ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ КОНТРОЛЮ ЗДОРОВ'Я ДОМАШНІХ ТВАРИН МОВОЮ C#**»

Виконала: студентка 4 курсу, групи ПД-41 спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності/спеціалізації)

Смирнова К.Б
(прізвище та ініціали)

Керівник Негоденко О.В.
(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення Ступінь вищої освіти - «Бакалавр»

Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Негоденко О.В.
"____" _____ 2023 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТА

СМИРНОВОЇ КАТЕРИНИ БОРИСІВНИ

(прізвище, ім'я, по батькові)

1. Тема роботи: « Розробка програмного забезпечення для контролю здоров'я домашніх тварин мовою C#»

Керівник роботи: Негоденко О.В. к.т.н., доц.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «24» лютого 2023 року №26.

2. Строк подання студентом роботи «01» червня 2023 року

3. Вхідні дані до роботи

3.1 Технічна документація Microsoft

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).

4.1. Аналіз предметної області

4.2. Технічне завдання

4.3. Засоби розробки

4.4. Вимоги до програмного забезпечення

4.5. Моделювання архітектури системи

4.6. Розробка мапи додатку
4.7. Опис розробки додатку
4.8. Висновки
5. Перелік демонстраційного матеріалу (назва основних слайдів)
5.1. Титульний слайд.
5.2. Мета, об’єкт, предмет, наукова новизна дослідження.
5.3. Актуальність.
5.4. Аналіз аналогів.
5.5. Технічні завдання.
5.6. Програмні засоби та інструменти реалізації.
5.7. Розробка архітектури.
5.8. Реалізація програми.
5.9. Висновки
5.10. Апробація результатів дослідження.
5.11. Кінцевий слайд.
6. Дата видачі завдання «25» лютого 2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	11.04-14.04	Виконано
2	Аналіз та дослідження існуючих аналогів	15.04-17.04	Виконано
3	Проектування системи	18.04-21.04	Виконано
4	Створення та тестування програмного рішення	21.04-05.05	Виконано
5	Підготовка розділу 1	05.05-07.05	Виконано
6	Підготовка розділу 2	07.05-09.05	Виконано
7	Підготовка розділу 3	08.05-11.05	Виконано
9	Вступ, висновки, реферат	11.05-12.05	Виконано
10	Розробка обов‘язкових демонстраційних матеріалів	12.05-15.05	Виконано
11	Попередній захист роботи та перевірка на плагіат	19.05-25.05	Виконано
12	Здача роботи	01.06	Виконано

Студент _____ Смирнова К.Б.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Негоденко О.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 46 с., 31 рис., 1 табл., 2 дод., 17 джерел.

C#, Xamarin, .NET, MVVM

Об'єкт дослідження – процес контролю здоров'я домашніх тварин.

Предмет дослідження – мобільний додаток для контролю здоров'я домашніх тварин.

Мета роботи – підвищення ефективності контролю здоров'я домашніх тварин за допомогою мобільного додатку мовою C#.

Методи дослідження – методи передачі, зберігання та обробки інформації, уніфікований процес створення програмного забезпечення, емпіричні методи.

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Проаналізувати вже існуючі програмні забезпечення, знайти їх головні переваги та недоліки.

2. Розробити функціональні та нефункціональні вимоги до додатку, грунтуючись на проаналізованих схожих додатках.

3. Дослідити інструменти для розробки додатку, розробити модель архітектури додатку та його дизайну.

4. Розробити додаток на основі обраних інструментів та завчасно визначених вимог.

5. Провести тестування додатку.

6. Пройти апробацію на Науково-технічних конференціях.

Практичне значення результатів: Даний додаток може бути використано для персональних цілей при наявності домашнього улюбленця.

Галузь використання – догляд за домашніми тваринам

ЗМІСТ

ВСТУП

Обґрунтування вибору теми та її актуальність: У контексті сучасності важливо розуміти тенденцію розвитку свідомості людей. Людина сучасності турбується не тільки про себе, про своє здоров’я, а також про інших, особливо за тих, кого вона приручила, а саме, домашніх улюбленців. Взявши до уваги український ринок додатків, можна зробити висновок, що існує тільки один додаток, який може хоча б на якийсь відсоток вирішити проблему контролю здоров’я тварини. Але все ж таки в ньому існує дуже мало функціоналу саме для нотування інформації відносно свого улюбленця, створення нагадування щодо ліків, щеплень, візитів до ветеринарів, рекомендації для різних видів тварин тощо. Для вирішення цієї дилеми було розроблено мобільний додаток, який може допомогти власникам тварин більш детальніше та зручніше контролювати стан здоров’я їх улюбленців.

Об’єкт дослідження – процес контролю здоров’я домашніх тварин.

Предмет дослідження – мобільний додаток для контролю здоров’я домашніх тварин.

Мета роботи – підвищення ефективності контролю здоров'я домашніх тварин за допомогою мобільного додатку мовою C#.

Методи дослідження – методи передачі, зберігання та обробки інформації, уніфікований процес створення програмного забезпечення, емпіричні методи.

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Проаналізувати вже існуючі схожі програмні забезпечення, знайти їх головні переваги та недоліки, а також звести таблицю необхідного функціонала.

2. Розробити функціональні та нефункціональні вимоги до додатку грунтуючись на проаналізованих схожих додатках.

3. Дослідити інструменти для розробки додатку, розробити модель архітектури додатку та його дизайну;

4. Розробити додаток на основі обраних інструментів та завчасно визначених вимог;

5. Провести тестування додатку;

6. Пройти апробацію на Науково-технічних конференціях;

Практичне значення результатів: Даний додаток може бути використано для персональних цілей при наявності домашнього улюбленця.

Для розробки застосунку використовуються такі інструменти як: мова програмування C#, платформа Xamarin, база даних SQLite, середа розробки Visual Studio, а також система контролю версій git.

Практична значущість результатів: Розроблений додаток надає користувачу можливість мати всю важливу інформацію, яка стосується його улюбленця, у швидкому доступі на його пристрої. Також додаток має базовий потрібний результат, який допоможе вирішити проблему нестачі схожих додатків для контролю здоров'я домашніх тварин.

Галузь використання – догляд за тваринами.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Аналіз предметної області

Важливим аспектом розробки програмного забезпечення є аналіз предметної області. Так як предметною областю додатку, що розробляється, є створення додатку для контроля здоров’я домашніх тварин, треба дослідити сфери догляду за тваринами, а також знайти та проаналізувати вже існуючі додатки зі схожим функціоналом, виявити їх переваги та недоліки.

Сфера догляду за домашніми тваринами досі складна, та власники, які тільки розпочали свій шлях з домашніми тваринами, можуть зіткнутися з великою кількістю проблем, на які натрапляють навіть досвідчені власники тварин. Основними складнощами можуть бути: часові обмеження, медичний догляд, проблеми поведінки, проблеми зі здоров’ям тварини тощо. Розглянувши кожен пункт окремо можна сказати про них більш детальніше та спробувати мінімізувати їх ефект.

До часових обмежень може відноситися щільний графік власника тварини. Наразі важко уявити своє життя без навчання, роботи та інших побутових справ, якими ми займаємось кожного дня, але у цей графік також треба додати час для своїх улюбленців, що буває дуже складно. Задля цього треба мати інструмент, який допоможе зробити графік тварини та налаштувати нагадування.

Якщо розглянути пункт медичного догляду, то домашні тварини потребують постійного графіку вакцинації, обробки від кліщів, а також інших процедур. Для цього потрібно мати щоденник, де це все буде занотовано, а також будуть нагадування про те, що скоро закінчується термін вакцини.

Проблеми зі здоров’ям тварини – це основний пункт, за яким власники тварин здійснюють пошук зручних інструментів контролю здоров’я. Атопічний дерматит, авітаміноз, діабет, артрит, серцево-судинні та інші захворювання за якими слідкувати та надавати своєчасне лікування. У таких випадках власникам домашніх улюбленців потрібно вести нотатки та календар подій, який буде давати

можливість чітко контролювати прийом ліків, записи до ветеринарів та інші процедури.

1.2 Animal ID

Додаток Animal ID – це програмне забезпечення для контролю здоров'я домашніх тварин, яке дозволяє власникам тварин слідкувати за станом здоров'я своїх улюбленців. Додаток пропонує широкий спектр функцій, які дозволяють власникам тварин зберігати всю необхідну інформацію про своїх улюбленців та ефективно контролювати їхнє здоров'я. Крім цього, Animal ID є базою реєстрації тварин, до якою можна додати QR паспорт тварини [1].

За допомогою додатку Animal ID власники тварин можуть створювати профілі своїх улюбленців, додавати фотографії, основну інформацію стосовно чіпів, вакцинації та іншого. Також є можливість відстежувати візити до ветеринарних клінік та зберігати медичні записи про кожен візит.

Однією з ключових особливостей додатку Animal ID є можливість вести детальний журнал здоров'я тварини. За допомогою журналу власники можуть детально стежити за здоров'ям своїх улюбленців, фіксувати стан їхнього здоров'я та відстежувати будь-які зміни в стані здоров'я. Наприклад, власники можуть реєструвати останнє щеплення, візит до ветеринара, вагу та інші важливі дані своїх тварин.

Через додаток можна додати тварину у базу розшуку. Якщо на нашийнику тварини був її QR паспорт, то будь-хто може просканувати код через телефон та перейти на публічний профіль тварини, де вже буде уся важлива інформація про тварину та її власника. Коли QR код буде просканово, то власник тварини отримує повідомлення, де було просканово код та у який час.

Також у додатку реалізован функціонал планування подій на 90 днів. Ось повний список функцій:

- Обробка від паразитів
- Тренування

- Контроль ваги
- Грумінг
- Огляд ветеринара
- Вакцинація
- Лікування
- Харчування
- Вечірка
- Моя подія

Дана сторінка додатку буде дуже корисна власникам тварин, яким треба ретельно слідкувати за здоров'ям тварини, але також можна використовувати цей функціонал для того, щоб у своєму щільному графіку не забути про справи домашнього улюбленця (див. рисунок 1.1 – 1.3).

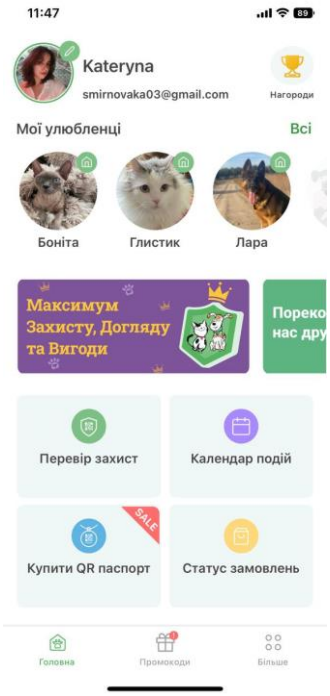


Рисунок 1.1 – Головна сторінка додатку Animal ID

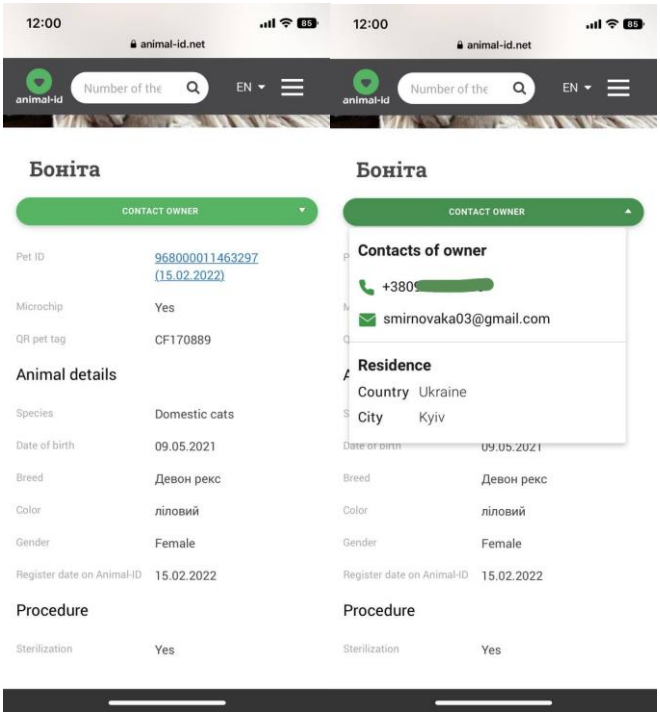


Рисунок 1.2 – Скріншот публічної сторінки домашньої тварини

Рисунок 1.3 – Скріншот календаря подій на наступні 90 дні

Переваги Animal ID:

- Легкий доступ до медичної інформації

- Детальний моніторинг за здоров'ям
- Дуже відомий додаток серед власників тварин
- Загальнодоступна база тварин
- Можливість додавання QR паспортів
- Календар подій на наступні 90 днів

Недоліки Animal ID:

- Багато платного функціоналу такого як SMS, що QR паспорт було просканово
- Є тільки два види тварин: собаки та коти
- Немає рекомендованої інформації для власників тварин
- Немає можливості слідкувати за раціоном тварин

1.3 VitusVet Pet Medical Records

Додаток VitusVet дозволяє легко відстежувати стан здоров'я домашнього улюбленця, збираючи всю необхідну інформацію про нього в одному місці. Це чудовий інструмент для тих, у кого є один або декілька домашніх улюбленців, для тих, хто подорожує з домашніми вихованцями, а також для тих, хто користується послугами кількох спеціалістів з догляду за тваринами, включаючи ветеринарів, грумерів, нянь і т.д. [2]

У додатку можна користуватися такими функціями як:

- Контроль за здоров'ям улюбленця

Через додаток можна контролювати вагу тварини, відстежувати його ліки, реєструвати дані про мікрочіп і страховку, а також відмічати будь-які алергії або медичні попередження для швидкого реагування.

- Доступ до інформації про вашого улюбленця в будь-який час

Можливість отримання доступу до медичних записів улюбленця на телефоні. Ця інформація знаходиться всього в декількох кліках в додатку. Можна легко поділитися нею з іншими спеціалістами з догляду за тваринами.

- Список справ для управління здоров'ям домашнього улюбленця

Встановлення нагадування про все, що потрібно пам’ятати про тварину – від дачі прийому ліків до запланованих зустрічей, купівлі корму тощо.

- Запис на прийом та поповнення запасів ліків за рецептом

Якщо ветеринарна клініка співпрацює з цим додатком, користувач може легко записуватись на прийом, поповнювати запаси ліків та продуктів безпосередньо через додаток (див. рисунок 1.4 – 1.5).

Рисунок 1.4 – Скріншот головної сторінки додатку VitusVet

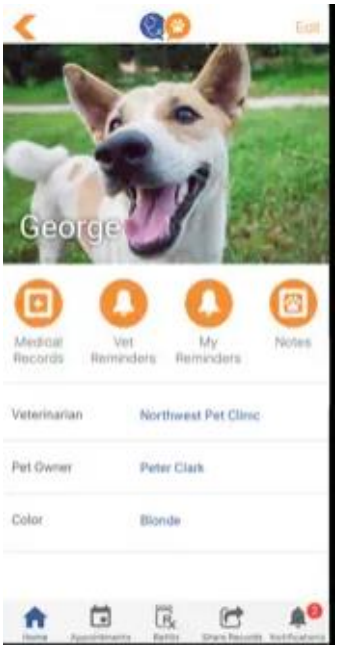


Рисунок 1.5 – Скріншот профілю тварини у додатку VitusVet

Переваги VitusVet:

- Легкий доступ до медичної інформації
- Моніторинг за здоров'ям
- Можливість додавати користувачів через email, щоб вони могли бачити користувача тварин та він міг бачити їх
- Можливість ділитися медичними нотатками тварини

Недоліки VitusVet:

- Немає адаптивного дизайну
- У додатку довго завантажується інформація
- Немає можливості чітко контролювати раціон тварини
- Немає рекомендації для власників тварин

1.4 PitPat

PitPat це додаток для контролю здоров’я домашнього улюбленця. За для більш точного контролю, компанія рекомендує користуватися їх PitPat Activity Monitor. Завдяки цьому пристрою у додаток будуть додаватися активності собаки, її можна буде бачити на карті через додаток у реальному часі. Крім цього у застосунку можна додавати вагу, раціон їжі, страховку. Також на головній сторінці можна отримати статистику відносно активності собаки [3].

Якщо більш детальніше розібрати прилади, які покращують роботу з додатком, то можна виділити 2 пристрої від компанії PitPat: Dog Activity Monitor та Dog GPS Tracker. До їх спільного функціоналу входять такі функції як:

- Повнофункціональний моніторинг активності
- Відстеження відстані
- Відстеження калорій
- Точні рекомендації щодо годування
- Контроль ваги
- Повна безкоштовна інтеграція з додатком

Тому можна зробити висновок, якщо власнику тварини дуже важливо отримувати детальну статистику навантажень тварини, витрачених калорій, кількість відпочинку і т.п. , то у доповнення до додатку треба мати ще такі пристрої (див. рисунок 1.6 – 1.7).

Крім цього, треба звернути увагу на функцію “поділитися профілем собаки”. Це можна зробити через email. Саме цей функціонал буде корисним для тих, хто залишає своїх улюбленців на інших людей та хоче, щоб ті також мали доступ до всієї інформації про вихованця.

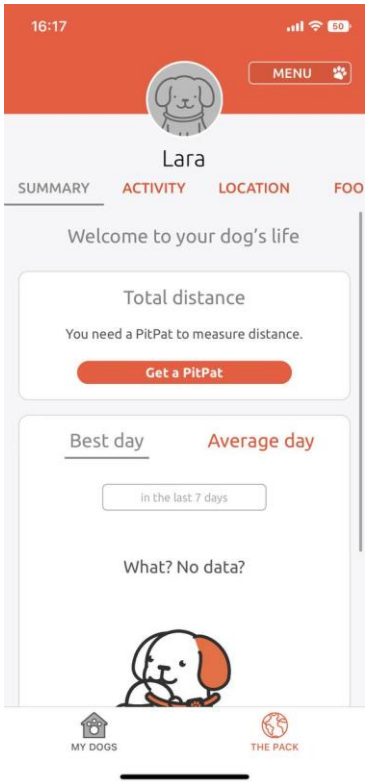


Рисунок 1.6 – Скріншот головной сторінки додатку PitPat

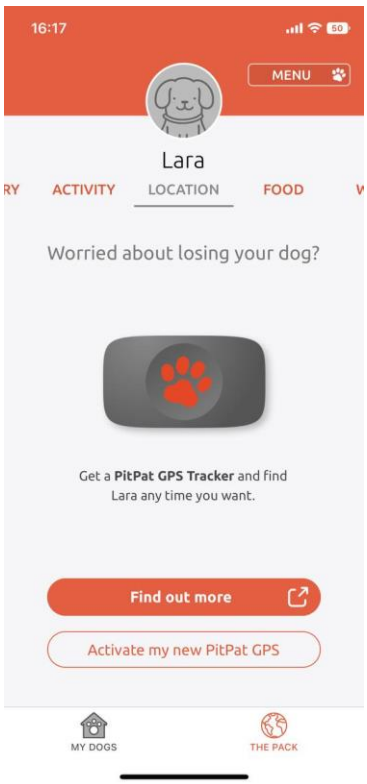


Рисунок 1.7 – Скріншот сторінки локації домашнього улюбленця

Переваги PitPat:

- Можна замовити у PitPat трекер для собаки та контролювати її активність
- Можна замовити у PitPat миску-ваги для собаки, щоб контролювати розмір порції
- Є наглядний графік активностей тварини
- Можна відслідковувати актуальне місцезнаходження через додаток завдяки трекеру
- Можна отримати рекомендації для раціону їжі, але тільки на території Великої Британії
- Додавання страховки тварини, але тільки на території Великої Британії

Недоліки PitPat:

- Тільки для власників собак
- Більшість функціоналу доступно тільки на території Великої Британії

1.5 Таблиця порівняння

За результатами аналізу аналогів можна зробити висновок, що вони мають значну кількість переваг, але при цьому є певні спільні недоліки, усунувши які можна отримати дійсно зручне рішення для власників тварин. Складемо таблицю (див. табл. 1) для порівняння аналогів з нашим додатком, де “+” – це наявність функціоналу, “-” – відсутність.

