

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ РОЗПІЗНАВАННЯ

ЗАГУБЛЕНИХ ТВАРИН

MOBILE APPLICATION FOR LOST PETS IDENTIFICATION

Виконав(ла): студент(ка) 2 курсу, групи КНТ-132м

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

КОЛЕСНИКОВА М.В.

(прізвище та ініціали)

Керівник ФЕДОРОНЧАК Т.В.

(прізвище та ініціали)

Рецензент ТЕРЕЩЕНКО Е.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти магістр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КОЛЕСНИКОВОЇ Марини Віталіївни
 (ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Мобільний застосунок для розпізнавання загублених тварин. Mobile Application for Lost Pets Identification.

керівник проєкту (роботи) к.т.н., доцент, ФЕДОРОНЧАК Тетяна Василівна,
 (науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)
 затверджені наказом закладу вищої освіти від “14” листопада 2023 року
 № 437

2. Строк подання студентом проєкту (роботи) 01 грудня 2023 року
 3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань роботи. Матеріали та методи. 3. Розробка архітектури програми. 4. Основні рішення щодо реалізації компонентів системи. 5. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1 - 5 Основна частина	ФЕДОРОНЧАК Т.В., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викл.		

7. Дата видачі завдання « 04 » вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз проблеми і предметної області	2 тиждень	Розділ 1
3	Вибір матеріалів, методів та інших технологій розробки.	3 - 4 тижні	Розділ 2
4	Розробка архітектури програми.	5 тиждень	Розділ 3
5	Розробка програми.	6 - 7 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	8 тиждень	Розділ 4, 5
7	Оформлення пояснювальної записки та документів до неї.	9 - 10 тижні	Додатки
8	Нормоконтроль та рецензування.	11 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

Підпис
(підпис)

Марина КОЛЕСНИКОВА
(ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Тетяна ФЕДОРОНЧАК
(ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
126 с., 18 табл., 66 рис., 3 дод., 7 формул, 36 джерел.

КЛАСИФІКАЦІЯ, ДОМАШНЯ ТВАРИНА, МОБІЛЬНИЙ
ЗАСТОСУНОК, FLUTTER, DART, RESNET50, PYTHON

Об'єкт дослідження – розпізнавання тварини з застосуванням штучного інтелекту.

Предмет дослідження – кросплатформний мобільний застосунок для пошуку загублених тварин «PawFinder»

Матеріали, методи та технічні засоби: претренована модель нейронної мережі, об'єктно-орієнтоване програмування, фреймворк Flutter, мова Dart, мова Python, бібліотека Keras, віддалене сховище CloudFirestore, комп'ютер MacBook Pro (2019), ОС «Sonoma 14.1.1».

Розроблено мобільний застосунок для розпізнавання тварини за породою та порівняння вхідних даних з існуючими записами.

Висновки. Розроблений застосунок дозволяє проводити пошук загубленої тварини, або повідомити про знайдену за допомоги смартфона, що може значно прискорити пошуку улюбленця та мінімізувати стрес господаря.

Галузь використання – власники собак, що мають смартфон з ОС Android чи IOS.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 126 pages, 66 figures, 18 tables, 3 appendixes, 7 formulas, 36 sources.

CLASSIFICATION, PET, MOBILE APP, FLUTTER, DART, RESNET50, PYTHON

The object of research is animal recognition using artificial intelligence.

The subject of the research is a cross-platform mobile application for finding lost animals «PawFinder».

Materials, methods and technical means: pretrained neural network model, object-oriented programming, Flutter framework, Dart language, Python language, Keras library, CloudFirestore remote storage, MacBook Pro computer (2019), «Sonoma 14.1.1 OS».

A mobile application has been developed for recognizing an animal by breed and comparing input data with existing records.

Conclusions. The developed application allows you to search for a lost animal, or report a found one using a smartphone, which can significantly speed up the search for a pet and minimize the owner's stress.

Field of use – dog owners who have a smartphone with Android or IOS OS.

ЗМІСТ

С.

Перелік скорочень та умовних позначок	8
Вступ.....	9
1 Аналіз проблеми та постановка завдань роботи.....	10
1.1 Аналіз предметної області	10
1.2 Дослідження існуючих підходів.....	10
1.3 Висновки за розділом 1	33
2 Матеріали та методи	35
2.1 Трансферне навчання з допомоги Keras	35
2.2 Модель Sequential.....	37
2.3 Висновки за розділом 2	37
3 Розробка архітектури програми.....	38
3.1 Опис вимог до програми	38
3.2 Вибір та обґрунтування засобів розробки	39
3.3 Вибір сховища даних для Flutter застосунку.....	46
3.4 Вибір сервісу для сповіщення користувача	47
3.5 Керування станами у Flutter застосунку	48
3.6 Архітектура Flutter застосунку	50
3.7 Висновки за розділом	51
4 Основні рішення щодо реалізації компонентів системи.....	53
4.1 Структура Flutter застосунку	53
4.2 Структура Python проєкту.....	55
4.3 Алгоритм функціонування розробленої програми.....	56
4.4 Алгоритми Python проєкту.....	67
4.5 Налаштування Firebase сервісів.....	73
4.6 Схема даних застосунку	77
4.7 Інтерфейс застосунку.....	80
4.8 Висновки за розділом 4	87
5 Експлуатація, тестування та експериментальне дослідження програм.....	89

5.1 Призначення програми та умови експлуатації	89
5.2 Звернення до програми.....	89
5.3 Тестування	91
5.4 Висновок за п'ятим розділом.....	93
Висновок	94
Перелік джерел посилання	96
Додаток А Технічне завдання	100
Додаток Б Текст програми	105
Додаток В Слайди презентації.....	114

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CPU	– Central Processing Unit;
CNN	– Convolutional Neural Networks;
GPU	– Graphics Processing Unit;
ID	– Identification;
IDE	– Integrated Development Environment;
IOS	– iPhone Operating System;
JIT	– Just-In-Time;
KNN	– K-Nearest Neighbors Algorithm;
FCM	– Firebase Cloud Messaging;
ML	– Machine Learning;
NFC	– Near Field Communication;
OS	– Operating System;
QR	– Quick Response Code;
ReLU	– Rectified Linear Unit;
SDK	– Software Development Kit;
SVM	– Support Vector Machine;
VS	– Visual Studio;
БД	– база даних;
ОС	– операційна система;
ІІ	– штучний інтелект;
ІІНМ	– штучні нейронні мережі.

ВСТУП

У світі, де домашні тварини стали улюбленими членами незліченних домогосподарств, зв'язок між людьми та їхніми чотирилапими супутниками беззаперечний. Значення домашніх тварин у нашому житті важко переоцінити. Для власників домашніх тварин емоційний зв'язок є глибоким, даруючи комфорт, любов і мовчазне розуміння, яке перевершує будь-які слова.

Незважаючи на радість, яку приносять домашні тварини, сумна реальність полягає в тому, що іноді вони зникають. Особливо зараз, коли тварини лякаються вибухів та гучних звуків ця проблема поширюється та завдає проблем та переживань власникам домашніх тварин. Думка про зникнення домашнього улюбленця — страшний сон, з яким власники сподіваються ніколи не зіткнутися, але на жаль це трапляється.

Проте на цьому виклик не закінчується. Постає проблема пошуку загубленої тварини, що може бути важким і емоційно виснажливим. Розклеювання оголошень та дописи в мережі часто виявляються неефективними, залишаючи власників домашніх тварин у стані постійної тривоги та невпевненості. Перебуваючи на вулиці домашні тварини дуже швидко перестають виглядати доглянутими та більше нагадують безхатніх, що ускладнює впізнання. Іншою проблемою є розповсюдження особистих номерів власників, якими можуть скористатися шахраї.

Актуальним стає створення мобільного застосунку, який би міг розпізнати тварину та зіставити її з загубленою, а також надати контакти власника. Програма має на меті спростити та прискорити процес возз'єднання загублених собак із їх сім'єю.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ РОБОТИ

1.1 Аналіз предметної області

Мобільний застосунок буде присвячений пошуку тварин за фотографіями завантаженими у застосунок користувачем та опрацювання їх алгоритмами штучного інтелекту для знаходження найбільш імовірного співпадіння з фотографіями, що робив інший користувач.

У предметній області будуть фігуруватимуть такі поняття як: штучний інтелект, штучні нейронні мережі, розпізнавання облич, а також поняття, що стосуються роботи з застосунком: домашня тварина, власник та профайл.

Штучний інтелект (ШІ) – область комп'ютерних наук, що зосереджується на створенні комп'ютерних систем, які виконують завдання притаманні людському інтелекту.

Штучні нейронні мережі (ШНМ) – математичні моделі, а також їх програмні або апаратні реалізації, побудовані за принципами подання й обробки інформації у біологічних нейронних мережах – мережах нервових кліток живого організму [1].

Домашня тварина – тварина, що знаходиться під опікою певної людини.

Власник – людина, що має у власності тварину.

Профайл – сторінка з детальною інформацією про одну чи декількох тварин, а також фото.

Основною цільовою аудиторією застосунку є власники домашніх тварин та працівники притулків.

1.2 Дослідження існуючих підходів

Враховуючи те, що завдання по розпізнаванню тварини є нетривіальним і відрізняється від розпізнавання людини, то необхідним є розглянути підходи як розпізнавання так і класифікації.

1.2.1 Розпізнавання облич

Розпізнавання облич (англ. face recognition) – це процес ідентифікації людини з фото чи відеозапису за допомоги аналізу зображення за допомоги штучної нейронної мережі. Варто зазначити, що розпізнавання досягається поєднанням комп'ютерного зору, машинного навчання та методами розпізнавання образів [2].

Для оцінки та опрацювання технологія використовує алгоритми, що аналізують біометричні дані людини, такі як: загальний контур обличчя, розташування очей, носа, рота та їх пропорції. Ці дані перетворюються у цифровий формат, що являє собою унікальний цифровий код, який іменують відбитком обличчя (англ. faceprint). Зіставлення цих кодів з кодами відомих облич і дозволяє ідентифікувати людину.

Як вже зазначалося вище, то завдання розпізнавання облича не обмежується лише виявленням рис обличчя. Для успішного розпізнавання спочатку необхідно виявити обличчя – алгоритм ідентифікує та знаходить потенційні обличчя в кадрі зображення чи відео. Після цього вирівняти його. Тобто після виявлення обличчя система може виконати вирівнювання обличчя. Цей крок передбачає нормалізацію положення, повороту та масштабування обличчя для створення узгодженої еталонної системи для подальшого аналізу [3]. Це гарантує, що обличчя знаходиться в стандартному положенні для розпізнавання.

Наступним є крок виділення ознак: на цьому етапі система виділяє відмінні риси, зазвичай зосереджуючись на ключових областях, таких як очі, ніс, рот і контури обличчя. Цей етап є важливим для створення відбитку обличчя, а також, він скорочує набір початкових даних у процесах опрацювання зображення.

Останнім етапом є безпосередньо розпізнавання. Після процесу виділення ознак вони порівнюються з вже відомими відбитками обличчя в базі даних, використовуючи різні показники подібності, наприклад евклідову

відстань або косинус подібності. Ідентифікація людини системою відбувається завдяки високим показникам подібності відбитку на виявлені риси.

Отже, після проходження всіх кроків, система приймає рішення про прийнятність результатів та видає результат. Процес розпізнавання зображено на рис. 1.1.

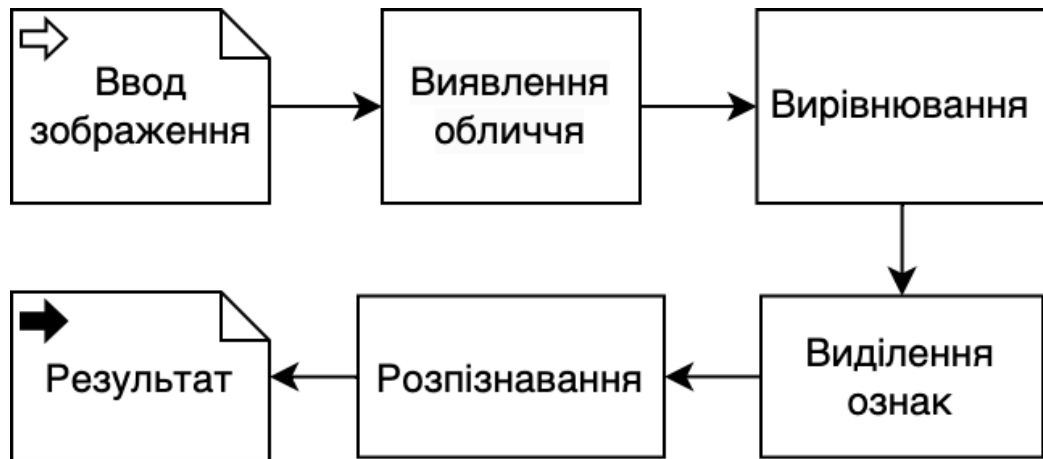


Рисунок 1.1 – Процес розпізнавання обличчя

1.2.1.1 Методи виявлення облич

Задача виявлення обличчя полягає в тому, щоб, маючи довільне зображення, визначити, чи є на ньому обличчя і, якщо є, повернути розміри та місцезнаходження. Проте при виявленні можуть впливати такі фактори як: поза, вираз обличчя, перекриття іншими предметами, наявність окулярів чи бороди у людини чи якість зображення можуть знатно ускладнити процес виявлення.

1.2.1.1.1 Метод Віоли–Джонсона

Метод Віоли–Джонсона засновано на інтегральному представленні зображення, завдяки чому можна швидко виявляти об'єкти. Також використовуються ознаки Хаара з допомоги яких можна виявити необхідний об'єкт. Кожна з ознак складається з суміжних прямокутників, що є фільтрами,

які застосовуються до зображення, щоб отримати дані про інтенсивність яскравості в певних зонах.

Алгоритм на основі ознак Хаара можна пояснити наступним чином: якщо потрібно знайти на зображенні якийсь об'єкт, то на зображення накладається фільтр (рис. 1.2), що асоціюється з ним, переміщується по вісям X та Y та обраховується різниця між яскравістю пікселів та фільтра. Різниця обчислюється за формулою 1.1:

$$\Delta = dark - white = \frac{1}{n} \sum_{dark}^n I(x) - \frac{1}{y} \sum_{white}^y I(x), \quad (1.1)$$

де n – кількість пікселів, що потрапили у темну область і є темними;

y – кількість пікселів, що потрапили у світлу область і є світлими.



Рисунок 1.2 – Ознака контуру Хаара

Значення функції обчислюється як одне число: сума значень пікселів у чорній області мінус сума значень пікселів у білій області. Якщо функція ближче до 1, то область вважається як та, що ймовірно містить об'єкт, що шукався.

Метод Віоли–Джонса також використовує метод покращення ефективності AdaBoost, що є алгоритмом машинного навчання та поєднує прогнози декількох слабких класифікаторів в один і знову опрацьовує ті дані, що не мали стовідсоткові результати. А також каскадні класифікатори – це алгоритми, що аналізують зображення, що попередньо розділені на певні

регіони і намагаються визначити чи зображене на них обличчя чи ні. Це робиться, щоб відкинути зони без лиця і не витратити час на обчислення. Під каскадністю мають на увазі, що метод спочатку застосовує найкращі ознаки для визначення, наприклад наявність ока чи рта, і якщо така ймовірність є, то зображення відправляється до наступної перевірки за більш неявною ознакою. Якщо всі каскади відповідають позитивно, то зображення одностайно кваліфікується як те, що містить людське обличчя [4]. І хоча цей алгоритм було винайдено більше 20 років тому, він досі має застосування через свою швидкодію, та високу ефективність. Проте навчання моделі відбувається дуже довго.

1.2.1.1.2 Методи на основі знань

Методи на основі знань ґрунтуються на зводі правил про те, як виглядає обличчя людини. Вони використовують попередньо визначені правила для опису рис, шаблони та моделі виходячи зі знань про обличчя. Здебільшого правила описують співвідношення між рисами та їх положення.

Складність застосування полягає у необхідності створення досить великої кількості директив. До того ж вони мають бути не дуже загальні чи детальні, можливі хибно–позитивні чи хибно–негативні результати [5]. Також можливі похибки при різному освітленні. Підходить для пошуку локалізації обличчя. Методи, засновані на знаннях можна віднести до підходу згори–вниз. Тобто визначення обличчя починається від загальних до конкретних правил.

Проблема такого підходу полягає в тому, що досить важким завданням є переведення людських знань у конкретні правила.

1.2.1.1.3 Методи на основі ознак

Методи на основі ознак, спрямовані знаходити структурні особливості обличчя. Вони навчаються як класифікатори і використовуються для пошуку

зон на яких наявне або відсутнє обличчя. Через це такі методи більше підходять для локалізації обличчя. Ці методи можна віднести до підходу знизу–вгору (англ. bottom–up). Тобто визначення йде від деталей до загального. Вони фокусуються на виявленні різноманітних рис обличчя, не базуючись на попередньо визначених правилах. Рот, ніс, очі, брови або лінія росту волосся зазвичай виділяються за допомогою детекторів країв. На основі виділених ознак будується статистична модель для опису їх зв'язків і перевірки існування обличчя. Одним з прикладів такого методу є розпізнавання облич по кольору, яке сканує кольорові зображення або відео на предмет областей із типовим кольором шкіри, а потім шукає сегменти обличчя [5].

Проблемою таких алгоритмів, що засновані на функціях є те, що межі, які виділяють ознаки на обличчі, можуть бути менш виразними через деякі фактори (освітлення, шум), в той час як тіні можуть створювати багато сильних граней, що можуть ускладнювати роботу алгоритмів, призначених для сприйняття облич.

Вибір між підходами згори–вниз та знизу–вгору залежить від конкретних вимог до завдання виявлення обличчя та мінливості даних. На практиці багато сучасних систем виявлення обличчя використовують комбінацію методів згори–вниз та знизу–вгору для покращення точності та надійності [6]. До методів на основі ознак відноситься згаданий вище метод ознак Хаара.

1.2.1.1.4 Методи порівняння з шаблоном

Методи порівняння з шаблоном використовують заздалегідь визначені або змінюванні шаблони для виявлення обличчя, шляхом пошуку кореляції між ними та зображенням на якому відбувається виявлення. Для заданого зображення використовуються кілька підшаблонів для контуру обличчя, очей, носа та рота і значення кореляції для них обчислюються окремо зі

стандартними шаблонами обличчя або його рис. Цей підхід може бути неефективним для виявлення обличчя при зміні освітлення, пози або масштабу.

Методи порівняння з шаблоном діляться на підходи, що використовують попередньо визначені шаблони та ті що деформуються (змінюванні). Перший підхід ділиться на дві фази. На першому етапі виділяються контури обличчя та визначаються потенційні ділянки для другої фази. На другому етапі ці ділянки детально досліджуються для визначення наявності обличчя. Недоліком цього підходу є обмеженість поз, тобто обличчя має знаходитися фронтально.

У змінюваних шаблонах риси обличчя описані параметризованими функціями. Вони базуються на апріорних знаннях про риси та їх просторові відносини під час етапу виявлення [7]. Проектування такого шаблону для виявлення ознак обличчя складається з двох кроків. Спочатку, шаблону надається геометрична модель, а потім відбувається специфікування моделі, відомої як модель зображення, для пояснення того, як шаблон відповідає конкретним даним. Зміни в параметрах відповідають допустимим деформаціям об'єкта і можуть бути визначені за допомогою ймовірнісної моделі. Виділені параметри змінюванного шаблону використовуються для виявлення обличчя.

Для кращої наглядності наведено зображення моделі обличчя (рис. 1.3), що використовується у підході Фішлера та Ельшлагера (1973). Обличчя змодельовано зображенням набору основних рис, що з'єднані пружинами. До набору рис входять лінія волосся, очі, ніс, рот та контури обличчя. Вони не деформуються і кожна з них має локальний показник відповідності до зображення. Під час узгодження з зображенням конструкція деформується до тих пір поки всі ознаки не будуть мати високий показник відповідності, а сили пружин збалансовані. Пружини необхідні для забезпечення адекватного просторового відношення між рисами, тобто губи знаходяться нижче ніж ніс, а ніс нижче очей.

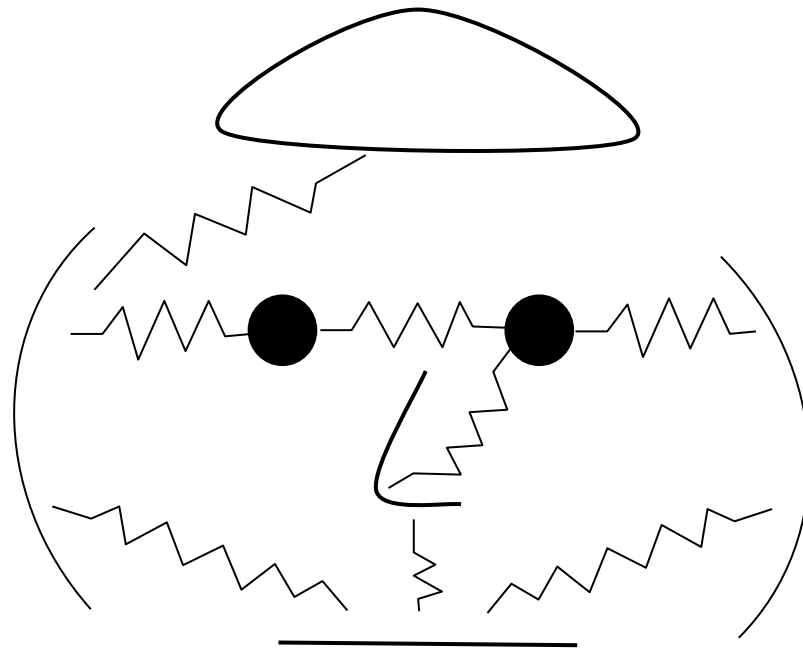


Рисунок 1.3 – Модель обличчя у підході Фішлера та Ельшлагера

Недоліком методів на основі шаблонів, що деформуються є їх «несамостійність». Перед їх застосуванням, потрібно надати початкову оцінку того, де на зображенні знаходиться обличчя чи об'єкт. До того ж процеси обчислення деформації та оптимізації обчислювально складні.

1.2.1.2 Методи розпізнавання облич

Методи розпізнавання облич є більш складними ніж виявлення і виконуються завдання не лише детекції обличчя, а і ідентифікації.

Для того, щоб визначати наявність обличчя на фото чи відео, такі методи базуються на статистичному аналізі та машинному навчанні. Існують декілька підходів, наприклад: методи на основі зовнішнього вигляду, на основі геометрії, засновані на статистичному аналізі чи на нейронних мережах.

Прикладами цих підходів є:

- Eigenface (нім. власне обличчя) метод;
- методи на основі розподілу;
- метод триплетних втрат (FaceNet).

1.2.1.2.1 Eigenface метод

В методах на основі зовнішнього вигляду обличчя є вхідними даними системи, воно розглядається як багатовимірний вектор. Концепція ознаки відрізняється від попередньої, і тепер є не тільки рисою обличчя, такою як, наприклад, очі чи ніс, а являється будь-якою характеристикою витягнутою з зображення. Вивчені ознаки виражені у формі моделей розподілу або дискримінантних функцій, які, в подальшому використовуються для виявлення обличчя.

Метод Eigenface має за мету представити зображення обличчя людини, як лінійну комбінацію з набору базових зображень, що називаються eigenfaces. Якщо припустити, що всі зображення обличчя мають певну однакову розмірність $W \times H$ і знаходяться у множині $R^{W \times H}$, то будь яке зображення обличчя може бути представлене лінійною комбінацією, за умови, що буде знайдено оптимальне eigenfaces [8].

Для того, щоб навчити модель необхідно створити вибірку попередньо оброблених зображень. Зазвичай це вирівнювання очей, носа, рота, нормалізація яскравості та позиції зображення. Завдяки цьому алгоритм зосереджується на зонах, пов'язаних із зовнішністю і може ігнорувати інші нерелевантні частини обличчя. Після цього зображення перетворюються в набір векторів ознак, що кодують візуальну інформацію. Матриця коваріації цих векторів відповідає кореляції між різними характеристиками обличчя у наборі даних. Обчислюються власні вектори коваріаційної матриці, які охоплюють прихований простір змін обличчя. Потім ці власні вектори будуть використані як базові зображення eigenfaces нового тестового зображення обличчя.

Тобто завдання алгоритму це інтерпретувати зображення як сумму базисних компонент — зображень. Приклад зображення, що складається з лінійної комбінації eigenfaces (рис 1.4).



Рисунок 1.4 – Зображення, що складається з eigenfaces [9]

1.2.1.2.2 Метод на основі розподілу

Основна ідея методів на основі розподілу полягає у доведенні того, що розподіли зображень об'єктів одного класу можуть бути навчені з позитивних та негативних прикладів цього об'єкта. Однак якщо створення репрезентативної вибірки позитивного набору прикладів, тобто зображень людських облич, не проблемою, то з негативними прикладами є проблема. Вона виникає через те, що можливі помилкові визначення на початкових етапах навчання. Це вирішується методом початкового навантаження, що додає зображення до навчальної вибірки поступово під час тренування.

Починаючи з невеликого набору негативних прикладів у навчальному наборі, класифікатор багат шарового персептрона навчається за допомогою цієї бази даних прикладів [6]. Після цього навченому детектору надається велика кількість випадкових зображень і всі, що були хибно визнані позитивними, додаються до навчальної бази у якості негативних шаблонів.

До методів на основі розподілу належать змішана Гаусова модель та ядерна оцінка щільності (англ. Kernel Density Estimation).

1.2.1.2.3 Метод триплетних втрат

Визначальною частиною методу ідентифікації обличчя на основі глибокого навчання є метод триплетних втрати. Щоб навчити мережу з втратою триплетів, потрібно визначити триплети зображень. Триплет складається з якірного зображення x_a , позитивного зображення x_p і негативного зображення x_n .

Якірне зображення – це зображення випадково обраної особи в наборі даних. Позитивне зображення – це інше зображення того самого об'єкта. Негативне зображення – це картина іншого об'єкта. Для зображення x мережа f створює вектор вкладеності $f(x)$. Вектор вкладеності це спосіб перетворення виділених властивостей у числа, які перетворюються на точки у багатовимірному просторі. Схожі дані будуть у просторі точками, що ближче одна до одної. Ілюстрація цього методу наведена на рис. 1.5.

