



Politechnika Wrocławska

“Lost + found pets”. Projekt aplikacji do identyfikacji zaginionych zwierząt

Prowadzący: dr inż. Jan Nikodem

Zespół:

Mateusz Oleksy, nr indeksu 272996

Jędrzej Radłowski, nr indeksu 272927

Bartosz Gruca, nr indeksu 272906

Robert Pytel, nr indeksu 272931

Godzina zajęć: piątek 08:45

Wrocław, 13.06.2025 r.

1. Opis projektu	4
2. Organizacja pracy w grupie	5
3. Wymagania zleceniodawcy	6
4. Założenia projektowe	7
5. Funkcje systemu	9
6. Scenariusze działania systemu	9
6.1. Zgłoszenie zaginięcia zwierzęcia	9
6.2. Zgłoszenie znalezienia zwierzęcia	10
6.3. Rejestracja i logowanie	10
6.4. Edycja konta i zgłoszeń	10
7. Schematy interfejsu graficznego (GUI)	11
8. Zestawienie zadań do realizacji	12
9. Analiza pracochłonności i kosztorys	13
10. Harmonogram realizacji projektu w postaci diagramu Adamieckiego	17
11. Projekt modelu sztucznej inteligencji	18
12. Projekt bazy danych	22
13. Projekt aplikacji serwerowej	25
13.1. Opis	25
13.2. Architektura aplikacji serwerowej:	26
13.3. Główne technologie i ich uzasadnienie	26
13.4. Powiązania z resztą systemu	27
13.5. Zabezpieczenia i utrzymanie	28
13.6. Utrzymanie i monitorowanie	28
14. Projekt interfejsu graficznego aplikacji użytkownika	29
15. Tworzenie i zapisywanie wzorców cech w bazie danych. Wyszukiwanie podobnych zwierząt na podstawie wzorców cech.	43
15.1 Tworzenie wzorców cech	43
15.2 Przykład wyszukiwania podobnych zwierząt na podstawie wzorców cech	43
16. Ekran powitalny, formularze logowania i rejestracji w aplikacji użytkownika	44
Formularze aplikacji	44
16.1 Stworzenie podstawowych komponentów aplikacji	44
16.2 Ekran powitalny	46
16.3 Formularz rejestracji i logowania	47
17. Obsługa rejestracji i uwierzytelniania użytkowników po stronie aplikacji serwerowej	48
18. Detektor psów i kotów na zdjęciu wejściowym	49
19. Testy skuteczności modelu w wyszukiwaniu zaginionych zwierząt	50
20. Wysłanie nowego raportu zaginionego zwierzęcia	52
20.1 Nowy raport dotyczący zaginionego zwierzęcia	52
21. Implementacja i integracja potoku przetwarzania sztucznej inteligencji	58
22. Widok profilu użytkownika	59
23. Edycja informacji o użytkowniku i usuwanie zgłoszeń	69
24. Dokumentacja programisty	70
24.1 Tworzenie wzorców cech na podstawie zdjęcia	70

24.2 Zapisywanie wzorca cech w bazie danych	71
24.3 Wyszukiwanie zdjęć przedstawiających podobne zwierzęta na podstawie wektorów cech	72
24.4 Ekran powitalny, formularze logowania i rejestracji w aplikacji użytkownika	74
24.5 Obsługa rejestracji i uwierzytelniania użytkowników po stronie aplikacji serwerowej	
76	
24.6 Detektor psów i kotów na zdjęciu wejściowym	80
24.7 Implementacja i integracja potoku przetwarzania sztucznej inteligencji	81
24.8 Implementacja zgłoszenia w aplikacji serwerowej	85
24.9 Integracja wysyłania nowego raportu z serwerem	88
24.10 Przesyłanie raportu	88
24.11 Wyświetlanie zdjęć pasujących	91
25. Edycja informacji o użytkowniku i usuwanie zgłoszeń	93
25.1 Sprawdzenie	97
26. Hostowanie aplikacji i utworzenie pliku .apk na podstawie plików projektu.	
Instalacja aplikacji na urządzeniu mobilnym.	99
26.1 Sprawdzenie	101
27. Instrukcja administratora	102
27.1 Uruchomienie na własnej infrastrukturze	102
27.2 Wymagania systemowe	102
27.3 Przygotowanie systemu	102
27.4 Pobieranie repozytorium projektu	102
27.6 Zarządzanie bazą danych PostgreSQL	103
27.7 Obsługa Swagger (dokumentacja API)	104
27.8 Monitorowanie i diagnostyka	104
28. Role instalacji	104
28.1 Administrator systemu (SysAdmin)	105
28.2 Administrator bazy danych (DBA)	105
28.3 DevOps / Inżynier wdrożeń	105
28.4 Programista / Zespół developerski	106
28.5 Konsultant wdrożeniowy / Kierownik projektu (PM)	106
28.6 Tester / QA	106
28.7 Użytkownik końcowy / Instalator (jeśli aplikacja instalowana lokalnie)	107
29. Instrukcja użytkownika	108
29.1 Instalacja aplikacji mobilnej na system Android	108
29.2 Rejestracja użytkownika w systemie	109
29.3 Logowanie użytkownika w systemie	110
29.4 Dodawanie zgłoszenia	111
29.5 Usuwanie zgłoszenia	115
29.6 Edycja danych użytkownika	117

1. Opis projektu

Wykonał: Robert Pytel

Celem projektu jest stworzenie aplikacji do identyfikacji znalezionych zwierząt. System porównuje zdjęcie znalezionego psa lub kota z bazą zaginionych zwierząt, używając algorytmów analizy obrazu do automatycznego dopasowania. W tym celu należy wykonać aplikację webową/mobilną umożliwiającą zgłaszać zaginionych zwierząt oraz wyszukiwanie dopasowań. Oprócz tego niezbędne będzie zaprojektowanie oraz implementacja bazy danych przechowującej informacje o zaginionych zwierzętach i ich zdjęciach. Finalnym elementem projektu będzie stworzenie modelu sztucznej inteligencji do analizy i porównywania zdjęć zwierząt.

Projekt dotyczy stworzenia systemu informatycznego, którego głównym celem będzie identyfikacja odnalezionych zwierząt domowych (psów oraz kotów) za pomocą technik przetwarzania obrazów oraz algorytmów sztucznej inteligencji. Realizacja tego przedsięwzięcia obejmuje stworzenie natywnej aplikacji mobilnej, umożliwiającej zgłaszać zaginięcia zwierząt przez ich właścicieli oraz automatyczne wyszukiwanie dopasowań ze zwierzętami znalezionymi przez osoby trzecie.

Aplikacja zostanie przygotowana w technologii *React Native 0.78* co umożliwi wdrożenie aplikacji na systemach operacyjnych Android oraz iOS. Minimalna wspierana wersja systemu Android to *Android 14*. W przypadku systemu firmy Apple, minimalna wspierana wersja to *iOS 17*.

System informatyczny będzie umożliwiał założenie konta użytkownikom aplikacji. W związku z tym podstawowym zadaniem będzie zabezpieczenie danych wrażliwych użytkowników. Podstawowym mechanizmem będzie szyfrowanie informacji przechowywanych w bazie danych za pomocą algorytmu SHA256.

Warstwa biznesowa aplikacji zostanie opracowana przy wykorzystaniu, co pozwoli na obsługę żądań użytkowników oraz integrację z modelem sztucznej inteligencji. Kluczowym elementem backendu będzie serwer aplikacyjny odpowiedzialny za logikę przetwarzania danych oraz komunikację z bazą danych. Do zarządzania bazą danych przewiduje się wykorzystanie systemu zarządzania relacyjnymi bazami danych (RDBMS), takiego jak PostgreSQL lub MySQL. W bazie danych przechowywane będą szczegółowe informacje o zgłoszonych zwierzętach, w tym dane właścicieli, lokalizacje, daty zaginięcia oraz zdjęcia umożliwiające późniejszą analizę porównawczą.

Istotnym aspektem technicznym będzie zaprojektowanie schematu bazy danych, uwzględniające relacje między tabelami zawierającymi rekordy zwierząt, użytkowników oraz zgłoszeń. W projekcie przewidziane jest użycie technologii przechowywania plików (np. Amazon S3) do przechowywania zdjęć.

Centralnym komponentem projektu będzie stworzenie i implementacja modelu sztucznej inteligencji opartego o algorytmy uczenia maszynowego oraz głębokie sieci neuronowe. Do analizy oraz porównywania zdjęć zwierząt wykorzystywane będą techniki rozpoznawania obrazów oparte o modele konwolucyjnych sieci neuronowych. Model będzie trenowany na zbiorze danych zdjęć psów i kotów, które zostały poddane selekcji i modyfikacji.

2. Organizacja pracy w grupie

Wykonał: Jędrzej Radłowski

Czas trwania projektu obejmuje okres od 21.03.2025 r. do 24.06.2025 r. W tym okresie zostało przewidziane 13 terminów spotkań na zajęciach projektowych. Zgodnie z kartą przedmiotu na każdego członka zespołu przewidziane zostało 100 godzin nakładu pracy. Zgodnie z opisem projektu należy wyróżnić trzy obszary pracy: model sztucznej inteligencji, aplikacja użytkownika oraz baza danych. Na podstawie wymagań projektowych najważniejszym i najbardziej priorytetowym aspektem projektu jest model sztucznej inteligencji.

Najważniejsze zadania związane z modelem sztucznej inteligencji:

- zbieranie i przetwarzanie wstępne różnych zdjęć psów i kotów
- stworzenie lub znalezienie istniejącego modelu klasyfikacji obrazów
- dopasowanie modelu w celu identyfikacji znalezionych zwierząt
- testowanie modelu

Najważniejsze zadania dotyczące bazy danych:

- projekt struktury bazy danych
- implementacja bazy danych
- testowanie bazy danych z naciskiem na zapytania

Najważniejsze zadania dotyczące aplikacji użytkownika:

- projekt graficzny aplikacji
- implementacja aplikacji użytkownika
- testowanie interfejsu użytkownika

3. Wymagania zleceniodawcy

Wykonał: Robert Pytel

W przypadku modelu sztucznej inteligencji zleceniodawca wyróżnił następujące wymagania:

1. Model sztucznej inteligencji ma działać dla dwóch gatunków zwierząt: psów i kotów.
2. Model porównuje zdjęcia wejściowego ze zdjęciami w bazie danych. Wyświetla te zdjęcia, które osiągnęły podobieństwo na poziomie minimum 70% ze zdjęciem wejściowym.

Wymagania dotyczące aplikacji użytkownika:

1. Aplikacja powinna umożliwiać przesyłanie obrazu na serwer. Obraz ten może być wybrany z galerii zdjęć lub wykonany poprzez aplikację przy użyciu aparatu fotograficznego.
2. Aplikacja ma udostępniać formularze logowania oraz rejestracji użytkownikowi, który chce zgłosić zaginięcie zwierzęcia.
3. Aplikacja ma działać zarówno na systemach Android jak iOS. Minimalna wspierana wersja systemu Android to *Android 14*. W przypadku systemu iOS minimalna wspierana wersja to *iOS 17*.
4. Aplikacja ma umożliwiać tworzenie zgłoszeń zaginionych zwierząt przez właścicieli.
5. Aplikacja ma umożliwiać zgłaszanie przypadków, w których użytkownicy aplikacji odnaleźli zaginione zwierzę na ulicy. Zdjęcie zwierzęta ma zostać przesłane do bazy danych, w której zostaną wyszukane dopasowania.

W momencie dopasowania zdjęcia, znalazca otrzymuje dane kontaktowe do właściciela zwierzęcia.

4. Założenia projektowe

Wykonał: Bartosz Gruca

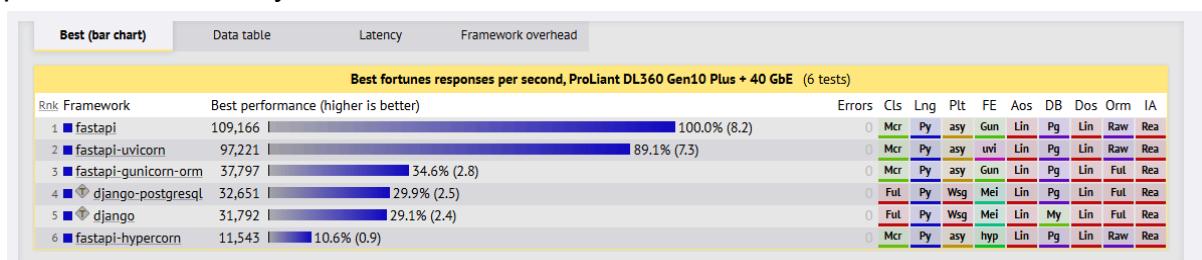
Projekt dzieli się na cztery główne obszary: model sztucznej inteligencji, aplikacja użytkownika, aplikację serwerową oraz bazę danych. Każdy z tych modułów będzie rozwijany równolegle, jednak priorytetem jest stworzenie skutecznego modelu sztucznej inteligencji.

Głównym zadaniem modelu jest analiza zdjęć zwierząt (psów i kotów) oraz wyszukanie i zwrócenie zdjęć z bazy danych. Zwrócone fotografie powinny uzyskać podobieństwo na poziomie minimum 70% z obrazem wejściowym, jeżeli tak się nie stanie, przyjmujemy że danego zwierzętka nie ma w bazie. Obiekty będą klasyfikowane na podstawie ekstrakcji cech jak krawędzie, tekstury.

Zbiór danych będzie musiał zostać poddany przetwarzaniu wstępemu. Niezbędne może być zastosowanie technik modyfikacji danych, aby model był odporny na różnice w oświetleniu, kącie fotografowania czy tle.

Aplikacja użytkownika zostanie zaimplementowana na urządzenia mobilne. Aplikacja zostanie przygotowana w technologii *React Native* 0.78 co umożliwi wdrożenie aplikacji na systemach operacyjnych Android oraz iOS. Minimalna wspierana wersja systemu Android to *Android 14*. W przypadku systemu firmy Apple, minimalna wspierana wersja to *iOS 17*. Zgodnie z wymaganiami aplikacja będzie umożliwiać przesyłanie obrazu na serwer. Obraz ten może być wybrany z galerii zdjęć lub wykonany poprzez aplikację przy użyciu aparatu fotograficznego.

Aplikacja serwerowa zostanie wykonana w oparciu o technologię *FastAPI* opartej na języku programowania *Python*. Wybór technologii został poparty wydajnością narzędzia na tle pozostałych technologii opartych na języku *Python*. Zostało to przedstawione na rysunku nr 1.



Rysunek nr 1: Zestawienie technologii aplikacji serwerowych opartych na języku Python z dnia 24.02.2025 r. Źródło: <https://www.techempower.com/benchmarks>

Oprócz kwestii wydajności narzędzia dodatkowym czynnikiem wyboru technologii jest doświadczeniu zespołu w użytkowaniu narzędzia *FastAPI*.

Aplikacja serwerowa będzie zabezpieczona pod względem treści przesyłanych przez użytkowników końcowych mających złe zamiary. Dotyczy to przede wszystkim ograniczenia ilości wysyłanych zapytań np. 10 zapytań na minutę. Oprócz tego ograniczenie będzie obejmowało również rozmiar przesyłanych zdjęć w celu eliminacji obciążenia łącza z serwerem oraz zaoszczędzenia miejsca na dysku serwera. Dodatkowo po spełnieniu powyższych warunków zdjęcia będą poddawane wstępnej klasyfikacji i walidacji pod względem zawartości. Jeżeli fotografia nie będzie przedstawiała psa lub kota nie zostanie dodana do bazy danych systemu. Zgłoszenie takie zostanie potraktowane jako niewłaściwe o czym użytkownik otrzyma komunikat.

Do implementacji bazy danych zostanie użyta relacyjna baza danych oparta na systemie relacyjnego zarządzania bazami danych PostgreSQL. Technologia PostgreSQL została wybrana na tle konkurencji ze względu na kilka czynników. Po pierwsze jest ona oparta o licencję typu *open-source*, co zapewnia jej darmowe użytkowanie. Dodatkowo kod aplikacji jest otwarty, co zapewnia ciągły rozwój produktu oraz pomoc ze strony użytkowników systemu. Oprócz tego system PostgreSQL został zaimplementowany w oparciu o zasady wyrażone za pomocą akronimu *ACID*, co zapewnia kolejno niepodzielność, spójność, izolację oraz trwałość danych. Ostatnim argumentem za technologią PostgreSQL jest wydajność pod względem szybkości przetwarzania licznych zbiorów danych. Podstawowymi elementami świata rzeczywistego, które należy uwzględnić w bazie danych są właściciele zwierząt, fotografie zwierząt oraz zgłoszenia zaginięć.

Dane osobowe użytkowników są danymi wrażliwymi. Niezbędnym warunkiem założenia konta w systemie jest zgoda na przechowywanie danych osobowych w systemie oraz udostępnianie wybranych informacji w postaci numeru telefonu oraz adresu e-mail właścicielom zwierząt. Kwestia bezpiecznego przechowywania danych zapewniona jest przez system zarządzania danymi PostgreSQL.

5. Funkcje systemu

Wykonał: Jędrzej Radłowski

Rejestracja i logowanie

- *Tworzenie konta użytkownika.*
- *Logowanie do systemu.*
- *Resetowanie hasła.*

Dodawanie zgłoszenia o zaginięciu zwierzęcia.

- *Dodawanie zdjęć zwierzęcia, wraz z dodatkowymi informacjami.*
- *Zapisanie zgłoszenia w bazie danych.*

Wyszukiwanie odnalezionych zwierząt i wymiana danych kontaktowych między właścicielem a znalazcą zwierzęcia

- *Wysyłanie zdjęcia znalezionej zwierzęcia do analizy.*
- *Otrzymanie listy podobnych zwierząt z bazy.*
- *Powiadomienie właściciela o potencjalnym znalezieniu zwierzęcia na podstawie otrzymanej listy podobnych zwierząt.*

Zarządzanie kontem użytkownika

- *Edycja danych użytkownika.*
- *Przegląd i edycja zgłoszeń.*

Dopasowanie podobnych obrazów zwierząt

- *Porównywanie zdjęcia wejściowego ze zdjęciami w bazie i wyświetlanie zdjęć, które osiągnęły podobieństwo na poziomie minimum 70%.*

6. Scenariusze działania systemu

Wykonał: Bartosz Gruca

6.1. Zgłoszenie zaginięcia zwierzęcia

1. Użytkownik loguje się do aplikacji.
2. Przechodzi do sekcji "Zgłoś zaginięcie".
3. Wypełnia formularz zgłoszenia i dodaje zdjęcie.
4. Aplikacja zapisuje dane w bazie.
5. Właściciel może edytować lub usunąć zgłoszenie.
6. Jeżeli model dopasuje zdjęcie z bazy danych do tego które zostało przesłane przez właściciela, właściciel otrzymuje dane kontaktowe do znalazcy.

6.2. Zgłoszenie znalezienia zwierzęcia

1. Użytkownik loguje się do aplikacji.
2. Przechodzi do sekcji "Znalazłem zwierzę".
3. Wgrywa zdjęcie zwierzęcia.
4. Aplikacja przesyła zdjęcie do modelu AI.
5. Model AI porównuje zdjęcie z bazą.
6. Aplikacja zwraca zdjęcia, które osiągnęły podobieństwo na poziomie minimum 70%.
7. Użytkownik może skontaktować się z właścicielem poprzez podane przez niego formy komunikacji.
8. W przypadku gdy model nie znajdzie takich obrazów w bazie danych wyświetlana jest informacja o braku dopasowań. Zdjęcie zgłoszone przez znalazcję jest zapisywane w bazie danych.

6.3. Rejestracja i logowanie

1. Użytkownik wybiera "Zarejestruj się".
2. Podaje dane (imię, nazwisko, e mail, hasło, numer telefonu).
3. System szyfruje hasło i zapisuje je w bazie.
4. Po rejestracji może się zalogować i korzystać z aplikacji.

6.4. Edycja konta i zgłoszeń

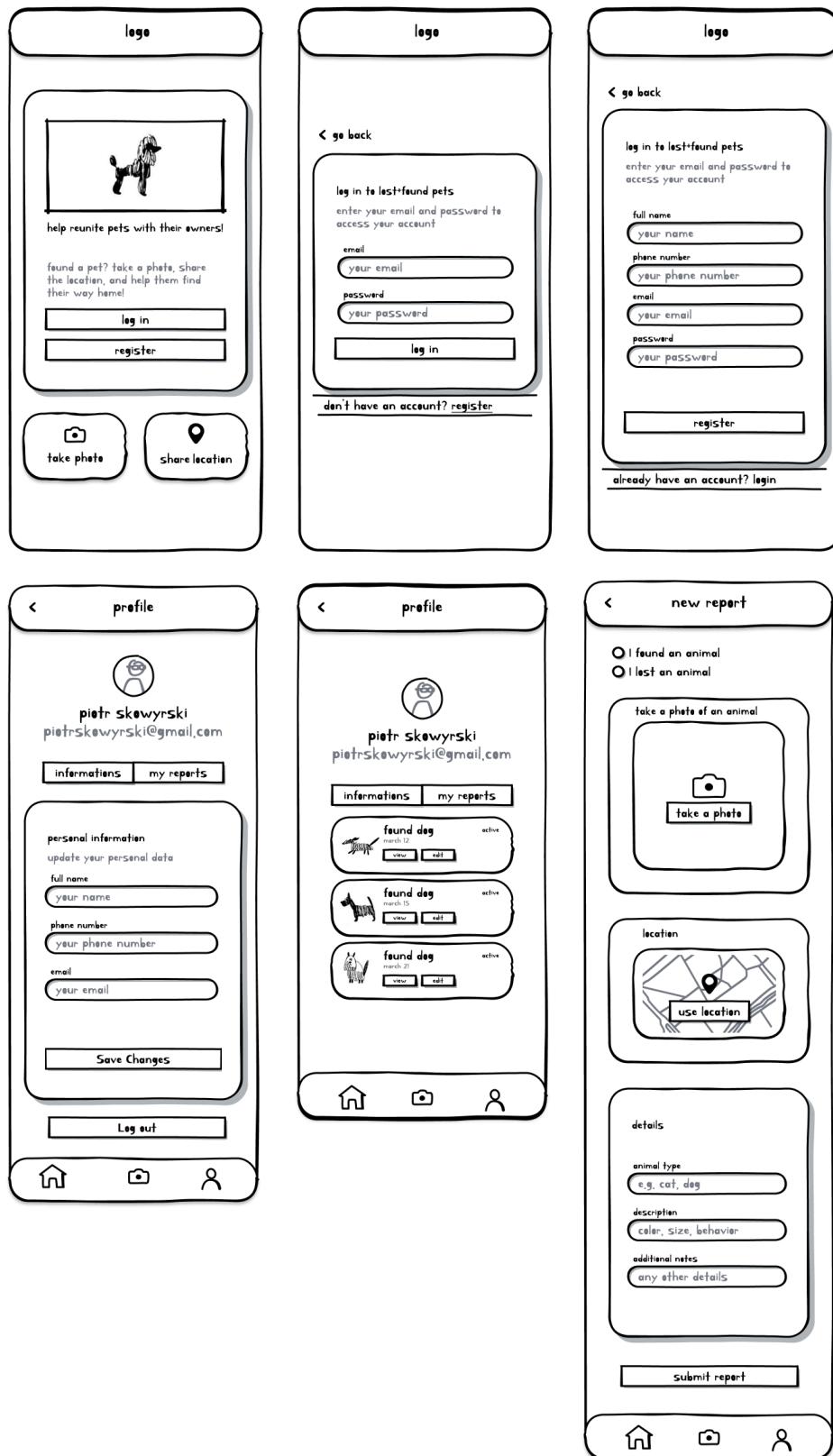
1. Użytkownik loguje się do aplikacji.
2. Przechodzi do sekcji "Moje konto".
3. Może edytować swoje dane (np. numer telefonu).
4. Może też edytować lub usunąć swoje zgłoszenia.

6.5. Walidacja zgłoszeń

1. Model klasyfikujący moderuje niepoprawne zgłoszenia pod względem zawartości przesyłanych obrazów.

7. Schematy interfejsu graficznego (GUI)

Wykonał: Mateusz Oleksy



Rysunek nr 2: Schematy interfejsu użytkownika (GUI)

Na rysunku numer 2 przedstawiono schematy interfejsu użytkownika (GUI). Przedstawione zostały na nich zarówno widoki profilu jak i interfejsów służących do zgłoszenia zaginionych zwierząt.

8. Zestawienie zadań do realizacji

Wykonał: Mateusz Oleksy

Zadania do realizacji można wyróżnić dla poszczególnych faz projektowych. Wyszczególniono następujące etapy realizacji projektu: planowanie, implementacja, testowanie oraz prezentacja projektu.

Planowanie:

- Projekt modelu sztucznej inteligencji
- Zbieranie i przetwarzanie zdjęć psów i kotów
- Projekt struktury bazy danych
- Projekt struktury aplikacji serwerowej
- Projekt interfejsu graficznego aplikacji użytkownika

Implementacja:

- Stworzenie lub znalezienie istniejącego modelu klasyfikacji obrazów
- Dopasowanie modelu w celu identyfikacji znalezionych zwierząt
- Implementacja bazy danych
- Implementacja aplikacji serwerowej
- Implementacja aplikacji użytkownika
- Integracja systemu
- Systematyczny opis poszczególnych etapów projektowych

Testowanie:

- Testowanie i walidacja poprawności modelu kategoryzacji obrazów
- Testowanie zapytań bazy danych
- Testowanie aplikacji serwerowej
- Testowanie interfejsu użytkownika
- Testowanie integracji serwisów

Prezentacja:

- Prezentacja podczas zajęć projektowych
- Prezentacja podczas konferencji projektów zespołowych

9. Analiza pracochłonności i kosztorys

Wykonał: Jędrzej Radłowski

Jak można zauważyć na podstawie zestawionych zadań do realizacji, projekt obejmuje szerokie grono ról związanych z branżą informatyczną. Podstawą do oszacowania kosztorysu jest ilość godzin poświęcona na poszczególne zadania. W związku z tym należy obliczyć całkowity koszt prac na podstawie stawki godzinowej dla poszczególnych zawodów wykonujących określone zadanie.

Zawód	Stawka godzinowa brutto (PLN/h)	Stawka godzinowa netto (PLN/h)	Stawka godzinowa brutto brutto (PLN/h)
administrator baz danych	38,75	30,43	46,99
programista aplikacji serwerowych	77,38	60,76	93,22
programista aplikacji mobilnych	48,63	38,18	58,58
kierownik projektu	93,75	73,62	112,95
projektant graficzny	43	31,38	51,81
inżynier uczenia maszynowego	144,46	208,89	251,67

Tabela 1: zestawienie stawek godzinowych brutto, netto oraz brutto brutto wyrażonych w jednostce PLN/h dla poszczególnych zawodów w branży informatycznej

Tabela numer 1 przedstawia średnie stawki godzinowe dla poszczególnych zawodów w branży informatycznej. Na podstawie tych danych zostanie obliczony finalny koszt projektu po wyszczególnieniu potrzebnych roboczogodzin na wykonanie poszczególnych zadań.

Zawód	Źródło informacji o zarobkach
administrator baz danych	https://praca.money.pl/wynagrodzenia/s/administrator-baz-danych/
programista aplikacji serwerowych	https://wynagrodzenia.pl/moja-placa/ile-zarabia-programista-python
programista aplikacji mobilnych	https://praca.money.pl/wynagrodzenia/s/programista-aplikacji-mobilnych/
kierownik projektu	https://wynagrodzenia.pl/moja-placa/ile-zarabia-kierownik-projektu-it
projektant graficzny	https://wynagrodzenia.pl/moja-placa/ile-zarabia-projektant-grafiki
inżynier uczenia maszynowego	https://pl.jooble.org/salary/machine-learning-engineer#hourly

Tabela 2: zestawienie zawodów i źródeł informacji o zarobkach

Na podstawie wyszczególnionych zadań do realizacji możliwa jest analiza pracochłonności projektu. Każdemu zadaniu została przypisana określona liczba roboczogodzin.

Zadania do realizacji	Ilość czasu potrzebna na realizację zadania [h]	Stawka godzinowa brutto brutto wedle ról z tabeli numer 1 [PLN/h]	Koszt [PLN]
Projekt modelu sztucznej inteligencji	17,6	251,61	4429,39
Zbieranie i przetwarzanie zdjęć psów i kotów	17,6	51,81	911,86
Projekt struktury bazy danych	17,6	46,49	821,74
Projekt struktury aplikacji serwerowej	17,6	93,22	1640,67
Projekt interfejsu graficznego aplikacji użytkownika	26,4	51,81	1367,78
Stworzenie lub znalezienie istniejącego modelu klasyfikacji obrazów	20,9	251,67	5259,90
Dopasowanie modelu w celu identyfikacji znalezionych zwierząt	24,2	251,67	6090,41
Implementacja bazy danych	26,4	46,49	1232,62
Implementacja aplikacji serwerowej	39,6	93,22	3691,51

Implementacja aplikacji użytkownika	39,6	58,58	2319,77
Integracja systemu	26,4	112,95	2981,88
Systematyczny opis poszczególnych etapów projektowych	44,0	112,95	4969,80
Testowanie i walidacja poprawności modelu kategoryzacji obrazów	17,6	251,67	4429,39
Testowanie zapytań bazy danych	17,6	46,69	821,74
Testowanie aplikacji serwerowej	17,6	93,22	1640,67
Testowanie interfejsu użytkownika	33,0	58,58	1933,14
Testowanie integracji serwisów	8,8	112,95	993,96
Prezentacja podczas zajęć projektowych	1	112,95	112,95
Prezentacja podczas konferencji projektów zespołowych	1	112,95	112,95

Tabela 3: zestawienie zadań do realizacji, czasu potrzebnego na ich wykonanie oraz kosztów z tym związanych.

Na podstawie tabeli numer 3 obliczono całkowity koszt projektu, który wyniósł: 45762,15 PLN.

10. Harmonogram realizacji projektu w postaci diagramu Adamieckiego

Wykonat: Bartosz Gruca

Lost+found pets

chlebkiNaN + Estigli sp.z.o.o.

Lider projektu: Robert Pytel



Rysunek 3: harmonogram Adamieckiego na okres od 21.03.2025 r. do 11.05.2025 r.

Lost+found pets

chlebkiNaN + Estigli sp.z.o.o.

Lider projektu: Robert Pytel



Rysunek 4: harmonogram Adamieckiego na okres od 12.05.2025 r. do 30.06.2025 r.

11. Projekt modelu sztucznej inteligencji

Wykonał: Robert Pytel, sprawdził: Jędrzej Radłowski, zatwierdził: Bartosz Gruca

Zgodnie z wymaganiami zleceniodawcy system identyfikujący zwierzęta ma korzystać z narzędzi sztucznej inteligencji. Wymaganie to jest jak najbardziej uzasadnione, ponieważ na przestrzeni ostatnich lat technologie wykorzystujące sztuczną inteligencję rozwinięły się w zawrotnym tempie i umożliwiają wykonywanie coraz to bardziej wymagających zadań.

Jednakże w jakim celu sztuczna inteligencja zostanie wykorzystana w systemie? Otóż ma ona wyręczyć człowieka oraz jego doskonały organ wzroku w przeszukiwaniu zbioru zdjęć w celu wyszukania fotografii zwierzęcia, które najbardziej przypomina zwierzaka na zdjęciu przesłanym przez użytkownika. W scenariuszu, gdzie człowiek ręcznie musiałby przejrzeć tysiące fotografii, zadanie to zajęłoby ogromną ilość czasu. Sztuczna inteligencja ma przyspieszyć ten proces zapewniając przy tym skuteczność w związku z wyszukiwaniem odpowiedniego zwierzaka.

Narzędziem, które zostanie wykorzystane w celu realizacji wyżej opisanego zadania jest sieć neuronowa służąca do klasyfikacji obiektów na obrazie. Najczęściej wykorzystywanym rodzajem sieci neuronowych do realizacji tego zadania są konwolucyjne sieci neuronowe. Jedną z nich jest sieć *ResNet-50*. Została ona opracowana przez jeden z oddziałów firmy Microsoft w 2015 roku.