Таблиця 1. – Порівняння аналогів з нашим додатком

Функціонал	**Animal ID**	**VitusVet**	**PitPat**	**Happy Pet**
Функція «щоденника» для кожного профілю улюбленця для відслідковування важливих події та можливість нагадування	+	-	-	+
Контроль за раціоном тварини	-	-	+	+
Можливість додавання користувачів через email до інформації про вихованця	-	+	+	+
Можливість додавання документів до додатку	+	-	-	+
Пошук ветлікарень за запитами користувача	-	-	-	+
Можливість дізнатися місцезнаходження вихованця у реальному часі	-	-	+	-

Можливість додавати важливу інформацію стосовно алергічних проявів на профіль тварини	-	-	-	+
Пошук зоомагазинів за запитами користувача	-	-	-	+
Великий список видів тварин (більше ніж пес, кіт, інше)	-	-	-	+

Під час аналізу предметної галузі та аналогів, виявлено, що розроблюваний продукт немає абсолютних аналогів, тому в даному розділі було розглянуто продукти, які мають схожий функціонал та галузь використання. За результатами порівняння було складено таблицю, яка описує ключовий функціонал, що повинен бути реалізований.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Технічне завдання

Розробка програмного забезпечення для контролю за здоров’ям домашніх тварин у вигляді мобільного додатку. За допомогою використання таких технологій як: .NET 4.7.2, C#, Xamarin, Xamarin Remouted Simulator, Google Android Emulator(Android SDK API lvl25/lvl30), MVVM паттерн для розробки мобільного додатку на основі Xamarin.Forms, для роботи з локальними базами даних використання SQLite та функціонал SQLiteConnection, використання XAML для створення UI елементів та написання кастомізації базових елементів операційної системи Android. Мобільний додаток повинен бути розроблений із використанням систем адаптивності, для більш красивого представлення на різних пристроях.

Як середовище розробки використовувати Visual Studio 2019, 2022, Google Android Emulator для розробки та налагодження у реальному часі . Для системи контролю версій буде використовуватися GitHub приватний репозиторій.

2.2 Засоби розробки

Мова програмування C#. Мова програмування C# є об’єктно-орієнтованою мовою, яка було розроблена компанією Microsoft у 2000 році як частина платформи .NET [4]. Її перша версія була випущена у 2002 році разом з першою версією середовища розробки Microsoft Visual Studio .NET.

C# дуже потужна та гнучка мова програмування, що дозволяє розробникам створювати різноманітні додатки, включаючи веб-сайти, додатки для настільних комп'ютерів, мобільні додатки та ігри. C# також має розширену підтримку для об'єктно-орієнтованого програмування, що сприяє покращенню якості коду та зручності його підтримки.

Кожна мова програмування має свої позитивні та негативні аспекти використання, що варіюється від сфери використання, досвіду роботи, а також ще багатьох факторів.

Платформа Xamarin. Xamarin – open source система для розробки кросплатформенних додатків на таких ОС як: IOS, Android, Windows. Працюючий на базі .Net та мови програмування C#, він виконує обов'язки абстракції, котра керує зв'язком між базовим кодом та кодом кожної системи. Виражаючись простіше, код написаний на C# буде за допомогою системи інтерпретовано до коду системи на який працює додаток, інтерпретація відбувається завдяки віртуальному образу котрий створює Xamarin на базі будь-якої ОС, C# віддає команди до віртуальної середи і вже вона віддає команди які зрозуміє цільова ОС. Так як він працює на базі .Net, ми маємо такі потужні інструменти як розподіл пам'яті та збирання і контроль сміття [5]. Використання цієї платформи прискорює розробку завдяки тому, що можна використовувати весь код на всіх платформах (див. рисунок 2.1).

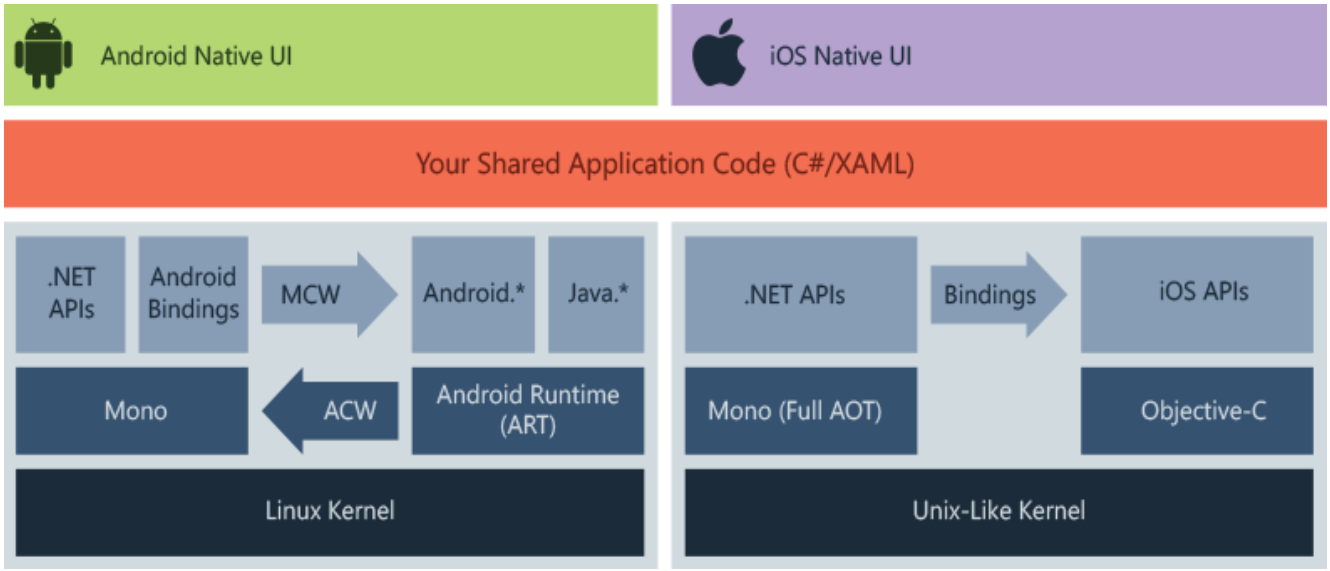


Рисунок 2.1 – Діаграма архутектури Xamarin

Xamarin поєднує в собі можливості нативних платформ, додаючи такі функції, як:

- Повне прив’язування базових SDK – платформа містить прив’язки майже для всіх базових SDK платформи в iOS і Android.
- Взаємодія Objective-C, Java, C і C++ – платформа надає можливості для прямого виклику бібліотек Objective-C, Java, C і C++, завдяки чому можна використовувати сторонній код.
- Xamarin використовує потужну колекцію класів .Net BCL, котра має комплексні та оптимізовані функції, такі як потужний XML, база даних, серіалізація, підтримка вводу-виводу, рядків, мережі тощо.
- Мобільна крос-платформна підтримка

Приклад роботи мобільного додатку Xamarin на базі Android. Після збору загального проекту (тобто додатку), він потрапляє у ізольоване середовище Xamarin.Android, звідки і виконується зв’язок C# із Java. Завдяки віртуальному середовищу забезпечується прив’язка .Net до функціоналу Android.Java. Віртуальна середа звертається до просторів імен завдяки MCW і надає ART обгортки Android Callable(ACW) , ця система дає змогу обом середовищам викликати код один в одного (див. рисунок 2.2).

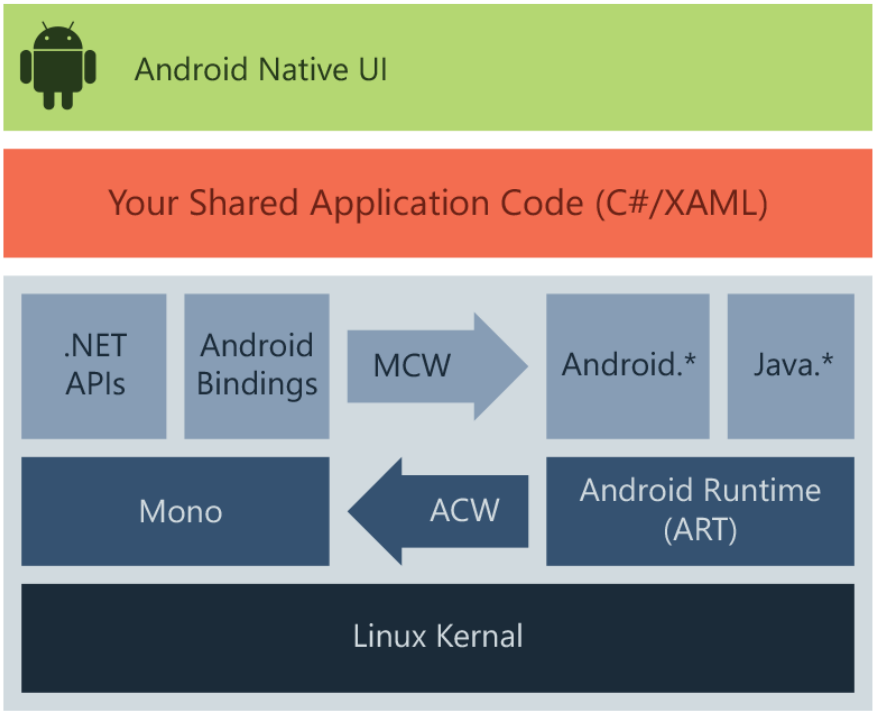


Рисунок 2.2 – Діаграма архутектури Xamarin.Android

Декларативна мова розмітки XAML. XAML – це декларативна мова розмітки. Застосовуючи до моделі програмування .NET, XAML спрощує створення інтерфейсу користувача для .NET-додатків. Ви можете створювати видимі елементи інтерфейсу в декларативній розмітці XAML, а потім відокремити визначення інтерфейсу від логіки виконання за допомогою файлів code-behind, які приєднуються до розмітки за допомогою часткових визначень класів. XAML безпосередньо представляє екземпляри об'єктів у певному наборі базових типів, визначених у збірках. Це відрізняється від більшості інших мов розмітки, які зазвичай є інтерпретованими мовами без такої прямої прив'язки до системи типів підкладки. XAML забезпечує робочий процес, в якому окремі сторони можуть працювати над інтерфейсом і логікою програми, використовуючи потенційно різні інструменти [6].

Представлені у вигляді тексту, файли XAML є файлами XML, які зазвичай мають розширення .xaml. Файли можуть бути закодовані будь-яким кодуванням XML, але типовим є кодування UTF-8.

База даних SQLite. Sqlite – це бібліотека на мові програмування C, котра реалізує невеликий, швидкий, автономний, надійний, функціональний механізм бази даних SQL. Ця СУБД є найпоширенішою у світі. Формат файлу є стабільним, кросплатформним та зворотньо сумісним. SQLite немає окремого серверного процесу, він читає та записує напряму у звичайні файлі у фізичному просторі. Повна база даних SQL з кількома таблицями, індексами, тригерами та поданнями міститься в одному файлі на диску. Формат файлу бази даних є кросплатформним – ви можете вільно копіювати базу даних між 32-розрядними і 64-розрядними системами або між архітектурами з прямим порядком байтів і прямим порядком байтів [7].

Середа розробки Visual Studio. Visual Studio – це інтегроване середовище розробки програмного забезпечення, яке було розроблено компанією Microsoft. У цю IDE також інтегровано систему контролю версій Git, що полегшує роботу разом, коли декілька розробників працюють над одним проєктом. Крім цього, є функція Live Share, яка також допомагає розробляти одночасно продукт у

реальному часі [8].

Система контролю версій – це система, що записує зміни стану файлів або груп файлів, через яку потім можна буде повернутися на будь-який потрібний стан [9].

Git зберігає дані у вигляді "комітів", які містять інформацію про змінені файли, їхній вміст та метадані, такі як автор коміту та дата його створення.

Git дозволяє використовувати гілки (branches), які дозволяють розробникам працювати паралельно над різними аспектами проекту. Крім того, Git дозволяє злити (merge) гілки, що дає змогу команді розробників об'єднувати роботу над різними гілками та розв'язувати конфлікти, що можуть виникати при роботі паралельно над однією гілкою.

2.3 Вимоги до програмного забезпечення

Безпека

- Програмне забезпечення повинне притримуватися стандартів ISO/IEC 27000-27008 [10] та Закона України “Про інформацію” [11].

Сумісність

- Програмне забезпечення повинно бути сумісне з операційною системою Android.

Ергономіка

- Програма повинна керуватися одним пальцем руки.

Зручність використання

- У додатку повинно бути меню зі сторінками додатку.
- Інтерфейс користувача повинен використовувати червоний колір та вікно підтвердження для дій пов’язаних з видаленням ресурсу.
- У додатку повинен бути popup на випадок невдалого доступу до бази даних.
- Додаток повинен мати функцію оновлення сторінок без втручання користувача.

Надійність

Система повинна задовільнять наступним вимогам надійності:

- Система повинна приймати запити та відповідати на них при виникненні неопрацьованих помилок.
- Система повинна приймати запити від користувачів та давати на них відповідь.

Масштабованість

- При випуску програмного забезпечення повинні використовуватися хмарні сховища даних та серверні технології задля збільшення продуктивності додатку.

Тестування

- Програмне забезпечення повинно бути повністю протестовано до випуску. Оновлення не можуть випускатися, якщо вони не пройшли валідацію або мають критичні помилки для користувача . Крім цього, застосунок буде покрито unit тестами.

2.4 Моделювання архітектури системи

Моделювання архітектури системи надає нам візуальне розуміння як різні частини додатку пов’язані між собою, їх процеси між ними, а також візуалізація бізнес–логіки та бізнес-процесів. Головне завдання у процесі виконання цього моделювання – це розробка зрозумілої та чіткої діаграми, яка показує всі елементи системи та зв’язки між ними. Було обрано декілька діаграм, які хотілось би висвітлили у процесі проєктування розробки.

Діаграма класів показує внутрішню структуру системи додатку, класи, які успадковуються та їх внутрішню структуру, параметри, методи, конструктори, а також класи уявлення даних. У нашому випадку можна побачити структуру управління базами даних у додатку.

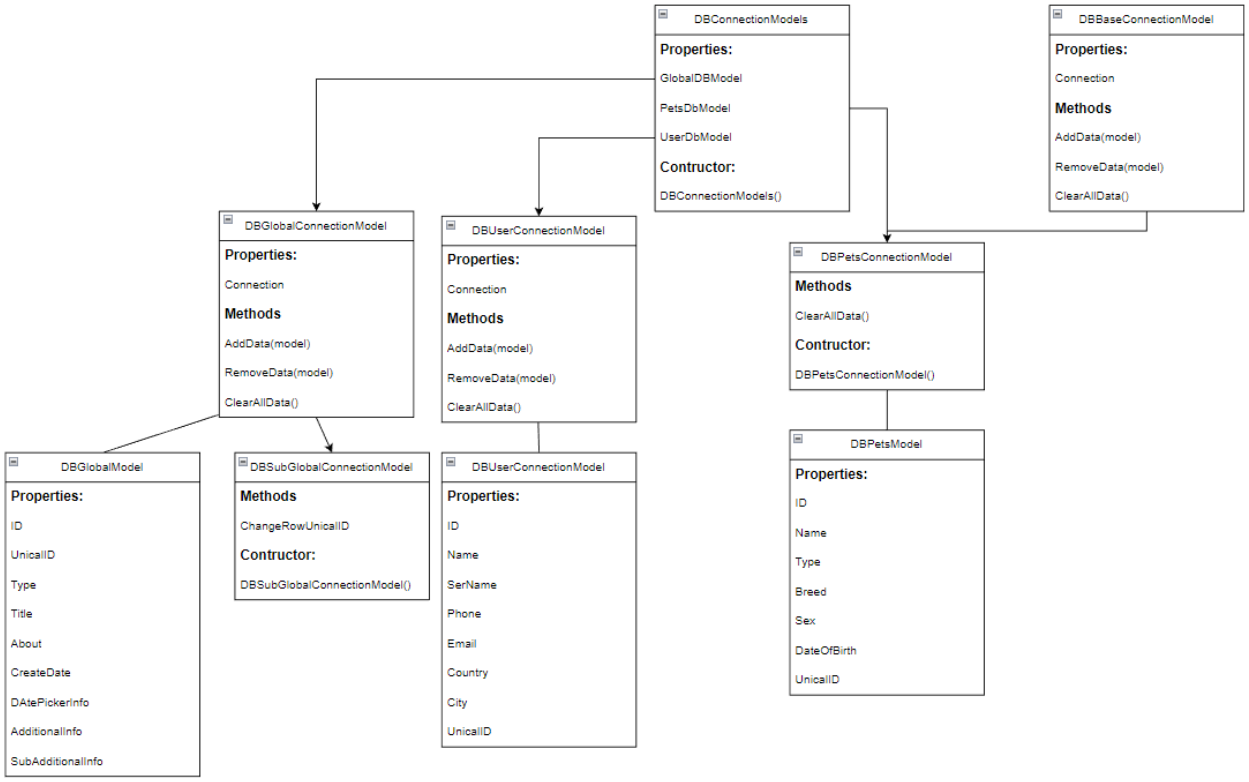


Рисунок 2.2. – Діаграма класів

Діаграма прецедентів – інструмент для візуальної фіксації вимог до системи. Ключовим поняття у цьому пункті є актор, варіанти використання та предмети. Користувачі та будь-які інші системи, які можуть взаємодіяти з об'єктом, представлені як Актори (Actors). Варіант використання - це специфікація поведінки акторів [12].

У нашому випадку в нас один користувач. На діаграмі можна побачити такі позначення як 1 та 0..1 і цьому є логічне пояснення. З точки зору діаграми прецедентів кожен варіант використання повинен мати актора, у нас це користувач, який створює процес, тому з’явилось таке позначення 1. Якщо користувач може брати або не брати участь у процесі, на діаграмі це позначається кратністю 0..1 (див. рисунок – 2.3).

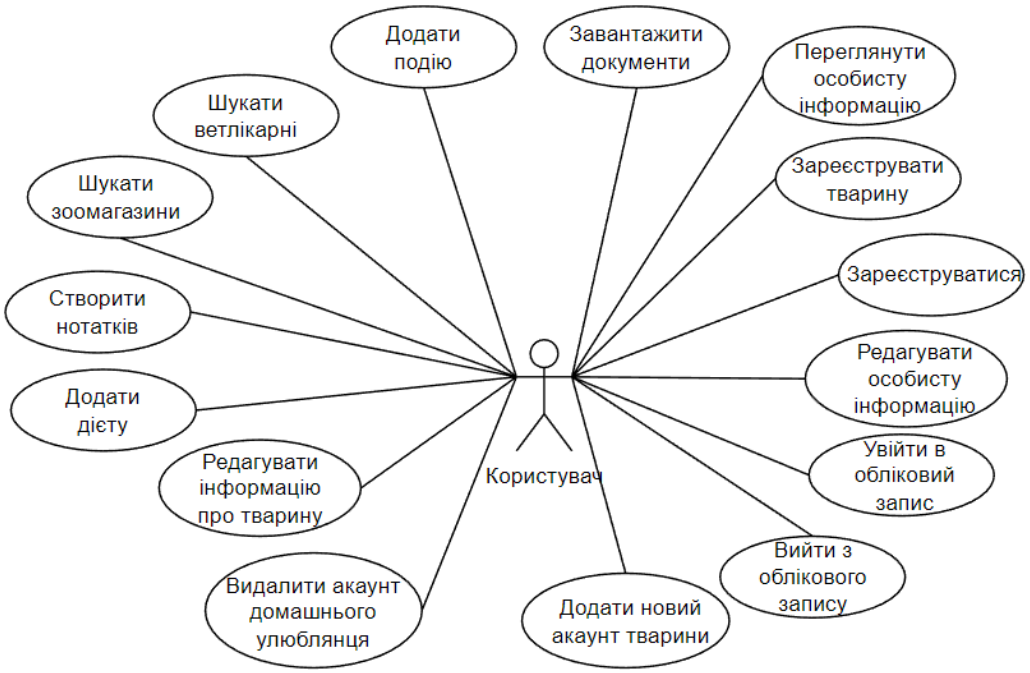


Рисунок 2.3 – Діаграма прецедентів

2.5 Розробка мапи та шаблону додатку

Для початку розробки програмного забезпечення було створено мапу додатку на онлайн-сервісі для розробки інтерфейсів Figma. Для створення зручного для користувача дизайну було проведено аналіз критерій якості дизайну, а саме:

- Швидкість та продуктивність дизайну. Користувач повинен мати можливість швидко та зручно користуватися додатком, всі елементи повинні бути розташовані зрозуміло для користувача та мати комфортні кольори для очей.
- Адаптованість до контексту.
- Відповідність функціоналу. Дизайн повинен підходити під функціонал додатку та доповнювати його зрозумілість.
- Цілісність та єдність. Дизайн повинен забезпечувати єдність усіх елементів додатку.
- Доступність. Дизайн додатку повинен бути доступним для користувачів з

обмеженими можливостями.

- Пристосованість. Дизайн повинен бути поєднан з сучасними представленнями та трендами у дизайні.

На кожному фреймі було прописано базову інформацію, приблизне розположення елементів, а також візуально показано функціонал, який буде на цій сторінці. Всього було розроблено 9 фреймворків (дви. рисунок 2.4 – 2.6).