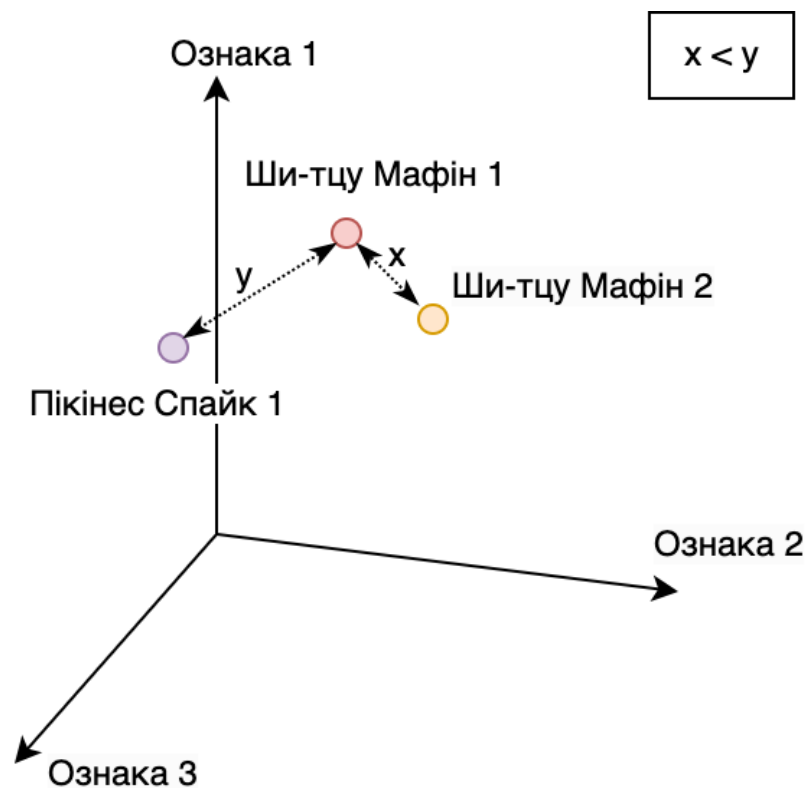


Рисунок 1.5 – Ілюстрація методу втрати триплетів, де x та y відстані між векторами

Мета моделі полягає в тому, щоб переконатися, що евклідова відстань між якірним вектором вкладеності $f(x_a)$ і позитивним зображенням $f(x_p)$ менша, ніж евклідова відстань між якірним вектором вкладеності та негативним зображенням $f(x_n)$. Щоб збільшити маржу між різними класами, до попередньої нерівності додається константа a (формула 1.2) [10].

$$\|f(x_a) - f(x_p)\|^2 + a < \|f(x_a) - f(x_n)\|^2 \quad (1.2)$$

Кінцевою метою мережі є мінімізація наступних втрат L , визначених як втрата триплетів (формула 1.3).

$$L = \sum_{i=0}^N \max \left(\|f(x_a^{(i)}) - f(x_p^{(i)})\|^2 - \|f(x_a^{(i)}) - f(x_n^{(i)})\|^2 + a, 0 \right), \quad (1.3)$$

де N – кількість триплетів на партію.

З застосуванням цього методу тренуються нейронні мережі для розпізнавання облич: FaceNet, SiameseNetwork, EfficientNet.

1.2.2 Класифікація

Класифікація відноситься до процесу розподілу набору даних за певними класами або групами. Алгоритмам класифікації потрібні дані для навчання з багатьма входами та їхніми відповідними виходами, за допомогою яких навчається модель. Важливо розуміти, що навчальні дані повинні включати всі типи даних (параметрів), які можна зустріти в тестовому наборі даних або в реальному світі.

Алгоритм класифікації, заснований на навчальних даних, є методикою навчання під наглядом, яка використовується для класифікації нових спостережень. У класифікації програма використовує набір даних або надані

спостереження, щоб навчитися класифікувати нові спостереження за різними класами чи групами. Для опису класів використовують цілі, мітки чи категорії. Алгоритм класифікації використовує позначені вхідні дані, оскільки це метод навчання під наглядом і містить вхідну та вихідну інформацію. Дискретна вихідна функція (y) передається вхідній змінній у процесі класифікації (x) [11].

За типом навчання класифікації поділяються на Lazy learning та Eager learning.

Lazy learning – спочатку зберігає навчальний набір даних, перш ніж чекає надходження тестового набору даних. При використанні ледачого навчання класифікація виконується з використанням найбільш відповідних даних навчального набору. Менше часу витрачається на навчання, але більше потребує часу на прогнози. Прикладом може виступати алгоритм k -найближчих сусідів.

Eager learning перш ніж отримати тестовий набір даних, будує модель класифікації за допомогою навчального набору даних. Такий підхід витрачає більше часу на вивчення і менше часу на передбачення. Як приклад може виступати ШНМ, наївні байєсівські дерева та дерева рішень [12].

1.2.2.1 Типи класифікації

Класифікація розподіляється на чотири типи, а саме:

- бінарна класифікація;
- багатокласова класифікація;
- класифікація з кількома мітками;
- незбалансована класифікація.

Бінарна класифікація – класифікація завданням якої є розподіл на два класи, тобто значення, отримане після класифікації даних, буде 0 або 1.

Зазвичай завдання бінарної класифікації моделюють за допомогою моделі, яка передбачає розподіл ймовірностей Бернуллі. Розподіл Бернуллі - є

дискретним розподілом ймовірностей. Він охоплює випадок, коли подія матиме двійковий результат як 0 або 1 [13].

Алгоритми, які використовуються для виконання двійкової класифікації: логістична регресія, дерева рішень, метод опорних векторів, наївний Байєс, knn (k найближчих сусідів).

Багатокласова класифікація – тип класифікації, при якому вхідний набір даних класифікується/позначається більш ніж у 2 класи.

Відрізняється від двійкової класифікації тим, що вона має не лише два класи, як—от 0 чи 1, а більше, і вони не обов’язково повинні бути 0 чи 1. Це можуть бути імена чи інші неперервні чи переривчасті числа. Точки даних класифікуються в один клас серед багатьох наданих інших.

Звичайним є моделювання багатокласової класифікаційної задачі за допомогою моделі, яка передбачає розподіл ймовірностей Мультинуллі для кожного прикладу. Розподіл Мультинуллі – це дискретний розподіл ймовірностей, що охоплює випадок, коли подія матиме категоричний результат, наприклад K у $\{1, 2, 3, \dots, K\}$. Для класифікації це означає, що модель передбачає ймовірність приналежності прикладу до кожної мітки класу [14].

Класифікація з кількома мітками – відноситься до проблем класифікації, які стосуються більш ніж одного класу, призначеного одній точці даних, тобто кожна точка даних може бути позначена більш ніж одним класом (міткою). Простим прикладом може бути фотографія, на якій зображено кілька людей, а не один. Відрізняється від двійкової та багатокласової класифікації, оскільки кількість міток, за якими класифікується одна точка даних, залишається незмінною, тобто одна.

Деякі алгоритми класифікації з кількома мітками включають: випадкові ліси з кількома мітками, посилення градієнта з кількома мітками, код для демонстрації класифікації з кількома мітками :

Незбалансована класифікація – тип класифікації, у якому кількість точок даних у наборі даних у кожному класі розподіляється нерівномірно. Це

означає, що незбалансована класифікація в основному є проблемою бінарної класифікації, яка не має рівномірного розподілу точок, один клас може містити надзвичайно велику кількість точок даних, а інший клас може містити дуже малу кількість точок даних [15].

Існують спеціальні алгоритми, які використовуються для класифікації таких даних у велику групу точок даних або групу малих точок даних:

- дерева рішень, чутливі до витрат;
- логістична регресія з урахуванням витрат;
- машини опорних векторів, чутливі до вартості.

1.2.2.1.1 Наївний класифікатор Байєса

Наївний класифікатор Байєса базується на теоремі Байєса – це спосіб обчислення ймовірності гіпотези (ситуації, яка могла б не відбутися в реальності) на основі нашого попереднього досвіду та знань, які ми отримали завдяки цьому.

Теорема Байєса виражено у формулі 1.4:

$$P_{(hypo|data)} = (P_{(data|hypo)} \cdot P_{(hypo)}) / P_{(data)}, \quad (1.4)$$

де $P_{(hypo | data)}$ – ймовірність гіпотези, коли надаються дані (так звана апостеріорна ймовірність);

$P_{(data | hypo)}$ - ймовірність даних, коли відомо, що конкретна гіпотеза вірна;

$P_{(hypo)}$ - ймовірність того, що гіпотеза є істинною незалежно від наявних даних (попередня ймовірність);

$P_{(data)}$ - це ймовірність даних (незалежно від гіпотези) [16].

Сутність полягає в отриманні значень апостеріорної ймовірності за іншими даними. Знаходиться апостеріорна ймовірність для ряду різних гіпотез і вибирається ймовірність, яка має найбільше значення (гіпотеза максимальної ймовірності або максимального апостериору – МАП (формула 1.5).

$$MAP_{(hypo)} = \max(P_{(hypo | data)}), \quad (1.5)$$

Якщо значення $P_{(hypo | data)}$ замінити значенням, даними з формули вище то результат можна виразити формулою 1.6:

$$MAP_{(hypo)} = \max \left((P_{(hypo | data)} \cdot P_{(hypo)}) / P_{(data)} \right), \quad (1.6)$$

$P_{(data)}$ розглядається як нормалізуючий що допомагає визначити ймовірність. Це значення можна ігнорувати, якщо потрібно, оскільки воно є постійним.

1.2.2.1.2 Класифікація к-найближчий сусідів

Класифікація к-найближчий сусідів (KNN) – полягає в тому що він не має іншої моделі, окрім тієї, яка зберігає весь набір даних, що означає, що машинне навчання не відбувається. Це означає, що KNN робить прогнози та витягує шаблони безпосередньо з самого набору навчальних даних.

Коли зустрічається нова точка даних, відповідне значення для неї можна знайти за допомогою KNN, переміщаючись по всьому навчальному набору даних, дивлячись на кількість «k» дуже схожих сусідів. Після ідентифікації «k» сусідів вони підсумовуються та виводиться результат для кожного екземпляра. У разі регресії середнє значення цього виходу є результатом, а у випадку класифікації мода цього виходу є результатом [17].

Щоб знайти «k» кількість екземплярів із навчального набору даних, які дуже схожі на нову точку даних, використовується коефіцієнт відстані, а найпопулярнішою метрикою є евклідова відстань.

Евклідову відстань можна визначити, знайшовши квадратний корінь із суми квадратів різниці між новою точкою та існуючою точкою в наборі даних, і ця сума складається зі значень у діапазоні (a, b) .

Можна використовувати інші відстані, як то: відстань хеммінга, манхеттенська відстань, відстань Мінковського.

Коли кількість точок даних у навчальному наборі збільшується, складність KNN також зростає.

1.2.2.1.3 Метод опорних векторів (SVM)

Метод опорних векторів (SVM) – полягає в тому що гіперплощина, присутня в лінійній МОВ, вивчається шляхом виконання простих перетворень за допомогою лінійної алгебри. Сума добутку кожної пари точок вхідних даних множиться, і це називається внутрішнім добутком. Основна ідея SVM полягає в тому, що скалярний добуток двох векторів можна виразити як суму добутку першого значення кожного вектора.

Щоб знайти внутрішній добуток двох вхідних векторів: $[a, b]$ і $[c, d]$, розраховують $[a * c + b * d]$.

Для передбачення нового значення, можна використати скалярний добуток, а опорний вектор можна розрахувати за допомогою рівняння, яке зображене у формулі 1.7:

$$f_{(x)} = coeff_1 + sum(coeff_2 \cdot (a, b)), \quad (1.7)$$

де a, b – вхідні вектори;

$coeff_1$ і $coeff_2$ – це коефіцієнти, які визначаються за допомогою навчального набору даних і алгоритму навчання [18].

Можна використовувати стохастичний градієнтний спуск або послідовну техніку мінімальної оптимізації. Ці методи оптимізації розбивають

основну проблему на підпроблеми, і кожна підпроблема вирішується шляхом обчислення необхідного значення.

1.2.2.1.4 Древа рішень

Древа рішень – класифікація яку можна візуалізувати як бінарне дерево, яке має корінь і багато гілок від нього та листя. Корінь – це одне вхідне значення, а гілки, які ведуть до листя, використовуються для прогнозування значень для даного вхідного.

Деревоподібна структура може зберігатися у вигляді графової структури або набору правил. Після того, як дані у формі дерева доступні, можна легко робити прогнози на них за допомогою листових вузлів. Конкретна гілка та її листовий вузол перевіряються, щоб досягти вузла.

Дані відфільтровуються з кореня дерева та надходять і зберігаються в гілці та листі, які мають відношення до нього.

1.2.2.1.5 Нейронні мережі

Ідея залучання штучної нейронної мережі полягає в тому, що вона приймає рішення подібно до того, як функціонують нейрони в мозку під час виконання функції або прийняття рішення.

Це називається глибоким навчанням, оскільки мережі мають різні рівні, і кожен рівень має велику кількість вузлів. Кожен рівень обробляє частину даних і передає обчислені дані на наступний рівень. Вхідними даними для одного шару є вихідні дані попереднього шару. Зазвичай кількість вузлів вхідного рівня є великою, а вихідний рівень має лише один вузол, що вказує на те, що дані були оброблені та отримано вихідні дані.

Безумовно вибір типу класифікації залежить від конкретної задачі, але використання нейромереж є одним з перспективних напрямів, який отримав відчутний розвиток в останньому десятилітті [19].

1.2.3 Порівняння підходів класифікації та розпізнавання

З метою досягання високої точності розпізнавання чи класифікації зображень, алгоритми штучного інтелекту значною мірою покладаються на методи глибокого навчання, зокрема на згорткові нейронні мережі (CNN). Ці мережі створені для автоматичного вивчення та вилучення значущих функцій із зображень, що дозволяє алгоритмам приймати обґрунтовані рішення щодо вмісту зображення. Розпізнавання і класифікація образів мають певні пов'язані між собою риси, але також і значні відмінності, які роблять їх спрямованими на вирішення різних задач. Розпізнавання зображень зосереджено на ідентифікації та пошуку конкретних об'єктів або візерунків на зображенні, тоді як класифікація зображень відносить зображення до категорії на основі його вмісту. За своєю суттю розпізнавання зображень полягає у виявленні об'єктів, а класифікація зображень - у категоризації зображень.

Розпізнавання зображень підходить для додатків, які потребують ідентифікації та локалізації об'єктів, таких як автономні транспортні засоби, системи безпеки та розпізнавання облич. Однак класифікація зображень більше підходить для завдань, які передбачають сортування зображень за категоріями, як-от упорядкування фотографій, визначення належності об'єкту до групи як то до певної раси, діагностика захворювань із зображень.

Розпізнавання зображень відбувається складніше, ніж їх класифікація потребує більше часу для обробки та вищі вимоги до обчислень. Оскільки класифікація зображень зосереджується виключно на розподілі зображень за категоріями, то розподіл відбувається швидше. Порівняльний аналіз класифікації та розпізнавання наведено в табл. 1.1 [20].

Таблиця 1.1 – Порівняльний аналіз класифікації та розпізнавання

Критерій	Розпізнавання	Класифікація
1	2	3
Функції	Ідентифікація об'єктів та локацій	Сортування об'єктів за категоріями

Продовження таблиці 1.1

1	2	3
Складність	Більш складний	Загалом простіше
Архітектурні рішення	Використовуйте такі архітектури, як Faster R-CNN, YOLO, SSD, які включають як CNN так і ін.	Часто використовують стандартні архітектури CNN
Навчальні вибірки	Потрібні позначені зображення з класом/категорією та точними анотаціями обмежувальної рамки	Потрібні зображення з мітками класу/категорії
Часові витрати	Більші	Здатно швидше надати висновок; застосовується, коли має значення лише категорія
Приклад	Виявлення та позначення пішоходів, автомобілів і дорожніх знаків на вулиці.	Визначення зображення собаки чи kota.

Варто зазначити, що процес розпізнавання обличчя собаки є більш складним ніж людини через низку причин. Наприклад, людські обличчя мають більш різючі відмінності ніж собачі. Так собаки однієї породи можуть бути дуже схожі одна на одну.

Іншою відмінністю є алгоритми. Для розпізнавання людського обличчя вони фокусуються на основних рисах обличчя (очі, ніс, губи), для собак вони менш виразні, тому алгоритму може знадобитися враховувати інші фактори, такі як візерунок хутра.

Для розпізнавання собачого обличчя можуть використовуватись ті ж алгоритми, що і для людського, проте вони мають бути адаптовані саме під завдання розпізнавання тварини. Ще однією відмінністю є кількість

навчальних даних. Доступна велика кількість даних для навчання алгоритмів розпізнавання обличчя людини, включаючи багато загальнодоступних наборів даних. Для собак доступність якісних наборів даних є більш обмеженою.

Враховуючи що кінцеву ідентифікацію тварини буде проводити власник шляхом комунікації з тим, хто її знайшов за особливими прикметами тварини, а також проведене дослідження та порівняльний аналіз, було прийнято рішення про застосування класифікаційного методу з залучанням нейронної мережі, оскільки він здатен надати ліпші і швидші результати за використання менших ресурсів.

1.2.4 Аналіз аналогів

1.2.4.1 NOSEiD

NOSEiD – An Easier Way to Help – застосунок для пошуку собак за відбитком носа. Застосунок розроблено з залучанням штучного інтелекту для співвідношення носа знайденого собаки і собаки що загубилась (рис. 1.6).

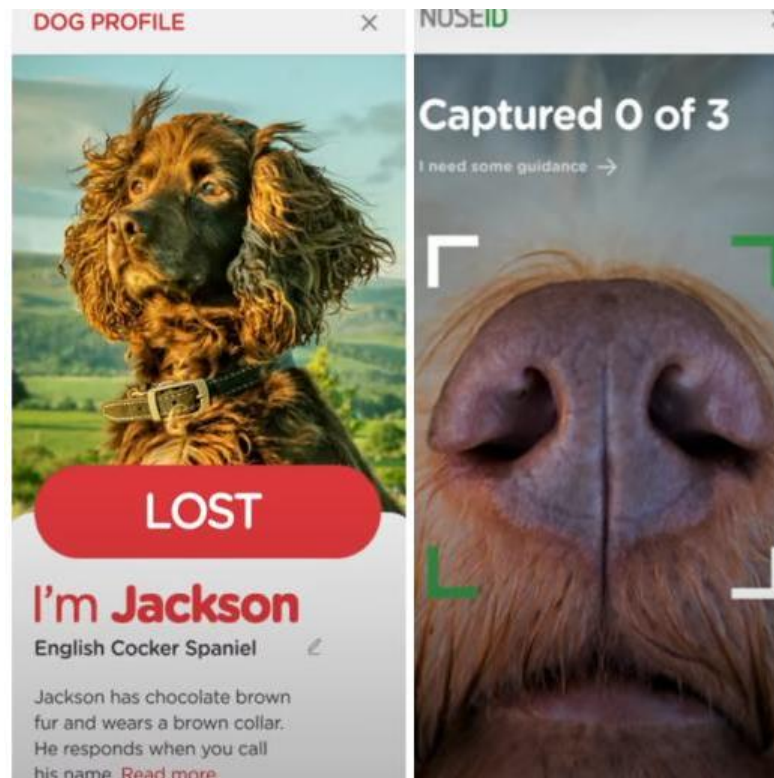


Рисунок 1.6 - NOSEiD [21]

Принцип взаємодії користувачів полягає у тому, що власник собаки повинен заздалегідь встановити застосунок на телефон і завантажити фото носа собаки. При втраті тварини хазяїн подає відповідний сигнал у застосунку. Будь-хто хто знайде собаку має сфотографувати її ніс і завантажити в програму. Застосунок за фото намагатиметься знайти її власника.

Серед переваг слід виділити високу заявлену точність розпізнавання собак. Недоліками програми є певна небезпека при застосуванні, оскільки не кожна собака здатна підпустити незнайому людину на близьку, достатню для фото носа відстань. Окрім цього наразі є обмеження на застосування програми за територіальною ознакою - наразі вона доступна лише в одному з штатів США.

1.2.4.2 WAUDOG

WAUDOG Smart ID – застосунок для пошуку собак за допомогою QR коду чи NFC мітки (рис. 1.7).

Принцип взаємодії користувачів полягає у тому, що власник собаки повинен заздалегідь купити металевий ярлик з QR та або NFC міткою і заповнити інформацію про свою тварину. При втраті тварини будь-хто хто знайде собаку має відскакувати мітку і йому буде відображено інформацію як зв'язатись з хазяїном тварини. Хазяїн отримає інформацію про те, що мітка була відсканована. Слід зазначити що власник собаки заздалегідь визначає яку інформацію буде відображено тому, хто її знайшов.

Серед переваг слід виділити 100% точність розпізнавання, оскільки виключається можливість випадкового співпадіння при скануванні мітки. Недоліками програми є те, що застосунок платний, для розпізнавання собаки тому, хто її знайшов слід близько підійти до собаки (на відстань сканування коду). Також слід зазначити що власнику собаки треба зробити певні кроки, а саме – додати інформацію у застосунок, придбати жетон. Тобто неможливо почати використовувати застосунок після втрати собаки.

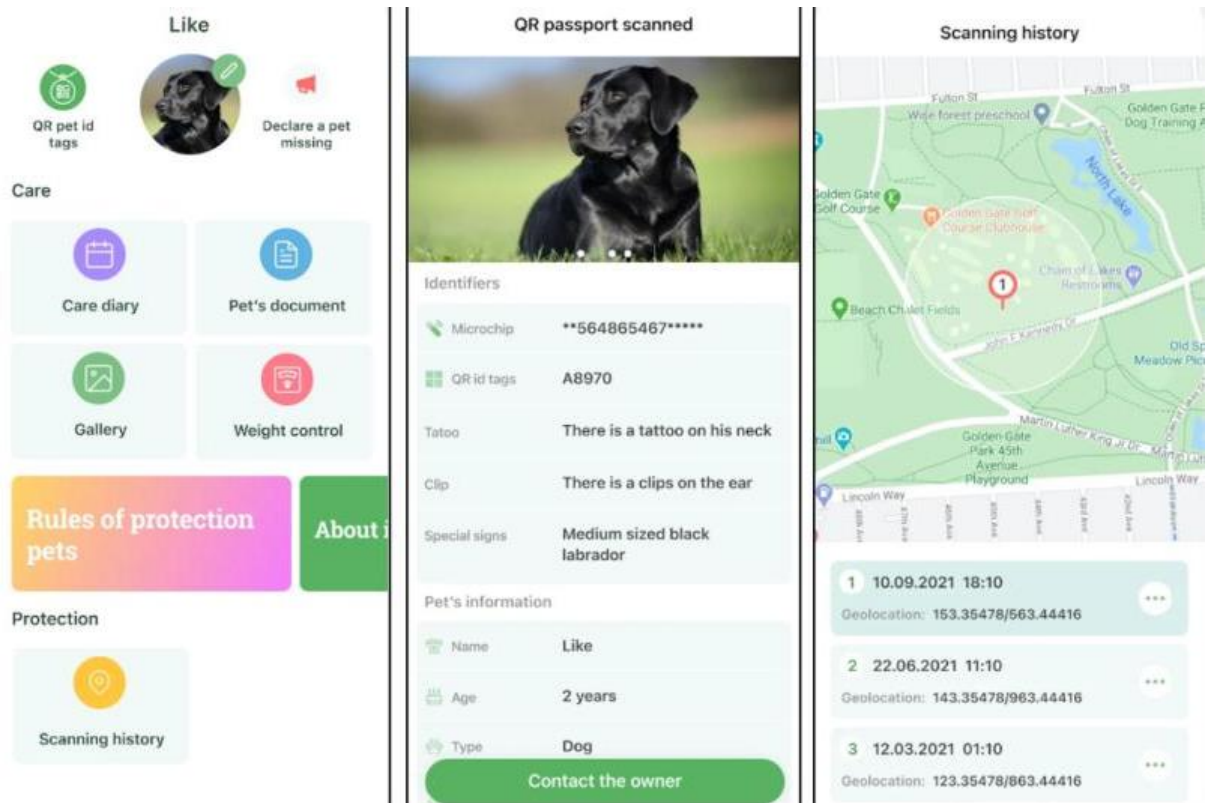


Рисунок 1.7 – WAUDOG [22]

1.2.4.3 Missing Pets

Missing Pets – Find Lost Pets – застосунок для пошуку собак через розміщення об'яви (рис. 1.8).

Принцип взаємодії користувачів полягає у тому, що власник собаки повинен розмістити повідомлення про втрату собаки з максимально детальним описом (фото, порода, особливі відмітки, остання локація). Той хто знайде собаку може розмістити відповідне оголошення з фото. Власники можуть передивлятись повідомлення про знайдені собаки, ті, хто знаходять собак можуть передивлятись поточні об'яви про втрати собак. Всі користувачі можуть почати спілкування через обмін контактами в соціальних мережах.

Серед переваг слід виділити можливість використовувати застосунок без попередньої підготовки (після втрати собаки). Застосунок безкоштовний. Недоліками програми є те, що вона потребує відносно багато часу оскільки

пошук собак відбувається безпосередньо самим користувачем шляхом перегляду усіх оголошень.

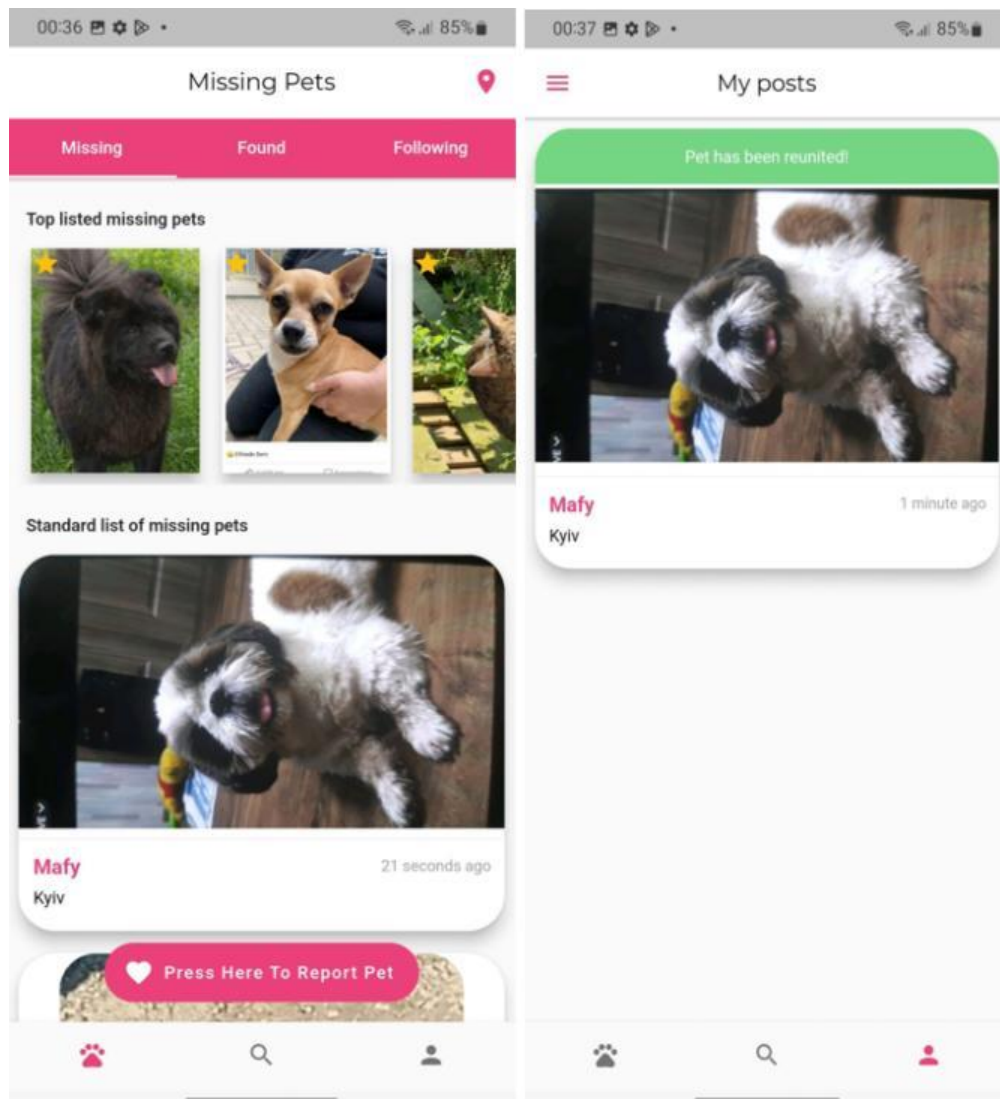


Рисунок 1.8 – Missing Pets [23]

1.3 Висновки за розділом 1

У першому розділі було проаналізовано предметну області і виділено основні поняття, зокрема штучний інтелект та штучні нейронні мережі. Розглянуто можливі підходи та існуючі методи для задач розпізнавання та класифікації. Також зроблено порівняльний аналіз для класифікації та розпізнавання в ході якого було обрано класифікацію для вирішення завдань застосунку.

