POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania
Inżynieria Oprogramowania i Baz Danych

Małgorzata Bieniek, s16026
Adrian Borkowski, s15013
Paweł Celejewski, s15799
Aleksandra Grabowska, s16195
Michał Jabłoński, s15747
Rafał Pajączkowski, s9647
Renata Pigoń, s16139
Marcin Szarek, s15541
Jan Szczawiński, s8840
Justyna Tomaszewska, s15313
Łukasz Zajkowski, s16347

@zor
System ułatwiający organizację pracy w schronisku dla zwierząt

Praca inżynierska
napisana pod kierunkiem:

dr inż. Mariusza Trzaski

Warszawa, lipiec 2020

Streszczenie

Celem pracy jest stworzenie aplikacji służącej do zarządzania schroniskiem dla zwierząt. System ma za zadanie pomóc pracownikom w zbieraniu i zarządzaniu informacjami na temat zwierząt, organizacji pracy i prowadzeniu strony internetowej schroniska. System składa się z aplikacji webowej i mobilnej na platformę Android. Obie zostały napisane w języku Java, ponadto wersja webowa wykorzystuje język JavaScript.

Słowa kluczowe

java, spring, hibernate, maven, mysql, html, css, javascript, bootstrap, rest api, intellij, postman, android, aplikacja webowa, aplikacja mobilna, aplikacja dla schronisk, zarządzanie schroniskiem, schronisko dla zwierząt, adopcja zwierząt, adopcja wirtualna zwierząt, zarządzanie pracownikami, zarządzanie zwierzętami w schronisku, zarządzanie systemem, strona internetowa dla schronisk, mobilna aplikacja do zarządzania zwierzętami

Spis treści

1.Wstęp

W Polsce istnieje ponad 200 zarejestrowanych schronisk dla zwierząt. Organizacje te, ze względu na charakter pracy oraz wymogi prawne, zbierają szeroki zakres danych dotyczących zwierząt i osób. Na polskim rynku brakuje systemów, które ułatwiałyby zarządzanie schroniskami, a te które są dostępne mają archaiczny wygląd nie odpowiadający aktualnym zasadom projektowania interfejsu użytkownika. Podobnie bywa ze stronami internetowymi placówek, a przecież najskuteczniejszym medium służącym szukaniu nowych właścicieli jest właśnie Internet. Dobrze zaprojektowana strona, po której łatwo się poruszać i można szybko znaleźć potrzebne informacje, pomaga zachęcić do przygarnięcia zwierzęcia lub wesprzeć schronisko w inny sposób. Współczesne technologie umożliwiają połączenie funkcji: zbierania danych i zarzadzania nimi oraz stworzenie strony zintegrowanej z systemem, dlatego nasz zespół postanowił stworzyć aplikację wspierającą schroniska na obu polach.

1.1. Cel pracy

Celem niniejszej pracy inżynierskiej jest stworzenie aplikacji służącej do zarządzania schroniskiem dla zwierząt oraz tekstu pracy dokumentującej proces jej powstawania. System ma pomagać pracownikom w zbieraniu i zarządzaniu informacjami na temat zwierząt oraz osób, organizacji pracy i prowadzeniu strony internetowej schroniska. Projekt jest realizowany w ramach specjalizacji Inżynieria Oprogramowania i Baz Danych w Polsko-Japońskiej Akademii Technik Komputerowych.

1.2. Organizacja pracy

Niniejsza praca opisuje proces tworzenia aplikacji i dokumentuje działalność całego zespołu. W rozdziale drugim przedstawiono specyfikę działania schronisk oraz kontekst prawny, w którym funkcjonują, bezpośrednio wpływający na wymagania stawiane aplikacji. W rozdziale trzecim można zapoznać się z wybranymi aplikacjami istniejącymi na rynku polskim i zagranicznym, wspomagającymi pracę w schroniskach dla zwierząt. Kolejne rozdziały są poświęcone powstałemu systemowi. Rozdział czwarty zawiera wymagania funkcjonalne i niefunkcjonalne oraz projekt aplikacji: diagramy przypadków użycia, wybrane scenariusze, analityczny i projektowy diagram klas oraz opis etapów powstawania projektu i zachodzących w nim zmian. Rozdział piąty opisuje wykorzystane technologie. W rozdziale szóstym i siódmym można zapoznać się ze szczegółami implementacyjnymi i interfejsem użytkownika. Przedostatni rozdział zawiera opis wykonanych testów, a po nim następuje podsumowanie całego procesu wytwórczego.