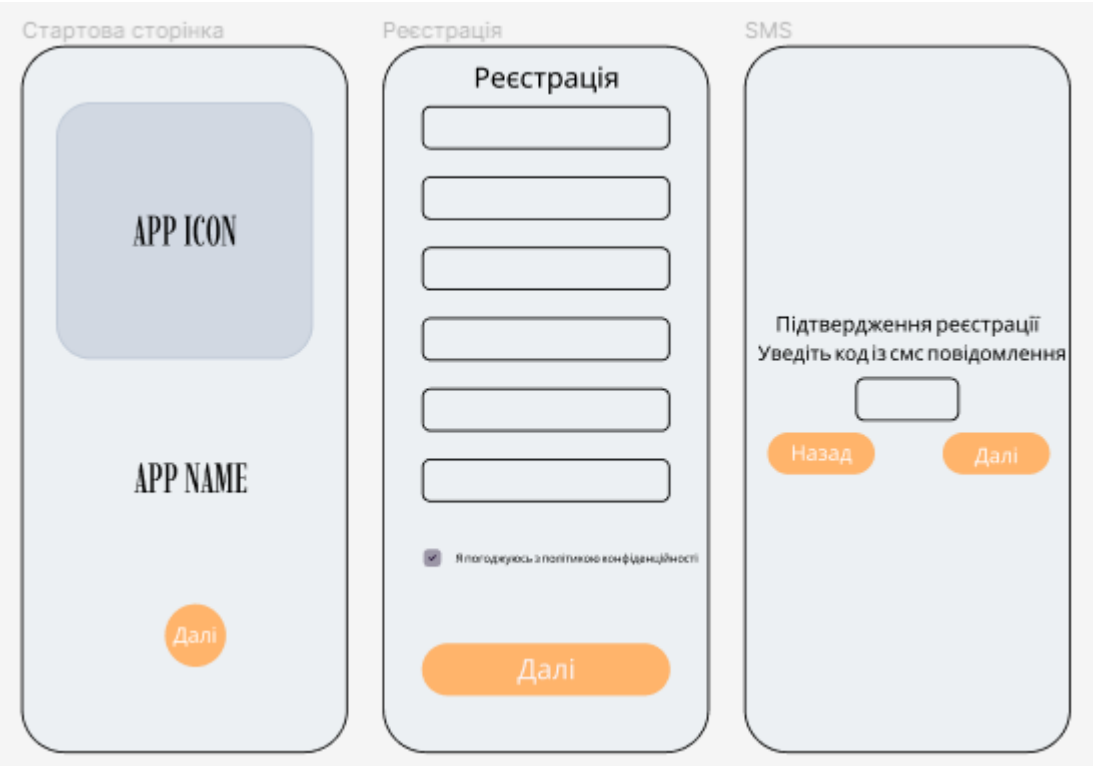


Рисунок 2.4 – Шаблони сторінок “Стартова”, “Реєстрація”, “Автентифікація”

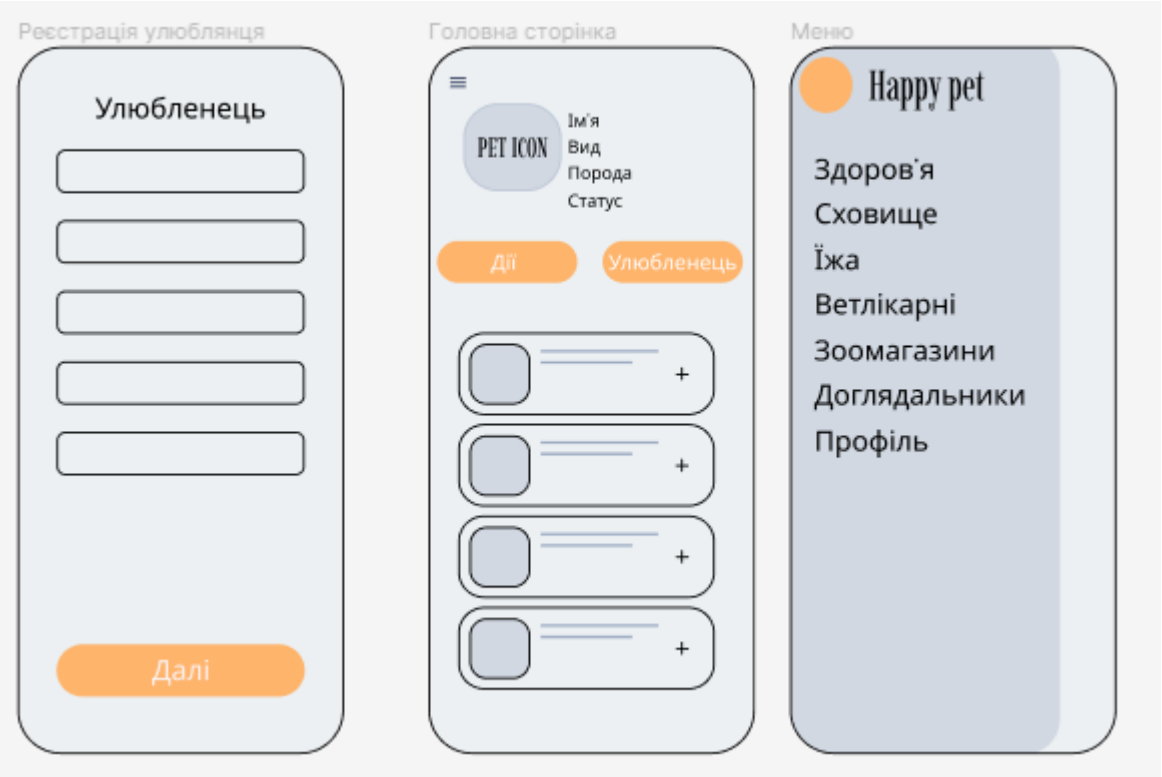


Рисунок 2.5 – Шаблони сторінок “Реєстрація улюбленця”, “Головна сторінка”, “Меню”

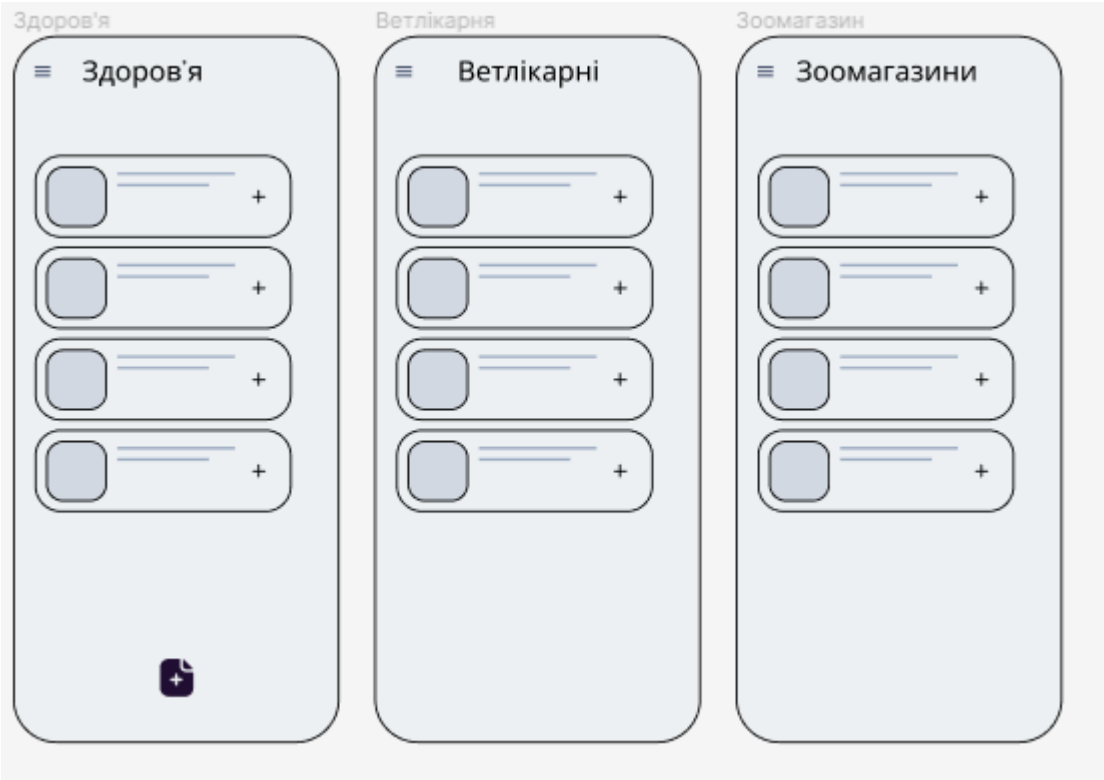


Рисунок 2.6 – Шаблони сторінок “Здоров’я”, “Ветлікарні”, “Зоомагазини”

3 ОПИС РОЗРОБКИ ДОДАТКУ

3.1 Опис сторінок додатку та їх взаємодії

Перелік сторінок додатку:

- Головна сторінка
- Сторінка профілю улюбленців
- Сторінка подій.
- Сторінка ведення здоров’я.
- Сторінка сховища.
- Сторінка контролю харчування.
- Сторінка ветеринарних лікарень.
- Сторінка зоомагазинів.
- Сторінка особистого профілю.

Головна сторінка – сторінка, яка відповідає за головну навігацію додатку, має систему бургерного меню з командами переходів між різними сторінками додатку. Має систему швидкого перегляду подій, а також має дві кнопки переходів на сторінки вибору улюбленців і додавання події.

Сторінка профілю улюбленця – сторінка, яка має функціонал оглядання та редагування особистого профілю тварини. Додатковий функціонал включає в себе можливість додавати декілька улюбленців із своїми профілями та даними.

Сторінка подій – сторінка редагування, додавання та оглядання, подій чи нотаток , які створив користувач або додаток із дозволу користувача. Додатковий функціонал – під кожну тварину є можливість створювати свої події.

Сторінка ведення здоров’я – сторінка, яка має функціонал додавання різного типу відомостей про здоров’я тварини. Ці записи можливо структурувати, оглядати, редагувати, зберігати та копіювати. Ці записи є унікальними під кожен окремий профіль тварини, і будуть доступні тільки при виборі тварини. Деякі записи є можливість комбінувати між тваринами.

Сторінка сховища – сторінка, яка має функціонал та можливості, що дозволяють зберігати файли, та інші типи інформації у додатку, із можливістю прив’язки до тварини, та перегляду тих чи інших документів із додатку.

Сторінка контролю харчування – сторінка, яка має функціонал ведення різноманітного типу дієт для тварини. Дієти можливо як створити самому, так і вибрати з переліку вже існуючих.

Сторінка ветлікарень – сторінка, яка має у собі перелік усіх ветлікарень розташованих поблизу користувача, а також має функціонал фільтрування тих чи інших їх характеристик. Таким чином є можливість обирати більш релевантні за запитом заклади поряд.

Сторінка зоомагазинів – сторінка переліку необхідних товарів для улюбленця, зібраних із більш релевантних веб-сайтів та площадок із продажу, є можливість фільтрування тих чи інших товарів та їх виробників.

Сторінка особистого профілю – сторінка, яка дає можливість перегляду та редагування особистої інформації. Має можливість зміни профілю.

3.2 Паттерн проектування

Model-View-ViewModel (MVVM) - це шаблон проєктування програмного забезпечення, структурований для розділення логіки програми та елементів керування користувацького інтерфейсу. MVVM також відомий як зв'язувач моделі та подання і був створений архітекторами Microsoft Кеном Купером і Джоном Госсманом [13].

Робота розробника Xamarin.Forms зазвичай передбачає створення користувацького інтерфейсу в XAML, а потім додавання коду програмної частини, який працює з користувацьким інтерфейсом. У міру того, як додатки змінюються і збільшуються в розмірах і масштабах, можуть виникнути складні проблеми з їх обслуговуванням. Ці проблеми включають тісний зв'язок між елементами управління призначеного для користувача інтерфейсу і бізнес-логікою, що

збільшує вартість внесення змін до призначеного для користувача інтерфейсу, а також складність модульного тестування такого коду.

Шаблон Model-View-ViewModel (MVVM) допомагає чітко відокремити бізнес-логіку та логіку представлення застосунку від його користувацького інтерфейсу (UI). Підтримка чіткого поділу між логікою застосунку та користувацьким інтерфейсом допомагає розв'язувати численні проблеми розроблення і може спростити тестування, обслуговування та розвиток застосунку. Це також може значно поліпшити можливості повторного використання коду та дає змогу розробникам і дизайнерам користувацького інтерфейсу легше співпрацювати під час розроблення відповідних частин застосунку.

У шаблоні MVVM є три основні компоненти: модель, подання та модель подання. Кожен служить певній меті (див. рисунок 3.1).

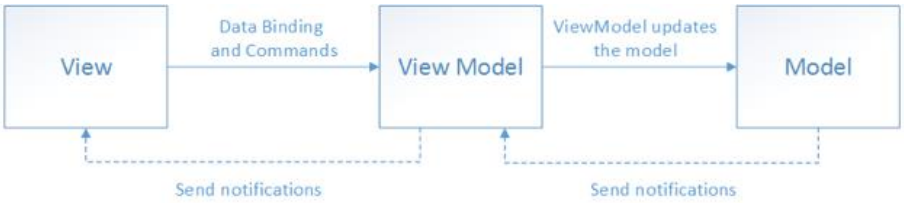


Рисунок 3.1 – MVVM структура

Крім розуміння обов'язків кожного компонента, також важливо розуміти, як вони взаємодіють один з одним. На високому рівні подання "знає" про модель подання, а модель подання "знає" про модель, але модель не знає про модель подання, а модель подання не знає про подання. Таким чином, модель подання ізолює подання від моделі та дає змогу моделі розвиватися незалежно від подання.

Переваги використання шаблону MVVM полягають у такому:

- Якщо існує реалізація моделі, яка інкапсулює наявну бізнес-логіку, змінити її може бути складно або ризиковано. У цьому сценарії модель представлення виступає як адаптер для класів моделі та дозволяє уникнути внесення серйозних змін до коду моделі.

- Розробники можуть створювати модульні тести для моделі подання і моделі без використання подання. Модульні тести для моделі подання можуть виконувати ті самі функції, що й подання.
- Користувацький інтерфейс додатка можна змінити, не зачіпаючи код, за умови, що подання повністю реалізовано на XAML. Тому нова версія подання має працювати з наявною моделлю подання.
- Дизайнери та розробники можуть працювати незалежно та одночасно над своїми компонентами в процесі розробки. Дизайнери можуть зосередитися на поданні, а розробники можуть працювати над моделлю подання та компонентами моделі.
- Ключ до ефективного використання MVVM полягає в розумінні того, як розкласти код додатка на правильні класи, і в розумінні того, як класи взаємодіють.

3.3 Підключення Бази Даних SQLite

Було розроблено клас базової моделі (DBConnectionModels), котрий відповідає за моделі підключення до всіх баз та таблиць. У ньому було створено моделі підключення кожної з 3-х баз, котрі використовуються у додатку (див. рисунок 3.2).

```
namespace HappyPet.DataBase
{
    /// <summary>
    /// Data base connection model class.
    /// </summary>
    public class DBConnectionModels : IConnectionModels
    {
        /// <summary>
        /// Global data base model.
        /// </summary>
        public IConnectionModel GlobalDBModel { get; }

        /// <summary>
        /// Pets data base model.
        /// </summary>
        public IConnectionModel PetsDbModel { get; }

        /// <summary>
        /// User data base model.
        /// </summary>
        public IConnectionModel UserDbModel { get; }

        /// <summary>
        /// Initialize <see cref="DBConnectionModels"/>.
        /// </summary>
        public DBConnectionModels()
        {
            GlobalDBModel = new DBGlobalConnectionModel();
            PetsDbModel = new DBPetsConnectionModel();
            UserDbModel = new DBUserConnectionModel();
        }
    }
}
```

Рисунок 3.2 – Клас DBConnectionModels

Перша – це модель підключення до глобальної бази даних, котра містить глобальні записи. Нижче зображено, що ця модель має публічний параметр connection завдяки котрому проходить підключення до нашої бази даних, викликавши у конструкторі цього класу метод GetConnection. Всі моделі мають ідентичний набір методів та публічний параметр, але відрізняються за рахунок різних шляхів до файлу бази даних та класу, котрий представляє дані для тієї чи іншої бази або таблиці (див. рисунок 3.3 – 3.4).

```
using SQLite;
using System;
using Xamarin.Forms;
using HappyPet.DataBase.Models;
using HappyPet.Interface;

namespace HappyPet.DataBase
{
    /// <summary>
    /// Data base global connection model.
    /// </summary>
    public class DBGlobalConnectionModel : IConnectionModel
    {
        /// <summary>
        /// SQLite connection.
        /// </summary>
        public SQLiteConnection Connection { get; }

        /// <summary>
        /// Initialize <see cref="DBGlobalConnectionModel"/>
        /// </summary>
        public DBGlobalConnectionModel()
        {
            Connection = DependencyService.Get<ISqLiteConnection>().GetConnection(DataBaseConstants.DataBaseGlobalPath);
            Connection.CreateTable<DBGlobalModel>();
        }

        /// <summary>
        /// Add data to DB.
        /// </summary>
        /// <param name="model">Data Model.</param>
        public void AddData(IDBBaseModel model)
        {
            try
            {
                Connection.Insert(model);
            }
            catch
            {
                throw new Exception();
            }
        }
```

Рисунок 3.3 – Клас DBClobalConnectionModule

```
        /// <summary>
        /// Remove data from DB.
        /// </summary>
        /// <param name="model">Data Model.</param>
        public void RemoveData(IDBBaseModel model)
        {
            try
            {
                Connection.Delete(model);
            }
            catch
            {
                throw new Exception();
            }
        }

        /// <summary>
        /// Clear all data in DB.
        /// </summary>
        /// <param name="model">Data Model.</param>
        public void ClearAllData()
        {
            try
            {
                Connection.DeleteAll<DBGlobalModel>();
            }
            catch
            {
                throw new Exception();
            }
        }
    }
}
```

Рисунок 3.4 – Базові методи

Друга та третя – це моделі підключення для баз даних улюбленців та користувача.

