Wyniki etapu II:
Definicja architektury systemu

System zarządzania schroniskiem dla zwierząt

Projektowanie systemów informatycznych

Skład zespołu:

Marta Puz 266879
Aleksandra Piątek 264004
Rafał Starypan 288753

Prowadzący:

dr inż. Bogumiła Hnatkowska

1. Cel

Dokument przedstawia decyzje i ich uzasadnienie oraz ograniczenia i ważne elementy projektu systemu rozwiązania, które wpływają na jego implementację.

2. Cele i ograniczenia architektoniczne

Na podstawie informacji zawartych we wcześniejszych dokumentach wyróżnione zostały następujące cele, które powinna spełniać architektura implementowanego systemu.

Funkcjonalne:

- System umożliwia przeglądanie katalogu zwierząt do adopcji
- Systyem umożliwia pracownikowi przeglądanie, dodawanie i edytowanie danych zwierząt
- System umożliwia wypełnianie formularza zgłoszenia adopcyjnego
- System umożliwia pracownikowi tworzenie historii zdrowia zwierzęcia
- System umożliwia pracownikowi przeglądanie stanu magazynu z zaopatrzeniem
- System umożliwia wolontariuszom i pracownikom przypisanie się do zadań z grafiku

Niefunkcjonalne:

- Zapewnienie niezawodnej komunikacji między modułami wchodzącymi w skład systemu.
- Umożliwienie przechowywania danych użytkowników do celów kontrolnych przez czas wskazany w rozporządzeniu regulującym pracę schronisk dla zwierząt:
    - Przechowywanie w schronisku przez 3 lata od dnia wydania zwierzęcia danych obejmujących imię, nazwisko, miejsce zamieszkania i adres osoby lub nazwę, siedzibę i adres podmiotu, któremu przekazano zwierzę.
    - Przechowywanie w schronisku do dnia wydania psa danych obejmujących imię i nazwisko osoby wyprowadzającej psa na wybieg lub spacer, ujętych w ewidencji, o której mowa w § 7 ust. 5.
    - Przechowywanie w schronisku kopii wypisu przy adopcji zwierzęcia przez 3 lata od dnia wydania zwierzęcia ze schroniska.
- Pokrycie kodu źródłowego automatycznymi testami w celu wykrycia błędów oraz zapewnienia utrzymywalności systemu.
- Autentykacja każdego użytkownika jako warunek uzyskania dostępu do funkcjonalności przypisanych jego roli. Udostępnienie publicznie oferty schroniska (lista zwierząt dostępnych do adopcji, regulamin wolontariatu itp.) bez wymogu logowania dla użytkowników niezalogowanych.
- Kontrola dostępu do funkcjonalności zależnie od roli
- Utrzymywalność i prostota wdrożeń.

3. Decyzje i ich uzasadnienie

Wymaganie niefunkcjonalne	Decyzja	Mechanizmy realizacji w systemie	Uzasadnienie

Niezawodna komunikacja między modułami systemu	Modularny monolit (podział logiczny na moduły domenowe)	Wspólny backend Django podzielony na aplikacje (animals/adoptions/volunteers/supplies/accounts), komunikacja modułów przez wywołania wewnętrzne i wspólny model danych	Zmniejsza ryzyko błędów integracyjnych i problemów sieciowych typowych dla mikroserwisów; ułatwia spójne transakcje i kontrolę zależności między modułami
Retencja danych zgodnie z regulacją (3 lata / do wydania psa / kopia wypisu 3 lata)	Centralna relacyjna baza + polityki retencji + kopie zapasowe	PostgreSQL (RDS), backupy pełne/przyrostowe, ograniczenia usuwania danych (soft delete lub flagi archiwizacji), mechanizmy audytowe (logi zmian)	RDBMS i transakcje ACID ułatwiają egzekwowanie integralności i przechowywanie historii; backup i retencja wspiera wymagania prawne oraz odtwarzanie danych
Pokrycie kodu automatycznymi testami	Testowalna architektura	Testy jednostkowe i integracyjne (Django test framework / pytest), testy API	Modularny podział backendu sprzyja testom w granicach modułów
Autentykacja każdego użytkownika dla funkcji zależnych od roli; część publiczna bez logowania	OAuth2 + wydzielenie publicznych endpointów	Tokeny OAuth2 dla części chronionej, osobne endpointy publiczne (np. lista zwierząt do adopcji), kontrola dostępu w warstwie API	OAuth2 dobrze pasuje do SPA (stateless API), umożliwia wygasanie/odświeżanie tokenów, a jednocześnie pozwala pozostawić część funkcji publicznych bez logowania
Kontrola dostępu do funkcjonalności zależnie od roli	RBAC (Role-Based Access Control)	Role: Pracownik / Wolontariusz / Odwiedzający, sprawdzanie uprawnień na endpointach i w logice aplikacji	RBAC jest czytelny dla domeny schroniska i łatwy do egzekwowania na poziomie API; upraszcza audyt i zarządzanie uprawnieniami
Utrzymywalność i prostota wdrożeń	Konteneryzacja + zarządzana infrastruktura	Docker + AWS ECS (frontend i backend), RDS dla bazy, ALB dla routingu	Usługi zarządzane obniżają koszt operacyjny; kontenery zapewniają powtarzalność środowisk i prostszą skalowalność

4. Mechanizmy architektoniczne

Poniższe decyzje architektoniczne wynikają z założeń projektu oraz celów niefunkcjonalnych (m.in. utrzymywalność, testowalność, bezpieczeństwo, retencja danych, dostęp publiczny do części funkcji oraz dostęp autoryzowany zależny od roli). Każdy mechanizm opisano w formacie ADR: kontekst → decyzja → konsekwencje (pozytywne i negatywne).

4.1 Backend – modularny monolit

Kontekst:
System obejmuje kilka obszarów domenowych (zwierzęta, adopcje, wolontariat, zaopatrzenie, raporty, użytkownicy). Wymagana jest spójność danych i transakcyjność (np. procesy wieloetapowe, zmiany statusów), a wdrożenie ma pozostać możliwie proste.

Decyzja:
Backend będzie **monolitem w Pythonie (Django/DRF)**, podzielonym logicznie na moduły (aplikacje Django) odpowiadające obszarom domenowym.

Konsekwencje (dobre):

- Niższa złożoność wdrożeniowa i utrzymaniowa niż przy mikroserwisach (mniej usług, mniej integracji).
- Łatwiejsze zapewnienie transakcyjności i integralności danych przy wspólnej bazie (ACID).
- Czytelne granice odpowiedzialności dzięki modularizacji (aplikacje Django), co wspiera utrzymywalność i testowanie.

Konsekwencje (złe):

- Ograniczona niezależność wdrożeń i skalowania modułów (skalowanie odbywa się dla całego backendu).
- Ryzyko „rozrostu" monolitu w czasie (jeśli granice modułów nie będą pilnowane).
- Zmiany w jednym module mogą wymuszać wspólny cykl wydawniczy i testy regresji dla całości.

4.2 Autentykacja - OAuth2

Kontekst:
Frontend jest aplikacją SPA, która komunikuje się z backendem po API. Część funkcji ma być publiczna (bez logowania), a część dostępna wyłącznie dla zalogowanych użytkowników. Istotne są bezpieczeństwo, możliwość audytu i kontrola czasu życia sesji.

Decyzja:
Uwierzytelnianie użytkowników będzie realizowane w oparciu o OAuth2 z użyciem tokenów.

Konsekwencje (dobre):

- API może pozostać „stateless" (ułatwia skalowanie poziome backendu).
- Tokeny umożliwiają mechanizmy bezpieczeństwa: czas życia, odświeżanie, unieważnianie, audyt.
- Spójny model dostępu: łatwe rozdzielenie funkcji publicznych i autoryzowanych w SPA.

Konsekwencje (złe):

- Dodatkowa złożoność implementacji i konfiguracji (obsługa tokenów, odświeżanie, unieważnianie).
- Wymaga bezpiecznego przechowywania tokenów po stronie klienta oraz właściwej konfiguracji CORS/CSRF (ryzyko podatności przy błędnej konfiguracji).
- Potencjalna potrzeba dodatkowych mechanizmów (np. rotacja refresh tokenów, rate limiting) dla ograniczenia ryzyka nadużyć.