Działanie modelu *ResNet-50* zostało pierwotnie dostosowane w oparciu o bazę danych *ImageNet*. Jest to baza danych zdjęć, która posiada ponad 14 milionów obrazów. Projekt *ImageNet* obejmuje łącznie ponad 1000 klas obiektów przedstawionych na grafikach. Sieć *ResNet-50* osiągnęła średnią skuteczność klasyfikacji na poziomie około 80% na tym zbiorze danych. Jest to bardzo dobry wynik, w porównaniu ze średnią skutecznością klasyfikacji na poziomie około 95% osiąganą przez człowieka.

Aby uzyskać lepszą dokładność działania modelu wykonuje się w tym celu trenowanie modelu na konkretnym zbiorze danych. Proces ten jest określany w języku angielskim jako *fine-tuning*. W związku z tym, że system będzie przetwarzał zdjęcia psów oraz kotów należy dostosować go do zbiorów danych dla tych gatunków zwierząt. W przypadku psów, zbiorem danych wykorzystanym w celu ulepszenia modelu jest [Stanford Dogs Dataset](#). Zbiór danych posiada 20 580

fotografii, które obejmują 120 ras psów. Model *ResNet-50* wytrenowany na tym zbiorze danych osiągnął dokładność klasyfikacji ras psów na poziomie 84%.

Kolejnym zagadnieniem, które należy rozważyć jest określenie w jaki sposób model będzie określał podobieństwo pomiędzy fotografiemi zwierząt. W tym przypadku dotyczy to zagadnienia przestrzeni cech w postaci wektorowej. Zgodnie z nią każda fotografia zwierzęcia reprezentowana jest za pomocą wektora liczb zwanego inaczej wartościami cech. Wartości te obliczane są przez model w momencie przetwarzania fotografii przez sieć neuronową. Za pomocą wyznaczonego wektora cech przypisanego do konkretnego obiektu możemy jednoznacznie określić położenie tego obiektu w przestrzeni cech, która ma tyle wymiarów ile liczb zawiera wektor (ile jest cech). W przypadku modelu *ResNet-50* wektor ten zawiera 2048 wartości, co daje przestrzeń o 2048 wymiarach. Od tego momentu fotografia zwierzaka może być pojmowana jako punkt w tak wielowymiarowej przestrzeni cech. Ważną informacją jest to, że obiekty o zbliżonych cechach będą znajdować się w niewielkiej odległości od siebie w przestrzeni cech.

Podsumowując powyższy opis, zdjęcia przesyłane przez użytkowników przetwarzane są przez model *ResNet-50*, na którego wyjściu możliwe jest wyodrębnienie informacji o wektorach cech zdjęcia w postaci wektora o 2048 wartościach. Następnie system zapamiętuje dany wektor cech w bazie danych. Wektor ten powiązany jest również z konkretnym zgłoszeniem znajdującym się w bazie danych.

Mając zdefiniowaną bazę danych wektorów cech fotografii oraz mając wektor cech zdjęcia wejściowego należy opracować metodę wyznaczania podobieństwa tych wektorów. Dwoma najczęściej wykorzystywanyimi funkcjami do obliczenia podobieństwa wektorów w przestrzeni jest odległość euklidesowa pomiędzy wektorami oraz podobieństwo kosinusowe wektorów.

$$d(x, y) = \sqrt{\sum_{i=1}^n (y_i - x_i)^2}$$

Wzór 1: odległość euklidesowa pomiędzy dwoma wektorami

Funkcja ta jest pierwiastkiem sumy kwadratów różnic wartości znajdujących się na kolejnych pozycjach wektorów. Wartość tą interpretujemy jako odległość w “prostej linii” pomiędzy końcami wektorów.

W przypadku wybrania tej metryki dążymy do minimalizacji tej funkcji. To znaczy, że obliczamy wartość tej funkcji dla wektora wejściowego oraz wektorów znajdujących

się w bazie i wybieramy ten wektor, dla którego funkcja osiągnęła najmniejszą wartość.

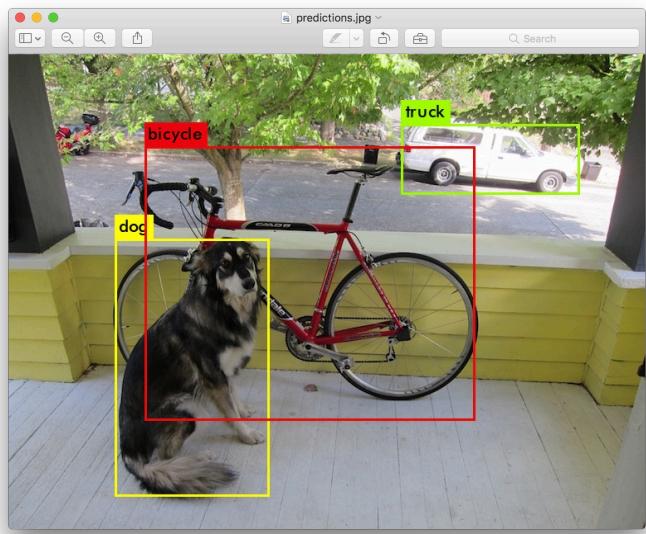
$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Wzór 2: podobieństwo kosinusowe pomiędzy dwoma wektorami

Kolejna funkcja jest iloczynem wektorowym pomiędzy dwoma wektorami podzielonym przez iloczyn długości tych wektorów. Wartość tej funkcji mieści się w przedziale od -1 do 1, gdzie 1 oznacza identyczne wektory, 0 oznacza, że wektory są niezależne (prostopadłe), natomiast wartość -1 określa, że wektory są swoim dokładnym przeciwnieństwem. W tym przypadku dąży się do maksymalizacji tej funkcji. Wartość funkcji obliczana jest dla wektora wejściowego oraz każdego wektora znajdującego się w bazie danych, a następnie wybierany jest wektor, dla którego uzyskano największą wartość funkcji.

Kolejnym ważnym zagadnieniem, które należy zaimplementować w systemie jest przetwarzanie zdjęcia wejściowego przesłanego przez użytkownika pod względem jego zawartości. System ma działać tylko na fotografiach psów i kotów, w związku z czym wstępne przetwarzanie musi zapewnić, że takie obiekty znajdują się na fotografii.

W tym celu zostanie wykorzystane narzędzie znane pod nazwą YOLO. Technologia ta przetwarza zdjęcie wejściowe, klasyfikuje znajdujące się na nim obiekty, a następnie zwraca taką informację użytkownikowi w postaci wyciętych fragmentów obrazu z klasą obiektu znajdującego się na danym fragmencie.



Przykład użycia biblioteki YOLO

Narzędzie to pozwoli na podstawowe zabezpieczenie systemu przed niepożądanymi zdjęciami, które mogą zostać przesłane przez użytkownika. Oprócz tego ważnym działaniem jest wycięcie fragmentu obrazu, na którym znajduje się pies czy kot. Przetwarzanie tylko tych fragmentów przez sieć neuronową pozwala na ekstrakcję cech dotyczących samego zwierzaka, a nie otoczenia w którym się znajduje czy też pozostały elementów tła.

12. Projekt bazy danych

Wykonał: Bartosz Gruca, sprawdził: Jędrzej Radłowski, zatwierdził: Robert Pytel

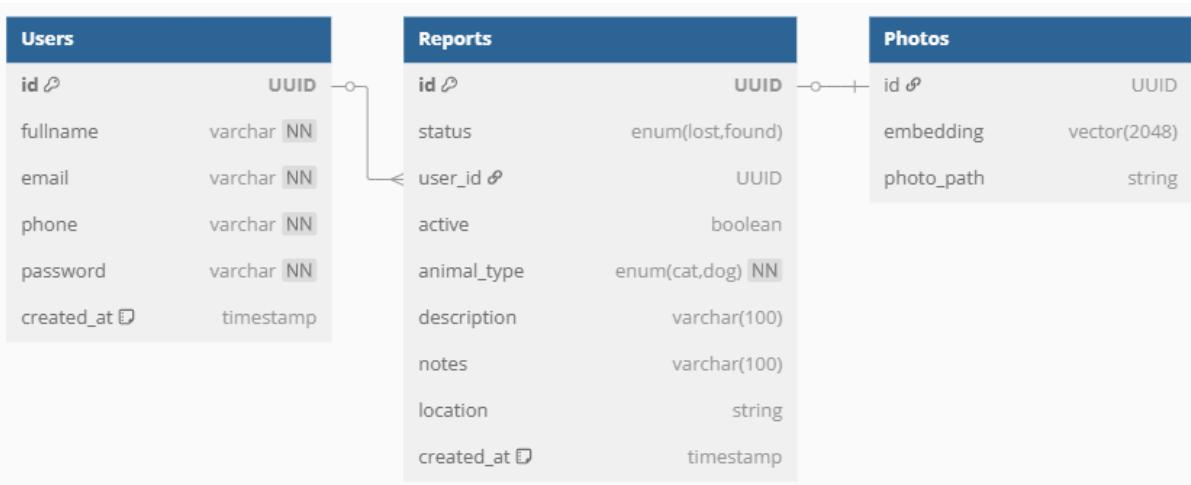
System służący identyfikacji znalezionych zwierząt wymaga zbierania danych w kilku obszarach. Po pierwsze podstawową kwestią są fotografie zwierząt. Zgodnie z założeniami projektowymi właściciele oraz znalazcy zwierząt mają możliwość przesyłania obrazu zwierzaka do systemu. Przesłane zdjęcia muszą być następnie zapisane w bazie danych w celu realizacji najważniejszych scenariuszy działań aplikacji.

Pierwszym scenariuszem jest utworzenie zgłoszenia zaginięcia pupila przez właściciela. W tym celu wypełnia on formularz, w którym wymagane jest umieszczenie zdjęcia zwierzaka. Oprócz tego w formularzu właściciel może podać przybliżoną lokalizację zaginięcia, cechy szczególne zwierzaka czy inne informacje, które mogą być pomocne w identyfikacji zwierzęcia. Dane z przesłanego formularza przetwarzane są przez aplikację serwerową, która w bazie danych zapisuje wszystkie informacje dotyczące zgłoszenia. Oprócz tego zdjęcie przesłane przez właściciela porównywane jest z danymi, które już znajdują się w bazie danych. Jeżeli aplikacja znajdzie w bazie danych zdjęcie zwierzaka, które przypomina pupila przesłanego przez właściciela, informacja ta zostanie przesłana z powrotem do właściciela. W momencie tym właściciel będzie mógł stwierdzić czy rzeczywiście fotografia przedstawia zdjęcie jego pupila. Jeżeli tak, będzie miał on możliwość nawiązania kontaktu ze znalazcą poprzez dane umieszczone w bazie danych.

Drugim scenariuszem wykorzystującym zdjęcie zapisane w bazie danych jest zgłoszenie znalezienia zwierzęcia na ulicy. Znajazca wypełnia formularz, w którym wymagane jest umieszczenie fotografii zwierzęcia. Może również zamieścić pozostałe dane jak cechy szczególne, lokalizacja czy inne ważne informacje. Formularz przesyłany jest do aplikacji serwerowej, która następnie porównuje przesłaną fotografię z fotografiemi zaginionych zwierząt z bazy danych. Jeżeli znajdzie dopasowania, zwraca informację znalazcy w postaci dopasowanej fotografii oraz danych kontaktowych właściciela zguby.

Przedstawione wyżej scenariusze działania aplikacji zakładają, że baza danych będzie przechowywać dane osób, które są znalazcami lub właścicielami zwierząt. W związku z tym struktura bazy danych oprócz przechowywania danych związanych z fotografią, musi również gwarantować możliwość przechowywania danych użytkowników aplikacji.

Mając na uwadze wszystkie elementy świata rzeczywistego wymienione w powyższym opisie utworzono model fizyczny bazy danych.



Rysunek 4: Podstawowy schemat bazy danych.

Na rysunku numer 4 przedstawiono model bazy danych. Wyszczególniono w nich trzy tabele *Users*, *Reports* oraz *Photos*.

W tabeli *Users* zdefiniowano pola na imię oraz nazwisko użytkownika, adres email, numer telefonu, hasło oraz pole ze znacznikiem czasowym, która zawiera informacje o momencie założenia konta przez użytkownika. Na rysunku zaznaczono również typ danych poszczególnych pól. Kluczem głównym tabeli jest pole *id* typu *UUID*. Warto zwrócić uwagę na aspekt bezpieczeństwa hasła. Pole *password* będzie przechowywała wartość wygenerowaną za pomocą algorytmu szyfrującego.

Tabela *Reports* służy do przechowywania informacji o zgłoszeniach utworzonych przez właścicieli lub znalazców pupili. Kluczem głównym tabeli jest pole *id* typu *UUID*. Pole *status* to typ wyliczeniowy, który świadczy o tym czy dane zgłoszenie jest poszukiwaniem (wartość *lost*) czy też znalezieniem (wartość *found*) zwierzęcia. Pole *active* jest wartością logiczną, która świadczy o aktualności danego zgłoszenia. W momencie gdy właściciel odnajdzie zwierzę może zmienić status ogłoszenia na nieaktualne co zostanie odpowiednio odzwierciedlone w tym polu. Pole *animal_type* to typ wyliczeniowy, który świadczy o tym, czy zgłoszenie dotyczy psa czy kota. Pola *description*, *notes* oraz *location* przechowują informacje przesłane w formularzu i dotyczą odpowiednio opisu zwierzaka, cech charakterystycznych oraz lokalizacji, w której zwierzę zostało znalezione lub zgubione. Podobnie jak w przypadku tabeli *Users* ostatnim polem jest *created_at*, która zawiera znacznik czasowy utworzenia zgłoszenia.

Ostatnia tabela zamieszczona w modelu fizycznym bazy danych to tabela *Photos*. Klucz główny tabeli to pole *id* typu *UUID*. Oprócz tego tabela posiada jeszcze dwa pola: *embedding* oraz *photo_path*. Pole *embedding* jest typu *vector(2048)*. Jest to oznaczenie wektora, który zawiera 2048 elementów. Rozmiar pola został dobrany na podstawie modelu sztucznej inteligencji *ResNet-50*, który jest używany w systemie.

Ostatnia warstwa sieci neuronowej modelu generuje wektory rozpięte w przestrzeni 2048 wymiarów. Oznacza to, że na podstawie zdjęcia, które zostało podane na wejście modelu generowany jest wektor o 2048 wymiarach. Logika modelu odpowiedzialna za dopasowanie zdjęć podobnych zwierząt wykorzystuje wartość zapisaną w polu *embedding*. Dokładny opis tego działania został przedstawiony w sekcji “*Projekt modelu sztucznej inteligencji*”. Aby umożliwić przechowywanie i wyszukiwanie podobnych wektorów bez potrzeby korzystania z zewnętrznych narzędzi czy baz danych, w projekcie wykorzystano rozszerzenie **pgvector** dla PostgreSQL. Z kolei pole *photo_path* to ciąg znaków wskazujący na położenie zdjęcia zwierzaka znajdującego się na serwisie hostingowym do przechowywania zdjęć.

Na diagramie oznaczono związki pomiędzy tabelami. Pierwszy związek łączy tabelę *Users* oraz tabelę *Reports*. Typ asocjacji związku to jeden-do-wielu. Oznacza to, że użytkownik aplikacji może stworzyć wiele raportów, jednakże pojedynczy raport jest powiązany tylko z jednym użytkownikiem.

Kolejny związek został oznaczony pomiędzy tabelą *Reports* oraz *Photos*. Typ asocjacji związku to jeden-do-jednego. Oznacza to, że pojedynczy raport powiązany jest tylko i wyłącznie z jednym zdjęciem a to zdjęcie powiązane jest tylko i wyłącznie z tym raportem.

13. Projekt aplikacji serwerowej

Wykonał: Jędrzej Radłowski, sprawdził: Bartosz Gruca, zatwierdził: Robert Pytel

13.1. Opis

Aplikacja serwerowa pełni rolę centralnego komponentu systemu "Lost + Found Pets". Odpowiada ona za przetwarzanie danych pochodzących z aplikacji mobilnej, komunikację z modelem sztucznej inteligencji oraz komunikację z bazą danych. Serwer realizuje kluczowe operacje takie jak rejestracja użytkowników, przetwarzanie zgłoszeń o zaginięciu lub odnalezieniu zwierząt oraz analizę zdjęć przesłanych przez użytkowników.

Cały system został zaprojektowany jako zestaw współpracujących ze sobą komponentów, których zadaniem jest realizacja wymagań zleceniodawcy dotyczących:

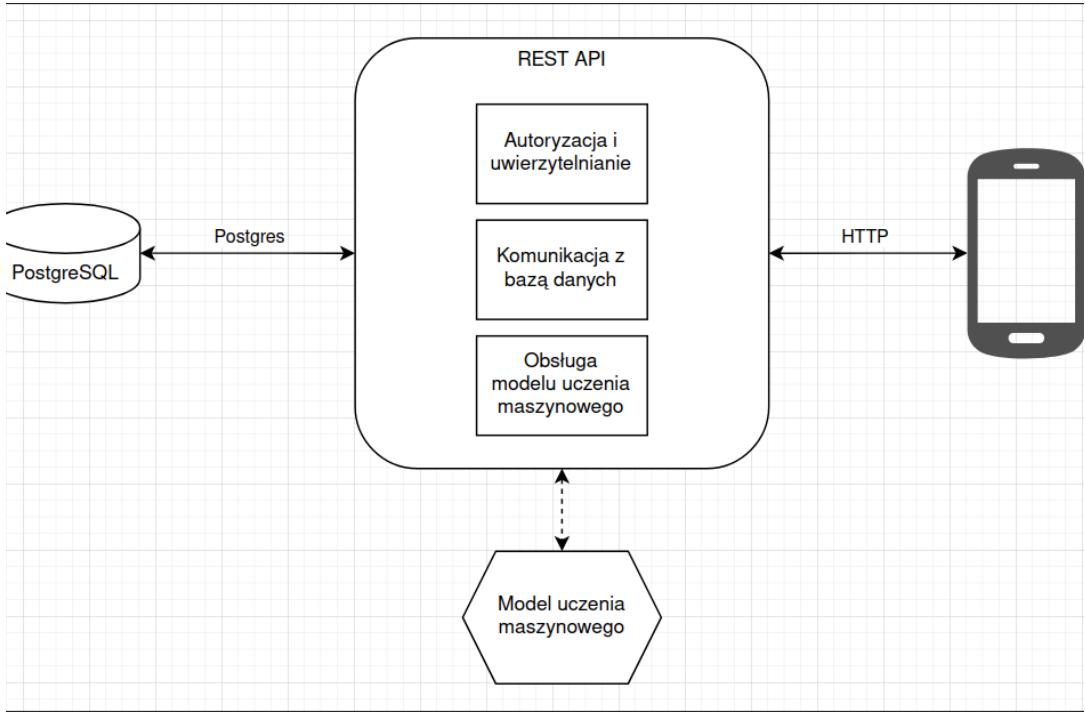
- sprawnego przesyłania i analizy zdjęć,
- automatycznego porównywania obrazów zwierząt przy pomocy AI,
- umożliwienia rejestracji, zgłaszania zaginięć i znalezień oraz przeglądanie dopasowań
- bezpiecznego przetwarzania danych użytkowników

Z powyższych wymagań wynikają kluczowe założenia projektowe:

- system musi działać niezawodnie
- dane muszą być przetwarzane zgodnie z zasadami integralności i spójności
- konieczna jest współpraca z modelem AI i aplikacją mobilną w czasie rzeczywistym
- wszystkie funkcje muszą być dostępne poprzez API

Aby sprostać tym oczekiwaniom, zastosowano architekturę opartą na wzorcu MVC oraz technologie, które bezpośrednio odpowiadają na powyższe potrzeby.

Poniższy diagram ilustruje rolę aplikacji serwerowej jako centralnego ogniska łączącego wszystkie główne moduły systemu.



Rysunek 5: Diagram logiczny projektu z uwzględnieniem aplikacji serwerowej jako centralnej części systemu.

13.2. Architektura aplikacji serwerowej:

Struktura aplikacji opiera się na wzorcu Model-View-Controller (MVC), co zapewnia przejrzystość kodu oraz ułatwia jego rozwój i utrzymanie:

- **Model:** Odpowiada za reprezentację danych oraz reguły biznesowe. Reprezentuje encje takie jak User, Reports, Photos. W tej warstwie realizowana jest walidacja danych przesyłanych przez aplikację mobilną.
- **Routes:** Odpowiada za obsługę żądań HTTP i przekazywanie ich do odpowiednich komponentów. Zapewnia mechanizmy uwierzytelniania i autoryzacji oraz zarządza przepływem danych między modułami.
- **Store:** Reprezentuje warstwę dostępu do bazy danych. Wykonuje operacje zapisu, odczytu oraz aktualizacji danych. Izolacja tej warstwy pozwala na łatwiejsze testowanie.

Dzięki tej architekturze możliwe jest efektywne zarządzanie komunikacją między modułem AI, bazą danych i aplikacją mobilną.

13.3. Główne technologie i ich uzasadnienie

- **FastAPI** – zapewnia wysoką wydajność i bardzo dobrą integrację z bibliotekami AI (Pillow, OpenCV). Automatycznie generuje dokumentację API (Swagger/OpenAPI), co znaczająco usprawnia współpracę z zespołem

frontendowym i przyspiesza iteracyjny rozwój.

- **Uvicorn (ASGI)** – wydajny serwer aplikacyjny, który obsługuje zapytania asynchroniczne, co jest istotne przy obsłudze zadań AI wymagających czasu (np. analiza zdjęć).
- **PostgreSQL + pgvector** – baza danych relacyjna gwarantująca zgodność z ACID, co jest kluczowe przy przechowywaniu danych wrażliwych. Rozszerzenie *pgvector* umożliwia przechowywanie i przeszukiwanie wektorów cech AI bez zewnętrznych narzędzi.
- **SQLModel + Alembic** – do mapowania obiektowo-relacyjnego (ORM) oraz migracji schematów bazodanowych.
- **Pydantic** – walidacja danych wejściowych i wyjściowych. Chroni przed błędami logicznymi oraz zapewnia integralność przesyłanych informacji.
- **JWT (PyJWT)** – mechanizm bezpiecznej autoryzacji oparty na tokenach. Spełnia wymagania RODO w zakresie bezpiecznego zarządzania sesją użytkownika.
- **Celery + Redis** – obsługa zadań asynchronicznych takich jak analiza zdjęć AI czy klasyfikacja obrazu.
- **Pillow / OpenCV** – narzędzia do wstępnego przetwarzania obrazów (np. kadrowanie, przeskalowywanie), co ułatwia analizę przez model AI.

13.4. Powiązania z resztą systemu

Aplikacja serwerowa jest ścisłe zintegrowana z:

- **aplikacją mobilną** – zapewnia obsługę rejestracji, przesyłania zdjęć, zgłoszeń i otrzymywania wyników dopasowania,
- **bazą danych** – zarządza wszystkimi encjami logicznymi systemu: użytkownicy, zdjęcia, zgłoszenia, wektory cech,
- **modelem AI** – realizuje komunikację z komponentem ResNet-50 oraz systemem klasyfikującym YOLO; otrzymuje wektory cech i wykorzystuje je do wyszukiwania podobieństw.

Taka struktura zapewnia **kompletność funkcjonalną** systemu – każda funkcja oferowana użytkownikowi końcowemu opiera się na współdziałaniu tych trzech modułów, z serwerem jako centralnym ogniwem.

13.5. Zabezpieczenia i utrzymanie

System spełnia standardy bezpieczeństwa wymagane przez RODO i dobre praktyki branżowe:

- hasła są szyfrowane algorytmem SHA256,
- tokeny JWT zapewniają bezpieczne sesje użytkownika,
- weryfikacja treści zdjęć (YOLO) ogranicza możliwość przesyłania niepożądanych treści,
- limity zapytań (rate-limiting) chronią system przed atakami typu DDoS (np. 10 zapytań/minutę dla AI),
- dane są archiwizowane regularnie,
- aplikacja serwerowa loguje operacje użytkowników i działania systemowe – co umożliwia audyt, raportowanie i wykrywanie anomalii.

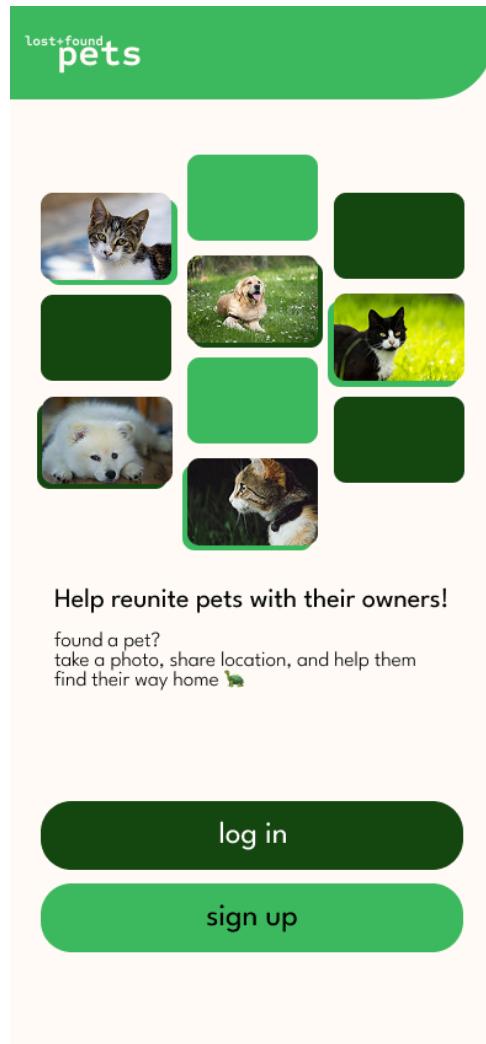
13.6. Utrzymanie i monitorowanie

Aplikacja serwerowa zapewnia mechanizmy utrzymania bazy danych, w tym regularną archiwizację danych oraz raportowanie statystyk działania systemu. Działania użytkowników oraz operacje serwera są logowane w celu zapewnienia transparentności oraz szybkiego wykrywania i eliminacji błędów lub nadużyć.

14. Projekt interfejsu graficznego aplikacji użytkownika

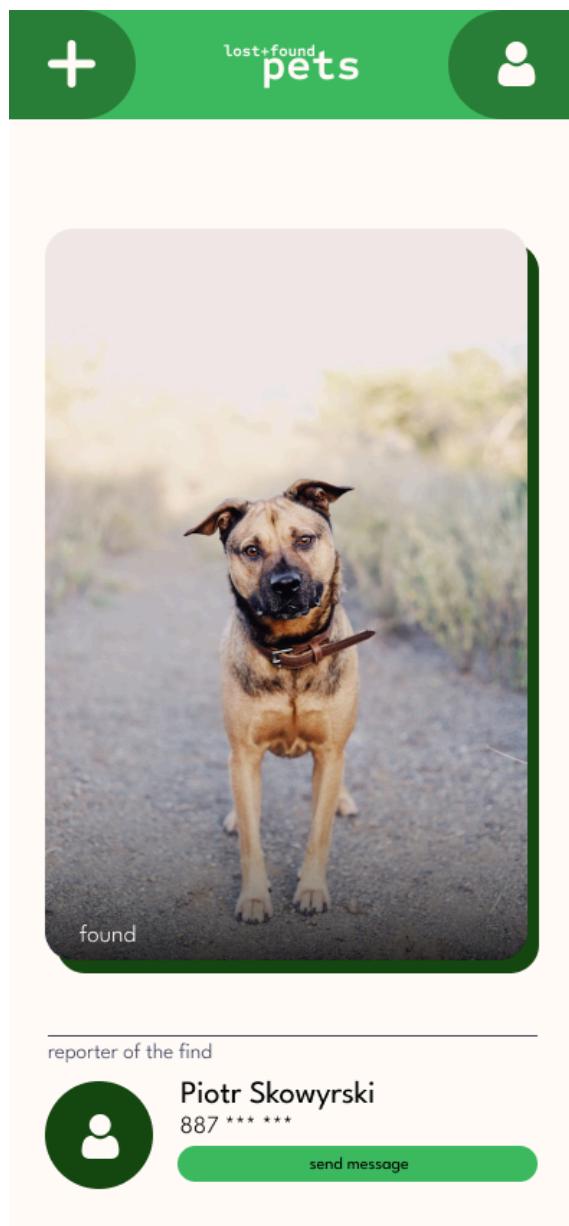
Wykonał: Mateusz Oleksy, sprawdził: Bartosz Gruca, zatwierdził: Robert Pytel

Projekt interfejsu graficznego obejmuje widoki wszystkich możliwych stanów, w których aplikacja mobilna może się znajdować.



Rysunek nr 6: Ekran główny aplikacji mobilnej dla niezalogowanego użytkownika.

Na rysunku 6 przedstawiono główny ekran aplikacji mobilnej użytkownika. Ekran ten przedstawiony po uruchomieniu aplikacji w momencie, gdy użytkownik nie ma zapisanej sesji logowania. W głównej części ekranu wyświetlana jest infografika przedstawiająca główny cel aplikacji, którym jest pomóc w odnalezieniu zaginionych zwierząt. Z kolei w dolnej części ekranu znajdują się przyciski, które przenoszą użytkownika odpowiednio do ekranu logowania oraz ekranu rejestracji.



Rysunek nr 7: Ekran główny aplikacji mobilnej dla zalogowanego użytkownika.

Na rysunku nr 7 Na rysunku 6 przedstawiono główny ekran aplikacji mobilnej użytkownika, który posiada aktywną sesję logowania. W górnej części ekranu znajduje się menu zawierające dwa przyciski. Po kliknięciu na ikonę plusa użytkownik zostaje przeniesiony do ekranu dodawania zgłoszenia. Z kolei po kliknięciu na ikonę człowieka, użytkownik zostaje przeniesiony do ekranu profilu użytkownika.

W głównej części ekranu wyświetlane jest losowe zgłoszenie zaginionego lub znalezionej zwierzęcia, które znajduje się w bazie danych systemu. Wyświetlane są zarówno informacje kontaktowe osoby zgłaszającej, a także grafika zwierzęcia.

The image shows a registration form for the 'lost+found pets' application. The form is contained within a dark green rounded rectangle. At the top, the text 'Sign in to lost+found pets' is displayed, followed by the instruction 'Enter your email and password to acces your account'. Below this, there are four input fields: 'full name' (placeholder: e.g. Piotr Skowyrski), 'phone number' (placeholder: e.g. 123 456 789), 'email' (placeholder: e.g. piotrskowyrski@gmail.com), and 'password' (placeholder: e.g. lostfoundpetsareisthebest). At the bottom of the form is a large green button with the text 'sign up'. Below the button, the text 'already have an account? [login](#)' is visible.

Sign in to lost+found pets
Enter your email and password to acces your account

full name
e.g. Piotr Skowyrski

phone number
e.g. 123 456 789

email
e.g. piotrskowyrski@gmail.com

password
e.g. lostfoundpetsareisthebest

sign up

already have an account? [login](#)

Rysunek nr 8: Formularz rejestracji użytkownika

Na rysunku 8 przedstawiono widok formularza rejestracji użytkownika. Formularz zawiera następujące pola: imię i nazwisko użytkownika, numer telefonu komórkowego użytkownika, e-mail użytkownika oraz hasło użytkownika. W dolnej części ekranu znajduje się zielony przycisk z napisem *sign up*, służący do przesłania formularza rejestracji do aplikacji serwerowej.

Pod zielonym przyciskiem przesyłania formularza znajduje się link przekierowujący użytkownika do widoku logowania. Jest on przeznaczony dla użytkowników, którzy posiadają już konto w systemie.

The screenshot shows the login interface for the lost+found pets application. At the top, there is a green header bar with the logo 'lost+found pets'. Below the header, the main content area has a dark green background. It features a title 'Log in to lost+found pets' and a subtitle 'Enter your email and password to access your account'. There are two input fields: 'email' with placeholder text 'e.g. piotrskowyrski@gmail.com' and 'password' with placeholder text 'e.g. lostfoundpetsareisthebest'. At the bottom of the form is a large green button labeled 'log in'. Below the button, a link 'don't have an account? [sign up](#)' is visible.