1.3. Zakres prac

Tworzenie aplikacji odbywało się w jedenastoosobowym zespole. Zespół został podzielony na pięć podzespołów:

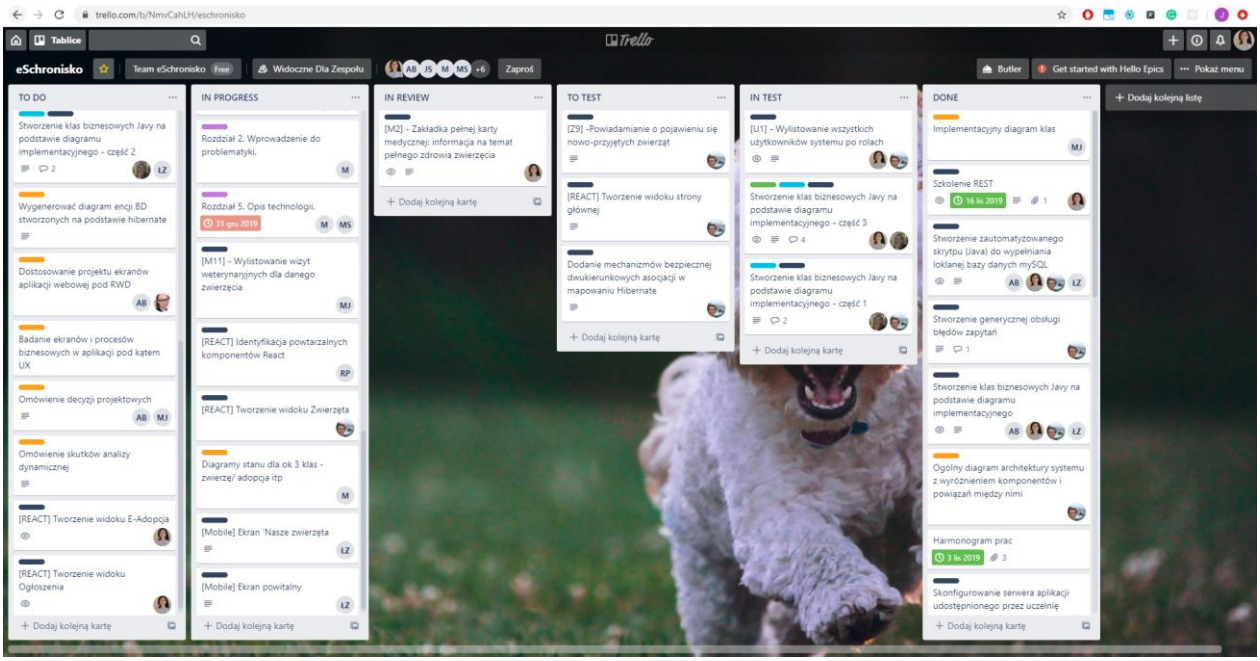
1. Analizy;
2. Projektowania;
3. Implementacji;
4. Testów;
5. Dokumentacji.

Każda z osób uczestniczących w projekcie należała do co najmniej dwóch zespołów.

Planowanie zadań odbywało się podczas zajęć oraz spotkań poza zajęciami i było dokumentowane za pomocą narzędzia Trello [1]. Zadanie umieszczone na tablicy Trello było oznaczone kolorem – innym dla każdego zespołu i przypisane konkretnej osobie (lub osobom) odpowiedzialnej za jego wykonanie. Śledzenie postępu prac było możliwe dzięki podziałowi na kategorie:

- To do;
- In progres;
- In review;
- To test;
- In test;
- Done.

Planowanie zadań i monitorowanie postępu ich wykonania należało do obowiązków kierownika zespołu. Rysunek 1 przedstawia zadania zaplanowane za pomocą tablicy Trello.