4.3 Autoryzacja - RBAC

- **Kontekst:**
  System obsługuje różne typy użytkowników (np. Pracownik, Wolontariusz, Odwiedzający) o wyraźnie odmiennych uprawnieniach. Kontrola dostępu powinna być czytelna i możliwa do egzekwowania w API.
- **Decyzja:**
  Autoryzacja zostanie oparta o RBAC (Role-Based Access Control).
- **Konsekwencje (dobre):**
  - Prosty, zrozumiały model uprawnień dopasowany do domeny.
  - Łatwe egzekwowanie uprawnień na poziomie endpointów i/lub warstwy serwisów.
  - Ułatwia testowanie bezpieczeństwa (scenariusze „rola vs. dostęp").
- **Konsekwencje (złe):**
  - Mniejsza elastyczność niż modele oparte o atrybuty (ABAC) przy bardziej złożonych regułach (np. „użytkownik może edytować tylko swoje zgłoszenia" – wymaga dodatkowych warunków poza rolą)
  - Z czasem może pojawić się „eksplozja ról" lub wyjątków, jeśli wymagania uprawnień zaczną się komplikować.
  - Wymaga utrzymywania spójnej macierzy uprawnień i kontroli zmian ról (ryzyko błędów konfiguracyjnych).

4.4 Baza danych – PostgreSQL (AWS RDS)

- **Kontekst:**
  Dane mają relacyjny charakter i zawierają liczne zależności między encjami. Wymagana jest spójność i transakcyjność. Projekt zakłada uruchomienie w AWS, z ograniczeniem nakładu administracyjnego.
- **Decyzja:**
  System będzie używał relacyjnej bazy PostgreSQL uruchomionej jako Amazon RDS. Moduły backendu współdzielą jedną bazę danych.
- **Konsekwencje (dobre):**
  - Naturalne dopasowanie do relacyjnego modelu danych, silna integralność referencyjna.
  - ACID wspiera procesy wieloetapowe i zmiany statusów bez ryzyka niespójności.
  - RDS ogranicza administrację (backupy, aktualizacje, wbudowane mechanizmy HA zależnie od konfiguracji).

    - Wspólna baza zmniejsza złożoność synchronizacji danych między modułami i ryzyko rozjazdów.
- **Konsekwencje (złe):**
    - Wspólna baza może stać się wąskim gardłem skalowania oraz zwiększa sprzężenie modułów (współdzielone schematy/tabele).
    - Potencjalny pojedynczy punkt awarii na poziomie bazy (wymaga odpowiedniej konfiguracji HA/failover w RDS).
    - Trudniejsze izolowanie zmian schematu między modułami (wymagane migracje i koordynacja).

4.5 Frontend – SPA w React

Kontekst:
System zawiera zarówno część publiczną, jak i rozbudowane ekrany operacyjne dla pracowników i wolontariuszy. Frontend ma integrować się z REST API i wspierać wygodną nawigację bez przeładowań.

Decyzja:
Interfejs użytkownika będzie aplikacją SPA w React.

Konsekwencje (dobre):

- Dobre wsparcie dla złożonych widoków i dynamicznej obsługi formularzy (np. adopcje, panel pracownika).
- Naturalna integracja z REST API i mechanizmami tokenowymi.
- Możliwość niezależnego cyklu wdrożeniowego frontend/backend (oddzielne kontenery/usługi).

Konsekwencje (złe):

- Większa złożoność po stronie klienta (stan aplikacji, routing, bezpieczeństwo tokenów).
- SEO i dostępność publicznych treści mogą wymagać dodatkowych działań (np. SSR/prerender), jeśli to będzie istotne.
- Ryzyko problemów z kompatybilnością/backward compatibility API (frontend zależny od kontraktu API).

4.6 Konteneryzacja i uruchomienie na AWS ECS

Kontekst:
Projekt zakłada wdrożenie w chmurze, rozdzielenie warstw frontend/backend i powtarzalność środowisk (dev/test/prod). Wymagane są podstawowe mechanizmy monitoringu i skalowania.

Decyzja:
Backend i frontend będą uruchamiane jako kontenery Docker w AWS ECS (np. Fargate), a baza danych jako RDS.

Konsekwencje (dobre):

- Powtarzalne środowisko uruchomieniowe (łatwiejsze testowanie i wdrożenia).
- Możliwość skalowania usług i integracji z usługami AWS (ALB, CloudWatch).
- Separacja frontendu i backendu jako osobnych usług (czytelniejsza architektura wdrożeniowa).

Konsekwencje (złe):

- Zależność od dostawcy chmury (vendor lock-in) i kosztów usług.
- Dodatkowa złożoność infrastrukturalna (konfiguracja sieci, task definitions, sekrety, logowanie).
- Potencjalnie trudniejsze debugowanie środowisk rozproszonych niż lokalne uruchomienie monolitu.

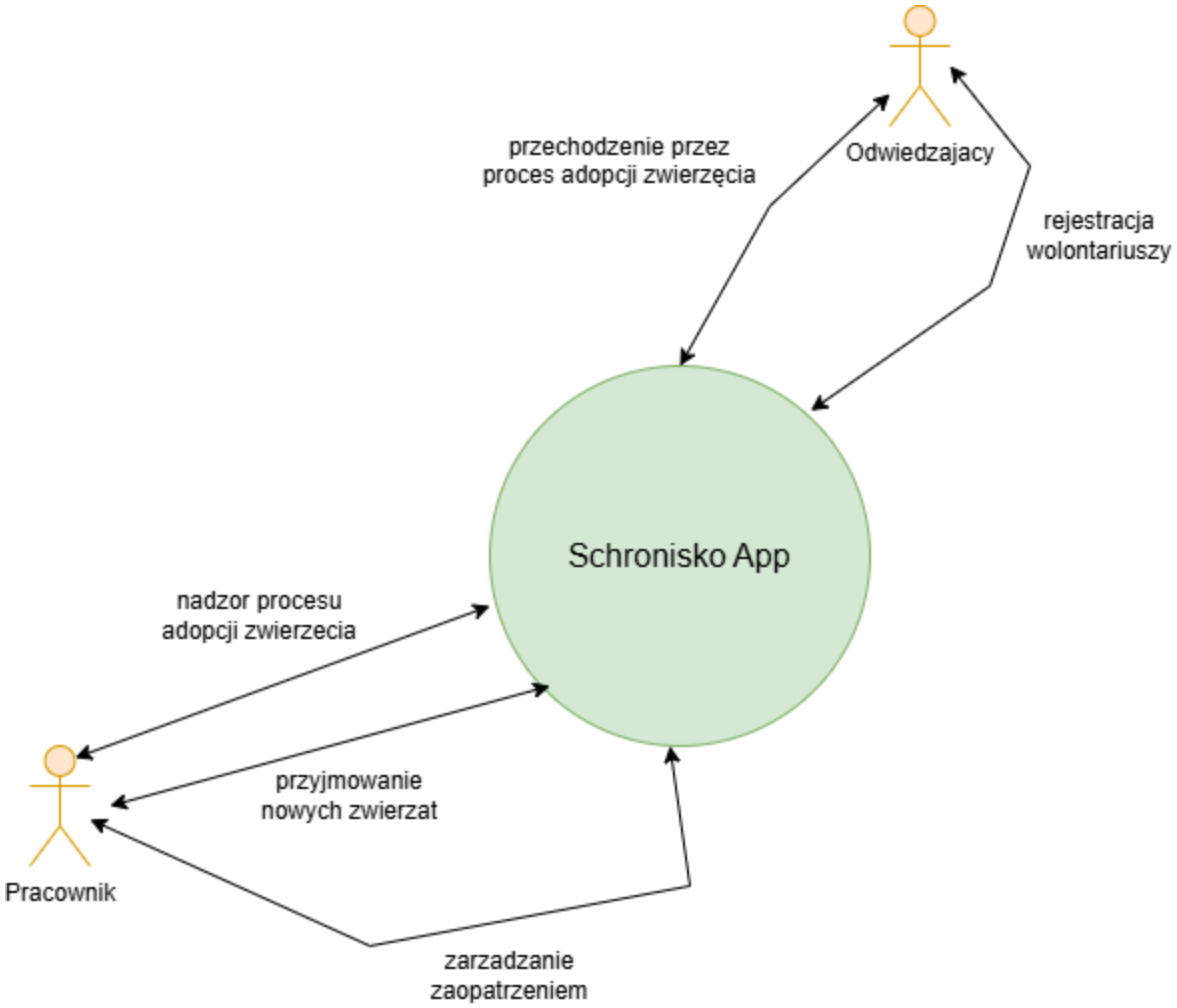
4.7 Ograniczenia realizowanej infrastruktury:

- **Konsekwencje (dobre):**
  - Architektura klient–serwer z API w chmurze ułatwia centralne zarządzanie bezpieczeństwem i wdrożeniami.
  - Jedna baza danych upraszcza spójność danych i ogranicza liczbę punktów integracji.
  - Konteneryzacja zwiększa powtarzalność środowisk.
- **Konsekwencje (złe):**
  - System wymaga stałego dostępu do internetu (brak trybu offline i lokalnego buforowania jako założenie).
  - Zależność od dostępności AWS (ECS/RDS/ALB) wpływa na dostępność systemu.
  - Centralna relacyjna baza danych może być pojedynczym punktem awarii i wąskim gardłem bez właściwego HA. AWS RDS dostarcza rozwiązania takie jak automatyczne tworzenie replik, przywracanie bazy z backupu w przypadku awarii, które pozwalają na ograniczenie negatywnych konsekwencji.
  - Ograniczona niezależność skalowania poszczególnych modułów backendu (modularny monolit).

5. Widoki architektoniczne

Widok kontekstowy

5.1.1. Diagram kontekstowy



5.1.2. Scenariusze interakcji

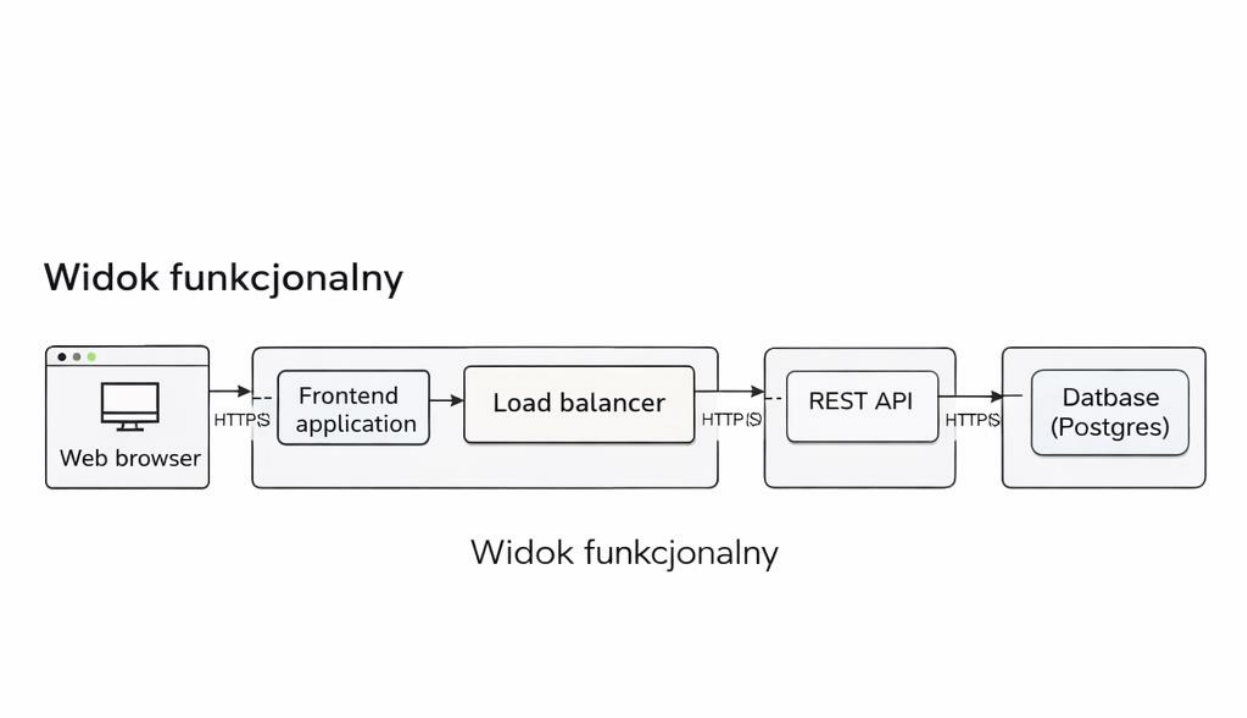
Nie dotyczy – brak zewnętrznych serwisów