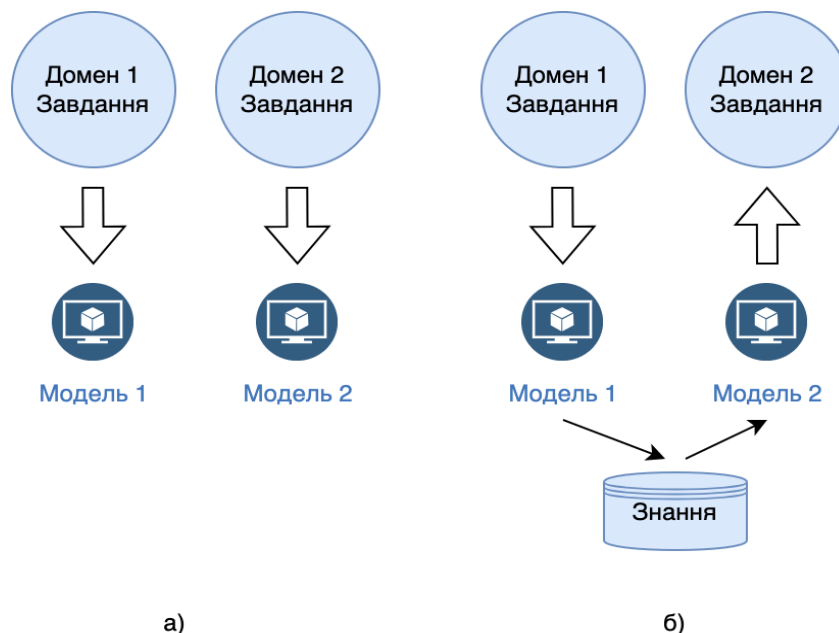
Було порівняно аналоги застосунків для пошуку домашніх тварин і виділено такі недоліки як небезпечність наближення до обличчя тварини для першого аналогу, неможливість використання застосунку після втрати тварини для другого і необхідність власноруч переглядати всі записи для третього. Розроблюваний застосунок повинен вирішити ці проблеми.

2 МАТЕРІАЛИ ТА МЕТОДИ

Враховуючи те, що у першому розділі було обґрунтовано вибір класифікації як підходу для вирішення завдань застосунку, то необхідно створити модель нейронної мережі для класифікації тварини. Для цього мережу необхідно навчити і навчання буде проходити на фото собак різних порід.

2.1 Трансферне навчання з використанням Keras

Беручи до уваги відносно велику складність поставленого завдання, необхідність досягнення високої точності роботи і чималий об'єм робіт, який необхідний для реалізації було прийнято рішення використати технологію трансферного навчання. Це підхід, що використовується в машинному навчанні і полягає в застосуванні попередньо-навчених моделей на одному домені до іншого проблемного домену [24].



а) ілюстрація традиційного навчання, б) ілюстрація трансферного навчання

Рисунок 2.1 – Ілюстрація різниці у традиційному та трансферному навчанні.

На відміну від традиційного навчання де вирішальними є розміри навчальних даних трансферне навчання може відбуватися на маленьких датасетах. Саме це і є привабливим для використання такого підходу для завдання класифікації породи через невеликий набір вхідних даних.

Попередньо-навчені моделі з бібліотеки Keras мають перевагу у вигляді відкаліброваних вагів для прогнозування. Для виявлення ознак собаки буде використано модель глибокого навчання ResNet50, що є досить популярною для завдання розпізнавання зображень. Її було навчено на великих датасетах, таких як наприклад ImageNet. ResNet50 має архітектуру залишкової нейронної мережі та 50 шарів.

ResNet50 може витягувати особливості обличчя собак та репрезентувати їх у вигляді щільних векторів (англ. embeddings) для подальшого використання для завдань кластеризації та верифікації.

За допомогою функції `load_dataset()` відбуватиметься зчитування кожного зображення разом із його міткою, яка буде зашифрована в двійковий вектор (так званий one-hot). Відповідно унітарний код матиме довжину, яка дорівнює кількості порід в датасеті. З метою перевірки успішності зчитування та завантаження даних було додано функцію `print_category()`, яка виводить на консоль статистику по кількості завантажених порід та фотографій.

Наступним кроком буде попередня обробка набору даних. Всі зображення мають бути прочитані і збережені у вигляді тривимірних масивів. Окрім того за допомогою `sk.transform.resize()` буде передбачено забезпечення відповідності зображень до вимог навчаючих моделей, а саме: для ResNet розмір зображення 224x224. Відповідність усім іншим вимогам буде забезпечено за допомогою функціоналу Keras.

Для того, щоб мати можливість контролювати вихідні дані буде видалено останній шар, що класифікує зображення на 1000 класів з ImageNet датасету. Зважаючи на те, що шари які залишилися вже мають відкалібровані ваги (для ResNet50 будуть використані ваги ImageNet), то вони будуть незмінні, для цього будуть встановлені необхідні параметри під час розробки.

Після видалення останнього шару вихідним даними моделі буде 2048-просторовий вектор властивостей. Він буде використовуватись як вхідні дані до іншої моделі.

2.2 Модель Sequential

Для поточної реалізації буде використано шар вилучення з параметром 0.7, це означає, що 70% вхідних даних відкидається на кожній ітерації. Цей параметр необхідний для попередження перенавчання. Зазвичай використовують значення від 0.2 до 0.5, проте для вхідних шарів може бути і більше [25]. Цей шар саме є вхідним для моделі, тому параметр обирався від 0.6 до 0.8 експериментальним шляхом.

Також було додано вихідний щільний шар Dense, в який передається кількість класів, та тип функції «softmax» для отримання передбачень у вигляді вірогідностей.

В функцію компіляції передається оптимізатор «adam» та «categorical_crossentropy», що використовується для багатокласової класифікації.

2.3 Висновки за розділом 2

У другому розділі було описано використані моделі та методи для створення моделі нейронної мережі для інтеграції у мобільний застосунок. Було обґрунтовано використання такого підходу як трансферне навчання та обрано претреновану модель ResNet50 завдяки наявній в ній можливості класифікувати 133 породи собак. Також було пояснено концепцію використання ResNet50 без останнього шару та використання додаткової моделі типу Sequential, що отримує на вхід виділені ResNet50 властивості з фото та повертає вірогідність співпадіння з породою.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

3.1 Опис вимог до програми

Програма для розпізнавання загублених тварин має бути виконана у вигляді мобільного застосунку, до якого висунуті наступні функціональні вимоги:

- авторизація та реєстрація за допомоги електронної поштової адреси, та сервісів Google, Facebook та Apple sign In (тільки для операційних систем ОС IOS);
- створення, зберігання та редагування записів про загублену домашню тварину з наступними параметрами: прізвисько, порода, фотокартки, адреса де було загублено тварину;
- розпізнавання тварини з фотокартки;
- видалення запису про загублену тварину;
- чат між власником та людиною, що знайшла домашню тварину;
- профайл тварини з наступними даними: прізвисько, порода, вік, фото, місто проживання, алергії, харчові вподобання, агресивність до людей та інших тварин;
- зберігання запису про загублену тварину у профайлі;
- стрічка новин з загубленими та знайденими тваринами з можливістю фільтрації за містом, породою, сортуванням за датою;
- сповіщення про знайдену схожу тварину.

Нефункціональні вимоги, видвинуті до системи:

- підримка платформ Android та IOS;
- доступний та зрозумілий інтерфейс;
- швидкодія застосунку.

Діаграму прецедентів використання зображено на рис. 3.1.



Рисунок 3.1 – Діаграма прецедентів використання застосунка клієнтом

3.2 Вибір та обґрунтування засобів розробки

3.2.1 Вибір фреймворку для мобільного застосунку

Для створення мобільного застосунку для розпізнавання загублених тварин було обрано фреймворк Flutter з використанням мови Dart. Це досить нова програмна платформа від компанії Google, що дозволяє створювати єдину кодову базу для платформ IOS, Android, Windows, MacOS, а також Web. Flutter має широкий спектр засобів для відладки та інспектування виконуваного коду. Також наявна технологія швидкого перезавантаження коду, що побудована на основі just-in-time (JIT) компіляції та дозволяє

виводити результати змін у коді за короткий час, що дуже прискорює процес розробки.

В табл. 3.1 наведено порівняння Flutter з іншими кросплатформними фреймворками: Kotlin Multiplatform, Xamarin, ReactNative.

Таблиця 3.1 – Порівняння фреймворків

Критерії	Flutter	Kotlin Multiplatform	Xamarin
Мова програмування	Dart	Kotlin	C#
Продуктивність (порівняно з нативною)	+	+	-
Єдиний користувацький інтерфейс для всіх платформ	+	-	-
ЛІТ компіляція	+	-	-
Оптимізація коду	+	+	-
Підтримка tree-shacking технології для збірок	+	Часткова	-
Середовища розробки	Android Studio, VS Code, IntelliJ IDE	Android Studio, IntelliJ IDE	Visual Studio

З порівняльної таблиці видно, що Flutter фреймворк має лідерство за рядом критеріїв і його беззаперечною перевагою є єдиний користувацький інтерфейс для різних систем, що спрощує тестування та розробки дизайну, наявність швидкого перезавантаження та високої продуктивності. Також варто зазначити, що Flutter має власну графічну бібліотеку Google Skia, а

також технологію, що контролює рендерінг об'єктів та дозволяє попереджувати зависання кадрів.

Flutter був створений для мови програмування Dart, що має наступні характеристики:

- об'єктно-орієнтована;
- має C-подібний синтаксис;
- підтримує написання як логіки програми, так і її інтерфейсу;
- однопоточна;
- динамічно типізована;
- компілюється на багато платформ;
- широкий спектр бібліотек та плагінів;
- має потужний інструментарій для налагодження програм.

3.2.2 Вибір середовища розробки мобільного застосунку

У якості середовища розробки було обрано Android Studio. Його було порівняно з іншими IDE, такими як: Android Studio, Visual Studio Code, IntelliJ IDEA. Результати порівняння наведено в табл. 3.2.

Таблиця 3.2 – Порівняння середовищ розробки

Критерій	VS Code	Android Studio	IntelliJ IDEA
1	2	3	4
Вбудовані засоби для git	+	+	+
Навантаження системи	Низьке	Високе	Середнє
Доповнення коду	+	+	+
Вбудований відлагоджувач	+	+	+
Вбудований емулятор	-	+	-

Продовження таблиці 3.2

1	2	3	4
Вбудований Android SDK	-	+	+
Безкоштовне	+	+	-

Незважаючи на високе навантаження на систему, Android Studio має велику перевагу через наявність Android SDK, без якого не можливо зробити збірку для файлу застосунку на Android.

3.2.3 Вибір мови програмування для машинного навчання

Для створення моделі машинного навчання було обрано мову програмування Python через низку переваг здебільшого пов'язаних зі зрілістю та широко-розповсюдженістю. Для Python створено близько 300 тисяч бібліотек [26] і 10 тисяч з них стосуються машинного навчання, глибокого навчання та доповненої реальності [27]. На ряду з Python також популярними у цьому напрямку є мови Java та C/C++. Їх порівняння наведене у табл. 3.3.

Таблиця 3.3 – Порівняння мов програмування для ML

Критерії	Python	Java	C/C++
1	2	3	4
Швидкість	Найменш швидка мова, інтерпритована	Середня за швидкістю з трьох. Компілюється та інтерпритується	Найбільш швидка, компілюєма мова
Легкість та швидкість написання коду	Має простий синтаксис, динамічно-типізований, містить велику кількість готових рішень	Синтаксис складніший ніж в Python, статично-типізований, менше готових рішень	Має низькорівневий і складний синтаксис, який може бути важким для освоєння

Продовження таблиці 3.3

1	2	3	4
Бібліотеки для ML	Великий обсяг бібліотек ML, таких як PyTorch, TensorFlow, Scikit-learn і тд.	Менший обсяг, порівняно з Python.	Найменша кількість бібліотек.
Популярність	Найбільш популярна для ML [28]	Середня за популярністю з трьох, проте всеж широко застосовувана для роботи з великими обсягами даних	Найменш популярна для ML через меншу кількість бібліотек та складний синтаксис
Маштабованість	Середня	Висока	Середня

Отже, за результатами порівняння було обрано мову Python завдяки найбільш швидкій для розробки, з більшим обсягом бібліотек для машинного навчання.

3.2.4 Вибір середовища розробки для алгоритмів машинного навчання

У попередньому розділі у якості мови програмування було обрано Python, тому необхідно обрати також середовище розробки і найпопулярнішими з них є Spyder, PyCharm та JupyterLab. Порівняння цих IDE наведено в табл. 3.4.

Таблиця 3.4 – Порівняння середовищ розробки

Критерій	PyCharm	Spyder	JupyterLab
Безкоштовна	Частково	+	+
Спеціалізована	Full-stack розробка	Наукові обчислення	Аналіз даних та візуалізація.
Підтримувані платформи	Windows, Mac	Windows	Web
Навантаження системи	Високе	Низьке	Середнє
Вбудований відлагоджувач	+	+	+
Доступні функції	Доповнення коду, підсвітка синтаксису, інспекція коду, авто-рефакторинг [29]	Доповнення коду, підсвітка синтаксису, аналіз стилю [30]	Доповнення коду, підсвітка синтаксису, згортання коду, інспекція [31]

В результаті порівняння було обрано JupyterLab, як оптимального середовища розробки, що призначено для аналізу даних та помірно навантажує систему. Також вона є повністю безкоштовною.

3.2.5 Вибір бібліотеки для машинного навчання

Для вирішення завдань створення та тренування нейронної мережі використовуючи мову Python існують декілька популярних бібліотек: Scikit-learn, TensorFlow та PyTorch.

Scikit-learn позиціонується як високорівнева бібліотека, що містить багато базових алгоритмів машинного навчання як з вчителем (класифікація, регресія), так і без (детекція аномалій, зменшення розмірності) [32]. TensorFlow – це низькорівнева бібліотека, що надає інструменти для створення та навчання нейронних мереж, навчальних графів. Також містить реалізації

базових операцій, наприклад, згортку чи множення матриць. Також забезпечує можливість обчислень GPU.

Фреймворк PyTorch призначений для створення та навчання глибоких нейронних мереж, з можливістю створення власних моделей та алгоритмів навчання. Використовується для вирішення завдань комп'ютерного зору чи обробки природної мови [29]. Також забезпечує можливість обчислень GPU. Порівняння бібліотек наведено у табл. 3.5.

Таблиця 3.5 – Порівняння ML бібліотек

Критерій	Scikit-learn	TensorFlow	PyTorch
Призначення бібліотеки	Машинне навчання	Глибоке навчання	Глибоке навчання
Інтеграція з Flutter	+	+	+
Обчислення з допомоги GPU	-	+	+
Розподілені обчислення	-	+	+
Візуалізація	-	+	+
Попередньо треновані моделі	Мало	Багато	Багато
Безкоштовна	+	+	+
Детальність документації	Висока	Висока	Середня

В результаті порівняння було обрано TensorFlow, як основну бібліотеку для виконання завдання завдяки обчисленням з GPU, що відрізняються за швидкістю від обчислень, які виконуються лише CPU. Також TensorFlow має детальну документацію та велику кількість попередньо тренованих моделей.

3.3 Вибір сховища даних для Flutter застосунку

Для зберігання користувацьких даних необхідно мати віддалену базу даних (БД) та арі з допомогою якого можна цими даними маніпулювати. Зважаючи на це, у якості сервісу для зберігання даних було обрано хмарне сховище від Google Firebase. Як вже зазначалося Google є розробником Flutter, тому між його сервісами та фреймворком наявний стабільний інтерфейс. Firebase має два типи сховищ RealtimeDatabase та CloudFirestore які буде порівняно у табл. 3.6.

Таблиця 3.6 – Порівняння сховищ RealtimeDatabase та CloudFirestore

Критерій	RealtimeDatabase	CloudFirestore
1	2	3
Спосіб представлення даних	Дерево json	Документи об'єднані в колекції
Маштабованість	Потребує сегментування на декілька БД при більш ніж 1000 записів в секунду [33]	Автоматично
Відслідковування активного користувача	+	-
Запити до бази даних	Сортування та фільтрація можливі не одночасно. Запроси не індексовані, продуктивність запиту зменшується з ростом розміру даних	Сортування та фільтрація можливі одночасно. Запроси індексовані, продуктивність залежить від розміру відповіді
Операції над даними	Підходить для декількох гігабайт даних, що часто змінюються	Підходить для сотні гігабайт даних, що помірно змінюються

Продовження таблиці 3.6

1	2	3
Можливість прослуховуванням застосунку оновлень	+	+
Надійність та доступність	Підтримується тільки регіональна конфігурація, тобто БД доступні в межах регіону для якого налаштовано конфігурацію. Мають високу швидкість відповіді клієнту.	Дані зберігаються в різних центрах обробки, що знаходяться в декількох регіонах. Підтримується як регіональна так і мультирегіональна конфігурації.

Враховуючи порівняння двох сховищ у табл. 3.6 та те, що проєкт потребує значної кількості місця в БД та обидва сховища підтримують підписку застосунку на оновлення то було обрано CloudFirestore.

3.4 Вибір сервісу для сповіщення користувача

Враховуючи те, що у розділі 3.4 для зберігання даних було обрано хмарне сховище CloudFirestore від Google, то вибір сервісу для нотифікацій стає очевидним - Firebase Cloud Messaging (FCM).

Враховуючи те, що зазвичай логіка для відправки нотифікацій реалізується з допомогою серверу то безпосередньо на сервісах таких як, наприклад, OneSignal чи Firebase Cloud Messaging, можливість керувати нотифікаціями досить вузька. І хоч, налаштування шаблонів для відправки на сервісі досить просте завдання, вони підтримують лише меседжі, що не прив'язані до даних.

Головною причиною наявності сповіщень у застосунку є необхідність сповістити користувача про нове повідомлення. Враховуючи це необхідність

серверної частини є невід’ємною частиною для реалізації сповіщень. Проте Firebase має рішення – це Firebase Functions. Цей сервіс надає можливість створення серверного коду. З його допомоги і буде реалізовано поєднання роботи сервісів CloudFirestore та Firebase Cloud Messaging. До того ж він має дуже простий спосіб налаштування конфігурації та дозволяє обирати мову для написання функцій.

Комбінація цих сервісів передбачається Google, тому є можливість застосувати їх до одного проєкту. Після налаштувань вони додаються у бокову панель (рис. 3.2).

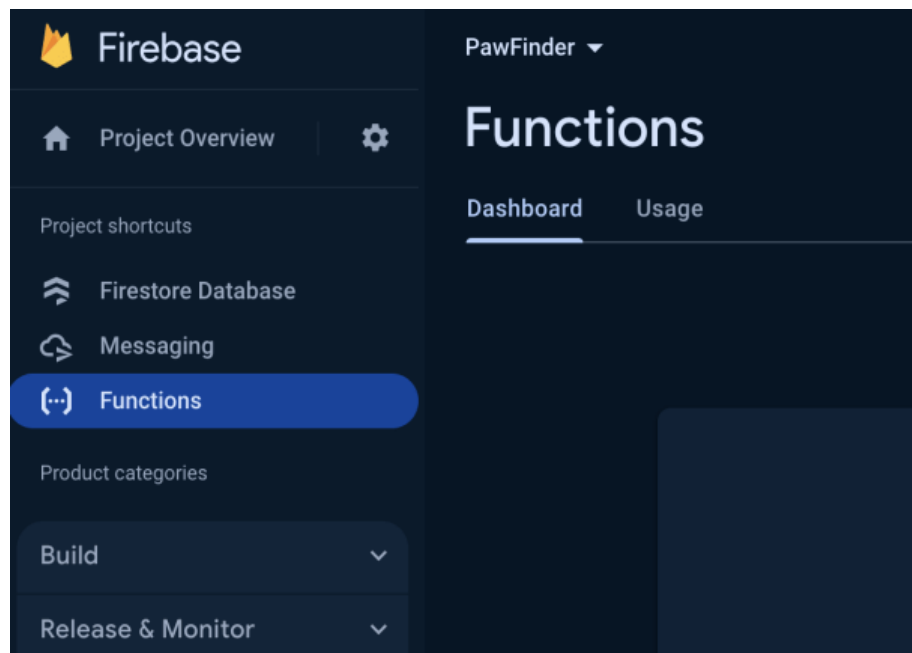


Рисунок 3.2 – Бокова панель проєкту

Тому поєднання саме сервісів Firebase від Google дають найбільш гнучкий функціонал для налагодження роботи застосунку без серверної частини.

3.5 Керування станами у Flutter застосунку

Для того, щоб застосунок відповідав реактивній парадигмі, необхідно обрати засоби керування станами (англ. state-management). Мобільні

застосунки мають 2 типи станів. Перший – це глобальний стан застосунку, він відповідає за зберігання даних між сесіями, наприклад: авторизаційний токен, вибір користувачем мови застосунку, дозволів на використання геоданих, файлів, камери, інтернету і т.д.. Другий тип – це стан застосунку у поточній сесії. Він відповідає за поточну сторінку, дані якими оперує користувач, цикли анімацій, стани інших сторінок. До того ж з допомоги засобів керування станами можна реалізувати архітектуру, що дозволяє відділити бізнес-логіку від користувацького інтерфейсу застосунку.

Фреймворк Flutter має вбудовані можливості для state-management. Реалізовані вони через наявність у елементів інтерфейсу власного стану. Проте, проблема постає у обміні станами між декількома віджетами, що є складним завданням, зокрема для середніх та великих застосунків. А також анти-паттерном для маленьких, через складність їх розширювати.

Отже, з'являється необхідність обрати бібліотеку для керування станами. Найбільш популярними засобами state-management є BLoC, Provider та Redux. Їх порівняння наведено у табл. 3.7.

Таблиця 3.7 – Порівняння бібліотек для керування станами

Критерій	Provider	Redux	BLoC
1	2	3	4
Парадигма	Реактивна	Функціональна	Реактивна
Маштабованість	Підходить для малих та середніх застосунків	Підходить для середніх та великих застосунків	Підходить для середніх та великих застосунків
Складність	Низька	Помірна	Висока
Зберігає послідовність реакцій на виклики	-	+	+

Продовження таблиці 3.7

1	2	3	4
Сховища даних	Різні, будь-яка кількість [35]	Одне спільне сховище [34]	Різні, будь-яка кількість [36]
Підписка на зміни окремих властивостей	-	+	+
Продуктивність	Висока [35]	Середня [34]	Висока [36]

Отже, в результаті порівняння бібліотек було обрано BLoC у якості засобу керування станами застосунку завдяки його перевагам у вигляді високої продуктивності, відповідності реактивній парадигмі та різним сховищам, що полегшують підтримку застосунку.

3.6 Архітектура Flutter застосунку

Згідно до розділу 3.3 було обрано бібліотеку BLoC для керування станами, що також допомагає у реалізації чистої архітектури, що відокремлює рівні даних, інтерфейсу та бізнес логіки. Такий підхід дозволяє зробити код підтримуваним, зрозумілим та легким у тестуванні, забезпечує контрольовану зв'язність між рівнями.

Архітектура BLoC реалізується таким чином, що будь-який клас призначений керувати станом, називається bloc, має розширитись абстрактним класом бібліотеки і це наділяє його потоком власних станів - state. Стан - клас, що містить дані та має способи порівнювати свої екземпляри між собою. Для того, щоб змінити state на новий bloc необхідно викликати подію - event, що додасть у потік інший екземпляр state. Клас-bloc містить логіку програми: запити до сервісів, обробляє дані та зберігає їх. Презентаційний рівень підписується на оновлення bloc, що сповіщає про новий state. На ньому можна визначити умови, за яких відбудеться перебудова

інтерфейсу або виконається якась функція. Презентаційний рівень може звернутися до bloc для того, щоб викликати функцію, що виконує якусь логіку, наприклад завантажує дані. Коли завантаження завершено, за необхідності, в потік додається event та оновлюються дані у state. Презентаційний рівень сповіщається про новий state. Для більшої наглядності архітектуру наведено на рис. 3.3.

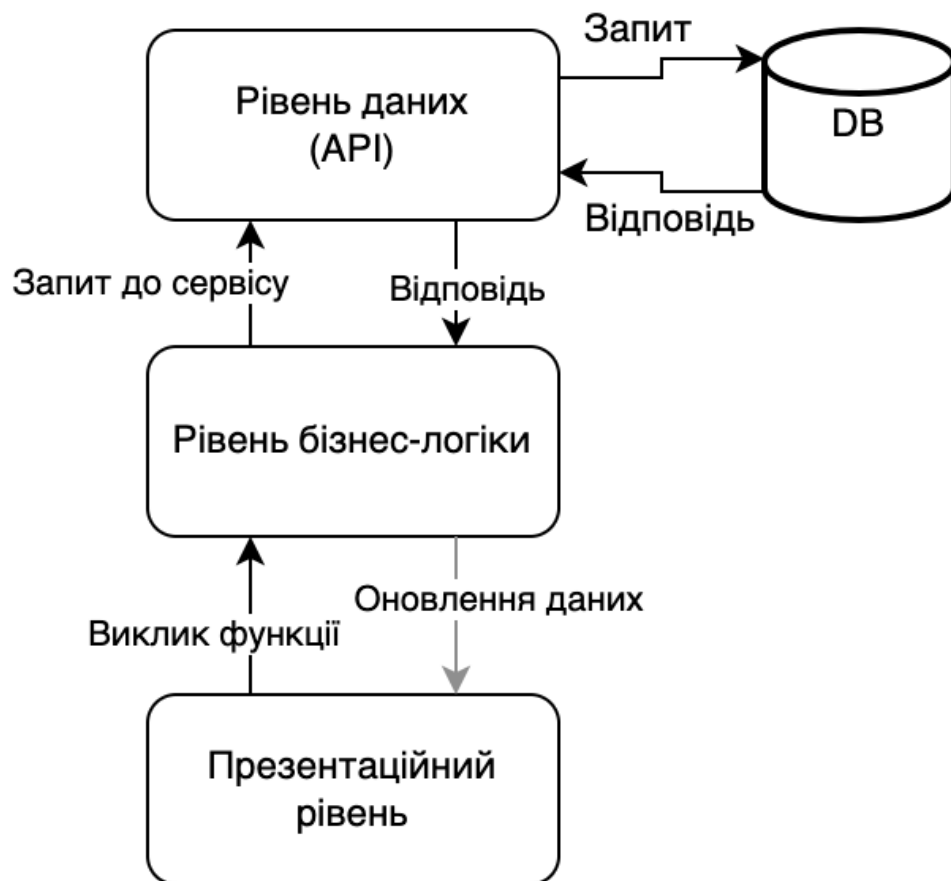


Рисунок 3.3 – Схема архітектури застосунку

3.7 Висновки за розділом

У третьому розділі було визначено функціональні та нефункціональні вимоги до мобільного застосунку. Також розглянуто та порівняно існуючі засоби для розробки. Для мобільного застосунку було обрано фреймворк Flutter, що якісно переважає над іншими, завдяки контролю над рендерінгом користувацького інтерфейсу, засобам для швидкої розробки та

продуктивністю застосунків. Android Studio було обрано в якості середовища розробки.

Для створення моделі машинного навчання було обрано мову Python, середовище JupyterLab та TensorFlow в якості основної бібліотеки. Таке поєднання засобів було визначено, як найбільш ефективне для виконання завдання розпізнавання загубленої домашньої тварини. В якості бібліотеки для керування станами було обрано VLoC, на основі якої було побудовано архітектуру застосунку.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

4.1 Структура Flutter застосунку

Структура мобільного застосунку представлена на рис. 4.1.