Rysunek 1. Zadania zaplanowane za pomocą tablicy Trello. Źródło: opracowanie własne.

Tabela 1 przedstawia skład podzespołów wraz z ich kierownikami oraz szczegółowy wykaz zadań wykonywanych w poszczególnych zespołach. Zadania zawierają informację o osobach odpowiedzialnych za ich wykonanie. W Dodatku A można zapoznać się z zakresem prac z podziałem na osoby.

Tabela 1. Podział na podzespoły i przydział zadań w projekcie @zor. Źródło: opracowanie własne.

Kierownik projektu @azor
Marcin Szarek

Zespół analityczny	
Kierownik	Jan Szczawiński
Członkowie zespołu	Małgorzata Bieniek Adrian Borkowski Aleksandra Grabowska Renata Pigoń Marcin Szarek
Podział zadań	
Stan sztuki	Małgorzata Bieniek Adrian Borkowski Aleksandra Grabowska
Funkcjonalności systemu	Jan Szczawiński Małgorzata Bieniek Adrian Borkowski Aleksandra Grabowska Renata Pigoń Marcin Szarek
Przypadki użycia	Jan Szczawiński Małgorzata Bieniek Aleksandra Grabowska Renata Pigoń Marcin Szarek
Analityczny diagram klas	Adrian Borkowski
Diagramy aktywności	Jan Szczawiński Małgorzata Bieniek Aleksandra Grabowska Marcin Szarek
Scenariusze	Małgorzata Bieniek Aleksandra Grabowska Renata Pigoń Marcin Szarek

Zespół projektowy	
Kierownik	Renata Pigoń

Członkowie zespołu	Małgorzata Bieniek Adrian Borkowski Paweł Celejewski Michał Jabłoński Rafał Pajączkowski Łukasz Zajkowski
Podział zadań	
Projektowy diagram klas	Michał Jabłoński
Diagramy stanu	Adrian Borkowski Małgorzata Bieniek
Ogólny diagram architektury systemu z wyróżnieniem komponentów i powiązań między nimi	Paweł Celejewski
Projektowanie ekranów aplikacji webowej	Adrian Borkowski Renata Pigoń
Projektowanie aplikacji mobilnej	Łukasz Zajkowski
Projektowanie kalendarza	Adrian Borkowski
Projektowanie raportów	Małgorzata Bieniek
Walidacja wymagań i funkcjonalności	Renata Pigoń
Logo aplikacji	Małgorzata Bieniek

Zespół implementacji	
Kierownik	Paweł Celejewski
Członkowie zespołu	Adrian Borkowski Aleksandra Grabowska Michał Jabłoński Rafał Pajączkowski Renata Pigoń Jan Szczawiński Justyna Tomaszewska Łukasz Zajkowski
Podział zadań	
Szkolenie REST/Git	Justyna Tomaszewska
Szkolenie SpringBoot/Hibernate	Paweł Celejewski
Szkolenie Android Studio	Łukasz Zajkowski
Szkolenie JavaScript Fetch API, React, Angular	Rafał Pajączkowski
Konfiguracja konta i tablicy Trello	Justyna Tomaszewska

Przygotowanie do konfiguracji stacji roboczych/stack technologiczny	Paweł Celejewski
Konfiguracja serwera uczelnianego oraz wdrożenie aplikacji	Paweł Celejewski
Implementacja klas z diagramu projektowego/mapowanie Hibernate/CRUD API	Paweł Celejewski Adrian Borkowski Justyna Tomaszewska Łukasz Zajkowski
Implementacja generycznej obsługi błędów zapytań REST	Paweł Celejewski
Implementacja skryptu wypełniającego bazę danych	Paweł Celejewski
Tworzenie widoków w REACT	Paweł Celejewski Aleksandra Grabowska Michał Jabłoński Rafał Pajączkowski Jan Szczawiński Justyna Tomaszewska Renata Pigoń
Integracja Front-end - Back-End	Adrian Borkowski Paweł Celejewski Aleksandra Grabowska Michał Jabłoński Rafał Pajączkowski Jan Szczawiński Justyna Tomaszewska
Implementacja modułów systemu	Adrian Borkowski Paweł Celejewski Aleksandra Grabowska Michał Jabłoński Rafał Pajączkowski Jan Szczawiński Justyna Tomaszewska
Szyfrowanie danych logowania	Paweł Celejewski Justyna Tomaszewska
Rejestracja użytkowników	Paweł Celejewski
Generowanie i wykresy raportów	Jan Szczawiński
Wysyłka mailingów	Jan Szczawiński
Wypełnianie bazy danych	Aleksandra Grabowska
Analiza UX	Renata Pigoń