5.1.3. Interfejsy integracyjne – poziom logiczny

Nie dotyczy – brak zewnętrznych serwisów

6. Widok funkcjonalny

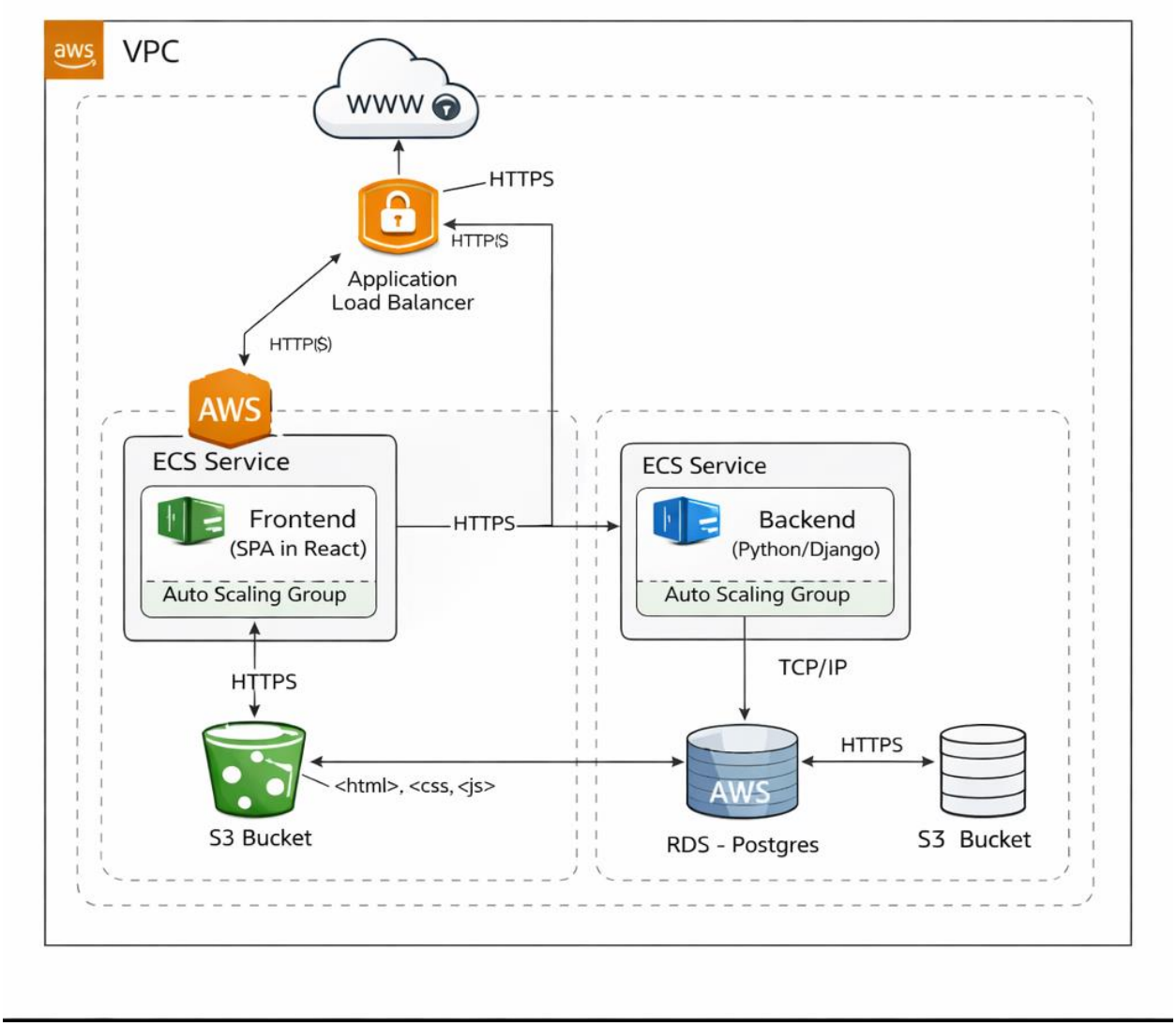
Diagram komponentów



Widok funkcjonalny

7. Widok rozmieszczenia

Diagram rozmieszczenia



Opis węzłów

Load balancer

Ogólne informacje	
Nazwa	ALB – Application Load Balancer
Węzeł wirtualny?	Tak (usługa zarządzana)
Centrum danych?	PDC (AWS – region wg wdrożenia, np. eu-central-1)
OS	Nie dotyczy (usługa zarządzana)
Opis	Terminacja TLS/HTTPS, routing ruchu: „/” → frontend, „/api/*” → backend, health-check usług

Konfiguracja sprzętowa	
Dostawca	AWS
Procesor	Nie dotyczy (usługa zarządzana)
RAM	Nie dotyczy (usługa zarządzana)
HDD	Nie dotyczy (usługa zarządzana)
RAID i HDD Netto	Nie dotyczy (usługa zarządzana)
RAID?	Nie dotyczy (usługa zarządzana)
Net cards bonding	Nie dotyczy (usługa zarządzana)

Konfiguracja oprogramowania	
Użytkownicy i grupy użytkowników	Nie dotyczy (usługa zarządzana)
Poziom pracy systemu, czy jest wymagane środowisko graficzne	Nie dotyczy (usługa zarządzana)
Dodatkowe pakiety z dystrybucji systemu	Nie dotyczy (usługa zarządzana)
Dodatkowe pakiety spoza dystrybucji systemu	Nie dotyczy (usługa zarządzana)

Frontend

Ogólne informacje	
Nazwa	ECS Service – Frontend (React SPA)
Węzeł wirtualny?	Tak (kontener w ECS/Fargate)
Centrum danych?	PDC (AWS – region wg wdrożenia, np. eu-central-1)
OS	Linux
Opis	Serwowanie statycznych plików SPA (build React) np. przez Nginx

Konfiguracja sprzętowa	
Dostawca	AWS
Procesor	Konfigurowane w ECS Task Definition
RAM	Konfigurowane w ECS Task Definition
HDD	Konfigurowane w ECS Task Definition
RAID i HDD Netto	Nie dotyczy
RAID?	Nie dotyczy

Net cards bonding	Nie dotyczy

Konfiguracja oprogramowania	
Użytkownicy i grupy użytkowników	Nie dotyczy
Poziom pracy systemu, czy jest wymagane środowisko graficzne	Nie dotyczy
Dodatkowe pakiety z dystrybucji systemu	Nie dotyczy
Dodatkowe pakiety spoza dystrybucji systemu	Nie dotyczy

Backend

Ogólne informacje	
Nazwa	ECS Service – Backend
Węzeł wirtualny?	Tak (kontener w ECS/Fargate)
Centrum danych?	PDC (AWS – region wg wdrożenia, np. eu-central-1)
OS	Linux
Opis	REST API (Django/DRF), autentykacja OAuth2 (tokeny), logika biznesowa monolitu modularnego; połączenie do RDS PostgreSQL

Konfiguracja sprzętowa	
Dostawca	AWS
Procesor	Konfigurowane w ECS Task Definition
RAM	Konfigurowane w ECS Task Definition
HDD	Konfigurowane w ECS Task Definition
RAID i HDD Netto	Nie dotyczy (usługa zarządzana)
RAID?	Nie dotyczy (usługa zarządzana)
Net cards bonding	Nie dotyczy (usługa zarządzana)

Konfiguracja oprogramowania	
Użytkownicy i grupy użytkowników	Nie dotyczy (usługa zarządzana)
Poziom pracy systemu, czy jest wymagane środowisko graficzne	Nie dotyczy (usługa zarządzana)
Dodatkowe pakiety z dystrybucji systemu	Nie dotyczy (usługa zarządzana)

Dodatkowe pakiety spoza dystrybucji systemu	Nie dotyczy (usługa zarządzana)

Postgres

Ogólne informacje	
Nazwa	Amazon RDS – PostgreSQL
Węzeł wirtualny?	Tak (usługa zarządzana)
Centrum danych?	PDC (AWS – region wg wdrożenia, np. eu-central-1)
OS	Nie dotyczy (usługa zarządzana)
Opis	Główna relacyjna baza danych PostgreSQL dla systemu; dostęp tylko z backendu (security group)

Konfiguracja sprzętowa	
Dostawca	AWS RDS
Procesor	Zależny od klasy instancji
RAM	Zależny od klasy instancji
HDD	rozmiar wg potrzeb
RAID i HDD Netto	Nie dotyczy (usługa zarządzana)
RAID?	Nie dotyczy (usługa zarządzana)
Net cards bonding	Nie dotyczy (usługa zarządzana)

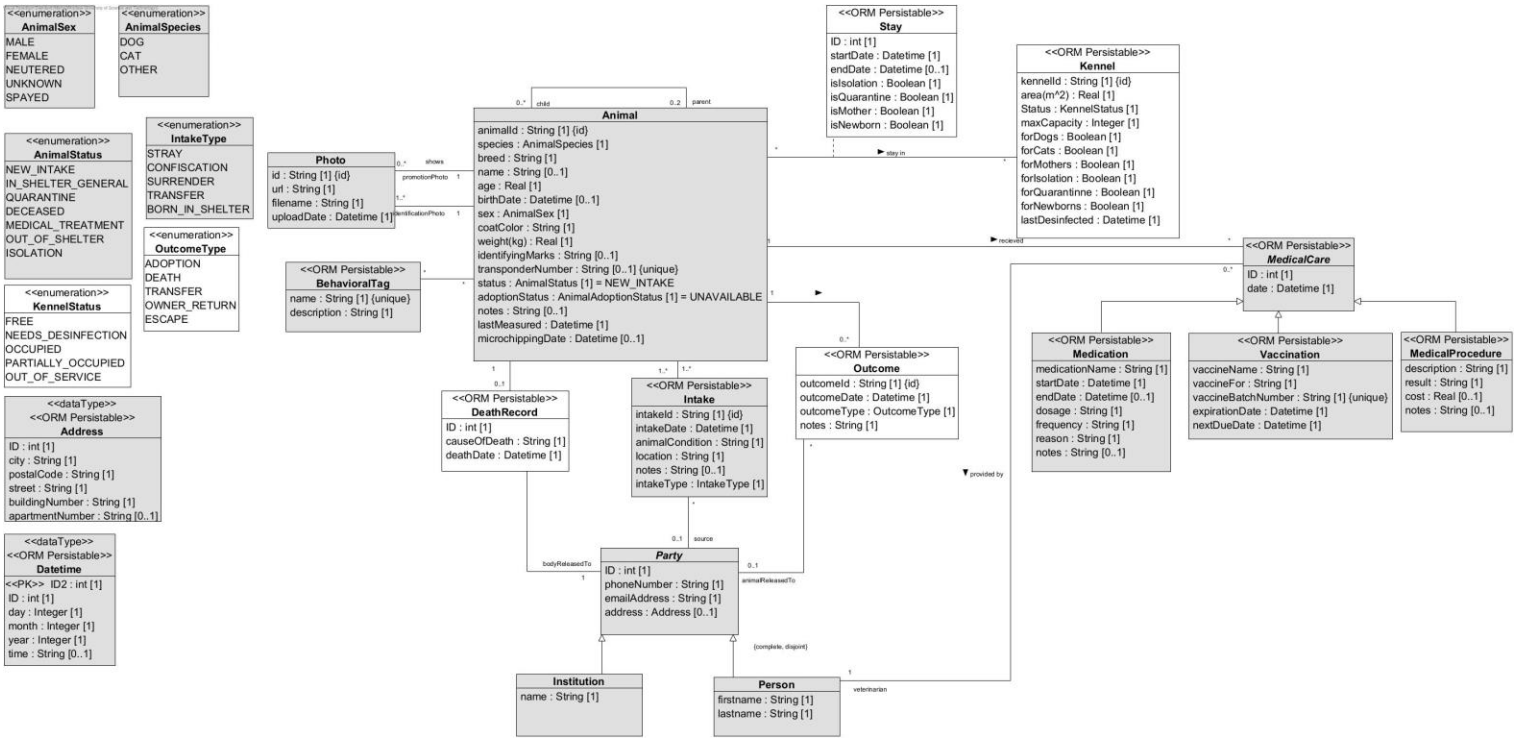
Konfiguracja oprogramowania	
Użytkownicy i grupy użytkowników	Nie dotyczy (usługa zarządzana)
Poziom pracy systemu, czy jest wymagane środowisko graficzne	Nie dotyczy (usługa zarządzana)
Dodatkowe pakiety z dystrybucji systemu	Nie dotyczy (usługa zarządzana)
Dodatkowe pakiety spoza dystrybucji systemu	Nie dotyczy (usługa zarządzana)