Log in to lost+found pets
Enter your email and password to access your account

email
e.g. piotrskowyrski@gmail.com

password
e.g. lostfoundpetsareisthebest

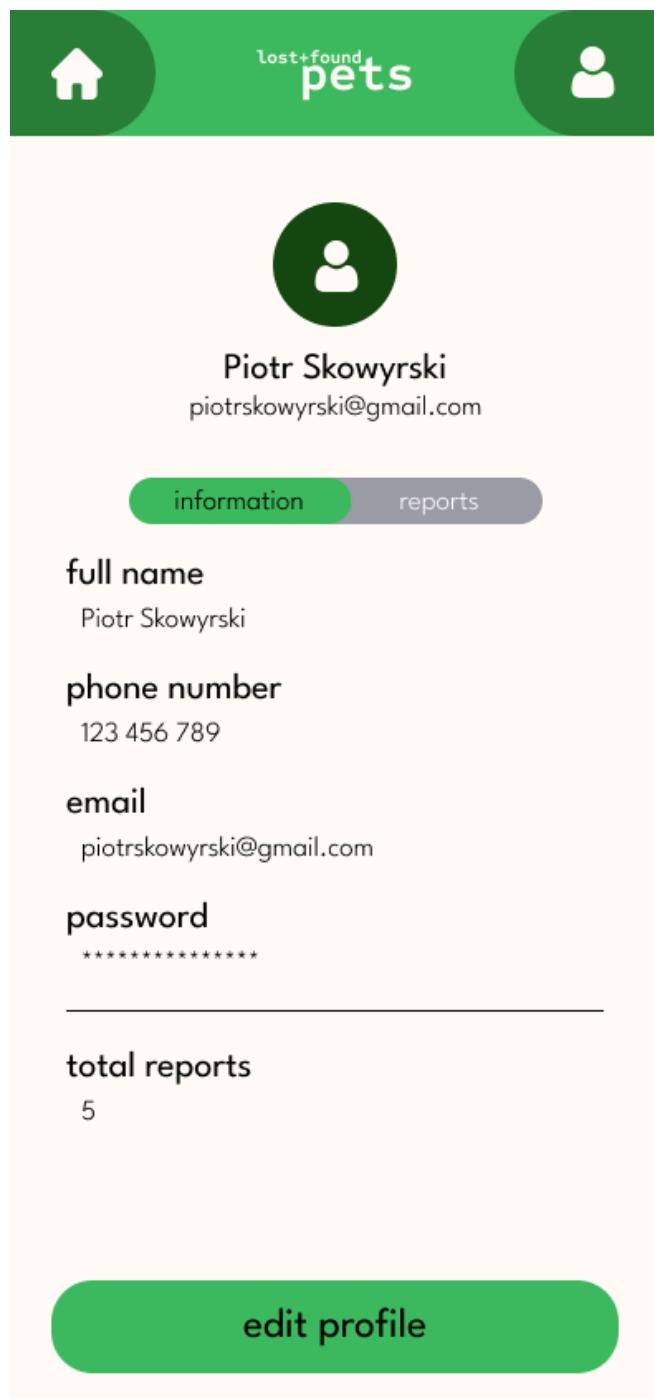
log in

don't have an account? [sign up](#)

Rysunek nr 9: Formularz logowania użytkownika

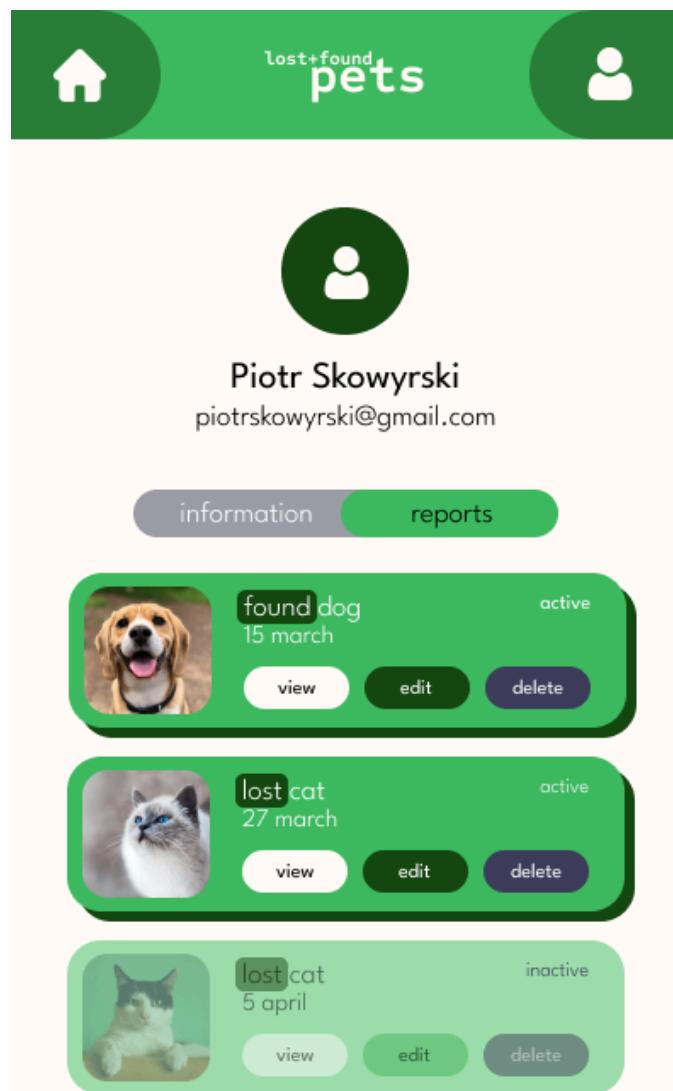
Na rysunku 9 przedstawiono widok formularza logowania użytkownika. Formularz zawiera następujące pola: e-mail użytkownika oraz hasło użytkownika. W dolnej części ekranu znajduje się zielony przycisk z napisem *log in*, służący do przesłania formularza logowania do aplikacji serwerowej.

Pod zielonym przyciskiem przesyłania formularza znajduje się link przekierowujący użytkownika do widoku rejestracji. Jest on przeznaczony dla użytkowników, którzy nie posiadają jeszcze konta w systemie.



Rysunek nr 10: Widok profilu użytkownika. Sekcja danych użytkownika

Na rysunku nr 10 przedstawiono widok profilu użytkownika, prezentujący dane użytkownika aplikacji mobilnej. Przedstawione są tutaj wszystkie informacje, które użytkownik podaje w formularzu rejestracji, a mianowicie imię i nazwisko użytkownika, numer telefonu komórkowego, e-mail oraz hasło. Pole hasło jest jednakże tylko symbolem zastępczym, nie zawiera ono faktycznego hasła użytkownika. Jest ono wykorzystywane w momencie gdy użytkownik naciśnie przycisk z napisem *edit profile*, który pozwala na zmianę poszczególnych informacji, w tym także hasła. Ostatnią podaną informacją jest łączna ilość wykonanych zgłoszeń wykonanych przez użytkownika.



Rysunek nr 11: Widok profilu użytkownika. Sekcja historii zgłoszeń

Na rysunku nr 11 przedstawiono widok profilu użytkownika, prezentujący zakładkę historii zgłoszeń wykonanych przez użytkownika. Widok pojedynczego zgłoszenia zawiera informacje o grafice zwierzaka, typie zgłoszenia (znalezienie lub zaginięcie zwierzaka). Oprócz tego zawiera datę utworzenia zgłoszenia oraz to czy jest ono aktywne. Poza tym każde zgłoszenie zawiera trzy przyciski. Pierwszy z nich opisany jako *view* przenosi użytkownika do widoku zawierającego pełne informacje o danym zgłoszeniu. Przycisk *edit* przenosi użytkownika do widoku, w którym możliwa jest edycja danych ogłoszenia. Ostatni przycisk *delete* służy do usunięcia zgłoszenia.

The image displays two side-by-side screenshots of a mobile application interface for reporting lost or found pets. Both screens have a green header bar with a home icon, user profile icon, and the text "lost+found pets".

Left Screen (Empty Form):

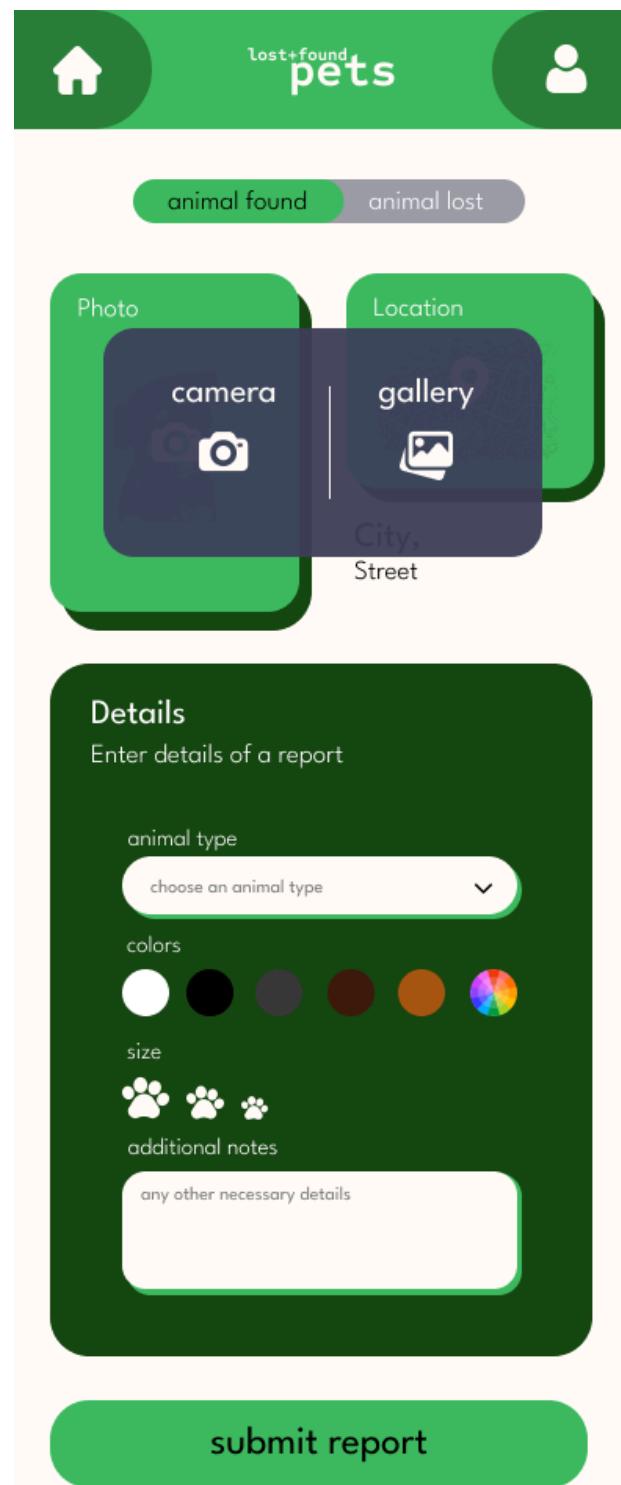
- At the top, there are two buttons: "animal found" (green) and "animal lost" (grey).
- Below them are two rounded square buttons: "Photo" (with a camera icon) and "Location" (with a map pin icon).
- Under the "Location" button is the text "City, Street".
- A large dark green rectangular area labeled "Details" contains the placeholder text "Enter details of a report".
- Below this are sections for "animal type" (a dropdown menu showing "choose an animal type"), "colors" (a row of six color swatches), "size" (three paw print icons), and "additional notes" (a text input field with placeholder "any other necessary details").
- At the bottom is a large green "submit report" button.

Right Screen (Completed Form):

- At the top, there are two buttons: "animal found" (green) and "animal lost" (grey).
- Below them are two rounded square buttons: "Photo" (with a dog image) and "Location" (with a map pin icon).
- Under the "Location" button is the text "Wroclaw, Plac grunwaldzki 5".
- A large dark green rectangular area labeled "Details" contains the placeholder text "Enter details of a report".
- Below this are sections for "animal type" (set to "cat"), "colors" (a row of six color swatches), "size" (three paw print icons), and "additional notes" (a text input field containing the note "The cat has wary golden eyes and watches closely, seeming hopeful but hesitant").
- At the bottom is a large green "submit report" button.

Rysunek nr 12 i 13: Widoki dodawania nowego zgłoszenia bez oraz z przykładowymi danymi

Na rysunkach 12 i 13 przedstawiono widok dodawania nowego zgłoszenia. Po lewej stronie widok bez wprowadzonych danych, natomiast po prawej stronie znajduje się uzupełniony formularz. Na początku użytkownik dokonuje wyboru za pomocą przełącznika *animal found / animal lost*. Pierwszy z nich oznacza, że użytkownik znalazł zwierzę, drugi, że poszukuje swojego pupila. Formularz zawiera pole na fotografię zwierzaka oraz lokalizację, w której zgubiono czy też znaleziono zwierzę. Kolejnym polem formularza jest wybór typu zwierzęcia (kot lub pies). Oprócz tego użytkownik może wybrać kolory zwierzęcia oraz jego przybliżony rozmiar. Dodatkowo w formularzu znajduje się pole tekstowe, w którym użytkownik może uwzględnić cechy szczególne zwierzęcia. W dolnej części ekranu znajduje się przycisk, który przesyła formularz do aplikacji serwerowej.



Rysunek nr 14: Widok dodawania nowego zgłoszenia. Wybór zdjęcia zwierzaka

Na rysunku 14 przedstawiono widok dodawania nowego zgłoszenia z uwzględnieniem widoku dodawania fotografii zwierzaka. Użytkownik ma do wyboru możliwość wykonania fotografii za pomocą aparatu lub też wysłania zdjęcia, które już znajduje się w jego galerii.



Rysunek nr 15: Widok edycji zgłoszenia

Na rysunku 15 przedstawiono widok edycji zgłoszenia. Zawiera on dokładnie te same pola co formularz dodawania zgłoszenia, który został przedstawiony na rysunku 12 i 13. W momencie załadowania tego widoku konkretne pola są już uzupełnione. Użytkownik może zmienić ich wartość a następnie przesłać formularz za pomocą przycisku *edit report*.

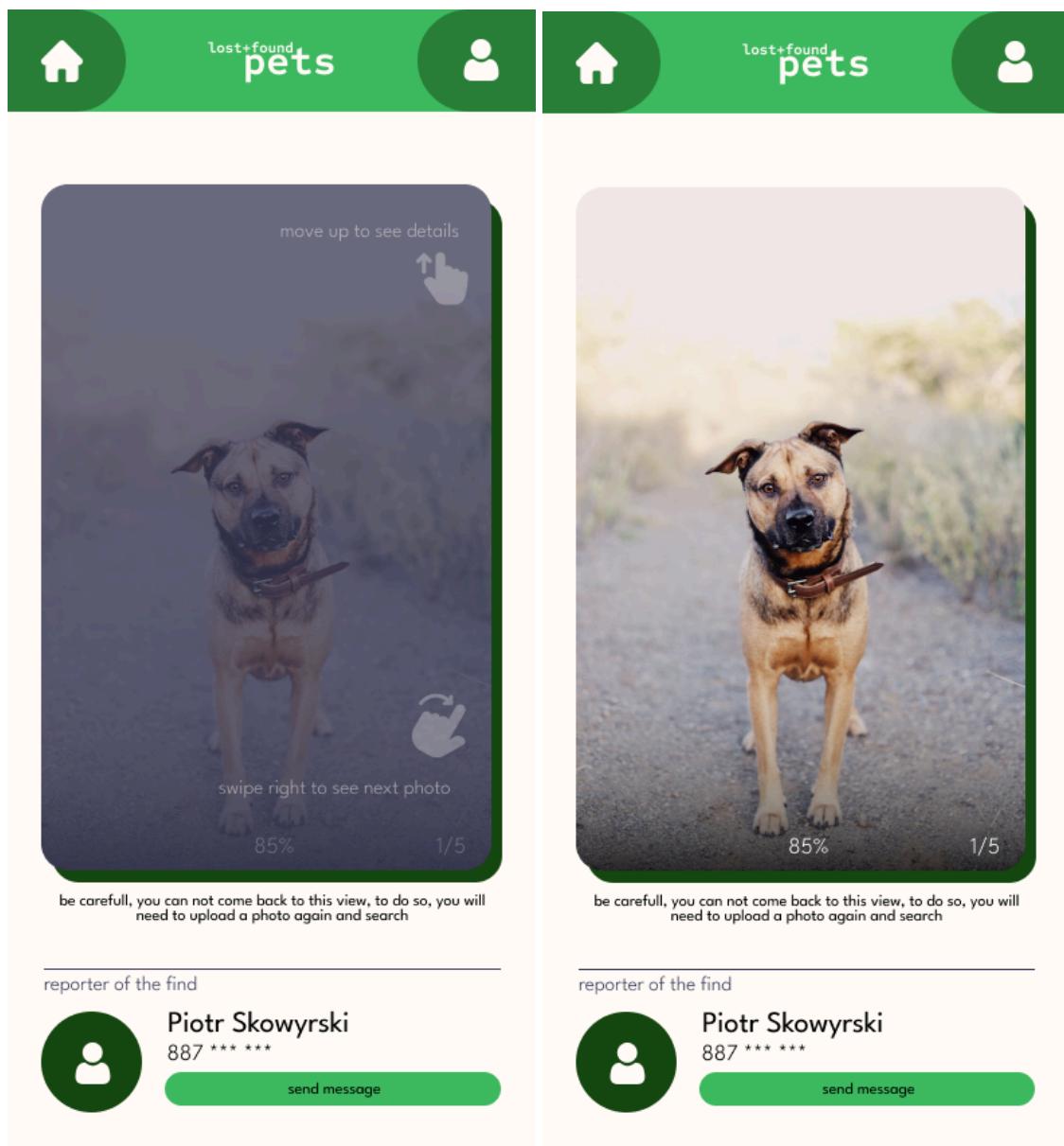


Searching matching photos...

We hope that your pet will get found soon ❤️

Rysunek nr 16: Widok oczekiwania na wyniki dopasowań

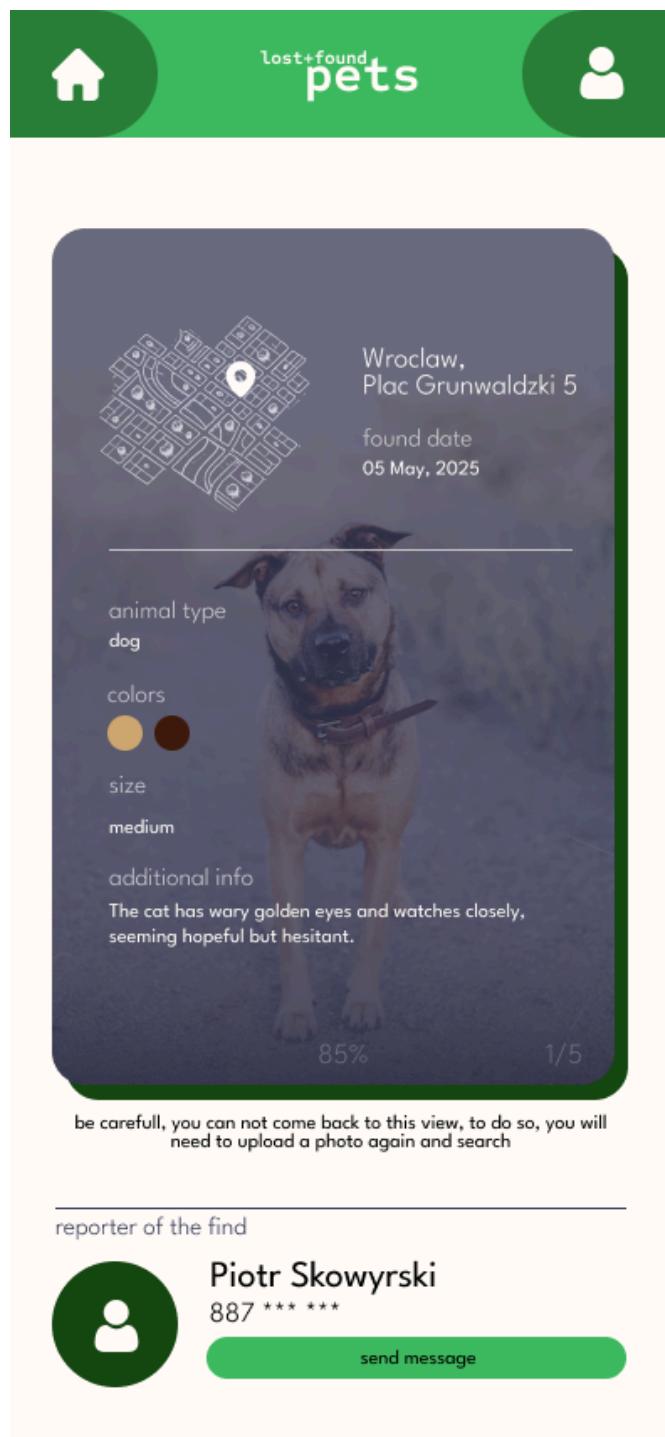
Na rysunku 16 przedstawiono ekran oczekiwania na wyniki dopasowań po wysłaniu formularza zgłoszenia. W środkowej części ekranu znajduje się okrągła ikonka. W momencie oczekiwania na odpowiedź ikonka kręci się, aby zasygnalizować użytkownikowi, że aplikacja mobilna wciąż pracuje i nie zawiesiła się.



Rysunek nr 17 i 18: Widoki dopasowanych zgłoszeń

Na rysunkach 17 i 18 przedstawiono widoki dopasowanych zgłoszeń. W środkowej części widoku znajduje się zdjęcie zwierzaka oraz wartość funkcji podobieństwa, wyrażona w procentach, której użyto w celu wyszukania podobnych zdjęć w bazie danych do zdjęcia zamieszczonego w formularzu. W prawym dolnym rogu ilustracji znajduje się liczba dopasowanych zgłoszeń. Rysunek nr 17 posiada dodatkowo instrukcję obsługi interfejsu. Aby uzyskać więcej informacji na temat zgłoszenia użytkownik ma przesunąć zdjęcie do góry. W celu wyświetlania kolejnego zdjęcia, które uzyskało wysoką wartość funkcji podobieństwa, użytkownik musi przesunąć palcem w prawo.

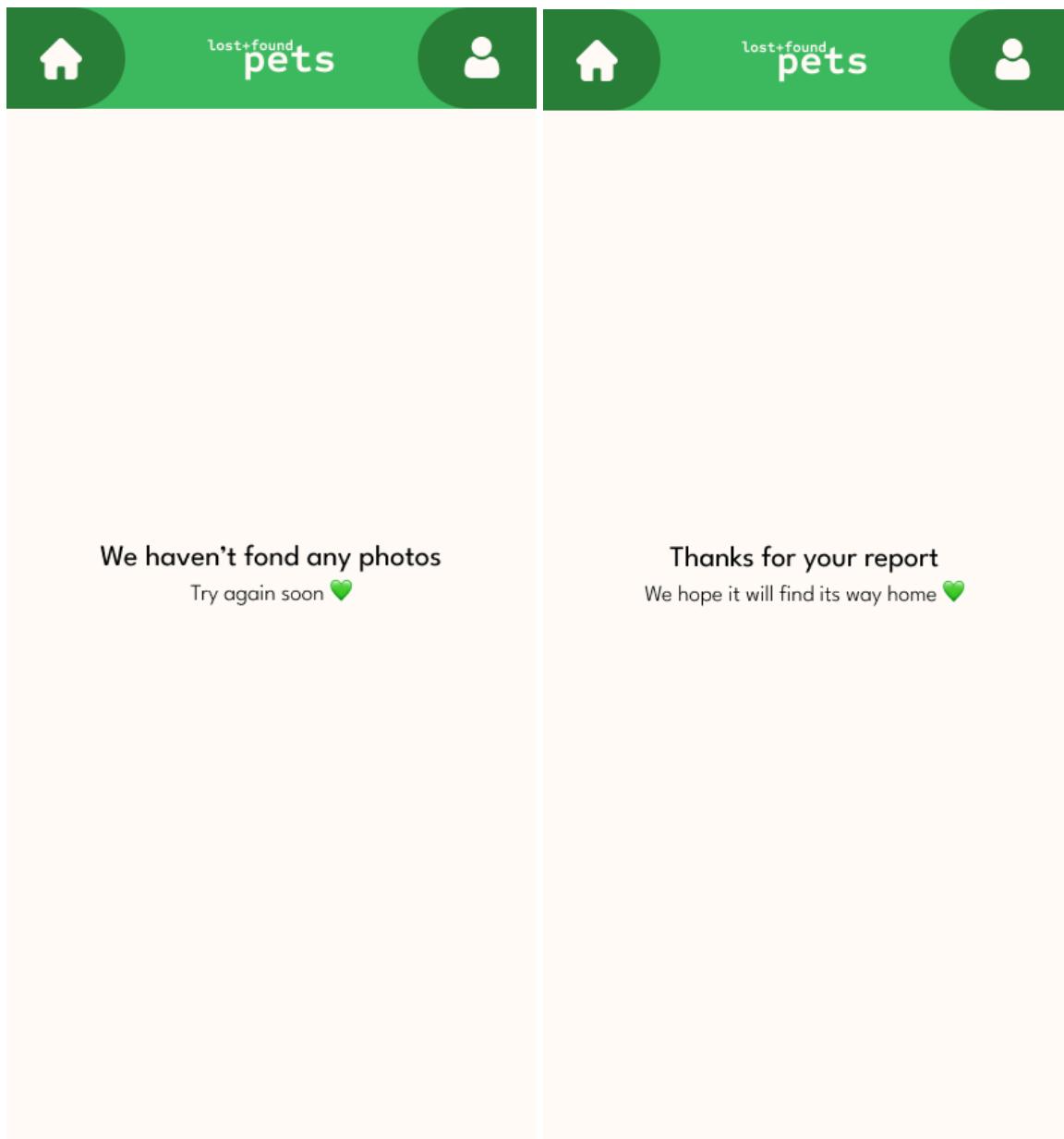
W dolnej części interfejsu znajdują się dane osobowe użytkownika, który jest autorem dopasowanego zgłoszenia. Informacje obejmują imię i nazwisko oraz numer telefonu komórkowego, który zostanie ukazany po kliknięciu przycisku *send message*.



Rysunek nr 19: Szczegóły dopasowanego zgłoszenia

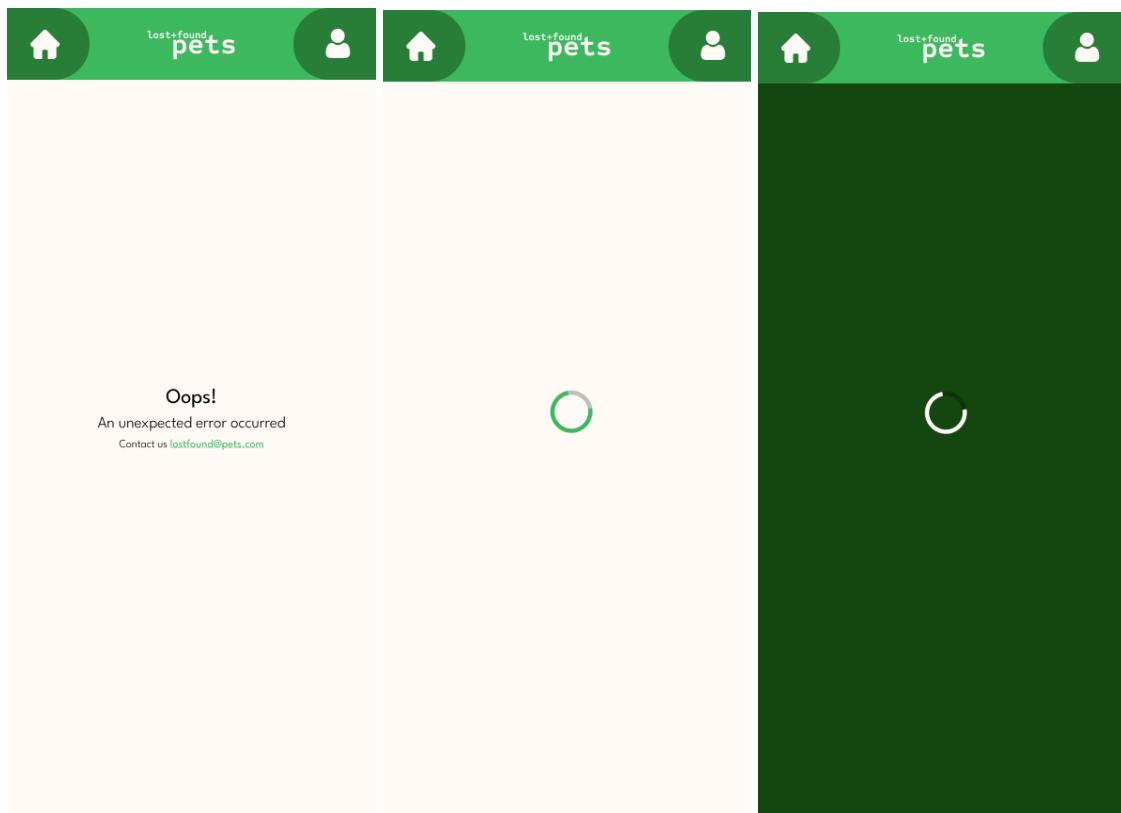
Rysunek nr 19 jest uzupełnieniem rysunków 17 i 18. Zawiera widok szczegółów dotyczących danego zgłoszenia. Widok ten widoczny jest po przesunięciu palcem w góre widoku przedstawionego na rysunku 18.

Informacje obejmują lokalizację zaginięcia lub znalezienia zwierzęcia. Oprócz tego widok zawiera datę utworzenia zgłoszenia, typ zwierzęcia, kolor i rozmiar zwierzaka, a także pole tekstowe zawierające dodatkowe informacje i cechy szczególne pupila.



Rysunek 20 i 21: Widoki informujące użytkownika

Na rysunku 20 przedstawiono widok, który jest wyświetlany użytkownikowi w przypadku braku zgłoszeń w bazie danych. Z kolei rysunek 21 przedstawia informację o pozytywnym utworzeniu zgłoszenia w bazie danych.



Rysunek 21, 22, 23: Widoki uzupełniające

Na rysunkach 21, 22 oraz 23 przedstawiono widoki uzupełniające używane w aplikacji mobilnej. Rysunek 21 przedstawia komunikat o wystąpieniu niespodziewanego błędu w aplikacji mobilnej. Oprócz tego podany jest adres e-mail, na który można zgłosić wystąpienie tego błędu.

Na rysunkach 22 oraz 23 przedstawiono z kolei ekran ładowania.

15. Tworzenie i zapisywanie wzorców cech w bazie danych. Wyszukiwanie podobnych zwierząt na podstawie wzorców cech.

Wykonał: Robert Pytel, sprawdził: Bartosz Gruca, zatwierdził: Mateusz Oleksy

15.1 Tworzenie wzorców cech

Zgodnie z informacjami zawartymi w sekcji poświęconej projekcie modelu sztucznej inteligencji, podstawową funkcjonalnością jest tworzenie wzorców cech na podstawie zdjęcia wejściowego.

Dla przypomnienia, każda fotografia zwierzęcia reprezentowana jest za pomocą wektora liczb zwanego inaczej wartościami cech. Wartości te obliczane są przez model w momencie przetwarzania fotografii przez sieć neuronową. Za pomocą wyznaczonego wektora cech przypisanego do konkretnego obiektu możemy jednoznacznie określić położenie tego obiektu w przestrzeni cech, która ma tyle wymiarów ile liczb zawiera wektor (ile jest cech). W przypadku modelu *ResNet-50* wektor ten zawiera 2048 wartości, co daje przestrzeń o 2048 wymiarach. Od tego momentu fotografia zwierzaka może być pojmowana jako punkt w tak wielowymiarowej przestrzeni cech. Ważną informacją jest to, że obiekty o zbliżonych cechach będą znajdować się w niewielkiej odległości od siebie w przestrzeni cech.

Podsumowując powyższy opis, zdjęcia przesyłane przez użytkowników przetwarzane są przez model *ResNet-50*, na którego wyjściu możliwe jest wyodrębnienie informacji o wektorach cech zdjęcia w postaci wektora o 2048 wartościach. Następnie system zapamiętuje dany wektor cech w bazie danych. Wektor ten powiązany jest również z konkretnym zgłoszeniem znajdującym się w bazie danych.