Poprawa błędów UX	Aleksandra Grabowska Renata Pigoń Justyna Tomaszewska
Budowa aplikacji mobilnej	Łukasz Zajkowski
API dla aplikacji mobilnej	Łukasz Zajkowski
Ekrany aplikacji mobilnej	Łukasz Zajkowski

Zespół testów	
Kierownik	Justyna Tomaszewska
Członkowie zespołu	Aleksandra Grabowska Michał Jabłoński Rafał Pajączkowski
Podział zadań	
Szkolenie Unit testy	Michał Jabłoński
Utworzenie wspólnego konta Postman	Justyna Tomaszewska
Testy automatyczne	Aleksandra Grabowska Michał Jabłoński Justyna Tomaszewska
Testy API	Aleksandra Grabowska Justyna Tomaszewska
Funkcjonalne testy manualne	Aleksandra Grabowska Michał Jabłoński Justyna Tomaszewska

Zespół dokumentacji	
Kierownik	Małgorzata Bieniek
Członkowie zespołu	Marcin Szarek
Podział zadań	
Wstęp	Małgorzata Bieniek
Rozdział 2	Małgorzata Bieniek
Rozdział 3: podrozdziały 3.1 i 3.4	Małgorzata Bieniek
Rozdział 3: podrozdziały 3.2 i 3.3	Marcin Szarek
Rozdział 4: podrozdziały 4.1 - 4.2, 4.4 - 4.6, 4.12-4.13	Małgorzata Bieniek
Rozdział 4: podrozdziały 4.3, 4.10 – 4.11	Marcin Szarek
Rozdział 5: podrozdziały 5.1 – 5.5, 5.11	Małgorzata Bieniek

Rozdział 5: podrozdziały 5.6 – 5.10, 5.12 – 5.13	Marcin Szarek
Rozdział 6	Małgorzata Bieniek
Rozdział 7	Marcin Szarek
Rozdział 8	Marcin Szarek
Podsumowanie	Marcin Szarek

2. Schroniska dla zwierząt w Polsce – kontekst prawny i codzienne funkcjonowanie

W Polsce istnieje prawny obowiązek opieki nad bezdomnymi zwierzętami. Do zajęcia się tym problemem zobowiązuje gminy Ustawa o ochronie zwierząt [2]. Art. 11a formułuje podstawowe zadania gminy w tym zakresie i mówi m.in. o obowiązku:

- zapewnienia bezdomnym zwierzętom miejsca w schronisku,
- odławiania bezdomnych zwierząt,
- obligatoryjnej kastracji i sterylizacji zwierząt w schronisku,
- poszukiwania właścicieli dla zwierząt.

Obowiązki nałożone przez ustawę są realizowane w schroniskach dla zwierząt prowadzonych przez instytucje samorządowe lub organizacje społeczne, których celem statutowym jest opieka nad bezdomnymi zwierzętami. Obecnie pod nadzorem Inspekcji Weterynaryjnej jest ponad 200 schronisk [3], które na zlecenie gmin realizują zapisy wynikające z ustawy.