8. Widok informacyjny

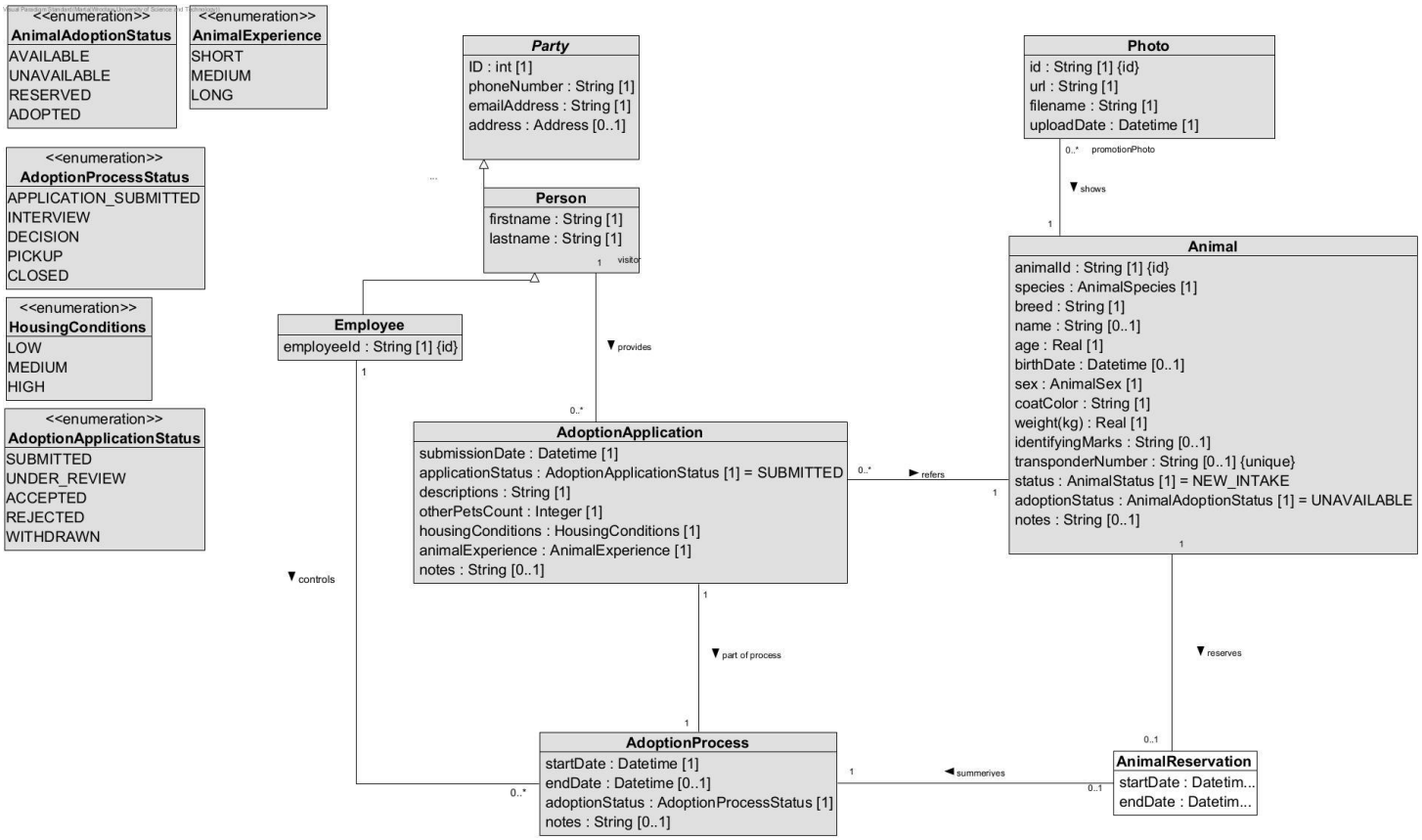
Kolorem szarym zostały oznaczone klasy związane z wybranymi przypadkami użycia, przeznaczone do implementacji.