15.2 Przykład wyszukiwania podobnych zwierząt na podstawie wzorców cech

W celu zaprezentowania działania wyszukiwania na podstawie wektorów cech wykonano następujące operacje na przykładzie psa imieniem Cyprus. Do bazy dodano wektory cech czterech fotografii przedstawiających pupila. Następnie wykorzystano funkcję wyszukiwania fotografii przedstawiających fotografie podobnych zwierząt, która została zdefiniowana na rysunku 26. Zdjęcie wzorcowe zostało przedstawione na rysunku 27, natomiast rezultaty wykonania funkcji przedstawiono na rysunku 28.



Rysunek 27: Fotografia wzorcowa



Rysunek 28: Fotografie z bazy danych, które osiągnęły najmniejsze wyniki pod względem funkcji dystansu kosinusowego w porównaniu ze zdjęciem wzorcowym

Na rysunku 28 przedstawiono wynik wykonania funkcji zdefiniowanej na rysunku 26 w postaci fotografii oraz funkcji dystansu kosinusowego, które uzyskały one w porównaniu ze zdjęciem wzorcowym. W tym przypadku trzy pierwsze fotografie, które osiągnęły największą wartość dystansu kosinusowego przedstawiają tego samego psa, który uchwycony jest na zdjęciu wzorcowym.

16. Ekran powitalny, formularze logowania i rejestracji w aplikacji użytkownika

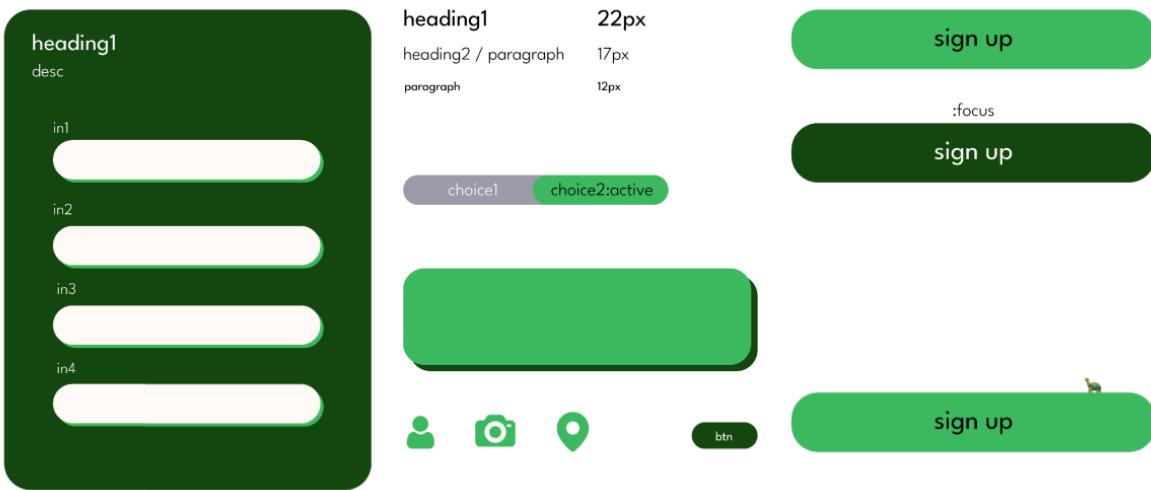
Wykonał: Mateusz Oleksy, sprawdził: Bartosz Gruca, zatwierdził: Robert Pytel

Formularze aplikacji

Formularze w naszej aplikacji użytkownika będą pozwalały na zarejestrowanie oraz zalogowanie się danemu użytkownikowi. Dane zostaną przekazane poprzez interfejs API na serwer, gdzie nastąpi wgląd do bazy danych.

16.1 Stworzenie podstawowych komponentów aplikacji

Zgodnie z wcześniej przygotowanym projektem interfejsu użytkownika, wyodrębniono podstawowe komponenty.



Rysunek 29: Podstawowe komponenty

Podstawowe komponenty, to takie komponenty, które będą potrzebne w całej aplikacji i powtarzają się więcej niż 2 razy, na różnych widokach, dlatego deklarujemy je jako globalne.

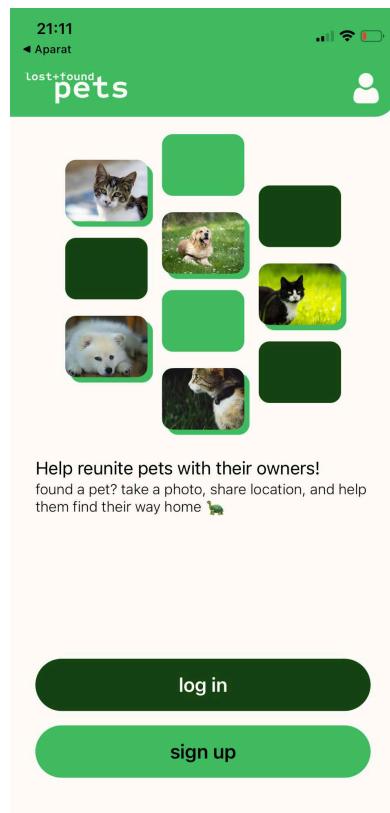
Z rysunku można wyróżnić:

- formularz
- nagłówki
- pole wyboru
- ikona
- drugorzędny przycisk
- pierwszorzędny przycisk

16.2 Ekran powitalny

Przy pierwszym wejściu do aplikacji wita nas ekran powitalny.

Ten widok ma za zadanie przenieść nas do formularza rejestracji lub logowania, gdzie użytkownik będzie mógł wpisać swoje dane.

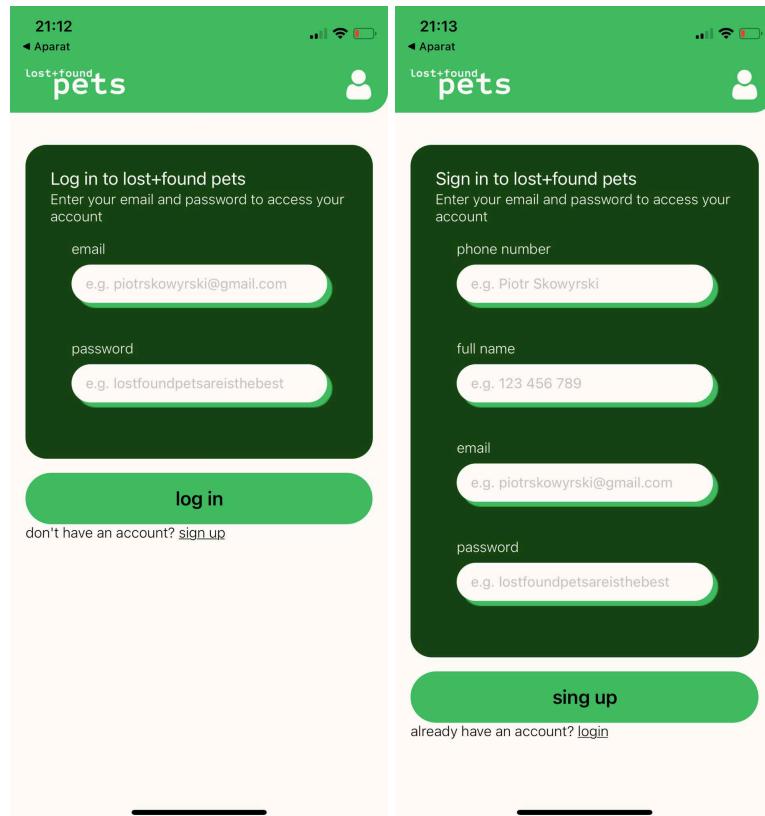


Rysunek 32: Widok powitalny

16.3 Formularz rejestracji i logowania

Zarówno formularz rejestracji i formularz logowania są stworzone przy użyciu wcześniejszych zdefiniowanych komponentów.

Użyto formularza oraz primaryButton.



Rysunek 33: Widok rejestracji i logowania

17. Obsługa rejestracji i uwierzytelniania użytkowników po stronie aplikacji serwerowej

Wykonał: Jędrzej Radłowski, sprawdził: Mateusz Oleksy, zatwierdził: Robert Pytel

Kolejną kluczową funkcjonalnością, która musi być obsługiwana przez aplikację serwerową jest rejestracja i uwierzytelnianie użytkowników aplikacji. Jest to niezbędna część aplikacji ze względu na możliwość nawiązywania kontaktu pomiędzy znajomymi pupilów na ulicy a ich właścicielami.

Oprócz tego uwierzytelnianie użytkowników pomaga monitorować przesyłane treści. Jeżeli użytkownicy będą przesyłali zdjęcia, które mają na uwadze celowe zaburzenie lub pogorszenie działania systemu, mogą zostać wykluczeni z użytkowania aplikacji.

Podstawową kwestią dotyczącą rejestracji i uwierzytelniania użytkowników po stronie aplikacji serwerowej jest obsługa wysłanych formularzy. Po stronie aplikacji użytkownika znajdują się formularze odpowiednio dla logowania oraz rejestracji, które są następnie przesyłane do aplikacji serwerowej. Podstawową funkcją wykonywaną po stronie serwera jest walidacja danych przesłanych w formularzach przez użytkownika pod kątem poprawności oraz spójności danych.

18. Detektor psów i kotów na zdjęciu wejściowym

Wykonał: Bartosz Gruca, sprawdził: Jędrzej Radłowski, zatwierdził: Robert Pytel

Aby baza danych nie była zaśmiecana przez przypadkowo lub złośliwie przesłane zdjęcia, które nie przedstawiają poszukiwanych pupili, aplikacja musi w wiarygodny sposób rozpoznawać i walidować treść fotografii. Do realizacji tego zadania wybraliśmy detektor YOLO.

Detektor ten wybrano ze względu na jego popularność, otwartą licencję oraz dopasowanie do naszych potrzeb, obejmujących:

- **Detekcję obiektów** – identyfikację i lokalizację elementów na obrazie.
- **Segmentację instancji** – wykrywanie obiektów wraz z precyzyjnym wyodrębnieniem ich krawędzi.
- **Klasifikację obrazów** – przypisywanie zdjęć do zdefiniowanych kategorii.

19. Testy skuteczności modelu w wyszukiwaniu zaginionych zwierząt

Wykonał: Robert Pytel, sprawdził: Jędrzej Radłowski, zatwierdził: Mateusz Oleksy

Mając opracowany klasyfikator rasy psa na podstawie zdjęcia wejściowego, należało sprawdzić jak model poradzi sobie z identyfikacją konkretnego osobnika na podstawie fotografii wejściowej oraz tych znajdujących się w bazie danych.

W związku z tym pierwszym krokiem do wykonania testów było zebranie zbioru danych zdjęć, w których każdy z osobników posiada przynajmniej dwie fotografie. Założenie to musi być prawdziwe, ponieważ jedna fotografia ma znajdować się w bazie danych, natomiast druga ma posłużyć jako wzorzec do przeszukiwania bazy danych.

Finalnie uzbierano zbiór danych obejmujący 7000 osobników, gdzie każdy osobnik posiada przynajmniej dwie fotografie.

Metoda testowania systemu została oparta na głównym scenariuszu działania aplikacji, w którym użytkownik wysyła zdjęcie znalezionego lub zaginionego psa do systemu, a następnie otrzymuje zbiór zdjęć wynikowych znajdujących się w bazie danych, które otrzymały najmniejszą wartość pod względem funkcji dystansu kosinusowego lub odległości euklidesowej.

W testach postanowiono sprawdzić jak rozmiar bazy danych, moc zbioru zdjęć wynikowych oraz użycie dystansu kosinusowego i odległości kosinusowej wpływa na skuteczność działania systemu. Skuteczność w tym przypadku określona jest wzorem:

$$S(N, top_k) = \frac{X}{N}$$

N – liczba osobników w bazie danych

top_k – moc zbioru zdjęć wynikowych

X – liczba szukań, w których

osobnik ze zdjęcia wejściowego znalazł się w zbiorze wynikowym o mocy top_k

Wzór 3: skuteczność modelu w wyszukiwaniu psów

Testy wykonano dla trzech wartości parametru N: 100, 200 oraz 400. W przypadku parametru top_k testy wykonano również dla trzech wartości: 5, 10 oraz 20.

Przebieg pojedynczego testu można przedstawić w postaci listy kroków:

1. Losowy wybór N różnych psów ze zbioru 7000 osobników.
2. Losowy wybór 2 fotografii dla każdego z N osobników.
3. Dodanie pierwszej z 2 fotografii do bazy danych dla każdego z N osobników.
4. Mając bazę danych N fotografii wykonaj przeszukiwanie bazy na podstawie fotografii, która nie została dodana do bazy danych dla każdego z N osobników.
5. Obliczenie liczby przypadków, w których osobnik z fotografii wejściowej znalazł się w zbiorze wynikowym o mocy top_k .
6. Obliczenie wartości skuteczności zgodnie ze wzorem nr 3.

Dla każdej kombinacji wartości parametrów N oraz top_k pojedynczy test został wykonany 10-krotnie. Na podstawie 10 wykonań obliczono średnią arytmetyczną wartość funkcji S dla danej kombinacji parametrów N oraz top_k .

Number of distinct dogs in database	Number of matches with highest similarity score					
	5		10		20	
	Vector metric		Vector metric		Vector metric	
Number of distinct dogs in database	Cosine distance	Euclidean distance	Cosine distance	Euclidean distance	Cosine distance	Euclidean distance
100	75.64	76.10	87.70	85.10	92.00	92.10
200	68.45	67.35	77.80	76.90	85.89	85.15
400	58.03	58.33	69.30	67.83	78.78	79.08

Tabela 4: Wyniki testów wyszukiwania osobnika na podstawie fotografii w bazie danych

W tabeli numer 4 przedstawiono wyniki wykonania testów wykonanych zgodnie z listą kroków, która znajduje się w górnej części strony. Lewa kolumna tabeli zawiera wartość parametru N czyli liczbę osobników znajdujących się w bazie danych. W prawej części tabeli trzy główne kolumny określają wartość parametru top_k . Każda główna kolumna podzielona jest na dwie mniejsze, jedną dla metryki dystansu kosinusowego, drugą dla odległości euklidesowej. Wartości zawarte w komórkach o kolorze białym określają średnią wartość funkcji skuteczności modelu opisaną na wzorze nr 3 dla 10 wykonań testu z ustalonymi parametrami N , top_k oraz wybraną funkcją pomiędzy wektorami cech.

Na podstawie wyników zawartych w tabeli numer 4 można zauważyc, że średnia wartość skuteczności spada wraz ze wzrostem ilości osobników w bazie danych. Jeżeli mowa o parametrze top_k oznaczającym moc zbioru wynikowego to wraz ze wzrostem tego parametru średnia wartość skuteczności modelu wzrasta.

20. Wysłanie nowego raportu zaginionego zwierzęcia

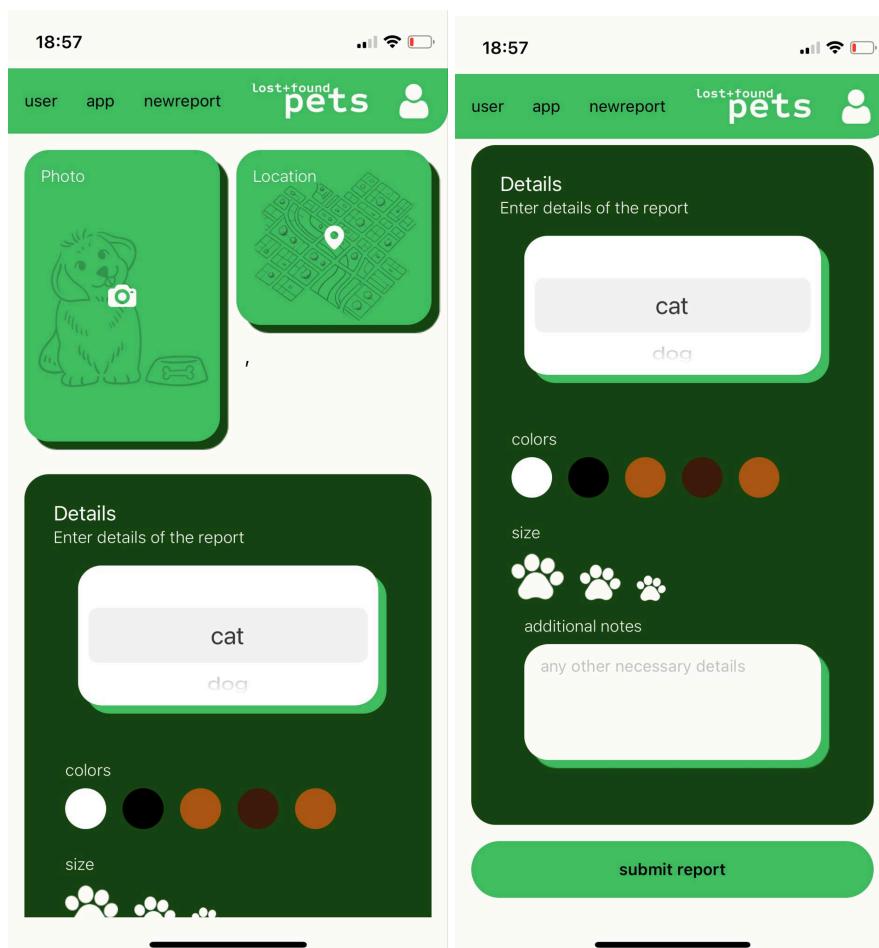
Wykonał: Mateusz Oleksy, sprawdził: Bartosz Gruca, zatwierdził: Robert Pytel

20.1 Nowy raport dotyczący zaginionego zwierzęcia

Wysyłanie raportów to główna część naszej aplikacji, dlatego po stronie użytkownika zapewniliśmy niezbędne elementy, które pozwolą na łatwe dodanie zgłoszenia.

Formularz nowego zgłoszenia wymaga podania od użytkownika:

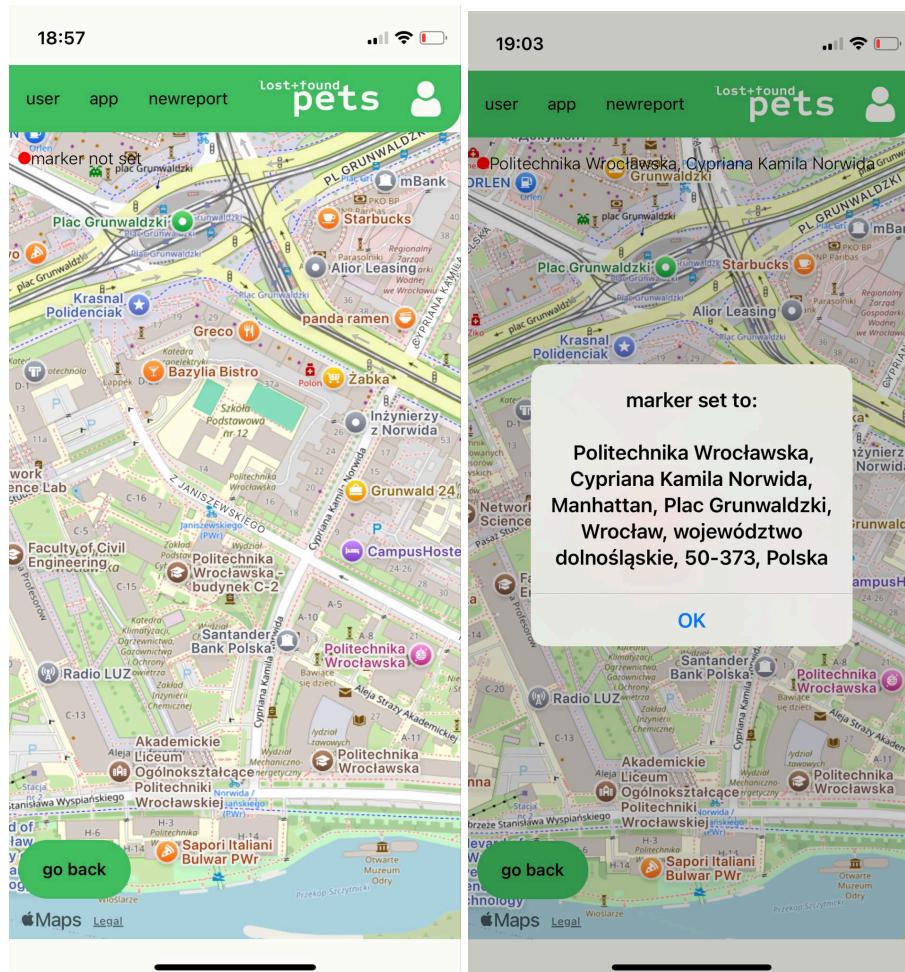
- lokalizacji
- zdjęcia
- typu (kot / pies)
- kolorów
- rozmiaru
- (opcjonalnie) dodatkowych informacji



Rysunek 44: Widok formularza nowego zgłoszenia

Kliknięcie w kafelek z mapą powoduje przejście użytkownika na widok mapy, gdzie może ustawić marker, odpowiadający lokalizacji znalezienia, lub zgubienia zwierzęcia.

Po wybraniu lokalizacji, użytkownik zostaje powiadomiony o miejscu które wybrał, w celu weryfikacji.

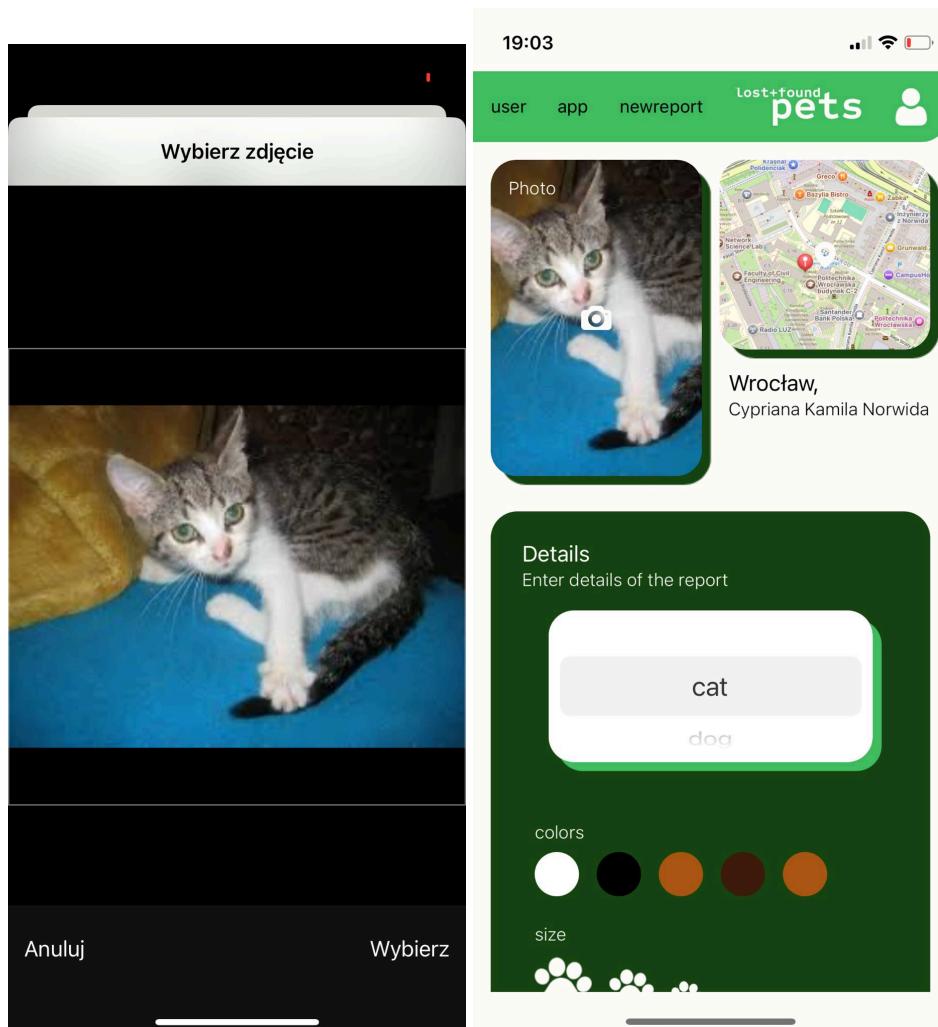


Rysunek 45: Widok mapy

Następnie, kliknięcie w kafelek ze zdjęciem powoduje przejście użytkownika na widok wyboru zdjęcia z galerii telefonu.

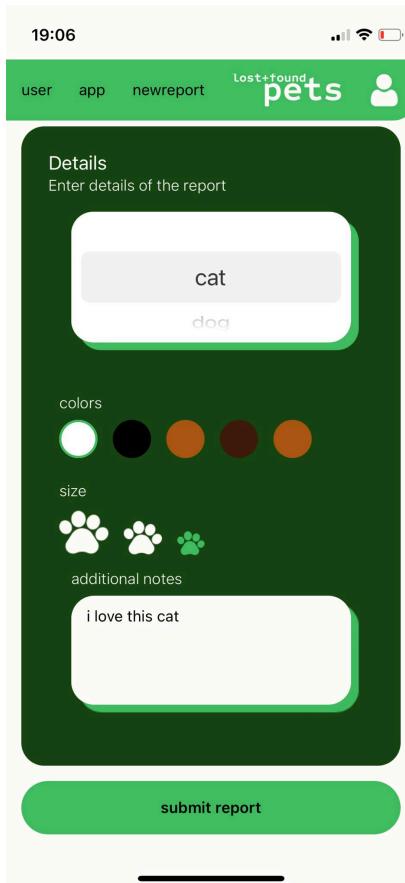
Po wyborze zarówno zdjęcia jak i lokalizacji, w ekranie formularza możemy zobaczyć nasze wybory. Na lewym kafelku widzimy wybrane zdjęcie, natomiast na prawym mapkę z wybraną lokalizacją, oraz poniżej tego kafelka, możemy zauważyc krótką informację na temat miejsca. **“Wrocław, Cypriana Kamila Norwida”**

Klikając ponownie na omówione wcześniej kafelki, użytkownik jest w stanie zmienić swoje wybory.



Rysunek 46: Widok wyboru zdjęcia z galerii Rysunek 47: Widok wyborów

Po wybraniu przez użytkownika wymaganych informacji, jest on uprawniony do przesłania raportu na nasz serwer.



Rysunek 48: Widok wypełnionego formularza

W tym momencie dane z formularza zostają przesłane na serwer poprzez interfejs API. Na serwerze zostaje stworzony nowy rekord w bazie danych, a także zapisane zostaje zdjęcie w celu późniejszej walidacji.

Zdjęcie poddawane jest walidacji po to, aby użytkownik nie był w stanie przesyłać innego zdjęcia niż zdjęcia zwierzaka.

Podczas działania serwera, użytkownik zostaje poinformowany o procesie ekranem ze znacznikiem ładowania.

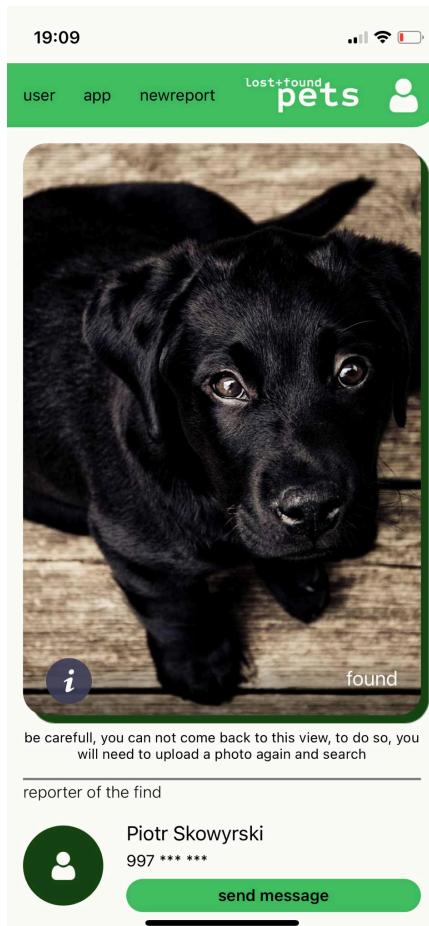


Rysunek 49: Widok znacznika ładowania podczas pracy serwera

Jeżeli zdjęcie **nie** przejdzie weryfikacji, użytkownik zostaje powiadomiony, rekord z bazy jest usuwany i cała operacja się kończy.

Jeżeli natomiast zdjęcie przejdzie weryfikację, zostaje podane dalej do modelu, który je przetwarza, następnie zwracając najbardziej zbliżone zwierzęta.

Po całym procesie, użytkownik dostaje X wyników (potencjalnych użytkowników, którzy zgłosili odnalezienie zwierzęcia oraz same zdjęcia)



Rysunek 50: Widok pojedynczego raportu po zwróceniu przez serwer

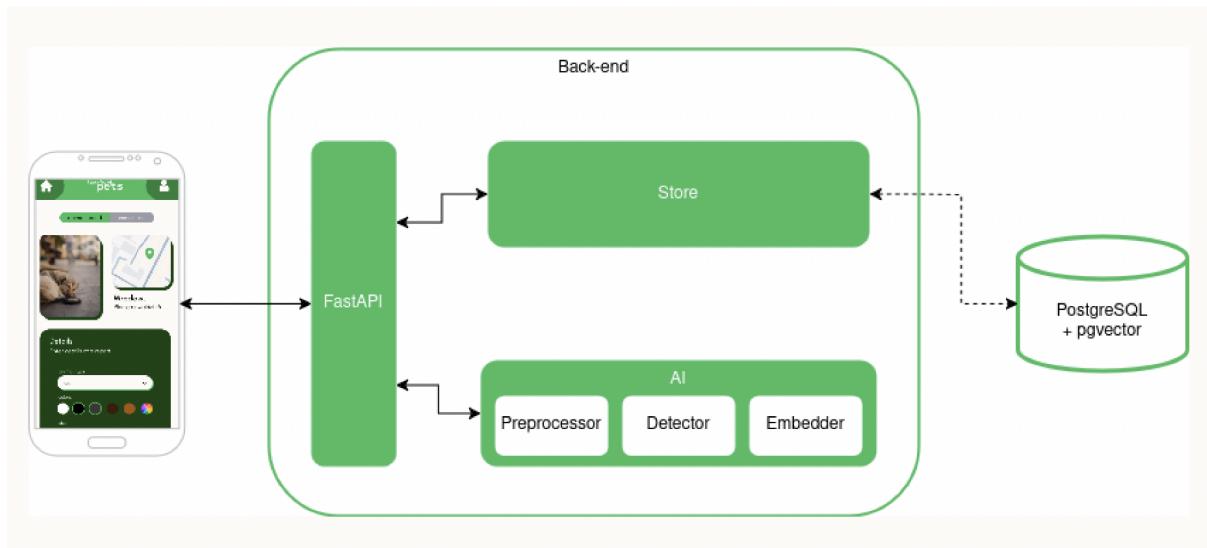
Użytkownik jest w stanie przejrzeć raporty, oraz skontaktować się z innym użytkownikiem, który potencjalnie znalazł zaginione zwierzę, poprzez wyświetlenie jego numeru telefonu.

***Wszystkie dane na poczet tej demonstracji zostały ustawione na stałe.
Nie jest to faktyczna praca modelu i zwrócenia przez niego wyników.***

21. Implementacja i integracja potoku przetwarzania sztucznej inteligencji

Wykonał: Bartosz Gruca, sprawdził: Jędrzej Radłowski, zatwierdził: Robert Pytel

Aby zrealizować kluczową funkcję aplikacji – wyszukiwanie zwierząt – zaimplementowano potok sztucznej inteligencji. Składa się on z trzech klas: Detector, Preprocessor oraz Embedder.