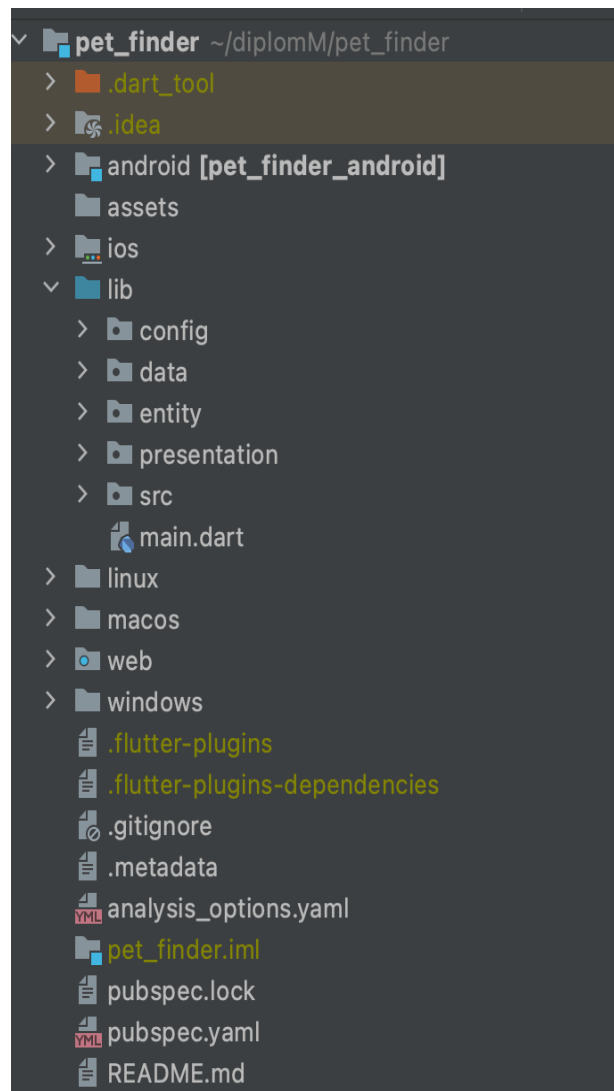


Рисунок 4.1 – Структура Flutter застосунку

Структура складається з таких файлів та папок як:

- папка `.dart_tools`, який містить файли, що використовуються Dart SDK. Автоматично генерується при створенні проєкту;
- папка `.idea` згенерований середовищем розробки, зберігає його налаштування;

- папка `android` містить папки та файли для збірки та виконання застосунку для Android;
- папка `assets` містить медіа-ресурси застосунку, там також буде зберігатись навчена модель;
- папка `ios` містить папки та файли для збірки та виконання застосунку для iOS;
- папка `lib` папка з клієнтським кодом, що містить папки: `presentation` - містить файли та папки з кодом елементів користувацького інтерфейсу, `config` - код для налаштування API та навігації, `data` - файли та папки з кодом для сервісів та локальних сховищ, `entity` - для моделей та об'єктів, `src` - код бізнес-логіки, `main.dart` - файл з якого виконується застосунок;
- папка `linux` містить папки та файли для конфігурації, збірки та виконання застосунку для Linux, таке ж призначення мають папки `windows`, `macos`;
- папка `web` крім файлів та папок для збірки містить вхідну точку для веб-застосунку - файл `index.html`;
- файл `.flutter_plugins` - автоматично-згенерований для зберігання шляхи до кешу бібліотек;
- файл `.flutter_plugins_dependencies` - автоматично-згенерований для зберігання залежностей бібліотек;
- файл `.gitignore` - зберігає шляхи до файлів, що мають ігноруватися git;
- файл `.metadata` - відслідковує властивості проекту Flutter для виконання оновлень;
- файл `.analysis_options.yaml` містить правила щодо стилістики коду, та аналізу коду загалом;
- файл `pet_finder.iml` - Java - модуль необхідний для зберігання структури проекту в середовищі розробки AndroidStudio;
- файл `pubspec.lock` зберігає інформацію про версії встановлених бібліотек;

- файл `pubspec.yaml` містить інформацію про проєкт: назву, версію SDK, перелік встановлених бібліотек, а також шляхи до папки з медіа-ресурсами;
- файл `Readme.md` містить супровідну інформацію для розробників проєкту.

4.2 Структура Python проєкту

Python проєкт містить таку структуру файлів та папок, що наведено на рис. 4.2.

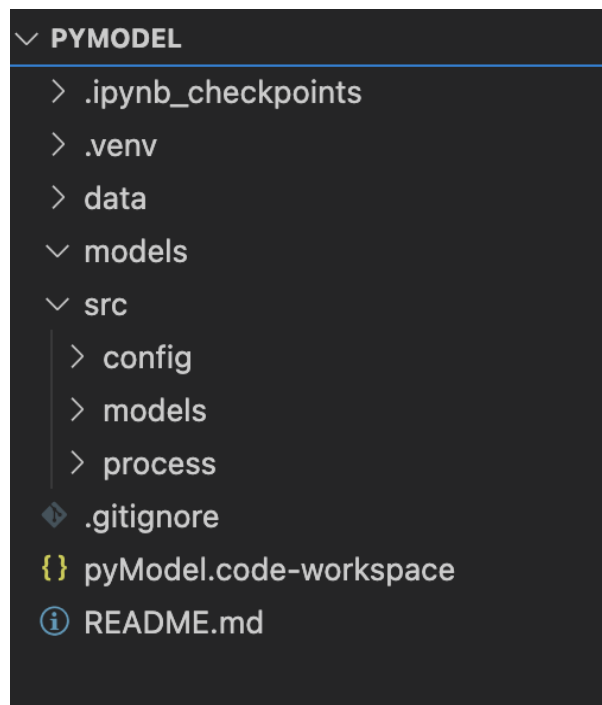


Рисунок 4.2 – Структура Python проєкту

Проєкт складається з таких папок і файлів:

- `.ipynb_checkpoints` – папка що створюється під час запуску блокнота Jupyter. І де Jupyter зберігає контрольні версії блокнотів;
- `.venv` – папка, що містить налаштування віртуального середовища. Воно необхідне для встановлення бібліотек без впливу на глобальні пакети Python;

- data – папка, що містить файли з сирими даними та готовим датасетом;
- models – папка, що зберігає натреновані моделі;
- src – папка з вихідним кодом проєкту, що також містить папки: config – де зберігаються необхідні константи, models - містить скрипти для тренування моделей та їх тестування та process - містить файли в яких відбувається завантаження даних, перетворення їх з допомоги ResNet50 та опрацювання;
- файл .gitignore - зберігає шляхи до файлів, що мають ігноруватися git;
- файл Readme.md містить супровідну інформацію для розробників проєкту.

4.3 Алгоритм функціонування розробленої програми

Основні процеси, що виконує застосунок представлені на рис. 4.3.

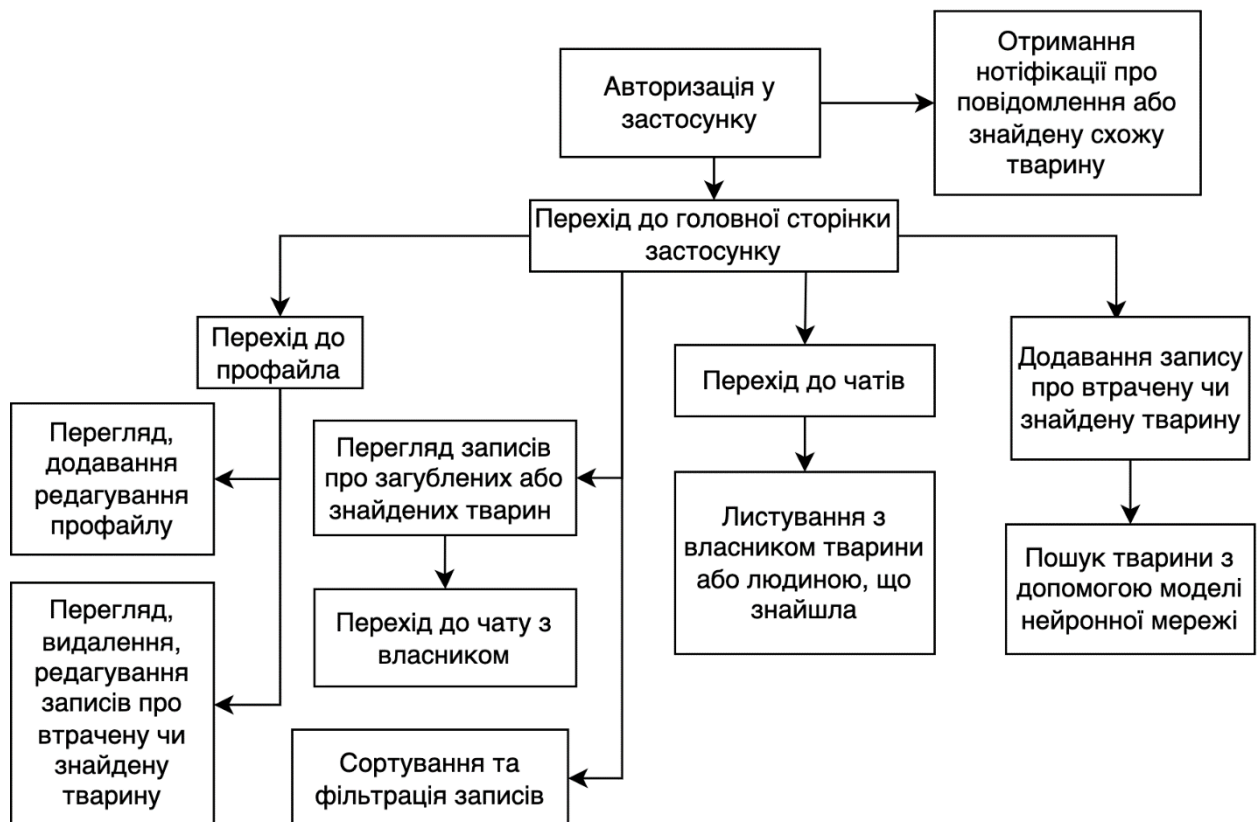


Рисунок 4.3 – Функціональна схема застосунку

З рис. 4.3 можна побачити, що під час роботи над застосунком було розроблено наступні алгоритми:

- реєстрація та авторизація за допомогою email;
- реєстрація та авторизація за допомогою сервісів Google та Facebook;
- алгоритм створення допису про втрачену тварину;
- алгоритм створення допису про знайдену тварину;
- алгоритм пошуку тварини за фотографією за допомогою навченої моделі нейронної мережі;
- видалення допису;
- алгоритм створення профайлу користувача;
- алгоритм сортування допису;
- алгоритм фільтрації попису;
- алгоритм відправки повідомлення автору допису;
- алгоритм прийняття push – нотіфікацій.

4.3.1 Опис алгоритму авторизації за допомоги email

Нижче представлено кроки алгоритму для авторизації за допомоги email:

Крок 1. Дані з форми яку заповнив користувач передаються у AuthorizationEvent() блоку авторизації AuthorizationBloc.

Крок 2. Запит до сервісу для перевірки відповідного користувача у базі даних.

Крок 3. Якщо запит повертає авторизаційний токен то перехід до кроку 4, якщо ні - крок 6.

Крок 4. Зберегти авторизаційний токен у локальному сховищі.

Крок 5. Відобразити користувачу головну сторінку. Перехід на крок 7.

Крок 6. Відобразити повідомлення про некоректний логін чи пароль.

Крок 7. Кінець алгоритму.

4.3.2 Опис алгоритму реєстрації за допомоги email

Нижче представлено роки алгоритму для реєстрації за допомоги email:

Крок 1. Дані з форми яку заповнив користувач передаються у `RegistrationEvent()` блоку авторизації `AuthorizationBloc`.

Крок 2. Запит до сервісу для перевірки існування email у базі даних.

Крок 3. Якщо запит повертає авторизаційний токен то перехід до кроку 4, якщо помилку - крок 6.

Крок 4. Зберегти авторизаційний токен у локальному сховищі.

Крок 5. Відобразити користувачу головну сторінку. Перехід на крок 7.

Крок 6. Відобразити повідомлення про існування користувача з таким же email.

Крок 7. Кінець алгоритму.

4.3.3 Опис алгоритму авторизації/реєстрації за допомоги сервісів Google та Facebook

Користувач може обрати один з сервісів для авторизації чи реєстрації у застосунку. У даному випадку алгоритм для обох завдань та сервісів співпадає та наведено нижче:

Крок 1. Для обраного користувачем метода викликається метод `ThirdPartyAuthBloc()` блоку авторизації `AuthorizationBloc` і передається параметр `AuthType.Google` або `AuthType.Facebook`.

Крок 2. Виконується запит до відповідного сервісу, що відкриває нативне вікно для авторизації в Facebook або Google.

Крок 3. Користувач вводить логін та пароль до свого акаунту. Якщо дані коректні, то сервіс повертає авторизаційний токен, перехід на крок 4. Якщо дані некоректні всередині нативного вікна висвічується повідомлення та перехід на крок 7.

Крок 4. Авторизаційний токен надсилається на сервер, якщо у відповідь повертається інший авторизаційний токен, то перехід до кроку 6, якщо ні перехід до кроку 5.

Крок 5. Відобразити користувачу повідомлення з помилкою. Перейти до кроку 8.

Крок 6. Зберегти авторизаційний токен у локальному сховищі.

Крок 7. Відобразити користувачу головну сторінку.

Крок 8. Кінець алгоритму.

4.3.4 Опис алгоритму створення допису про втрачену тварину

Нижче наведено алгоритм створення допису про втрачену тварину:

Крок 1. Користувач викликає форму створення допису.

Крок 2. Виводиться діалогове вікно з вибором типу допису.

Крок 3. Користувач обирає допис про втрачену тварину, викликається `FoundPetFormEvent()` блоку `PostBuilderBloc`.

Крок 4. Відкривається сторінка для створення допису про втрачену тварину.

Крок 5. Користувач обирає заповнити форму з профайлу - перехід на крок 7. Якщо обирає заповнити форму самостійно - крок 7.

Крок 6. Користувач заповнює пусті поля: локацію де втрачено тварину, дату коли було втрачено тварину та додає фото. Перехід на крок 8.

Крок 7. Користувач вводить дані: прізвисько, порода, вік, локацію де втрачено тварину, алергії, харчові вподобання, агресивність до людей та інших тварин, дату коли було втрачено тварину та фото.

Крок 8. Дані з форми у вигляді класу `LostPetDTO` передаються `PetWasLostEvent()` блоку `PetsProcesignBloc`.

Крок 9. Заповнені дані додаються до бази даних.

Крок 10. Виконується алгоритм розпізнавання тварини (розд. 4.3.7).

Крок 11. Якщо схожі тварини знайдені, то користувачу відображаються дописи з ними. Якщо ні - повідомлення про те, що дописи не знайдені.

Крок 12. Кінець алгоритму.

4.3.5 Опис алгоритму створення допису про знайдену тварину

Нижче наведено алгоритм створення допису про знайдену тварину:

Крок 1. Користувач викликає форму створення допису про знайдену тварину.

Крок 2. Виводиться діалогове вікно з вибором типу допису.

Крок 3. Користувач обирає допис про знайдену тварину викликається `FoundPetFormEvent()` блоку `PostBuilderBloc`.

Крок 4. Відкривається сторінка для створення допису про знайдену тварину.

Крок 5. Користувач вводить дані: імовірна порода, локацію де знайдено тварину, дата коли було втрачено тварину та фото;

Крок 6. Дані з форми у вигляді класу `FoundPetDTO` передаються `PetWasFoundEvent()` блоку `PetsProcessignBloc`.

Крок 7. Заповнені дані додаються до бази даних.

Крок 8. Виконується алгоритм розпізнавання тварини (розд. 4.3.7).

Крок 9. Якщо схожі тварини знайдені, то користувачу відображаються дописи з ними. Якщо ні - повідомлення про те, що дописи не знайдені.

Крок 10. Кінець алгоритму.

Для наглядності взаємодії алгоритмів 4.3.5, 4.3.6 та 4.3.7 наведено блоксхему (рис. 4.4).

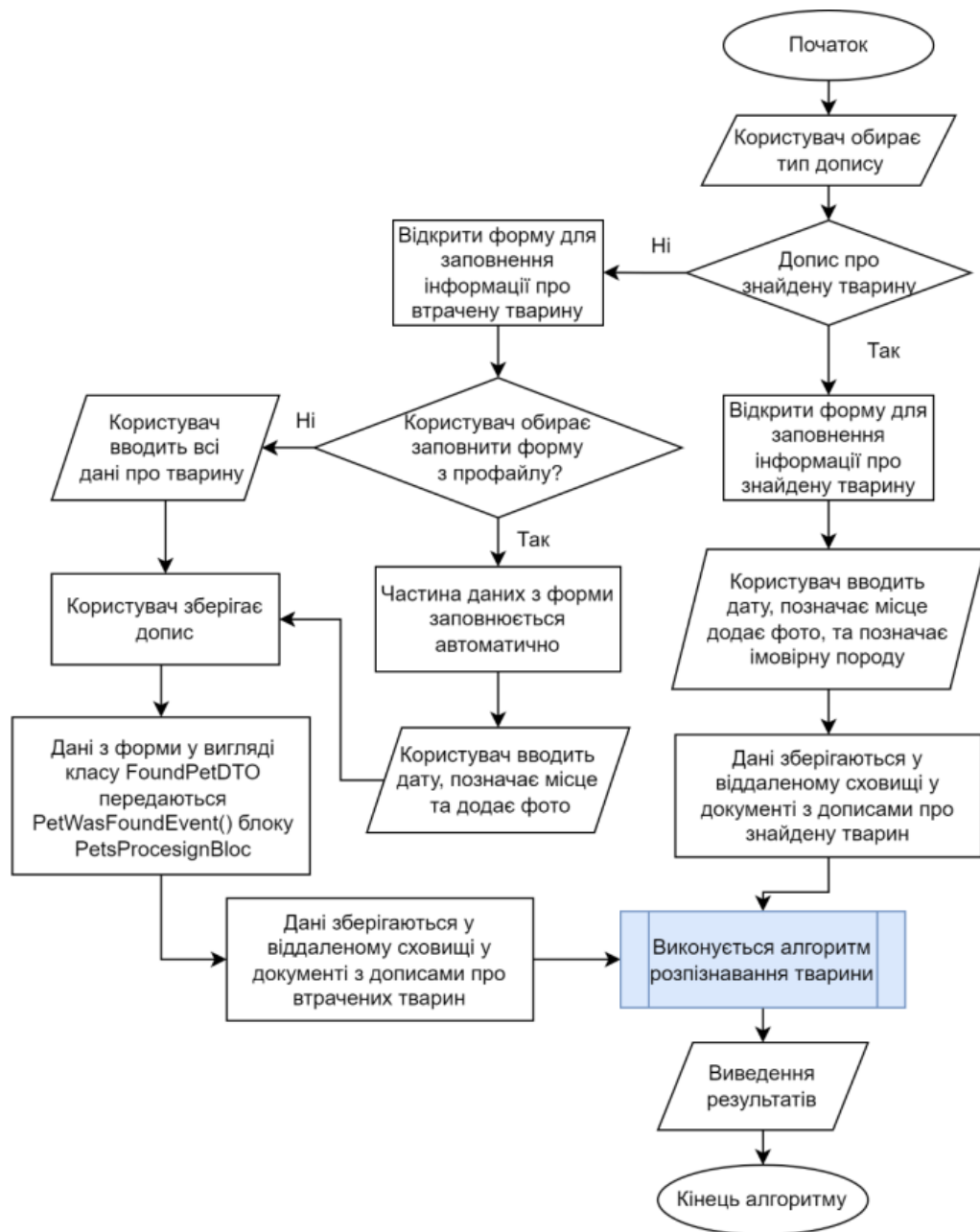


Рисунок 4.4 – Блок-схема створення записів про знайдену та втрачену тварину

4.3.6 Опис алгоритму пошуку тварини за фотографією за допомогою навченої моделі нейронної мережі

Крок 1. Список зображень, локація де знайдено тварину, ймовірна порода (не обов'язкове поле), дата коли втрачено тварину, передаються у функцію `breedClassifier()` сервісу `DogRecognitionService`.

Крок 2. Підвантаження моделі нейронної мережі та позначок функцією `loadModel()` .

Крок 3. Створюється мапа, де ключ це порода, а значення - кількість співпадінь.

Крок 4. Цикл. Якщо змінна `i` рівна довжині списку - перехід на крок 6. Якщо ні, то `i`-те зображення передається у функцію `Tflite.runModelOnImage()`. Параметр кількості результатів, що видасть модель встановлено як 2.

Крок 5. Результати функції - породи та ймовірності перевіряються на наявність в мапі. Якщо ключ існує, то до значення додається ймовірність. Якщо ні - до мапи додається ключ зі значенням відповідної ймовірності.

Крок 6. Змінна `i` збільшується на 1. Перехід на крок 4.

Крок 7. Якщо ймовірна порода вказана, то вона додається до мапи зі значенням 1.

Крок 8. Отримана мапа сортується за величиною значення.

Крок 9. За кожним ключом з мапи, типом допису(знайдені, втрачені) та за містом з локації відбувається запит до бази даних.

Крок 10. Якщо у відповідь повертається список `id` записів, то вони додаються у сет.

Крок 11. За кожним `id` з сету отримується його запис.

Крок 12. Записи сортуються за датою в межах однієї породи.

Крок 13. Якщо записи є алгоритм повертає відсортований список з ними, якщо ні - пустий.

Крок 14. Кінець алгоритму.

Блок-схему алгоритму наведено на рис. 4.5.

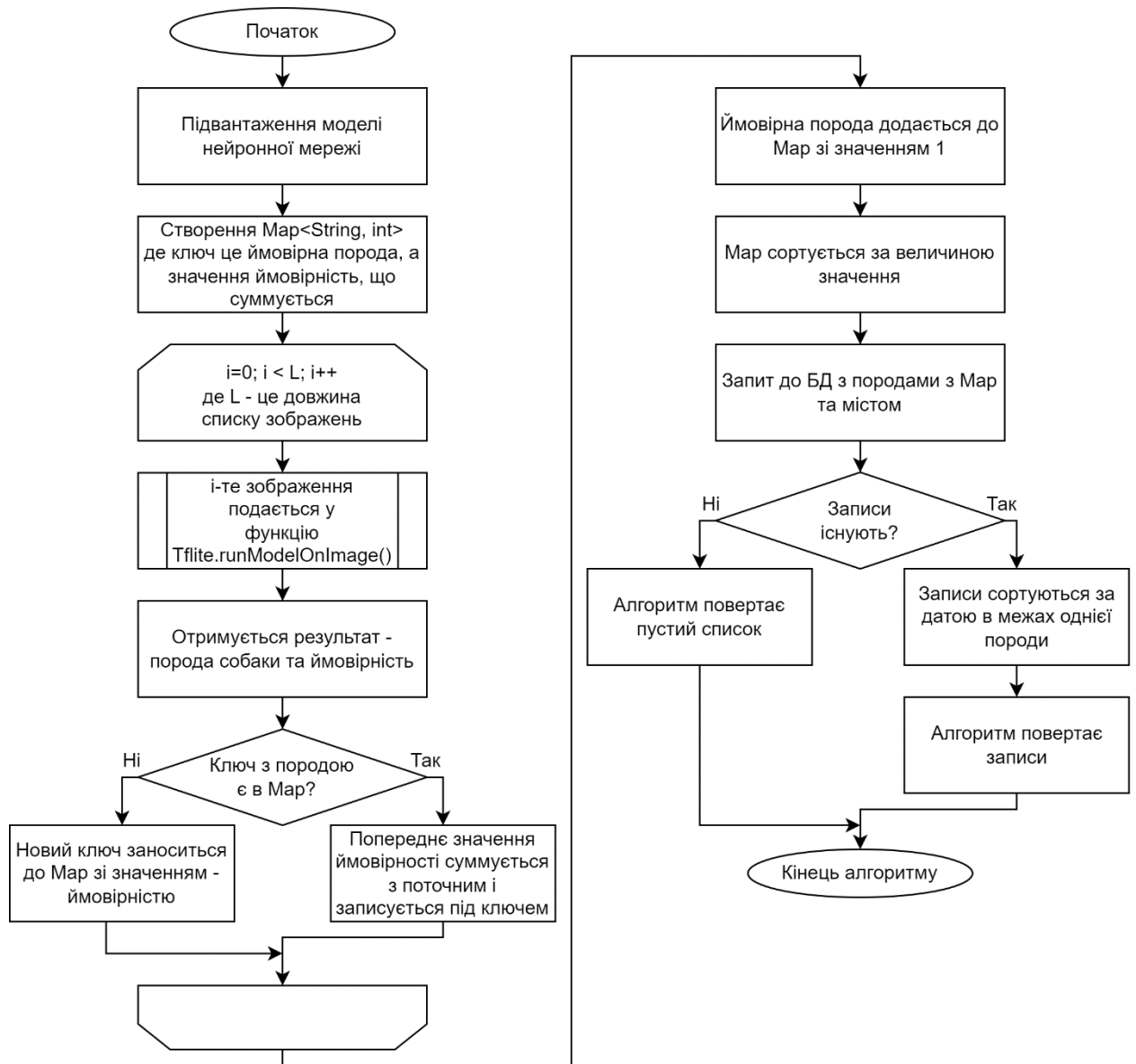


Рисунок 4.5 – Блок-схема алгоритму пошуку тварини за фотографією з допомогою навченої моделі нейронної мережі

4.3.7 Опис алгоритму видалення допису

Нижче буде наведено алгоритм видалення допису про знайдену/втрачену тварину:

Крок 1. Користувач в профайлі обирає видалити певний допис. Викликається подія `DeletePostEvent()` блоку `PetsProcesignBloc`.

Крок 2. Тип запису (знайдено/втрачено) та ідентифікатор надходять у DeletePostEvent(). Якщо тип запису - знайдена тварина, то крок 3. Якщо втрачена - крок 4.

Крок 3. До бази даних робиться запит про видалення запису з списку дописів про знайдених тварин. Якщо повертається відповідь про успішне видалення - крок 5.

Крок 4. До бази даних робиться запит про видалення запису з списку дописів про втрачених тварин. Якщо повертається відповідь про успішне видалення - крок 5.

Крок 5. Видалення з локального списку дописів про знайдених тварин. Перехід до кроку 7.

Крок 6. Видалення з локального списку дописів про втрачених тварин.

Крок 7. Оновлення стану PetsProcesignBloc та сторінки з дописами користувача.

Крок 8. Кінець алгоритму.

4.3.8 Опис алгоритму сортування дописів

Наразі сортування відбувається лише у новинній стрічці за датою створення допису. За замовчуванням застосовується сортування записів за «від нових до старих» дописів. Нижче наведено алгоритм сортування записів.

Крок 1. Користувач застосовує сортування для дописів. Викликається подія SortPostListEvent() блоку PetsProcesignBloc.

Крок 2. В подію передається тип записів, що сортуються. Перевіряється яке сортування застосовано зараз і записується обернене. Якщо записано «від старих до нових» - крок 3, якщо ні - крок 4.

Крок 3. Відбувається сортування списку дописів за датою «від старих до нових», перехід до крок 5.

Крок 4. Відбувається сортування списку дописів за датою «від нових до старих», перехід до крок 5.

Крок 5. Оновлення стану PetsProcesignBloc та новинної стрічки з дописами.

Крок 6. Кінець алгоритму.

4.3.9 Опис алгоритму фільтрації дописів

Фільтрація дописів відбувається у новинній стрічці за критеріями порода та місто. Алгоритм фільтрації наведено нижче.

Крок 1. Користувач обирає за якими саме критеріями будуть відфільтровані записи. Списки обраних пород та міст передаються в FilterPostListEvent() блоку PetsProcesignBloc.

Крок 2. Зі списків знайдених та втрачених тварин обираються записи, що містять породи тварин задані фільтром та створюються два списки.

Крок 3. Записи з отриманих списків перевіряються на задані в фільтрі міста, та ті що не співпадають - видаляються зі списку.