Усі шляхи до файлів бази даних лежать у класі DataBaseConstants. У ньому можна побачити поля, які відповідають за назви файлів баз даних, та публічні властивості, котрі повертають повний шлях до необхідного файлу бази. Використання одного з цих полів можна побачити нижче на рисунках (див. рисунок 3.5 – 3.6).

```
namespace HappyPet.DataBase
{
    /// <summary>
    /// Data base constants
    /// </summary>
    public static class DataBaseConstants
    {
        /// <summary>
        /// Data base user file name.
        /// </summary>
        public const string DataBaseUserFilename = "UserSQLite.db3";

        /// <summary>
        /// Data base pets file name.
        /// </summary>
        public const string DataBasePetsFilename = "PetsSQLite.db3";

        /// <summary>
        /// Data base global file name.
        /// </summary>
        public const string DataBaseGlobalFilename = "GlobalSQLite.db3";

        /// <summary>
        /// Flags to data base controls.
        /// </summary>
        public const SQLite.SQLiteOpenFlags Flags =
            // open the database in read/write mode
            SQLite.SQLiteOpenFlags.ReadWrite |
            // create the database if it doesn't exist
            SQLite.SQLiteOpenFlags.Create |
            // enable multi-threaded database access
            SQLite.SQLiteOpenFlags.SharedCache;
```

Рисунок 3.5 – Клас DataBaseConstants

```
        /// <summary>
        /// Data base user path property.
        /// </summary>
        public static string DataBaseUserPath
        {
            get
            {
                var basePath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
                return Path.Combine(basePath, DataBaseUserFilename);
            }
        }

        /// <summary>
        /// Data base pets path property.
        /// </summary>
        public static string DataBasePetsPath
        {
            get
            {
                var basePath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
                return Path.Combine(basePath, DataBasePetsFilename);
            }
        }

        /// <summary>
        /// Data base global path property.
        /// </summary>
        public static string DataBaseGlobalPath
        {
            get
            {
                var basePath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
                return Path.Combine(basePath, DataBaseGlobalFilename);
            }
        }
    }
}
```

Рисунок 3.6 – Публічні властивості класу DataBaseConstants

Створення базово класу моделей виконується у класі App, котрий відповідає за виклик усього додатку (див. рисунок 3.7). У ньому також передається базова модель до головної сторінки, або стартової сторінки, якщо додаток був запущений вперше.

```
/// <summary>
/// Initialize Application on Xamarin Forms side.
/// </summary>
public App()
{
    var DBConnectionModels = new DBConnectionModels();
    var navigationPage = new CustomNavigationPage(new StartPage(DBConnectionModels));
    object startTrigger;
    if (Current.Properties.TryGetValue("startPageTrigger", out startTrigger))
    {

        if ((bool)startTrigger)
        {
            MainPage = new FlyoutPageMenu(DBConnectionModels);
        }
        else
        {
            MainPage = navigationPage;
        }
    }
    else
    {
        _properties = new CommonAppProperties();
        _properties.RegisterProperties();
        MainPage = navigationPage;
    }
    InitializeComponent();
}
```

Рисунок 3.7 – Головний клас додатку App

Також кожна з моделей підключення має модель представлення даних, котрі мають свої параметри та перелік даних, що будуть записані у таблицю. Наприклад клас представлення даних глобальної бази має такі поля: ID, UnicalID, Type, Title, About, CreateDate, DatePickerInfo, AdditionalInfo, SubAdditionalInfo. Завдяки таким полям як Type та CreateDate, присутня можливість фільтрувати, та за необхідністю вибирати тільки ті записи з бази, котрі нам необхідні. Зараз у додатку передбачено тільки 4 типи даних, що можуть зберігатися у цій таблиці: Hospital – тип котрий відповідає за те, що цей запис належить до модуля додатку із ветлікарнями. ZooShop – по аналогії з минулим, відповідає за запис котрий належить до модуля із зоомагазинами. HealthInfo – надається тільки тим записам, котрі належать до модуля контролю здоров’я. Default – надається записам, що не належать до інших категорій (див. рисунок 3.8 – 3.9).

```
using HappyPet.Interface;
using SQLite;
using System;

namespace HappyPet.DataBase.Models
{
    /// <summary>
    /// Data base global model.
    /// </summary>
    public class DBGlobalModel : IDBBaseModel
    {
        [PrimaryKey, AutoIncrement]
        public int ID { get; set; }
        public string UnicalID { get; set; }
        public DBGlobalTypesOfData Type { get; set; }
        public string Title { get; set; }
        public string About { get; set; }
        public DateTime CreateDate { get; set; }
        public DateTime DatePickerInfo { get; set; }
        public string AdditionalInfo { get; set; }
        public string SubAdditionalInfo { get; set; }
    }
}
```

Рисунок 3.8 – Клас уявлення DBGlobalModel

```
namespace HappyPet.DataBase.Models
{
    /// <summary>
    /// Data base global types of data.
    /// </summary>
    public enum DBGlobalTypesOfData
    {
        Hospital = 0,
        ZooShop = 1,
        HealthInfo = 2,
        Default = -999
    }
}
```

Рисунок 3.9 – Перелічуваний тип даних DBGlobalTypesOfData

Вперше використання моделі бази даних виконується у класі RegisterPage задля створення перших даних користувача. Наступний запис створюється в класі PetRegisterPage та додає відомості про тварину. Далі ці записи зберігаються та використовуються насамперед у особистих профілях тварини та користувача, де є можливість переглядати та редагувати цю інформацію.

3.4 Опис головної сторінки

Головна сторінка – це сторінка, на якій користувач так чи інакше буде проводити більше всього часу. Із розумінням цього було розроблено найбільш гнучку версію головної сторінки за принципом «нічого зайвого». Головна сторінка має такі можливості:

Головне меню. Бургерне меню, задля найбільш зручнішої системи навігації. Завдяки цьому користувач має змогу переміститися куди завгодно всього лише за декілька секунд і так само повернутися назад. Сторінки додатку існують тільки на момент, поки користувач знаходиться на якійсь з них. Ця система дозволяє зробити використання максимально швидким і не завантажувати процеси пристрою. З першого погляду може здатися, що навпаки при такій системи, створення кожної сторінки може тільки погіршити швидкість. Але така позиція була б актуальна у більшості систем розробки окрім Xamarin. Розробка і всі процеси всередині додатку, написаному на цій системі, завжди виконуються із використанням асинхронного та багатопотокового програмування, що дозволяє використовувати усі процеси із максимальною гнучкістю. Також дуже важливу роль грає той факт, що використання MVVM паттерну, котрий у об'єднанні із Xamarin дозволяє завжди зберігати UI елементи окремо від бізнес–логіки, та використовувати логіку тільки при необхідності, та одразу ж при неактивності тієї чи іншої зони заморожувати її до наступного використання.

Кнопки навігації. Кнопки навігації до сторінки вибору улюбленця або сторінки додавання нагадувань. "Якщо маєш змогу зробити щось максимально просто – зроби це" – це непрямий переклад принципу програмування KISS, який було використано для реалізації. Такий принцип був використаний із навігацією та цими двома сторінками. Сторінки, котрі мають безпосередній вплив на додаток, знаходяться у найзручнішому місці.

Скролова частина з нагадуваннями. Не можливо було не додати це сюди, бо саме головна сторінка кожен раз буде з'являтися перед користувачем і він повинен мати змогу одразу переглядати найважливіші події, щоб не потрібно було переходити із цієї сторінки кудись далі.

Інформація про улюбленця. Концепція додатку розроблена по системі один улюбленець – один додаток але нічого не заважало розробити образи одного і того ж додатку на кожну тварину. І це було зроблено, тепер не буде проблемою пошук записів про того чи іншого улюбленця, бо кожен має все своє окремо. А користувач має контроль над усіма тваринами. Власник може мати більше ніж одного улюбленця, та може мати змогу зробити профіль кожному з них у додатку, та переключатися між ними у будь який час, усі сторінки додатку будуть зав'язані на улюбленця, тобто кожного разу коли користувач обирає іншу тварину, усі сторінки та модулі оновлюють дані таким чином, щоб інформація, яка міститься на них, була тільки про цього улюбленця. Цей функціонал дає змогу не втратити чогось важливого про кожну окрему тварину у великому переліку усієї інформації про інших . На головній сторінці завжди буде інформація про того улюбленця, котрий зараз обраний, буде його особиста фотографія або стікер, коротка інформація та статус.

Статус улюбленця. Функціонал, котрий був введений задля підвищення емоційного настрою власників та змогу дійсно дізнатися, чи зможе додаток розуміти кожного хатнього улюбленця. Наприклад, статус песика у вечірній час може бути – бешкетує. Присвоєння статусу спочатку буде здійснюватися випадково але з часом, коли додаток буде отримувати більше інформації, алгоритми будуть направленні на те, щоб після аналізу отриманих даних присвоювати статус спираючись на ту інформацію, котру мають.

3.5 Опис системи реєстрації

Стартова сторінка. Стартова сторінка додатку з'явиться тільки якщо додаток був вперше запущений на пристрої. Відокремлюючи частину про перший запуск додатку зі сторони UI\UX ефективності, ця сторінка несе у собі важливу логічну частину, при створенні цієї сторінки вперше, разом із нею створюються перші необхідні образи даних та параметрів, які буде використовувати додаток у майбутньому. Якщо додаток у подальшому буде видалено та переустановлено,

саме ця сторінка знов створить усі образи, та зможе відновити всі раніше збережені дані про користувача, у тому випадку якщо реєстрація була успішна, та з моменту видалення додатку не пройшло більш ніж 3 місяці.

Реєстрація. Після привітання додаток запропонує користувачу пройти швидку реєстрацію, де потрібно буде ввести такі дані: Ім'я*, Прізвище, Телефон*, Електронну Адресу*, Країну* та Місто. Після цього буде потрібно підтвердити, що користувач ознайомився із політикою конфіденційності і надає згоду на обробку і використання персональних даних і також надає доступ додатку до використання геолокації пристрою. Після заповнення цієї інформації, користувача переведе на сторінку активації профілю , на котрій буде потрібно ввести 7 цифр, отриманих в смс повідомленні .

Наступний крок – це додавання улюбленця до додатку. Перед тим як це буде запропоновано користувачу, додаток надасть інформацію про функціонал, котрий не може бути активовано без додавання тварини. На цьому етапі потрібно буде заповнити такі дані як: Ім'я улюбленця*, Вид тварини*, Породу*, Стать* та Дату народження*. Після цього користувача буде переведено до головного меню, та він отримує повний доступ до функціоналу.

3.6 Опис системи контролю здоров'я

Записи здоров'я. Основна частина додатку зав'язана на функціоналі контролю здоров'я домашнього улюбленця. Для досягнення цієї мети, було розроблено спеціальний комплекс, котрий має максимально осягнути усі найважливіші аспекти контролю за здоров'ям. Перше чому була приділена увага, це ведення особистого щоденника тварини, у який користувач має змогу вносити свої відомості про улюбленця. При створенні запису, надається можливість детально описати усю необхідну інформацію, починаючи із типу запису, чи то похід до лікаря чи прийом ліків, або ж якісь особисті зауваження чи події. Записи про тварину зберігаються в окремій базі даних, та на основі побажань користувача, із допомогою цієї інформації додаток може рекомендувати ту чи іншу інформацію

власнику тварини. Це може бути як рекомендація записати на прийом до ветлікарні, чи змінити корм або ж рекомендація приділяти більше уваги своєму улюбленцю у ті чи інші моменти часу. Головна мета цієї системи – налаштувати додаток максимально гнучко під потреби користувача та його улюбленців. У фінальному вигляді, це повинно стати системою персонального помічника по догляду за твариною, який бути знати усе необхідне та зможе миттєво надавати потрібну інформацію. Задля досягнення цього функціоналу розроблені алгоритми розумної фільтрації та пошуку у середовищі записів. Наступною частиною може стати інтеграція штучного інтелекту як допоміжного функціоналу у пристосуванні додатку до користувача та його тварин.

Система нагадування. Розроблена система нагадування, котра буде орієнтуватися за допомогою записів про здоров'я тварини. Вона зможе нагадати коли надійшов час відвідати лікаря, провести ревакцинацію, зробити улюбленцю різноманітні процедури, чи просто необхідно приділити більше уваги. Спочатку система буде нагадувати лише ті записи, котрі користувач сам обере при створенні записів здоров'я. Але з часом, додаток буде налаштовувати свої алгоритми під потреби користувача, та нагадувати йому рекомендації щодо тварини. Функціонал розумного нагадування буде активний за замовчуванням, але користувач в будь-який час матиме змогу його відключити.

Ведення документації. Разом із системою ведення записів та нагадування, було розроблене сховище, у якому користувач матиме змогу зберігати усі необхідні документи, фото чи іншу інформацію задля того, щоб мати змогу швидко використати їх за необхідністю. Як по аналогії із додатком Дія, розробленим для людей зі збереженням усіє найважливішої електронний документації про особу, буде можливість зберігати електронну документацію про свого улюбленця.

3.7 Опис додаткового функціоналу

Додавання доглядача до профілю, із можливістю поширення інформації про тварину. Кожному із власників тварин, колись було необхідно залишити тварину

на деякий час і зазвичай при цьому власник не має змоги передати усю необхідну інформацію про улюбленця та хвилюється за нього , за його емоційний стан і здоров'я. Завдяки наявності цього функціоналу можна додати цій події трохи більше відповідальності. Власник не буде хвилюватися, що той кому він залишив свого улюбленця щось забуде , в свою чергу та людина яка залишилася з твариною буде мати змогу більше розуміти про тварину, ті чи інші її особливості, та процес адаптації пройде набагато легше

Сторінка зоомагазинів та ветеринарних закладів. Ці сторінки були розроблені за для спрощення пошуків необхідних товарів та ветлікарень, котрі необхідні при різних випадках. Ця ідея виникла із того, що не кожна ветлікарня може приймати незвичайних до повсякдення тварин, наприклад павуків, змій, та навіть папуг. Задля цього у базу даних додано дуже велика кількість різного типу ветлікарень, котрі можуть дати необхідну допомогу самим різним тваринам. Частина із товарами для улюбленців, розроблена за принципом найнеобхідніше найближче, якщо користувач дасть дозвіл на використання геолокації пристрою, алгоритм буде підбирати товари та заклади якнайближче до користувача, а також надасть змогу переглядати веб-сайти магазинів, та обирати товари там.

Алгоритми фільтрації налаштовані завдяки геолокації та актуальності товарів. Якщо користувач не надав дозвіл на використання геолокації, алгоритми будуть використовувати тільки актуальність того чи іншого товару, а також їх доступність у інтернет–магазинах. Користувач буде мати змогу сам налаштовувати фільтри для цих двох сторінок.

3.8 Принцип розробки SOLID

SOLID – набір п’яти базових принципів об’єктно-орієнтованого програмування, деякі з них були використані у додатку [14].

Перший – це принцип єдиного обов’язку, це означає, що клас має виконувати тільки один обов’язок. Тобто виконувати тільки той функціонал, для котрого він був задуманий. Цей принцип добре можна побачити по моделям управління базами

даних у додатку, в котрих немає нічого крім підключення до бази, та методів управління записами, детальніше, додавання записів, видалення, та повна очистка бази. Також ми можемо бачити цей принцип у класі CommonAppProperties, котрий створений для того щоб реєструвати, читати та змінювати постійні властивості котрі ми можемо зберігати у внутрішньому сховищі. Приклад до цього принципу наведено завдяки класу CustomNavigationPage, котрий спадкується від базового класу, додає необхідну логічну структуру та використовується замість базового класу. Можемо бачити що цей клас не виконує ніяких сторонніх дій (див. рисунок 3.10).

```
using Xamarin.Forms;

namespace HappyPet.Model.Common
{
    /// <summary>
    /// Custom navigation page.
    /// </summary>
    public class CustomNavigationPage : NavigationPage
    {
        /// <summary>
        /// Initializer <see cref="CustomNavigationPage"/>
        /// </summary>
        /// <param name="page">Page object</param>
        public CustomNavigationPage(Page page) : base(page)
        {
            ActiveStartPage = page;
        }

        /// <summary>
        /// Active start page.
        /// </summary>
        public Page ActiveStartPage { get; set; }
    }
}
```

Рисунок 3.10 – Клас CustomNavigationPage

Наступний принцип, котрий є можливість добре бачити у реалізаціях додатку це принцип розділення інтерфейсів. Він допомагає обходити ситуації, коли похідні класи логічно не повинні успадковувати якісь методи. Цей принцип ми бачимо у таких інтерфейсах як ISqLiteConnection, IConnectionModel, IDBBaseModel. Якщо інтерфейси будуть мати дуже багато чого для наслідування, це може призвести до

обставин коли логіка класів може не відповідати дійсності. Наприклад у об’єктів, що не повинні мати функціонал «рухатися» він буде вже закладений (див. рисунок 3.11).

```
using SQLite;

namespace HappyPet.Interface
{
    /// <summary>
    /// Sql lite connection interface.
    /// </summary>
    public interface ISqLiteConnection
    {
        /// <summary>
        /// Get connection method.
        /// </summary>
        /// <param name="path">File path.</param>
        /// <returns>SQL connection.</returns>
        SQLiteConnection GetConnection(string path);
    }
}
```

Рисунок 3.11 – Інтерфейс IsqLiteConnection

Ще один із принципів, котрий був використаний це – принцип відкритості\закритості. Класи, котрі використовуються у додатку, мають бути відкритим до розширення, але закритими до змін. Тобто розробка класів повинна мати змогу додавання різного функціоналу методами наслідування чи проекції класів, але ніяк не повинно бути ситуацій, котрі змінюють базові та основні класи. Для кожного нового функціоналу повинно бути запроваджено новий клас, який може базуватися на тому, котрий вже має деякий необхідний функціонал. Цей принцип бачимо у моделях з’єднання до бази даних, зараз вони вже мають усю базову реалізацію, але якщо буде потреба розширювати функціонал, ми можемо спадкуватися від цих класів та робити вже більш функціональну реалізацію різних методів. Наприклад задля роботи із фільтраціями та конкретними записами у базі. Був розроблений клас DBSubGlobalConnectionModel, у якому є вся базова реалізація і доданий метод із додатковою логікою (див. рисунок 3.12).

```
using HappyPet.DataBase.Models;
using HappyPet.Interface;
using System.Linq;

namespace HappyPet.DataBase
{
    /// <summary>
    /// Data base user connection model.
    /// </summary>
    public class DBSubGlobalConnectionModel : DBGlobalConnectionModel
    {
        /// <summary>
        /// Initialize <see cref="DBSubGlobalConnectionModel"/>
        /// </summary>
        public DBSubGlobalConnectionModel():base()
        {
        }

        /// <summary>
        /// Change row Unical Id.
        /// </summary>
        /// <param name="model"></param>
        public void ChangeRowUnicalID(IDBBaseModel model)
        {
            var rows =  (from x in Connection.Table<DBGlobalModel>() select x).Where(p => p.Type == DBGlobalTypesOfData.HealthInfo).ToList();
            rows.First().UnicalID = model.UnicalID;
        }
    }
}
```

Рисунок 3.12 – Клас DBSubGlobalConnectionModel

3.9 Принцип розробки DIP

DIP – останній принцип із SOLID, котрому приділено трохи більше уваги, бо він один із найважливіших принципів проектування архітектури коду [14].

Перше чого потрібно дотримуватися при взаємодії із цим принципом – це те, що код повинен бути написаний у порядку залежності даних для користувача від бізнес логіки і не навпаки.

Друге правило – це використання непрямих залежностей. Більшу їх частину або, по можливості, навіть усі, потрібно абстрагувати за допомогою інтерфейсів, щоб бізнес–логіка та основні частини працювали з усім, щоб ви їм не надали, вимагаючи для цього лише якийсь образ просто «мосту», завдяки котрому логіка зможе взаємодіяти із тим функціоналом та об’єктами, котрий їй буде потрібен.

Одним із основних класів, котрий використовується у цьому принципу це базовий клас DependencyResolver, який має метод SetResolver(object), що приймає об’єкт, який повинен мати реалізацію використання зіставлення сутності з її представленням. Цей принцип можемо бачити у конструкторі моделей з’єднання із базою. Завдяки DependencyService ми викликаємо метод Get і отримаємо реалізацію сутності ISqlLiteConnection (див. рисунок 3.13).

```
/// <summary>
/// Initialize <see cref="DBUserConnectionModel"/>
/// </summary>
public DBUserConnectionModel()
{
    Connection = DependencyService.Get<ISqLiteConnection>().GetConnection(DataBaseConstants.DataBaseUserPath);
    Connection.CreateTable<DBUserModel>();
}
```

Рисунок 3.13 – Конструктор класу DBUserConnectionModule

3.10 Принцип розробки DRY

DRY – це принцип запобігання копіювання одних і тих самих фрагментів коду [15]. Задля того, щоб дотримуватись цього принципу, використовуються абстракції та базові реалізації. Тобто, якщо ви маєте функції або набір функцій, котрі можуть використовуватися у багатьох місцях вашої програми, ви повинні помістити їх або у абстрактний клас або у базовий, чи запровадити систему створення спеціального класу тільки із необхідним функціоналом, щоб кожен раз не переписувати один і той самий код. Прикладом цього є дуже поширені функції конвертації простих типів даних, таких як: конвертування типів double, int, float у string чи навпаки, конвертування різних enum(перечислень) у string. Ця практика одна із найпоширеніших. Із цим принципом був розроблений клас DBBaseConnectionModel, котрий має усю базову реалізацію з’єднання, та спадкові класи можуть використовувати це (див. рисунок 3.14).

```
using HappyPet.Interface;
using SQLite;
using System;

namespace HappyPet.DataBase
{
    /// <summary>
    /// Data base, base connection model.
    /// </summary>
    public class DBBaseConnectionModel : IConnectionModel
    {
        /// <summary>
        /// SQLite connection.
        /// </summary>
        public SQLiteConnection Connection { get; private set; }

        /// <summary>
        /// Initialize <see cref="DBBaseConnectionModel"/>
        /// </summary>
        public DBBaseConnectionModel(SQLiteConnection connection)
        {
            Connection = connection;
        }

        /// <summary>
        /// Add data to DB.
        /// </summary>
        /// <param name="model">Data Model.</param>
        public virtual void AddData(IDBBaseModel model)
        {
            try
            {
                Connection.Insert(model);
            }
            catch
            {
                throw new Exception();
            }
        }
```

Рисунок 3.14 – Клас DBBaseConnectionModel

3.11 Принцип розробки SLAP

Принципи SLAP – єдина абстрактність функції або об’єктів та класів. Єдина абстрактність – це єдина зона відповідальності [16]. Функція не повинна виконувати одразу більше чотирьох–п’яти дій. Якщо ваша функція займає дуже багато строк, треба замислитися над реструктуризацією та заміною однієї функції на декілька. У цьому випадку є можливість використовувати минули принципи, та перенести деякі дії котрі повторюються у абстрактні та базові класі вашої програми. Цей принцип ми можемо бачити на прикладі методів у моделях роботи із базами даних та методах обробки подій (див. рисунок 3.15 – 3.16).

```
/// <summary>
/// Add data to DB.
/// </summary>
/// <param name="model">Data Model.</param>
public virtual void AddData(IDBBaseModel model)
{
    try
    {
        Connection.Insert(model);
    }
    catch
    {
        throw new Exception();
    }
}
```

Рисунок 3.15 – Метод AddData

```
/// <summary>
/// Pets button clicked.
/// </summary>
/// <param name="sender"> Object sender. </param>
/// <param name="args"> Event args. </param>
private async void PetsButtonClicked(object sender, EventArgs args)
{
    await Navigation.PushAsync(new PetCarouselMain(_connectionModels));
}
```

Рисунок 3.16 – Метод PetsButtonClicked

3.12 Тестування проєкту

Додаток описаний модульними тестами на 70%. Модульні тести додатку можно побачити нижче (див. рисунок 3.17)

AppInitializeUnitTests.cs	21.05.2023 18:38	C# Source File	1 КБ
DBCollectionUnitTests.cs	21.05.2023 18:38	C# Source File	1 КБ
DBModelsUnitTests.cs	21.05.2023 18:38	C# Source File	1 КБ
DocumentsDBModelUnitTests.cs	21.05.2023 18:38	C# Source File	1 КБ
DocumentsModelUnitTests.cs	21.05.2023 18:38	C# Source File	1 КБ
HealthModelUnitTests.cs	21.05.2023 18:38	C# Source File	1 КБ
MainPageModelUnitTests.cs	21.05.2023 18:38	C# Source File	1 КБ
NavigationServiceUnitTests.cs	21.05.2023 18:38	C# Source File	1 КБ
PetCarouselUnitTests.cs	21.05.2023 18:38	C# Source File	1 КБ
RegistrationModelUnitTests.cs	21.05.2023 18:39	C# Source File	1 КБ
ViewDataUnitTests.cs	21.05.2023 18:39	C# Source File	1 КБ

Рисунок 3.17 – Розробка тестів

Модульні тести – це такий тип тестів, який пишеться самими розробниками для перевірки окремих модулів та компонентів програмного коду таких як функції, методи та класи. Ці тести ізольовані та виконуються автоматично.