Przyjmowanie, opieka w trakcie pobytu i wydawanie zwierząt do adopcji stanowią trzon działalności jednostek zajmujących się zwierzętami. Podopieczni są przyprowadzani zarówno przez mieszkańców, jak i przejmowani z innych schronisk. Miejsce trzeba zapewnić również tym, które zostały złapane na zlecenie gminy. W zależności od decyzji władz, to zadanie może być powierzone odpowiednim służbom lub też samemu schronisku i jego pracownikom. W 2018 roku przyjęto do schronisk ponad 70 tys. zwierząt [3]. Przez każde schronisko w ciągu roku przewija się kilkaset lub, w większych schroniskach, nawet kilka tysięcy zwierząt, a średni czas przebywania psa w schronisku to 6 miesięcy [3]. Rejestracja nowego zwierzęcia wymaga zebrania danych nie tylko na jego temat, ale także na temat osoby przyprowadzającej lub okoliczności oddania. Każdy nowy mieszkaniec musi zostać opisany, zostaje mu założony chip, a weterynarz sprawdza stan jego zdrowia. Po przyjęciu zwierzęta podlegają dwutygodniowej kwarantannie. W wielu schroniskach zdjęcie psa lub kota od razu pojawia się na stronie, aby szukający go właściciele mogli odnaleźć swojego pupila. Jeśli po tym czasie nie zgłosi się właściciel, zwierzęta zostają zaszczepione, odrobaczone, wysterylizowane lub wykastrowane i przeniesione do boksów [4] [5]. Od tego momentu są też czynione starania, by nowo przybyły mieszkaniec został jak najszybciej adoptowany.

Dane na temat zwierzęcia jakie muszą być przechowywane są ściśle określone w Rozporządzeniu Ministra Rolnictwa i Rozwoju Wsi z dnia 23 czerwca 2004 r. w sprawie szczegółowych wymagań weterynaryjnych dla prowadzenia schronisk dla zwierząt [6]. Paragraf 6 punkt 1 nakłada obowiązek prowadzenia wykazu, a punkt 2 szczegółowo wymienia informacje, jakie mają być w nim zawarte:

1) opis zwierzęcia, w tym jego gatunek, wiek, płeć, maść i oznakowanie;
2) datę przyjęcia do schroniska oraz imię, nazwisko i adres osoby przekazującej zwierzę do schroniska;

3) dane dotyczące kwarantanny;
4) dane dotyczące przeprowadzonych szczepień i zabiegów weterynaryjnych;
5) datę opuszczenia schroniska oraz imię, nazwisko i adres osoby, której przekazano zwierzę;
6) datę śmierci z podaniem przyczyny.

Większość mieszkańców to psy i koty, choć zdarzają się i inne gatunki np. króliki, świnki morskie. Zakres i rodzaj opieki jest zależny od gatunku. Zwierzęta muszą być czesane, kąpane, a psy trzeba wyprowadzać na spacery. Każde zwierzę potrzebuje kontaktu z człowiekiem i chce się bawić lub choćby być pogłaskane. Personel nie byłby w stanie zapewnić podopiecznym wystarczającej ilości ruchu i kontaktu z ludźmi, dlatego często zajmują się tym wolontariusze. Wiele schronisk korzysta z ich pomocy w sposób stały, a wypełniane przez nich obowiązki są uwzględniane w harmonogramie podobnie jak zadania pracowników.

Nieodzownym elementem pobytu jest opieka weterynaryjna. Nadzór nad schroniskami prowadzi Inspekcja Weterynaryjna, a kwestię zdrowia zwierząt reguluje wspomniana już Ustawa o ochronie zwierząt oraz Ustawa z dnia 11 marca 2004 r. o ochronie zdrowia zwierząt oraz zwalczaniu chorób zakaźnych zwierząt [7]. Zwierzęta są regularnie szczepione i odrobaczane a te, które wymagają leczenia otrzymują leki. Większe schroniska posiadają całe zaplecze medyczne: własne sale operacyjne i diagnostyczne oraz laboratoria [4]. Weterynarze współpracujący ze schroniskiem mogą na podstawie przepisów weterynaryjnych oraz w/w Ustawy o ochronie zwierząt uśpić zwierzę, jeśli zachodzi taka konieczność. Jak już wspomniano do standardowych czynności należy też kastracja i sterylizacja.