Крок 4. Оновлення стану PetsProcesignBloc() та новинної стрічки з дописами.

Крок 5. Кінець алгоритму.

4.3.10 Опис алгоритму відправки повідомлення автору допису

Перейти до листування з автором допису можна викликавши чат на дописі. Алгоритм відправки повідомлення наведено нижче.

Крок 1. Користувач натискає кнопку чату на дописі після чого викликається подія GetChatDetailsEvent() блоку ChattingBloc.

Крок 2. У подію надходить ідентифікатор і тип допису. За цими даними робиться запит до БД. Якщо повертаються дані з деталями чату - крок 3, якщо помилка - крок 4.

Крок 3. Деталі чату повертаються у місце виклику функції та із цими даними користувач перенаправляється в чат з автором посту. Перехід на крок 5.

Крок 4. Користувачу відображається повідомлення про помилку.

Крок 5. Користувач вводить повідомлення та відправляє. Викликається подія `SendMessageEvent()` блоку `ChattingBloc`.

Крок 6. В `SendMessageEvent()` надходить текст повідомлення та деталі чату. Відправляється запит до БД на додавання повідомлення. Якщо повернулась відповідь про успішне створення - крок 7, якщо помилка - крок 8.

Крок 7. Оновлення стану `ChattingBloc` та зміна стану повідомлення на «відправлене».

Крок 8. Оновлення стану `ChattingBloc` та зміна стану повідомлення на «помилка».

Крок 9. Кінець алгоритму.

4.3.11 Опис алгоритму отримання push – нотифікації

Для того, щоб застосунок міг отримувати нотифікації необхідно виконати кроки, що наведені нижче.

Налаштування проєкту:

Крок 1. Створити та налаштувати проєкт `Firebase` та додати туди застосунки. Детальніший опис наведено у розділі 4.5.

Крок 2. Дозволити запит дозволу на нотифікації для `IOS` у `IDE XCode` та `Android` у файлі `build.gradle`.

Алгоритм, що виконується у клієнтському застосунку:

Крок 1. Отримати та завантажити до `Firebase` токен, що є унікальним ідентифікатором девайсу користувача.

Крок 2. Надати дозволи на отримання нотифікацій для платформ `IOS` та `Android`.

Крок 3. Якщо користувач дозволив отримання нотифікацій - крок 4.
Якщо ні - крок 5.

Крок 4. Ініціалізація обробника нотифікацій.

Крок 5. Кінець алгоритму.

4.4 Алгоритми Python проєкту

Python проєкт потрібний для створення та навчання моделі нейронної мережі, що буде отримана у якості вихідного файлу і може бути інтегрована з Flutter застосунком.

В ході роботи над ним було створено наступні алгоритми:

- обробка та збереження датасету;
- імпортування датасету;
- створення та модифікація моделі;
- тренування;
- оцінка.

4.4.1 Опис алгоритму імпортування датасету

Для імпортування датасету необхідно додати його у проєкт та вказати шлях до нього, константу названо PATH. Алгоритм імпортування наведено нижче:

Крок 1. Перевірка, чи існує заданий PATH для набору даних. Якщо так, то крок 2, ні - крок 15.

Крок 2. Друкується повідомлення про те, що набір даних завантажується.

Крок 3. Ініціалізація масивів для імен файлів, міток.

Крок 4. Об'являється ітератор $ind = 0$.

Крок 5. Цикл. Якщо в кореневій директорії було перебрано останій каталог з файлами - крок 15 . Якщо ні - крок 6.

Крок 6. Якщо кількість файлів у каталозі більше 1, то крок 7. Якщо ні - крок 15.

Крок 7. Цикл. Об'являється ітератор $i = 0$ для перебору файлів у кожному каталозі визначеному РАТН.

Крок 8. Якщо i менше довжини масиву файлів, то крок 9. Якщо ні – крок 10.

Крок 9. i -ий шлях до файлу додається до масиву імен файлів та йому призначається мітка на основі значення ind помноженого на довжину масиву файлів.

Крок 10. Збільшення значення ind та i . Перехід на крок 7.

Крок 11. Якщо принаймні одну мітку було призначено файлам, то крок 12, якщо ні – крок 15.

Крок 12. Виведення повідомлення про завершення завантаження набору даних, а також загальної кількості імпортованих зображень.

Крок 13. Обчислюється загальна кількість класів, шляхом пошуку кількості унікальних міток у масиві міток.

Крок 14. Виведення загальної кількості класів.

Крок 15. Кінець алгоритму.

4.4.2 Опис алгоритму обробки датасету

Як вже зазначалося для класифікації буде використано заздалегідь навчену модель ResNet50. Проте вона приймає на вхід зображення 224 на 224 пікселів, що мають три канали кольору.

Нижче наведено алгоритм обробки датасету:

Крок 1. Проініціалізовано список, `resized_data`, яки зберігатиме зображення зі зміненим розміром для набору даних, та список `resized_labels` для міток.

Крок 2. Цикл. Об'явлено ітератор $i = 0$.

Крок 3. Якщо ітератор дорівнює кількості файлів в наборі даних, то крок 8. Якщо ні, крок 4.

Крок 4. З *i*-го файлу зчитується зображення в пам'ять за допомогою бібліотеки обробки зображень, перетворюючи зображення на числовий масив, яким можна маніпулювати в Python.

Крок 5. Перевіряється, чи має масив зображень три виміри, що відповідають ширині, висоті та кольоровим каналам зображення. Якщо так - крок 6, якщо ні, зображення виключається з датасету - крок 4.

Крок 6. Розмір зображення змінюється до заданої форми (за замовчуванням [224, 224, 3]) за допомогою функції зміни розміру з *skimage*.

Крок 7. Зображення зі зміненим розміром додається до *resized_data*, додаються відповідні мітки до *resized_labels*.

Крок 9. Повернення списків *resized_data* та *resized_labels*.

Крок 10. Кінець алгоритму

4.4.3 Опис алгоритму створення та модифікації моделі

Враховуючи те, що для класифікації порід собак використовується заздалегідь навчена модель ResNet50, то алгоритм буде наступний:

Крок 1. Модель ResNet50 ініціалізується з наступними параметрами *weight* - *imagenet*, *include_top* - *False* (для видалення останнього шару), *pooling* – «avg» (глобальний шар усереднення пулінгу) та *input_shape* - очікує вхідні зображення форми (224, 224, 3)

Крок 2. Змінна *tenX* ініціалізується тензором виводу базової моделі.

Крок 3. До *tenX* додається щільний (повністю з'єднаний) шар з 512 нейронами до моделі.

Крок 4. Виконується функція активації ReLU (Rectified Linear Unit) до виводу попереднього шару.

Крок 5. Встановлення регуляризації випадання до моделі, випадково встановлюючи 50% входних одиниць на 0 на кожному оновленні під час тренування, щоб запобігти перенавчанню.

Крок 6. Додається щільний шар з кількістю нейронів, що дорівнює кількості класів і застосовується функція активації softmax до виводу. Це останній шар моделі, і його вивід буде розподілом ймовірностей по класах.

Крок 7. Створення нової моделі, яка включає як базову модель, так і додаткові шари, що було додано.

Крок 8. Замороження шарів базової моделі, крім останнього, щоб значення вагів не змінювались під час навчання.

Крок 9: Модель компілюється з оптимізатором стохастичного градієнтного спуску, функцією втрат категоріальної перехресної ентропії та точністю як метрикою оцінки.

Крок 10. Повернення скомпільованої моделі.

Крок 11. Кінець алгоритма.

4.4.4 Опис алгоритму тренування моделі

Перед тим, як почати тренування необхідно нормалізувати значення пікселів зображень з 0...255 до 0...1. Для цього використовується ImageDataGenerator від Keras. Також мітки з породами необхідно привести у вигляд двійкових векторів, що виконуються функцією flow_from_directory, що міститься у ImageDataGenerator. Тому алгоритм тренування моделі буде наступний:

Крок 1. Завантаження набору даних та зміна розміру алгоритмом описаним в розділі 4.4.2.

Крок 2. Набори зображень та міток розбиваються на тренувальні та валідаційні функцією train_test_split.

Крок 3. Створення векторів - міток для тренувального та валідаційного набору набору даних.

Крок 4. Нормалізація значень пікселів для валідаційного набору.

Крок 5. Доповнення даних для тренувального набору (для штучного збільшення набору). У тому числі: нормалізація значень пікселів, випадкове обертання та масштабування зображення, зсув, відображення по горизонталі.

Крок 6. Генерація даних для навчальної та тестової вибірки функцією `flow` з `ImageDataGenerator`.

Крок 7. Витягування ознак методом `.predict` з моделі, що була розроблена описана в розділі 4.3.4.

Крок 8. Створення зворотнього виклику для збереження найкращої моделі методом `ModelCheckpoint` з параметром `val_acc` для порівняння.

Крок 9. Створення зворотнього виклику методом `EarlyStopping` для зупинки навчання якщо метрика `val_acc` перестане покращуватись впродовж 4-х епох.

Крок 10. Навчання моделі методом `.fit`. В якості аргументів передаються ознаки моделі, кількість епох навчання, розмір вибірки після якої оновлюються ваги моделі, процент даних для валідації після кожної епохи та зворотні виклики з кроків 8 та 9.

```

training = model.fit(
    features,
    resizedLabels,
    batch_size=128,
    epochs=20,
    validation_split=0.2,
    callbacks=[checkpoint, early_stopping])

```

```

... Epoch 1/20
1/14 [====>.....] - ETA: 8s - loss: 4.4439 - accuracy: 0.0602
Epoch 1: val_accuracy improved from -inf to 0.48714, saving model to breed_cl_model.h5
14/14 [=====] - 2s 22ms/step - loss: 2.1993 - accuracy: 0.4236 - val_loss: 0.29885 - val_accuracy: 0.8987
Epoch 2/20
1/14 [====>.....] - ETA: 0s - loss: 0.6004 - accuracy: 0.8018
Epoch 2: val_accuracy improved from 0.89874 to 0.92469, saving model to breed_cl_model.h5
14/14 [=====] - 0s 9ms/step - loss: 0.3541 - accuracy: 0.8738 - val_loss: 0.1648 - val_accuracy: 0.9247
Epoch 3/20
13/14 [=====>....] - ETA: 0s - loss: 0.1981 - accuracy: 0.9013
Epoch 3: val_accuracy improved from 0.92469 to 0.93410, saving model to breed_cl_model.h5
14/14 [=====] - 0s 12ms/step - loss: 0.1933 - accuracy: 0.9105 - val_loss: 0.1422 - val_accuracy: 0.9341
Epoch 4/20
1/14 [====>.....] - ETA: 0s - loss: 0.1717 - accuracy: 0.9388
Epoch 4: val_accuracy improved from 0.93410 to 0.94217, saving model to breed_cl_model.h5
14/14 [=====] - 0s 8ms/step - loss: 0.1482 - accuracy: 0.9359 - val_loss: 0.1299 - val_accuracy: 0.9422
Epoch 5/20
1/14 [====>.....] - ETA: 0s - loss: 0.2522 - accuracy: 0.9004
Epoch 5: val_accuracy improved from 0.94217 to 0.94443, saving model to breed_cl_model.h5
14/14 [=====] - 0s 8ms/step - loss: 0.1170 - accuracy: 0.9444 - val_loss: 0.1213 - val_accuracy: 0.9444
Epoch 6/20
1/14 [====>.....] - ETA: 0s - loss: 0.1143 - accuracy: 0.9388
Epoch 6: val_accuracy improved from 0.94443 to 0.94894, saving model to breed_cl_model.h5
14/14 [=====] - 0s 8ms/step - loss: 0.1004 - accuracy: 0.9484 - val_loss: 0.1098 - val_accuracy: 0.9489
Epoch 7/20
...
Epoch 10/20
1/14 [====>.....] - ETA: 0s - loss: 0.0540 - accuracy: 0.9773
Epoch 10: val_accuracy did not improve from 0.94894
14/14 [=====] - 0s 6ms/step - loss: 0.0616 - accuracy: 0.9676 - val_loss: 0.1095 - val_accuracy: 0.9467

```

Рисунок 4.6 – Результати тренування моделі

На рис. 4.6 наведено результати тренування моделі, що набула максимальної точності на 6 епосі - 94.89% і зупинила навчання на 10й через відсутність покращення у метриці val_accrasy.

4.4.5 Опис алгоритму оцінки моделі

Для відображення ефективності моделі буде застосовано матрицю помилок (англ. confusion matrix). Алгоритм оцінки моделі наведено нижче:

Крок 1. Завантаження збереженої моделі.

Крок 2. На основі витягнутих з ResNet50 ознак зробити передбачення.

Крок 3. Отримання передбачення та конвертація його в мітки класу.

Крок 4. Конвертування справжніх міток набору даних у формат який відповідає прогнозованим міткам.

Крок 5. Створення матриці помилок на основі реальних міток та передбачених.

Крок 6. Налаштування та виведення матриці (рис 4.7).

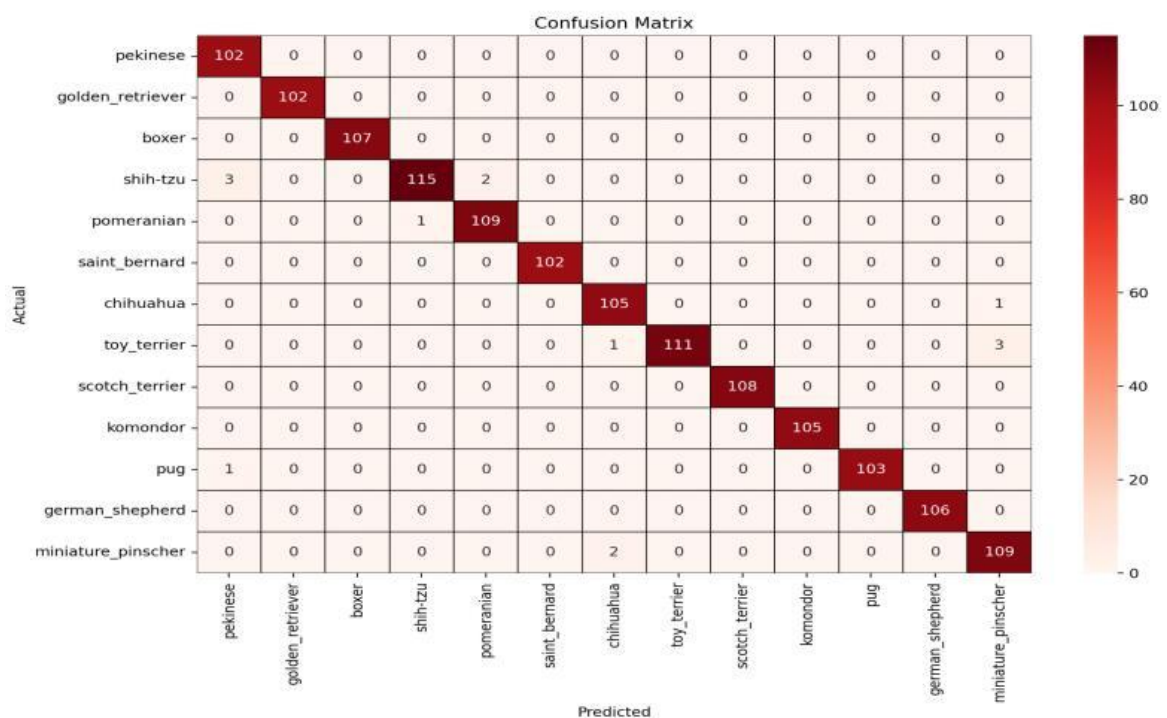


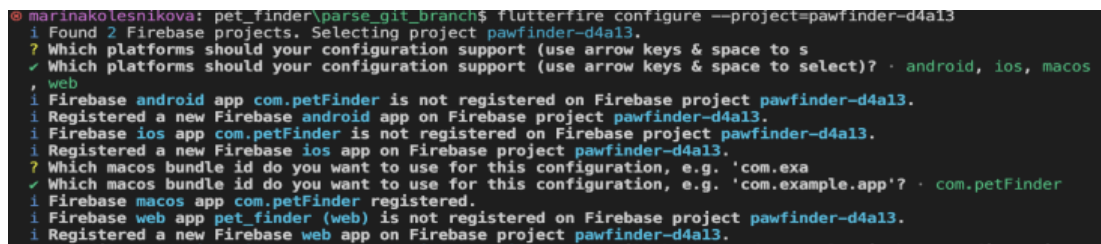
Рисунок 4.7 – Матриця помилок розробленої моделі

Як видно з рис. 4.7 модель є досить точною і правильно класифікує породи собак, проте іноді помиляються. Так, наприклад декілька зображень той-тер'єра були сплутані з мініатюрним пінчером.

4.5 Налаштування Firebase сервісів

Для налаштування Firebase сервісів було виконано ініціалізацію проєктів. Вона виконується набором команд:

- `$ curl -sL https://firebase.tools | bash` – встановлення Firebase SDK;
- `$ dart pub global activate flutterfire_cli` - активація командної строки Firebase;
- `$ flutterfire configure --project=pawfinder-d4a13` - конфігурація проєктів (рис. 4.8).



```
marinakolesnikova: pet_finder\parse_git_branch$ flutterfire configure --project=pawfinder-d4a13
i Found 2 Firebase projects. Selecting project pawfinder-d4a13.
? Which platforms should your configuration support (use arrow keys & space to s
✓ Which platforms should your configuration support (use arrow keys & space to select)? · android, ios, macos
, web
i Firebase android app com.petFinder is not registered on Firebase project pawfinder-d4a13.
i Registered a new Firebase android app on Firebase project pawfinder-d4a13.
i Firebase ios app com.petFinder is not registered on Firebase project pawfinder-d4a13.
i Registered a new Firebase ios app on Firebase project pawfinder-d4a13.
? Which macos bundle id do you want to use for this configuration, e.g. 'com.exa
✓ Which macos bundle id do you want to use for this configuration, e.g. 'com.example.app'? · com.petFinder
i Firebase macos app com.petFinder registered.
i Firebase web app pet_finder (web) is not registered on Firebase project pawfinder-d4a13.
i Registered a new Firebase web app on Firebase project pawfinder-d4a13.
```

Рисунок 4.8 – Конфігурація проєкту для кожної платформи

Якщо всі команди виконано без помилок, то для кожної платформи було створено власний проєкт і вони відобразяться у налаштуваннях, у блоку застосунків, що прив'язані до проєкту (рис. 4.9).

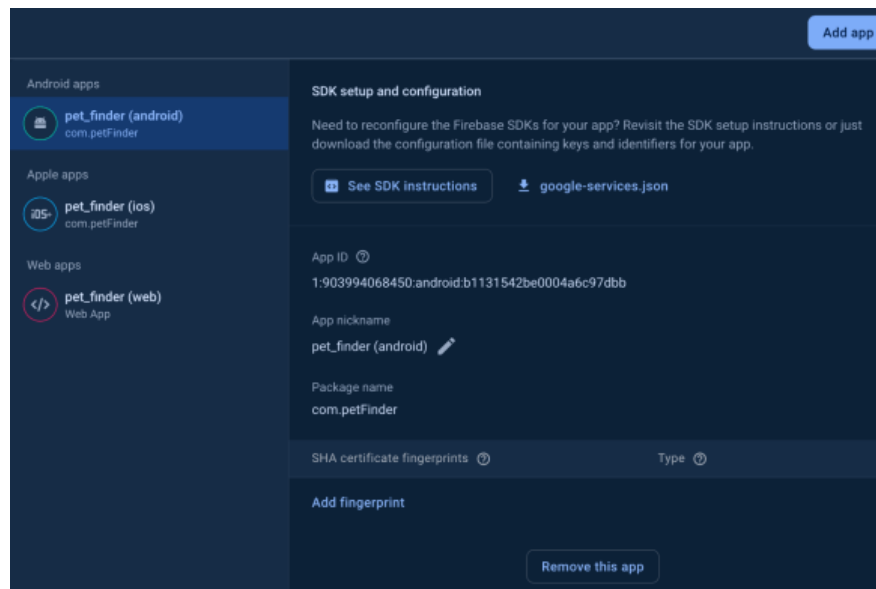


Рисунок 4.9 – Блок застосунків у налаштуваннях Firebase проєкту

Після успішного налаштування сховища, в застосунку необхідно встановити бібліотеки Cloud Firestore та Auth Firestore. Cloud Firestore надає арі для запитів у БД, а Auth Firestore авторизує користувачів. Також проініціалізувати сервіс `Firebase.initializeApp()` перед розгортанням програми.

Документи до бази даних (рис. 4.10) додаються безпосередньо через застосунок. Вони співпадають за структурою з моделями в застосунку, а обмін даними відбувається у форматі json.

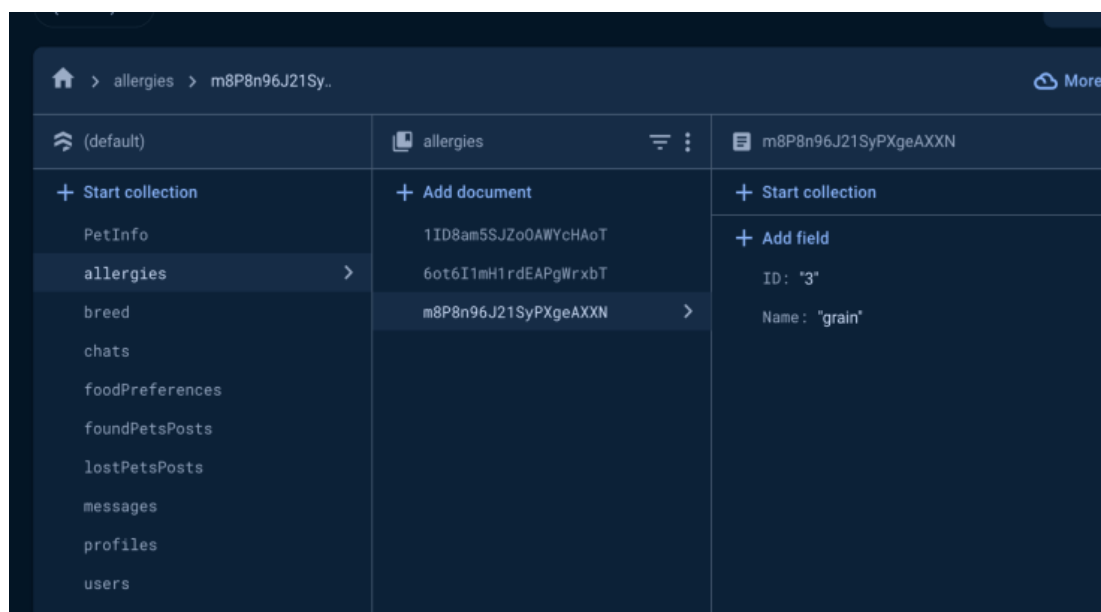


Рисунок 4.10 – Створені документи у сховищі

Враховуючи те, що FirestoreDatabase є нереляційною базою даних, то зв'язків між полями документів немає. Вони реалізовані лише на рівні моделей в застосунку. Причин для цього декілька: по-перше при необхідності міграції до іншої БД записи будуть прив'язані до id документів і не матимуть власних id, що може ускладнити перенесення даних. По-друге щоб створити посилання в одному документі на інший необхідно вказувати конкретну колекцію та id документа, що генерується автоматично. Для того, щоб не прив'язуватись до конкретного інтерфейсу потрібно оброблювати такі id та вирізати з них посилання на колекцію.

Для конфігурації push-нотифікацій необхідно створити функцію у Firebase Functions. Це можна зробити з консолі GoogleCloud і відбувається створення функції у 2 етапи: спочатку налаштовується регіон в якому користувачі будуть отримувати нотифікації та тригер-подія, наприклад, додавання або видалення документу до колекції (рис. 4.11).

The screenshot shows the Google Cloud console interface for creating a new function. The top navigation bar includes the Google Cloud logo, a user profile dropdown labeled 'PawFinder', and a search bar. Below the navigation bar, the 'Cloud Functions' section is active, with a 'Create function' button. The 'Configuration' tab is selected, showing the 'Basics' section. In the 'Basics' section, the 'Environment' is set to '1st gen', the 'Function name' is 'NewMessage', and the 'Region' is 'europe-central2 (Warsaw)'. The 'Trigger' section is expanded, showing the 'Cloud Firestore' trigger type, 'Event type' set to 'create', and 'Document Path' set to 'messages'.

Google Cloud PawFinder Search (/) for resources, do

Cloud Functions Create function

Configuration Code

Basics

Environment
1st gen

Function name *
NewMessage

Region *
europe-central2 (Warsaw)

Trigger

► Cloud Firestore

Trigger type
Cloud Firestore

Event type *
create

Document Path *
messages

Рисунок 4.11 – Налаштування тригерів та регіону

Другий етап – створення коду функції та тестування (рис. 4.12).

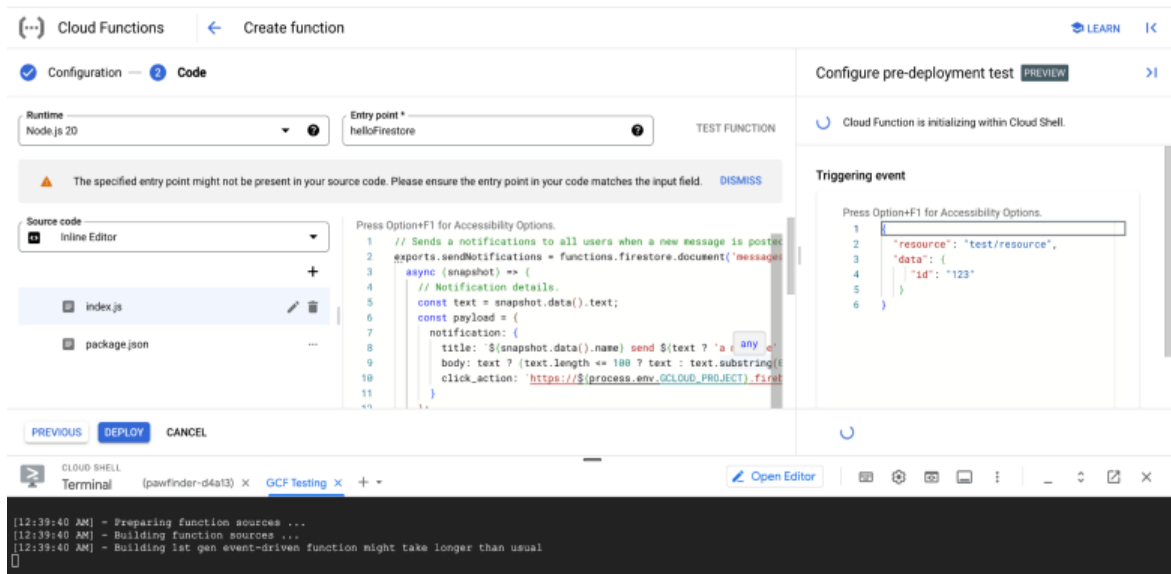


Рисунок 4.12 – Створення та тестування коду функції

Також необхідно налаштувати зовнішній вигляд push-нотифікації (рис. 4.13). Після вдалого проходження тесту можна виконати тестову нотифікацію на девайс (підставити id тестового девайса, що має FCMToken).



Рисунок 4.13 – Тестова нотифікація

В якості сховища для медіафайлів використовується Firebase Storage.

4.6 Схема даних застосунку

Як вже зазначалося раніше зв'язки між моделями даних реалізовані на рівні застосунку, схему зв'язків наведено на рис. 4.14.