Код додатку покривався модульними тестами для того, щоб забезпечити стабільність роботи компонентів програми, також щоб виявити помилки. Виявлення помилок завдяки модульним теастами досі популярний метод, бо так можна знайти якесь неправильне обчислення, логіку, або іншого. Після кожної зміни коди треба запускати тести, щоб бути впевненим, що все працює так як і повинно.

Отже, модульні тести корисні тим, що вони надають змогу виявляти помилки, підтримувати рефакторинг кода, забезпечувати стабільність роботи компонентів програми для запобігання непередбачуваних помилок, щоб гарантувати надійність та безпечність продукту.

ВИСНОВКИ

У результаті виконання дипломної роботи розроблено мобільний додаток для контролю здоров’я домашніх тварин. Її актуальність полягає в тому, що наразі процес контролю здоров’я домашніх тварин досі важка справа для господарів.

1. Проаналізовано предметну область дипломної роботи та встановлено основні недоліки і переваги існуючих рішень щодо контролю здоров’я домашніх тварин.

2. Розроблено функціональні та нефункціональні вимоги до додатку та встановлено необхідний функціонал.

3. Дослідивши існуючі інструменти розробки програмного забезпечення, обрано такі засоби реалізації: мову програмування C#, платформу Xamarin, базу даних SQLite, середовище розробки Visual Studio, систему версій git та паттерн проєктування MVVM.

4. Розроблено додаток для контролю здоров’я домашніх тварин. Встановлено, що платформа Xamarin –зручний та гнучкий інструмент із кросплатформної розробки мобільних додатків. Для керування всім цим використовується мова C#, яка має чітку та широку документацію. Завдяки цьому є можливість реалізовувати різні патерни, системи, бази даних і додаткові інструменти, такі як Syncfusion або Lottie. Крім того, мова C# надає широкий спектр можливостей для роботи з JSON і зручне використання SQLite бази даних. Найголовніше, ця логіка розроблена на рівні, який дозволяє використовувати її як на системі Android, так і на iOS. Це дозволить зробити додаток доступним для всіх користувачів без необхідності переробки, а лише додати кілька класів, що відповідатимуть вимогам Apple і адаптуються під UI/UX структури.

5. Проведено тестування додатку та перевірено виконання усіх функціональних та нефункціональних вимог.

6. Робота пройшла апробацію на Науково-технічних конференціях «Застосування програмного забезпечення в інфокомунікаційних технологіях», м.

Київ, ДУТ, 20 квітня 2023 року та «Сучасні інтелектуальні інформаційні технології в науці та освіті», м. Київ, ДУТ, 16 травня 2023 року.

СПИСОК ЛІТЕРАТУРИ

1. Animal ID [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://animal-id.net/ua/ Дата звернення: 2023
2. VitusVet Pet Medical Records [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://vitusvet.com/pet-owners/ Дата звернення: 2023
3. PitPat [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://www.pitpat.com/ Дата звернення: 2023
4. A tour of the C# language [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/dotnet/csharp/tour-of-csharp/ Дата звернення: 2023
5. What is Xamarin? [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/xamarin/get-started/what-is-xamarin Дата звернення: 2023
6. XAML overview [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/dotnet/desktop/wpf/xaml/?view=netdesktop-7.0 Дата звернення: 2023
7. SQLite [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://sqlite.org/index.html Дата звернення: 2023
8. What is Visual Studio? [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/visualstudio/get-started/visual-studio-ide?view=vs-2022 Дата звернення: 2023
9. Git [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://git-scm.com/ Дата звернення: 2023
10. ISO/IEC TS 27008:2019(en) Information technology – Security techniques – Guidelines for the assessment of information security controls. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://www.iso.org/obp/ui/#iso:std:iso-iec:ts:27008:ed-1:v1:en

11. Закон України “Про інформацію” [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2657-12#Text Дата звернення: 2023
12. OMG® Unified Modeling Language® (OMG UML®) [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://www.omg.org/spec/UML/2.5.1/PDF.
13. Model-View-ViewModel (MVVM)[Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm Дата звернення: 2023
14. C# Best Practices : Dangers of Violating SOLID Principles in C# [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/archive/msdn-magazine/2014/may/csharp-best-practices-dangers-of-violating-solid-principles-in-csharp
15. Vaskaran Sarcar. Simple and Efficient Programming with C# / Vaskaran Sarcar., 2021.
16. Composed Method and SLAP [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://www.oreilly.com/library/view/the-productive-programmer/9780596519780/ch13.html Дата звернення: 2023

ДОДАТОК А

Class App

```
using Xamarin.Forms;
using HappyPet.Model.Common;
using HappyPet.Common;
using HappyPet.UI.MenuFlyoutPage;
using HappyPet.DataBase;

namespace HappyPet
{
   /// <summary>
   ///
   /// </summary>
   public partial class App : Application
   {
      /// <summary>
      ///
      /// </summary>
      private readonly CommonAppProperties _properties;

      /// <summary>
      /// Initialize Application on Xamarin Forms side.
      /// </summary>
      public App()
      {
         var DBConnectionModels = new DBConnectionModels();
         var navigationPage = new CustomNavigationPage(new
StartPage(DBConnectionModels));
         object startTrigger;
         if (Current.Properties.TryGetValue("startPageTrigger", out startTrigger))
         {

            if ((bool)startTrigger)
            {
               MainPage = new FlyoutPageMenu(DBConnectionModels);
            }
            else
            {
               MainPage = navigationPage;
            }
         }
         else
         {
```

```
            _properties = new CommonAppProperties();
            _properties.RegisterProperties();
            MainPage = navigationPage;
        }
        InitializeComponent();
    }
```

Start page

```
using System;
using Xamarin.Forms;
using HappyPet.UI.RegisterPages;
using HappyPet.Interface;

namespace HappyPet
{
    public partial class StartPage : ContentPage
    {
        private IConnectionModels _connectionModels;
        public StartPage(IConnectionModels models)
        {
            _connectionModels = models;
            NavigationPage.SetHasNavigationBar(this, false);
            InitializeComponent();
        }

        async void OnButtonClicked(object sender, EventArgs args)
        {
            await Navigation.PushAsync(new RegisterPage(_connectionModels));
        }
    }
}
```

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HappyPet.StartPage">

    <StackLayout BackgroundColor="#F5F5FF">
        <Grid BackgroundColor="Transparent">
            <Grid.RowDefinitions>
                <RowDefinition Height="0.1*"/>
                <RowDefinition/>
                <RowDefinition Height="0.3*"/>
                <RowDefinition Height="0.3*"/>
```

```
            <RowDefinition Height="0.3*"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="0.5"/>
            <ColumnDefinition/>
            <ColumnDefinition Width="0.5"/>
        </Grid.ColumnDefinitions>
        <Image Source="res/drawable/logoakita.png" Grid.Row="1"
Grid.Column="1"></Image>
        <Label Grid.Row="2" Text="HAPPY PET" Grid.Column="1"
HorizontalTextAlignment="Center" FontSize="38" FontFamily=""
FontAttributes="Bold" TextColor="Black"></Label>
        <Button Clicked="OnButtonClicked" TextColor="White" Text="next"
HeightRequest="85" WidthRequest="85" FontAttributes="Bold"
BackgroundColor="#FFB46B" FontSize="Large" Grid.Row="3" Grid.Column="1"
VerticalOptions="Center" HorizontalOptions="Center"
CornerRadius="100"></Button>
    </Grid>
  </StackLayout>

</ContentPage>
```

Registration

```
using HappyPet.DataBase.Models;
using HappyPet.Interface;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace HappyPet.UI.RegisterPages
{
  [XamlCompilation(XamlCompilationOptions.Compile)]
  public partial class RegisterPage : ContentPage
  {
    private List<Entry> _entrys = new List<Entry>();
    private List<Picker> _pickers = new List<Picker>();
    private IConnectionModels _connectionModels;
    public RegisterPage(IConnectionModels models)
```

```
        {
            _connectionModels = models;

            NavigationPage.SetHasNavigationBar(this, false);
            InitializeComponent();
            AddEntryesAndPickersToList();

            var tapConfidetialCommand = new TapGestureRecognizer();
            tapConfidetialCommand.Tapped += OnConfidetialPolitikCommand;
            confidentialLabel.GestureRecognizers.Add(tapConfidetialCommand);
        }

        private async void OnConfidetialPolitikCommand(object sender, EventArgs args)
        {
            await Navigation.PushModalAsync(new ConfidentialPage());
        }

        private async void OnNextButtonClicked(object sender, EventArgs args)
        {
            if(Validation())
            {
                //AddDataToDb();
                await Navigation.PushAsync(new ApplyInfoPage(_connectionModels));
            }
        }

        private void AddDataToDb()
        {
            var data = new DBUserModel();
            data.Name = EntryName.Text;
            data.SerName = EntrySerName.Text;
            data.Phone = EntryPhone.Text;
            data.Email = EntryEmail.Text;
            data.Country = PickerCountry.SelectedItem.ToString();
            data.City = PickerCity.SelectedItem.ToString();
            data.UnicalID = "#A000001";
            _connectionModels.UserDbModel.AddData(data);
        }

        private bool Validation()
        {
            var validation = true;
            validation = _entrys.Select(p => !string.IsNullOrEmpty(p.Text)).All(e => e ==
true);
```

```
            validation = _pickers.Select(p => !string.IsNullOrEmpty(p.SelectedItem?.ToString())).All(e => e == true);
            return validation;
        }

        private void AddEntryesAndPickersToList()
        {
            _entrys.Add(EntryName);
            _entrys.Add(EntrySerName);
            _entrys.Add(EntryPhone);
            _entrys.Add(EntryEmail);
            _pickers.Add(PickerCountry);
            _pickers.Add(PickerCity);
        }
    }
}
```

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HappyPet.UI.RegisterPages.RegisterPage">
    <ContentPage.Content>
        <Grid VerticalOptions="FillAndExpand" BackgroundColor="#F5F5FF">
            <Grid.RowDefinitions>
                <RowDefinition Height="0.05*"/>
                <RowDefinition Height="0.1*"/>
                <RowDefinition/>
                <RowDefinition Height="0.05*"/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="0.05*"/>
                <ColumnDefinition/>
                <ColumnDefinition Width="0.05*"/>
            </Grid.ColumnDefinitions>
            <Label Text="Реєстрація" Grid.Row="1" Grid.Column="1" TextColor="Black" FontSize="24" HorizontalTextAlignment="Center" FontAttributes="Bold"/>
            <StackLayout Grid.Row="2" Grid.Column="1">
                <Entry x:Name="EntryName" Placeholder="Ім'я" ClearButtonVisibility="WhileEditing" HorizontalOptions="FillAndExpand" Margin="0" Keyboard="Text" MaxLength="25"/>
                <Entry x:Name="EntrySerName" Placeholder="Прізвище" ClearButtonVisibility="WhileEditing" HorizontalOptions="FillAndExpand" Margin="0" Keyboard="Text" MaxLength="25"/>
                <Picker x:Name="PickerCountry" Title="Країна">
```

```
                <Picker.Items>
                    <x:String>Україна</x:String>
                </Picker.Items>
            </Picker>
            <Picker Title="Місто" x:Name="PickerCity">
                <Picker.Items>
                    <x:String>Київ</x:String>
                </Picker.Items>
            </Picker>
            <Entry x:Name="EntryPhone" Placeholder="+380000000000"
ClearButtonVisibility="WhileEditing" HorizontalOptions="FillAndExpand"
Margin="0" Keyboard="Telephone" MaxLength="13"/>
            <Entry x:Name="EntryEmail" Placeholder="email@email.com"
ClearButtonVisibility="WhileEditing" HorizontalOptions="FillAndExpand"
Margin="0" Keyboard="Email"/>

            <Label x:Name="confidentialLabel" Text="Політика конфіденційності"
HorizontalOptions="Center" TextDecorations="Underline" FontAttributes="Italic"/>

            <StackLayout Orientation="Horizontal">
                <CheckBox Color="Orange" x:Name="confidentialCheckBox"/>
                <Label Text="Я пододжуюся із політикою конфіденційності."
FontAttributes="Italic" VerticalOptions="Center"/>
            </StackLayout>
            <Button Clicked="OnNextButtonClicked" IsEnabled="False" Text="Далі"
VerticalOptions="EndAndExpand" HorizontalOptions="Center"
BackgroundColor="#FFB46B" CornerRadius="10" FontAttributes="Bold"
FontSize="18">
                <Button.Triggers>
                    <DataTrigger TargetType="Button" Binding="{Binding
Source={x:Reference confidentialCheckBox}, Path=IsChecked}" Value="true">
                        <Setter Property="IsEnabled" Value="true"/>
                    </DataTrigger>
                </Button.Triggers>
            </Button>
        </StackLayout>
    </Grid>
  </ContentPage.Content>
</ContentPage>

using HappyPet.Common;
using HappyPet.DataBase.Models;
using HappyPet.Interface;
using HappyPet.UI.MenuFlyoutPage;
```

```
using System;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace HappyPet.UI.RegisterPages
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class PetRegisterPage : ContentPage
    {
        private IConnectionModels _connectionModels;
        public PetRegisterPage(IConnectionModels models)
        {
            _connectionModels = models;
            NavigationPage.SetHasNavigationBar(this, false);
            InitializeComponent();
            foreach (var type in RegisterConstants.AnimalTypesList)
            {
                PickerType.Items.Add(type);
            }
            pushModalAssistentPage();
            PickerType.SelectedIndexChanged += OnPickerTypeIndexChanged;
        }

        private async void OnNextButtonClicked(object sender, EventArgs args)
        {
            if(Validation())
            {
                //AddDataToDb();
                Application.Current.MainPage = new FlyoutPageMenu(_connectionModels);
            }
        }

        private async void pushModalAssistentPage()
        {
            await Navigation.PushModalAsync(new ModalAssistentPage());
        }

        private void OnPickerTypeIndexChanged(object sender, EventArgs args)
        {
            if(PickerBreed.Items.Count>0)
            {
                PickerBreed.Items.Clear();
            }
            var type = (AnimalTypes)PickerType.SelectedIndex;
```

```
            foreach (var greed in RegisterConstants.AnimalsDictionary[type])
            {
                PickerBreed.Items.Add(greed);
            }
        }

        private bool Validation()
        {
            var valid = !string.IsNullOrEmpty(EntryName.Text);
            valid = !string.IsNullOrEmpty(PickerType.SelectedItem?.ToString());
            valid = !string.IsNullOrEmpty(PickerBreed.SelectedItem?.ToString());
            valid = !string.IsNullOrEmpty(PickerSex.SelectedItem?.ToString());
            valid = !string.IsNullOrEmpty(DateOfBirthdayPicker.Date.ToString());
            return valid;
        }

        private void AddDataToDb()
        {
            var data = new DBPetsModel();
            data.Name = EntryName.Text;
            data.Sex = PickerSex.SelectedItem.ToString();
            data.Type = PickerType.SelectedItem.ToString();
            data.Breed = PickerBreed.SelectedItem.ToString();
            data.DateOfBirth = DateOfBirthdayPicker.Date;
            data.UnicalID = "#A0000001";
            _connectionModels.PetsDbModel.AddData(data);
        }
    }
}<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HappyPet.UI.RegisterPages.PetRegisterPage">
    <ContentPage.Content>
        <Grid VerticalOptions="FillAndExpand" BackgroundColor="#F5F5FF">
            <Grid.RowDefinitions>
                <RowDefinition Height="0.05*"/>
                <RowDefinition Height="0.1*"/>
                <RowDefinition/>
                <RowDefinition Height="0.05*"/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="0.05*"/>
                <ColumnDefinition/>
                <ColumnDefinition Width="0.05*"/>
```

```
        </Grid.ColumnDefinitions>
        <Label Text="Улюбленець" Grid.Row="1" Grid.Column="1"
TextColor="Black" FontSize="24" HorizontalTextAlignment="Center"
FontAttributes="Bold"/>
        <StackLayout Grid.Row="2" Grid.Column="1">
          <Entry x:Name="EntryName" Placeholder="Ім'я"
ClearButtonVisibility="WhileEditing" HorizontalOptions="FillAndExpand"
Margin="0" Keyboard="Text" MaxLength="25"/>
          <Picker Title="Вид" x:Name="PickerType">
          </Picker>
          <Picker Title="Порода" x:Name="PickerBreed">
          </Picker>
          <Picker Title="Стать" x:Name="PickerSex">
            <Picker.Items>
              <x:String>М</x:String>
              <x:String>Ж</x:String>
            </Picker.Items>
          </Picker>
          <Label Text="Дата народження:" Margin="4,5,0,0"/>
          <DatePicker x:Name="DateOfBirthdayPicker"/>

          <Button Clicked="OnNextButtonClicked" Text="Далі"
VerticalOptions="EndAndExpand" HorizontalOptions="Center"
BackgroundColor="#FFB46B" CornerRadius="10" FontAttributes="Bold"
FontSize="18"/>

        </StackLayout>
      </Grid>
    </ContentPage.Content>
</ContentPage>using HappyPet.Interface;
using System;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace HappyPet.UI.RegisterPages
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class ApplyInfoPage : ContentPage
    {
        private IConnectionModels _connectionModels;
        public ApplyInfoPage(IConnectionModels models)
        {
            _connectionModels = models;
            NavigationPage.SetHasNavigationBar(this, false);
```

```
        InitializeComponent();
    }

    private async void OnNextButtonClicked(object sender, EventArgs args)
    {
        if(EntrySecurityCode.Text == "07535715")
        {
            await Navigation.PushAsync(new PetRegisterPage(_connectionModels));
        }
    }
    private async void OnPreviousButtonClicked(object sender, EventArgs args)
    {
        await Navigation.PopAsync();
    }
  }
}<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HappyPet.UI.RegisterPages.ApplyInfoPage">
  <ContentPage.Content>
    <StackLayout BackgroundColor="#F5F5FF"
HorizontalOptions="FillAndExpand">
      <StackLayout VerticalOptions="CenterAndExpand">
        <Label TextColor="Black" Text="Підтвердження реєстрації" FontSize="28"
HorizontalOptions="Center"/>
        <Label TextColor="Black" Text="Уведіть код із смс повідомлення"
FontSize="24" HorizontalOptions="Center"/>
        <Entry x:Name="EntrySecurityCode" Placeholder="07535715"
FontSize="22" Keyboard="Numeric" MaxLength="10" HorizontalOptions="Center"/>
        <StackLayout Orientation="Horizontal">
          <Button CornerRadius="10" Clicked="OnPreviousButtonClicked"
HorizontalOptions="CenterAndExpand" Text="Назад"
BackgroundColor="#FFB46B"/>
          <Button CornerRadius="10" Clicked="OnNextButtonClicked"
HorizontalOptions="CenterAndExpand" Text="Далі" BackgroundColor="#FFB46B"/>
        </StackLayout>
      </StackLayout>
    </StackLayout>
  </ContentPage.Content>
</ContentPage>
```