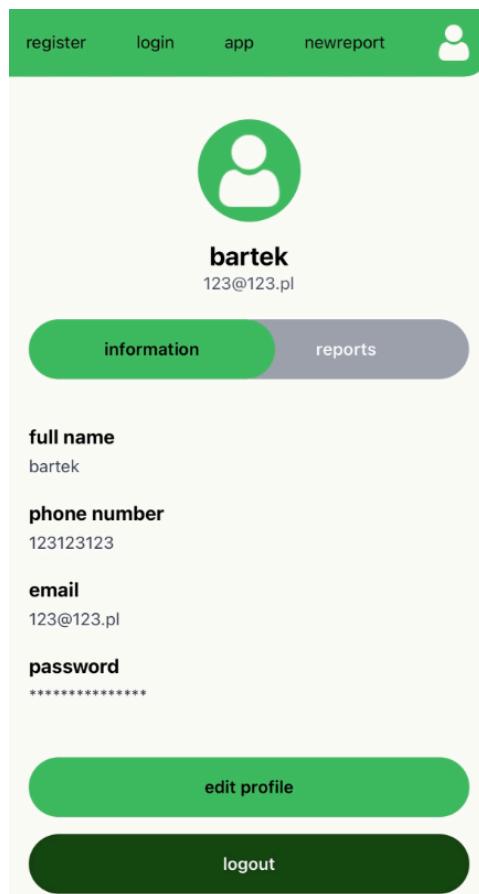


Rysunek 51: Schemat backendu uwzględniający moduł sztucznej inteligencji

22. Widok profilu użytkownika

Wykonał: Bartosz Gruca, sprawdził: Mateusz Oleksy, zatwierdził: Robert Pytel

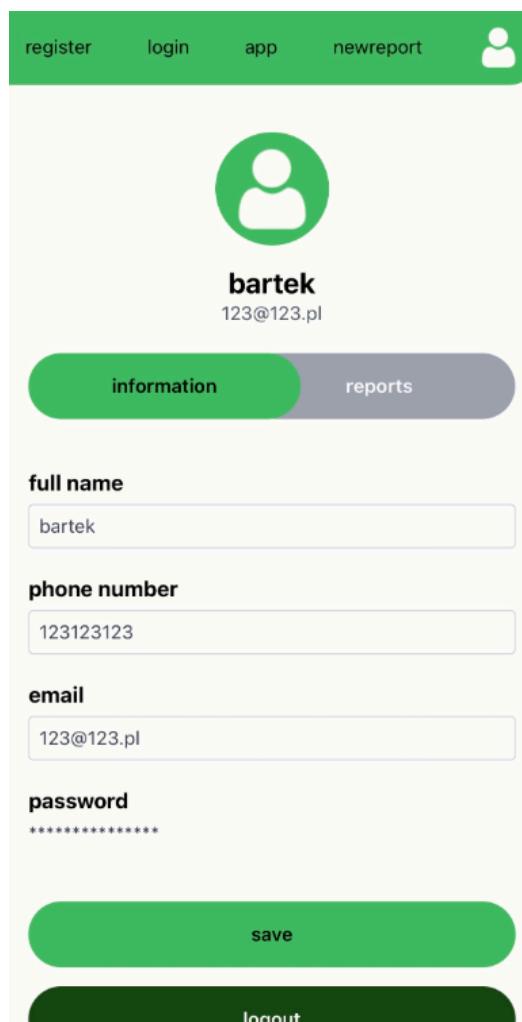
W aplikacji został stworzony dedykowany widok użytkownika, który pełni rolę osobistego panelu zarządzania kontem oraz zgłoszeniami. Celem tego widoku jest zapewnienie użytkownikowi kontroli nad jego danymi osobowymi oraz zgłoszeniami, jakie złożył za pośrednictwem systemu – zarówno tymi aktywnymi, jak i archiwalnymi. Widok ten jest dostępny po zalogowaniu się do aplikacji. Aby odwiedzić ten widok, użytkownik powinien kliknąć ikonę sylwetki w prawym górnym rogu aplikacji.



Rysunek 71: Widok użytkownika

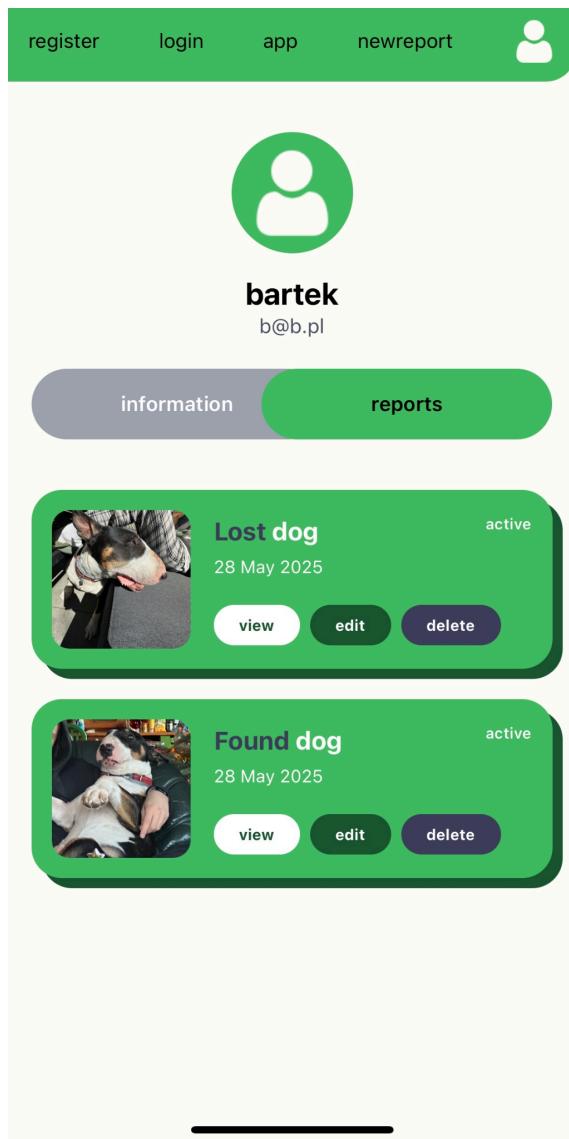
Użytkownik ma możliwość przeglądania i edytowania swoich danych. Może na przykład zaktualizować numer telefonu w przypadku jego zmiany, uzupełnić brakujące dane lub poprawić błędnie wprowadzone informacje. Dzięki temu system kontaktu między użytkownikami a osobami, które znajdą zgłoszone zwierzę, jest zawsze aktualny.

Po kliknięciu przycisku edit profile w widoku użytkownika, interfejs przechodzi w tryb edycji, co pozwala użytkownikowi na modyfikację swoich danych kontaktowych. Chwilowo nie można zmienić hasła, przewidziany jest do tego inny mechanizm.



Rysunek 72: Widok użytkownika po kliknięciu przycisku edycji danych

Z poziomu tego widoku użytkownik ma również możliwość wylogowania się. Proces ten polega na usunięciu tokenu sesji z pamięci aplikacji oraz przekierowaniu użytkownika na ekran logowania. Dostęp do pozostałych funkcji systemu możliwy jest dopiero po ponownym uwierzytelnieniu.



Rysunek 73: Widok użytkownika po przejściu do sekcji reports

Oprócz możliwości zarządzania swoimi danymi osobowymi, użytkownik aplikacji ma także dostęp do sekcji "Reports", która pełni rolę centrum zarządzania zgłoszeniami – zarówno tymi aktywnymi, jak i archiwalnymi (nieaktywnymi).

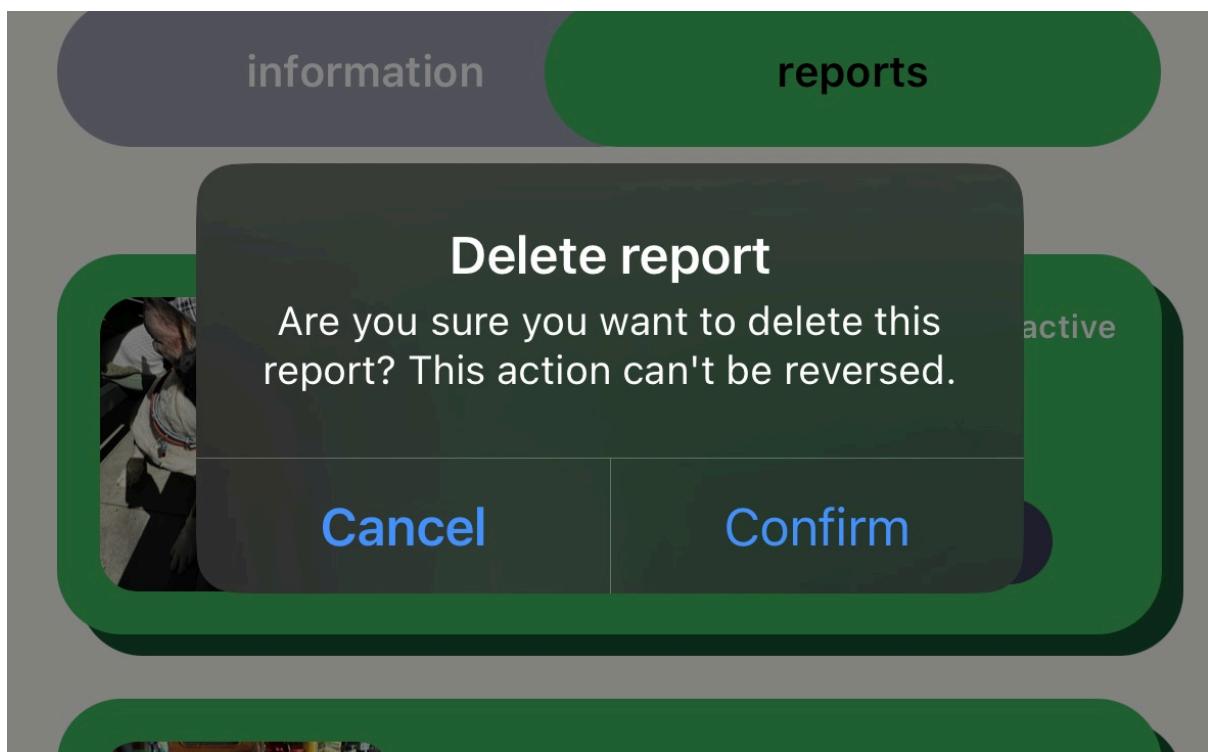


Rysunek 74: Nieaktywny report

Po przejściu do zakładki "Reports" użytkownik widzi listę swoich zgłoszeń przedstawioną w formie osobnych kart. Każda karta zawiera kluczowe informacje, które umożliwiają zorientowanie się w szczegółach danego zgłoszenia. Pierwszym elementem wizualnym jest zdjęcie zgłoszonego zwierzęcia, które pozwala natychmiast zidentyfikować zgłoszenie. Następnie widoczny jest tytuł, taki jak "Lost Dog" lub "Found Dog", informujący, czy zgłoszenie dotyczy zaginięcia, czy znalezienia zwierzęcia. Karta zawiera również datę zgłoszenia, co umożliwia użytkownikowi śledzenie chronologii wydarzeń. Dodatkowo, każdemu zgłoszeniu przypisany jest status - "active" (aktywny) lub "inactive" (nieaktywny).

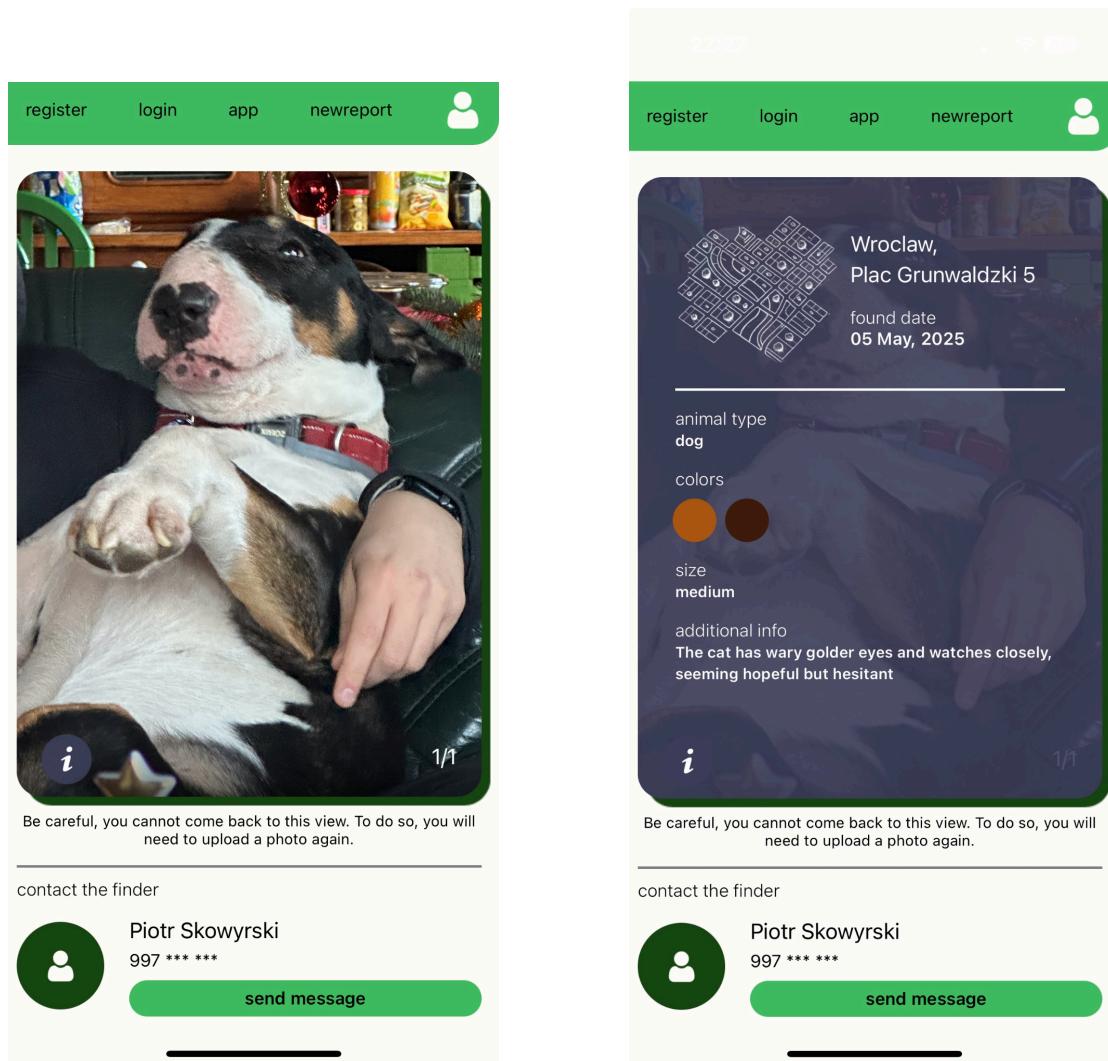
Klikając przycisk „edit” znajdujący się na karcie, użytkownik będzie miał możliwość edycji swojego zgłoszenia. Ta funkcjonalność, która zostanie wprowadzona w najbliższym punkcie kontrolnym, będzie szczególnie przydatna w sytuacjach, gdy użytkownik dysponuje innym, lepszym zdjęciem pupila i chce je podmienić, aby zwiększyć szansę na jego odnalezienie. Może również dodać nowe informacje, na przykład zaktualizować opis zachowania zwierzęcia, uwzględnić jego reakcję na ludzi lub wskazać nową lokalizację, w której zostało ostatnio zauważone. Ponadto edycja umożliwia poprawę ewentualnych błędów w danych, takich jak błędnie wskazane miejsce zdarzenia.

Jeżeli zwierzę zostanie odnalezione, użytkownik może usunąć zgłoszenie za pomocą przycisku "delete". Celem tej funkcji jest uporządkowanie historii oraz zminimalizowanie dezinformacji - nieaktualne zgłoszenia mogą wprowadzać w błąd, dlatego aplikacja daje możliwość ich szybkiego usunięcia z historii konta.



Rysunek 75: Komunikat przy próbie usunięcia zgłoszenia

Kolejnym dostępnym przyciskiem jest “view”, który umożliwia przejście do pełnego podglądu szczegółów zgłoszenia. Jest to ważna funkcjonalność, ponieważ pozwala użytkownikowi zobaczyć, jak jego zgłoszenie będzie widoczne dla innych użytkowników aplikacji.



Rysunek 76, 77: Widok po wykonaniu akcji view

Sprawdzenie:

Sprawdzanie polega na manualnym zmianie adresu ip postawionego lokalnie co nie jest zbyt dobrym zachowaniem. Aplikacja powinna uruchamiać się automatycznie z serwerem za pomocą jednego kontenera dockerowego, lub aplikacja użytkownika powinna udostępniać plik .env do lepszego zachowania linków do API.

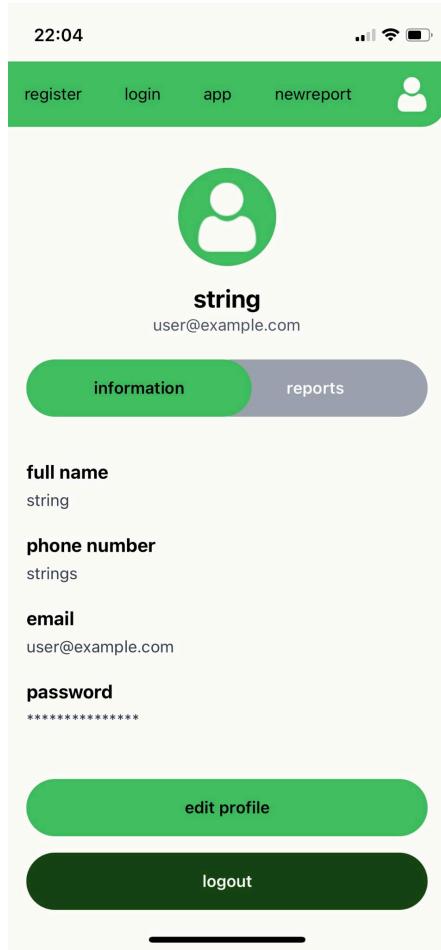
*Czasami również mogą wystąpić problemy z adresowaniem i przekazywaniem portów. Wówczas należy skorzystać z programu **ngrok**, uruchomić aplikację lokalnie i użyć komendy: **ngrok http 8000***

Najlepszą praktyką jest również odinstalowanie i zainstalowanie paczek od nowa, gdyż mogą wystąpić potencjalne problemy z zainstalowanymi dotychczas.

*Wersja systemu na moim telefonie - **IOS 18.5***

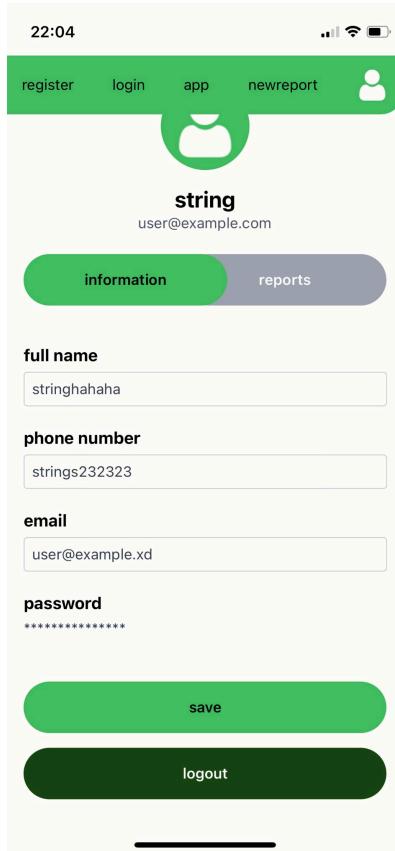
Widok informacji

Po zalogowaniu się do aplikacji i kliknięciu ikony użytkownika, wyświetla się widok panelu użytkownika. Widnieją tam informacje wprowadzone przeze mnie przy rejestracji - wszystko się zgadza.



Rysunek 78: Widok profilu użytkownika

Po kliknięciu przycisku “edit profile”, istnieje możliwość edycji danych. Jedynym mankamentem to brak możliwości edycji hasła, co może być przydatne dla potencjalnego użytkownika.



Rysunek 79: Widok edycji profilu użytkownika

Przy edycji problemem może być, iż klawiatura zasłania ostatnie pola z formularza. Po zmianie danych użytkownik zostaje poprawnie poinformowany o zaistniałej sytuacji za pomocą powiadomienia co daje świetne doświadczenie użytkownika.

Przycisk wylogowania działa poprawnie i nie pozwala użytkownikowi na ponowne wejście do aplikacji bez ponownej autentykacji.

Po zmianie adresu email, następne logowanie następuje za pomocą nowego maila, a więc wszystko się zgadza i zapisuje się na serwerze.

Widok reportów

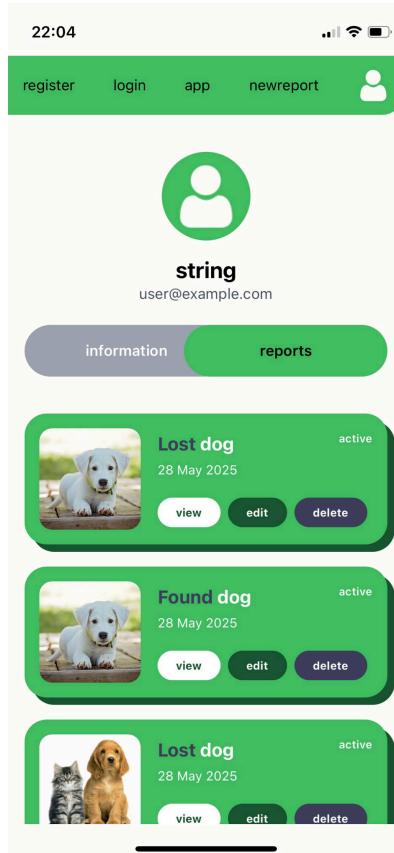
Po manualnym dodaniu nowego reporta, report widnieje w ekranie użytkownika

Na reporcie widzimy zmiany wcześniejszej opisane, są to:

- data stworzenia
- status
- zdjęcie
- typ zwierzęcia

oraz 3 przyciski:

- view
- edit
- delete

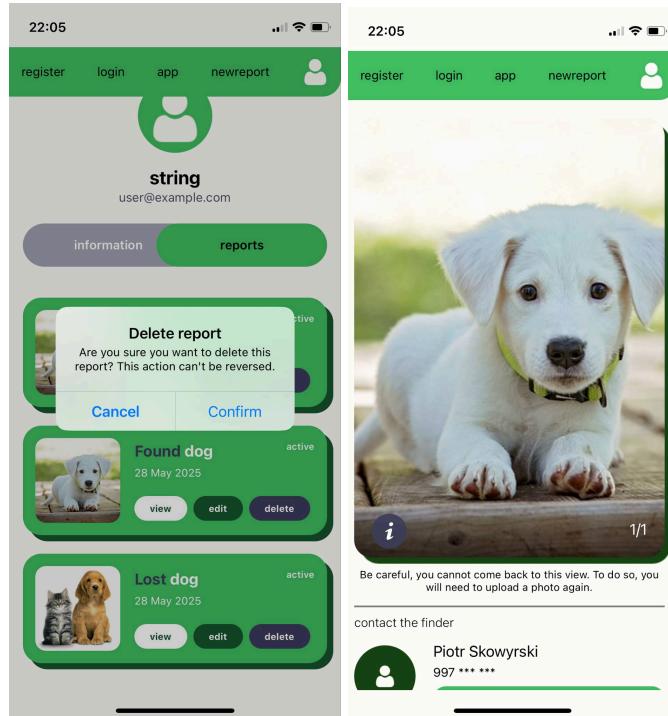


Rysunek 80: Widok historii zgłoszeń użytkownika

funkcjonalne są przyciski **view** oraz **delete**

- **view** powoduje wyświetlenie pełnego reporta
- **delete** powoduje usunięcia reporta z naszego profilu

Przed usunięciem dostajemy stosowny komunikat o konsekwencjach naszego działania - świetne doświadczenie użytkownika.



Rysunek 81, 82: Widok usuwania zgłoszenia oraz podglądu zgłoszenia

Podsumowując, opisane funkcjonalności działają, natomiast występuje parę mankamentów w postaci niemożności edycji hasła, oraz zasłaniania pola wejściowego w formularzu przez klawiaturę.

23. Edycja informacji o użytkowniku i usuwanie zgłoszeń

Wykonał: Robert Pytel, sprawdził: Jędrzej Radłowski, zatwierdził: Bartosz Gruca

Użytkownik aplikacji ma mieć możliwość modyfikacji swoich danych osobowych a mianowicie: nazwy użytkownika, numeru telefonu oraz adresu e-mail. W związku z tym widok profilu użytkownika ukazany na rysunku numer 66 oraz 67 został dopracowany w celu utrwalenia wprowadzonych przez użytkownika zmian na serwerze.

24. Dokumentacja programisty

Wykonał: Jędrzej Radłowski, sprawdził: Bartosz Gruca, zatwierdził: Robert Pytel

24.1 Tworzenie wzorców cech na podstawie zdjęcia

W celu utworzenia wzorca cech użyto biblioteki *PyTorch* napisanej w języku *Python*, która często wykorzystywana jest w przypadku zadań związanych z uczeniem maszynowym.

```
37  def get_embedding(image_path):
38      image = Image.open(image_path).convert('RGB')
39      inputs = processor(images=image, return_tensors="pt")
40
41      with torch.no_grad():
42          backbone_output = model.base_model(**inputs)
43
44      if isinstance(backbone_output, torch.Tensor):
45          features = backbone_output
46      else:
47          features = backbone_output['last_hidden_state']
48
49      if len(features.shape) == 4:
50          pooled_features = F.adaptive_avg_pool2d(features, (1, 1)).squeeze()
51      else:
52          pooled_features = features.squeeze()
53
54      normalized_embedding = F.normalize(pooled_features, p=2, dim=0)
55
56      return normalized_embedding.cpu().numpy()
```

Rysunek 24: Fragment kodu aplikacji służący do generowania wzorca cech na podstawie zdjęcia wejściowego

Na rysunku 24 przedstawiono fragment kodu aplikacji, który służy do generowania wzorca cech na podstawie zdjęcia wejściowego. Argumentem funkcji jest ścieżka do zdjęcia wejściowego. W linii 38 zdjęcie wejściowe konwertowane jest na format *RGB*, ponieważ model *ResNet 50* operuje na zdjęciu wejściowym zapisanym w tym formacie. Następnie w linii 39 obraz wejściowy dostosowywany jest do warstwy wejściowej modelu *ResNet 50* pod względem wymiarów wektora wejściowego oraz oczekiwów biblioteki *PyTorch*.

W liniach 41-42 następuje przetworzenie zdjęcia wejściowego przez sieć neuronową modelu, które z języka angielskiego określone jest jako *forward pass*. Przejście to charakteryzuje się tym, że zmieniane są tylko i wyłącznie wartości aktywacji poszczególnych neuronów sieci. Z kolei wartości wag oraz “biasów” nie są aktualizowane. W związku z tym, takie przetworzenie zdjęcia wejściowego nie jest

związane z “uczeniem sieci”. Dzięki *forward pass*, warstwa wyjściowa sieci neuronowej przyjmie wartości charakterystyczne dla tych, które zostały podane w warstwie wejściowej.

Po przejściu zdjęcia wejściowego przez strukturę sieci neuronowej następuje wyodrębnienie wektora cech. Instrukcje temu służące zawarte są w liniach 44-47. Ostatecznym krokiem jest normalizacja tensora, która następuje w linii 54. Operacja ta dzieli każdą składową wektora przez sumę kwadratów wszystkich składowych wektora. Finalnie w linii 56 wektor cech zwracany jest w postaci macierzy biblioteki *numpy* co umożliwia jego dalsze przetwarzanie w aplikacji.

24.2 Zapisywanie wzorca cech w bazie danych

Opisana wyżej funkcja *get_embedding* wykorzystywana jest w kolejnym kroku, w którym wektor zapisywany jest w bazie danych.

```
58 def save_image_embedding(image_path):
59     embedding = get_embedding(image_path)
60
61     conn = connect_db()
62     cursor = conn.cursor()
63
64     filename = os.path.basename(image_path)
65     embedding_list = embedding.tolist()
66     print(embedding_list)
67     insert_query = """
68         INSERT INTO photos (filename, embedding)
69         VALUES (%s, %s)
70     """
71     cursor.execute(insert_query, (filename, embedding_list))
72     conn.commit()
73
74     cursor.close()
75     conn.close()
```

Rysunek 25: Fragment kodu aplikacji służący do zapisu wzorca cech w bazie danych

Na rysunku 25 przedstawiono kod funkcji *save_image_embedding*, która służy do zapisania wektora cech zdjęcia wejściowego w bazie danych. Pierwszym krokiem jest uzyskanie wektora cech przy użyciu funkcji *get_embedding* zdefiniowanej na rysunku 24. Następnie nawiązywane jest połączenie z bazą danych, po którym następuje wykonanie kwerendy zdefiniowanej w linii 67, gdzie do tabeli *photos* dodawany jest nowy rekord składający się z nazwy zdjęcia oraz wzorca cech uzyskanego na podstawie tego zdjęcia.

24.3 Wyszukiwanie zdjęć przedstawiających podobne zwierzęta na podstawie wektorów cech

W momencie, gdy w bazie danych przechowywane są już wektory cech zdjęć przesłanych przez użytkowników aplikacji, możliwe jest wyszukiwanie fotografii przedstawiających zwierzęta o podobnych aspektach wizualnych na podstawie wcześniej utworzonych wektorów cech.

W tym celu wykorzystano wartość funkcji podobieństwa kosinusowego pomiędzy dwoma wektorami.

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Wzór na obliczenie podobieństwa kosinusowego pomiędzy dwoma wektorami

Biblioteka użyta do implementacji bazy danych o nazwie *pgvector* udostępnia operator, który oblicza wartość funkcji dystansu kosinusowego między dwoma wektorami cech. Wzór funkcji dystansu konsinusowego wygląda następująco:

$$\text{cosine distance} = 1 - \text{cosine similarity}$$

Wartość funkcji dystansu kosinusowego zawiera się w przedziale od 0 do 2. Wartość zera oznacza, że wektory cech są identyczne, natomiast wartość 2 określa, że wektory są swoim przeciwnieństwem.

```

119  def search_similar_images(image_path, top_k=10):
120      embedding = get_embedding(image_path)
121
122      # Convert numpy array to a comma-separated string for vector casting
123      embedding_str = ",".join([str(x) for x in embedding.tolist()])
124      embedding_str = "[" + embedding_str + "]"
125      conn = connect_db()
126      cursor = conn.cursor()
127
128      search_query = f"""
129          SELECT id, filename, embedding <=> (%s::vector) AS similarity
130          FROM photos
131          ORDER BY similarity ASC
132          LIMIT {top_k};
133      """
134
135      cursor.execute(search_query, (embedding_str,))
136      results = cursor.fetchall()
137      cursor.close()
138      conn.close()

```

Rysunek 26: Fragment kodu aplikacji służący do wyszukiwania wzorców cech zawartych w bazie danych

Na rysunku 26 przedstawiono fragment kodu aplikacji, który służy do wyszukiwania wzorców cech zawartych w bazie danych. Jako argument funkcji przyjmowana jest ścieżka do pliku, które jest zdjęciem wzorcowym oraz parametr *top_k*, który określa ilość zwracanych wyników.

W pierwszym kroku na podstawie zdjęcia wejściowego generowany jest wektor cech przy użyciu funkcji *get_embedding* zdefiniowanej na rysunku oznaczonym numerem 24. Następnie wektor ten konwertowany jest na postać napisową, aby móc wykorzystać go w kwerendzie języka SQL. Operator *<=>* użyty w zapytaniu języka SQL określa obliczenie wartości funkcji dystansu kosinusowego pomiędzy wejściowym wektorem cech a wektorami cech zgromadzonymi w bazie danych. Wyniki zapytania sortowane są rosnąco względem obliczonej wartości dystansu kosinusowego. Pierwsze *top_k* wyników zwracane jest w ramach wykonania kwerendy. W liniach 135-138 zapytanie do bazy danych jest wykonywane, a wyniki wykonania kwerendy zapisywane są w zmiennej *results*.

24.4 Ekran powitalny, formularze logowania i rejestracji w aplikacji użytkownika

Wykonał: Mateusz Oleksy, sprawdził: Bartosz Gruca, zatwierdził: Robert Pytel

Poniżej przykładowe implementacje kilku komponentów w języku javascript (React Native Expo). Stylingowanie odbywa się poprzez bibliotekę *tailwindcss* (<https://tailwindcss.com/>)

```
4  export default function Heading({ Show usages  ✎ xmavv *
5    type = "primary",
6    children,
7    className,
8  }: {
9    type?: "primary" | "secondary";
10   children: ReactNode;
11   className?: string;
12 }) : Element {
13   if (type === "primary")
14     return <Text className={`${text-xl ${className}`}>{children}</Text>;
15   else return <Text className={`${font-light ${className}`}>{children}</Text>;
16 }
17 }
```

Rysunek 30: Komponent Heading

```
4  interface ButtonProps extends TouchableOpacityProps { Show usages  ✎ xmavv *
5    type?: "primary" | "secondary";
6  }
7
8  const Button: React.FC<ButtonProps> = ({ Show usages  ✎ xmavv *
9    children,
10   type: "primary" | "secondary" = "primary",
11   className: string | undefined,
12   ...props: Omit<ButtonProps, ...>
13 }) : Element => {
14   if (type === "primary")
15     return (
16       <TouchableOpacity
17         className={`${relative bg-primary w-full rounded-full py-4 ${className}`}`}
18         {...props}
19       >
20         <Text className={`${text-center text-[19px] font-semibold`}>
21           {children}
22         </Text>
23       </TouchableOpacity>
24     );
25 }
```

Rysunek 31: Komponent Button

Do sprawdzenia poprawności wpisanych danych przez użytkownika, użyto biblioteki *Zod* (<https://zod.dev/>).