Jednym z najistotniejszych zadań jest znalezienie podopiecznym nowego domu. Schronisko ma być tylko chwilowym miejscem pobytu, a celem jest przekazanie zwierzaka nowym właścicielom. Pracownicy aktywnie starają się, by adopcja nastąpiła jak najszybciej. Organizowane są różne akcje, a zdjęcia zwierząt i ich opisy pojawiają się na stronach internetowych oraz w portalach społecznościowych. Aby zabrać kota wystarczy jednorazowa wizyta i odpowiednia klatka, by przewieźć swojego nowego pupila. Proces adopcyjny w przypadku psów może być kilkuetapowy. Nowy właściciel powinien wcześniej poznać psa, wyprowadzić go na spacer, a nawet zabrać do domu, by zobaczyć, jak zachowuje się w nowym otoczeniu. Pracownicy mają obowiązek upewnienia się, czy warunki, w których będzie przebywało zwierzę są odpowiednie, a osoba, która deklaruje chęć zabrania zwierzęcia jest świadoma swych obowiązków. W tym celu są podpisywane odpowiednie umowy, a nierzetelne osoby, które porzucają lub krzywdzą zaadoptowane zwierzęta, są wpisywane na tzw. „czarną listę”. Proces adopcyjny nie kończy się na wydaniu zwierzęcia. Pracownicy mają możliwość przeprowadzenia kontroli poadopcyjnych w nowym domu i w przypadku stwierdzenia nieprawidłowości cofnięcia pozytywnej decyzji o adopcji.

Nowy dom, to nie tylko adopcje. Część zwierząt przebywa w domach tymczasowych. Jest to szczególnie ważne w przypadku tych, które są chore, a ciepłe i spokojne miejsce zapewni im lepsze warunki do rekonwalescencji. Domy tymczasowe są również ratunkiem w czasie zimy oraz dla starszych osobników, które mają małą szansę na znalezienie nowego właściciela na stałe [8], [9]. Takie

rozwiązanie, przydaje się również bardzo młodym zwierzętom. W schronisku nie ma wystarczających warunków by oswoiły się ze stałą obecnością człowieka, więc nawet czasowe schronienie w domu pozwala im odpowiednio się przystosować do życia między ludźmi. W przypadku domów tymczasowych część kosztów utrzymania pokrywa schronisko.

Jeśli ktoś nie ma warunków do zaopiekowania się psem lub kotem możliwa jest również inna pomoc, np. pomoc rzeczowa lub wpłata na konto schroniska. Ciekawym rozwiązaniem jest adopcja wirtualna. Adopcja wirtualna polega na comiesięcznym wpłacaniu pewnej kwoty (zazwyczaj ustalanej przez schronisko) na rzecz konkretnego zwierzęcia. Na stronie internetowej pod zdjęciem zwierzęcia pojawia się imię i nazwisko lub pseudonim osoby adoptującej wirtualnie zwierzę. Zdobyte w ten sposób pieniądze są przeznaczane nie tylko na dane zwierzę, ale na potrzeby całej instytucji [9], [10].

Szeroko zakrojona działalność schroniska wymaga gromadzenia różnorodnych danych nie tylko na potrzeby wewnętrzne, ale i zewnętrzne np. dla Inspekcji Weterynaryjnej. Niezależnie od wspomnianych przepisów prawa prowadzone rejestry są użyteczne dla pracowników schroniska umożliwiając im stały nadzór i aktualną wiedzę o zwierzęciu nawet, gdy zmieniają się jego opiekunowie. Uzupełnienie wymaganych danych o informacje dotyczące bieżącej opieki, przyjmowanych leków lub szczególnych zaleceń znacząco ułatwia pracę, a szybki dostęp do dodatkowych informacji np. na temat temperamentu, lub preferencji adopcji może decydować, jak szybko dany zwierzak znajdzie nowy dom. Uzupełnianie dokumentacji jest jednym ze stałych elementów pracy w schronisku, im szybciej przebiega ten proces, tym więcej czasu pozostaje na właściwą opiekę nad jego mieszkańcami, dlatego stworzenie użytecznej aplikacji stało się celem niniejszej pracy inżynierskiej.