Main Page

```
using HappyPet.DataBase.Models;
using HappyPet.Interface;
```

```
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
using System.Linq;
using System.Collections.Generic;
using System;
using HappyPet.UI.PetCerouselPages;
using HappyPet.UI.ViewModels;
using HappyPet.UI.SubPages;

namespace HappyPet.UI
{
   [XamlCompilation(XamlCompilationOptions.Compile)]
   public partial class MainMenuPage : ContentPage
   {
      private bool _selectedFlag = false;
      private MainMenuAdditionalItemViewModel _itemViewModel;
      private IConnectionModels _connectionModels;
      public MainMenuPage(IConnectionModels models)
      {
         _connectionModels = models;
         InitializeComponent();
         ConnectionDB();
         _itemViewModel = new MainMenuAdditionalItemViewModel();
         BindingContext = _itemViewModel;
         MainMenuInfoListView.ItemSelected += OnItemSelected;
         MainMenuInfoListView.ItemTapped += OnItemTapped;
      }

      private void ConnectionDB()
      {
         var detailsPet = (from x in
_connectionModels.PetsDbModel.Connection.Table<DBPetsModel>() select
x).ToList();
         ConnectPetsLabelsToDB(detailsPet);
      }

      private void ConnectPetsLabelsToDB(List<DBPetsModel> db)
      {
         var details = db.First();
         PetNameLabel.Text = details.Name;
         PetTypeLabel.Text = details.Type;
         PetBreedLabel.Text = details.Breed;
         PetStatysLabel.Text = "Бешкетує";
      }
```

```
        /// <summary>
        /// Pets button clicked.
        /// </summary>
        /// <param name="sender"> Object sender. </param>
        /// <param name="args"> Event args. </param>
        private async void PetsButtonClicked(object sender, EventArgs args)
        {
            await Navigation.PushAsync(new PetCarouselMain(_connectionModels));
        }


        private async void OnItemSelected(object sender, SelectedItemChangedEventArgs
e)
        {
            _selectedFlag = true;
            await Navigation.PushModalAsync(new MainMenuItemsSubPage());
        }

        private async void OnItemTapped(object sender, EventArgs e)
        {
            if (!_selectedFlag)
            {
                await Navigation.PushModalAsync(new MainMenuItemsSubPage());
            }
            _selectedFlag = false;
        }
    }
}
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HappyPet.UI.MainMenuPage">
    <ContentPage.Content>
        <StackLayout BackgroundColor="#F5F5FF">
            <Grid>
                <Grid.RowDefinitions>
                    <RowDefinition Height="0.1*" />
                    <RowDefinition Height="170"/>
                    <RowDefinition Height="0.2*"/>
                    <RowDefinition/>
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="0.1*"/>
```

```
            <ColumnDefinition/>
            <ColumnDefinition/>
            <ColumnDefinition Width="0.1*"/>
        </Grid.ColumnDefinitions>
        <BoxView Grid.Row="1" Grid.Column="1" BackgroundColor="White"
CornerRadius="100"/>
        <Image Source="res/drawable/logoakita.png" Grid.Row="1"
Grid.Column="1"></Image>
        <ScrollView Grid.Row="3" Grid.Column="1" Grid.ColumnSpan="2"
BackgroundColor="Transparent">
            <ListView x:Name="MainMenuInfoListView" HeightRequest="100"
BackgroundColor="Transparent"
                      SeparatorVisibility="None"
                      HasUnevenRows="true"
                      ItemsSource="{Binding MainMenuInfoItems}">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <Frame Padding="10" CornerRadius="10" Margin="10"
BackgroundColor="White">
                                <StackLayout Orientation="Horizontal" Padding="15,10">
                                    <Image Source="res/drawable/user.png"
HeightRequest="50"/>
                                    <StackLayout HorizontalOptions="FillAndExpand">
                                        <Label
                                            VerticalTextAlignment="Center"
                                            Text="{Binding Title}"
                                            FontSize="22" TextColor="Black"
FontAttributes="Bold"/>
                                        <Label Text="{Binding MainInfo}"
VerticalOptions="EndAndExpand" FontSize="18"/>
                                        <Line/>
                                    </StackLayout>
                                </StackLayout>
                            </Frame>
                        </ViewCell>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
        </ScrollView>
        <Button Grid.Row="2" HeightRequest="60" Grid.Column="1" Text="Дії"
CornerRadius="10" FontSize="22" FontAttributes="Bold"
BackgroundColor="#FFB46B"/>
```

```
            <Button Grid.Row="2" Clicked="PetsButtonClicked" Grid.Column="2" Text="Улюбленець" CornerRadius="10" FontSize="22" FontAttributes="Bold" BackgroundColor="White"/>
            <StackLayout Grid.Row="1" Grid.Column="2" VerticalOptions="Center">
              <StackLayout Orientation="Horizontal">
                <Label Text="Ім'я: " FontAttributes="Bold" FontSize="18" TextColor="Black"></Label>
                <Label x:Name="PetNameLabel" Text="Лара " FontAttributes="Bold" FontSize="18" TextColor="Black"></Label>
              </StackLayout>
              <StackLayout Orientation="Horizontal">
                <Label Text="Вид: " FontAttributes="Bold" FontSize="18" TextColor="Black"></Label>
                <Label x:Name="PetTypeLabel" Text="Акіта " FontAttributes="Bold" FontSize="18" TextColor="Black"></Label>
              </StackLayout>
              <StackLayout Orientation="Horizontal">
                <Label Text="Порода: " FontAttributes="Bold" FontSize="18" TextColor="Black"></Label>
                <Label x:Name="PetBreedLabel" Text="27 кг " FontAttributes="Bold" FontSize="18" TextColor="Black"></Label>
              </StackLayout>
              <StackLayout Orientation="Horizontal">
                <Label Text="Статус: " FontAttributes="Bold" FontSize="18" TextColor="Black"></Label>
                <Label x:Name="PetStatysLabel" Text="Спить" FontAttributes="Bold" FontSize="18" TextColor="Black"></Label>
              </StackLayout>
            </StackLayout>
          </Grid>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Pages

```
using HappyPet.UI.SubPages;
using HappyPet.UI.ViewModels;
using System;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
using HappyPet.Interface;

namespace HappyPet.UI.FunctionalPages
{
```

```
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class ShopPage : ContentPage
    {
        private IConnectionModels _connectionModels;
        private bool _selectedFlag = false;
        private ShopInfoViewModel _itemViewModel;
        public ShopPage(IConnectionModels models)
        {
            _connectionModels = models;
            InitializeComponent();
            _itemViewModel = new ShopInfoViewModel();
            BindingContext = _itemViewModel;
            ShopInfoListView.ItemSelected += OnItemSelected;
            ShopInfoListView.ItemTapped += OnItemTapped;
        }

        private async void OnItemSelected(object sender, SelectedItemChangedEventArgs
e)
        {
            _selectedFlag = true;
            await Navigation.PushModalAsync(new ShopSubPage());
        }

        private async void OnItemTapped(object sender, EventArgs e)
        {
            if (!_selectedFlag)
            {
                await Navigation.PushModalAsync(new ShopSubPage());
            }
            _selectedFlag = false;
        }
    }
}
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HappyPet.UI.FunctionalPages.ShopPage">
    <ContentPage.Content>
        <StackLayout                                    HorizontalOptions="FillAndExpand"
BackgroundColor="#F5F5FF">
            <ListView x:Name="ShopInfoListView"
              SeparatorVisibility="None"
              HasUnevenRows="true"
              ItemsSource="{Binding ShopInfo}">
```

```
            <ListView.ItemTemplate>
               <DataTemplate>
                  <ViewCell>
                     <StackLayout Orientation="Horizontal" Padding="15,10">
                        <Image Source="res/drawable/user.png" HeightRequest="50"/>
                        <StackLayout HorizontalOptions="FillAndExpand">
                           <Label
                        VerticalTextAlignment="Center"
                        Text="{Binding Title}"
                        FontSize="22" TextColor="Black" FontAttributes="Bold"/>
                           <Label       Text="Text"       VerticalOptions="EndAndExpand"
FontSize="18"/>
                        </StackLayout>
                     </StackLayout>
                  </ViewCell>
               </DataTemplate>
            </ListView.ItemTemplate>
         </ListView>
      </StackLayout>
   </ContentPage.Content>
</ContentPage>
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using HappyPet.DataBase.Models;
using HappyPet.Interface;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace HappyPet.UI.FunctionalPages
{
   [XamlCompilation(XamlCompilationOptions.Compile)]
   public partial class ProfilePage : ContentPage
   {
      private IConnectionModels _connectionModels;
      public ProfilePage(IConnectionModels models)
      {
         _connectionModels = models;
         InitializeComponent();
         ConnectionToDb();
      }
```

```
        private void ConnectionToDb()
        {
            var detailsUser = (from x in _connectionModels.UserDbModel.connection.Table<DBUserModel>() select x).ToList();
            ConnectDataToLabels(detailsUser);
        }

        private void ConnectDataToLabels(List<DBUserModel> db)
        {
            var details = db.First();
            LabelCity.Text = details.City;
            LabelCountry.Text = details.Country;
            LabelEmail.Text = details.Email;
            LabelName.Text = details.Name;
            LabelPhone.Text = details.Phone;
            LabelSerName.Text = details.SerName;
            LabelUnicalID.Text = details.UnicalID;

        }
    }
}
```

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HappyPet.UI.FunctionalPages.ProfilePage">
    <ContentPage.Content>
        <Grid BackgroundColor="#F5F5FF">
            <Grid.RowDefinitions>
                <RowDefinition Height="0.2*"/>
                <RowDefinition/>
                <RowDefinition Height="0.1*"/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="0.1*"/>
                <ColumnDefinition/>
                <ColumnDefinition Width="0.1*"/>
            </Grid.ColumnDefinitions>
            <ImageButton Padding="10, 0, 0, 0" Source="res/drawable/user.png" Grid.Row="0" Grid.Column="1" BackgroundColor="Transparent"></ImageButton>
            <StackLayout Grid.Row="1" Grid.Column="1">
                <StackLayout Orientation="Horizontal" Padding="5, 0">
                    <Label Text="Ім'я:" FontSize="24" TextColor="Black"/>
```

```
            <Label      x:Name="LabelName"      Text="Катерина"      FontSize="24"
TextColor="Black"/>
          </StackLayout>
          <StackLayout Orientation="Horizontal" Padding="5, 0">
            <Label Text="Прізвище:" FontSize="24" TextColor="Black"/>
            <Label    x:Name="LabelSerName"    Text="Смирнова"    FontSize="24"
TextColor="Black"/>
          </StackLayout>
          <StackLayout Orientation="Horizontal" Padding="5, 0">
            <Label Text="Телефон:" FontSize="24" TextColor="Black"/>
            <Label   x:Name="LabelPhone"   Text="+3800000000000"   FontSize="24"
TextColor="Black"/>
          </StackLayout>
          <StackLayout Orientation="Horizontal" Padding="5, 0">
            <Label Text="Email:" FontSize="24" TextColor="Black"/>
            <Label   x:Name="LabelEmail"   Text="email@email.com"   FontSize="24"
TextColor="Black"/>
          </StackLayout>
          <StackLayout Orientation="Horizontal" Padding="5, 0">
            <Label Text="Країна:" FontSize="24" TextColor="Black"/>
            <Label      x:Name="LabelCountry"      Text="Україна"      FontSize="24"
TextColor="Black"/>
          </StackLayout>
          <StackLayout Orientation="Horizontal" Padding="5, 0">
            <Label Text="Місто:" FontSize="24" TextColor="Black"/>
            <Label         x:Name="LabelCity"         Text="Київ"         FontSize="24"
TextColor="Black"/>
          </StackLayout>
          <StackLayout Orientation="Horizontal" Padding="5, 0">
            <Label Text="User ID:" FontSize="24"/>
            <Label x:Name="LabelUnicalID" Text="#A0000001" FontSize="24"/>
          </StackLayout>
          <StackLayout Orientation="Horizontal" VerticalOptions="End">
            <Button       Text="Вийти"       FontSize="18"       FontAttributes="Bold"
TextColor="White"              HorizontalOptions="Start"              CornerRadius="10"
BackgroundColor="#FFB46B"/>
            <Button          WidthRequest="50"          Text="|"          FontSize="18"
FontAttributes="Bold"     TextColor="White"     HorizontalOptions="EndAndExpand"
CornerRadius="50" BackgroundColor="#FFB46B"/>
          </StackLayout>
        </StackLayout>
      </Grid>
    </ContentPage.Content>
</ContentPage>
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using HappyPet.Interface;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace HappyPet.UI.FunctionalPages
{
   [XamlCompilation(XamlCompilationOptions.Compile)]
   public partial class PetSitterPage : ContentPage
   {
      private IConnectionModels _connectionModels;
      public PetSitterPage(IConnectionModels models)
      {
         _connectionModels = models;
         InitializeComponent();
      }
   }
}
```

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
          xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
          x:Class="HappyPet.UI.FunctionalPages.PetSitterPage">
   <ContentPage.Content>
      <StackLayout>
         <Label Text="Welcome to Xamarin.Forms!"
            VerticalOptions="CenterAndExpand"
            HorizontalOptions="CenterAndExpand" />
      </StackLayout>
   </ContentPage.Content>
</ContentPage>
```

```
using HappyPet.UI.SubPages;
using HappyPet.UI.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using HappyPet.Interface;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace HappyPet.UI.FunctionalPages
{
   [XamlCompilation(XamlCompilationOptions.Compile)]
   public partial class HospitalPage : ContentPage
   {
      private IConnectionModels _connectionModels;
      private bool _selectedFlag = false;
      private HospitalInfoViewModel _itemViewModel;
      public HospitalPage(IConnectionModels models)
      {
         _connectionModels = models;
         InitializeComponent();
         _itemViewModel = new HospitalInfoViewModel();
         BindingContext = _itemViewModel;
         HospitalInfoListView.ItemSelected += OnItemSelected;
         HospitalInfoListView.ItemTapped += OnItemTapped;
      }

      private async void OnItemSelected(object sender, SelectedItemChangedEventArgs
e)
      {
         _selectedFlag = true;
         await Navigation.PushModalAsync(new HospitalSubPage());
      }

      private async void OnItemTapped(object sender, EventArgs e)
      {
         if (!_selectedFlag)
         {
            await Navigation.PushModalAsync(new HospitalSubPage());
         }
         _selectedFlag = false;
      }
   }
}

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
          xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
          x:Class="HappyPet.UI.FunctionalPages.HospitalPage">
```

```
    <ContentPage.Content>
        <StackLayout                                    HorizontalOptions="FillAndExpand"
BackgroundColor="#F5F5FF">
            <ListView x:Name="HospitalInfoListView"
              SeparatorVisibility="None"
              HasUnevenRows="true"
              ItemsSource="{Binding HospitalInfo}">
               <ListView.ItemTemplate>
                  <DataTemplate>
                     <ViewCell>
                        <StackLayout Orientation="Horizontal" Padding="15,10">
                           <Image Source="res/drawable/user.png" HeightRequest="50"/>
                           <StackLayout HorizontalOptions="FillAndExpand">
                              <Label
                           VerticalTextAlignment="Center"
                           Text="{Binding Title}"
                           FontSize="22" TextColor="Black" FontAttributes="Bold"/>
                              <Label        Text="Text"        VerticalOptions="EndAndExpand"
FontSize="18"/>
                           </StackLayout>
                        </StackLayout>
                     </ViewCell>
                  </DataTemplate>
               </ListView.ItemTemplate>
            </ListView>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>

using HappyPet.Items;
using HappyPet.UI.SubPages;
using HappyPet.UI.ViewModels;
using System;
using System.Linq;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
using HappyPet.Interface;
using HappyPet.DataBase.Models;
using System.Collections.Generic;

namespace HappyPet.UI.FunctionalPages
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class HealthPage : ContentPage
```

```
    {
        private IConnectionModels _connectionModels;
        private bool _selectedFlag = false;
        private HealthInfoViewModel _itemViewModel;
        public HealthPage(IConnectionModels models)
        {
            _connectionModels = models;
            InitializeComponent();
            HealthInfoListView.ItemSelected += OnItemSelected;
            HealthInfoListView.ItemTapped += OnItemTapped;
            var details = DBConnection();
            _itemViewModel = new HealthInfoViewModel(details);
            BindingContext = _itemViewModel;
        }

        private async void OnItemSelected(object sender, SelectedItemChangedEventArgs
e)
        {
            _selectedFlag = true;
            await                                    Navigation.PushModalAsync(new
HealthInfoSubPage(_connectionModels));
        }

        private async void OnItemTapped(object sender, EventArgs e)
        {
            if (!_selectedFlag)
            {
                await                                Navigation.PushModalAsync(new
HealthInfoSubPage(_connectionModels));
            }
            _selectedFlag = false;
        }

        private async void OnAddButtonClicked(object sender, EventArgs args)
        {
            await                                    Navigation.PushModalAsync(new
HealthInfoSubPage(_connectionModels));
        }

        private List<DBGlobalModel> DBConnection()
        {
            return                (from                x                in
_connectionModels.GlobalDBModel.Connection.Table<DBGlobalModel>()        select
x).Where(p => p.Type == DBGlobalTypesOfData.HealthInfo).ToList();
```

```
        }
    }
}
```

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HappyPet.UI.FunctionalPages.HealthPage">
    <ContentPage.Content>
        <StackLayout HorizontalOptions="FillAndExpand"
BackgroundColor="#F5F5FF">
            <ListView x:Name="HealthInfoListView"
              SeparatorVisibility="None"
              HasUnevenRows="true"
              ItemsSource="{Binding HealthInfo}">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <StackLayout Padding="15,10"
HorizontalOptions="FillAndExpand">
                                <Label
                                VerticalTextAlignment="Center"
                                Text="{Binding Title}"
                                FontSize="22" TextColor="Black" FontAttributes="Bold"/>
                                <Label Text="{Binding MainInfo}"
VerticalOptions="EndAndExpand" FontSize="16"/>
                            </StackLayout>
                        </ViewCell>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
            <ImageButton Clicked="OnAddButtonClicked" Source="res/drawable/plus.png"
BackgroundColor="Transparent" HeightRequest="100" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Data base and functional

```
using Xamarin.Forms;

namespace HappyPet.Model.Common
{
    /// <summary>
```

```
    /// Custom navigation page.
    /// </summary>
    public class CustomNavigationPage : NavigationPage
    {
        /// <summary>
        /// Initializer <see cref="CustomNavigationPage"/>
        /// </summary>
        /// <param name="page">Page object</param>
        public CustomNavigationPage(Page page) : base(page)
        {
            ActiveStartPage = page;
        }

        /// <summary>
        /// Active start page.
        /// </summary>
        public Page ActiveStartPage { get; set; }
    }
}

using System;

namespace HappyPet.Items
{
    public class HealthInfoItem
    {
        public HealthInfoItem()
        {
            TargetType = typeof(HealthInfoItem);
        }

        public int Id { get; set; }

        public string Title { get; set; }

        public string MainInfo { get; set; }

        public string AdditionalInfo { get; set; }
        public string SubAdditionalInfo { get; set; }

        public DateTime PickerDate { get; set; }
        public Type TargetType { get; set; }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;

namespace HappyPet.Items
{
    public class HospitalInfoItem
    {
        public HospitalInfoItem()
        {
            TargetType = typeof(ShopInfoItem);
        }

        public int Id { get; set; }

        public string Title { get; set; }

        public string MainInfo { get; set; }

        public Type TargetType { get; set; }
    }
}

using System;

namespace HappyPet.Items
{
    public class MainMenuInfoItem
    {
        public MainMenuInfoItem()
        {
            TargetType = typeof(MainMenuInfoItem);
        }

        public int Id { get; set; }

        public string Title { get; set; }

        public string MainInfo { get; set; }

        public string AdditionalInfo { get; set; }
        public string SubAdditionalInfo { get; set; }
```

```
        public DateTime PickerDate { get; set; }
        public Type TargetType { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.Text;

namespace HappyPet.Items
{
    public class ShopInfoItem
    {
        public ShopInfoItem()
        {
            TargetType = typeof(HospitalInfoItem);
        }

        public int Id { get; set; }

        public string Title { get; set; }

        public string MainInfo { get; set; }

        public Type TargetType { get; set; }
    }
}

using SQLite;

namespace HappyPet.Interface
{
    public interface IConnectionModel
    {
        void AddData(IDBBaseModel model);

        void RemoveData(IDBBaseModel model);

        void ClearAllData();

        SQLiteConnection Connection { get; }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;

namespace HappyPet.Interface
{
    public interface IConnectionModels
    {
        IConnectionModel GlobalDBModel { get; }