8.1. Model informacyjny

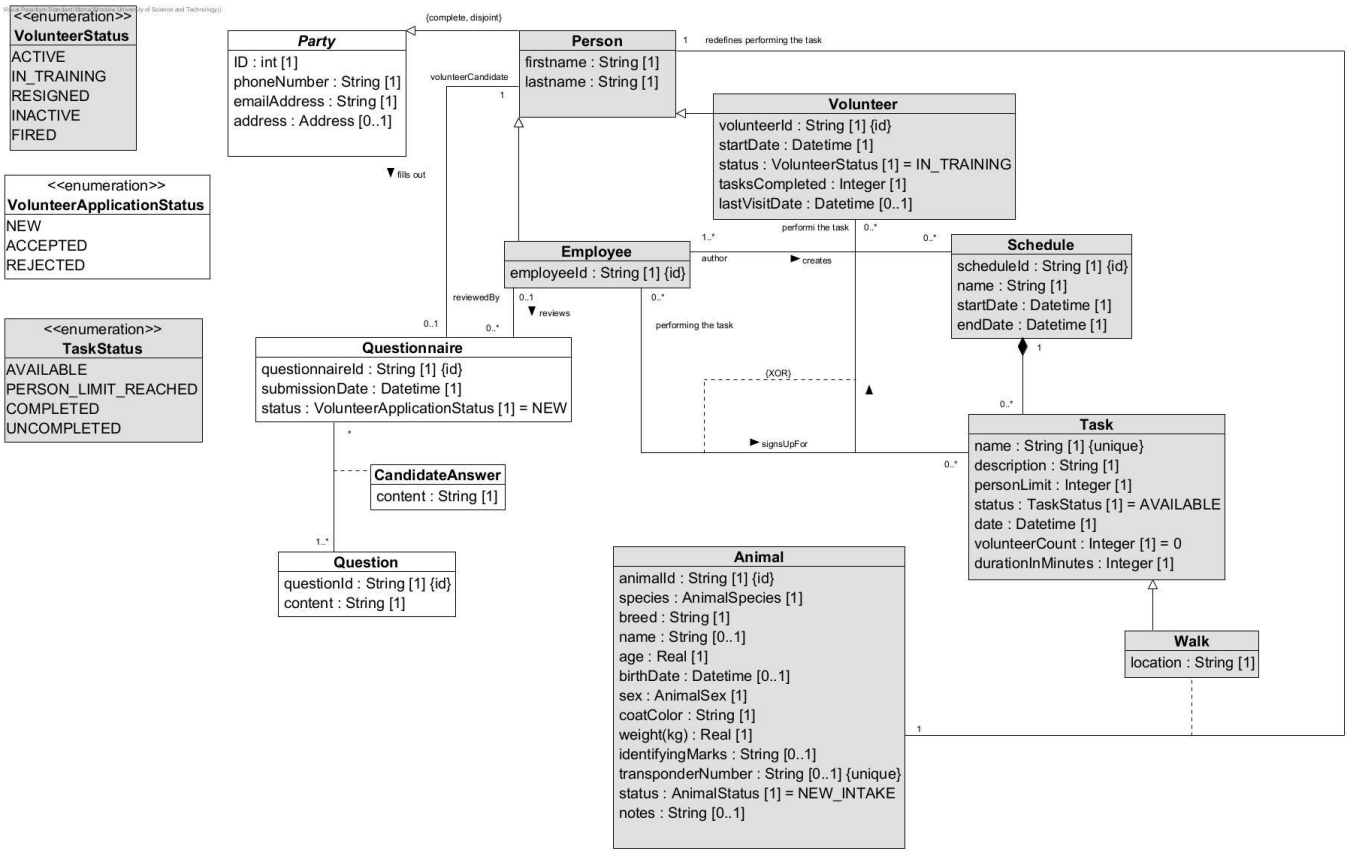
Opieka nad zwierzętami



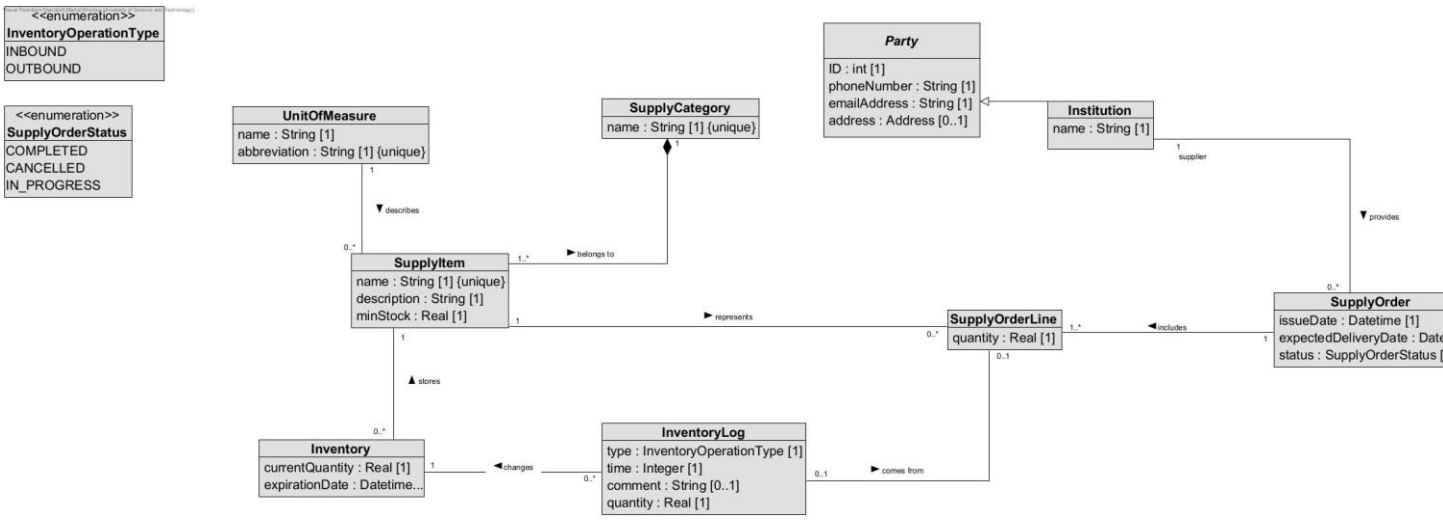
Adopcje


<<enumeration>>
AnimalAdoptionStatus
AVAILABLE
UNAVAILABLE
RESERVED
ADOPTED
<<enumeration>>
AnimalExperience
SHORT
MEDIUM
LONG
<<enumeration>>
AdoptionProcessStatus
APPLICATION_SUBMITTED
INTERVIEW
DECISION
PICKUP
CLOSED
<<enumeration>>
HousingConditions
LOW
MEDIUM
HIGH
<<enumeration>>
AdoptionApplicationStatus
SUBMITTED
UNDER_REVIEW
ACCEPTED
REJECTED
WITHDRAWN
Party
ID : int [1]
phoneNumber : String [1]
emailAddress : String [1]
address : Address [0..1]
Person
firstname : String [1]
lastname : String [1]
visitor
Employee
employeeId : String [1] {id}
provides
controls
Photo
id : String [1] {id}
url : String [1]
filename : String [1]
uploadDate : Datetime [1]
promotionPhoto
shows
Animal
animalId : String [1] {id}
species : AnimalSpecies [1]
breed : String [1]
name : String [0..1]
age : Real [1]
birthDate : Datetime [0..1]
sex : AnimalSex [1]
coatColor : String [1]
weight(kg) : Real [1]
identifyingMarks : String [0..1]
transponderNumber : String [0..1] {unique}
status : AnimalStatus [1] = NEW_INTAKE
adoptionStatus : AnimalAdoptionStatus [1] = UNAVAILABLE
notes : String [0..1]
AdoptionApplication
submissionDate : Datetime [1]
applicationStatus : AdoptionApplicationStatus [1] = SUBMITTED
descriptions : String [1]
otherPetsCount : Integer [1]
housingConditions : HousingConditions [1]
animalExperience : AnimalExperience [1]
notes : String [0..1]
refers
part of process
reserves
AdoptionProcess
startDate : Datetime [1]
endDate : Datetime [0..1]
adoptionStatus : AdoptionProcessStatus [1]
notes : String [0..1]
summerives
AnimalReservation
startDate : Datetim...
endDate : Datetim...