Biblioteka zod pozwala w bardzo łatwy sposób podać typ danych, który jest wymagany, minimalną oraz maksymalną długość, a także pola opcjonalne.

```
1 import { z } from "zod";
2
3 export const UserBaseSchema: ZodObject<{...}, "strip", ZodTypeAny, objectOutputType = z.object({ shape: {
4   display_name: z.string().min(minLength: 3).max(maxLength: 255),
5   email: z.string().email().max(maxLength: 255),
6   phone: z.string().min(minLength: 7).max(maxLength: 20),
7   created_at: z.string().datetime().optional(),
8   updated_at: z.string().datetime().optional(),
9   deleted_at: z.string().datetime().nullable().optional(),
10 });
11 }
```

Rysunek 34: Definicja pól przy użyciu biblioteki Zod

Po wcisnięciu przycisku, następuje walidacja a następnie dane przekazywane są na serwer, za pomocą protokołu HTTP, przy użyciu funkcji fetch, co widać na rysunku 34.

Jeżeli cały proces obejdzie się bez błędów, użytkownik wykona pożądaną akcję i zostanie przeniesiony do aplikacji jako zweryfikowany, co widnieje na rysunku 33.

```
23 if (!result.success) {
24   const errors: {email?: (string[] | undefined)... = result.error.flatten().fieldErrors;
25   const firstError: string = Object.values(errors).flat()[0];
26   Alert.alert(title: "Validation Error", message: firstError ?? "Something went wrong");
27   console.log("Validation Error", firstError ?? "Something went wrong");
28   return;
29 }
30
31 const payload: {email: string, password: stri... = {
32   email: result.data.email,
33   password: result.data.password,
34   phone: result.data.phone,
35   display_name: result.data.display_name,
36 };
37
38 const isAuthorized: boolean | undefined = await registerApi(payload);
39 if (isAuthorized) router.navigate( href: "/application");
40 };
41 }
```

Rysunek 35: Walidacja danych

```
9   try {
10     const response : Response = await fetch( input: "http://localhost:8000/api/v1/users/" , init: {
11       ↑
12       ↑
13       method: "POST",
14       headers: {
15         "Content-Type": "application/json",
16       },
17       body: JSON.stringify(payload),
18     });
19
20     if (!response.ok) {
21       const error = await response.json();
22       console.error("API Error", error.message ?? "Something went wrong");
23       Alert.alert( title: "API Error", message: error.message ?? "Something went wrong");
24
25       return false;
26     }
27
28     const data = await response.json();
29     console.log("User registered:", data);
30     Alert.alert( title: "Success", message: "You have successfully registered!");
```

Rysunek 36: Wysłanie żądania na serwer

24.5 Obsługa rejestracji i uwierzytelniania użytkowników po stronie aplikacji serwerowej

Wykonał: Jędrzej Radłowski, sprawdził: Mateusz Oleksy, zatwierdził: Robert Pytel

Oprócz kwestii walidacji przesyłanych danych biblioteka *FastAPI* (<https://fastapi.tiangolo.com/>) umożliwia integrację z narzędziem *alembic*. Jest to przykład systemu *ORM* (*Object Relational Mapper*), który służy do łączenia klas zdefiniowanych w kodzie programu oraz tabel znajdujących się w bazie danych. Podejście to umożliwia pobieranie, modyfikowanie oraz tworzenie rekordów w tabelach bez znajomości języka SQL.

```
8     # Shared user model
9     class UserBase(SQLModel):
10         display_name: str = Field(min_length=3, max_length=255)
11         email: EmailStr = Field(unique=True, index=True, max_length=255)
12         phone: str = Field(unique=True, index=True, min_length=7, max_length=20)
13         created_at: datetime = Field(default_factory=datetime.now)
14         updated_at: datetime = Field(default_factory=datetime.now)
15         deleted_at: datetime | None = Field(default=None)
```

Rysunek 37: Klasa bazowa *UserBase*

Na rysunku 37 zdefiniowano kod klasy *UserBase*, która reprezentuje wspólny (bazowy) model użytkownika w aplikacji serwerowej. Model ten określa strukturę danych użytkownika, które mogą być zapisywane w bazie danych i jednocześnie poddawane walidacji.

Pole *display_name* przechowuje nazwę wyświetlaną użytkownika, która musi mieć długość od 3 do 255 znaków. Email to adres e-mail użytkownika, który może mieć poprawną strukturę adresu e-mail, dzięki użyciu typu *EmailStr* z biblioteki *Pydantic*, a dodatkowo jest oznaczony jako unikalny co zapobiega duplikatom. Podobnie pole *phone* przechowuje numer telefonu użytkownika, który również musi być unikalny, a jego długość musi się mieścić w przedziale od 7 do 20 znaków.

Pole *created_at* zawiera znacznik czasu, który jest automatycznie ustawiany na aktualny moment przy tworzeniu rekordu – służy do śledzenia, kiedy dany użytkownik został utworzony. Analogicznie *updated_at* informuje o dacie i czasie ostatniej aktualizacji danych użytkownika. Ostatnie pole *deleted_at* może przyjmować wartość *None* lub datę.

```
18     # Database user model
19     class User(UserBase, table=True):
20         __tablename__: str = "users"
21         id: UUID = Field(default_factory=uuid4, primary_key=True)
22         password: str = Field(nullable=False)
```

Rysunek 38: Model użytkownika używany w bazie danych

Na rysunku 38 przedstawiono kod klasy *User* użytej w celu obsługi połączenia z

bazą danych. Klasa ta dziedziczy po wcześniej utworzonej klasie *UserBase*, co oznacza, że automatycznie zawiera wszystkie pola zdefiniowane w bazowym modelu.

Dodatkowym polem zdefiniowanym dla klasy *User* jest pole *id*, które pełni rolę głównego klucza. Jego typ to UUID, a wartość domyślnie generowana jest automatycznie przy pomocy funkcji `uuid4`. Dzięki temu każdy użytkownik w bazie danych ma unikalny identyfikator.

Drugim dodatkowym polem jest *password*, które przechowuje hasło użytkownika. Jest ono oznaczone jako wymagane (`nullable=False`), co oznacza, że nie można zapisać użytkownika bez podania hasła.

Oprócz tego klasa *User* ma specjalne pole `__tablename__`, które określa nazwę tabeli zdefiniowanej w bazie danych, z którą klasa ma być połączona w logice ORM.

```
25      # Public user model
26      class UserPublic(UserBase):
27          id: UUID
```

Rysunek 39: Klasa modelu *UserPublic*

Na rysunku 39 przedstawiono kod klasy *UserPublic*, która również dziedziczy po klasie *UserBase*. Ma ona zdefiniowane dodatkowo jedno pole *id* typu UUID. Klasa ta wykorzystywana jest jako struktura odpowiedzi w przypadku formularza rejestracji użytkownika.

```
35      # Register user model
36  ▼  class UserCreate(SQLModel):
37      email: EmailStr = Field(max_length=255)
38      password: str = Field(min_length=8, max_length=72)
39      phone: str = Field(unique=True, index=True, min_length=7, max_length=20)
40      display_name: str = Field(min_length=3, max_length=255)
```

Rysunek 40: Klasa modelu *UserCreate*

Na rysunku 40 zdefiniowano klasę modelu *UserCreate*, która wykorzystywana jest w formularzu rejestracji użytkownika. Jest to klasa bazowa, która ma zdefiniowane wszystkie pola wymagane w formularzu rejestracji użytkownika, a mianowicie pole *email*, *password*, *phone* oraz *display_name*.

```

46     @router.post("/", response_model=UserPublic)
47     async def create_user(user_create: UserCreate, session: SessionDep) -> UserPublic:
48         """
49             Register a new user.
50         """
51
52         user = User.model_validate(
53             user_create, update={"password": get_password_hash(user_create.password)})
54         )
55         user = await store.create_user(session=session, user=user)
56         if not user:
57             raise HTTPException(
58                 status_code=status.HTTP_400_BAD_REQUEST, detail="Invalid form data"
59             )
60         return user

```

Rysunek 41: Obsługa formularza rejestracji po stronie aplikacji serwerowej

Na rysunku 41 zdefiniowano instrukcje odpowiedzialne za obsługę formularza rejestracji użytkownika. Parametr wejściowy `user_create` zawiera dane przesłane przez klienta z poziomu aplikacji użytkownika. W liniach 52-54 następuje walidacja danych przesłanych przez użytkownika oraz szyfrowanie hasła, które zapobiega przechowywaniu hasła w pierwotnej postaci. Następnie w linii 55 następuje próba dodania użytkownika do bazy danych. Jeżeli akcja zakończy się niepowodzeniem w zostanie zwrócona informacja o błędzie w linii 57. W przeciwnym przypadku użytkownik otrzyma informacje o swoim identyfikatorze UUID oraz pozostałe informacje, które zdefiniował w formularzu.

```

13     @router.post("/login")
14     async def login(
15         session: SessionDep, form_data: Annotated[OAuth2PasswordRequestForm, Depends()]
16     ) -> Token:
17         """
18             OAuth2 compatible token login, get an access token for future requests.
19         """
20
21         user = await authenticate(
22             session=session, email=form_data.username, password=form_data.password
23         )
24         if not user:
25             raise HTTPException(
26                 status_code=status.HTTP_404_NOT_FOUND, detail="Incorrect email or password"
27             )
28         # elif not user.is_active:
29         #     raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail="Inactive user")
30         token = create_access_token(user.id)
31         return Token(token=token, token_type="Bearer")

```

Rysunek 42: Wysłanie żądania na serwer

Na rysunku 42 zdefiniowano instrukcje odpowiedzialne za uwierzytelnianie użytkowników aplikacji. Użytkownik w formularzu przesyła dane w postaci nazwy użytkownika oraz hasła. Jeżeli wprowadzone dane są niepoprawne to w linii 26 zostanie zwrócony błąd o tym informujący. W przeciwnym przypadku w linii 30 następuje stworzenie tokenu uwierzytelniającego JWT, który jest następnie zwracany użytkownikowi w linii 31. Token ten będzie służył aplikacji użytkownika do wykonywania zapytań do aplikacji serwerowej w przyszłości.

24.6 Detektor psów i kotów na zdjęciu wejściowym

Wykonał: Bartosz Gruca, sprawdził: Jędrzej Radłowski, zatwierdził: Robert Pytel

W implementacji użyliśmy wersji YOLO v11 wytrenowanej na zbiorze COCO (Common Objects in Context), który zawiera 330 tys. zdjęć (200 tys. z adnotacjami) należących do 80 klas obiektów, m.in. kotów i psów. YOLO v11m osiąga wyższą średnią precyzję (mAP) przy 22% mniejszej liczbie parametrów niż starszy YOLO v8m, dzięki czemu jest wydajniejszy obliczeniowo bez utraty dokładności.

```
class Detector:
    def __init__(self, detect_classes, min_conf = 0.5):
        self.model = YOLO("/content/yolo11n.pt")
        self.min_conf = min_conf
        if isinstance(detect_classes, str):
            self.detect_classes = [detect_classes]
        else:
            self.detect_classes = detect_classes

    def detect(self, image):
        results = self.model.predict(image)
        boxes = []
        for result in results:
            names = [result.names[cls.item()] for cls in result.boxes.cls.int()]
            for idx, name in enumerate(names):
                if result.boxes.conf[idx] > self.min_conf and name in self.detect_classes:
                    boxes.append(result.boxes.xyxy[idx].tolist())
        return boxes if len(boxes) > 0 else None
```

Rysunek 43: Implementacja klasy detektora wykorzystującego YOLO

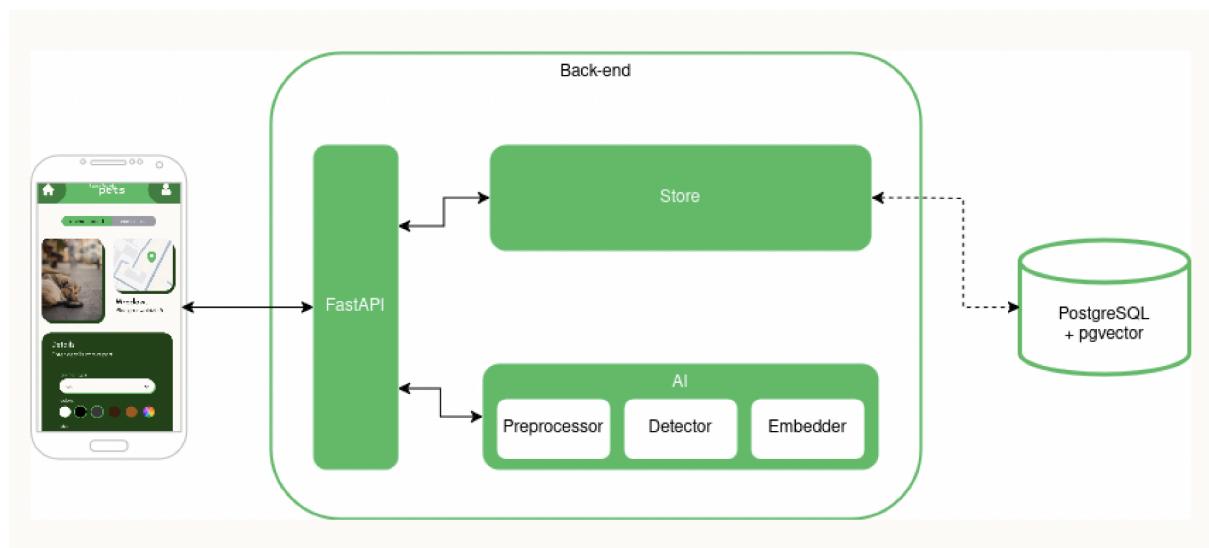
Po przetworzeniu fotografii detektor zwraca współrzędne obwiedni w przestrzeni 2D. Pozwala to wyciąć zwierzę i usunąć tło, które mogłoby negatywnie wpływać na wyszukiwanie podobieństw. Co ważne, YOLO zwraca obwiednie wyłącznie dla obiektów wskazanych w parametrach, skutecznie filtrując niepożądane fragmenty lub całe zdjęcia.

Klasa `Detector` implementuje tę logikę: podczas inicjalizacji ładowa wagi modelu, próg ufności oraz listę interesujących klas, a metoda `detect()` wykonuje predykcję, filtry wyniki według progu i nazwy klasy, po czym zwraca współrzędne każdej zaakceptowanej obwiedni ($[xmin, ymin, xmax, ymax]$) bądź `None`, gdy nic nie wykryto. Dzięki temu pojedyncze wywołanie klasy dostarcza gotowych danych do dalszej obróbki.

24.7 Implementacja i integracja potoku przetwarzania sztucznej inteligencji

Wykonał: Bartosz Gruca, **sprawdził:** Jędrzej Radłowski, **zatwierdził:** Robert Pytel

Aby zrealizować kluczową funkcję aplikacji – wyszukiwanie zwierząt – zaimplementowano potok sztucznej inteligencji. Składa się on z trzech klas: `Detector`, `Preprocessor` oraz `Embedder`.



Rysunek 51: Schemat backendu uwzględniający moduł sztucznej inteligencji

Pierwszym krokiem jest wykrycie zwierzęcia na zdjęciu. Wykorzystano do tego model YOLO (omówiony szczegółowo na stronie 56). Klasa `Detector` została zrefaktoryzowana tak, aby zwracała wyłącznie obiekt znajdujący się na pierwszym planie. Dzięki temu, nawet jeśli na fotografii znalazły się kilka psów, do dalszego przetwarzania trafia tylko jeden – ten, który został sklasyfikowany przez model jako najpewniejszy.

Następna jest faza preprocessingu:

- Standardowe kadrowanie – zwierzę jest wycinane i kadrowane tak by zajmowało większość fotografii z pozostawiając przy tym kawałek tła.
- Eksperimentalne usuwanie tła – do klasy Preprocessor dodano funkcję dokładnego odseparowania zwierzęcia od tła z wypełnieniem pustych obszarów kolorem czarnym (wykorzystano narzędzie rembg). Jednak – jak pokazują badania Junhui Lianga, Ying Liu i Vladimira Vlassova w artykule dostępnym pod adresem - <https://arxiv.org/abs/2308.09764>, tak dokładne oddzielenie obiektu od tła niekoniecznie poprawia skuteczność całego modelu, dlatego funkcja pozostaje domyślnie wyłączona i traktowana jako eksperimentalna.

Ostatnim krokiem jest proces stworzenia wektora cech. Najpierw AutoImageProcessor z biblioteki pytorch skaluje zdjęcie i normalizuje jego kanały RGB, zamieniając je w tensor. Następnie obraz przechodzi przez “kręgosłup” sieci ResNet-50 (bez warstwy klasyfikacyjnej), która zwraca mapy cech. Pooling (łączenie) kondensuje je do pojedynczego wektora 2048 liczb opisujących całe zwierzę. Na końcu ten wektor jest normalizowany względem normy L2, dzięki czemu można porównywać zdjęcia cosinusową miarą podobieństwa. Kod klasy Embedder można zobaczyć poniżej.

```
class Embedder:
    def __init__(self):
        self.processor = AutoImageProcessor.from_pretrained("amayel5/microsoft-resnet-50-batch32-lr0.0005-standford-dogs")
        self.model = AutoModelForImageClassification.from_pretrained("amayel5/microsoft-resnet-50-batch32-lr0.0005-standford-dogs")
        self.model.eval()

    def embed(self, image):
        inputs = self.processor(images=image, return_tensors="pt")

        with torch.no_grad():
            backbone_output = self.model.base_model(**inputs)

        if isinstance(backbone_output, torch.Tensor):
            features = backbone_output
        else:
            features = backbone_output['last_hidden_state']

        if len(features.shape) == 4:
            pooled_features = F.adaptive_avg_pool2d(features, (1, 1)).squeeze()
        else:
            pooled_features = features.squeeze()

        normalized_embedding = F.normalize(pooled_features, p=2, dim=0)

        return normalized_embedding.cpu().numpy()      Bartosz, przedwczoraj • feat: implement AI pipeline for image processin...
```

Rysunek 52: Kod klasy Embedder

Cały proces został zebrany do funkcji implementującej również obsługę wyjątków, które mogą wystąpić podczas działania. Poniżej znajduje się kod, wyjątku oznaczającego, że na zdjęciu nie zostało wykryte żadne zwierze.

```

try:
    box = self.detector.detect(image)
    if not box:
        raise NoObjectsDetectedError(
            "No relevant objects detected with sufficient confidence in the image."
        )
except Exception as e:
    if not isinstance(e, NoObjectsDetectedError):
        raise NoObjectsDetectedError(f"Detection failed: {str(e)}")
    raise

```

Rysunek 53: Przykładowy wyjątek, który może wystąpić po przesłaniu zdjęcia przez użytkownika

```

except NoObjectsDetectedError as e:
    raise HTTPException(
        status_code=status.HTTP_400_BAD_REQUEST,
        detail={
            "error_type": "NO_PET_DETECTED",
            "message": str(e),
            "suggestion": "We couldn't detect a pet in this photo. Please upload a clearer photo where the pet is the main subject and well-lit.",
        },
    )

```

Rysunek 54: Odpowiedź HTTP z sugestiami, którą odbiera frontend po nieudanym przetworzeniu zdjęcia (nie wykryto zwierzęcia) - do wykorzystania w alertach dla klienta

Dzięki przechwyceniu tych wyjątków w trakcie przetwarzania, użytkownik będzie otrzymywał odpowiednie wskazówki i sugestie co do zdjęcia, które dodaje w formularzu.

Oprócz informacji zwrotnych co do działania modelu, użytkownik będzie również otrzymywał inne informacje, jeśli przykładowo spróbuje dodać plik niewłaściwego formatu.

```

try:
    image_data = await photo.read()
    image = Image.open(io.BytesIO(image_data)).convert("RGB")
except Exception:
    raise HTTPException(
        status.HTTP_400_BAD_REQUEST,
        detail={
            "error_type": "INVALID_IMAGE_FILE",
            "message": "The uploaded file could not be processed as an image.",
            "suggestion": "Please ensure you are uploading a valid image format (e.g., JPEG, PNG).",
        },
    )
finally:
    await photo.close()

```

Rysunek 55: Kolejna odpowiedź HTTP z sugestiami, którą odbiera frontend jeśli przesłany przez klienta plik, jest w niepoprawnym formacie

Jeśli model “zaakceptuje” zdjęcie, zgłoszenie zostanie oznaczone jako aktywne; w przeciwnym razie użytkownik nie będzie musiał wypełniać ponownie formularza, dostanie za to możliwość dosłania samego zdjęcia. System oczekuje na aktywację raportu maksymalnie 5 minut — po ich upływie próba zostaje uznana za nieudaną, a zgłoszenie odrzucone. Proces ten opisany jest dokładniej na stronie 70.

Kiedy model zaakceptuje zdjęcie i utworzy poprawnie wektor cech tworzone jest zapytanie do bazy danych gdzie wykonuje się porównanie podobieństwa kosinusowego.

```

async def get_similar_photos(
    *, session: AsyncSession, embedding: list[float], limit: int = 5
) -> list[Photo]:
    """
    Find similar photos in the database using the provided embedding.
    Returns photos with related report and user information sorted by similarity to the provided embedding.
    """

    statement = (
        select(Photo)
        .where(Photo.embedding is not None)
        .order_by(literal_column(f"embedding <-> '{embedding}'"))
        .limit(limit)
    )
    result = await session.scalars(statement)
    return result.all()

```

Rysunek 56: Kod zapytania do bazy danych zwracający informacje o podobnych zwierzętach

Jako odpowiedź od bazy otrzymujemy ustaloną ilość zdjęć, wraz z innymi informacjami, które mogą być dla klienta istotne, takimi jak: dane kontaktowe znalazcy, przybliżona lokalizacja zwierzęcia w momencie odnalezienia itp. Informacje przesyłane są do aplikacji i wyświetlane w odpowiednim widoku.

24.8 Implementacja zgłoszenia w aplikacji serwerowej

Wykonał: Jędrzej Radłowski, sprawdził: Mateusz Oleksy, zatwierdził: Robert Pytel

Po stronie aplikacji serwerowej proces związany z obsługą zgłoszeń o zagubionych lub znalezionych zwierzętach rozpoczyna się w momencie, gdy serwer otrzymuje żądanie z danymi nowego zgłoszenia. Taki formularz trafia na serwer jako struktura danych w postaci JSON i posiada on wszystkie niezbędne informacje do stworzenia zgłoszenia. Gdy dane trafią do serwera następuje ich przetworzenie do postaci dopasowanej do modelu bazy danych. Konwersja ta polega na przypisaniu danych w odpowiedniej formie, np. rozdzieleniu lokalizacji na osobne pola "latitude" i "longitude"

```

25     @router.post("/", response_model=ReportPublic)  2 usages (2 dynamic)
26     async def create_report(
27         report_create: ReportCreate,
28         session: SessionDep,
29         current_user: CurrentUserDep,
30         background_tasks: BackgroundTasks,
31     ) -> Report:
32         """
33             Create a new report.
34         """
35
36         report = Report.model_validate(
37             report_create,
38             update={
39                 "user_id": current_user.id,
40                 "latitude": report_create.location.latitude,
41                 "longitude": report_create.location.longitude,
42                 "colors": map(Color.as_hex, report_create.colors),
43             },
44         )

```

Rysunek 57: Otrzymanie zgłoszenia na serwerze

Obiekt do którego trafiają te dane, jest modelem odwzorowującym strukturę tabeli w bazie danych. Zawiera on takie informacje jak identyfikator użytkownika, pozycję GPS, status informujący o stanie zgłoszenia, informacje o tym czy jest to przypadek gdy użytkownik odnalazł lub zgubił zwierzę oraz relację do powiązanych zdjęć. Definicja widoczna na Rysunku 57, który znajduje się poniżej.

```
41     class Report(ReportBase, table=True):
42         __tablename__ = "reports"
43         id: UUID = Field(default_factory=uuid4, primary_key=True)
44         user_id: UUID = Field(foreign_key="users.id")
45         latitude: float
46         longitude: float
47
48         photos: list[Photo] = Relationship()
```

Rysunek 58: Klasa Report

Po udanym zapisie system nie kończy jeszcze pracy, sprawdzane jest bowiem czy przypisane jest do zgłoszenia zdjęcie. Służy do tego pole “is_active”, a stworzone zostały uniknąć gromadzenia się niekompletnych zgłoszeń w bazie danych. W tym celu dodawane jest w tle zadanie, które będzie uruchomione dopiero po upływie określonego czasu, a w momencie gdy przekroczy zostanie limit czasu, zgłoszenie zostanie usunięte. Zostało to zaimplementowane w celu uniknięcia sytuacji gdy użytkownik będzie miał problem z przesaniem zdjęcia, lub gdy zdjęcie nie zostanie przez model odpowiednio przetworzone. W punkcie opisany wyżej przedstawiony jest potok przetwarzania przez sztuczną inteligencję i w momencie gdy użytkownik prześle zdjęcie niewyraźne bądź takie, które w ogóle nie będzie przedstawało zwierzęcia, to zdjęcie zostanie odrzucone. W takiej sytuacji użytkownik zostanie o tym poinformowany i będzie miał kilka minut na przesłanie odpowiedniego zdjęcia. Zgłoszenie które w ciągu pięciu minut zostanie uzupełnione o zdjęcie jest chronione przed usunięciem dzięki zmianie wartości “is_active” na False.

```
17     async def remove_invalid(*, session: AsyncSession, report_id: UUID): 1 usage
18         await asyncio.sleep(60 * 5)
19
20         report = await report_store.get_report_by_id(session=session, report_id=report_id)
21         if report and not report.is_active:
22             report_store.delete_report(session=session, report=report)
```

Rysunek 59: Zadanie czyszczące

Dodanie zdjęcia odbywa się poprzez wysłanie pliku do endpointu, który przyjmuje identyfikator zgłoszenia oraz plik zdjęcia. Po odebraniu pliku serwer sprawdza czy zgłoszenie o podanym identyfikatorze istnieje w bazie danych, a następnie tworzony jest nowy obiekt, który zawiera adres URL, oraz wektor cech reprezentujący zawartość zdjęcia.

```
55     @router.post("/{report_id}/photos", response_model=PhotoPublic)
56     async def upload_photo(report_id: UUID, photo: UploadFile, session: SessionDep):
57         """
58             Upload a new photo.
59         """
60
61         report = await report_store.get_report_by_id(session=session, report_id=report_id)
62         if not report:
63             raise HTTPException(
64                 status.HTTP_404_NOT_FOUND, detail=f"Report with id '{report_id}' not found"
65             )
66
67         photo = Photo(url="", embedding=[1.0], report_id=report_id)
68         session.add(photo)
69
70         report.is_active = False
71
72         report = await report_store.create_report(session=session, report=report)
73         if not report:
74             raise HTTPException(
75                 status_code=status.HTTP_400_BAD_REQUEST, detail="Invalid form data"
76             )
77     return photo
```

Rysunek 60: Otrzymanie zdjęcia przez serwer

Po poprawnym przesłaniu zdjęcia zgłoszenie mu odpowiadające zostaje zaktualizowane, tak aby nie zostało usunięte przez zadanie czyszczące.

24.9 Integracja wysyłania nowego raportu z serwerem

Wykonał: Mateusz Oleksy, sprawdził: Bartosz Gruca, zatwierdził: Robert Pytel

24.10 Przesyłanie raportu

Po wypełnieniu przez użytkownika wszystkich dostępnych pól w formularzu, możemy go przesłać na serwer a następnie do bazy danych.

Oczywiście na samym początku przesyłania dokonujemy walidacji pól w formularzu:

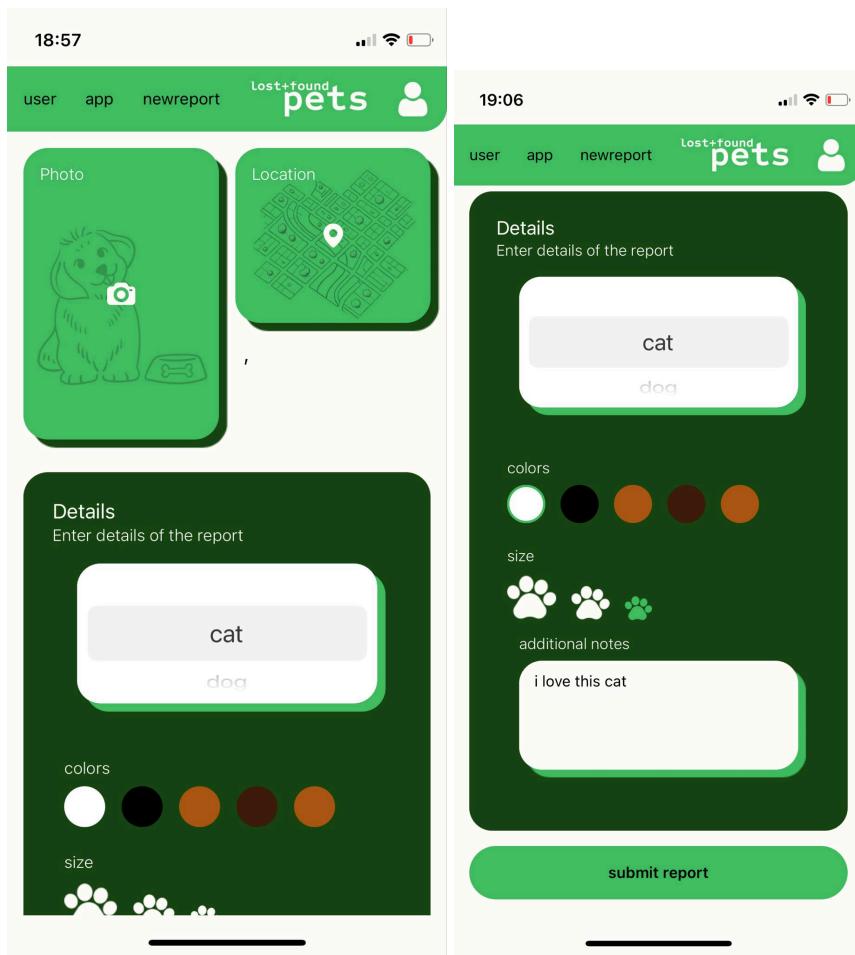
```
if (!selectedImage || !selectedImage.uri) {
    alert("Please choose a photo!");
    return;
}
if (!formData.animalType) {
    alert("Please select an animal type!");
    return;
}
if (formData.colors.length === 0) {
    alert("Please select at least one color!");
    return;
}
if (!formData.size) {
    alert("Please select a size!");
    return;
}
```

Rysunek 61: Walidacja formularza

Report przesyłany jest w dwóch częściach.