        IConnectionModel PetsDbModel { get; }

        IConnectionModel UserDbModel { get; }

    }
}

using System;
using System.Collections.Generic;
using System.Text;

namespace HappyPet.Interface
{
    public interface IDBBaseModel
    {
        int ID { get; set; }

        string UnicalID { get; set; }
    }
}

using SQLite;

namespace HappyPet.Interface
{
    /// <summary>
    /// Sql lite connection interface.
    /// </summary>
    public interface ISqLiteConnection
    {
        /// <summary>
        /// Get connection method.
        /// </summary>
        /// <param name="path">File path.</param>
```

```
        /// <returns>SQL connection.</returns>
        SQLiteConnection GetConnection(string path);
    }
}

using HappyPet.DataBase.Models;
using HappyPet.Interface;
using SQLite;
using System;
using Xamarin.Forms;

namespace HappyPet.DataBase
{
    /// <summary>
    /// Data base user connection model.
    /// </summary>
    public class DBUserConnectionModel : IConnectionModel
    {
        /// <summary>
        /// SQLite connection.
        /// </summary>
        public SQLiteConnection Connection { get; }

        /// <summary>
        /// Initialize <see cref="DBUserConnectionModel"/>
        /// </summary>
        public DBUserConnectionModel()
        {
            Connection                                                    =
DependencyService.Get<ISqLiteConnection>().GetConnection(DataBaseConstants.Dat
aBaseUserPath);
            Connection.CreateTable<DBUserModel>();
        }

        /// <summary>
        /// Add data to DB.
        /// </summary>
        /// <param name="model">Data Model.</param>
        public void AddData(IDBBaseModel model)
        {
            try
            {
                Connection.Insert(model);
            }
```

```
            catch
            {
                throw new Exception();
            }
        }

        /// <summary>
        /// Remove data from DB.
        /// </summary>
        /// <param name="model">Data Model.</param>
        public void RemoveData(IDBBaseModel model)
        {
            try
            {
                Connection.Delete(model);
            }
            catch
            {
                throw new Exception();
            }
        }

        /// <summary>
        /// Clear all data in DB.
        /// </summary>
        /// <param name="model">Data Model.</param>
        public void ClearAllData()
        {
            try
            {
                Connection.DeleteAll<DBUserModel>();
            }
            catch
            {
                throw new Exception();
            }
        }
    }
}

using HappyPet.DataBase.Models;
using HappyPet.Interface;
using System.Linq;
```

```
namespace HappyPet.DataBase
{
    /// <summary>
    /// Data base user connection model.
    /// </summary>
    public class DBSubGlobalConnectionModel : DBGlobalConnectionModel
    {
        /// <summary>
        /// Initialize <see cref="DBSubGlobalConnectionModel"/>
        /// </summary>
        public DBSubGlobalConnectionModel():base()
        {
        }

        /// <summary>
        /// Change row Unical Id.
        /// </summary>
        /// <param name="model"></param>
        public void ChangeRowUnicalID(IDBBaseModel model)
        {
            var rows =  (from x in Connection.Table<DBGlobalModel>() select x).Where(p
=> p.Type == DBGlobalTypesOfData.HealthInfo).ToList();
            rows.First().UnicalID = model.UnicalID;
        }
    }
}

using HappyPet.DataBase.Models;
using HappyPet.Interface;
using System;
using Xamarin.Forms;

namespace HappyPet.DataBase
{
    /// <summary>
    /// Data base pets connection model.
    /// </summary>
    public class DBPetsConnectionModel : DBBaseConnectionModel
    {
        /// <summary>
        /// Initialize <see cref="DBPetsConnectionModel"/>
        /// </summary>
```

```
        public
DBPetsConnectionModel():base(DependencyService.Get<ISqLiteConnection>().GetCo
nnection(DataBaseConstants.DataBasePetsPath))
        {
            Connection.CreateTable<DBPetsModel>();
        }

        /// <summary>
        /// Clear all data in DB.
        /// </summary>
        /// <param name="model">Data Model.</param>
        public override void ClearAllData()
        {
            try
            {
                Connection.DeleteAll<DBPetsModel>();
            }
            catch
            {
                throw new Exception();
            }
        }
    }
}

using SQLite;
using System;
using Xamarin.Forms;
using HappyPet.DataBase.Models;
using HappyPet.Interface;

namespace HappyPet.DataBase
{
    /// <summary>
    /// Data base global connection model.
    /// </summary>
    public class DBGlobalConnectionModel : IConnectionModel
    {
        /// <summary>
        /// SQLite connection.
        /// </summary>
        public SQLiteConnection Connection { get; }

        /// <summary>
```

```
        /// Initialize <see cref="DBGlobalConnectionModel"/>
        /// </summary>
        public DBGlobalConnectionModel()
        {
            Connection = DependencyService.Get<ISqLiteConnection>().GetConnection(DataBaseConstants.DataBaseGlobalPath);
            Connection.CreateTable<DBGlobalModel>();
        }

        /// <summary>
        /// Add data to DB.
        /// </summary>
        /// <param name="model">Data Model.</param>
        public void AddData(IDBBaseModel model)
        {
            try
            {
                Connection.Insert(model);
            }
            catch
            {
                throw new Exception();
            }
        }

        /// <summary>
        /// Remove data from DB.
        /// </summary>
        /// <param name="model">Data Model.</param>
        public void RemoveData(IDBBaseModel model)
        {
            try
            {
                Connection.Delete(model);
            }
            catch
            {
                throw new Exception();
            }
        }

        /// <summary>
        /// Clear all data in DB.
```

```
        /// </summary>
        /// <param name="model">Data Model.</param>
        public void ClearAllData()
        {
            try
            {
                Connection.DeleteAll<DBGlobalModel>();
            }
            catch
            {
                throw new Exception();
            }
        }
    }
}


using HappyPet.Interface;

namespace HappyPet.DataBase
{
    /// <summary>
    /// Data base connection model class.
    /// </summary>
    public class DBConnectionModels : IConnectionModels
    {
        /// <summary>
        /// Global data base model.
        /// </summary>
        public IConnectionModel GlobalDBModel { get; }

        /// <summary>
        /// Pets data base model.
        /// </summary>
        public IConnectionModel PetsDbModel { get; }

        /// <summary>
        /// User data base model.
        /// </summary>
        public IConnectionModel UserDbModel { get; }

        /// <summary>
        /// Initialize <see cref="DBConnectionModels"/>.
        /// </summary>
```

```
        public DBConnectionModels()
        {
            GlobalDBModel = new DBGlobalConnectionModel();
            PetsDbModel = new DBPetsConnectionModel();
            UserDbModel = new DBUserConnectionModel();
        }
    }
}

using HappyPet.Interface;
using SQLite;
using System;

namespace HappyPet.DataBase
{
    /// <summary>
    /// Data base, base connection model.
    /// </summary>
    public class DBBaseConnectionModel : IConnectionModel
    {
        /// <summary>
        /// SQLite connection.
        /// </summary>
        public SQLiteConnection Connection { get; private set; }

        /// <summary>
        /// Initialize <see cref="DBBaseConnectionModel"/>
        /// </summary>
        public DBBaseConnectionModel(SQLiteConnection connection)
        {
            Connection = connection;
        }

        /// <summary>
        /// Add data to DB.
        /// </summary>
        /// <param name="model">Data Model.</param>
        public virtual void AddData(IDBBaseModel model)
        {
            try
            {
                Connection.Insert(model);
            }
            catch
```

```
            {
                throw new Exception();
            }
        }

        /// <summary>
        /// Remove data from DB.
        /// </summary>
        /// <param name="model">Data Model.</param>
        public virtual void RemoveData(IDBBaseModel model)
        {
            try
            {
                Connection.Delete(model);
            }
            catch
            {
                throw new Exception();
            }
        }

        /// <summary>
        /// Clear all data in DB.
        /// </summary>
        /// <param name="model">Data Model.</param>
        public virtual void ClearAllData()
        {
            try
            {
                Connection.DeleteAll<IDBBaseModel>();
            }
            catch
            {
                throw new Exception();
            }
        }
    }
}


using System;
using System.IO;

namespace HappyPet.DataBase
```

```
{
    /// <summary>
    /// Data base constants
    /// </summary>
    public static class DataBaseConstants
    {
        /// <summary>
        /// Data base user file name.
        /// </summary>
        public const string DataBaseUserFilename = "UserSQLite.db3";

        /// <summary>
        /// Data base pets file name.
        /// </summary>
        public const string DataBasePetsFilename = "PetsSQLite.db3";

        /// <summary>
        /// Data base global file name.
        /// </summary>
        public const string DataBaseGlobalFilename = "GlobalSQLite.db3";

        /// <summary>
        /// Flags to data base controls.
        /// </summary>
        public const SQLite.SQLiteOpenFlags Flags =
            // open the database in read/write mode
            SQLite.SQLiteOpenFlags.ReadWrite |
            // create the database if it doesn't exist
            SQLite.SQLiteOpenFlags.Create |
            // enable multi-threaded database access
            SQLite.SQLiteOpenFlags.SharedCache;

        /// <summary>
        /// Data base user path property.
        /// </summary>
        public static string DataBaseUserPath
        {
            get
            {
                var basePath =
Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
                return Path.Combine(basePath, DataBaseUserFilename);
            }
        }
```

```
        /// <summary>
        /// Data base pets path property.
        /// </summary>
        public static string DataBasePetsPath
        {
            get
            {
                var basePath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
                return Path.Combine(basePath, DataBasePetsFilename);
            }
        }

        /// <summary>
        /// Data base global path property.
        /// </summary>
        public static string DataBaseGlobalPath
        {
            get
            {
                var basePath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
                return Path.Combine(basePath, DataBaseGlobalFilename);
            }
        }
    }
}


using HappyPet.Interface;
using SQLite;
using System;

namespace HappyPet.DataBase.Models
{
    /// <summary>
    /// Data base global model.
    /// </summary>
    public class DBGlobalModel : IDBBaseModel
    {
        [PrimaryKey, AutoIncrement]
        public int ID { get; set; }
        public string UnicalID { get; set; }
```

```
        public DBGlobalTypesOfData Type { get; set; }
        public string Title { get; set; }
        public string About { get; set; }
        public DateTime CreateDate { get; set; }
        public DateTime DatePickerInfo { get; set; }
        public string AdditionalInfo { get; set; }
        public string SubAdditionalInfo { get; set; }
    }
}

namespace HappyPet.DataBase.Models
{
    /// <summary>
    /// Data base global types of data.
    /// </summary>
    public enum DBGlobalTypesOfData
    {
        Hospital = 0,
        ZooShop = 1,
        HealthInfo = 2,
        Default = -999
    }
}


using HappyPet.Interface;
using SQLite;
using System;

namespace HappyPet.DataBase.Models
{
    /// <summary>
    /// Data base pets model.
    /// </summary>
    public class DBPetsModel : IDBBaseModel
    {
        [PrimaryKey, AutoIncrement]
        public int ID { get; set; }
        public string Name { get; set; }
        public string Type { get; set; }
        public string Breed { get; set; }
        public string Sex { get; set; }
        public DateTime DateOfBirth { get; set; }
        public string UnicalID { get; set; }
```

```
    }
}

using HappyPet.Interface;
using SQLite;

namespace HappyPet.DataBase.Models
{
    /// <summary>
    /// Data base user model.
    /// </summary>
    public class DBUserModel : IDBBaseModel
    {
        [PrimaryKey, AutoIncrement]
        public int ID { get; set; }
        public string Name { get; set; }
        public string SerName { get; set; }
        public string Phone { get; set; }
        public string Email { get; set; }
        public string Country { get; set; }
        public string City { get; set; }
        public string UnicalID{ get; set;  }
    }
}

using HappyPet.Interface;

namespace HappyPet.Common
{
    public class CommonAppProperties : IBaseProperties
    {
        public CommonAppProperties()
        {
        }

        public void RegisterProperties()
        {
            App.Current.Properties.Add("startPageTrigger", true);
        }

        public void ReadProperties()
        {

        }
```

```
        public void ChangeProperties()
        {

        }
    }
}


using System.Collections.Generic;

namespace HappyPet.Common
{
    public enum AnimalTypes
    {
        Dogs = 0,
        Cats = 1,
        Birds = 2,
        Snakes = 3,
    }
    public static class RegisterConstants
    {
        public static readonly IReadOnlyList<string> AnimalTypesList = new
List<string>()
        {
            "Собака",
            "Кішка",
            "Птиця",
            "Змія",
        };
        private static readonly IReadOnlyList<string> DogGreed = new List<string>()
        {
            "Вівчарка",
            "Хаскі",
            "Акіта",
            "Пудель",
        };
        private static readonly IReadOnlyList<string> CatGreed = new List<string>()
        {
            "Девон рекс",
            "Британець",
            "Карликовий",
            "Дворовий",
        };
```

```
        private static readonly IReadOnlyList<string> BirdGreed = new List<string>()
        {
            "Попуга",
            "Ворон",
            "Ястреб",
            "Курка",
        };
        private static readonly IReadOnlyList<string> SnakeGreed = new List<string>()
        {
            "Пітон",
            "Удав",
            "Вуж",
            "Гадюка",
        };
        public static readonly IReadOnlyDictionary<AnimalTypes,
IReadOnlyList<string>> AnimalsDictionary = new Dictionary<AnimalTypes,
IReadOnlyList<string>>()
        {
            {AnimalTypes.Dogs, DogGreed },
            {AnimalTypes.Cats, CatGreed },
            {AnimalTypes.Birds, BirdGreed },
            {AnimalTypes.Snakes, SnakeGreed }
        };
    }
}

using HappyPet.Interface;
using HappyPet.Model;
using System;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace HappyPet.UI.MenuFlyoutPage
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class FlyoutPageMenu : FlyoutPage
    {
        private IConnectionModels _connectionModels;
        public FlyoutPageMenu(IConnectionModels models)
        {
            _connectionModels = models;
            InitializeComponent();
            FlyoutPage.ListView.ItemSelected += ListView_ItemSelected;
            FlyoutPage.homeButton.Clicked += OnHomeButtonClicked;
```

```
            var page = new MainMenuPage(_connectionModels);

            Detail = new NavigationPage(page);
            IsPresented = false;
        }

        private void ListView_ItemSelected(object sender, SelectedItemChangedEventArgs
e)
        {
            var item = e.SelectedItem as FlyoutPageMenuFlyoutMenuItem;
            if (item == null)
                return;


            var page = (Page)Activator.CreateInstance(item.TargetType, new object[] {
_connectionModels });
            page.Title = item.Title;

            Detail = new NavigationPage(page);
            IsPresented = false;

            FlyoutPage.ListView.SelectedItem = null;
        }

        private void OnHomeButtonClicked(object sender, EventArgs args)
        {
            var page = new MainMenuPage(_connectionModels);

            Detail = new NavigationPage(page);
            IsPresented = false;

            FlyoutPage.ListView.SelectedItem = null;
        }
    }
}
```

```
<?xml version="1.0" encoding="utf-8" ?>
<FlyoutPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="HappyPet.UI.MenuFlyoutPage.FlyoutPageMenu"
            xmlns:pages="clr-namespace:HappyPet.UI.MenuFlyoutPage"
            xmlns:subpages="clr-namespace:HappyPet.UI">
    <FlyoutPage.Flyout>
        <pages:FlyoutPageMenuFlyout x:Name="FlyoutPage" />
```

```
    </FlyoutPage.Flyout>
</FlyoutPage>

using HappyPet.UI.FunctionalPages;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace HappyPet.UI.MenuFlyoutPage
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class FlyoutPageMenuFlyout : ContentPage
    {
        public ListView ListView;

        public ImageButton homeButton;

        public FlyoutPageMenuFlyout()
        {
            InitializeComponent();

            BindingContext = new FlyoutPageMenuFlyoutViewModel();
            ListView = MenuItemsListView;
            homeButton = HomeButton;
        }

        private async void OnHomeButtonClicked(object sender, EventArgs args)
        {
        }

        class FlyoutPageMenuFlyoutViewModel : INotifyPropertyChanged
        {
            public  ObservableCollection<FlyoutPageMenuFlyoutMenuItem>  MenuItems  {
get; set; }

            public FlyoutPageMenuFlyoutViewModel()
```

```
        {
            MenuItems = new ObservableCollection<FlyoutPageMenuFlyoutMenuItem>(new[]
            {
                new FlyoutPageMenuFlyoutMenuItem { Id = 0, Title = "Здоров'я", TargetType = typeof(HealthPage) },
                new FlyoutPageMenuFlyoutMenuItem { Id = 1, Title = "Сховище", TargetType = typeof(DocumentsPage) },
                new FlyoutPageMenuFlyoutMenuItem { Id = 2, Title = "Їжа", TargetType = typeof(FoodPage) },
                new FlyoutPageMenuFlyoutMenuItem { Id = 3, Title = "Спорт", TargetType = typeof(SportPage) },
                new FlyoutPageMenuFlyoutMenuItem { Id = 4, Title = "Ветклініки", TargetType = typeof(HospitalPage) },
                new FlyoutPageMenuFlyoutMenuItem { Id = 5, Title = "Зоомагазини", TargetType = typeof(ShopPage) },
                new FlyoutPageMenuFlyoutMenuItem { Id = 6, Title = "Доглядальники", TargetType = typeof(PetSitterPage) },
                new FlyoutPageMenuFlyoutMenuItem { Id = 7 , Title = "Профіль", TargetType = typeof(ProfilePage) },
            });
        }

        #region INotifyPropertyChanged Implementation
        public event PropertyChangedEventHandler PropertyChanged;
        void OnPropertyChanged([CallerMemberName] string propertyName = "")
        {
            if (PropertyChanged == null)
                return;

            PropertyChanged.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
        #endregion
    }
  }
}
```

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HappyPet.UI.MenuFlyoutPage.FlyoutPageMenuFlyout"
             Title="Flyout">
  <StackLayout BackgroundColor="#F5F5FF">
```

```
    <ListView x:Name="MenuItemsListView"
              SeparatorVisibility="None"
              HasUnevenRows="true"
              ItemsSource="{Binding MenuItems}">
      <ListView.Header>
        <Grid BackgroundColor="#FFB46B">
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="10"/>
          <ColumnDefinition Width="*"/>
          <ColumnDefinition Width="10"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
          <RowDefinition Height="80"/>
          <RowDefinition Height="Auto"/>
          <RowDefinition Height="0.1*"/>
        </Grid.RowDefinitions>
                <StackLayout Orientation="Horizontal" Grid.Column="1" Grid.Row="0">
                  <ImageButton x:Name="HomeButton" BackgroundColor="Transparent"
Source="res/drawable/logoakita.png"                                    HeightRequest="70"
WidthRequest="70"></ImageButton>
                  <Label
                    VerticalOptions="CenterAndExpand"
                    Text="Happy pet"
                    TextColor="Black" FontAttributes="Bold" FontSize="28"/>
                </StackLayout>
            </Grid>
      </ListView.Header>
      <ListView.ItemTemplate>
        <DataTemplate>
          <ViewCell>
                  <StackLayout   Padding="15,10"   HorizontalOptions="FillAndExpand"
BackgroundColor="#F5F5FF">
            <Label VerticalOptions="FillAndExpand"
                VerticalTextAlignment="Center"
                Text="{Binding Title}"
                FontSize="24" TextColor="Black" FontAttributes="Bold"/>
          </StackLayout>
          </ViewCell>
        </DataTemplate>
      </ListView.ItemTemplate>
    </ListView>
  </StackLayout>
</ContentPage>
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HappyPet.UI.MenuFlyoutPage
{
    public class FlyoutPageMenuFlyoutMenuItem
    {
        public FlyoutPageMenuFlyoutMenuItem()
        {
            TargetType = typeof(FlyoutPageMenuFlyoutMenuItem);
        }
        public int Id { get; set; }
        public string Title { get; set; }

        public Type TargetType { get; set; }
    }
}

using HappyPet.DataBase.Models;
using HappyPet.Interface;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace HappyPet.UI.SubPages
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class HealthInfoSubPage : ContentPage
    {
        private IConnectionModels _connectionModels;
        public HealthInfoSubPage(IConnectionModels models)
        {
            _connectionModels = models;
            InitializeComponent();
        }
```

```
        private bool Validation()
        {
            return true;
        }

        private void AddDataToDb()
        {
            var data = new DBGlobalModel()
            {
                Type = DBGlobalTypesOfData.HealthInfo,
                Title = HealthTitle.Text,
                About = "Type:" + HealthProcedure.SelectedItem?.ToString() + "SubType: " +
HealthProcedureSubType.SelectedItem?.ToString(),
                DatePickerInfo = HealthDatePicker.Date,
                CreateDate = DateTime.Now,
                AdditionalInfo = HealthAdditionalInfo.Text,
                UnicalID = DateTime.Now.ToString() + "#001"
            };
            _connectionModels.GlobalDBModel.AddData(data);
        }

        private async void OnBackButtonClicked(object sender, EventArgs args)
        {
            if(Validation())
            {
                AddDataToDb();
                await Navigation.PopModalAsync();
            }
        }
    }
}

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HappyPet.UI.SubPages.HealthInfoSubPage">
    <ContentPage.Content>
        <ScrollView BackgroundColor="#F5F5FF" HorizontalOptions="FillAndExpand">
            <StackLayout                                    BackgroundColor="Transparent"
HorizontalOptions="FillAndExpand">
                <Entry Keyboard="Text" Placeholder="Title" x:Name="HealthTitle"/>
                <Picker Title="Category" x:Name="HealthProcedure">
                    <Picker.Items>
```

```
                <x:String>1</x:String>
                <x:String>2</x:String>
                <x:String>3</x:String>
                <x:String>4</x:String>
                <x:String>5</x:String>
            </Picker.Items>
        </Picker>
        <Picker Title="Category" x:Name="HealthProcedureSubType"
IsVisible="true">
            <Picker.Items>
                <x:String>1</x:String>
                <x:String>2</x:String>
                <x:String>3</x:String>
                <x:String>4</x:String>
                <x:String>5</x:String>
            </Picker.Items>
        </Picker>
        <DatePicker x:Name="HealthDatePicker">
        </DatePicker>
        <Editor HeightRequest="140" x:Name="HealthAdditionalInfo"/>
        <Button Clicked="OnBackButtonClicked" Text="Back"
        VerticalOptions="EndAndExpand"
        HorizontalOptions="CenterAndExpand" />
      </StackLayout>
    </ScrollView>
  </ContentPage.Content>
</ContentPage>

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="HappyPet.UI.SubPages.HospitalSubPage">
  <ContentPage.Content>
    <StackLayout VerticalOptions="FillAndExpand" BackgroundColor="#F5F5FF">
      <Image/>

      <Label Text="Shop!"
        VerticalOptions="CenterAndExpand"
        HorizontalOptions="CenterAndExpand" />
      <Button Clicked="OnBackButtonClicked" Text="Back"
        VerticalOptions="EndAndExpand"
        HorizontalOptions="CenterAndExpand" />
    </StackLayout>
  </ContentPage.Content>
```

```
</ContentPage>

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace HappyPet.UI.SubPages
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class HospitalSubPage : ContentPage
    {
        public HospitalSubPage()
        {
            InitializeComponent();
        }

        private async void OnBackButtonClicked(object sender, EventArgs args)
        {
            await Navigation.PopModalAsync();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace HappyPet.UI.SubPages
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class MainMenuItemsSubPage : ContentPage
    {
        public MainMenuItemsSubPage()
        {
```

```
            InitializeComponent();
        }
        private async void OnBackButtonClicked(object sender, EventArgs args)
        {
            await Navigation.PopModalAsync();
        }
    }
}
```