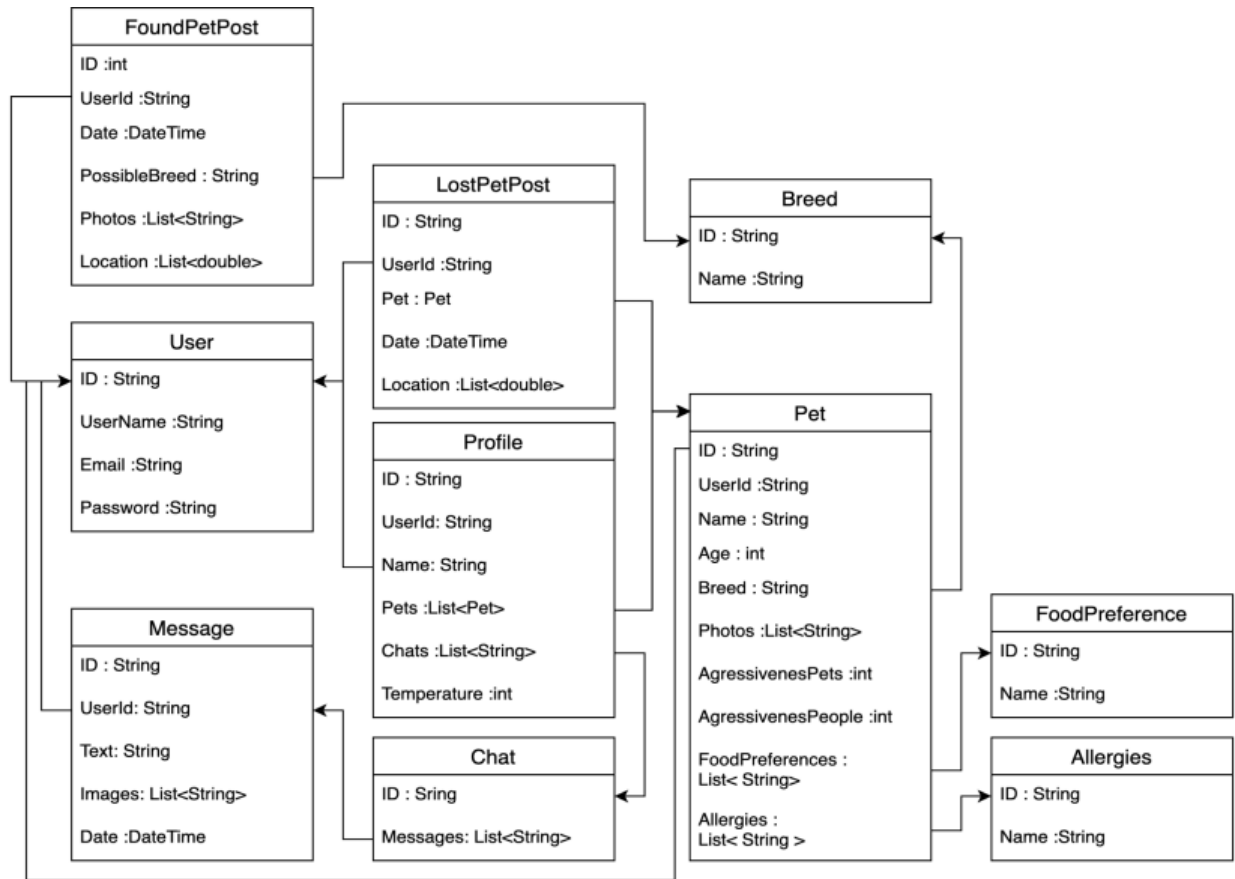


Рисунок 4.14 – Зв'язки моделей даних у застосунку

БД застосунку представлена такими сутностями:

- User – зберігає користувачів програми;
- Profile – зберігає дані про користувача, його домашніх тварин, чати;
- Messages – зберігає повідомлення одного користувача іншому;
- Chat – зберігає листування між двома користувачами;
- Pet – зберігає дані про домашню тварину, якою володіє користувач;
- LostPetPost – зберігає дописи про зниклу тварину;
- FoundPetPost – зберігає дописи про знайдену тварину;
- Breed – зберігає перелік порід собак;

- FoodPreference – зберігає перелік харчових вподобань улюбленця;
- Allergies – зберігає перелік алергій, що має домашня тварина;

Сутність FoundPetPost, що містить поля: ID, UserId, Date, PossibleBreed, Photos, Location наведено у табл. 4.1.

Таблиця 4.1 – Опис сутності FoundPetPost

№	Назва	Тип
1	ID	String
2	UserId	String
3	Date	DateTime
4	PossibleBreed	String
5	Photos	List<String>
6	Location	List<double>

Сутність User, що містить поля: ID, UserName, Email, Password наведено у табл. 4.2.

Таблиця 4.2 – Опис сутності User

№	Назва	Тип
1	ID	String
2	UserName	String
3	Email	String
4	Password	String

Сутність LostPetPost, що містить поля: ID, UserId, Date, Pet, Location наведено у табл. 4.3.

Таблиця 4.3 – Опис сутності LostPetPost

№	Назва	Тип
1	ID	String
2	Userld	String
3	Pet	Pet
4	Date	DateTime
5	Location	List<double>

Сутність Pet, що містить поля: ID, Userld, Name, Age, Breed, Photos, AgressivenesPet, AgressivenesPeople, FoodPreferences, Allergies наведено у табл. 4.4.

Таблиця 4.4 – Опис сутності Pet

№	Назва	Тип
1	ID	String
2	Userld	String
3	Name	String
4	Age	int
5	Breed	String
6	Photos	List<String>
7	AgressivenesPet	int
8	AgressivenesPeople	int
9	FoodPreferences	List<String>
10	Allergies	List<String>

Сутність Profile, що містить поля: ID, Userld, Name, Pets, Chats наведено у табл. 4.5.

Таблиця 4.5 – Опис сутності Profile

№	Назва	Тип
1	2	3
1	ID	String
2	Userld	String

Продовження таблиці 4.5

1	2	3
3	Name	String
4	Pets	List<String>
5	Chats	List<double>

Сутність Chat, що містить поля: ID та Messages наведено у табл. 4.6.

Таблиця 4.6 – Опис сутності Chat

№	Назва	Тип
1	ID	String
2	Messages	List<String>

Сутність Chat, що містить поля: ID та Name наведено у табл. 4.7.

Таблиця 4.7 – Опис сутності Breed

№	Назва	Тип
1	ID	String
2	Name	String

Сутності FoodPreference та Allergies мають однакову структуру з сутністю Breed.

Документи Breed, FoodPreference та Allergies є підставними. Вони стягуються зі сховища при завантаженні застосунку і використовуються користувачем при заповненні форм.

4.7 Інтерфейс застосунку

Завдяки тому, що Flutter контролює рендерінг об'єктів, інтефейс на платформах IOS та Android є ідентичним і не потребує стилізування нативних елементів окремо.

Інтерфейс користувача застосунку представлено екранами, що наведено нижче:

Екран авторизації користувача (рис. 4.15), що дозволяє як реєстрацію, так і вхід в існуючий запис і містить поля для вводу email та паролю, а також кнопки авторизації з допомоги Google та Facebook.

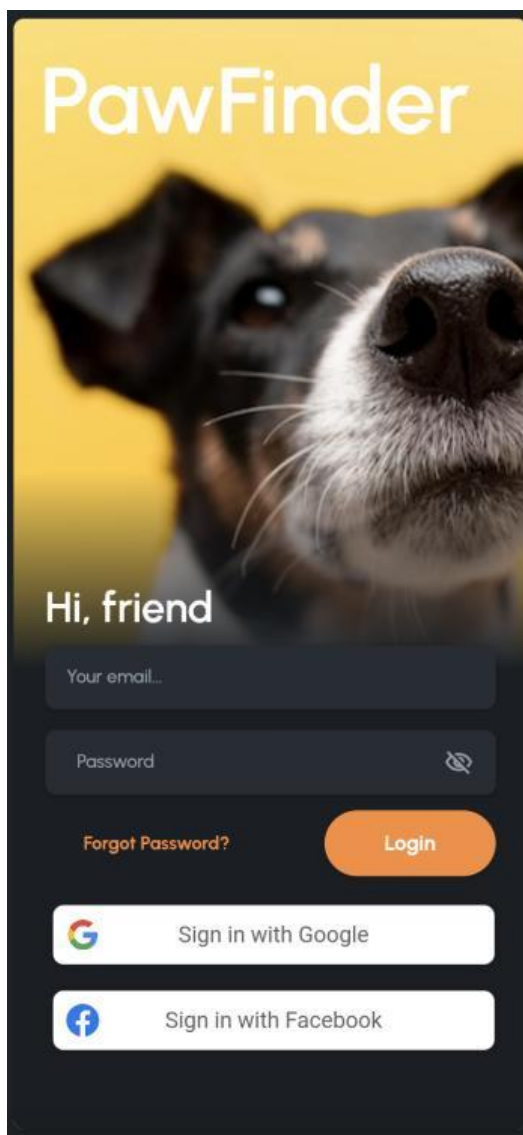


Рисунок 4.15 – Екран авторизації

Після авторизації користувач переходить на екран зі стрічкою новин, що у застосунку називається домашньою сторінкою. З неї за допомоги кнопки «+» можна додати оголошення, обравши його тип із спливаючого повідомлення (рис. 4.16).

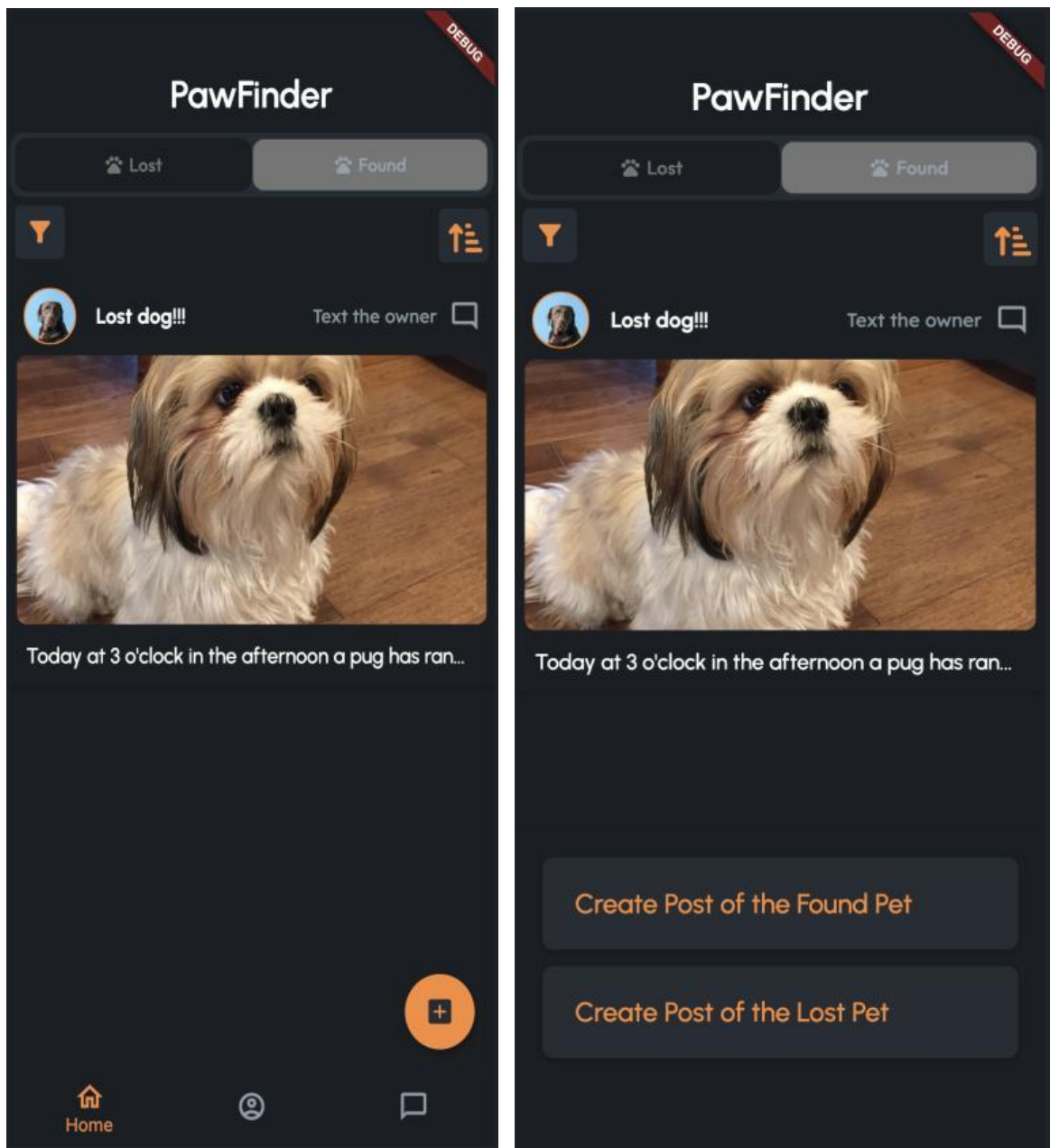


Рисунок 4.16 – Домашня сторінка та спливаюче повідомлення про вибір типу допису

При створенні оголошення користувачу відкриваються форми для створення запису або про втрачену, або про знайдену тварину. Ці екрани відображено на рисунках 4.17, 4.18.

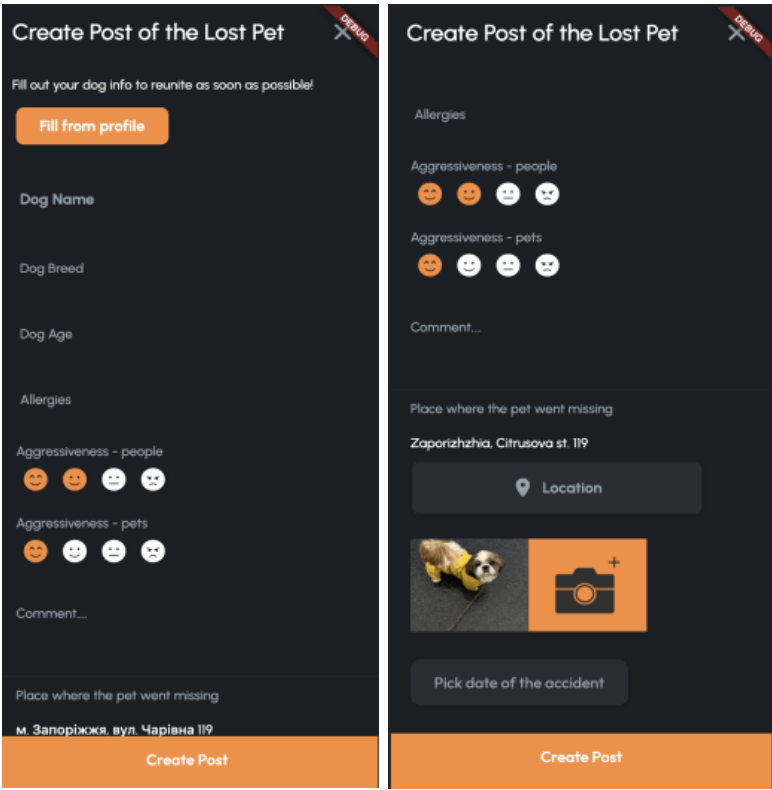


Рисунок 4.17 – Екран зі створенням допису про втрачену тварину

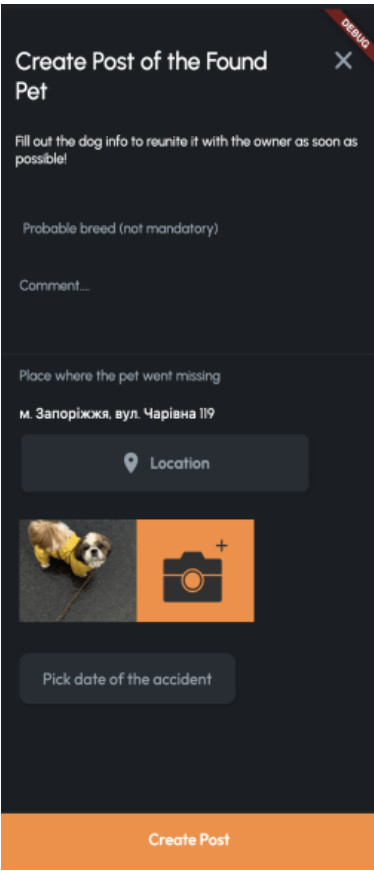


Рисунок 4.18 – Екран зі створенням допису про знайдену тварину

Після створення допису користувач побачить діалог про завантаження (рис 4.19). Якщо дописи зі схожими тваринами знайдено, то відображається екран зі списком (рис. 4.20).

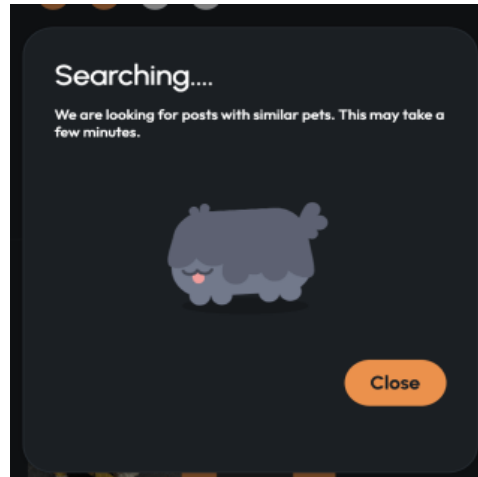


Рисунок 4.19 - Діалог про завантаження

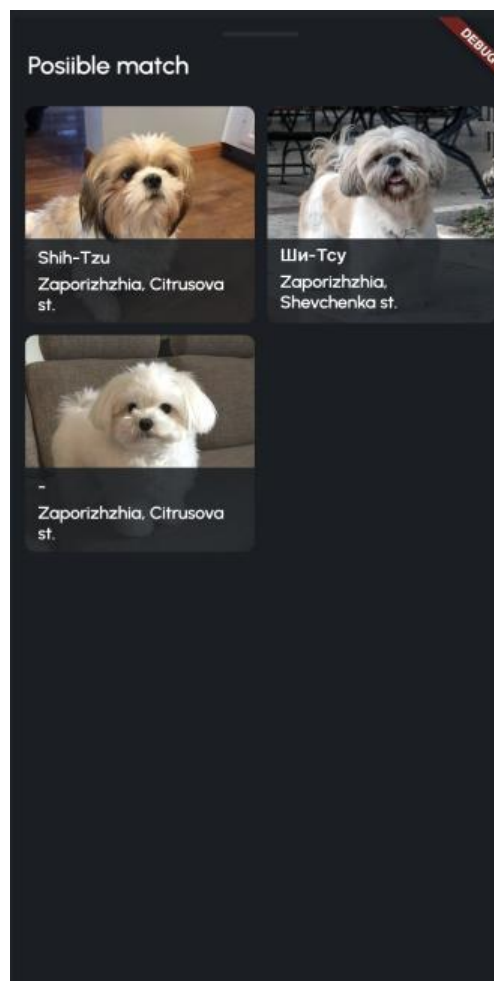


Рисунок 4.20 – Екран з дописами про схожих тварин

Користувач може перейти до допису для того, щоб подивитись деталі. Сторінку з деталями допису наведено на рис. 4.21.



Рисунок 4.21 – Екран з деталями допису

Навігація у застосунку здійснюється з допомоги віджета, що називається таб-бар (рис. 4.22).

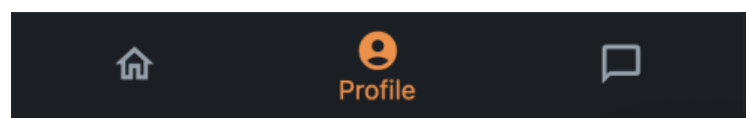


Рисунок 4.22 – Навігаційний віджет таб-бар

За допомоги таб бара користувач може переміщатись між вкладками застосунку. Такими як домашня сторінка, профайл, що містить записи про домашніх тварин та дописи, а також сторінка чатів (рис 4.23).

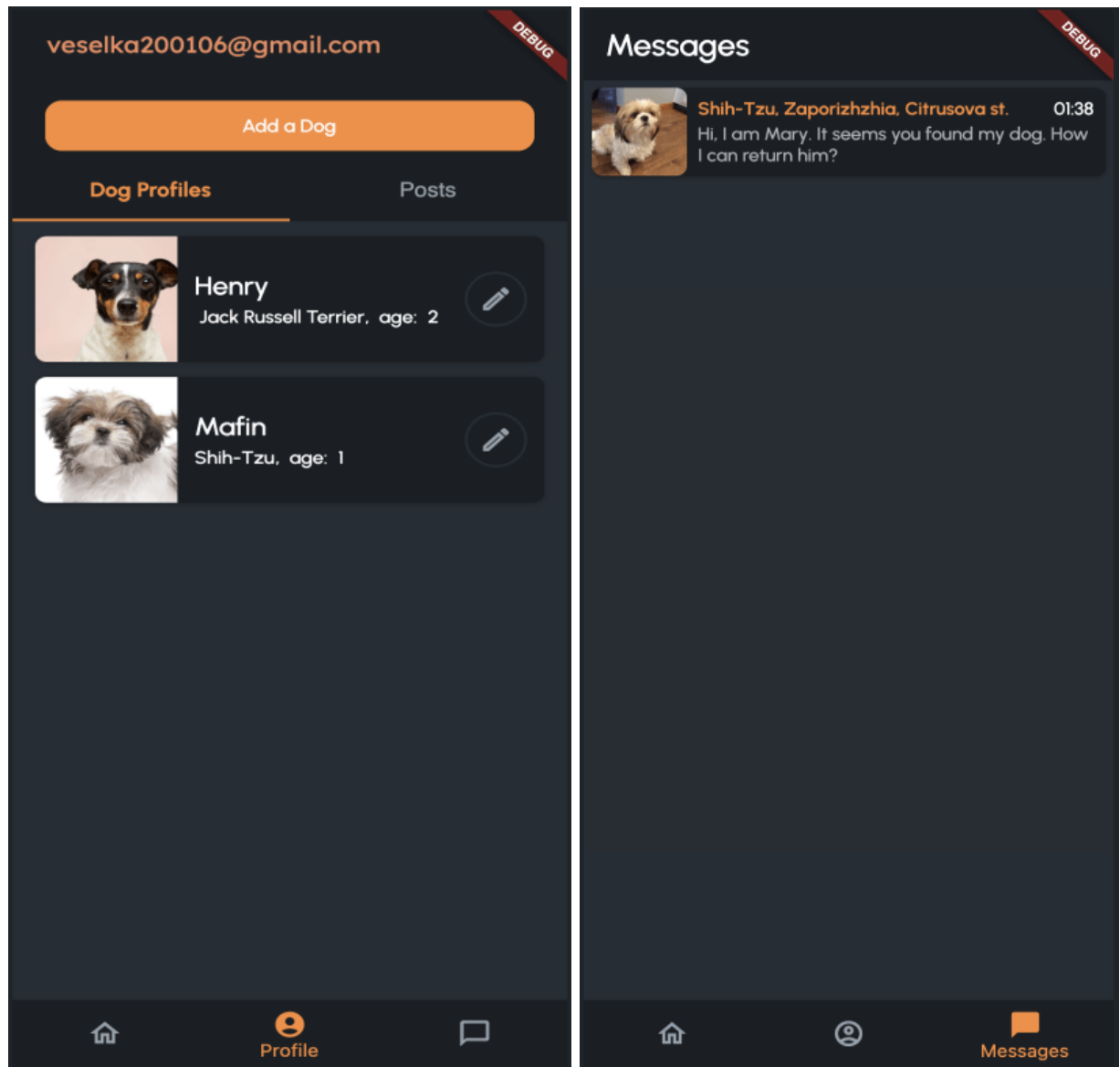


Рисунок 4.23 – Профайл та екран чатів користувача

На сторінці чатів відображені чати, які було створено при виборі користувача відгукнутися на допис і написати автору.

Щоб відобразити структуру навігації у застосунку, наведено схему архітктурної структури застосунку, що показує переходи між сторінками (рис. 4.24).

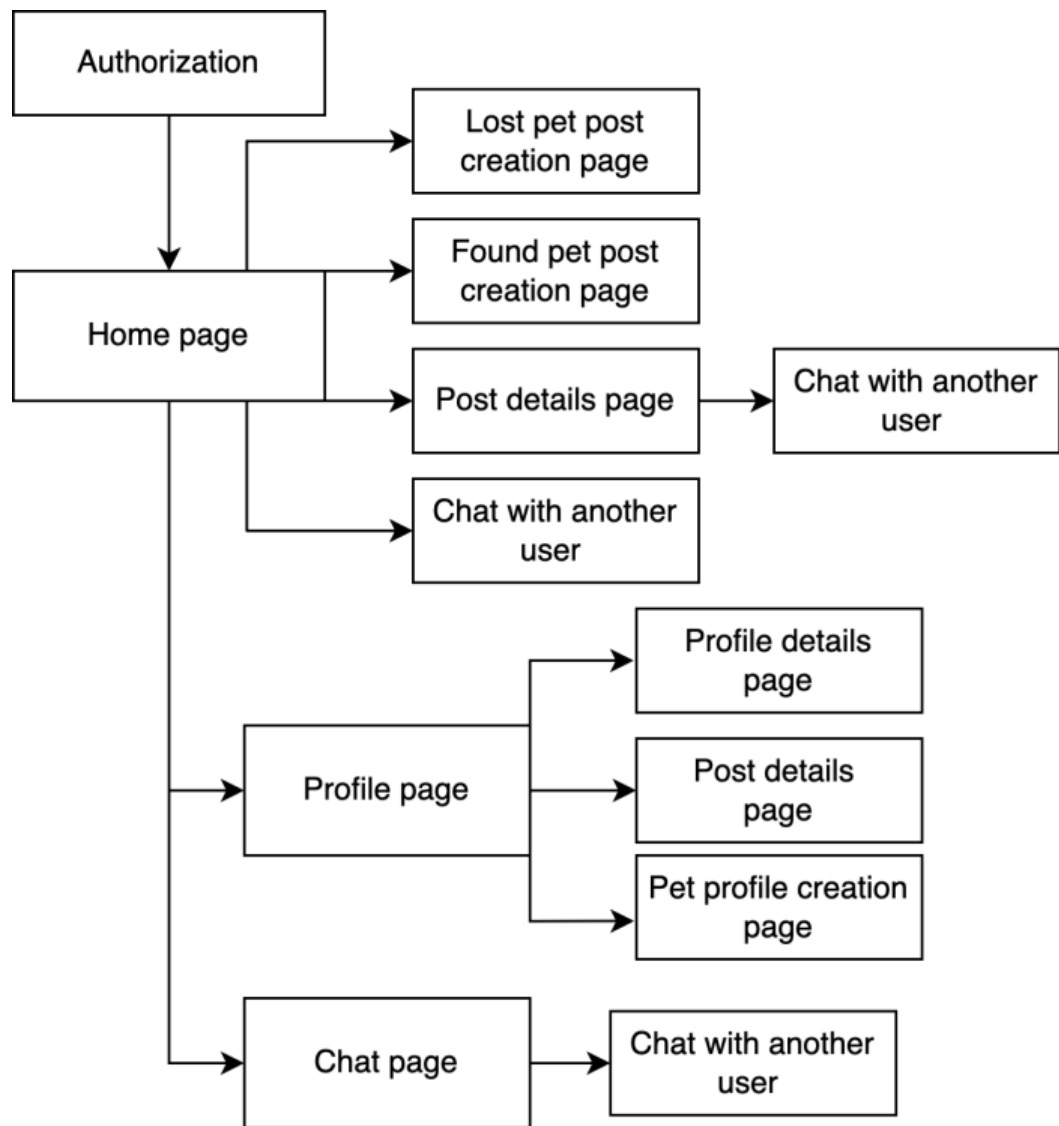


Рисунок 4.24 – Інформаційна архітектура застосунку

4.8 Висновки за розділом 4

В четвертому розділі було розглянуто структуру застосунків Python та Flutter та наведено алгоритми, що розроблено для обох програм. Проведено оцінку точності розробленої моделі, яку наглядно продемонстровано у матриці помилок розробленої моделі (рис. 4.7). Також було налаштовано сервіси Firebase для зберігання даних та push-нотифікацій та інтегровано їх у мобільний застосунок. Наведено схему даних застосунку.