Na początku przesyłamy surowe dane, czyli dane pochodzące z komponentu *Map.tsx*, oraz dane z formularza:

- szerokość geograficzna
- długość geograficzna
- rodzaj zwierzęcia
- kolor
- rozmiar
- dodatkowe notatki



Rysunek 62: Widoki wysyłania formularza

```
const formDataToSend = {
  location: {
    latitude: -90,
    longitude: -180,
  },
  colors: formData.colors,
  species: formData.animalType.toUpperCase(),
  size: formData.size.toUpperCase(),
  note: formData.additionalNotes,
};
```

Rysunek 63: Przygotowanie informacji do wysłania

```

const tokenResponse = await apiRequest<any>({
  method: "POST",
  endpoint: "/reports/",
  payload: formDataToSend,
  contentType: "application/json",
  token: auth.token as string
});
console.log(tokenResponse);      robibobi, 6
reportId = tokenResponse["data"]["id"];
} catch (e: any) {
  console.error(e);
}

```

Rysunek 64: Wysłanie żądania “surowych” danych

Po wysłaniu tych danych otrzymujemy utworzony id utworzonego *reporta* w bazie danych. Report tworzony jest na samym początku bez zdjęcia. Gdy aplikacja użytkownika otrzyma id reporta, może wysłać zdjęcie do podanego reporta za pomocą kolejnego endpointa API - to druga część przesyłania reporta.

```

try {
  const photoFormData = new FormData();
  const uri = selectedImage.uri;
  const filename = uri.split("/").pop();
  const fileType = selectedImage.type;

  if (!filename) {
    console.error("Could not determine filename from URI:", uri);
    Alert.alert("Upload Error", "Could not upload.");
    return;
  }
  Po wysłaniu tych danych otrzymujemy utworzony id utworzonego reporta.
  Report tworzony jest na samym początku bez zdjęcia.
  Aplikacja użytkownika otrzyma id reporta, może wysłać zdjęcie do podanego reporta za pomocą kolejnego endpointa API - to druga część przesyłania reporta.
  const photoFormData.append("photo", selectedImage);
  name: filename,          Aby dodać zdjęcie do żądania.
  type: fileType,           Serwer otrzymując zdjęcie przetwarza je.
  } as any);                Jeżeli na zdjęciu nie zostanie wykryty pies, lub model napotka problem, serwer przesyła informację o błędzie do użytkownika.

  const uploadPhotoResponse = await apiRequest<report>({
    method: "POST",
    endpoint: `/reports/${reportId}/photos`,
    payload: photoFormData,
    token: auth.token as string,
  });

  console.log("uploadPhotoResponse", uploadPhotoResponse);
  matchedReports = uploadPhotoResponse.data ?? [];
}

```

Rysunek 65: Wysłanie żądania zdjęcia

Serwer otrzymując zdjęcie przetwarza je.

Jeżeli na zdjęciu nie zostanie wykryty pies, lub model napotka problem, serwer przesyła informację o błędzie do użytkownika.

Użytkownik po otrzymaniu komunikatu błędu o nie wykryciu psa na zdjęciu, ma 5 minut na dosłanie poprawnego zdjęcia, po tym czasie report zostaje usunięty z bazy.

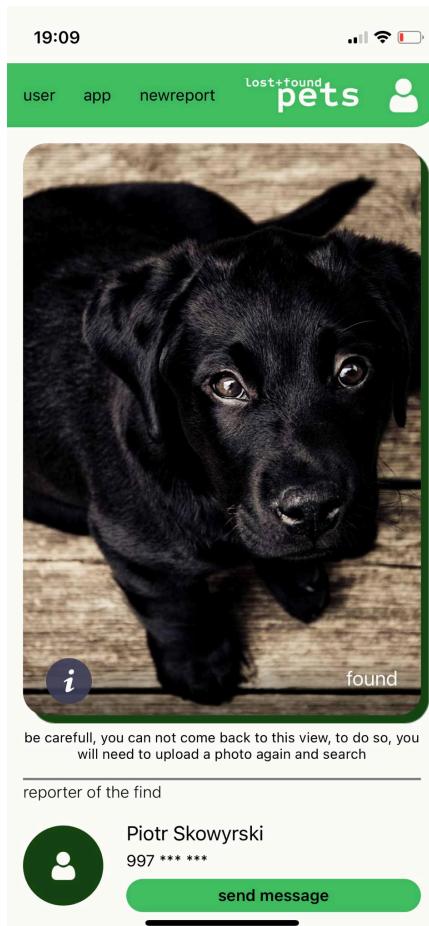
Jeżeli wszystko poszło sprawnie aplikacja otrzymuje *id reportów*, do których przypisane są najlepiej pasujące zdjęcia, generuje odpowiedni *URL* i przenosi użytkownika na widok reportów.

24.11 Wyświetlanie zdjęć pasujących

Po przekierowaniu na odpowiedni *URL* odczytujemy go i na tej podstawie pobieramy reporty pasujące i wyświetlamy je w widoku, tak aby użytkownik mógł porównać swoje zdjęcie do potencjalnie pasujących i skontaktować się ze znalazcą.

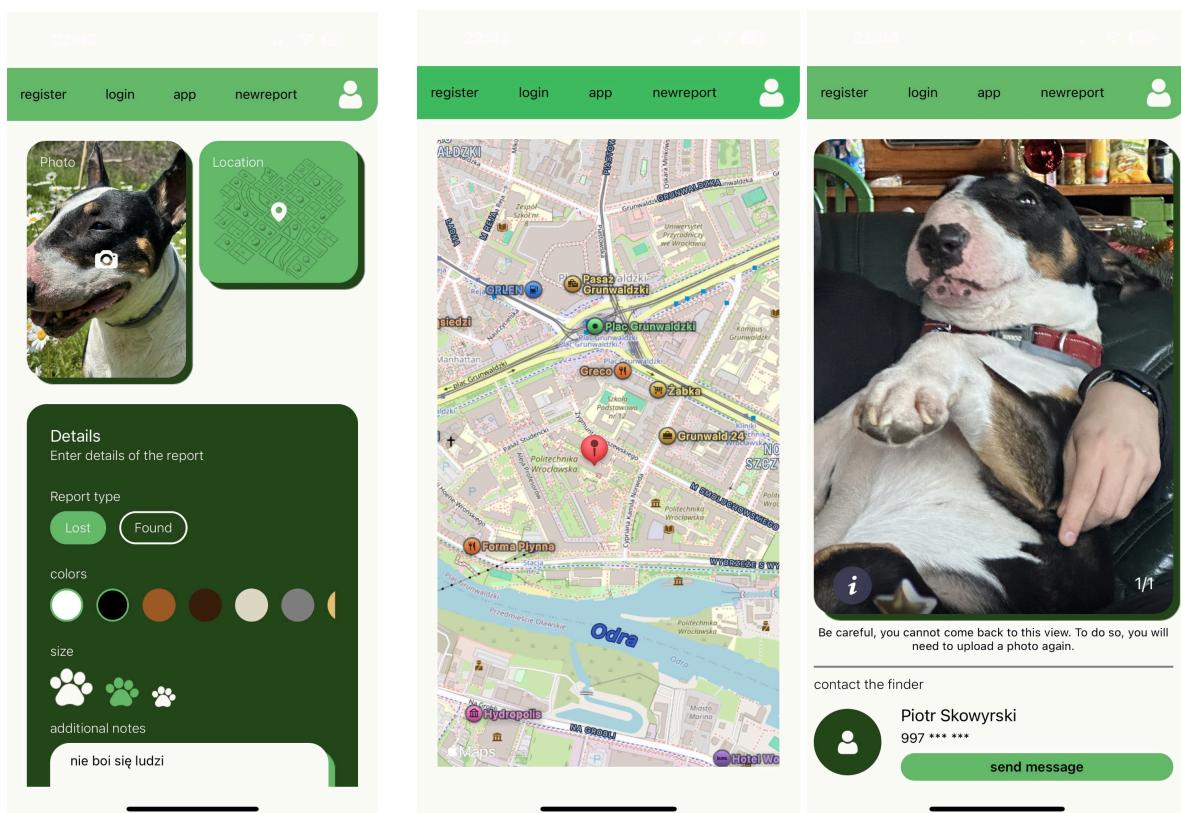
```
const response = await request<any>({
  method: "GET",
  endpoint: `/reports/${id}`,
});  
if (response.isSuccess && response.data) {  
  return response.data;  
}  
console.error("Failed to fetch report:", response.error);  
return null;  
  
//  
42 //  
43 //  
44 //  
45 //  
46 //  
47 //  
48 //  
49 //  
50 //  
51 //  
52 //  
53 //  
54 //  
55 //  
56 //  
57 //  
58 //  
59 //  
60 //  
61 //  
62 //  
63 //  
64 //  
65 //  
66 //  
67 //  
68 //  
69 //  
70 //
```

Rysunek 66: Odbieranie reporta z serwera



Rysunek 67: Widok wyświetcania raportów

Sprawdzenie: Po wypełnieniu formularza i dodaniu zdjęcia oraz lokalizacji, wcisnąłem przycisk submit report i w mniej niż sekundę otrzymałem odpowiedni zdjęcie psa. Ta część aplikacji działa zgodnie z opisem.



Rysunek 68, 69, 70: Kolejne kroki sprawdzenia funkcjonalności

25. Edycja informacji o użytkowniku i usuwanie zgłoszeń

Wykonał: Robert Pytel, sprawdził: Jędrzej Radłowski, zatwierdził: Bartosz Gruca

Użytkownik aplikacji ma mieć możliwość modyfikacji swoich danych osobowych a mianowicie: nazwy użytkownika, numeru telefonu oraz adresu e-mail. W związku z tym widok profilu użytkownika ukazany na rysunku numer 66 oraz 67 został dopracowany w celu utrwalenia wprowadzonych przez użytkownika zmian na serwerze.

Patrząc z perspektywy aplikacji serwerowej pierwszym krokiem w realizacji zadania aktualizacji profilu użytkownika było zdefiniowane ścieżki HTTP, która wykorzystuje metodę PUT. Zgodnie z dokumentacją *Mozilla Developer* metoda PUT służy do utworzenia nowego zasobu lub modyfikacji już istniejącego zasobu. W tym przypadku zostanie wykorzystany drugi scenariusz. Klasyczną metodą do tworzenia zasobów na serwerze jest metoda POST. Jednakże istnieje różnica pomiędzy metodą PUT a POST, a mianowicie metoda PUT cechuje się idempotentnością. Słowo to oznacza, że wynik operacji PUT z określonym zestawem parametrów

zawsze daje ten sam wynik. W przeciwnieństwie do metody POST, w której próba wykonania dwóch zapytań z tym samym zestawem parametrów może prowadzić do różnych rezultatów.

```
42  class UserUpdate(BaseModel):
43      display_name: Optional[str] = Field(None, min_length=3, max_length=255)
44      email: Optional[EmailStr] = Field(None, max_length=255)
45      phone: Optional[str] = Field(None, min_length=7, max_length=20)
```

Rysunek 83: Model *UserUpdate* służący do walidacji danych w formularzu aktualizacji profilu użytkownika

Na rysunku 83 przedstawiono model danych *UserUpdate* służący do walidacji danych w formularzu aktualizacji profilu użytkownika. Zdefiniowano trzy pola, które użytkownik może zmodyfikować: *display_name*, które odpowiada nazwie użytkownika, *email* będący adresem email oraz *phone*, który reprezentuje numer telefonu komórkowego. Pola te posiadają zdefiniowane ograniczenia co do ilości znaków. Oprócz tego pola te oznaczone są jako *Optional*, co umożliwia użytkownikowi modyfikację tylko wybranych informacji np. wyłącznie wyświetlanej nazwy użytkownika lub wyłącznie numeru telefonu komórkowego. Łączenie zmian jest również możliwe.

```

75 @router.put("/{user_id}", response_model=UserPublic)
76 async def update_user(
77     user_id: UUID,
78     user_update: UserUpdate,
79     current_user: CurrentUserDep,
80     session: SessionDep,
81 ) -> UserPublic:
82     """
83     Update a user's display name, email, or phone number.
84     """
85     user = await user_store.get_user_by_id(session=session, user_id=user_id)
86     if not user:
87         raise HTTPException(status.HTTP_404_NOT_FOUND, detail="User not found")
88
89     if current_user.id != user.id:
90         raise HTTPException(status.HTTP_403_FORBIDDEN, detail="Not allowed")
91
92     update_data = user_update.dict(exclude_unset=True)
93     if "email" in update_data:
94         existing_email_user = await user_store.get_user_by_email(session=session, email=update_data["email"])
95         if existing_email_user and existing_email_user.id != current_user.id:
96             raise HTTPException(
97                 status.HTTP_409_CONFLICT,
98                 detail=f"Email '{update_data['email']}' is already in use by another user.",
99             )
100    if "phone" in update_data:
101        existing_phone_user = await user_store.get_user_by_phone(session=session, phone=update_data["phone"])
102        if existing_phone_user and existing_phone_user.id != current_user.id:
103            raise HTTPException(
104                status.HTTP_409_CONFLICT,
105                detail=f"Phone '{update_data['phone']}' is already in use by another user.",
106            )
107
108    for key, value in update_data.items():
109        setattr(user, key, value)
110
111    user.updated_at = datetime.now(timezone.utc)
112    result = await user_store.create_user(session=session, user=user)
113    return result

```

Rysunek 84: Definicja ścieżki HTTP z metodą PUT do aktualizacji profilu użytkownika

Na rysunku 84 przedstawiono definicję ścieżki PUT służącej do aktualizacji profilu użytkownika. Użytkownik musi być zalogowany, aby edytować swoje dane. Logika działania powyższego kodu uwzględnia unikalność wartości pól *email* oraz *phone*. W momencie gdy użytkownik zmieni swój email lub numer telefonu na taki, który już istnieje w systemie, zostanie o tym poinformowany a jego zmiana nie zostanie zapisana. Dzięki temu system zabezpieczony jest przed duplikatami adresów email oraz numerów telefonu.

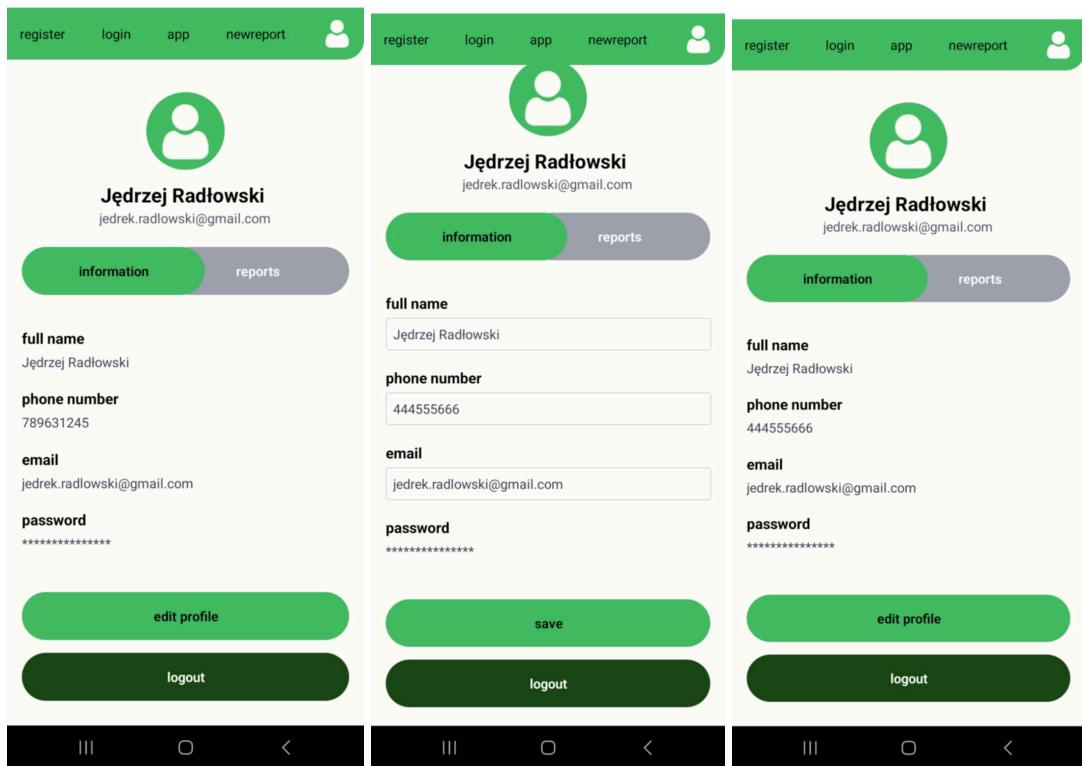
W przypadku usuwania zgłoszeń widok tej akcji w aplikacji użytkownika został uwzględniony na rysunku 75. W momencie potwierdzenia usunięcia zgłoszenia aplikacja mobilna wysyłanie żądanie do serwera za pomocą metody DELETE podając przy tym identyfikator zgłoszenia, które ma zostać usunięte.

```
205 @router.delete("/{report_id}")
206     async def delete_report(
207         report_id: UUID,
208         session: SessionDep,
209         current_user: CurrentUserDep,
210         response: Response
211     ):
212         """
213             Delete report by id.
214         """
215         report = await report_store.get_report_by_id_with_user_photo(
216             session=session, report_id=report_id
217         )
218         if report is None:
219             raise HTTPException(
220                 status_code=status.HTTP_404_NOT_FOUND, detail="Report not found."
221             )
222         if report.user.id != current_user.id:
223             raise HTTPException(
224                 status_code=status.HTTP_403_FORBIDDEN, detail="Forbidden."
225             )
226         result = await report_store.delete_report(session=session, report=report)
227         dir_url = f"app/static/photos/{report_id}"
228         if os.path.isdir(dir_url):
229             shutil.rmtree(dir_url)
230         if result is True:
231             response.status_code = status.HTTP_200_OK
232             message = "Report has been deleted."
233         else:
234             response.status_code = status.HTTP_500_INTERNAL_SERVER_ERROR
235             message = "Error when deleting report."
236     return message
```

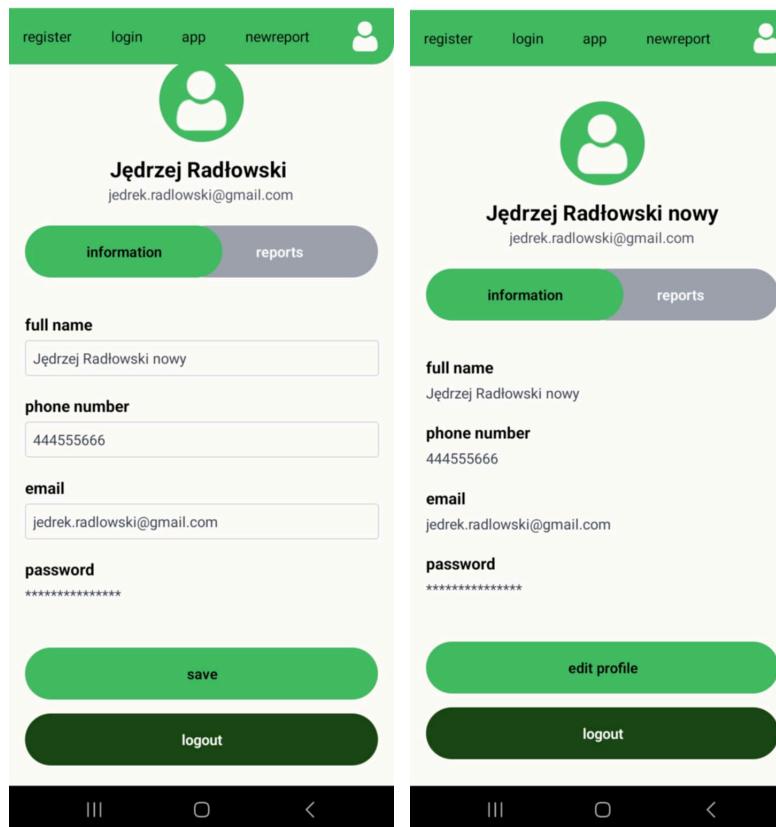
Rysunek 85: Definicja ścieżki HTTP z metodą DELETE do usuwania zgłoszeń użytkownika.

Na rysunku 85 przedstawiono fragment kodu aplikacji serwerowej odpowiedzialny za usuwanie zgłoszenia o identyfikatorze podanym jako parametr `report_id`. W pierwszym kroku sprawdzane jest czy użytkownik wykonujący zapytanie jest autorem zgłoszenia. Tylko w tym przypadku usunięcie zgłoszenia jest możliwe. Oprócz usunięcia zgłoszenia z bazy danych, następuje usunięcie zdjęcia przesłanego przez użytkownika z dysku twardego serwera.

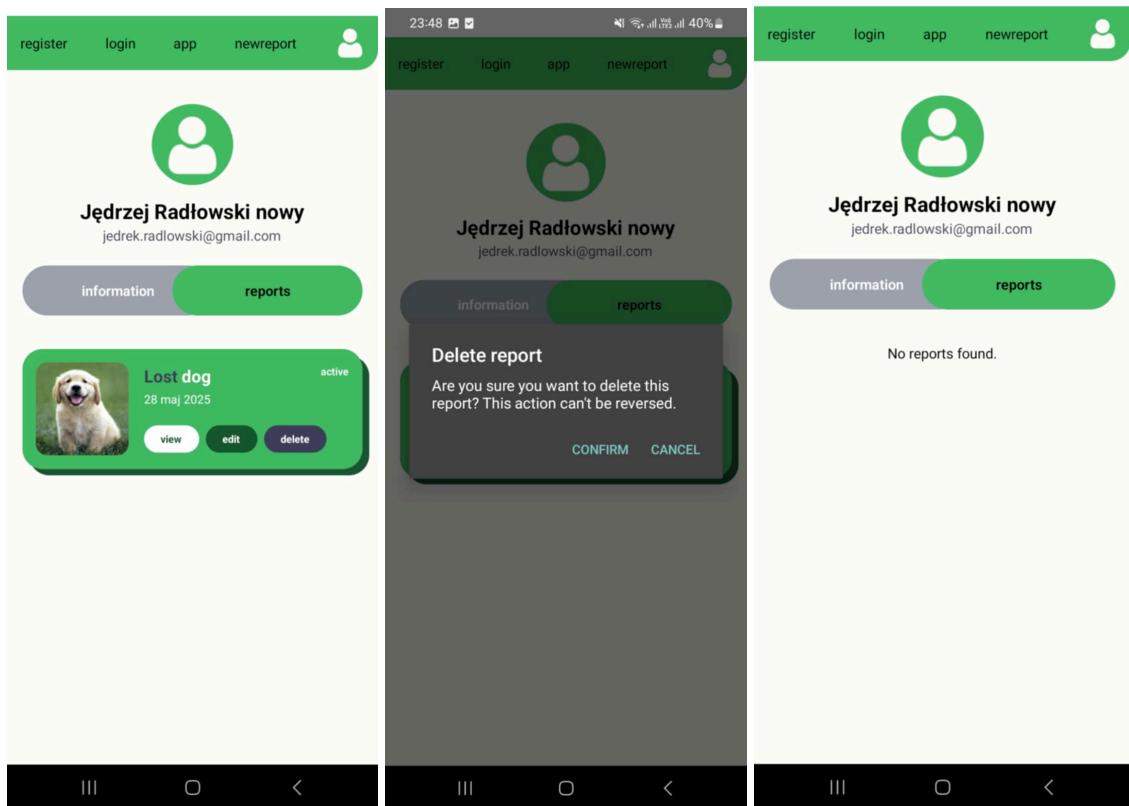
25.1 Sprawdzenie



Rysunek 86, 87, 88: Aktualizacja numeru telefonu użytkownika



Rysunek 89, 90, 91: Aktualizacja nazwy użytkownika



Rysunek 92, 93, 94: Usuwanie zgłoszenia użytkownika

Na rysunkach 86, 87, 88 przedstawiono przebieg zmiany numer telefonu użytkownika. Akcja przebiegła pomyślnie. Numer telefonu użytkownika został zaktualizowany.

Na rysunkach 89, 90, 91 przedstawiono przebieg zmiany nazwy użytkownika. Akcja przebiegła pomyślnie. Nazwa użytkownika została zaktualizowana.

Na rysunkach 92, 93, 94 przedstawiono przebieg usuwania zgłoszenia użytkownika. Akcja przebiegła pomyślnie. Zgłoszenie użytkownika zostało usunięte.

26. Hostowanie aplikacji i utworzenie pliku .apk na podstawie plików projektu. Instalacja aplikacji na urządzeniu mobilnym.

Wykonał: Jędrzej Radłowski, sprawdził: Robert Pytel, zatwierdził: Mateusz Oleksy

W celu uruchomienia aplikacji lokalnie z wykorzystaniem dockera oraz tunelowaniem dostępu zewnętrznego za pomocą ngrok wykonano następujące kroki.

Krok 1: docker-compose up –build -d

Za pomocą tej komendy zostają zbudowane obrazy Dockerowe oraz uruchamiane są w tle kontenery. Dzięki temu aplikacja jest gotowa do działania na porcie lokalnym.

Krok 2:

docker ps -a

docker logs -f id

Te dwie komendy używa się w celu sprawdzenia uruchomionych kontenerów oraz wypisania logów. Zostały one użyte aby upewnić się, że aplikacja wszystko przebiegło poprawnie.

Krok 3: ngrok http <http://localhost:8000>

Ngrok to narzędzie do tunelowania, które umożliwia wystawienie lokalnie działającej aplikacji na publiczny adres URL. Po uruchomieniu tego polecenia ngrok wygenerował publiczny adres URL: <https://f84a-185-72-184-52.ngrok-free.app>

Rysunek 95. Widok z FastAPI - Swagger UI

W celu przygotowania instalacyjnego pliku .apk na podstawie projektu frontendowego stworzonego w technologii React Native (z wykorzystaniem Expo), należało wykonać kilka kroków przygotowawczych oraz kompilacyjnych. Poniżej przedstawiono pełny proces:

Krok 1:

Pierwszym krokiem jest instalacja Android SDK. Jest to niezbędny zestaw programistyczny do budowania aplikacji na platformę Android.

Krok 2:

Po zainstalowaniu tej aplikacji trzeba było w folderze *frontend* projektu uruchomić następującą komendę: *npx expo prebuild*

Polecenie to generuje strukturę projektu natywnego co umożliwia budowanie aplikacji przy pomocy narzędzia takiego jak Gradle.

Krok 3:

Kolejnym krokiem było przejście do folderu *android* i uruchomienie programu *./gradlew assembleRelease*

Gradle zbudował aplikację w trybie “release”, co oznacza, że została przygotowana wersja zoptymalizowana do dystrybucji.

Krok 4:

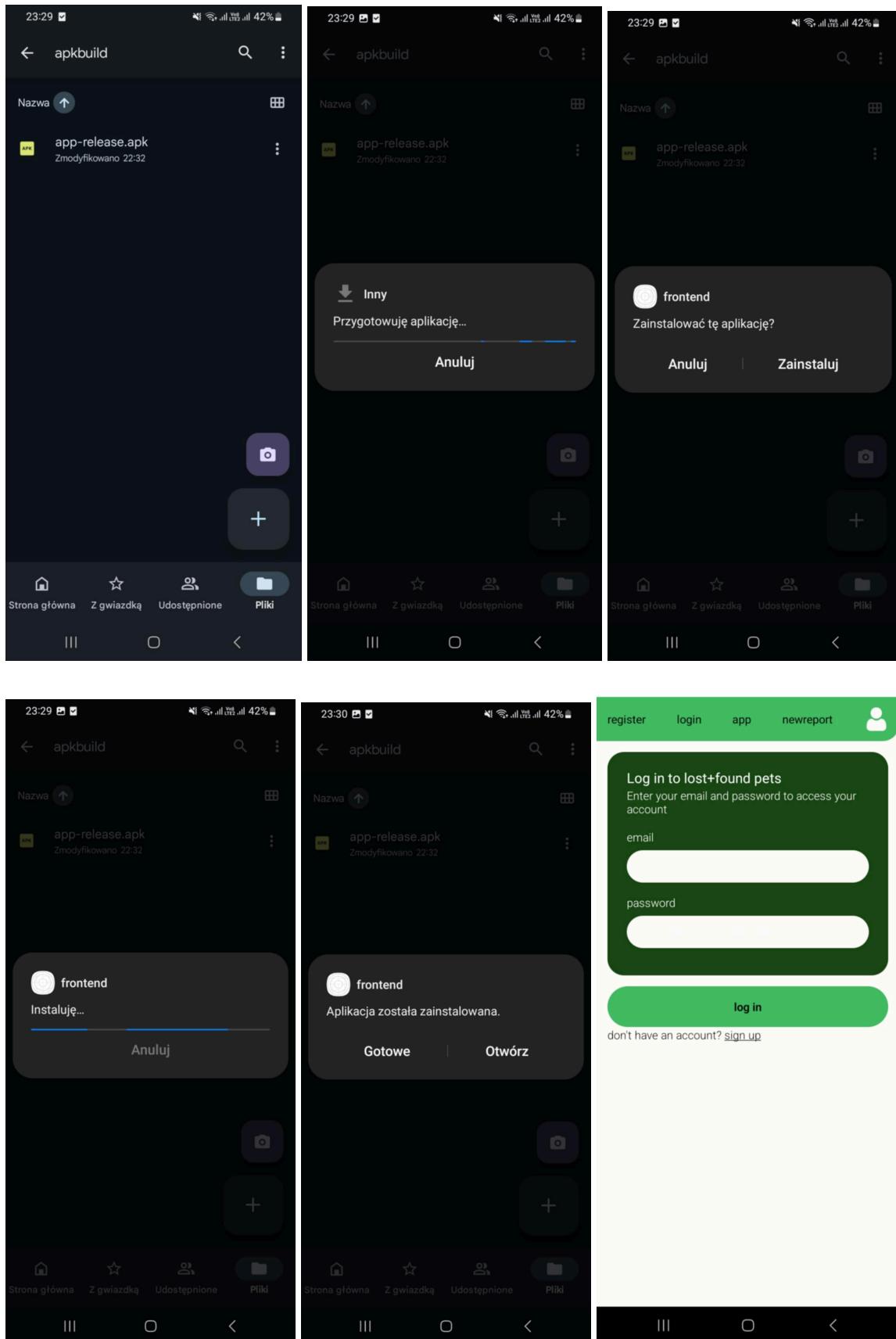
W wyniku uruchomienia programu w folderze *android\app\build\outputs\apk\release* znajdował się plik *.apk*

Ostatnim krokiem jest pobranie owego pliku na urządzenie.

Expo SDK version	Android version	compileSdkVersion	iOS version	Xcode version
53.0.0	7+	35	15.1+	16.0+

Rysunek 96. Informacja o wymaganej wersji systemu mobilnego.

26.1 Sprawdzenie



Rysunek 97, 98, 99, 100, 101, 102, 103: Przebieg instalacji aplikacji z pliku .apk.

Na rysunkach 97, 98, 99, 100, 101, 102 i 103 zaprezentowano przebieg instalacji aplikacji mobilnej z pliku .apk. Instalacja aplikacji została przetestowana na systemie Android 13. Przebiegła pomyślnie.

27. Instrukcja administratora

Wykonał: Bartosz Gruca, sprawdził: Mateusz Oleksy, zatwierdził: Robert Pytel

27.1 Uruchomienie na własnej infrastrukturze

27.2 Wymagania systemowe

Przed rozpoczęciem instalacji należy upewnić się, że system spełnia minimalne wymagania. Potrzebny jest Windows 10 lub Windows 11, włączony WSL (Windows Subsystem for Linux), minimum 4GB pamięci RAM oraz co najmniej 20 GB wolnego miejsca na dysku.

27.3 Przygotowanie systemu

Pierwszym krokiem jest włączenie WSL. Otwieramy PowerShell jako administrator i wykonujemy `wsl --install`. Następnie instalujemy dystrybucję Ubuntu wykonując `wsl --install -d Ubuntu`. Po instalacji restartujemy komputer.

Pobieramy Git dla Windows ze strony <https://git-scm.com/download/win> i instalujemy z domyślnymi ustawieniami. Po instalacji sprawdzamy wersję w Command Prompt lub PowerShell wykonując `git --version`.

Pobieramy Docker Desktop ze strony <https://docs.docker.com/desktop/install/windows/> i instalujemy aplikację.

Po instalacji uruchamiamy Docker Desktop i czekamy na pełne uruchomienie. Sprawdzamy instalację otwierając Command Prompt lub PowerShell i wykonując `docker --version` oraz `docker compose version`.

27.4 Pobieranie repozytorium projektu

Otwieramy Command Prompt lub PowerShell. Przechodzimy do wybranego katalogu i klonujemy repozytorium projektu używając `git clone https://github.com/mpieczaba/LostPlusFoundPets.git`. Po sklonowaniu przechodzimy do katalogu projektu przez `cd LostPlusFoundPets`. W folderze backend/app należy stworzyć pusty folder static.

27.5 Uruchomienie systemu

Należy upewnić się że znajdujemy się w folderze, który zawiera plik docker compose. Uruchamiamy kontenery w tle komendą `docker compose up -d`.

W przypadku potrzeby zatrzymania systemu wykonujemy `docker compose down`. Jeśli chcemy zatrzymać system z usunięciem wszystkich danych, używamy `docker compose down -v`.

