typeIcons,
    required this.selectedCategory,
    required this.onCategorySelected,});
  @override
  Widget build(BuildContext context) {
    return SizedBox(
      height: 120,
      child: ListView.builder(
        scrollDirection: Axis.horizontal,
        shrinkWrap: true,
        itemCount: typeDoctor.length,
        itemBuilder: (context, index) {
          final category = typeDoctor[index];
          return Category(
            icon: typeIcons[index],
            title: category,
            isSelected: category == selectedCategory,
            onTap: () => onCategorySelected(category),);},),);}}
```

appointment_screen.dart

```
import 'package:cached_network_image/cached_network_image.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:table_calendar/table_calendar.dart';
import 'package:uuid/uuid.dart';
import 'package:vet_clinic/core/config/applocation_images.dart';
import 'package:vet_clinic/common/widgets/custom_circle_elavated_button.dart';
import 'package:vet_clinic/core/theme/theme.dart';
import 'package:vet_clinic/feature/auth/presentation/cubits/auth_cubit.dart';
import 'package:vet_clinic/feature/doctors/domain/entities/doctor.dart';
import 'package:vet_clinic/feature/pets/presentation/cubits/pet/pet_cubit.dart';
```

```
import 'package:vet_clinic/feature/pets/presentation/cubits/pet/pet_state.dart';
import
'package:vet_clinic/feature/schedule/presentation/cubits/appointment_cubit.dart'
;
import
'package:vet_clinic/feature/schedule/presentation/cubits/appointment_state.dart'
;
import 'package:vet_clinic/generated/l10n.dart';

class AppointmentScreen extends StatefulWidget {
  const AppointmentScreen({super.key, required this.doctor});
  final Doctor doctor;
  @override
  State<AppointmentScreen> createState() => _AppointmentScreenState();}
class _AppointmentScreenState extends State<AppointmentScreen> {
  CalendarFormat _format = CalendarFormat.month;
  DateTime _focusDay = DateTime.now();
  DateTime _currentDay = DateTime.now();
  int? _currentIndex;
  bool _isWeekend = false;
  bool _dateSelected = true;
  bool _timeSelected = false;
  bool _isLoadingSlots = false;
  String? _selectedAnimal;
  String? _selectedAnimalId;
  String? _petImage;
  List<DateTime> _bookedSlots = [];
  String? uid;
  bool _isUidInitialized = false;
  @override
  void didChangeDependencies() {
    super.didChangeDependencies();
    if (!_isUidInitialized) {
      uid = BlocProvider.of<AuthCubit>(context).getCurrentUid();
      BlocProvider.of<PetCubit>(context).fetchPets(uid!);
      _isUidInitialized = true;}}
  void createAppointment() async {
    final selectedHour = _currentIndex! + 9;
    final combinedDateTime = DateTime(
      _currentDay.year,
      _currentDay.month,
      _currentDay.day,
      selectedHour,);
```

```
    final isBooked = _bookedSlots.any((slot) {
      return slot.hour == combinedDateTime.hour &&
          slot.day == combinedDateTime.day;});
    if (isBooked) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
            content: Text(S.of(context).theSelectedTimeSlotIsAlreadyBooked)),);
      return;}
    final dateTime = Timestamp.fromDate(combinedDateTime);
    final docId = widget.doctor.docId;
    final petId = _selectedAnimalId;
    final appointId = const Uuid().v4();
    const status = 'upcoming';
    BlocProvider.of<AppointmentCubit>(context).createAppointment(
      appointId,status,dateTime,uid!,docId,petId!,);}
  @override
  Widget build(BuildContext context) {
    final theme = context.theme;
    return Scaffold(
      appBar: AppBar(),
      body: BlocListener<AppointmentCubit, AppointmentState>(
        listener: (context, state) {
          if (state is AppointmentSlotsLoaded) {
            setState(() {
              _bookedSlots = state.slots;
              _isLoadingSlots = false;});}},
        child: CustomScrollView(
          slivers: [
            SliverToBoxAdapter(
              child: Column(
                children: [
                  _tableCalendar(context),
                  Padding(
                    padding:
const EdgeInsets.symmetric(horizontal: 10, vertical: 10),child: Center(
                      child: Text(S.of(context).selectTime,style: const
TextStyle(fontWeight: FontWeight.bold, fontSize: 20),),),),],),),
            _isWeekend
                ? _weekendMessage()
                : _isLoadingSlots
                    ? SliverToBoxAdapter(
                        child: Center(
                          child: Padding(
                            padding: const EdgeInsets.all(20.0),
```

```
                          child: CircularProgressIndicator(
                            color: theme.primary,),),),)
                  : _timeSelectionGrid(theme),
          if (_timeSelected) _animalSelection(),
          _makeAppointmentButton(),],),),);}
Widget _tableCalendar(BuildContext context) {
  final theme = Theme.of(context).colorScheme;
  return TableCalendar(
    focusedDay: _focusDay,
    firstDay: DateTime.now().subtract(const Duration(days: 30)),
    lastDay: DateTime.now().add(const Duration(days: 365)),
    calendarFormat: _format,
    currentDay: _currentDay,
    rowHeight: 48,
    calendarStyle: CalendarStyle(
      selectedDecoration: BoxDecoration(
        color: theme.primary,
        shape: BoxShape.circle,),),
    availableCalendarFormats: const {CalendarFormat.month: 'Month'},
    onFormatChanged: (format) {
      setState(() {
        _format = format;});},
    onDaySelected: (selectedDay, focusedDay) {
      setState(() {
        _currentDay = selectedDay;
        _focusDay = focusedDay;
        _dateSelected = true;
        _isLoadingSlots = true;
        if (selectedDay.weekday == 6 || selectedDay.weekday == 7) {
          _isWeekend = true;
          _timeSelected = false;
          _currentIndex = null;
          _selectedAnimal = null;
          _isLoadingSlots = false;} else {
          _isWeekend = false;
          BlocProvider.of<AppointmentCubit>(context)
              .fetchAvailableSlotsForDay(
            widget.doctor.docId,
            selectedDay,);}});},);}
Widget _timeSelectionGrid(ColorScheme theme) {
  return SliverGrid(
    delegate: SliverChildBuilderDelegate(
      (context, index) {
        final currentHour = index + 9;
```

```
        final isBooked = _bookedSlots.any((slot) {
          return slot.hour == currentHour && slot.day == _currentDay.day;});
        return InkWell(
          splashColor: theme.surface,
          onTap: isBooked
              ? null: () {
                  setState(() {
                    _currentIndex = index;
                    _timeSelected = true;});},
          child: Container(
            margin: const EdgeInsets.all(5),
            decoration: BoxDecoration(
              border: Border.all(
                color: _currentIndex == index
                    ? theme.primary
                    : isBooked
                        ? theme.inversePrimary
                        : theme.secondary,),
              borderRadius: BorderRadius.circular(15),
              color: _currentIndex == index
                  ? theme.primary
                  : isBooked
                      ? theme.surface
                      : null,),
            alignment: Alignment.center,
            child: Text(
              '$currentHour:00 ${currentHour >= 12 ? "PM" : "AM"}',
              style: TextStyle(
                fontWeight: FontWeight.bold,
                color: isBooked
                    ? Colors.grey
                    : _currentIndex == index
                        ? Colors.white
                        : theme.primary,),),),);},
      childCount: 8,),
    gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
      crossAxisCount: 4,
      childAspectRatio: 1.5,),);}
Widget _weekendMessage() {
  return SliverToBoxAdapter(
    child: Container(
      width: double.infinity,
      padding: const EdgeInsets.symmetric(horizontal: 10, vertical: 30),
      alignment: Alignment.center,
```

```
      child: Text(
        S.of(context).weekendIsNotAvailablePleaseSelectAnotherDate,
        style: const TextStyle(
          fontSize: 18,
          fontWeight: FontWeight.bold,),
        textAlign: TextAlign.center,),),);
Widget _animalSelection() {
  return SliverToBoxAdapter(
    child: Padding(
      padding: const EdgeInsets.all(10),
      child: Card(
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(12),),
        elevation: 2,
        child: InkWell(
          borderRadius: BorderRadius.circular(12),
          onTap: () {_showAnimalSelectionModal(context);},
          child: Padding(
            padding: const EdgeInsets.symmetric(vertical: 15, horizontal: 20),
            child: Row(children: [ClipRRect(
                  borderRadius: BorderRadius.circular(12),
                  child: CachedNetworkImage(
                    imageUrl: _selectedAnimal == null
                        ? ApplicationImages.petDefaultImage
                        : _petImage!,width: 48,height: 48,
                    fit: BoxFit.cover,),),
                const SizedBox(width: 16),
                Expanded(child: Text(
                    _selectedAnimal == null
                        ? S.of(context).selectAnimal
                        : _selectedAnimal!,
                    style: const TextStyle(
                      fontSize: 18,fontWeight: FontWeight.w600,),
                    overflow: TextOverflow.ellipsis,),),
                const Icon(Icons.arrow_drop_down,
                    size: 28, color: Colors.grey),],),),),),),),);}
Widget _makeAppointmentButton() {
  return SliverToBoxAdapter(
    child: Padding(
      padding: const EdgeInsets.symmetric(horizontal: 20, vertical: 20),
      child: CustomCircleElavatedButton(
        width: double.infinity,
        onPressed: () {
          createAppointment();
```

```
          Navigator.pop(context);},
        disable: !(_selectedAnimal != null && _timeSelected && _dateSelected),
        title: S.of(context).makeAppointment,),),);}
void _showAnimalSelectionModal(BuildContext context) {
  final theme = Theme.of(context).colorScheme;
  showModalBottomSheet(
    backgroundColor: theme.surface,
    context: context,
    builder: (context) {
      return BlocBuilder<PetCubit, PetState>(
        builder: (context, state) {
          if (state is PetSuccess) {
            return Padding(
              padding:const EdgeInsets.symmetric(horizontal: 16, vertical: 8),
              child: ListView.separated(
                itemCount: state.pets.length,
                separatorBuilder: (context, index) =>
                    const SizedBox(height: 12),
                itemBuilder: (context, index) {
                  final pet = state.pets[index];
                  return Card(shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(16),),
                    elevation: 4,
                    child: InkWell(
                      borderRadius: BorderRadius.circular(16),
                      onTap: () {setState(() {_selectedAnimal = pet.name;
                          _selectedAnimalId = pet.petId;
                          _petImage = pet.petImageUrl;});
                        Navigator.pop(context);},
                      child: Padding(
                        padding: const EdgeInsets.all(12),
                        child: Row(
                          children: [
                            ClipRRect(
                              borderRadius: BorderRadius.circular(12),
                              child: CachedNetworkImage(
                                imageUrl: pet.petImageUrl!,
                                width: 56,
                                height: 56,
                                fit: BoxFit.cover,),),
                            const SizedBox(width: 16),
                            Expanded(child: Text(pet.name,
                                style: TextStyle(
                                  color: theme.primary,
```

```
                                    fontSize: 20,
                                    fontWeight: FontWeight.bold,),),),
                                const Text("Chose",
                                  style: TextStyle(
                                    fontSize: 20,
                                    fontWeight: FontWeight.w400,
                                   color: Colors.grey),),],),),),);},),);} else
{return const Center(child: CircularProgressIndicator());}},);},);}}
```

home_screen.dart

```
import 'package:flutter/material.dart';
import 'package:font_awesome_flutter/font_awesome_flutter.dart';
import 'package:vet_clinic/core/theme/theme.dart';
import
'package:vet_clinic/feature/doctors/presentation/screens/doctors_screen.dart';
import 'package:vet_clinic/feature/pets/presentation/screens/pets_screen.dart';
import
'package:vet_clinic/feature/schedule/presentation/screens/schedule_screen.dart';
import
'package:vet_clinic/feature/settings/presentation/screens/settings_screen.dart';
import 'package:vet_clinic/generated/l10n.dart';
class HomeScreen extends StatefulWidget {
  const HomeScreen({super.key});
  @override
  State<HomeScreen> createState() => _HomeScreenState();}
class _HomeScreenState extends State<HomeScreen> {
  var _selectedPageIndex = 0;
  final _pageController = PageController();
  @override
  Widget build(BuildContext context) {
    final theme = context.theme;
    List<BottomNavigationBarItem> bottomNavigationBarItems = [
      BottomNavigationBarItem(
          icon: const Icon(Icons.pets), label: S.of(context).pets),
      BottomNavigationBarItem(
          icon: const Icon(FontAwesomeIcons.userDoctor),
          label: S.of(context).doctors),
      BottomNavigationBarItem(
          icon: const Icon(Icons.schedule), label: S.of(context).schedule),
      BottomNavigationBarItem(
          icon: const Icon(Icons.settings), label: S.of(context).settings),];
    return Scaffold(
      body: SafeArea(
        child: PageView(
```