Створено і описано інтерфейс користувацького застосунку та його компонентів, навігаційну структуру наведено на схемі інформаційної архітектури застосунку.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМ

5.1 Призначення програми та умови експлуатації

У попередніх розділах було розглянуто та розроблено модель нейронної мережі та мобільний застосунок, що використовує її як ядро для розпізнавання порід собак.

Створений застосунок призначено для користувачів, що мають смартфон з ОС Android чи IOS та домашню тварину. Варто зазначити, що наразі застосунок може розпізнавати лише собак, але в подальшому можливе його розширення для розпізнавання інших тварин.

Для стабільної роботи застосунку необхідно мати смартфон, що підтримує сервіси Google та має версію ОС Android не нижче 8.0, а IOS не нижче 10. Варто зазначити, що цільові версії операційних систем Android 11.0 та IOS 13.0. Девайс повинен мати не менше ніж 6 ГБ оперативної пам'яті та 150 МГБ вільної пам'яті. Розміри девайсу можуть варіюватись, але не бути меншими за 350x850. Також необхідний інтернет зв'язок та GPS.

5.2 Звернення до програми

Користувацький застосунок може бути встановлено на дві ОС : Android та IOS. Для смартфона з Android інсталяція може відбутися або з застосунку GooglePlay або шляхом завантаження .apk файлу на девайс та його подальшого розгортання. Після встановлення необхідно пройти авторизацію одним з наявних способів.

Для смартфонів IOS завантаження можливе лише з застосунку Appstore для кінцевих користувачів, або TestFlight для тестувальників. Будь-які інші способи є недоступними через обмеження накладені політикою безпеки IOS. Після завантаження застосунку на екрані з'являється нативне вікно, що запитає дозвіл на використання геолокації. Після успішної авторизації користувач

переходить на домашню сторінку застосунку з якої може додати запис натиснувши на кнопку, що зображено на рис. 5.1.

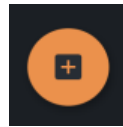


Рисунок 5.1 – Кнопка для додавання запису про загублену або знайдену тварину

Вибрати тип допису можна з меню, що відображено на рис. 5.2.

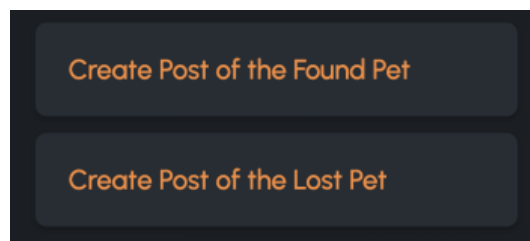


Рисунок 5.2 – Меню для вибору типу допису

Після заповнення форми допис можна зберегти натиснувши кнопку «Create Post» (рис. 5.3).



Рисунок 5.3 – Кнопка для збереження допису

Після того як допис успішно збережено, то його можна побачити у профілі користувача на вкладці «Posts» (рис. 5.4). Також можна відредагувати допис натиснувши на іконку олівця.

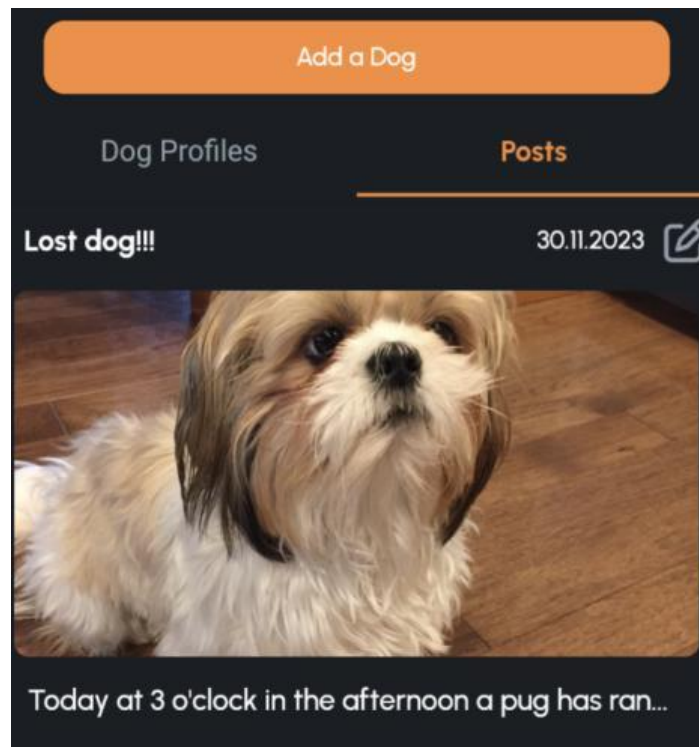


Рисунок 5.4 – Профіль користувача з відображенням дописом

Усі дописи можна побачити на домашній сторінці, а також відсортувати та відфільтрувати натиснувши на кнопки фільтра та сортування (рис. 5.5).



Рисунок 5.5 – Кнопки для виклику вікна фільтрації чи сортування

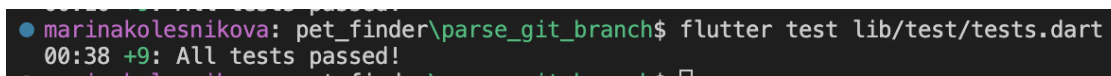
5.3 Тестування

Flutter дозволяє створювати три типи тестів: юніт-тести для покриття логіки, віджет-тести для покриття елементів інтерфейсу та інтеграційні – для тестування логіки та інтерфейсу разом. У застосунку використовуються лише юніт- та віджет- тести.

Наразі тестами покрити наступні компоненти:

- `FirebaseApiService` – тестування викликів та парсингу даних в моделі;

- AuthorizationBloc – тестування валідації даних та арі викликів;
 - PostBuilderBloc та HomePage – тестування відображення записів, сортування та фільтрація списків;
 - ProfileBloc – тестування додавання тварин та відображення дописів;
 - DogRecognitionService – тестування завантаження моделі та перевірка на тестовому зображенні;
 - ChattingBloc та ChatPage – додавання та відображення повідомлення.
- Результати наведено на рис. 5.6.



```

marinakolesnikova: pet_finder\parse_git_branch$ flutter test lib/test/tests.dart
00:38 +9: All tests passed!

```

Рисунок 5.6 – Результати тестування застосунку

Також було проведено тест за наступними чек-листами 5.1 – 5.3

Таблиця 5.1 – Чек-лист тестування профайлу

Тест-кейс	Android	IOS
Додавання запису про тварину до профайлу	+	+
Редагування запису про тварину до профайлу	+	+
Видалення запису про тварину	+	+
Відображення записів про тварин у вкладці 2	+	+
Відображення дописів у вкладці 1	+	+

Таблиця 5.2 – Чек-лист тестування домашньої сторінки

Тест-кейс	Android	IOS
1	1	2
Додавання запису про втрачену тварину	+	+
Додавання запису про знайдену тварину	+	+
Відображення нових записів у стрічці	+	+
Фільтрація записів	+	+
Сортування записів	+	+

Продовження таблиці 5.2

1	1	2
Відправка повідомлень іншому користувачу з стрічки	+	+
Відкриття допису про тварину	+	+
Відправка повідомлень іншому користувачу з допису	+	+

Таблиця 5.3 – Чек-лист тестування сторінки з повідомленнями

Тест-кейс	Android	IOS
Вибір та відкриття діалогу	+	+
Відправка повідомлення з текстом	+	+
Відправка повідомлення з текстом та зображенням	+	+
Отримання повідомлення з текстом	+	+
Отримання повідомлення з текстом та зображенням	+	+

5.4 Висновок за п'ятим розділом

У п'ятому розділі було описано програму, визначено умови експлуатації та мінімальні необхідні вимоги для платформ Android та IOS та зазначено наявні функції. Цільові версії операційних систем визначені як Android 11.0 та IOS 13.0.

Також було описано звернення до програми та описано роботу з основним функціоналом застосунку.

Для перевірки стабільності та надійності застосунку, було протестовано основні модулі системи за допомоги юніт- та віджет- тестів. Також було розроблено і пройдено тест кейси до основного функціоналу застосунку.

ВИСНОВОК

В результаті виконання дипломної роботи було проаналізовано предметну область і виділено основні поняття, а також розглянуто можливі підходи та існуючі методи для задач розпізнавання та класифікації. Було проведено порівняльний аналіз для класифікації та розпізнавання на основі якого було обрано класифікацію як підхід до вирішення завдання застосунку - розпізнавання втраченої тварини.

Також порівняно аналоги застосунків для пошуку домашніх тварин та доведено доцільність цієї розробки, що має вирішувати проблеми знайдені у аналогах.

Для застосунку було обрано технологію трансферного навчання на основі претренованої моделі ResNet завдяки наявній в ній можливості класифікувати 133 породи собак. Також було пояснено концепцію використання ResNet50 без останнього шару та використання додаткової моделі типу Sequential, що отримує на вхід виділені ResNet50 властивості з фото та повертає вірогідність співпадіння з породою.

У якості технології для розробки мобільного застосунку було обрано фреймворк Flutter, а в якості віддаленого сховища Firestore Database. А для реалізації push-нотифікацій використано Firebase Cloud Messaging.

В четвертому розділі було приділено увагу структурі застосунків Python та Flutter та наведено основні алгоритми. Проведено оцінку точності розробленої моделі, яку наглядно продемонстровано у матриці помилок розробленої моделі.

Створено і описано інтерфейс користувацького застосунку та його компонентів, навігаційну структуру наведено на схемі інформаційної архітектури застосунку.

Останнім етапом було тестування застосунку як за допомоги автоматичних юніт- та віджет- тестів, так і мануально за тест-кейсами.

У підсумку було розроблено мобільний застосунок, що відповідає заявленому функціоналу, а основним його призначенням є допомогти користувачу знайти його загублену тварину, або знайти власників для знайденої.

Наразі застосунок має можливість розпізнавати лише породи собак, тому в якості можливого покращення планується зробити модель для котів. А також на перспективу, можливе поєднання алгоритмів класифікації та розпізнавання для визначення не лише породи, а ідентифікації тварини.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Субботін, С. О. Нейронні мережі : теорія та практика: навчальний посібник / С. О. Субботін. – Житомир, 2020. – С. 8.
2. Shang-Hung, L. An Introduction to Face Recognition Technology / L. Shang-Hung // Informing Science The International Journal of an Emerging Transdiscipline – 2000. – P. 84–86.
3. Gu, L. Encyclopedia of Biometrics: Face Alignment / L. Gu, T. Kanade. – Cham : Springer, 2009. – P. 291–294.
4. Face Detection using Viola Jones Algorithm. [Electronic resource] / GreatLearning. – Access mode: <https://www.mygreatlearning.com/blog/viola-jones-algorithm/#sh23>.
5. Face Detection Explained: State-of-the-Art Methods and Best Tools. [Electronic resource] / Medium. – Access mode: <https://medium.com/sciforce/face-detection-explained-state-of-the-art-methods-and-best-tools-f730fca16294>.
6. Orman, Z. A Study On Face, Eye Detection And Gaze Estimation / Z. Orman, A. Battal, E. Kemmer // International Journal of Computer Science & Engineering Survey. – 2011. – №3. – P. 29–46.
7. Yuille, A. L. Deformable Templates for Face Recognition / A. L. Yuille // Journal of Cognitive Neuroscience. – 1991. – №1. – P. 59–70.
8. How Do Eigenfaces Work? [Electronic resource] / Baeldung. – Access mode: <https://www.baeldung.com/cs/cv-eigenfaces>.
9. ML | Face Recognition Using Eigenfaces (PCA Algorithm) [Electronic resource] / GeeksForGeeks. – Access mode: <https://www.geeksforgeeks.org/ml-face-recognition-using-eigenfaces-pca-algorithm>.
10. Lossless Triplet loss [Electronic resource] / Medium. – Access mode: <https://towardsdatascience.com/lossless-triplet-loss-7e932f990b24>.
11. Soofi, A. A. Classification Techniques in Machine Learning: Applications and Issues / A. A. Soofi, A. Awan // Journal of Basic & Applied Sciences. – 2017. – №13. – P. 459–465.

12. Classification in Machine Learning: An Introduction [Electronic resource] / DataCamp. – Access mode: <https://www.datacamp.com/blog/classification-machine-learning>
13. Kumari, R. Machine Learning: A Review on Binary Classification / R. Kumari, S. K. Srivastava // International Journal of Computer Applications – 2017. – №70. – P. 11–15.
14. Alsafy, B. M. Multiclass Classification Methods: A Review / B. M. Alsafy, Z. Mosad, W. K. Mutlag // International Journal of Advanced Engineering Technology and Innovative Science. – 2020. – №5. – P. 1–10.
15. A Gentle Introduction to Imbalanced Classification [Electronic resource] / Machine Learning Mastery. – Access mode: <https://machinelearningmastery.com/what-is-imbalanced-classification/#:~:text=Imbalanced%20classification%20is%20the%20problem,and%20may%20require%20specialized%20techniques.>
16. Naïve Bayes classifiers [Electronic resource] / IBM. –Access mode: <https://www.ibm.com/topics/naive-bayes#:~:text=The%20Na%C3%AFve%20Bayes%20classifier%20is,a%20given%20class%20or%20category.>
17. K-Nearest Neighbors Algorithm [Electronic resource] / IBM. –Access mode: <https://www.ibm.com/topics/knn#:~:text=Next%20steps-,K%2DNearest%20Neighbors%20Algorithm,of%20an%20individual%20data%20point.>
18. Awad, M. Support Vector Machines for Classification / M. Awad, R. Khanna. – Efficient Learning Machines, 2015. – P. 39–66.
19. Classification in Machine Learning: What it is & Classification Models [Electronic resource] / SimpliLearning. – Access mode: <https://www.simplilearn.com/tutorials/machine-learning-tutorial/classification-in-machine-learning.>
20. Javidi, B. Image Recognition and Classification. Algorithms, Systems, and Applications / B. Javidi. – NewYork: Basel, 2002. – 520 p.
21. NOSEID [Electronic resource] / Google Play. – Access mode: <https://play.google.com/store/apps/details?id=com.mars.noseid.>

22. WAUDOG [Electronic resource] / Google Play. – Access mode: <https://play.google.com/store/apps/details?id=com.collar.waudog.europe&hl=en&gl=US>.

23. MissingPets [Electronic resource] / Google Play. – Access mode: <https://play.google.com/store/apps/details?id=com.pet.missingpets&hl=en&gl=US>.

24. Transfer Learning Using Keras and Tensorflow PyData [Electronic resource] / A. Kapoor. – Access mode: <https://pydata.org/delhi2017/speaker/profile/59/>

25. A Gentle Introduction to Dropout for Regularizing Deep Neural Networks [Electronic resource] / Machine Learning Mastery. – Access mode: <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks>.

26. Top 30 Python Libraries To Know in 2023 [Electronic resource] / MyGreatLearning. – Access mode: <https://www.mygreatlearning.com/blog/open-source-python-libraries>.

27. 9 Best Python Libraries for Machine Learning [Electronic resource] / Coursera. – Access mode: <https://www.coursera.org/articles/python-machine-learning-library>.

28. C++ vs Java vs Python [Electronic resource] / GeeksForGeeks. – Access mode: <https://www.geeksforgeeks.org/c-vs-java-vs-python>.

29. PyCharm vs. Spyder: Choosing the Right Python IDE [Electronic resource] / UniteAI. – Access mode: <https://www.unite.ai/pycharm-vs-spyder-choosing-the-right-python-ide>.

30. User Guide: Text Editor [Electronic resource] / JupyterLab – Access mode: https://jupyterlab.readthedocs.io/en/stable/user/file_editor.html.

31. Перші кроки в NLP: розглядаємо Python-бібліотеку scikit-learn в реальному завданні [Електронний ресурс] / DOU – Режим доступу: <https://dou.ua/lenta/articles/first-steps-in-nlp-scikit-learn>.

32. Scikit-learn vs. TensorFlow vs. PyTorch vs. Keras [Electronic resource] / Ritza – Access mode: <https://ritza.co/articles/scikit-learn-vs-tensorflow-vs-pytorch-vs-keras>.

33. Choose a Database: Cloud Firestore or Realtime Database [Electronic resource] / Firebase – Access mode: <https://firebase.google.com/docs/database/rtdb-vs-firestore>.

34. flutter_redux [Electronic resource] / pub.dev – Access mode: https://pub.dev/packages/flutter_redux.

35. provider [Electronic resource] / pub.dev – Access mode: <https://pub.dev/packages/provider>.

36. flutter_bloc [Electronic resource] / pub.dev – Access mode: https://pub.dev/packages/flutter_bloc.

ДОДАТОК А
Технічне завдання

Вступ

Значення домашніх тварин у нашому житті важко переоцінити. Для власників домашніх тварин емоційний зв'язок є глибоким, даруючи комфорт, любов і мовчазне розуміння, яке перевершує будь-які слова. Проте, незважаючи на радість, яку приносять домашні тварини, сумна реальність полягає в тому, що іноді вони зникають. Актуальним стає створення мобільного застосунку, який би міг розпізнати тварину та зіставити її з загубленою, а також надати контакти власника. Програма призначена спростити та прискорити процес возз'єднання загублених собак із їх сім'єю.

A.1 Назва програми

Мобільний застосунок називається "PawFinder".

A.2 Підстави для розробки

Підставою для розробки є наказ №437 від 14 листопада 2023 р. по Національному університету "Запорізька політехніка", що затверджує завдання на дипломну кваліфікаційну роботу магістра з теми: "Мобільний застосунок для розпізнавання загублених тварин".

A.3 Призначення розробки

Застосунок призначено для розпізнавання породи собаки, знаходження відповідностей між записами про породу та містом де загублено тварину. Користувач отримує можливі співпадиння, якщо такі існують у вигляді дописів інших користувачів. Також є можливість почати чат з іншим користувачем для більш детальної інформації.

A.4 Вимоги до програми

A.4.1 Вимоги до функціональних характеристик

Програма для розпізнавання загублених тварин має бути виконана у вигляді мобільного застосунку, до якого висунуті наступні функціональні вимоги:

- авторизація та реєстрація за допомоги електронної поштової адреси, та сервісів Google, Facebook та Apple sign In (тільки для операційних систем ОС IOS);
- створення, зберігання та редагування записів про загублену домашню тварину з наступними параметрами: прізвисько, порода, фотокартки, адреса де було загублено тварину;
- розпізнавання тварини з фотокартки;
- видалення запису про загублену тварину;
- чат між власником та людиною, що знайшла домашню тварину;
- профайл тварини з наступними даними: прізвисько, порода, вік, фото, місто проживання, алергії, харчові вподобання, агресивність до людей та інших тварин;
- зберігання запису про загублену тварину у профайлі;
- стрічка новин з загубленими та знайденими тваринами з можливістю фільтрації за містом, породою, сортуванням за датою;
- сповіщення про знайдену схожу тварину.

A.4.2 Вимоги до надійності

Застосунок має стабільно працювати, оброблювати некоректні вхідні дані та повідомляти користувача про них. Також застосунок має оброблювати помилки, що викликані внутрішніми процесами програми, наприклад арі викликів за відсутності інтернету і не переривати роботи застосунку.

Користувач має бути повідомлений про операції, що потребують часу за допомоги відображення елементів інтерфейсу, а також мати можливість скасовувати операції.

A.4.3 Умови експлуатації

Для стабільної роботи застосунку необхідно мати смартфон, що підтримує сервіси Google та має версію ОС Android не нижче 8.0, а IOS не нижче 10. Варто зазначити, що цільові версії операційних систем Android 11.0 та IOS 13.0. Девайс повинен мати не менше ніж 6 ГБ оперативної пам'яті та 150 МГБ вільної пам'яті. Розміри девайсу можуть варіюватись, але не бути меншими за 350x850. Також необхідний інтернет зв'язок та GPS.

A.5 Стадії і етапи розробки

A.5.1 Стадії розробки

Для розробки застосунку було пройдено наступні стадії: аналіз вимог, аналіз існуючих підходів, проєктування, розробка та тестування програми та розробка документації.

A.5.2 Етапи розробки

На стадії аналізу вимог було виконано їх збір, аналіз та систематизування, а також визначення та розв'язання суперечностей. На стадії аналізу існуючих підходів було проаналізовано готові існуючі програмні засоби, метода та підходи до вирішення проблеми.

На стадії проєктування виконується аналіз існуючих технологій та визначення архітектури застосунку, проєктування бази даних та інтерфейсу користувача.

На стадії розробки та тестування відбувається налагодження необхідних сервісів, створення програмного коду та тестування застосунку.

На стадії розробки документації відбувається розробка опису програми та супровідної документації.

A.6 Порядок контролю та приймання

Порядок контролю та приймання визначено згідно до порядку приймання дипломних кваліфікаційних робіт магістрів. Застосунок має відповідати усім вимогам, що заявлені у розділі A.4.

ДОДАТОК Б
Текст програми

Б.1 Текст файла preprocess_data.py для Python проекта

```

from keras.preprocessing.image import ImageDataGenerator
import numpy as np
import os
import skimage as sk
from tqdm import tqdm

PATH = '../data/breeds/'

def dataResizing(filenameesList, labelsList,
IMGShape=[224,224,3]):
    fileListR= np.empty([0] + IMGShape)
    labelsListR = np.empty((0))
    for i in tqdm(range(len(filenameesList))):
        image = sk.io.imread(filenameesList[i])
        tqdm.write(str(np.shape(image)))
        if len(np.shape(image))==3:
            imageResized = sk.transform.resize(image, IMGShape,
mode='reflect')
            fileListR = np.append(fileListR,
np.expand_dims(imageResized, axis=0), axis=0)
            labelsListR = np.append(labelsListR, labelsList[i])

    return fileListR, labelsListR

def saveResized(filenameesList,
labelsList, path='../data/breeds/'):
    path_resized = path + 'resized/'
    for i in range(len(filenameesList)):
        sk.io.imsave(path_resized + str(i) + '.jpg',
filenameesList[i])
    np.save(path_resized + 'labels.npy', labelsList)

```

Б.2 Текст файла model_creation_RESNET_2.py для Python проекта

```

import os
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
from process.training_evaluation_model import evaluation,
training
import seaborn as sns
import cv2
from tensorflow.keras.models import Sequential
from keras.layers import Dense,Input,Dropout
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import GlobalAveragePooling2D

```

```

from tensorflow.keras.callbacks import EarlyStopping,
ModelCheckpoint
from tensorflow.keras.models import Model

IMG_SIZE = 224
NUM_CLASSES = 120
IMG_SHAPE = (IMG_SIZE, IMG_SIZE, 3)
RSD_IMG_PATH = '../data/breeds/resized/'

df = pd.read_csv('../data/breeds/resized_labels.csv')
df.head()

RSDImg = []
for idx, filename in enumerate(df['id']):
    img_path = os.path.join(RSD_IMG_PATH, filename + '.jpg')
    resized_img = cv2.imread(img_path)
    RSDImg.append(resized_img)
RSDImg_array = np.array(RSDImg)

inputLayer = Input(IMG_SHAPE)
resnet50Processing =
tf.keras.applications.resnet50.preprocess_input
preprocessor = resnet50Processing(inputLayer)
baseModel = ResNet50(weights='imagenet', include_top=False,
input_shape=input_size)(preprocessor)
avg = GlobalAveragePooling2D()(baseModel)
extraction = Model(inputs=inputLayer, outputs=avg)
features = extraction.predict(RSDImg_array, verbose=1)

def createModelOnResNet50():

    newModel = Sequential()
    newModel.add(
        Dropout(0.7,
                input_shape=(features.shape[1],),),)
    newModel.add(
        Dense(120,
              activation='softmax'))
    newModel.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy'])

    return(newModel)

training(features, df['breed'])
evaluation(features, df['breed'])

```

Б.3 Текст файла training_evaluation_model.py проекта Python

```

from models.model_creation_RESNET_2 import createModelOnResNet50
from tensorflow.math import confusion_matrix
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

import pandas as pd
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import EarlyStopping,
ModelCheckpoint
from tensorflow.keras.models import Model

def training(features, labels):
    modelCheckpoint = ModelCheckpoint(
        '../models/breeds_recognition_model.h5',
        monitor='val_accuracy',
        save_best_only=True,
        verbose=1,
        mode='max')

    earlyStopping = EarlyStopping(
        monitor='val_accuracy',
        patience=4,
        restore_best_weights=True)

    createModelOnResNet50().fit(
        features,
        labels,
        batch_size=128,
        epochs=18,
        validation_split=0.2,
        callbacks=[modelCheckpoint, earlyStopping])

def evaluation(features, labels):
    modelBr = load_model(
        '../models/breeds_recognition_model.h5', )

    prediction = modelBr.predict(features)
    predictedLabels = np.argmax(prediction, axis=1)
    correctLabels = np.argmax(labels, axis=1)

    conf_matrix = confusion_matrix(correctLabels,
predictedLabels)
    plt.figure(figsize=(15, 15))
    sns.heatmap(conf_matrix, annot=True, fmt="d", cmap='Reds',
linecolor='black', linewidths=0.1)
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')

```

```
plt.ylabel('Actual')
plt.show()
```

Б.4 Текст файла `firebase_api_service.dart` проекта Flutter

```
class FirebaseApiService {
  final FirebaseFirestore _firestore =
    FirebaseFirestore.instance;

  Future<void> createUser(String userId, Map<String, dynamic>
    userData) async {
    await
    _firestore.collection('users').doc(userId).set(userData);
  }

  Future<Map<String, dynamic>> getUser(String userId) async {
    final userDoc = await
    _firestore.collection('users').doc(userId).get();
    return userDoc.data() ?? {};
  }

  Future<void> updateUser(String userId, Map<String, dynamic>
    updatedData) async {
    await
    _firestore.collection('users').doc(userId).update(updatedData);
  }

  Future<void> deleteUser(String userId) async {
    await _firestore.collection('users').doc(userId).delete();
  }

  Future<void> createPet(String petId, Map<String, dynamic>
    petData) async {
    await _firestore.collection('pets').doc(petId).set(petData);
  }

  Future<Map<String, dynamic>> getPet(String petId) async {
    final petDoc = await
    _firestore.collection('pets').doc(petId).get();
    return petDoc.data() ?? {};
  }

  Future<void> updatePet(String petId, Map<String, dynamic>
    updatedData) async {
    await
    _firestore.collection('pets').doc(petId).update(updatedData);
  }

  Stream<List<String>> getAllBreeds() {
    return
    _firestore.collection('breeds').snapshots().map((snapshot) {
```

```

        return snapshot.docs.map((doc) => doc.data()['name'] as
String).toList();
    });
}

Future<void> createProfile(String profileId, Map<String,
dynamic> profileData) async {
    await
_firestore.collection('profiles').doc(profileId).set(profileData
);
}

Future<Map<String, dynamic>> getProfile(String profileId)
async {
    final profileDoc = await
_firestore.collection('profiles').doc(profileId).get();
    return profileDoc.data() ?? {};
}

Future<void> updateProfile(String profileId, Map<String,
dynamic> updatedData) async {
    await
_firestore.collection('profiles').doc(profileId).update(updatedD
ata);
}

Future<void> deleteProfile(String profileId) async {
    await
_firestore.collection('profiles').doc(profileId).delete();
}

...
//code was cropped
}
}

```

Б.5 Текст файла found_post_service.dart проекта Flutter

```

import 'package:pet_finder/data/api/FirebaseApiService.dart';
import
'package:pet_finder/entity/models/post_models/found_pet_model.da
rt';

class FoundPetPostService {
    final FirebaseApiService _apiService = FirebaseApiService();

    Future<void> createPost(FoundPetPost post) async {
        await _apiService.createPost(post.id, post.toJson());
    }

    Future<FoundPetPost> getPost(String postId) async {

```

```

    final postData = await _apiService.getPost(postId);
    return FoundPetPost.fromJson(postData);
  }

  Future<void> updatePost(FoundPetPost post) async {
    await _apiService.updatePost(post.id, post.toJson());
  }

  Future<List<FoundPetPost>> getPostByBreedLocation(String breed,
String location) async {
    final postDataList = await
_apiService.getPostByBreedLocation(breed, location);
    return postDataList.map((postData) =>
FoundPetPost.fromJson(postData)).toList();
  }

  Future<void> deletePost(String postId) async {
    await _apiService.deletePost(postId);
  }
}

```

Б.6 Текст файла dog_recognition_service.dart проекта Flutter

```

import
'package:pet_finder/data/api/modelServices.dart/found_post_servi
ce.dart';
import
'package:pet_finder/data/api/modelServices.dart/lost_post_servic
e.dart';
import 'package:tflite/tflite.dart';

class DogRecognitionService {
  Map<String, double> breedCount = {};
  final LostPetPostService _lostPetService =
LostPetPostService();
  final FoundPetPostService _foundPetService =
FoundPetPostService();

  Future<void> loadModel() async {
    await Tflite.loadModel(
      model: "assets/model.tflite",
      labels: "assets/labels.txt",
    );
  }

  Future<List<String>> breedClassifier(List<String> images,
String location, String probableBreed, DateTime lostDate,
PostType postType,) async {
    await loadModel();
    if (probableBreed != null) {
      breedCount[probableBreed] = 1.0;
    }
  }
}

```

```

    }
    for (var i = 0; i < images.length; i++) {
      var output = await Tflite.runModelOnImage(
        path: images[i],
        numResults: 2,
      );
      output.forEach((element) {
        breedCount[element['label']] =
        breedCount.containsKey(element['label'])
          ? breedCount[element['label']] +
            element['confidence']
          : element['confidence'];
      });
    }
    var sortedKeys = breedCount.keys.toList(growable:false)
      ..sort((k1, k2) =>
        breedCount[k2].compareTo(breedCount[k1]));
    Map<String, double> sortedMap = new
    Map.fromIterable(sortedKeys, key: (k) => k, value: (k) =>
    breedCount[k]);

    Set<String> postIds = {};
    for (var breed in sortedMap.keys) {
      var lostPetPosts = await
    _lostPetService.getPostsByBreedAndLocation(breed, location);
      var foundPetPosts = await
    _foundPetService.getPostsByBreedAndLocation(breed, location);
      postIds.addAll(lostPetPosts.map((post) => post.id));
      postIds.addAll(foundPetPosts.map((post) => post.id));
    }
    return postIds;
  }
}

```

Б.7 Текст файла found_pet_post_model.dart проекта Flutter

```

import 'package:freezed_annotation/freezed_annotation.dart';
import

part 'content_object.freezed.dart';
part 'content_object.g.dart';

@freezed
class FoundPetPost with _$FoundPetPost {
  factory FoundPetPost({
    required int id,
    required int userId,
    required DateTime date,
    required String possibleBreed,
    required List<String> photos,
    required List<double> location,

```

```
    }) = _FoundPetPost;

    factory FoundPetPost.fromJson(Map<String, dynamic> json) =>
    _$FoundPetPostFromJson(json);
    Map<String, dynamic> toJson() => _$FoundPetPostToJson(this);
  }
```

ДОДАТОК В
Слайди презентації

Мобільний застосунок для розпізнавання загублених тварин

Дипломна робота

Виконала:
ст. гр. КНТ-132м

М. В. Колесникова

Керівник:
к. т. н. доцент

Т. В. Федорчук

Рисунок В.1 – Слайд 1

Мета роботи

- створення кросплатформного мобільного застосунку для пошуку загублених тварин "PawFinder"

Об'єкт дослідження

- Розпізнавання тварини з застосуванням штучного інтелекту.