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HappyPet.UI.SubPages.MainMenuItemsSubPage">
    <ContentPage.Content>
        <StackLayout VerticalOptions="FillAndExpand" BackgroundColor="#F5F5FF">
            <Image/>

            <Label Text="Shop!"
                VerticalOptions="CenterAndExpand"
                HorizontalOptions="CenterAndExpand" />
            <Button Clicked="OnBackButtonClicked" Text="Back"
                VerticalOptions="EndAndExpand"
                HorizontalOptions="CenterAndExpand" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace HappyPet.UI.SubPages
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class ShopSubPage : ContentPage
    {
        public ShopSubPage()
        {
            InitializeComponent();
```

```
        }

        private async void OnBackButtonClicked(object sender, EventArgs args)
        {
            await Navigation.PopModalAsync();
        }
    }
}
```

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HappyPet.UI.SubPages.ShopSubPage">
    <ContentPage.Content>
        <StackLayout VerticalOptions="FillAndExpand" BackgroundColor="#F5F5FF">
            <Image/>

            <Label Text="Shop!"
                VerticalOptions="CenterAndExpand"
                HorizontalOptions="CenterAndExpand" />
            <Button Clicked="OnBackButtonClicked" Text="Back"
                VerticalOptions="EndAndExpand"
                HorizontalOptions="CenterAndExpand" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

```
using HappyPet.DataBase.Models;
using HappyPet.Items;
using HappyPet.UI.SubPages;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;

namespace HappyPet.UI.ViewModels
{
    public class HealthInfoViewModel
    {
        public ObservableCollection<HealthInfoItem> HealthInfo { get; set; }

        public HealthInfoViewModel(List<DBGlobalModel> data)
        {
            if(data.Any())
            {
```

```
            HealthInfo = new ObservableCollection<HealthInfoItem>();
            foreach (var d in data)
            {
                HealthInfo.Add(new HealthInfoItem()
                {
                    Id = d.ID,
                    Title = d.Title,
                    MainInfo = d.About,
                    AdditionalInfo = d.AdditionalInfo,
                    SubAdditionalInfo = d.SubAdditionalInfo,
                    PickerDate = d.DatePickerInfo,
                    TargetType = typeof(HealthInfoSubPage)
                });
            }
        }
    }
}


using HappyPet.Items;
using HappyPet.UI.SubPages;
using System.Collections.ObjectModel;

namespace HappyPet.UI.ViewModels
{
    public class HospitalInfoViewModel
    {
        public ObservableCollection<HospitalInfoItem> HospitalInfo { get; set; }

        public HospitalInfoViewModel()
        {
            HospitalInfo = new ObservableCollection<HospitalInfoItem>(new[]
            {
                new HospitalInfoItem { Id = 0, Title = "Здоров'я", TargetType =
typeof(HospitalSubPage) },
            });
        }
    }
}


using HappyPet.DataBase.Models;
using HappyPet.Items;
```

```
using HappyPet.UI.SubPages;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;

namespace HappyPet.UI.ViewModels
{
   public class MainMenuAdditionalItemViewModel
   {
      public ObservableCollection<MainMenuInfoItem> MainMenuInfoItems { get; set;
}

      public MainMenuAdditionalItemViewModel()
      {
         MainMenuInfoItems = new ObservableCollection<MainMenuInfoItem>(
            new[]
            {
               new MainMenuInfoItem()
               {
                  Id = 0, Title = "Запись 1", TargetType =
typeof(MainMenuItemsSubPage), MainInfo = "Additional info about event."
               },
               new MainMenuInfoItem()
               {
                  Id = 1, Title = "Запись 2", TargetType =
typeof(MainMenuItemsSubPage), MainInfo = "Additional info about event."
               },
               new MainMenuInfoItem()
               {
                  Id = 2, Title = "Запись 3", TargetType =
typeof(MainMenuItemsSubPage), MainInfo = "Additional info about event."
               },
               new MainMenuInfoItem()
               {
                  Id = 3, Title = "Запись 4", TargetType =
typeof(MainMenuItemsSubPage), MainInfo = "Additional info about event."
               },
            }
            );
      }
   }
}
```

```
using HappyPet.Items;
using HappyPet.UI.SubPages;
using System.Collections.ObjectModel;

namespace HappyPet.UI.ViewModels
{
   public class ShopInfoViewModel
   {
      public ObservableCollection<ShopInfoItem> ShopInfo { get; set; }

      public ShopInfoViewModel()
      {
         ShopInfo = new ObservableCollection<ShopInfoItem>(new[]
         {
               new ShopInfoItem { Id = 0, Title = "Здоров'я", TargetType =
typeof(ShopSubPage) },
            });
      }
   }
}
```

Android workflow

```
using System;

using Android.App;
using Android.Content.PM;
using Android.Runtime;
using Android.OS;

namespace HappyPet.Droid
{
   [Activity(Label = "HappyPet", Icon = "@mipmap/icon", Theme =
"@style/MainTheme", MainLauncher = true, ConfigurationChanges =
ConfigChanges.ScreenSize | ConfigChanges.Orientation | ConfigChanges.UiMode |
ConfigChanges.ScreenLayout | ConfigChanges.SmallestScreenSize )]
   public class MainActivity :
global::Xamarin.Forms.Platform.Android.FormsAppCompatActivity
   {
      protected override void OnCreate(Bundle savedInstanceState)
      {
         base.OnCreate(savedInstanceState);

         Xamarin.Essentials.Platform.Init(this, savedInstanceState);
         global::Xamarin.Forms.Forms.Init(this, savedInstanceState);
```

```
            LoadApplication(new App());
        }
        public override void OnRequestPermissionsResult(int requestCode, string[] permissions, [GeneratedEnum] Android.Content.PM.Permission[] grantResults)
        {
            Xamarin.Essentials.Platform.OnRequestPermissionsResult(requestCode, permissions, grantResults);

            base.OnRequestPermissionsResult(requestCode, permissions, grantResults);
        }
    }
}

using SQLite;
using Xamarin.Forms;
using HappyPet.Droid.DataBases;
using HappyPet.Interface;
using HappyPet.DataBase;

[assembly: Dependency(typeof(SqLiteUser))]
namespace HappyPet.Droid.DataBases
{
    public class SqLiteUser : ISqLiteConnection
    {
        public SQLiteConnection GetConnection(string path)
        {
            var connection = new SQLiteConnection(path);
            return connection;
        }
    }
}
```

ДОДАТОК В

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення для контролю здоров'я домашніх тварин мовою C#

Виконала студентка 4 курсу
групи ПД-41
Смирнова Катерина Борисівна
Керівник роботи
К.т.н, доц, завідувач кафедри ІПЗ Негоденко Олена Василівна

Київ – 2023

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – підвищення ефективності контролю здоров'я домашніх тварин за допомогою мобільного додатку мовою C#.
- **Об'єкт дослідження** – процес контролю здоров'я домашніх тварин.
- **Предмет дослідження** – мобільний додаток для контролю здоров'я домашніх тварин.

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати вже існуючі програмні забезпечення, знайти їх головні переваги та недоліки.
2. Розробити функціональні та нефункціональні вимоги до додатку, грунтуючись на проаналізованих схожих додатках.
3. Дослідити інструменти для розробки додатку, розробити модель архітектури додатку та його дизайну.
4. Розробити додаток на основі обраних інструментів та завчасно визначених вимог.
5. Провести тестування додатку.
6. Пройти апробацію на Науково-технічних конференціях.

3

АНАЛІЗ АНАЛОГІВ

Характеристика	Animal ID	Vitus Vet	PitPat	Happy Pet
Функція «щоденника» для кожного профілю улюбленця для відслідковування важливих події та можливості нагадування	+	-	-	+
Контроль за раціоном тварини	-	-	+	+
Можливість додавання користувачів через email до інформації про вихованця	-	+	+	+
Можливість додавання документів до додатку	+	-	-	+
Пошук ветлікарень за запитами користувача	-	-	-	+
Можливість дізнатися розположення вихованця у реальному часі	-	-	+	-
Можливість додавати важливу інформацію стосовно алергії на профіль тварини	-	-	-	+
Пошук зоомагазинів за запитами користувача	-	-	-	+
Великий список видів тварин (більше ніж пес, кіт, інше)	-	-	-	+

4

ВИМОГИ ДО ДОДАТКУ

1. Реєстрація домашньої тварини.
2. Додавання декількох тварин.
3. Редагування, додавання та оглядання, подій чи нотаток, які створив користувач або додаток із дозволу користувача.
4. Додавання різного типу відомостей про здоров'я тварини.
5. Збереження файлів.
6. Контроль раціону тварини.
7. Наявність переліку ветлікарень.
8. Наявність переліку зоомагазинів.

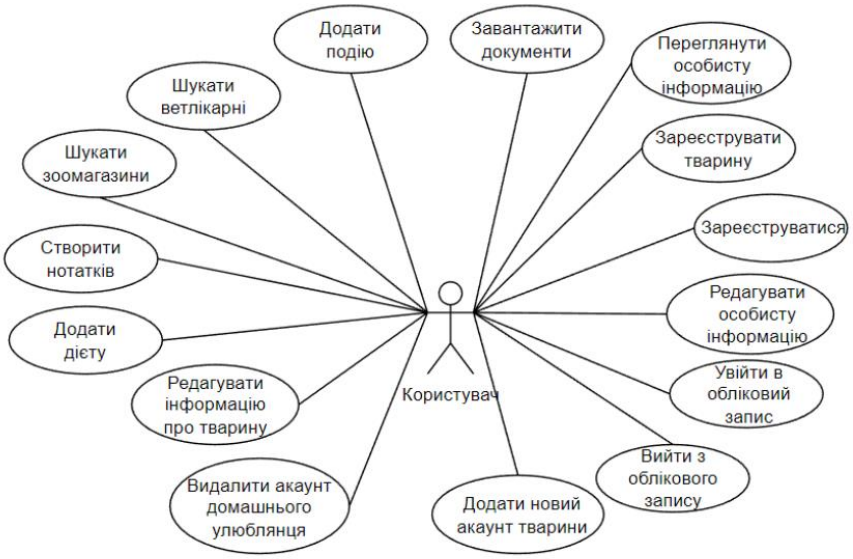
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

MVVM
MODEL VIEW VIEWMODEL

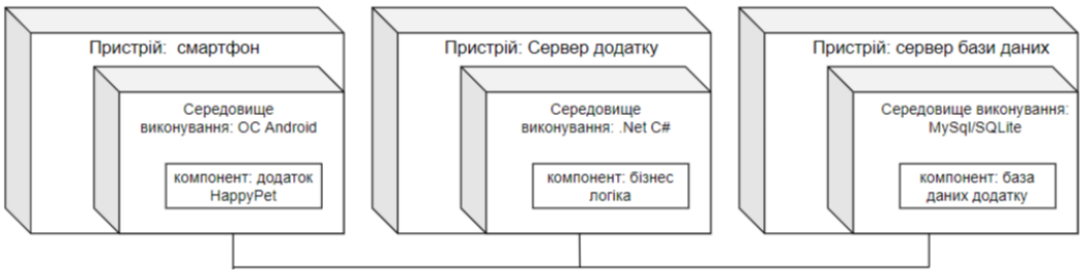
6

ДІАГРАМА ПРЕЦЕДЕНТІВ



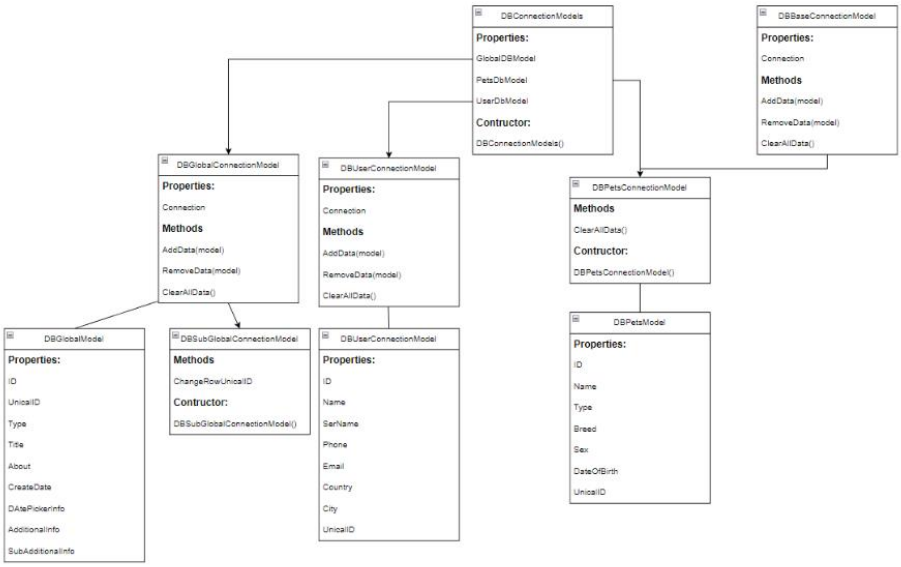
7

ДІАГРАМА РОЗГОРТАННЯ



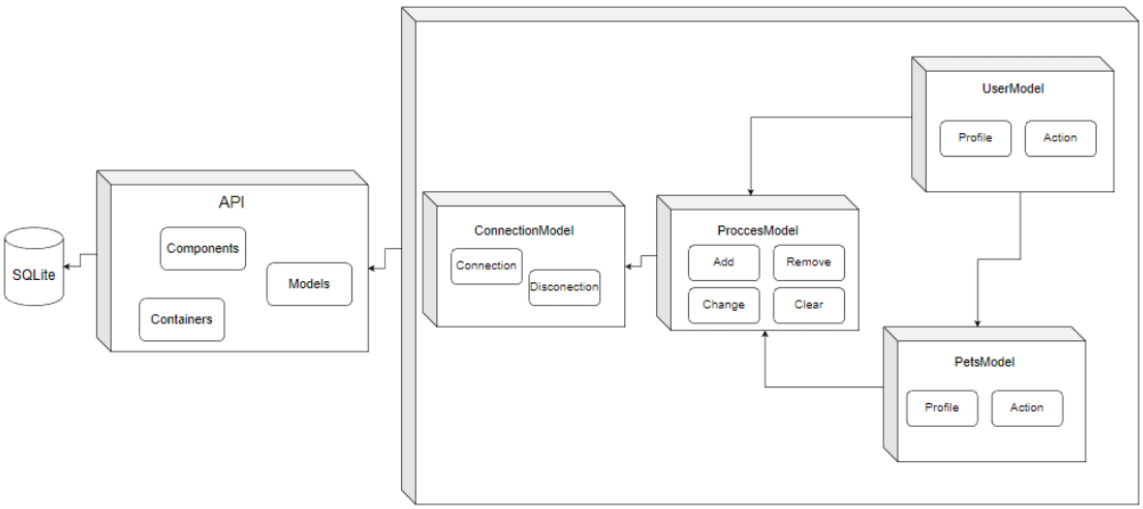
8

ДІАГРАМА КЛАСІВ



9

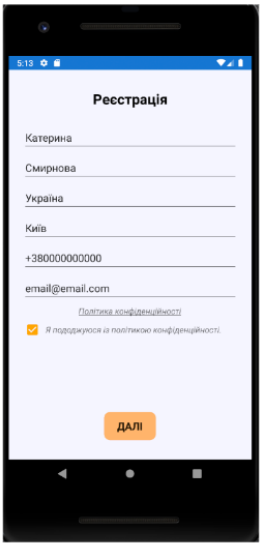
АРХІТЕКТУРА ДОДАТКУ



10

ЕКРАННІ ФОРМИ

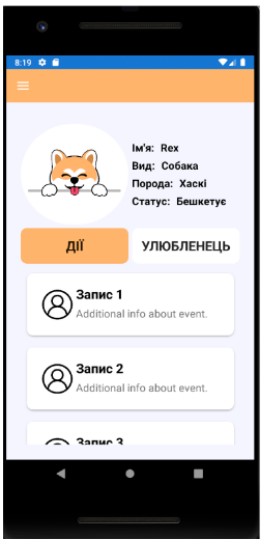
Стартова сторінка



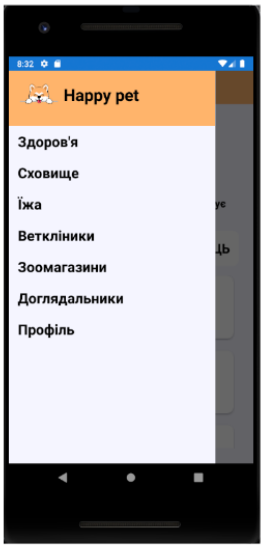
Сторінка реєстрації користувача

11

ЕКРАННІ ФОРМИ



Головна сторінка



Меню додатку

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Смирнова К.Б., Негоденко О.В. Розробка програмного забезпечення для контролю здоров'я домашніх тварин мовою C#. // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інфокомунікаційних технологіях», 20 квітня 2023. Збірник тез. К.: ДУТ, 2023. — С. 33.
2. Смирнова К.Б., Негоденко О.В. Програмне забезпечення для контролю здоров'я домашніх тварин на основі машинного навчання. //III Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 16 травня 2023. Збірник тез. К.: ДУТ, 2023. — Подано до друку.

13

ВИСНОВКИ

1. Проаналізовано предметну область дипломної роботи та виявлено три схожі за функціоналом та головною метою програмного забезпечення, а саме: Animal ID, VitusVet Pet Medical Records та PitPat.
2. Розроблено технічне завдання, визначено функціональні та нефункціональні вимоги щодо розробки програмного забезпечення.
3. Досліджено існуючі інструменти розробки програмного забезпечення та обрано такі засоби реалізації як: мова програмування C#, платформа Xamarin, база даних SQLite, середа розробки Visual Studio, система версій git та паттерн проєктування MVVM.
4. Розроблено мобільний додаток на платформі Xamarin дотримуючись функціональних та нефункціональних вимог та міжнародних вимог на площадках App Store Google Play. Було використано знання із розробки серверної архітектури та архітектури мобільних додатків на основі .NET.
5. Проведено модульне тестування додатку. Перевірено виконання усіх функціональних та нефункціональних вимог.

14

ДЯКУЮ ЗА УВАГУ!