```
          controller: _pageController,
          onPageChanged: (value) {
            setState(() => _selectedPageIndex = value);},
          children: const [
            PetsScreen(),
            DoctorsScreen(),
            ScheduleScreen(),
            SettingsScreen()],),),
      bottomNavigationBar: SizedBox(
        height: 60,
        child: BottomNavigationBar(
            backgroundColor: theme.surface,
            type: BottomNavigationBarType.fixed,
            selectedLabelStyle: const TextStyle(
              fontWeight: FontWeight.bold,
              fontSize: 15,),
            currentIndex: _selectedPageIndex,
            onTap: _openPage,
            items: bottomNavigationBarItems),),);}
  void _openPage(int index) {
    setState(() => _selectedPageIndex = index);
    _pageController.jumpToPage(index);}}
```

pets_screen.dart

```
import 'package:cached_network_image/cached_network_image.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter_svg/flutter_svg.dart';
import 'package:image_picker/image_picker.dart';
import 'package:vet_clinic/common/widgets/custom_square_elevated_button.dart';
import 'package:vet_clinic/common/widgets/search_field.dart';
import 'package:vet_clinic/core/config/applocation_images.dart';
import 'package:vet_clinic/common/widgets/custom_field.dart';
import 'package:vet_clinic/common/widgets/loader.dart';
import 'package:vet_clinic/core/theme/theme.dart';
import 'package:vet_clinic/feature/auth/presentation/cubits/auth_cubit.dart';
import 'package:vet_clinic/feature/pets/presentation/cubits/pet/pet_cubit.dart';
import 'package:vet_clinic/feature/pets/presentation/cubits/pet/pet_state.dart';
import 'package:vet_clinic/generated/l10n.dart';
enum ViewMode { list, grid }
class PetsScreen extends StatefulWidget {
  const PetsScreen({super.key});
  @override
  State<PetsScreen> createState() => _PetsScreenState();}
```

```
class _PetsScreenState extends State<PetsScreen> {
  String? uid;
  File? imagePetPickedFile;
  ViewMode viewMode = ViewMode.list;
  final nameController = TextEditingController();
  final breedController = TextEditingController();
  final genderController = TextEditingController();
  final ageController = TextEditingController();
  @override
  void initState() {
    super.initState();
    uid = BlocProvider.of<AuthCubit>(context).getCurrentUid();
    final petCubit = BlocProvider.of<PetCubit>(context);
    if (petCubit.state is! PetSuccess ||
        (petCubit.state as PetSuccess).pets.isEmpty) {
      petCubit.fetchPets(uid!);}}
  void clearFields() {
    nameController.clear();
    breedController.clear();
    genderController.clear();
    ageController.clear();
    setState(() {
      imagePetPickedFile = null;});}
  Future<void> pickImage() async {
    final picker = ImagePicker();
    final XFile? pickedFile =
        await picker.pickImage(source: ImageSource.gallery);
    if (pickedFile != null) {
      setState(() {
        imagePetPickedFile = File(pickedFile.path);});}}
  void createPet() {
    final name = nameController.text;
    final breed = breedController.text;
    final gender = genderController.text;
    final ageString = ageController.text;
    if (name.isEmpty) {
      ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(S.of(context).pleaseEnterAValidNameForThePet)),);return;}
    final age = int.tryParse(ageString);
    final imagePath = imagePetPickedFile?.path;
    BlocProvider.of<PetCubit>(context).createPet(
      uid!,name, breed,gender,age,imagePath,"",);Navigator.pop(context);
clearFields();}
```

```
  @override
  Widget build(BuildContext context) {
    final theme = context.theme;
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.transparent,
        actions: [
          AnimatedSwitcher(
            duration: const Duration(milliseconds: 300),
            transitionBuilder: (child, animation) {
              return RotationTransition(
                turns: animation,
                child: child,);},
            child: IconButton(
              key: ValueKey<ViewMode>(viewMode),
              icon: Icon(
                viewMode == ViewMode.list ? Icons.grid_view : Icons.list,),
              onPressed: () {setState(() {
                  viewMode = viewMode == ViewMode.list ? ViewMode.grid :
ViewMode.list;});},),),],),
      body: BlocBuilder<PetCubit, PetState>(
        builder: (context, state) {
          if (state is PetSuccess) {
            return Column(children: [const SearchField(),
                Expanded(child: viewMode == ViewMode.list
                      ? _buildListView(state)
                      : _buildGridView(state),),],);
          } else if (state is PetLoading) {
            return const Loader();
          } else if (state is PetEmpty) {
            return Center(child: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [Text(
                    state.text,style: const TextStyle(
                      fontSize: 18,fontWeight: FontWeight.bold,
                    ),textAlign: TextAlign.center,),
                  SvgPicture.asset(ApplicationImages.sadSmileImage,
                    colorFilter: ColorFilter.mode(theme.primary,
                      BlendMode.srcIn,),
                    width: 75,height: 75,),],),);} else {
            return const SizedBox.shrink();}},),
      floatingActionButton: FloatingActionButton(
        onPressed: () {_showAddPetBottomSheet(context);},
        backgroundColor: theme.primary,
```

```
      child: const Icon(Icons.add,color: Colors.white,size: 26,),),
    floatingActionButtonLocation: FloatingActionButtonLocation.endFloat,);}
ListView _buildListView(PetSuccess state) {
  return ListView.builder(
    itemCount: state.pets.length,
    itemBuilder: (context, index) {
      final pet = state.pets[index];
      return ListViewPetCard(
        onTap: () {Navigator.push(context,
            MaterialPageRoute(builder: (context) => PetDetailsScreen(
                pet: pet,uid: uid!,),),);},
        name: pet.name,image: pet.petImageUrl,breed: pet.breed,
        gender: pet.gender,age: pet.age,);},);}
GridView _buildGridView(PetSuccess state) {
  return GridView.builder(
    gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
      crossAxisCount: 2,
      crossAxisSpacing: 16,
      mainAxisSpacing: 16,
      childAspectRatio: 0.85,),
    itemCount: state.pets.length,
    itemBuilder: (context, index) {
      final pet = state.pets[index];
      return GridViewPetCard(onTap: () {Navigator.push(context,
            MaterialPageRoute(builder: (context) => PetDetailsScreen(
                pet: pet,uid: uid!,),),);},
        name: pet.name,image: pet.petImageUrl,breed: pet.breed,
        gender: pet.gender,age: pet.age,);},);}
Future<void> _showAddPetBottomSheet(BuildContext context) {
  String? selectedGender;
  return showModalBottomSheet(
    context: context,
    isScrollControlled: true,
    builder: (context) {return StatefulBuilder(
        builder: (BuildContext context, StateSetter modalSetState) {
          return SizedBox(height: 650,
            child: Padding(padding: const EdgeInsets.all(10.0),
              child: Column(
                mainAxisSize: MainAxisSize.min,
                children: [Text(S.of(context).addANewPet,
                    style: const TextStyle(
                        fontWeight: FontWeight.bold, fontSize: 16),
                  ),const SizedBox(height: 16),
                  CustomField(
```

```
    labelText: S.of(context).name,
    controller: nameController),
CustomField(
    labelText: S.of(context).breed,
    controller: breedController),
CustomDropdown<String>(
  selectedValue: selectedGender,
  controller: genderController,
  labelText: S.of(context).gender,
  items: [DropdownMenuItem(
        value: "", child: Text(S.of(context).notSpecified)),
    DropdownMenuItem(
      value: "Male",child: Row(children: [
          Text(S.of(context).male),
          const Icon(Icons.male, color: Colors.blue)],),),
    DropdownMenuItem(value: "Female",child: Row(
        children: [Text(S.of(context).female),
     const Icon(Icons.female, color: Colors.pink)],),),],
  onChanged: (value) {
    modalSetState(() {
      selectedGender = value;
      genderController.text = value ?? '';});},),
CustomField(
    labelText: S.of(context).age,
    controller: ageController,
    isNumeric: true),
const SizedBox(height: 16),
GestureDetector(
  onTap: () async {
    await pickImage();
    modalSetState(() {});},
  child: Container(
    height: 150,
    width: 150,
    decoration: const BoxDecoration(shape: BoxShape.circle),
    clipBehavior: Clip.hardEdge,
    child: (imagePetPickedFile != null)
        ? Image.file(File(imagePetPickedFile!.path),
            fit: BoxFit.cover)
        : CachedNetworkImage(
            imageUrl: ApplicationImages.petDefaultImage,
            placeholder: (context, url) =>
                const CircularProgressIndicator(),
            errorWidget: (context, url, error) =>
```

```
                                    Image.network(
                                        ApplicationImages.petDefaultImage,
                                        fit: BoxFit.cover),),),),
                    const SizedBox(height: 16),
                    CustomSquareElevatedButton(
                      onTap: createPet,
                      text: S.of(context).add,),],),),);},);},);}}
```

create_reminder_screen.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:table_calendar/table_calendar.dart';
import 'package:vet_clinic/common/widgets/custom_circle_elavated_button.dart';
import 'package:vet_clinic/common/widgets/custom_field.dart';
import 'package:vet_clinic/common/widgets/custom_square_elevated_button.dart';
import 'package:vet_clinic/feature/auth/presentation/cubits/auth_cubit.dart';
import 'package:vet_clinic/feature/pets/domain/entities/pet.dart';
import 'package:vet_clinic/feature/reminder/cubit/reminder_cubit.dart';
import 'package:vet_clinic/feature/reminder/model/reminder.dart';
import 'package:vet_clinic/generated/l10n.dart';
class AddNotificationsScreen extends StatefulWidget {
  const AddNotificationsScreen({super.key, required this.pet});
  final Pet pet;
  @override
 State<AddNotificationsScreen> createState() => _AddNotificationsScreenState();}
class _AddNotificationsScreenState extends State<AddNotificationsScreen> {
  CalendarFormat _format = CalendarFormat.month;
  final DateTime _focusedDay = DateTime.now();
  DateTime _currentDay = DateTime.now();
  int _selectedHour = TimeOfDay.now().hour;
  int _selectedMinute = TimeOfDay.now().minute;
  final titleController = TextEditingController();
  @overrideWidget build(BuildContext context) {
    return Scaffold(
      resizeToAvoidBottomInset: true,
      appBar: AppBar(leading: IconButton(
          onPressed: () => Navigator.pop(context),
          icon: const Icon(Icons.arrow_back),),
        title: Text(S.of(context).createAReminder),),
      body: SingleChildScrollView(
        padding: const EdgeInsets.symmetric(horizontal: 8),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: [CustomField(
```

```
            labelText: S.of(context).enterYourReminderText,
            controller: titleController,),
          const SizedBox(height: 6),
          _buildTableCalendar(),
          const SizedBox(height: 9),
          _buildTimePicker(),
          const SizedBox(height: 9),
          CustomCircleElavatedButton(
            title: S.of(context).schedule,
            onPressed: () => _scheduleReminder(context),
            disable: false,),],),),);}
TableCalendar<dynamic> _buildTableCalendar() {
  final theme = Theme.of(context).colorScheme;
  return TableCalendar(
    focusedDay: _focusedDay,
    firstDay: DateTime.now().subtract(const Duration(days: 30)),
    lastDay: DateTime.now().add(const Duration(days: 365)),
    calendarFormat: _format,
    currentDay: _currentDay,
    rowHeight: 60,
    calendarStyle: CalendarStyle(
      selectedDecoration: BoxDecoration(
        color: theme.primary,
        shape: BoxShape.circle,),),
    availableCalendarFormats: const {CalendarFormat.month: 'Month'},
    enabledDayPredicate: (day) {
   return !day.isBefore(DateTime.now().subtract(const Duration(days: 1)));},
    onFormatChanged: (format) {setState(() {_format = format;});},
    onDaySelected: (selectedDay, focusedDay) {
      if (!selectedDay.isBefore(DateTime.now())) {setState(() {
          _currentDay = selectedDay;});}},);}
Widget _buildTimePicker() {
  return Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      DropdownButton<int>(
        value: _selectedHour,
        items: List.generate(24, (index) {
          return DropdownMenuItem(
            value: index,
            child: Text(index.toString().padLeft(2, '0'),
                style: const TextStyle(
     fontSize: 18, fontWeight: FontWeight.bold)),);}),
        onChanged: (value) {setState(() {
```

```
              _selectedHour = value!;});},),
        const Text("  :  ",
            style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold)),
        DropdownButton<int>(
          value: _selectedMinute,
          items: List.generate(60, (index) {
            return DropdownMenuItem(
              value: index,
              child: Text(index.toString().padLeft(2, '0'),
                  style: const TextStyle(
                      fontSize: 18, fontWeight: FontWeight.bold)),);}),
          onChanged: (value) {
            setState(() {
              _selectedMinute = value!;});},),],);}
  void _scheduleReminder(BuildContext context) {
    final String description = titleController.text;
    if (description.isEmpty) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
            content: Text(S.of(context).pleaseFillOutTheFields,
                style: const TextStyle(fontSize: 16))),);return;}
    final DateTime reminderTime = DateTime(
  _currentDay.year,_currentDay.month,_currentDay.day,
_selectedHour,_selectedMinute,);
    final uId = BlocProvider.of<AuthCubit>(context).currentUser?.uid;
    final int reminderId = DateTime.now().millisecondsSinceEpoch % (1 << 31);
    final reminder = Reminder(
      reminderId: reminderId,
      title: description,dateTime: reminderTime,uId: uId!,petId:
widget.pet.petId,petName: widget.pet.name,petImage: widget.pet.petImageUrl!,);
    BlocProvider.of<ReminderCubit>(context).createReminder(reminder);
    BlocProvider.of<NotificationCubit>(context).scheduleNotification(reminder);
    Navigator.pop(context);}}
```