27.6 Zarządzanie bazą danych PostgreSQL

Dostęp do bazy danych można uzyskać wchodząc do kontenera przez `docker compose exec postgres bash`, a następnie `psql -U postgres -d nazwa_bazy`.

Można także zainstalować pgAdmin (<https://www.pgadmin.org/>) jako graficzny interfejs do zarządzania bazą danych. Po instalacji łączymy się z bazą używając hosta `localhost` oraz portu `5432`, nazwy użytkownika i hasła zgodnie z plikiem `.env`.

```
POSTGRES_HOST=postgres
POSTGRES_PORT=5432
POSTGRES_DB=<nazwa_bazy_danych>
POSTGRES_USER=<nazwa_uzytkownika>
POSTGRES_PASSWORD=<haslo>

AUTH_SECRET="okoń"
```

27.7 Obsługa Swagger (dokumentacja API)

Aby korzystać ze dokumentacji API, należy otworzyć przeglądarkę i przejść do <http://localhost:8000/docs> na urządzeniu z którego został odpalony system (alternatywnie można skorzystać z opisanego w punkcie 26 sposobu wykorzystującego ngrok do wystawienia systemu na publiczny adres). Zobaczysz interaktywną dokumentację API z wszystkimi dostępnymi endpointami. Kliknij na endpoint, aby rozwinąć szczegóły, użyj przycisku "Try it out" do testowania API, wypełnij wymagane parametry i kliknij "Execute".

Swagger UI oferuje możliwość testowania każdego endpointu bezpośrednio z interfejsu, automatyczne generowanie przykładowych żądań oraz wyświetlanie odpowiedzi w czasie rzeczywistym. Jeśli API wymaga autoryzacji, należy użyć przycisku "Authorize" i wprowadzić token lub dane uwierzytelniające zgodnie z konfiguracją. Możliwy jest także eksport specyfikacji OpenAPI i import do narzędzi takich jak Postman czy Insomnia.

27.8 Monitorowanie i diagnostyka

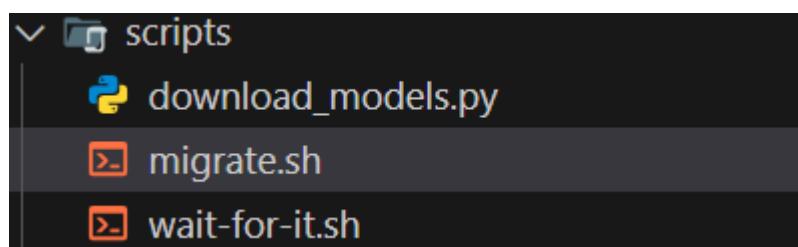
Do sprawdzania logów używamy `docker compose logs` dla wszystkich kontenerów, `docker compose logs backend` lub `docker compose logs postgres` dla konkretnego serwisu, oraz `docker compose logs -f backend` do śledzenia logów na żywo.

Sprawdzanie zasobów obejmuje `docker compose ps` dla statusu kontenerów, `docker stats` dla zużycia zasobów, oraz `docker compose top` dla informacji o kontenerach.

Sprawdzenie:

Opisany scenariusz jest scenariuszem idealnym, jednakże czasem mogą wystąpić problemy podczas instalacji, których nie uwzględniono w opisie.

27.5 Przy uruchamianiu kontenera na windowsie, czasem występuje błąd w kodowaniu plików. Należy wówczas zmienić kodowanie plików z CRLF, na LF w pokazanych plikach.



*27.7 Może pojawić się problem z adresacją, gdzie telefon nie będzie poprawnie odczytywał adresu IP, wówczas należy użyć **ngrok'a**, narzędzia pozwalającego przesyłać chwilowo lokalny serwis na serwis on-line.*

28. Role instalacji

Wykonał: Mateusz Oleksy, sprawdził: Jędrzej Radłowski, zatwierdził: Robert Pytel

W celu prawidłowego przeprowadzenia procesu instalacji aplikacji, poniżej przedstawiono role uczestniczące w procesie wraz z ich odpowiedzialnościami:

28.1 Administrator systemu (SysAdmin)

Zakres odpowiedzialności:

- Przygotowanie środowiska serwerowego (system operacyjny, wymagane zasoby sprzętowe i sieciowe).

Odnosi się to do sprawdzenia wersji systemu Windows: 11 lub 10.

Sprawdzenie czy aplikacja spełnia wymagania systemu na telefonie, oraz czy ma odpowiednie specyfikacje, aby zainstalować aplikację.

- Instalacja niezbędnych komponentów systemowych oraz zależności środowiska.

Jak opisano powyżej w instrukcji administratora, instalacja **Windows Subsystem Linux (WSL)** oraz wybranego distro (preferowane Ubuntu).

- Konfiguracja zapór sieciowych (firewall), DNS, SSL oraz dostępów użytkowników technicznych.

Dotyczy instalacji **ngroka**, jeżeli wymagany, przy potencjalnych problemach na systemie Windows. Ngrok pozwala udostępnić aplikację serwerową na tymczasowy serwer on-line. Jest to wymagane przy budowaniu aplikacji androidowej, tak aby miała ona dostęp do serwera, chodzi tutaj o **SSL**, szyfrowany przesył danych.

- Zapewnienie ciągłości działania środowiska (monitoring, aktualizacje systemowe).

28.2 Administrator bazy danych (DBA)

Zakres odpowiedzialności:

- Utworzenie instancji bazy danych zgodnie z wymaganiami aplikacji.
- Konfiguracja uprawnień, kont użytkowników oraz backupów.

Uruchomienie potencjalnego skryptu tworzącego kopię bazy danych i zapisującego kopię na dysku w celu uchronienia się od potencjalnych przerw u dostawców.

- Zapewnienie bezpieczeństwa danych i ich integralności.
- Wsparcie w przechowywaniu danych lokalnych.

Tworzenie woluminu występującego w kontenerze, tak aby dane mogły być przechowywane pomiędzy wersjami.

28.3 DevOps / Inżynier wdrożeń

Zakres odpowiedzialności:

- Automatyzacja procesu instalacji (np. poprzez skrypty, konteneryzację lub systemy zarządzania infrastrukturą).

Tworzenie kontenerów dockerowych, integrujących nasz system serweru API z systemem modelu AI.

- Wsparcie w deploymencie aplikacji na środowiskach testowych i produkcyjnych.

Konfiguracja Google Cloud, utworzenie serwera dostępnego z każdego urządzenia.

28.4 Programista / Zespół developerski

Zakres odpowiedzialności:

- Dostarczenie aplikacji w formie paczki instalacyjnej, obrazu kontenera lub kodu źródłowego.

Dotyczy developmentu aplikacji, implementacji kodu źródłowego, rozwijanie i utrzymywanie go za pomocą systemu kontroli wersji.

- Przygotowanie instrukcji instalacyjnej i dokumentacji technicznej.

Np. przygotowanie readme na repozytorium, które dotyczy instalacji jak i samej architektury oprogramowania i tego jak rozwiązano potencjalne problemy podczas implementacji.

- Zapewnienie wsparcia technicznego podczas pierwszej instalacji.

28.5 Konsultant wdrożeniowy / Kierownik projektu (PM)

Zakres odpowiedzialności:

- Koordynacja całego procesu instalacji i komunikacja między zespołami.

Przesył informacji pomiędzy teamem serwerowym, teamem modelu AI, oraz teamem aplikacji użytkownika

- Nadzór nad harmonogramem wdrożenia.

Dotyczy to między innymi wykresu Adamieckiego i przestrzegania jego terminów.

28.6 Tester / QA

Zakres odpowiedzialności:

- Weryfikacja poprawności działania aplikacji po instalacji.
Zainstalowanie pliku APK i testowanie funkcjonalności systemu.
- Zgłaszanie błędów i nieprawidłowości.
- Walidacja poprawności konfiguracji formularzy..

28.7 Użytkownik końcowy / Instalator

Zakres odpowiedzialności:

- Uruchomienie instalatora zgodnie z dokumentacją.
- Sprawdzenie poprawności instalacji.
- Kontakt z działem wsparcia w przypadku problemów.

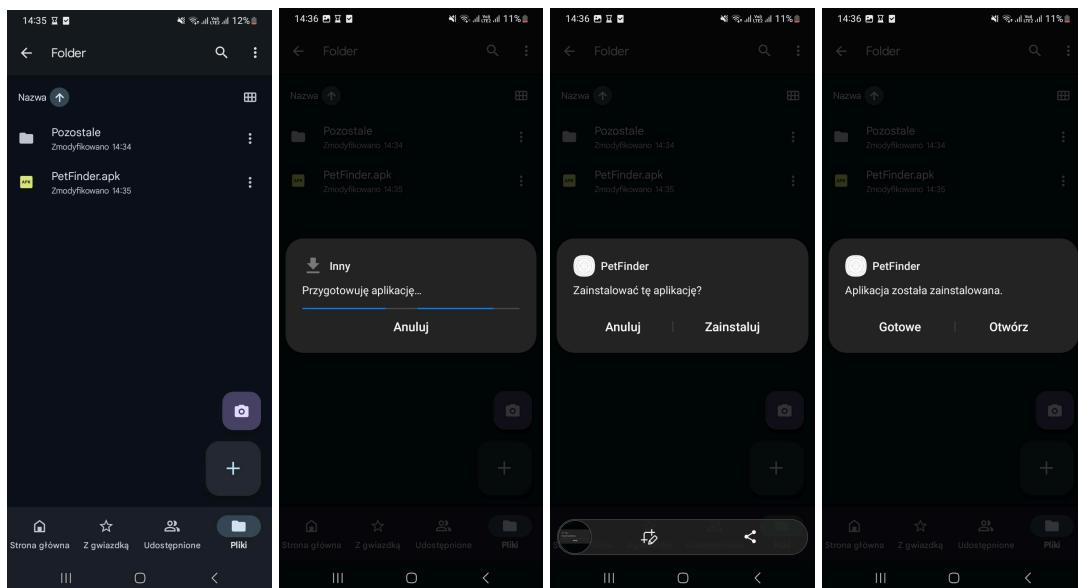
29. Instrukcja użytkownika

Wykonał: Robert Pytel, sprawdził: Bartosz Gruca, zatwierdził: Jędrzej Radłowski

29.1 Instalacja aplikacji mobilnej na system Android

W celu zainstalowania aplikacji mobilnej użytkownik musi być wyposażony w smartfon z systemem Android, którego minimalna wersja to 7.0. Smartfon wymaga połączenia z Internetem. Oprócz tego smartfon użytkownika musi mieć przynajmniej 125 megabajtów wolnego miejsca na dysku. Jeżeli wszystkie powyższe warunki są spełnione użytkownik może przystąpić do procedury instalacji.

Pierwszym krokiem jest pozyskanie pliku *PetFinder.apk*. W tym celu należy skontaktować się z autorami aplikacji. Po uzyskaniu dostępu do pliku *PetFinder.apk* należy umieścić go na dysku smartfona. Następnie należy kliknąć w ikonę pliku w celu rozpoczęcia instalacji.

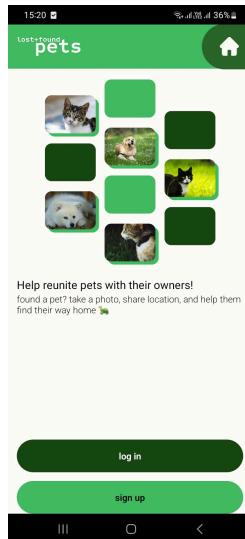


Rysunek 104, 105, 106, 107: Przebieg instalacji aplikacji z pliku .apk.

Na rysunkach 104, 105, 106, 107 przedstawiono przebieg instalacji aplikacji użytkownika na smartfonie z systemem Android. Po zainstalowaniu na pulpicie smartfona powinna pojawić się ikona aplikacji *PetFinder*. Kliknięcie ikony spowoduje uruchomienie aplikacji.

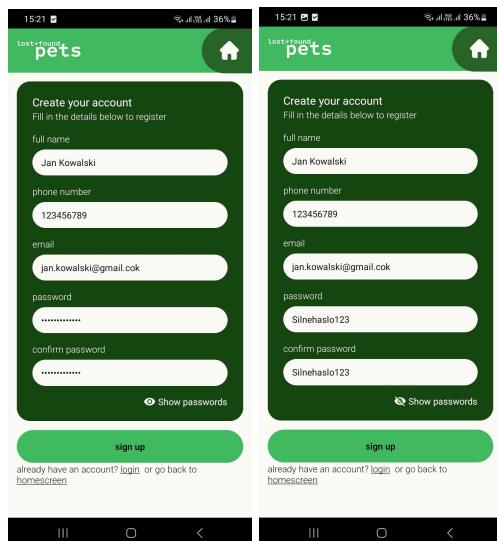
29.2 Rejestracja użytkownika w systemie

Po uruchomieniu aplikacji ukazuje się panel startowy aplikacji w momencie gdy użytkownik nie jest zalogowany.



Rysunek 108: Ekran powitalny aplikacji PetFinder

W celu przejścia do strony zawierającej formularz rejestracji należy nacisnąć jasno zielony przycisk z napisem **sign up**. Użytkownik zostanie przekierowany do widoku z formularzem rejestracji.



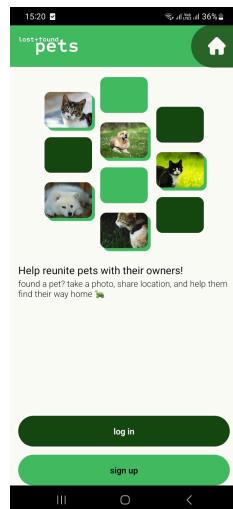
Rysunek 109, 110: Formularz rejestracji użytkownika

Na rysunkach 109, 110 przedstawiono formularz rejestracji użytkownika. Użytkownik wypełnia następujące pola: nazwa użytkownika, numer telefonu komórkowego, adres e-mail oraz hasło i potwierdzenie hasła. Wybrany adres e-mail oraz hasło będą następnie ważne w procesie logowania, ponieważ są to wymagane informacje w formularzu logowania. W prawym dolnym rogu formularza znajduje się ikona oka z napisem *Show passwords*. Jej kliknięcie powoduje ukazanie wpisanych haseł na ekranie. W celu zakończenia rejestracji użytkownik musi nacisnąć zielony przycisk

sign up. W przypadku pomyślnej rejestracji użytkownik zostanie przekierowany do formularzu logowania. Jeżeli formularz będzie posiadał błędy, użytkownik zostanie o tym poinformowany za pomocą komunikatu na ekranie.

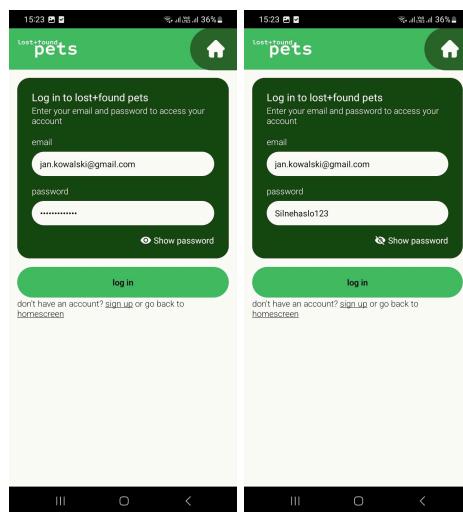
29.3 Logowanie użytkownika w systemie

Po uruchomieniu aplikacji ukazuje się panel startowy aplikacji w momencie gdy użytkownik nie jest zalogowany.



Rysunek 111: Ekran powitalny aplikacji PetFinder

W celu przejścia do widoku formularza logowania użytkownik musi nacisnąć na ciemnozielony przycisk **log in**.



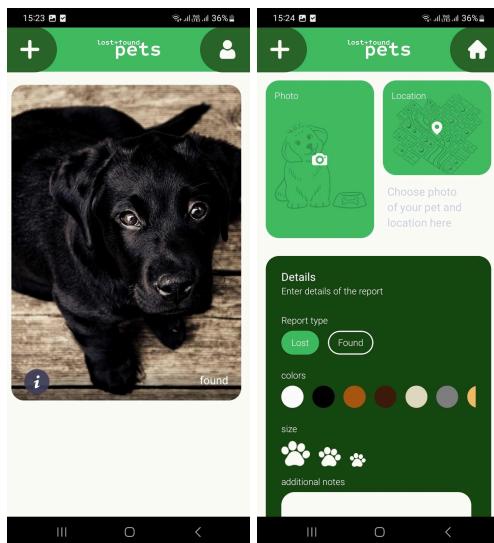
Rysunek 112, 113: Formularz logowania użytkownika

Na rysunkach 112, 113 przedstawiono formularz logowania użytkownika. Zawiera on dwa pola: e-mail użytkownika oraz hasło użytkownika. Podobnie jak formularz rejestracji w prawym dolnym rogu formularza znajduje się ikona do wyświetlenia zawartości pola do wpisywania hasła w celu weryfikacji poprawnego uzupełnienia. W

celu wysłania formularza należy zielony przycisk ***log in***. W przypadku wprowadzenia prawidłowych danych użytkownik zostanie przekierowany do widoku głównego aplikacji. W przypadku gdy logowanie nie powiedzie się, użytkownik zostanie poinformowany o tym za pomocą komunikatu.

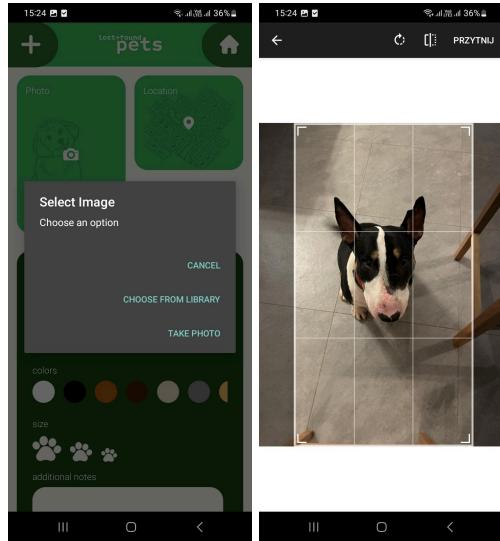
29.4 Dodawanie zgłoszenia

W celu dodania zgłoszenia użytkownik aplikacji musi posiadać konto oraz być zalogowany. W celu przejścia do widoku zawierającego formularz zgłoszenia należy nacisnąć ikonę plusa znajdująca się w lewym górnym rogu aplikacji.



Po lewej stronie rysunek nr 114: Widok główny aplikacji z uwzględnieniem ikony plusa w lewym górnym rogu ekranu. Po prawej stronie rysunek nr 115: formularz zgłoszenia.

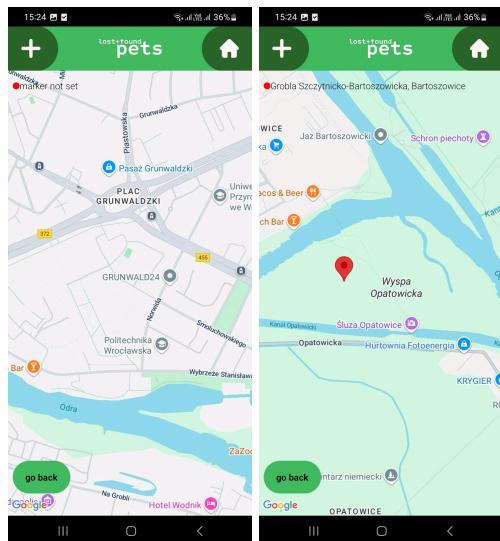
Na rysunku 114 przedstawiono formularz zgłoszenia zaginięcia czy też znalezienia psa. Formularz uwzględnia następujące informacje: zdjęcie zwierzaka, lokalizacja zaginięcia/znalezienia, typ zgłoszenia (zaginięcie/znalezienie), barwy sierści zwierzaka, przybliżony rozmiar zwierzaka oraz dodatkowe informacje o zwierzaku przydatne w celu jego identyfikacji.



Po lewej stronie rysunek nr 116: Wybór zdjęcia z możliwością wykonania zdjęcia lub wyboru zdjęcia znajdującego się w galerii. Po prawej stronie rysunek nr 117: kadrowanie wybranego zdjęcia.

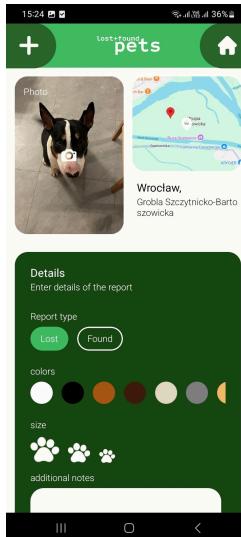
Na rysunkach 116, 117 przedstawiono widok aplikacji mobilnej w przypadku wyboru zdjęcia do formularza zgłoszenia. Aplikacja umożliwia wykonanie zdjęcia za pomocą wbudowanego w smartfon aparatu fotograficznego lub wybór zdjęcia znajdującego się w galerii smartfona. Po wybraniu lub wykonaniu zdjęcia aplikacja umożliwia wykadrowanie zdjęcia.

Kolejnym polem formularza jest lokalizacja zaginięcia czy też znalezienia zwierzęcia. W tym celu należy kliknąć na widok mapy. Po wykonaniu tej akcji użytkownikowi ukaże się podgląd map z systemu Google Maps.



Rysunek nr 118, 119: Wybór lokalizacji zaginięcia lub znalezienia zwierzaka.

W celu zaznaczenia miejsca zdarzenia należy nacisnąć palce na miejscem na mapie. Po wykonaniu tej akcji w miejscu kliknięcia pojawi się czerwona pinezka symbolizująca wybrane miejsce. Po wykonaniu tej akcji użytkownik może powrócić do wypełniania formularza za pomocą zielonego przycisku z napisem **go back**.

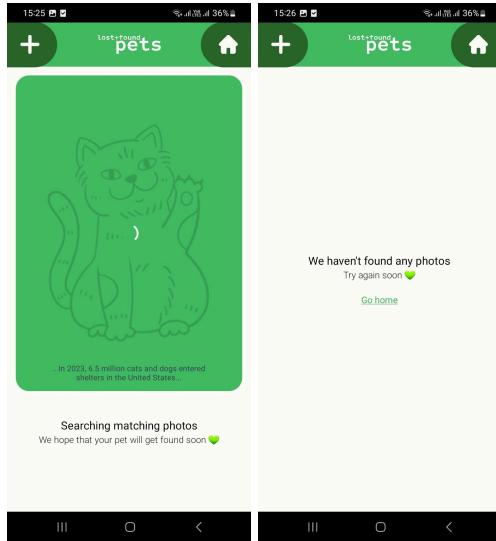


Rysunek nr 120: Widok formularza po wybraniu zdjęcia oraz lokalizacji.



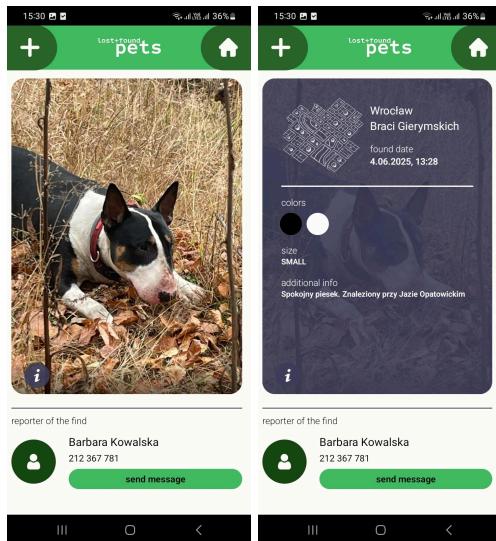
Rysunek nr 121: Widok formularza po wybraniu rodzaju zgłoszenia, barwy sierści, rozmiaru oraz dodatkowych cech zwierzaka.

W przypadku wyboru barwy sierści zwierzęcia możliwe jest wybranie kilku kolorów jednocześnie. W celu wysłania wypełnionego formularza należy nacisnąć na przycisk **submit report**.



Po lewej stronie rysunek nr 122: Widok przetwarzania zawartości formularza przez serwer. Po prawej stronie rysunek nr 123: odpowiedź serwera w przypadku gdy nie znaleziono podobnego zwierzęcia w bazie danych.

Na rysunku nr 122 przedstawiono widok, który ukazuje się użytkownikowi po wysłaniu formularza w oczekiwaniu odpowiedź ze strony serwera. Z kolei na rysunku numer 123 przedstawiono widok aplikacji po otrzymaniu odpowiedzi ze strony serwera, w momencie gdy nie znaleziono podobnego zwierzęcia w bazie danych do tego przedstawionego na zdjęciu w formularzu.



Rysunek nr 124, 125: Widok zgłoszeń, na których znajdują się zdjęcia podobnych zwierząt w porównaniu do zdjęcia przesłanego w formularzu.

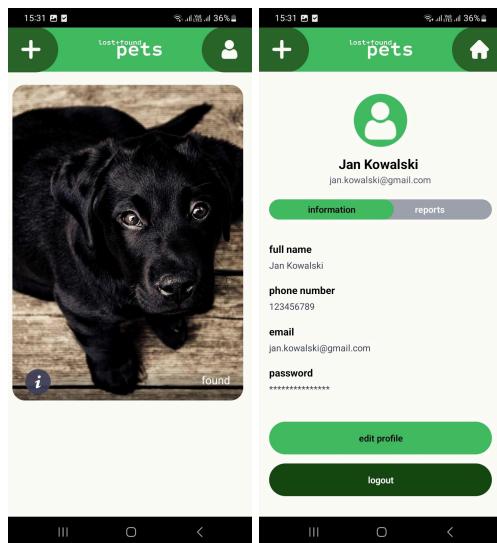
Na rysunku nr 124 przedstawiono widok odpowiedzi serwera na wysłanie formularza w momencie, gdy w bazie danych znajdowały się zwierzęta podobne do tego przedstawionego na fotografii zawartej w formularzu. Maksymalna liczba zdjęć

dopasowanych osobników, która może znaleźć się w odpowiedzi ze strony serwera to 10 psów. W celu przejścia między zwróconymi zgłoszeniami należy przeciągnąć ekran poziomo w lewo lub w prawo. W górnej części znajduje się zdjęcie dopasowanego zwierzaka, z kolei w dolnej części widoku znajdują się dane, które utworzyła zgłoszenie. Znajdziemy tutaj nazwę użytkownika oraz numer telefonu. Dla powyższego przykładu numer telefonu będzie miał początkowo postać **212 *** *****. W celu zobaczenia pełnego numeru należy nacisnąć zielony przycisk **send message**.

Na rysunku 125 przedstawiono widok szczegółów dotyczących dopasowanego zgłoszenia, który ukazuje się po naciśnięciu ikony przedstawiającej małą literę *i*. Ikona ta znajduje się w lewym dolnym rogu fotografii zwierzęcia. Widok szczegółowych informacji o zgłoszeniu uwzględnia lokalizację w postaci miejscowości oraz ulicy, barwy sierści zwierzaka, rozmiar zwierzęcia oraz opis dodatkowych informacji oraz cech psa.

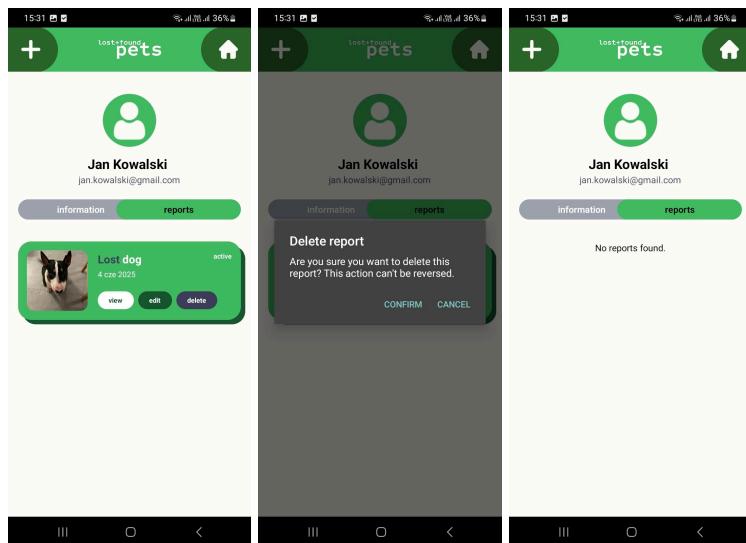
29.5 Usuwanie zgłoszenia

W celu usunięcia dodanego zgłoszenia należy przejść do widoku profilu użytkownika. W tym celu startując z poziomu widoku startowego aplikacji należy nacisnąć ikonę użytkownika znajdującą się w prawym górnym rogu.



Po lewej stronie rysunek nr 126: Widok główny aplikacji z uwzględnieniem ikony użytkownika w prawym górnym rogu ekranu. Po prawej stronie rysunek nr 127: widok profilu użytkownika.

Na rysunku nr 127 przedstawiono widok profilu użytkownika. W celu usunięcia zgłoszenia należy przejść do zakładki **reports**.

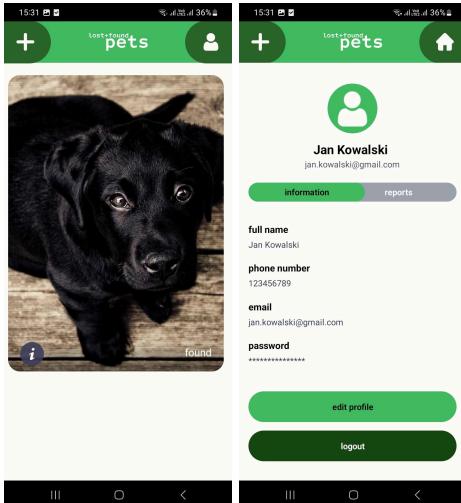


Rysunek nr 128: Widok zgłoszeń użytkownika. Komunikat o usuwaniu zgłoszenia. Widok zgłoszeń użytkownika w przypadku braku posiadanych zgłoszeń.

Na rysunku nr 128 przedstawiono widok zgłoszeń użytkownika. W celu usunięcia zgłoszenia należy nacisnąć szary przycisk z białym napisem **delete**. Po wykonaniu tej akcji na ekranie pojawi się komunikat, który informuje o tym, że operacja usuwania jest nieodwracalna. Użytkownik może przerwać usuwanie naciskając *Cancel* lub potwierdzić za pomocą przycisku *Confirm*. W momencie usunięcia widok zgłoszenia usuwa się z listy zgłoszeń. Sytuację, gdy użytkownik nie posiada żadnych zgłoszeń przedstawia rysunek nr znajdujący się po prawej stronie.

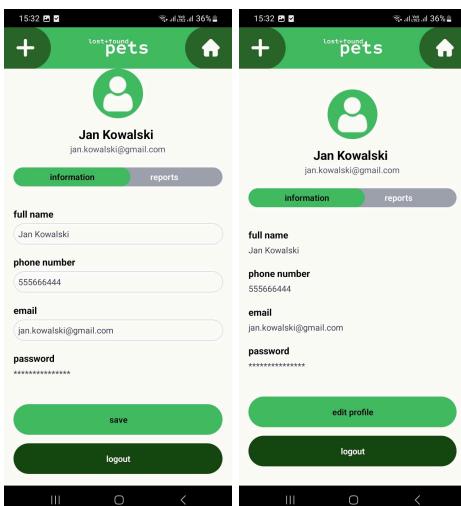
29.6 Edycja danych użytkownika

W celu edycji danych użytkownika takich jak adres e-mail, numer telefonu czy nazwa użytkownika należy przejść do widoku profilu użytkownika za pomocą ikony użytkownika znajdującej się w prawym górnym rogu widoku głównego aplikacji.



Po lewej stronie rysunek nr 129: Widok główny aplikacji z uwzględnieniem ikony użytkownika w prawym górnym rogu ekranu. Po prawej stronie rysunek nr 130: widok profilu użytkownika.

W celu edytowania informacji użytkownika należy nacisnąć zielony przycisk **edit**.



Po lewej stronie rysunek nr 131: modyfikacja informacji o użytkowniku. Po prawej stronie rysunek nr 132: widok profilu użytkownika po zaktualizowaniu danych.

Po naciśnięciu przycisku **edit**, pola możliwe są do edycji. W celu zapisania wprowadzonych zmian należy nacisnąć zielony przycisk **save**.

Sprawdzenie:

Instrukcja została wykonana krok po kroku, wszystkie funkcjonalności działają zgodnie z opisem.