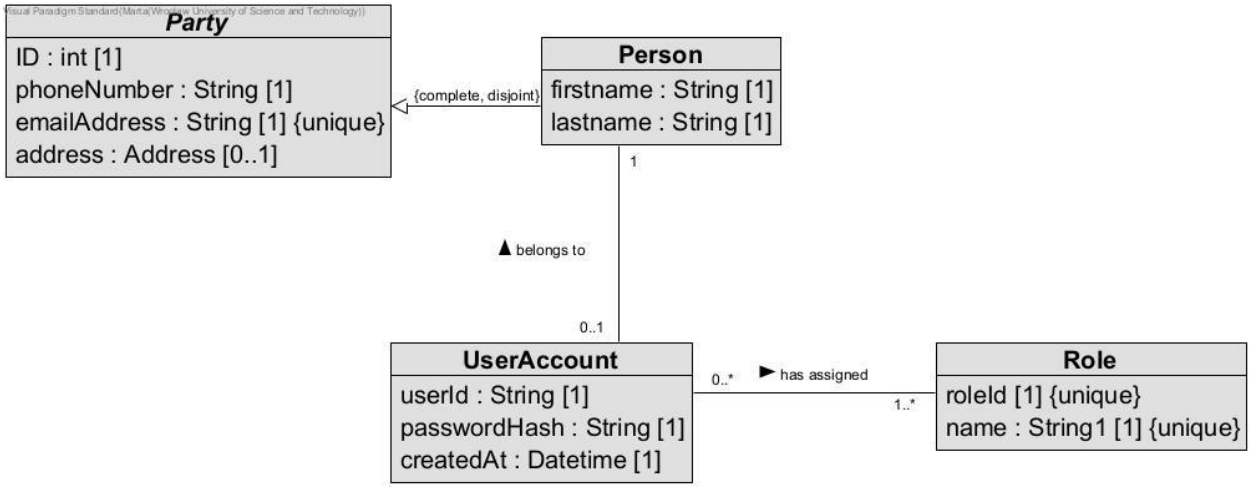
Wolontariat



Zaopatrzenie



Użytkownicy systemu



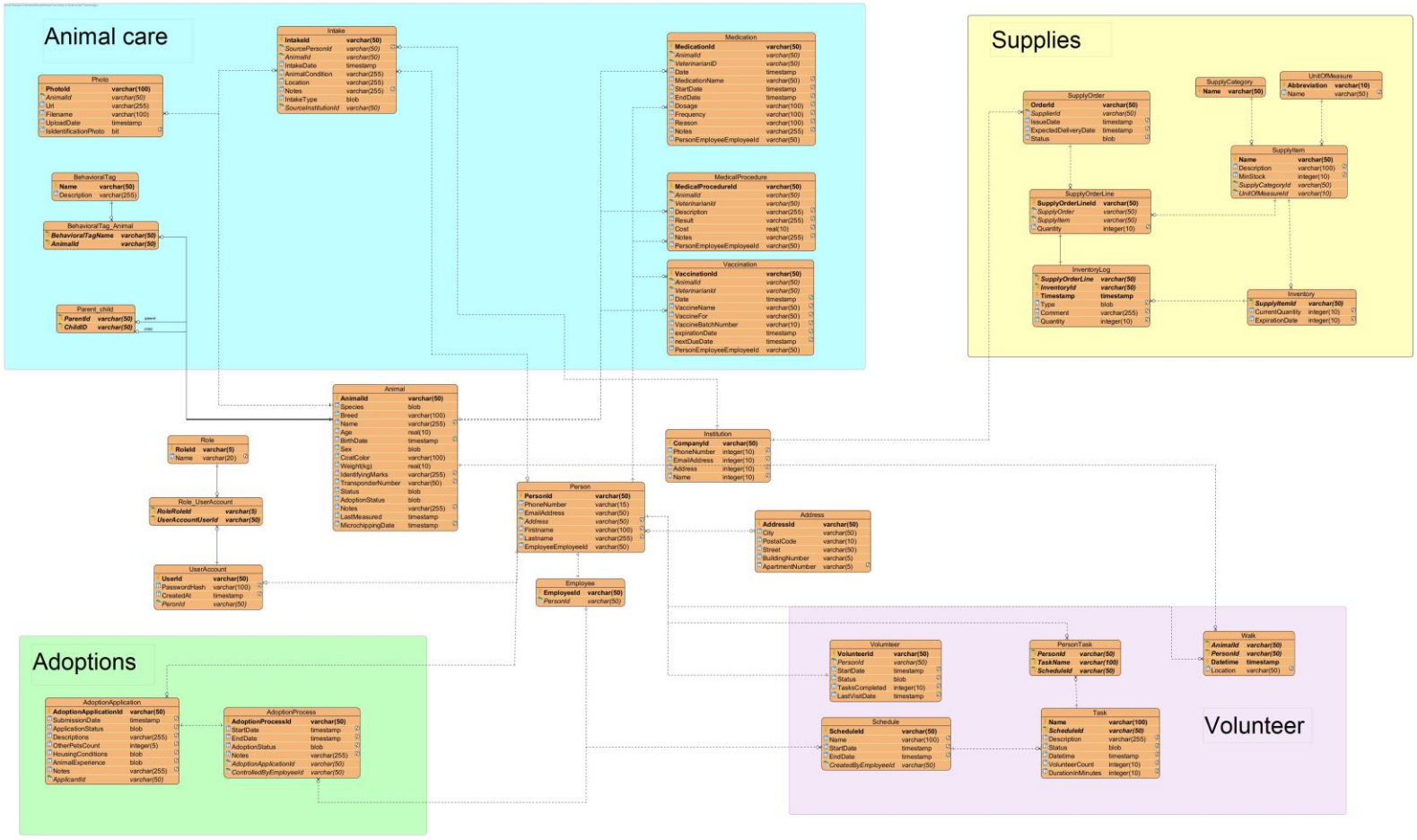
9. Projekt bazy danych

Ogólne informacje nt. bazy danych (osobna tabela dla każdej bazy)	
SID/Service Name	postgres
Nazwa serwera	serwer-db01
Port	5432
Type	PostgreSQL
Kodowanie znaków	UTF8
Opis	Relacyjna baza danych PostgreSQL, głowna baza systemu.

Backup

Backupy obejmują całość danych systemu, w tym dane operacyjne i historyczne wszystkich modułów backendu. Pełne kopie zapasowe wykonywane są automatycznie raz w tygodniu, a kopie przyrostowe codziennie. Okres retencji kopii zapasowych wynosi co najmniej 3 lata, co zapewnia zgodność z wymogami regulacji dotyczących pracy schronisk dla zwierząt. Kopie zapasowe muszę przechowywane w dwóch lokalizacjach i regularnie testowane pod kątem odtworzenia.

Schemat 1 - System zarządzania schroniskiem



Oszacowanie rozmiaru bazy

Pojemność bazy danych została oszacowana na podstawie analizy diagramu ERD oraz przyjętych założeń dotyczących skali działalności schroniska. Przyjęto, że początkowy stan systemu obejmuje 20 zwierząt, natomiast roczna rotacja zwierząt wynosi około 100 przypadków. Liczbę użytkowników oszacowano na 10 pracowników oraz 20 wolontariuszy w momencie uruchomienia systemu, z przewidywanym wzrostem liczby wolontariuszy do około 30 po roku. Dodatkowo założono około 50 procesów adopcyjnych rocznie. Aktywność systemu oszacowano na podstawie średniej liczby zdarzeń, takich jak około 100 zadań miesięcznie oraz średnio 5 zabiegów medycznych przypadających na jedno zwierzę w skali

roku. Przyjęto, że schronisko będzie operować z pomocą 50 środków zaopatrzenia, dostarczanych średnio raz na miesiąc w 5 zamówieniach.

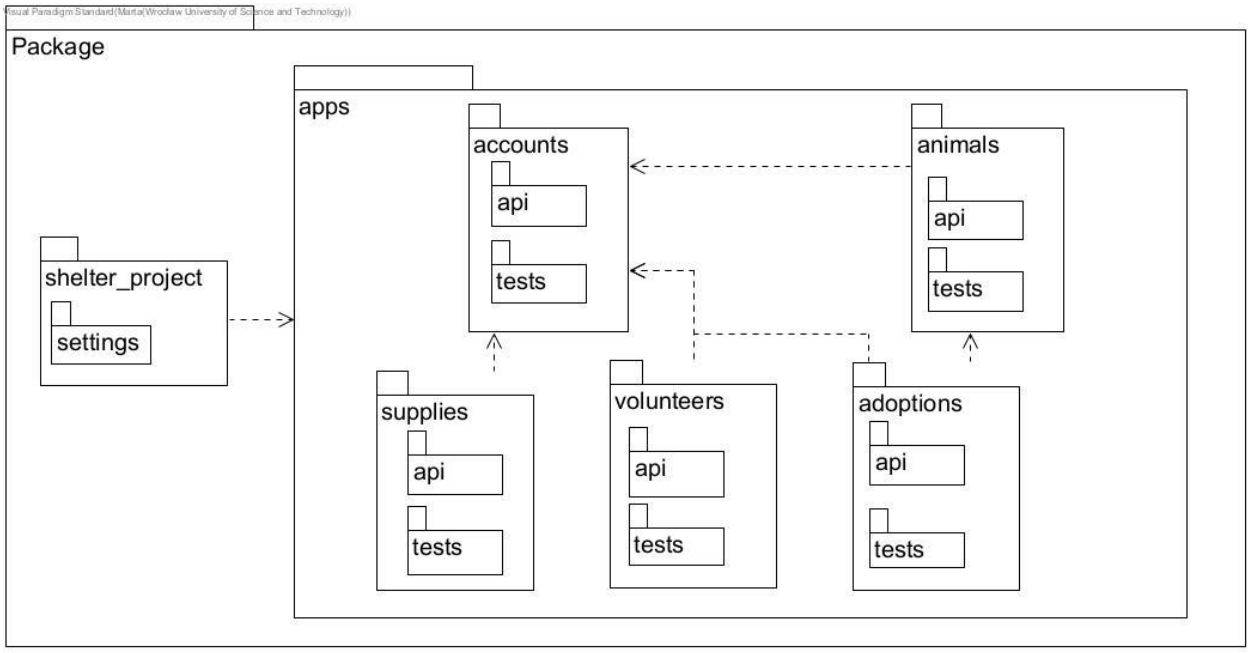
Informacje o schemacie	
Nazwa	System zarządzania schroniskiem
Początkowa pojemność	~3MB
Przyrost pojemności (rok)	~17MB
Niezbędne prawa	FULL_ACCESS
Inne	PostgreSQL przewiduje możliwość użycia typu enumeracyjnego, jedank w Visual Paradigm 17.3 enum nie jest jednym z dostępnych typów w diagramach ERD. W związku z tym, tam na diagramie zastąpiono go typem blob. W implementacji typ zostanie ustawiony na enum z iterałami odpowiadającymi tym z diagramów klas.