pet_details_screen.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:open_file/open_file.dart';
import 'package:vet_clinic/common/utils/formated_date.dart';
import 'package:vet_clinic/common/utils/formated_time.dart';
import 'package:vet_clinic/core/config/applocation_images.dart';
import 'package:vet_clinic/core/theme/theme.dart';
import 'package:vet_clinic/feature/pets/domain/entities/pet.dart';
import 'package:vet_clinic/feature/pets/presentation/cubits/pet/pet_cubit.dart';
```

```
import 'package:cached_network_image/cached_network_image.dart';
import 'package:vet_clinic/generated/l10n.dart';
class PetDetailsScreen extends StatefulWidget {
  const PetDetailsScreen({super.key, required this.pet, required this.uid});
  final Pet pet;
  final String uid;
  @override
  State<PetDetailsScreen> createState() => _PetDetailScreenState();}
class _PetDetailScreenState extends State<PetDetailsScreen> {
  late Pet _currentPet;
  @override
  void initState() {
    super.initState();
    _currentPet = widget.pet;
    BlocProvider.of<AppointmentCubit>(context)
        .fetchAppointmentsWithDetailsByPet(widget.uid, widget.pet.petId);}
  void deletePet() {
    BlocProvider.of<PetCubit>(context).deletePet(widget.uid, widget.pet.petId);}
  void _showPdfOptionsDialog(
      BuildContext context, String url, String fileName) {
    showDialog(
      context: context,
      builder: (_) => AlertDialog(
        title: const Text('Оберіть дію'),
        content: const Text('Що ви хочете зробити з файлом PDF?'),
        actions: [
          TextButton(
            onPressed: () async {
              Navigator.of(context).pop();
              final file = await context.read<PetPdfCubit>()
                  .storageRepository.downloadPdfWithFirebase(url,fileName,);
              if (file != null) {
                ScaffoldMessenger.of(context).showSnackBar(
             const SnackBar(content: Text("Файл збережено на пристрій")),);}},
            child: const Text('Завантажити на пристрій'),),
          TextButton(
            onPressed: () {
              Navigator.of(context).pop();
              context.read<PetPdfCubit>().downloadPdf(url, fileName);},
            child: const Text('Переглянути'),),],),);}
  @override
  Widget build(BuildContext context) {
    return MultiBlocListener(
      listeners: [
```

```
        BlocListener<PetCubit, PetState>(
          listener: (context, state) {
            if (state is PetLoaded) {
              setState(() {
                _currentPet = state.pet;});}},),
        BlocListener<PetPdfCubit, PetPdfState>(
          listener: (context, state) async {
            if (state is PetPdfDownloaded) {
              await OpenFile.open(state.file.path);
            } else if (state is PetPdfError) {
              ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(content: Text(state.message)),);}},),],
      child: Scaffold(
        extendBodyBehindAppBar: true,
        appBar: _appBar(context),
        body: Column(children: [_PetAvatar(currentPet: _currentPet),
            Padding(padding: const EdgeInsets.symmetric(horizontal: 25,
vertical: 10),child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [const SizedBox(height: 5),
             _PetName(currentPet: _currentPet),const SizedBox(height: 20),
              _rowPetInfoBlock(context),const SizedBox(height: 20),
               _buildPdfSection(context),const SizedBox(height: 30),],),),
            _buildAppointmentsSection(),],),),);}
  Widget _buildPdfSection(BuildContext context) {
    final pdfUrl = _currentPet.petPdfUrl;
    if (pdfUrl != null && pdfUrl.isNotEmpty) {
      return Card(elevation: 4,
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),
        color: Colors.red[50],
        child: ListTile(contentPadding:
              const EdgeInsets.symmetric(horizontal: 16, vertical: 12),
          leading:
              const Icon(Icons.picture_as_pdf, color: Colors.red, size: 36),
          title: Text(
            "Document about the animal ",
            style:
                TextStyle(fontWeight: FontWeight.w600, color: Colors.red[800]),
          ),subtitle: const Text(
            "Tap for more actions",
            style: TextStyle(fontStyle: FontStyle.italic, color: Colors.grey),),
          onTap: () => _showPdfOptionsDialog(context,
            pdfUrl,"${_currentPet.name}_document",),),);
    } else {
```

```
      return Container(
        width: double.infinity,
        padding: const EdgeInsets.all(16),
        decoration: BoxDecoration(
          color: Colors.grey[100],
          borderRadius: BorderRadius.circular(12),
          border: Border.all(color: Colors.grey.shade300),),
        child: const Row(
          children: [Icon(Icons.info_outline, color: Colors.grey, size: 28),
            SizedBox(width: 12),Expanded(
              flex: 3,child: Text(
                "No documents about the animal have been added",
                textAlign: TextAlign.center,
                style: TextStyle(
                  fontSize: 16,fontStyle: FontStyle.italic,color:
Colors.black87,),),),],),);}}
  Widget _buildAppointmentsSection() {
    return BlocBuilder<AppointmentCubit, AppointmentState>(
      builder: (context, state) {
        if (state is AppointmentLoading) {
          return const Center(child: CircularProgressIndicator());
        } else if (state is AppointmentError) {
          return Center(child: Text('Error: ${state.error}'));
        } else if (state is AppointmentEmpty) {
          return Center(
            child: Text(
              S.of(context).yourAnimalHasNoVeterinaryRecordsYet,
         style: const TextStyle(fontSize: 16, fontWeight: FontWeight.w600),),);
        } else if (state is AppointmentLoaded) {
          return _currentPetAppointments(state.appointments);
        } else {return const SizedBox.shrink();}},);}
  Flexible _currentPetAppointments(List<AppointmentWithDetails> appointments) {
    return Flexible(
      child: ListView.builder(
        padding: EdgeInsets.zero,
        itemCount: appointments.length,
        itemBuilder: (context, index) {
          final appointment = appointments[index];
          final formattedDate = formatedDate(appointment.appointment.dateTime);
          final formattedTime = formatedTime(appointment.appointment.dateTime);
          return ScheduleMiniCard(
            doctorMajor: appointment.doctor.major,date: formattedDate,
            time: formattedTime,doctorName: appointment.doctor.name,
            doctorImage: appointment.doctor.docImagePath,);},),);}
```

```
  Row _rowPetInfoBlock(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        PetInfoBlock(title: S.of(context).breed, subtitle: _currentPet.breed),
        PetInfoBlock(
   title: S.of(context).age, subtitle: _currentPet.age.toString()),
   PetInfoBlock(title: S.of(context).gender, subtitle: _currentPet.gender),],);}
  AppBar _appBar(BuildContext context) {
    return AppBar(
      elevation: 0,leading: Padding(
        padding: const EdgeInsets.all(8.0),
        child: _buildIconButton(icon: Icons.arrow_back,
          onPressed: () => Navigator.pop(context),),),
      actions: [_buildIconButton(icon: Icons.share, onPressed: () {}),
        const SizedBox(width: 5),
      _buildIconButton(
      icon: Icons.more_vert,onPressed: () => _showPetOptionsMenu(context)),],);}
  Widget _buildIconButton({
    required IconData icon,
    required VoidCallback onPressed,}) {
    return Container(
    decoration:const BoxDecoration(color: Colors.white, shape: BoxShape.circle),
      child: IconButton(icon: Icon(icon), onPressed: onPressed),);}
  void _showPetOptionsMenu(BuildContext context) {
    showMenu(color: context.theme.surface,context: context,
      position: const RelativeRect.fromLTRB(100.0, 75.0, 10.0, 10.0),
      items: [_buildMenuItem(icon: Icons.edit,
  text: S.of(context).edit,onTap: () async {
    final result = await Navigator.push(context,
      MaterialPageRoute(
      builder: (context) => PetEditScreen(uid: widget.uid,
          pet: _currentPet,),),);
    if (result == true) {
      final petCubit = context.read<PetCubit>();
      final updatedPet = (petCubit.state as PetSuccess)
          .pets.firstWhere((p) => p.petId == _currentPet.petId, orElse: () =>
_currentPet);
setState(() {_currentPet = updatedPet;});}},),
        _buildMenuItem(
          icon: Icons.add,
          text: S.of(context).addAReminder,
          onTap: () {
            Navigator.push(context,
```

```
              MaterialPageRoute(  builder: (context)
=>AddNotificationsScreen(pet: _currentPet)),);},),
        _buildMenuItem(icon: Icons.delete,
          text: S.of(context).delete,onTap: () {
    Navigator.pop(context);_showDeleteConfirmationDialog(context);},),],);}
  PopupMenuItem _buildMenuItem({
    required IconData icon,
    required String text,
    required VoidCallback onTap,}) {
    return PopupMenuItem(
      child: ListTile(
        leading: Icon(icon, color: Colors.black),
        title: Text(text, style: const TextStyle(fontSize: 16)),
        onTap: onTap,),);}
  void _showDeleteConfirmationDialog(BuildContext context) {
    showDialog(
      context: context,
      builder: (_) => AlertDialog(
        title: Text(S.of(context).areYouSureYouWantToDeleteThePet),
        actions: [TextButton(
            onPressed: () => Navigator.pop(context),
            child: Text(S.of(context).cancel,style: const TextStyle(color:
Colors.grey)),),
          TextButton(onPressed: () {deletePet();
              Navigator.popUntil(context, (route) => route.isFirst);},
            child: Text(S.of(context).yes,
                style: const TextStyle(color: Colors.red)),),],),);}}
class _PetName extends StatelessWidget {
  const _PetName({required this.currentPet});
  final Pet currentPet;
  @override
  Widget build(BuildContext context) {
    return Text(
      currentPet.name,
      style: TextStyle(
          fontSize: 26,
          fontWeight: FontWeight.bold,
          color: context.theme.primary),);}}
class _PetAvatar extends StatelessWidget {
  const _PetAvatar({required this.currentPet});
  final Pet currentPet;
  @override
  Widget build(BuildContext context) {
    return SizedBox(
```