Предмет дослідження

- Кросплатформний мобільний застосунок для пошуку загублених тварин "PawFinder"

2

Рисунок В.2 – Слайд 2

Постановка завдання

Завдання дипломної роботи, це програмна реалізація мобільного застосунку для пошуку загублених тварин.

Вимоги до застосунку:

- авторизація та реєстрація користувача за адресою електронної скриньки або сторонніх сервісів;
- створення, зберігання, редагування та видалення записів про загублену або знайдену домашню;
- розпізнавання тварини з фотокартки;
- чат між власником та людиною, що знайшла домашню тварину;
- створення, редагування та видалення профайлу улюбленця;
- стрічка новин з загубленими та знайденими тваринами з можливістю фільтрації та сортування;
- сповіщення про знайдену схожу тварину.

3

Рисунок В.3 – Слайд 3

Порівняння підходів розпізнавання та класифікації

Критерій	Розпізнавання	Класифікація
1	2	3
Функції	Ідентифікація об'єктів та локацій	Сортування об'єктів за категоріями
Складність	Більш складний	Загалом простіше
Критерій	Розпізнавання	Класифікація
Архітектурні рішення	Використовуйте такі архітектури, як Faster R-CNN, YOLO, SSD, які включають як CNN так і ін.	Часто використовують стандартні архітектури CNN
Навчальні вибірки	Потрібні позначені зображення з класом/категорією та точними анотаціями обмежувальної рамки	Потрібні зображення з мітками класу/категорії
Часові витрати	Більші	Здатно швидше надати висновок; застосовується, коли має значення лише категорія
Приклад	Виявлення та позначення пішоходів, автомобілів і дорожніх знаків на вулиці.	Визначення зображення собаки чи kota.

4

Рисунок В.4 – Слайд 4

Порівняння існуючих аналогів

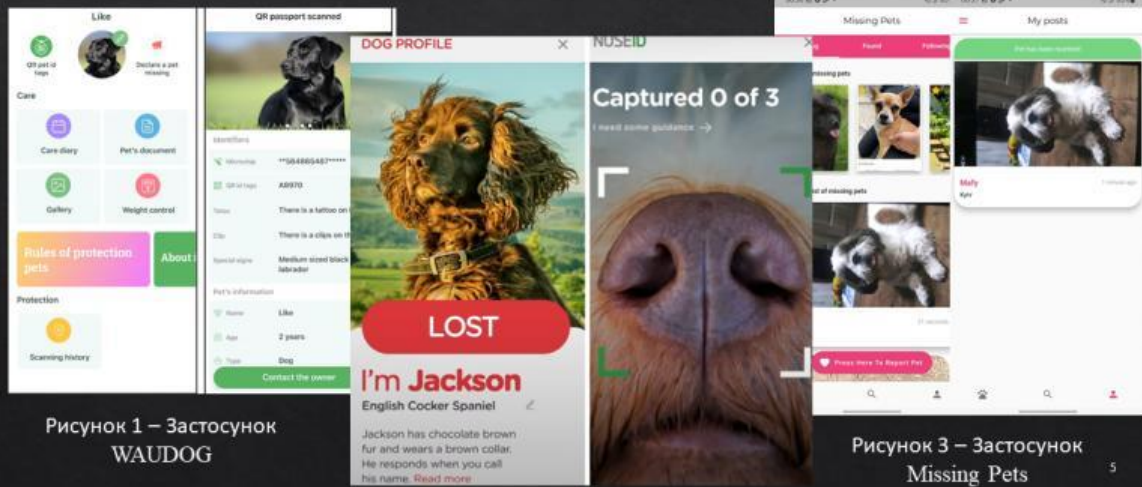


Рисунок 1 – Застосунок WAUDOG

Рисунок 2 – Застосунок NOSEiD

Рисунок 3 – Застосунок Missing Pets

Рисунок В.5 – Слайд 5

Порівняння існуючих аналогів

Критерій	WAUDOG	Missing Pets	NOSEiD	PawFounder
Створення допису про зниклу чи знайдену тварину	+	+	-	+
Доступна на платформах IOS та Android	+	+	-	+
Наявність профайла користувача з можливістю додавання тварин	+	+	+	+
Можливість створити допис з авто-заповненням з профайлу	-	-	-	+
Автоматичний пошук співпадінь з тваринами	+	-	+	+
Можливість почати використання застосунку вже після втрати тварини	-	+	-	+
Перегляд стрічки новин зі всіма дописами	++	+	-	+

Рисунок В.6 – Слайд 6

Матеріали і методи Трансферне навчання

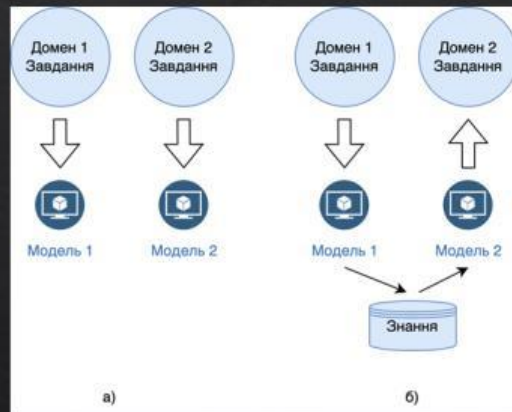


Рисунок 4 - а) ілюстрація традиційного навчання, б) ілюстрація трансферного навчання



7

Рисунок В.7 – Слайд 7

Аналіз існуючих технологій Порівняння фреймворків

Критерії	Flutter	Kotlin Multiplatform	Xamarin	ReactNative
Мова програмування	Dart	Kotlin	C#	JavaScript
Продуктивність (порівняно з нативною)	+	+	-	-
Єдиний користувацький інтерфейс для всіх платформ	+	-	-	+
ІТ компіляція	+	-	-	+
Оптимізація коду	+	+	-	-
Підтримка tree-shacking технології для збірок	+	часткова	-	-
Середовища розробки	Android Studio, Visual Studio Code, IntelliJ IDEA	Android Studio, IntelliJ IDEA	Visual Studio	Visual Studio Code, Atom
Оптимізація коду	+	+	-	-

8

Рисунок В.8 – Слайд 8

Аналіз існуючих технологій Порівняння середовищ розробки

Критерій	Visual Studio Code	Android Studio	IntelliJ IDEA
Вбудовані засоби для git	+	+	+
Навантаження системи	Низьке	Високе	Середнє
Доповнення коду	+	+	+
Вбудований відлагоджувач	+	+	+
Вбудований емулятор	-	+	-
Вбудований Android SDK	-	+	+
Безкоштовне	+	+	-

9

Рисунок В.9 – Слайд 9

Аналіз існуючих технологій Порівняння мов машинного навчання

Критерії	Python	Java	C/C++
Швидкість	Найменш швидка мова, інтерпритована	Середня за швидкістю з трьох. Компілюється та інтерпритується.	Найбільш швидка, компілюєма мова.
Легкість та швидкість написання коду	Має простий синтаксис, динамічно-типізований, містить велику кількість готових рішень.	Синтаксис складніший ніж в Python, статично-типізований, менше готових рішень.	Має низькорівневий і складний синтаксис, який може бути важким для освоєння.
Бібліотеки для ML	Великий обсяг бібліотек ML, таких як PyTorch, TensorFlow, Scikit-learn і тд.	Менший обсяг, порівняно з Python.	Найменша кількість бібліотек.
Популярність	Найбільш популярна для ML [28].	Середня за популярністю з трьох, проте всеж широко застосовувана для роботи з великими обсягами даних.	Найменш популярна для ML через меншу кількість бібліотек та складний синтаксис.
Маштабованість	Середня	Висока	Середня

10

Рисунок В.10 – Слайд 10

Аналіз існуючих технологій Порівняння сховищ даних

Критерій	RealtimeDatabase	CloudFirestore
Спосіб представлення даних	Дерево json.	Документи об'єднані в колекції
Маштабованість	Потребує сегментування на декілька БД при більш ніж 1000 записів в секунду	Автоматично
Відслідковування активного користувача	+	-
Запити до бази даних	Сортування та фільтрація можливі не одночасно. Запроси не індексовані, продуктивність запиту зменшується з ростом розміру даних.	Сортування та фільтрація можливі одночасно. Запроси індексовані, продуктивність залежить від розміру відповіді.
Операції над даними	Підходить для декількох гігабайт даних, що часто змінюються.	Підходить для сотні гігабайт даних, що помірно змінюються.
Прослуховування застосунком оновлень	+	+
Надійність та доступність	Підтримується тільки регіональна конфігурація, тобто БД доступні в межах регіону для якого налаштовано конфігурацію. Мають високу швидкість відповіді клієнту.	Дані зберігаються в різних центрах обробки, що знаходяться в декількох регіонах. Підтримується як регіональна так і мультирегіональна конфігурації.

Рисунок В.11 – Слайд 11

Архітектура застосунку



Рисунок 5 - Діаграма обраної архітектури

Рисунок В.12 – Слайд 12

Функціональні можливості розробленої програми



Рисунок 6 – Функціональна схема розробленої програми

13

Рисунок В.13 – Слайд 13

Алгоритми мобільного застосунку

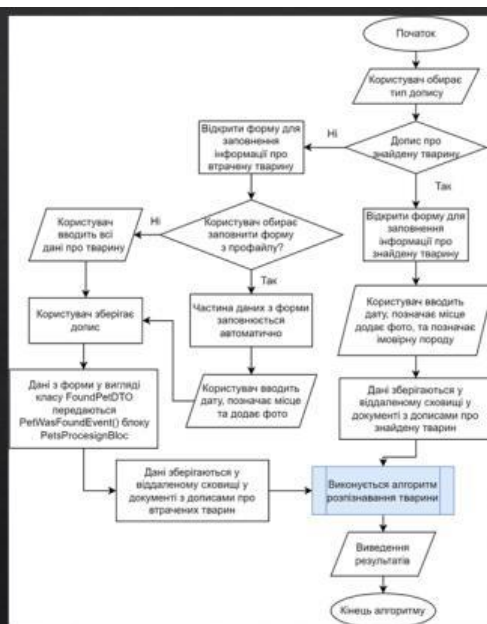


Рисунок 7 - Блок-схема створення записів про знайдену та втрачену тварину

- реєстрація та авторизація за допомоги email;
- реєстрація та авторизація за допомоги сервісів Google та Facebook;
- алгоритм створення допису про втрачену тварину;
- алгоритм створення допису про знайдену тварину;
- алгоритм пошуку тварини за фотографією за допомогою навченої моделі нейронної мережі;
- видалення допису;
- алгоритм створення профайлу користувача;
- алгоритм сортування допису;
- алгоритм фільтрації допису;

14

Рисунок В.14 – Слайд 14



Рисунок В.15 – Слайд 15

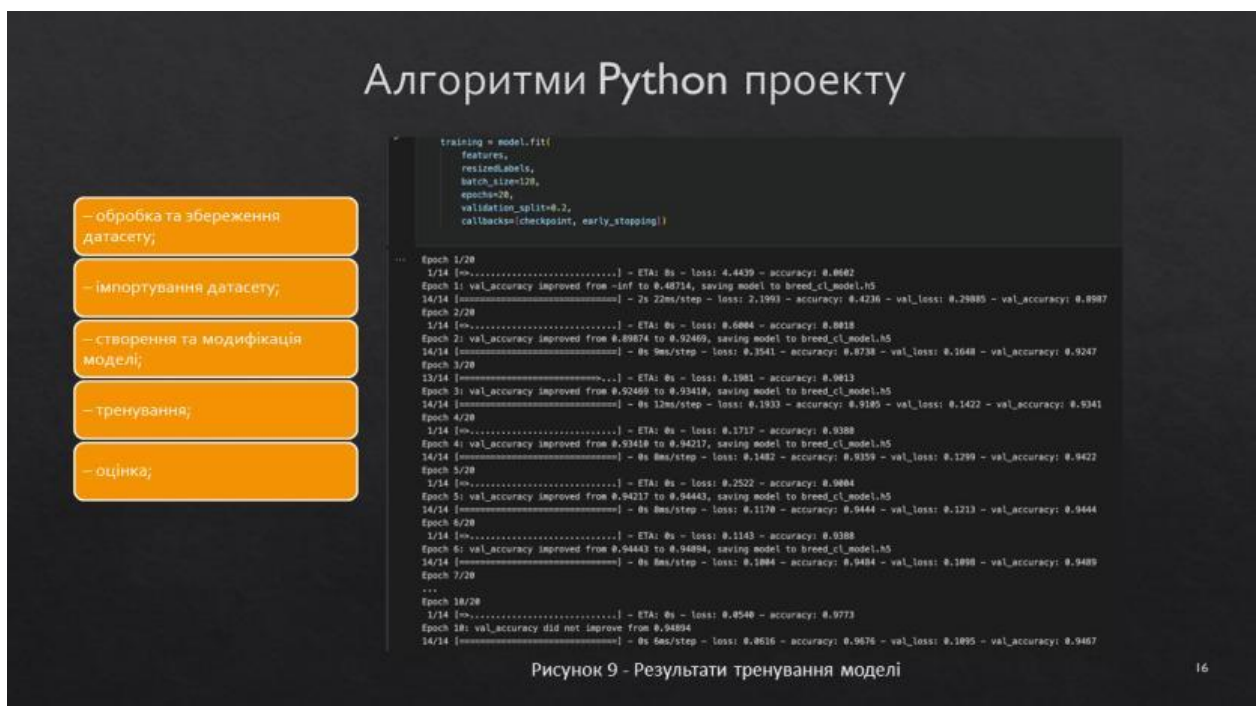


Рисунок В.16 – Слайд 16

Матриця помилок розробленої моделі

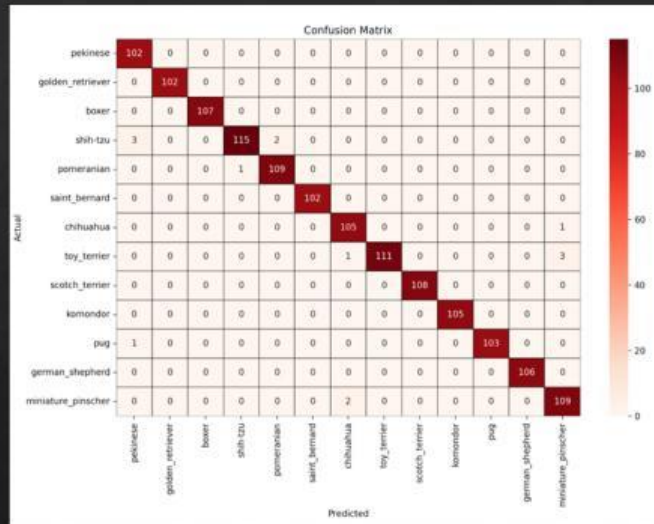


Рисунок 10 - Матриця помилок розробленої моделі

17

Рисунок В.17 – Слайд 17

Налаштування нотифікацій



Рисунок 11 - Налаштування тригерів та регіонів



Рисунок 12 - Створення та тестування коду функції

18

Рисунок В.18 – Слайд 18

Налаштування нотифікацій

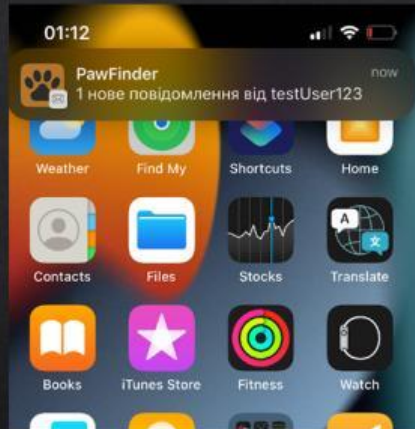


Рисунок 13 - Тестова нотифікація

19

Рисунок В.19 – Слайд 19

Приклад інтерфейсу застосунку

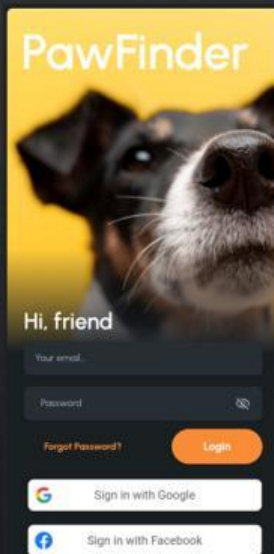


Рисунок 14 - Екран авторизації

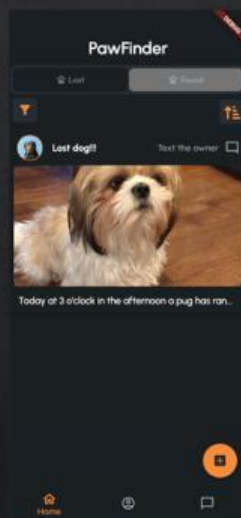


Рисунок 15 – Домашня сторінка

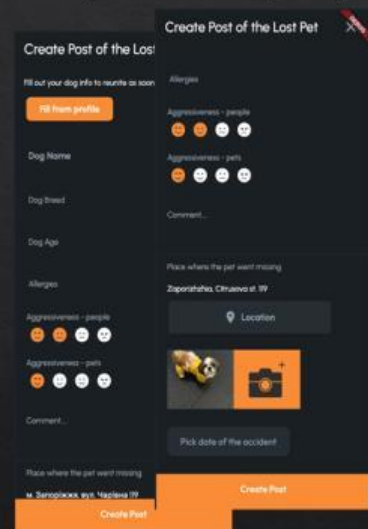


Рисунок 16 - Екран зі створенням допису про втрачену тварину

20

Рисунок В.20 – Слайд 20

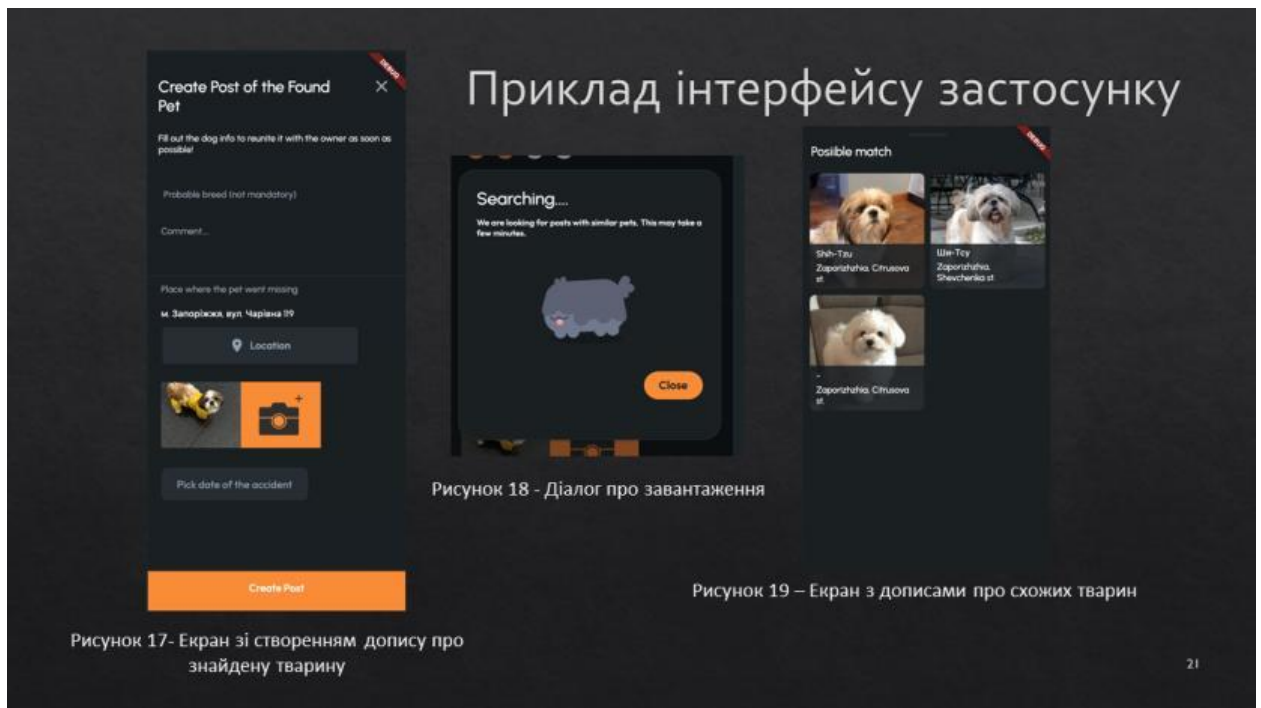


Рисунок В.21 – Слайд 21

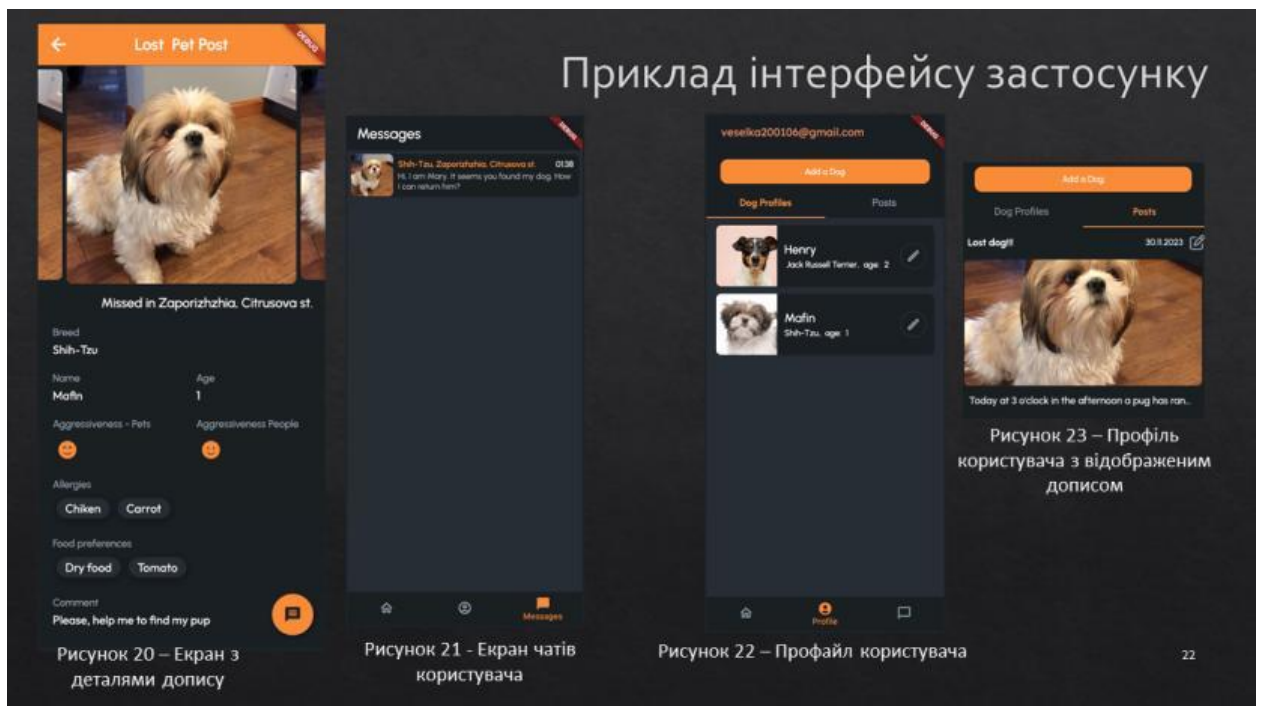


Рисунок В.22 – Слайд 22

Висновки

В ході дипломної роботи:

- Було розглянуто проблемну область застосунку, розглянуто можливі підходи та існуючі методи для задач розпізнавання та класифікації, проаналізовано існуючі аналоги програм;
- Визначено вимоги до застосунку, проаналізовано існуючі технології для Flutter та Python, обрано архітектуру;
- Розроблено та протестовано програмний продукт "PawFinder", що відповідає заявленим до нього вимогам, та стане помічником у пошуках загубленого улюбленця.

23

Рисунок В.23 – Слайд 23

Дякую за увагу!

24

Рисунок В.24 – Слайд 24