10. Widok wytwarzania

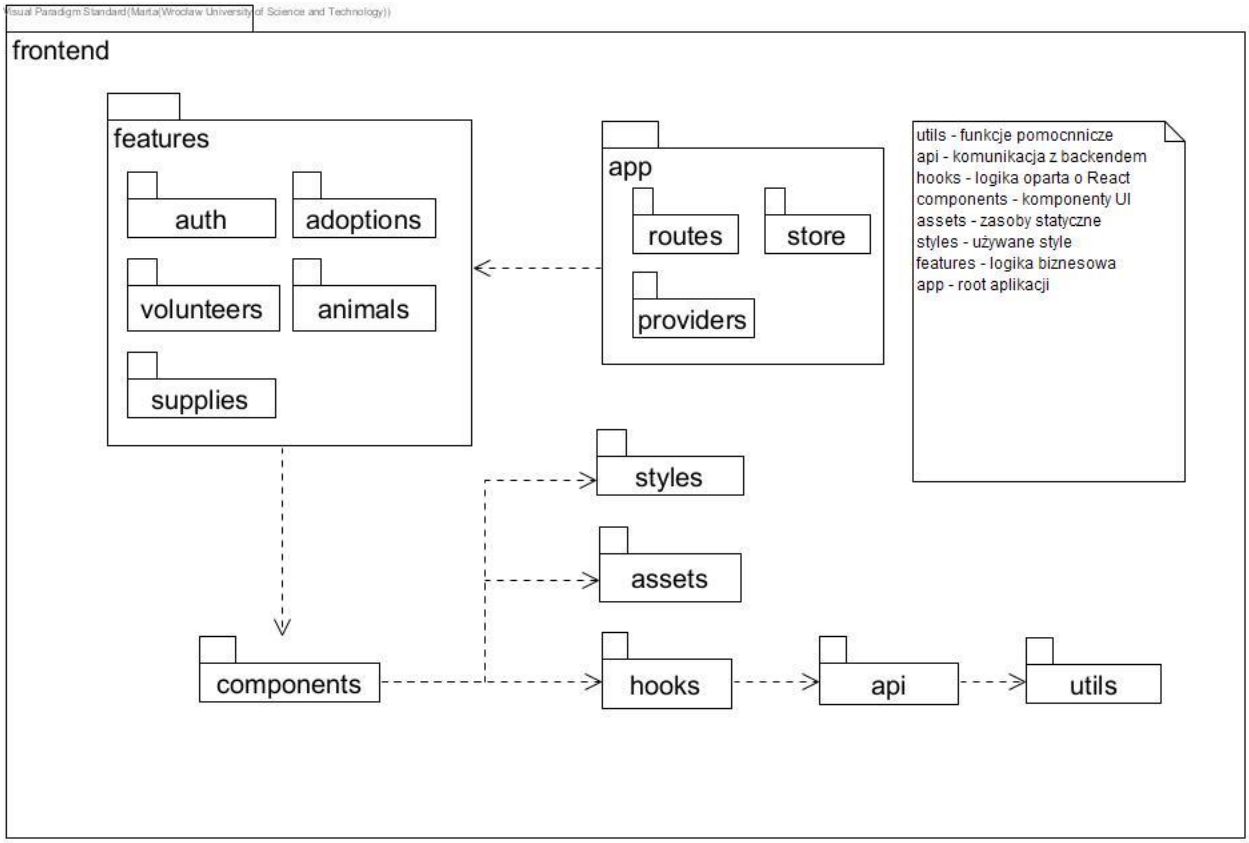
Struktura repozytorium

```
repo/
├── backend/
├── frontend/
├── infrastructure/
```

Struktura kodu backendowego

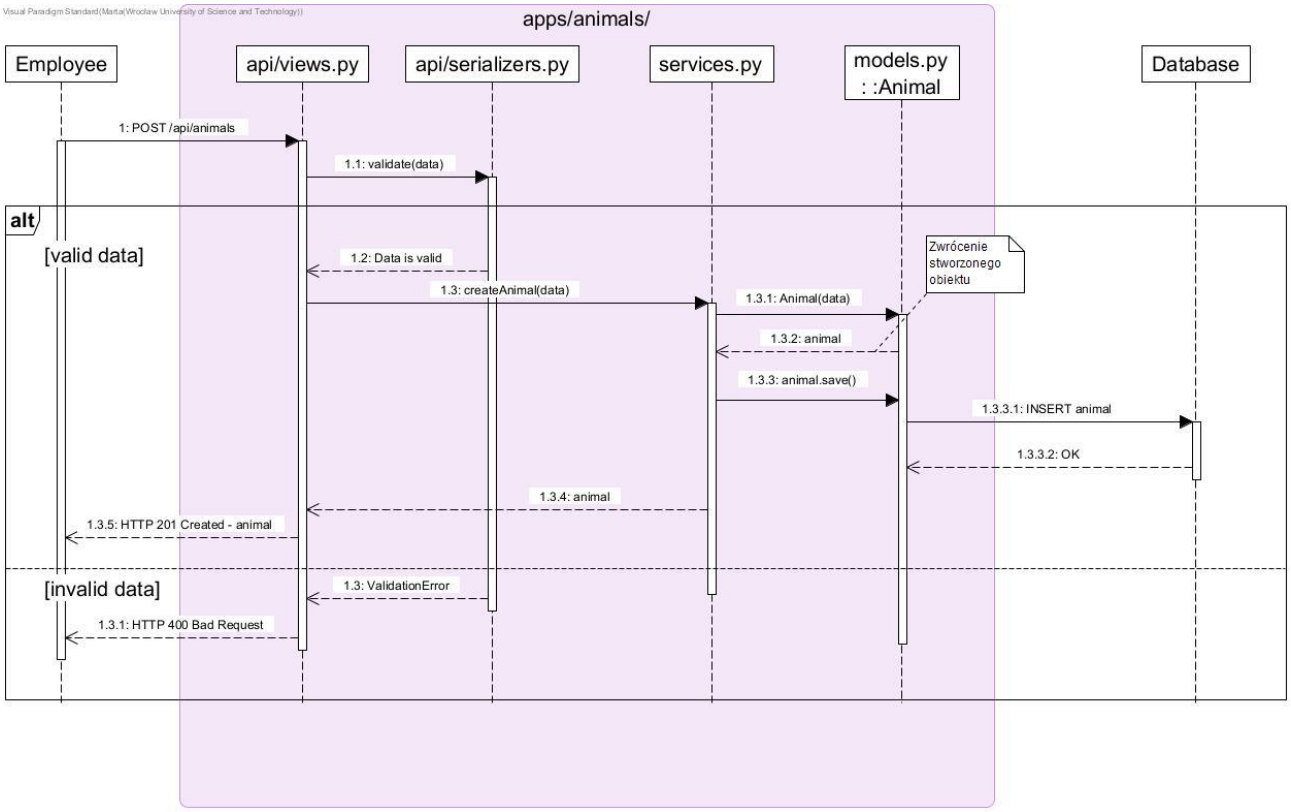


Struktura kodu frontendowego



11. Realizacja przypadków użycia

Dodawanie danych nowego zwierzęcia


Visual Paradigm Standard(Marta(Wroclaw University of Science and Technology))
apps/animals/
Employee
api/views.py
api/serializers.py
services.py
models.py
: :Animal
Database
1: POST /api/animals
1.1: validate(data)
alt
[valid data]
1.2: Data is valid
1.3: createAnimal(data)
1.3.1: Animal(data)
Zwrócenie
stworzonego
obiektu
1.3.2: animal
1.3.3: animal.save()
1.3.3.1: INSERT animal
1.3.3.2: OK
1.3.4: animal
1.3.5: HTTP 201 Created - animal
[invalid data]
1.3: ValidationError
1.3.1: HTTP 400 Bad Request