```
      height: 300,
      width: double.infinity,
      child: CachedNetworkImage(
        imageUrl: currentPet.petImageUrl ?? ApplicationImages.petDefaultImage,
        fit: BoxFit.cover,),);}}
```

profile_screen.dart

```
import 'package:cached_network_image/cached_network_image.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:vet_clinic/common/widgets/custom_circle_elavated_button.dart';
import 'package:vet_clinic/core/theme/theme.dart';
import 'package:vet_clinic/feature/auth/domain/entities/profile_user.dart';
import
class ProfileScreen extends StatefulWidget {
  const ProfileScreen({super.key, required this.uid});
  final String uid;
  @override
  State<ProfileScreen> createState() => _ProfileScreenState();}
class _ProfileScreenState extends State<ProfileScreen> {
  @override
  void initState() {
    super.initState();
    BlocProvider.of<ProfileCubit>(context).fetchUserProfile(widget.uid);}
  @override
  Widget build(BuildContext context) {
    final theme = context.theme;
    return Scaffold(
      appBar: _buildAppBar(context, theme),
      extendBodyBehindAppBar: true,
      body: SafeArea(
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal: 15),
          child: BlocBuilder<ProfileCubit, ProfileState>(
            builder: (context, state) {
              if (state is ProfileLoaded) {final user = state.profileUser;
                return SingleChildScrollView(child: Column(
                    crossAxisAlignment: CrossAxisAlignment.start,children: [
                      _PrtofileScreenAvatar(user: user),
                      const SizedBox(height: 5),
                      _UserEmailText(user: user),
                      const SizedBox(height: 10),Padding(
                        padding: const EdgeInsets.symmetric(horizontal: 10),
                        child: _ProfileNameText(theme: theme), ),
```

```
                        ProfileField(readOnly: true, initialText: user.name),
                        Padding(
                          padding: const EdgeInsets.symmetric(horizontal: 10),
                          child: _ProfileBioText(theme: theme),),
                        ProfileField(readOnly: true, initialText: user.bio),
                        Padding(
                          padding: const EdgeInsets.symmetric(horizontal: 10),
                          child: _ProfilePasswrodText(theme: theme),),
                        ProfileField(
                            readOnly: true,
                            obscureText: true,
                            initialText: user.password),
                        Padding(
                          padding: const EdgeInsets.symmetric(horizontal: 10),
                          child: _ProfilePhoneText(theme: theme),),
                        ProfileField(readOnly: true, initialText: user.phone),
                        const SizedBox(height: 10),
                        _ProfileEditInfoButton(user: user),],),);
       } else if (state is ProfileLoading) {
       return const Center(child: CircularProgressIndicator());
    } else {return Center(child: Text(S.of(context).noProfileFound));}},),),),);}
  AppBar _buildAppBar(BuildContext context, ColorScheme theme) {
    return AppBar(
      backgroundColor: Colors.transparent,
      elevation: 0,
      actions: [
        IconButton(onPressed: () {
            Navigator.push(context,
 MaterialPageRoute(builder: (context) => const RemindersScreen()),);},
  icon: Icon(Icons.notifications, color: theme.onSurface, size: 28),),],);}}
class _ProfileNameText extends StatelessWidget {
  const _ProfileNameText({
    required this.theme,});
  final ColorScheme theme;
  @override
  Widget build(BuildContext context) {
    return Text(
      S.of(context).name,
      style: TextStyle(
      color: theme.primary, fontWeight: FontWeight.bold, fontSize: 16),);}}
class _ProfileEditInfoButton extends StatelessWidget {
  const _ProfileEditInfoButton({required this.user});
  final ProfileUser user;
```

```
  @override
  Widget build(BuildContext context) {
    return Align(
      alignment: Alignment.center,
      child: CustomCircleElavatedButton(
        title: S.of(context).editInfo,
        onPressed: () => Navigator.push(
          context,
          MaterialPageRoute(
              builder: (context) => EditProfileScreen(user: user)),
        ),disable: false,),);}}
class _ProfilePhoneText extends StatelessWidget {
  const _ProfilePhoneText({required this.theme});
  final ColorScheme theme;
  @override
  Widget build(BuildContext context) {
    return Text(S.of(context).phone,style: TextStyle(
      color: theme.primary, fontWeight: FontWeight.bold, fontSize: 16),);}}
class _ProfilePasswrodText extends StatelessWidget {
  const _ProfilePasswrodText({required this.theme});
  final ColorScheme theme;
  @override
  Widget build(BuildContext context) {
    return Text(
      S.of(context).password,
      style: TextStyle(
          color: theme.primary, fontWeight: FontWeight.bold, fontSize: 16),);}}
class _ProfileBioText extends StatelessWidget {
  const _ProfileBioText({required this.theme});
  final ColorScheme theme;
  @override
  Widget build(BuildContext context) {
    return Text(
      S.of(context).bio,
      style: TextStyle(
          color: theme.primary, fontWeight: FontWeight.bold, fontSize: 16),);}}
class _UserEmailText extends StatelessWidget {
  const _UserEmailText({required this.user});
  final ProfileUser user;
  @override
  Widget build(BuildContext context) {
    return Align(alignment: Alignment.center,
      child: Text(user.email,
   style: const TextStyle(fontSize: 20, fontWeight: FontWeight.bold),),);}}
```

```
class _PrtofileScreenAvatar extends StatelessWidget {
  const _PrtofileScreenAvatar({required this.user});
  final ProfileUser user;
  @override
  Widget build(BuildContext context) {
    return Align(
      alignment: Alignment.center,
      child: CircleAvatar(
      radius: 70,
      backgroundImage: CachedNetworkImageProvider(user.profileImageUrl),),);}}
```

settings_screen.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';


class SettingsScreen extends StatefulWidget {
  const SettingsScreen({super.key});
  @override
  State<SettingsScreen> createState() => _SettingsScreenState();}
class _SettingsScreenState extends State<SettingsScreen> {
  @override
  Widget build(BuildContext context) {
    final theme = context.theme;
    String uid = BlocProvider.of<AuthCubit>(context).getCurrentUid();
    final isDarkTheme = context.watch<ThemeCubit>().state.isDark;
    final isUkrainianLanguage =
        context.watch<LanguageCubit>().state.isUkrainian;
    final currentLanguage = isUkrainianLanguage ? 'Українська' : 'English';
    return Scaffold(
      appBar: AppBar(),
      body: Column(
        children: [          Row(
            mainAxisAlignment: MainAxisAlignment.start,
            children: [  Padding(padding:
               const EdgeInsets.symmetric(horizontal: 15, vertical: 6),
                child: Text(S.of(context).applicationSettings,
              style: const TextStyle(color: Colors.grey, fontSize: 20),),),],),
          SwitchSettingsTile(
            title: S.of(context).darkTheme,
            icon: MdiIcons.weatherNight,
            value: isDarkTheme,
            onChanged: (value) => _setThemeBrightness(context, value),),
          DropdownSettingsTile(
```

```
          title: S.of(context).selectTheLanguage,
          icon: MdiIcons.translate,
          value: currentLanguage,
          items: const ['English','Українська',],
          onChanged: (value) {
            if (value != null) {_setLanguage(context, value);}},),
        TileButton(
          title: "Reminders",
          icon: Icons.notifications,
          color: theme.primary,
          onTap: () {
            Navigator.push(
                context,
                MaterialPageRoute(
                    builder: (context) => const RemindersScreen()));},),
        Row(mainAxisAlignment: MainAxisAlignment.start,
          children: [
            Padding(
              padding: const EdgeInsets.symmetric(horizontal: 15),
              child: Text(
             S.of(context).accountSettings,
              style: const TextStyle(color: Colors.grey, fontSize: 20),),),],),
        TileButton(
          title: S.of(context).profile,
          icon: Icons.arrow_forward,
          color: theme.primary,
          onTap: () {
            Navigator.push(
                context,
                MaterialPageRoute(
                    builder: (context) => ProfileScreen(uid: uid)));},),
        TileButton(
          key: const Key('logoutButton'),
          title: S.of(context).logout,
          icon: Icons.logout,
          color: Colors.red,
          onTap: () {
          BlocProvider.of<AuthCubit>(context).logout();},),],),);}
void _setThemeBrightness(BuildContext context, bool value) {
  context
      .read<ThemeCubit>()
      .setThemeBrightness(value ? Brightness.dark : Brightness.light); }

void _setLanguage(BuildContext context, String language) {
```

```
    final isUkrainian = language == 'Українська';
    context .read<LanguageCubit>()
        .setLanguage(isUkrainian ? const Locale("uk") : const Locale('en'));}}
```

doctor_details_screen.dart

```
import 'package:flutter/material.dart';
import 'package:vet_clinic/common/widgets/custom_square_elevated_button.dart';
import 'package:vet_clinic/feature/doctors/domain/entities/doctor.dart';
class DoctorDetailsScreen extends StatelessWidget {
  final Doctor doctor;
  const DoctorDetailsScreen({super.key, required this.doctor});
  @override
  Widget build(BuildContext context) {
    final textTheme = Theme.of(context).textTheme;
    final theme = Theme.of(context).colorScheme;
    return Scaffold(
      appBar: AppBar(
        title: const Text("Профіль лікаря"),
        centerTitle: true,),
      body: SingleChildScrollView(
        padding: const EdgeInsets.all(16),
        child: Column(
          children: [
            Card(elevation: 4,
              shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(20),),
              child: Padding(
                padding: const EdgeInsets.all(20),
                child: Column(
                  children: [CircleAvatar(
                      radius: 60,backgroundImage:
NetworkImage(doctor.docImagePath),),
                    const SizedBox(height: 16),Text(
                      doctor.name,style: textTheme.headlineSmall?.copyWith(
                        fontWeight: FontWeight.bold,),),
                    const SizedBox(height: 8),
                    Text(doctor.major,
                      style: textTheme.titleMedium?.copyWith(fontStyle:
FontStyle.italic,),  ),const SizedBox(height: 24),
                    Align(alignment: Alignment.centerLeft,
                      child: Text("Про лікаря",
                        style: textTheme.titleLarge?.copyWith(
                          fontWeight: FontWeight.bold,  ),),),),
```

```
                    const SizedBox(height: 8),Text(  doctor.bio,style:
textTheme.bodyMedium?.copyWith(),textAlign: TextAlign.justify,),],),),),
            const SizedBox(height: 24),
            Card(elevation: 4,shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(20),  ),
              child: Padding(
                padding: const EdgeInsets.all(20),
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Text(
                      "Контактна інформація",
                      style: textTheme.titleLarge?.copyWith(
                        fontWeight: FontWeight.bold,
                      ),),
                    const SizedBox(height: 12),
                    Row(
                      children: [
                        Icon(Icons.phone, color: theme.primary),
                        const SizedBox(width: 8),
                        Text(doctor.phone,
                            style: textTheme.bodyMedium?.copyWith()),],),
                    const SizedBox(height: 8),
                    Row(
                      children: [
                        Icon(Icons.email, color: theme.primary),
                        const SizedBox(width: 8),
                        Text(doctor.email,
                            style: textTheme.bodyMedium?.copyWith()),],),
                    const SizedBox(height: 8),],),),),
            const SizedBox(height: 20),
            CustomSquareElevatedButton(
                text: "Зробити замовлення",
                onTap: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(
      builder: (context) => AppointmentScreen(doctor: doctor),),);})],),),);}}
```

custom_loader.dart

```
import 'package:flutter/material.dart';
import 'package:vet_clinic/generated/l10n.dart';

class Loader extends StatelessWidget {
```

```
  const Loader({super.key});
  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context).colorScheme;
    return Scaffold(
      backgroundColor: theme.surface,
      body: Stack(
        children: [
          Positioned.fill(
            child: Container(),),
          Center(
            child: Column(
              mainAxisSize: MainAxisSize.min,
              children: [
                const SizedBox(height: 16),
                CircularProgressIndicator(
                  color: theme.primary,
                  strokeWidth: 5,),
                const SizedBox(height: 16),
                AnimatedOpacity(
                  duration: const Duration(milliseconds: 500),
                  opacity: 1,
                  child: Text(
                    S.of(context).loading,
                    style: TextStyle(
                      fontSize: 16,
                      fontWeight: FontWeight.w600,
                      color: theme.onSurface,),),),],),),);}}
```

custom_field.dart

```
import 'package:flutter/material.dart';
class CustomField extends StatelessWidget {
  const CustomField({super.key, this.readOnly = false,
this.obscureText = false,required this.labelText,
this.controller,this.initialText = '',
    this.isNumeric = false,});
  final TextEditingController? controller;
  final String labelText; final bool obscureText; final bool readOnly;final bool
isNumeric; final String initialText;
  @override
  Widget build(BuildContext context) {
    final TextEditingController effectiveController =
        controller ?? TextEditingController(text: initialText);
    return Padding(
```

```
      padding: const EdgeInsets.all(15.0),
      child: TextFormField(
        readOnly: readOnly,
        controller: effectiveController,
        decoration: InputDecoration(
          labelText: labelText,
          border: OutlineInputBorder(borderRadius: BorderRadius.circular(12)),
          hintStyle: const TextStyle(color: Colors.grey),),
        obscureText: obscureText,
     keyboardType: isNumeric ? TextInputType.number : TextInputType.text,),);}}
```

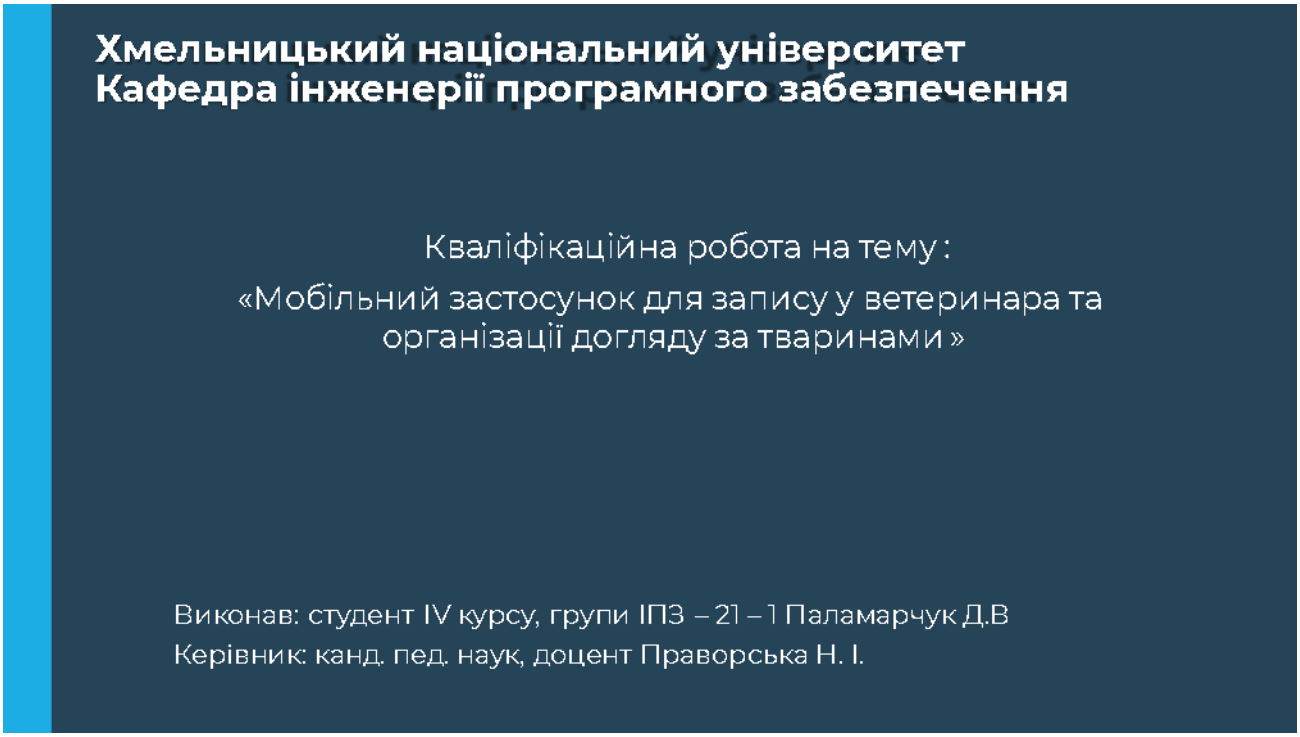
tile_button.dart

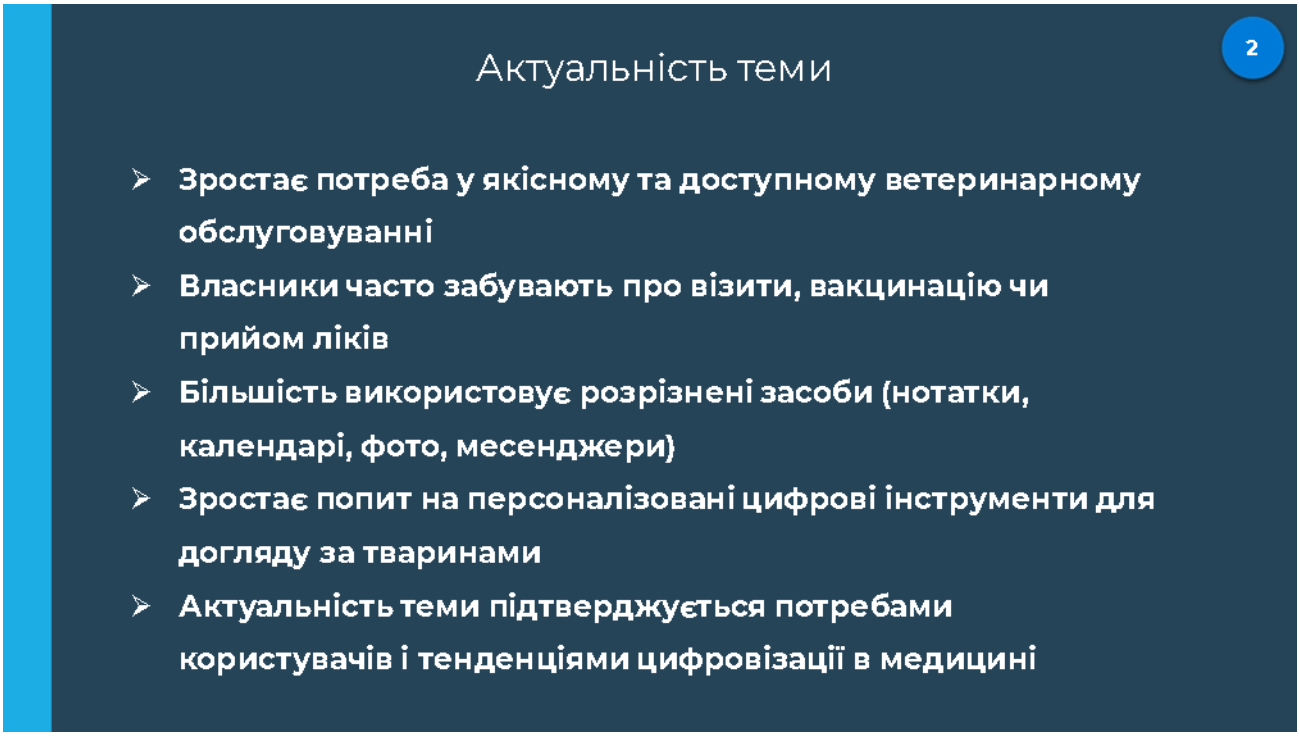
```
import 'package:flutter/material.dart';
class TileButton extends StatelessWidget {
  const TileButton({
    super.key, required this.title, required this.icon,required
this.onTap,this.color,});
  final String title; final IconData icon;
  final Color? color;
  final void Function() onTap;
  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context).colorScheme;
    return GestureDetector(
      onTap: onTap,
      child: Padding(
        padding: const EdgeInsets.symmetric(horizontal: 12, vertical: 16),
        child: Container(
          decoration: BoxDecoration(
            borderRadius: BorderRadius.circular(12),
            color: theme.secondary.withOpacity(0.1),),
          padding: const EdgeInsets.all(20),
          width: double.infinity,
          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              Text(
                title,
                style:
               const TextStyle(fontSize: 18, fontWeight: FontWeight.bold),),
              Icon(
                icon,
                color: color,
                size: 24,),],),),),);}
```

ДОДАТОК Г

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ СЛАЙДИ





Мета та завдання

Метою є створення мобільного застосунку, що дозволяє власникам тварин записуватись до ветеринара, зберігати медичну інформацію, отримувати нагадування про візити та спрощує взаємодію з ветеринарними клініками.

Етап	Завдання
Аналіз	Вивчення існуючих застосунків і потреб користувачів
Вимоги	Формування функціональних і нефункціональних вимог
Архітектура	Вибір архітектури та технологій для реалізації застосунку
Проєктування	Створення структури бази даних і логіки взаємодії між модулями
Розробка	Реалізація клієнтського інтерфейсу та серверної логіки
Функціонал	Реєстрація, записи на прийом, збереженнядокументів
Тестування	Перевірка відповідності вимогам і стабільності роботи
Оцінка	Аналіз результатів і визначення напрямків подальшого розвитку

Змістовний аналіз предметної області, її структурних та функціональних особливостей

У межах дослідження предметної області було розглянуто типи мобільних застосунків та обґрунтовано вибір кросплатформеного підходу на базі сучасних технологій, що дозволяє охопити широку аудиторію при оптимальних витратах.

Цільовою аудиторією застосунку є власники домашніх тварин, які потребують зручного інструменту для запису до ветеринара та догляду за улюбленцями. Потенційними користувачами також можуть бути ветеринарні клініки та притулки.

Ринкова частка мобільних операційних систем у світі

Операційна система	Android	iOS	Samsung	Unknown	KaiOS	Windows
Ринкова частка (%)	72.24%	27.28%	0.31%	0.10%	0.03%	0.01%

Ринкова частка мобільних операційних систем в Україні

Операційна система	Android	iOS	Samsung	Linux	Windows	Symbian
Ринкова частка (%)	66.48%	33.15%	0.26%	0.05%	0.04%	0.01%

Аналіз наявного програмно-технічного забезпечення предметної області

5

Порівняння функцій вже існуючих застосунків

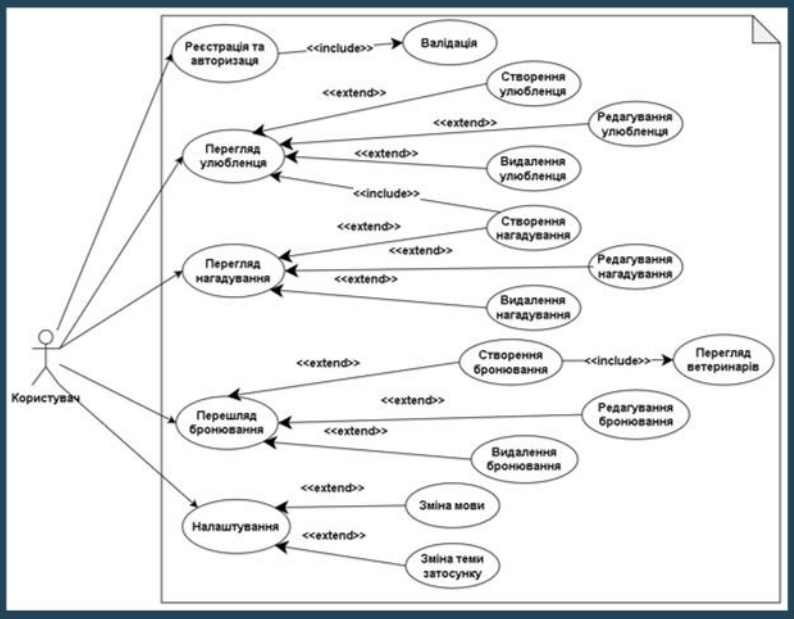
Функціонал	11Pets	AnimalID	DogCatApp
Реєстрація	+	+	+
Введення медичних записів(документів)	+	+	+
Нагадування	-	+	+
Фотогалерея тварин	+	-	+
Можливість бронювання послуг	+	-	-
Інтеграція з клініками	+/-	+	-
Пошук загублених тварин	-	+	-
Зміна теми або елементів інтерфейсу	-	-	+
Зміна мови, формату часу	+	+	+
Підтримка української мови	-	+	+
Наявність преміум-аккаунт для усіх функцій	+	+	+

Визначення функціональних та нефункціональних вимог до програмного забезпечення

6

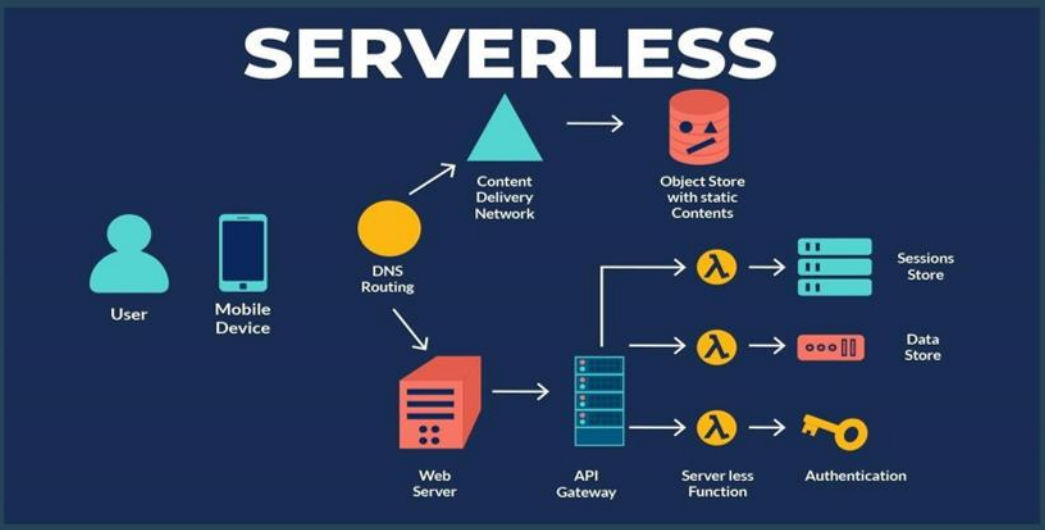
Функціональні вимоги	Нефункціональні вимоги
Створення облікового запису через email, Google або соцмережі	Підтримка Android 8+ та iOS 14+
Створення, редагування й видалення профілів домашніх улюбленців	Хешування паролів (SHA-256)
Прикріплення фото та файлів до профілю тварини	Використання Firebase Security Rulesдля захисту даних
Створення та керування нагадуваннями	Підтримка понад 10 000 одночасних користувачів
Запис на прийом до ветлікаря	Відповідність Material Design (Android)
Перемикання між темною та світлою темою	Відповідність Human Interface Guidelines (iOS)
Підтримка кількох мов (українська, англійська)	Підтримка різних розмірів екранів і орієнтацій (портретна/ландшафтна)
Кастомізація інтерфейсу (кольори, шрифти, елементи)	Оптимізація ресурсів для зменшення споживання батареї та оперативної пам'яті.
Пошук та фільтрація тварин або записів	Резервне копіювання даних

Аналіз предметної області



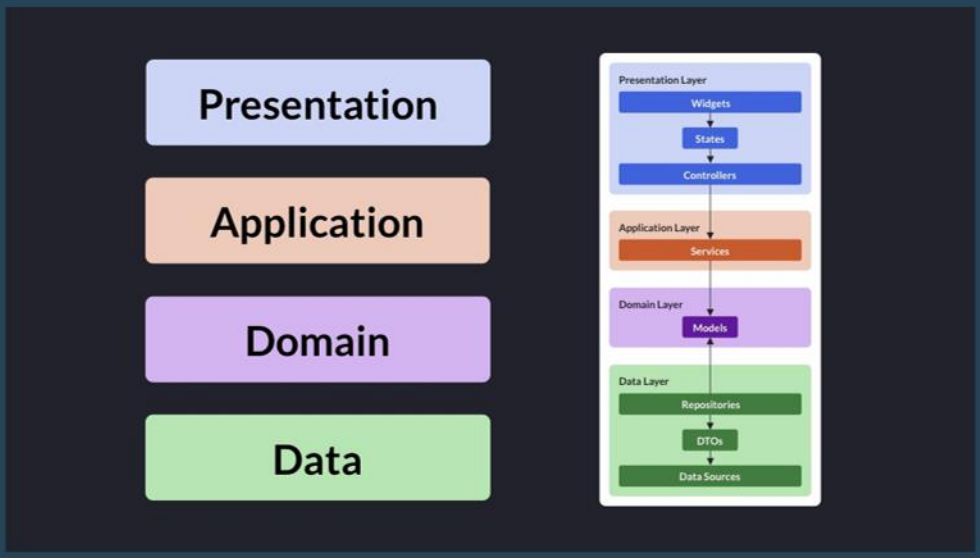
Діаграма варіантів використання

Проєктування архітектури та структури системи



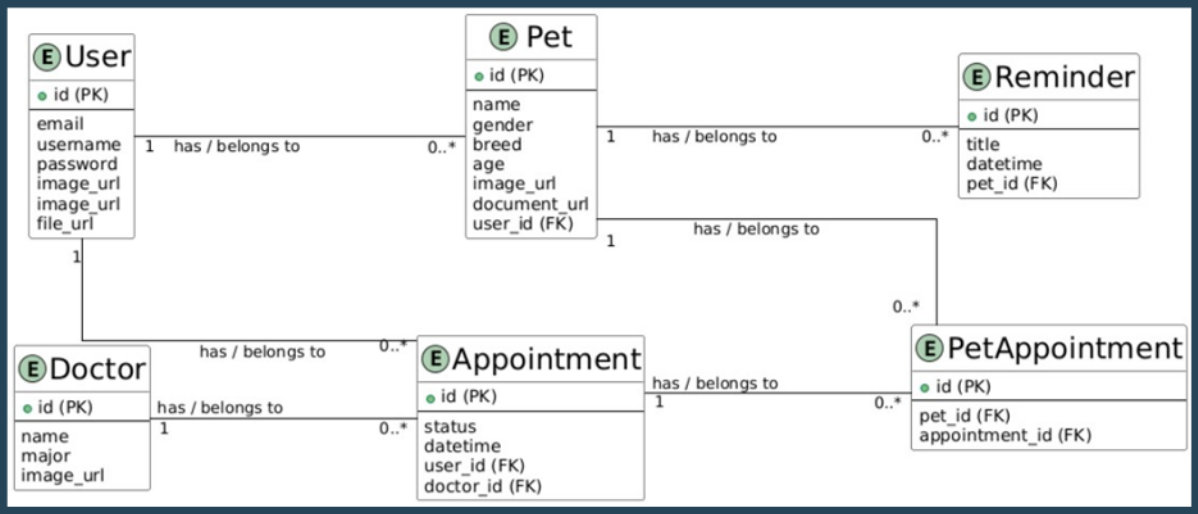
Зображення безсервернерної архітектури

Реалізація мобільного застосунку



Структура feature-first

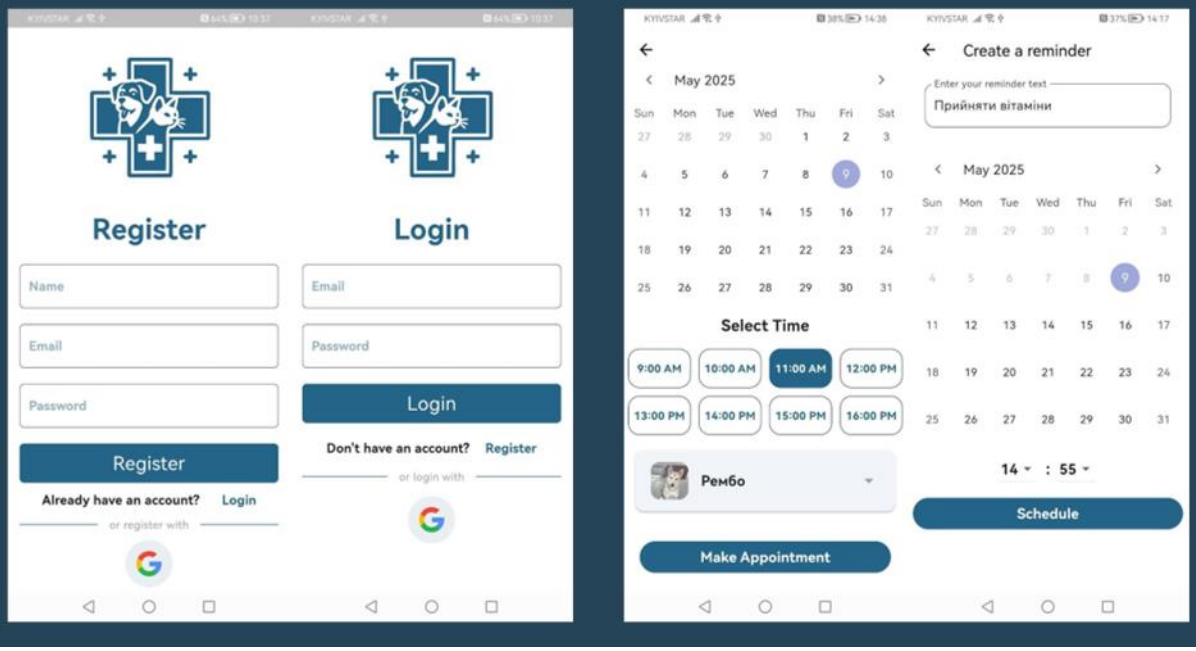
Проєктування структури даних та логічної моделі бази даних



ER-діаграма

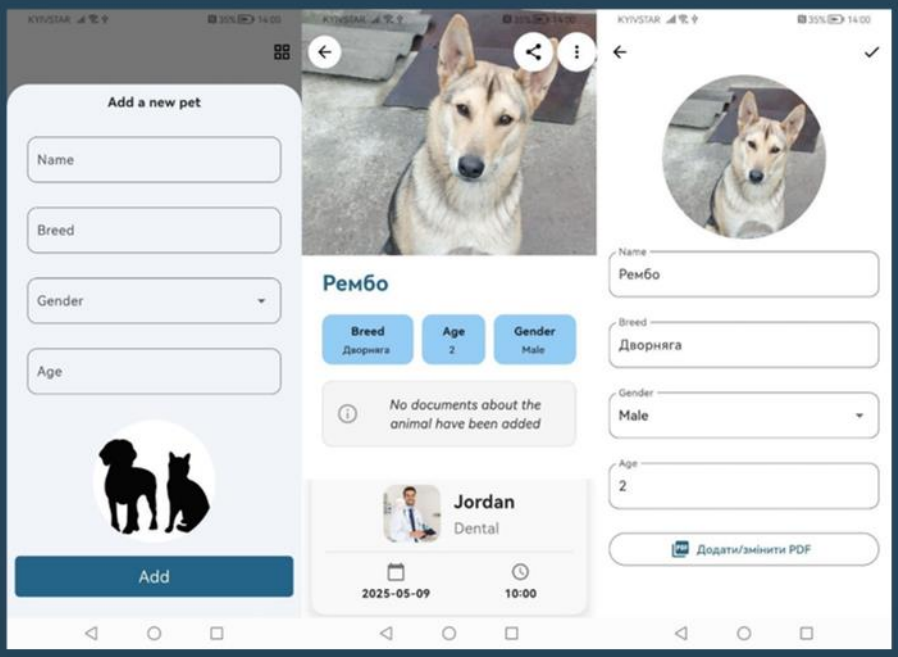
Проєктування інтерфейсу користувача

11



Проектування мобільного застосунку

12



Проектування мобільного застосунку

13

Аналіз та вибір технологій і методів реалізації застосунку

Dart

Мова програмування для Flutter

Flutter

UI-фреймворк для застосунків

Firebase

Сервіс для база даних та інших функцій

BLoC

Технологія для управління станом застосунку

Реалізація мобільного застосунку

14

```dart
abstract interface class PetRepositoryInterface {
  Future<Pet?> createPet(String uid,Pet pet);
  Future<List<Pet>> fetchAllPets(String uid);
  Future<void> updatePet(Pet updatePet, String uid);
  Future<void> deletePet(String uid,String petId);
}
class FirebasePetRepository implements PetRepositoryInterface {
  final FirebaseAuth firebaseAuth = FirebaseAuth.instance;
  final FirebaseFirestore firebaseFirestore = FirebaseFirestore.instance;
  final FirebaseStorage firebaseStorage = FirebaseStorage.instance;

  @override
  Future<Pet> createPet(String uid, Pet pet) async {…

 @override
  Future<void> deletePet(String uid, String petId) async {…

  @override
  Future<List<Pet>> fetchAllPets(String uid) async {…

  @override
  Future<void> updatePet(Pet updatePet, String uid) async {…
}
```

```dart
@override
Future<Pet> createPet(String uid, Pet pet) async {
    try {
      final petId = const Uuid().v4();
      final newPet = Pet( petId: petId,
      uid: uid,name: pet.name, breed:
      pet.breed,gender: pet.gender,
      age: pet.age,
      petImageUrl:
      pet.petImageUrl
      ?? ApplicationImages.petDefaultImage,
      petPdfUrl: pet.petPdfUrl ?? "", );
      await firebaseFirestore.collection('Users')
      .doc(uid).
      collection('Pets').doc(petId)
      .set(newPet.toJson());
      return newPet;
    } catch (e) {
      throw Exception('Failed to create an animal');
    }
}
```

Розробка бази даних

15

Firebase Collections Structure

Users

{uid} (Document)

```
uid :string
name :string
email :string
phone :string
password :string
bio :string
profileImageUrl :string
```

Pets

{petId} (Document)

```
petId :string
uid :string
name :string
breed :string
gender :string
age :number
petImageUrl :string?
petPdfUrl :string?
```

Appointments

{appointId} (Document)

```
appointId :string
uid :string
docId :string
petId :string
status :string
dateTime :timestamp
```

Doctors

{docId} (Document)

```
docId :string
name :string
major :string
docImagePath :string
bio :string
experience :number
```

Вимоги до технічних та програмних засобів

16

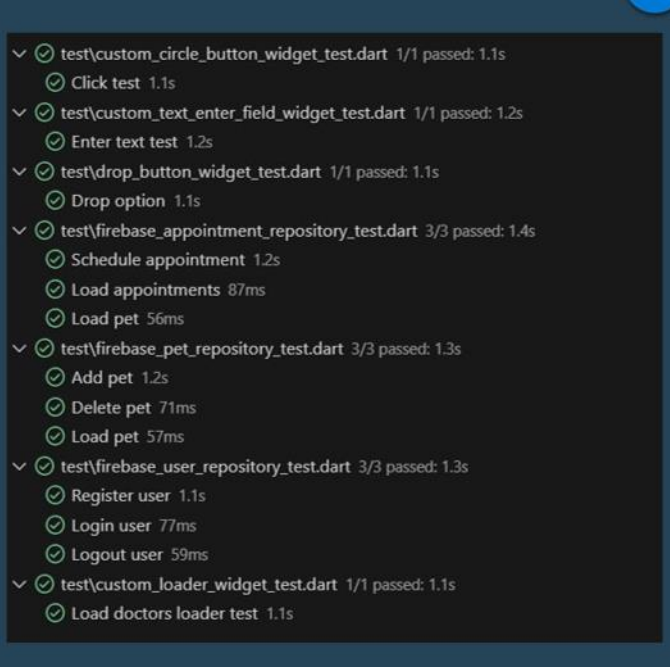
Категорія	Вимога
ОС пристрою	Android 8.0 (API 26)+ / iOS 13.0+
Процесор	64-бітний, мінімум 2 ядра
Оперативна пам'ять	Не менше 2 ГБ
Вільна пам'ять	Щонайменше 150 МБ
Інтернет-з'єднання	Стабільне Wi-Fi або мобільний інтернет (2G+)
Системне ПЗ	Android OS / iOS з підтримкою Google Play Services або Apple Services
Драйвери та утиліти	Камера, сховище, мережа (встановлюються автоматично системою)
Доступ до ресурсів	Камера, локальне сховище, мережа
Інтернет-залежність	Потрібне постійне з'єднання з інтернетом

Тестування

Mockito

Bloc_test

Patrol



Роботу було також опубліковано в ЗБІРНИК НАУКОВИХ ПРАЦЬ за матеріалами XVI Всеукраїнської науково-практичної конференції

«Актуальні проблеми комп'ютерних наук АПКН-2024»

УДК 004.4

Паламарчук Д.В., Праворська Н.І.

Хмельницький національний університет

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ЗАПИСУ ДО ВЕТЕРИНАРА ТА ДОГЛЯДУ ЗА ТВАРИНАМИ

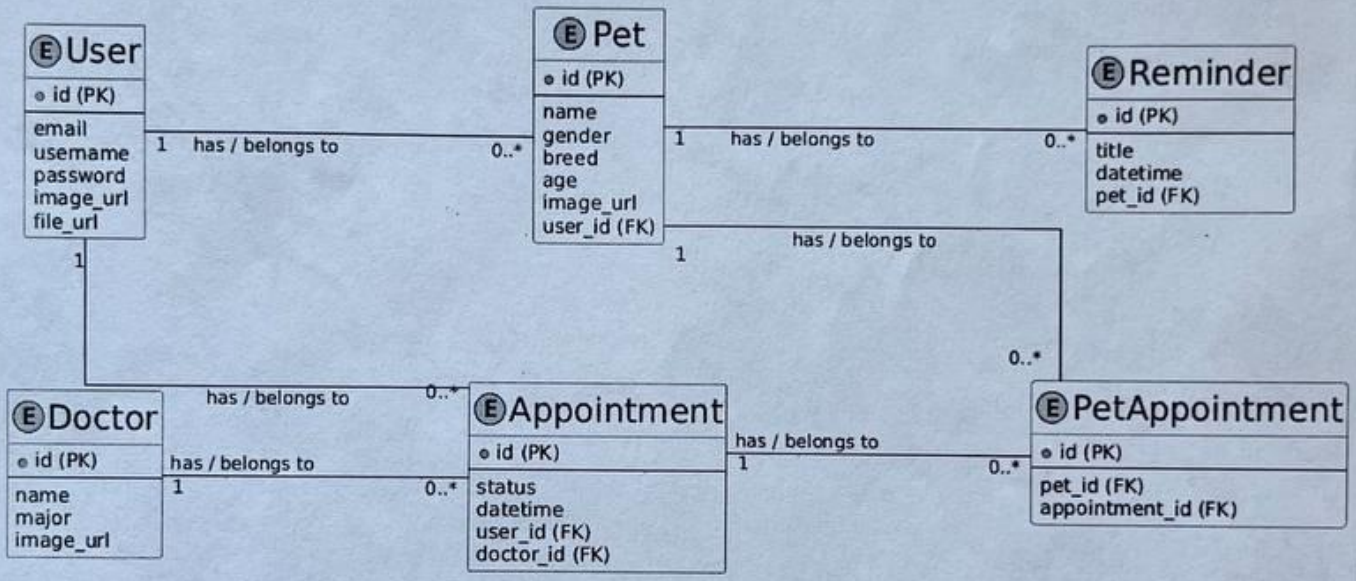
Розглянуто аспекти розробки мобільного застосунку для запису у ветеринара та догляду за тваринами. Запропоновано розробку зручного інтерфейсу, що дозволяє швидко записатися на візит до ветеринара, а також спрощує управління даними про тварин та нагадуваннях про заплановані процедури. В процесі розробки застосунку було проаналізовано основні переваги та недоліки мобільних застосунків у цій сфері, та на основі цього визначено їх позитивні та негативні сторони. На основі цього було розроблено вимоги до розроблюваного застосунку.

Aspects of developing a mobile application for making an appointment with a veterinarian and caring for animals. It is proposed to develop a user-friendly interface that allows you to quickly make an appointment with a veterinarian, as well as simplifies the management of animal data and reminders of planned procedures. In the process of developing the application, we analysed the main advantages and disadvantages of mobile applications in this area, and based on this, we identified their positive and negative aspects. Based on this, the requirements for the application were developed.

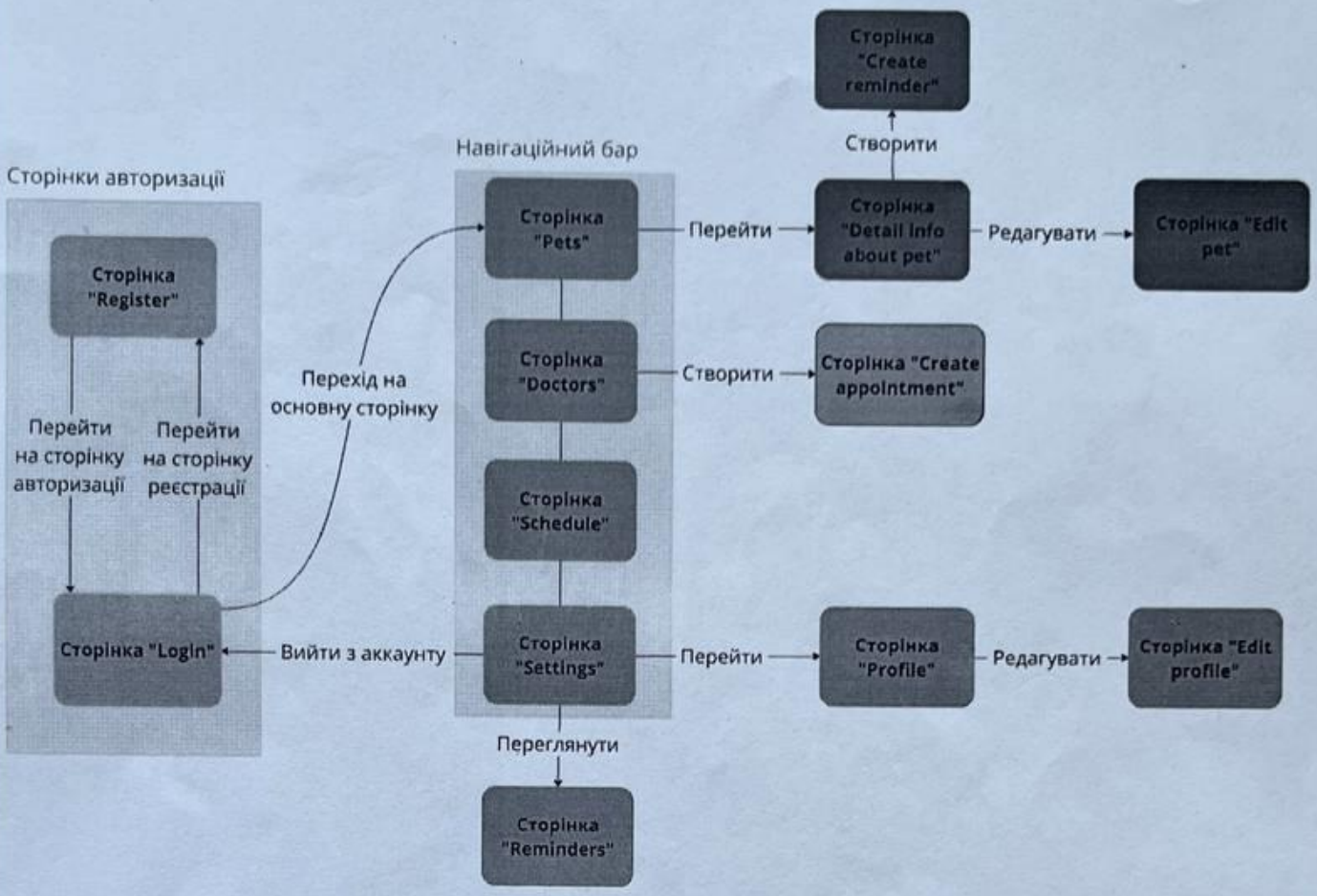
Висновки

Поставлене завдання	Результат виконання
Проаналізувати існуючі рішення	Проведено огляд застосунків 11 Pets, AnimalID, DogCatApp; виявлено їх переваги та недоліки
Визначити функціональні та нефункціональні вимоги	Сформульовано перелік вимог, зокрема реєстрація, нагадування, підтримка тем, безпека, масштабованість
Обґрунтувати вибір архітектури та структури системи	Обрано serverless-архітектуру з використанням Firebase, застосовано патерн BLoC для управління станом
Спроєктувати логіку та модель БД	Розроблено ER-діаграму з основними сутностями: Користувач, Тварина, Прийом, Нагадування, Ветеринар
Реалізувати інтерфейс користувача	Розроблено дизайн сторінок застосунку: авторизація, домашні тварини, лікарі, розклад, налаштування, профіль
Здійснити програмну реалізацію клієнтської та серверної частини	Використано Flutter (Dart) для клієнта, Firebase (Auth, Firestore, Cloud Functions, Storage) для серверної частини
Забезпечити захист персональних даних	Реалізовано аутентифікацію через Firebase Auth, хешування паролів, обмеження доступу через Firebase Security Rules
Реалізувати функцію збереження медичних записів і зображень	Реалізовано завантаження фото та документів у Firebase Storage, їх прив'язка до тварини
Провести тестування функціоналу	Проведено тестування UI та взаємодії з Firebase, підтверджено працездатність основних сценаріїв

ГРАФІЧНІ МАТЕРІАЛИ



					КвРІПЗ.2101082.01.10.Е8	
Змн.	Арк.	№ докум.	Підпис	Дата	Мобільний застосунок для запису до ветеринара та організації догляду за тваринами	Літ. / Маса. / Масштаб
Розробив		Паламарчук Д.В.		02.06		
Керівник		Праворська Н.І.		02.06		
Консультант		Калустян І.В.		0.206	Логічна модель бази даних	Аркуш 1 / Аркушів 2
Н. Контр.		Бедратюк Г.І.		02.06		ХНУ, ІПЗ-21-1
Зав. каф.		Бедратюк Л. П.		02.06		



					КвРІПЗ.2101082.01.10.Е8		
					Мобільний застосунок для запису до ветеринара та організації догляду за тваринами	Літ.	Маса.	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата				
Розробив		Паламарчук Д.В.		02.06				
Керівник		Праворська Н.І.		02.06				
Консультант		Конустян М.В.		02.06	Схема екранів застосунку	Аркуш 2	Аркушів 2	
Н. Контр.		Бедратюк Г.І.		02.06		ХНУ, ІПЗ-21-1		
Зав. каф.		Бедратюк Л. П.		02.06				

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Паламарчука Дениса Васильовича
факультет ІТ, IV курс, група ІПЗ-21-1

ЗАЯВА

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

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

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

07.06.25
дата

підпис

Anti-Plagiarism v-15.274 Educational

The maximum coincidence with one document 3.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 15%**

	Document		Sum coincidence on the DB	
ID: 241887 Title: ДП_Мобільний застосунок для запису у ветеринара та організації догляду за тваринами Added in a DB: 2025-05-26 Authors: Денис ПАЛАМАРЧУК Heads: канд. пед. наук, доцент Наталія ПРАВОРСЬКА Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	89505	1355	5172 (6%)	73 (5%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності науковим керівником

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

Автор: Паламарчук Денис

Співавтор:

Назва: БКР_Мобільний застосунок для запису у ветеринара та організації догляду за тваринами Паламарчук_Денис

Науковий керівник:

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:3.8%

Коефіцієнт подібності 2:0.9%

Мікропробіли: 0

Заміна букв: 1

Інтервали: 0

Білі знаки: 4

Дата створення звіту: 2025-05-28 19:46:09.0

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 05.28.2025

експерт

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «Бакалавр»

Дипломник Паламарчук Денис Васильович

Тема Мобільний застосунок для запису до ветеринара та організації догляду за тваринами

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 2 кількість сторінок записки 76

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі розглянуто створення мобільного застосунку для запису до ветеринара та організації догляду за тваринами. Проведено аналіз предметної області, сформульовано функціональні та нефункціональні вимоги, обґрунтовано вибір архітектурних та технологічних рішень. Реалізовано функціональний мобільний застосунок із можливістю запису на прийом, ведення профілю тварини, збереження медичних документів і нагадувань про догляд. Проведено тестування, яке підтвердило коректну роботу 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 У вступі обґрунтовано актуальність теми, сформульовано мету, завдання, об’єкт і предмет дослідження. У першому розділі проведено аналіз предметної області, розглянуто наявні сервіси, виявлено їхні переваги та недоліки, сформульовано функціональні та нефункціональні вимоги до застосунку. У другому розділі спроєктовано архітектуру системи на основі Serverless-підходу, побудовано логічну модель даних, ER-діаграму та розроблено інтерфейс з урахуванням вимог UX/UI. У третьому розділі реалізовано функціонал застосунку за допомогою Dart/Flutter і Firebase: створення профілів, облік тварин, запис на прийом, зберігання медичних даних та нагадувань. Проведено модульне та інтеграційне тестування. Результатом роботи став сучасний, зручний і масштабований мобільний застосунок, що відповідає поставленим вимогам і має потенціал для подальшого розвитку.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною, оскільки є потреба у зручному цифровому рішенні для організації ветеринарного догляду та зберігання інформації про тварин. Робота вирізняється всебічним аналізом аналогів, продуманим архітектурним проєктуванням, використанням сучасних технологій (Flutter, Firebase, SQLite, BLoC), а також орієнтацією на зручність і потреби кінцевого користувача

5. Негативні сторони роботи застосунок не реалізує розширену систему пошуку чи фільтрації за різними параметрами тварин або прийомів.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики проектування. Графічний матеріал дає можливість наочно побачити деталі проектування системи.

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 КІС, Капустянн Марія Вікторівна

"2" червня 2025 р. _____
(підпис)

SemanticAI for Education

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

першого освітнього рівня «Бакалавр»

Студента: Паламарчука Дениса Васильовича
Група: ІПЗ-21-1

Тема: *«Мобільний застосунок для запису до ветеринара та організації догляду за тваринами»*

Спеціальність: 121 – Інженерія програмного забезпечення

Короткий зміст пояснювальної записки

У вступі кваліфікаційної роботи акцентується на важливості мобільних застосунків у сучасному житті, зокрема у ветеринарній сфері. Актуальність теми обґрунтована потребою створення зручного мобільного застосунку для запису до ветеринара та моніторингу стану домашніх тварин. Мета роботи полягає в розробці інноваційного рішення, яке спростить взаємодію між власниками тварин та ветеринарними клініками. Для досягнення цієї мети визначено кілька завдань, включаючи аналіз існуючих рішень, визначення вимог, розробку архітектури та програмну реалізацію застосунку.

Відповідність отриманих результатів роботи поставленим завданням

Завдання, сформульовані у вступі, включають: 1. Аналіз існуючих мобільних застосунків для ветеринарних послуг. 2. Встановлення основних вимог до розроблюваного застосунку. 3. Вибір та розробка архітектури мобільного застосунку. 4. Програмна реалізація застосунку. 5. Проведення тестування.

Результати, викладені у висновках, свідчать про те, що: - Аналіз існуючих застосунків проведено, але з недоліками в деталізації. - Вимоги до застосунку визначено, проте їх реалізація потребує уточнень. - Архітектура застосунку описана, але не всі аспекти враховані. - Програмна реалізація виконана, але без достатньої кількості прикладів коду. - Тестування проведено, але результати не візуалізовані.

Таким чином, результати в цілому відповідають поставленим завданням, але є певні недоліки, які потребують доопрацювання. Оцінка відповідності: в цілому відповідають.

Оцінка розділів

Розділ 1

У розділі 1 представлено змістовний аналіз предметної області, що охоплює різноманітність типів мобільних застосунків та їх використання в різних сферах, зокрема у ветеринарії. Описано ринкові частки мобільних платформ, що підкреслює актуальність теми. Однак, аналіз потреб користувачів є поверхневим, а приклади успішних застосунків відсутні. Рекомендується додати більше деталей щодо специфічних потреб користувачів.

В цілому, розділ демонструє розуміння предметної області, але потребує глибшого аналізу.

Розділ 2

У розділі 2 розглянуто архітектуру та структуру системи, зокрема вибір архітектури, декомпозицію та взаємодію між модулями. Описано кілька архітектур, але не зазначено, яка з них була обрана для проекту. Враховано специфіку мобільної платформи, але без конкретних прикладів реалізації. Декомпозиція застосунку описана чітко, але не надано зв'язків між модулями та сценаріями використання.

Розділ демонструє розуміння архітектурних принципів, але потребує уточнень у виборі архітектури та деталізації реалізації.

Розділ 3

У розділі 3 детально описано програмну реалізацію та тестування. Описано структуру застосунку та реалізацію основних елементів, але відсутні конкретні приклади коду. Тестування проведено, але результати не візуалізовані, що ускладнює оцінку якості. Загалом, розділ містить важливу інформацію, але потребує покращення в деталізації та візуалізації.

Позитивні сторони

Кваліфікаційна робота демонструє оригінальність у виборі теми та підходу до розробки мобільного застосунку. Високий рівень аналітичного мислення проявляється в детальному описі предметної області та ринку. Використання сучасних технологій, таких як Flutter, свідчить про прагнення до інноваційності. Також позитивно оцінюється структурованість роботи та логічність викладу.

Недоліки

Серед недоліків варто відзначити недостатню глибину аналізу потреб користувачів, відсутність конкретних прикладів успішних застосунків, а також недостатню деталізацію реалізації функціональних вимог. Відсутність візуалізації результатів тестування та конкретних прикладів коду ускладнює сприйняття роботи.

Відгук в цілому

Тема роботи є актуальною та має практичну значущість, оскільки мобільні застосунки у ветеринарії можуть суттєво полегшити життя власникам тварин. Зміст роботи відповідає темі та завданню, але потребує доопрацювання в частині деталізації та обґрунтування. Новизна ідей та рішень присутня, але їх реалізація потребує уточнень. Об'єктивність та обґрунтованість викладу загалом на високому рівні.

Оцінка кваліфікаційної роботи

Кваліфікаційна робота заслуговує оцінки "добре". Вона виконана в повному обсязі, але має деякі недоліки, які потребують доопрацювання. Оцінка: **добре**.

Рекомендації

Рекомендується доопрацювати роботу, зокрема: - Додати більше деталей щодо потреб користувачів та прикладів успішних застосунків. - Уточнити вибір архітектури та надати приклади реалізації функціональних вимог. - Візуалізувати результати тестування та надати більше прикладів коду для ключових функцій. - Чітко сформулювати основну задачу розробки та обґрунтувати її актуальність.

OpenAI API-асистент
Session ID: f31a2373-945b-4edb-95fe-95734972d7c6
Підписано автоматично, модель gpt-4o-mini
Дата: 26.05.2025

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

Назва кваліфікаційної роботи Мобільний застосунок для запису до ветеринара та організації догляду за тваринами

Автор Паламарчук Денис Васильович

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Рівень вищої освіти Перший (бакалаврський)

Спеціальність 121 «Інженерія програмного забезпечення»

Науковий керівник: Праворська Наталія Іванівна, канд. пед. наук, доцент

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

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

Підтвердження:

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

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, завдання, анотація, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій та у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 3,8%, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 02.06.25

Завідувач кафедри _____ Підпис — Леонід БЕДРАТЮК Ім'я, ПРІЗВИЩЕ

Гарант освітньої програми _____ Підпис — Леонід БЕДРАТЮК Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Підпис — Наталія ПРАВОРСЬКА Ім'я, ПРІЗВИЩЕ