3. Istniejące systemy wspierające zarządzanie schroniskiem dla zwierząt

Na świecie dostępnych jest wiele aplikacji wspierających pracę w schronisku dla zwierząt. Przed przystąpieniem do projektowania aplikacji będącej tematem niniejszej pracy, zespół zapoznał się z ośmioma propozycjami anglojęzycznymi oraz jedną polską. W tym rozdziale zostaną przedstawione trzy propozycje:

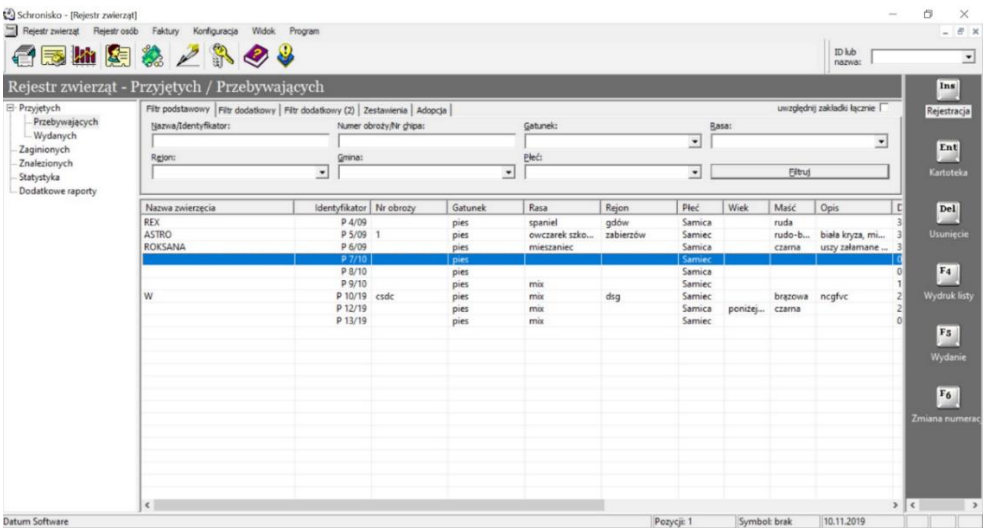
- Schronisko dla zwierząt, program do obsługi schroniska polskiej firmy Datum Software [11],
- Pawlytics, aplikacja mająca na celu wspieranie adopcji zwierząt posiadająca moduły do zarządzania również innymi danymi dotyczącymi zwierząt [12],
- Shelterluv, aplikacja do obsługi schronisk posiadająca szeroki wybór funkcjonalności [13].

Wszystkie opisane aplikacje posiadają wersje demonstracyjne i zostały przetestowane. Pozostałe systemy, z którymi zapoznał się zespół to: RescueConnection Software [14], The Pet Friend [15], Sheltermanager [16], PetPal Manager [17], BARRK [18], Shelter Pro [19].

3.1. Schronisko dla zwierząt

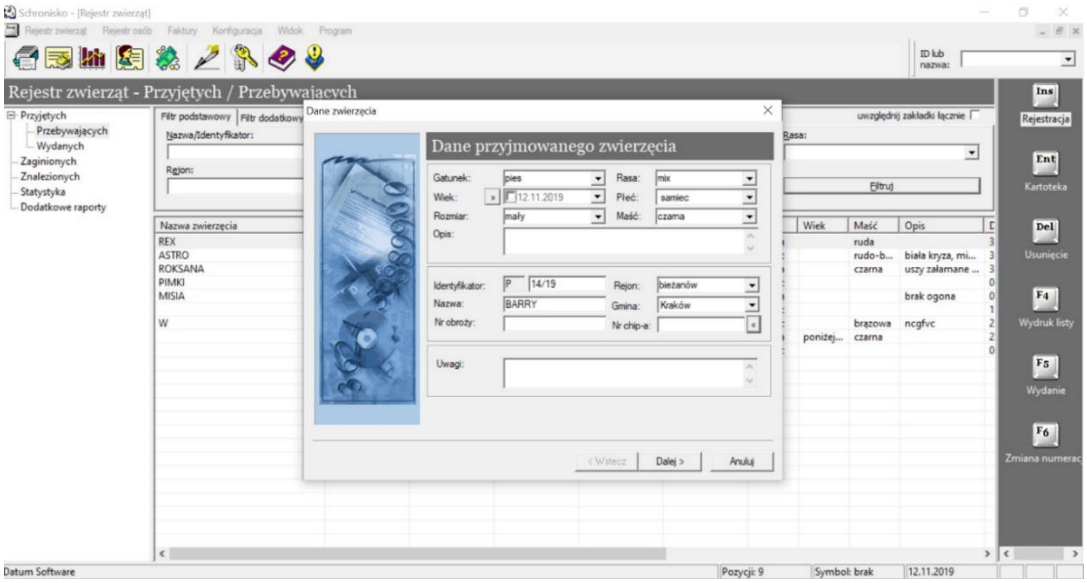
Program Schronisko dla zwierząt [11] został napisany przez krakowską firmę Datum Software. Jest to propozycja programu jedno lub wielostanowiskowego instalowanego bezpośrednio u klienta. Jak podaje producent, jego celem jest wspomaganie prowadzenia schroniska dla zwierząt, poprzez prowadzenie ewidencji zwierząt i szybki dostęp do pełnej informacji o nich [11]. Dostępne moduły to:

1. Rejestr zwierząt.
2. Rejestr osób.
3. Faktury.
4. Edytor szablonów.



Rysunek 2. Rejestr zwierząt przyjętych/przebywających. Źródło: opracowanie własne.

Główne składowe aplikacji to rejestry zwierząt oraz osób. Baza obejmuje dane na temat zwierząt przyjętych i wydanych, dodatkowo możliwa jest także rejestracja zgłoszeń dotyczących zwierząt zaginionych oraz znalezionych. Rysunek 2 pokazuje rejestr zwierząt przyjętych i przebywających w schronisku. W bazie danych gromadzone są podstawowe informacje np. gatunek, rasa, wiek, płeć oraz informacje dotyczące jego pobytu w schronisku np. historia medyczna, zmiana boksów, preferencje adopcji. Program krok po kroku przeprowadza nas przez rejestrację zwierzęcia wyświetlając kolejne okna dialogowe. Przykład okna dialogowego pokazuje Rysunek 3.



Rysunek 3. Rejestracja zwierzęcia: okienko dialogowe. Źródło: opracowanie własne.

W rejestrze osób wyróżniono osoby przyprowadzające i odbierające oraz zgłaszające zaginięcie lub znalezienie zwierzęcia. Możliwe jest także stworzenie czarnej listy osób, którym nie należy wydawać podopiecznych. Tabela 2 przedstawia obszary, w których system wspiera użytkowników wraz z dostępnymi funkcjonalnościami.

Tabela 2. Obszary wspierane przez program Schronisko dla zwierząt. Źródło: Opracowanie własne.

Lp.	**Obszar**	**Opis podstawowych funkcjonalności**
1.	Ewidencja zwierząt	- wyświetlenie kartoteki; - edycja danych, dodanie zdjęcia; - usunięcie zwierzęcia; - wprowadzenie danych dotyczących wydania zwierzęcia; - załączenie dokumentów; - wyszukiwanie z filtrowaniem.
2.	Zarządzanie przyjęciami zwierząt	- wprowadzenie podstawowych danych np. gatunek, rasa, wiek, płeć, rozmiar, maść, nr chipa, nr obroży; - wprowadzenie danych osoby przyprowadzającej;

		- wprowadzenie danych rejestracyjnych np. data, boks, czas kwarantanny, kastracja/sterylizacja.
3.	Historia medyczna	- w karcie zwierzęcia możliwość wpisu dot. szczepień i leczenia.
4.	Zarządzanie adopcją	- w karcie zwierzęcia można uzupełnić dane osoby odbierającej i preferencje adopcji.
5.	Ewidencja osób	- wyświetlanie danych osób przyprowadzających/odbierających zwierzę; - wyświetlanie danych osób zgłaszających zaginięcie/znalezienie zwierzęcia; - czarna lista.
6.	Raporty	- zwierzęta przyjęte do schroniska; - zwierzęta przebywające; -zwierzęta zaszczepione; - zwierzęta wydane; - kontrole poadopcyjne.
7.	Faktury	- wprowadzanie faktur; - lista faktur rozliczonych, nierozliczonych, przeterminowanych.

Program umożliwia łatwą konfigurację niektórych elementów np. numeracji boksów, dostępnych szczepień oraz tworzenie własnych szablonów pism, dostarcza też szeroki wybór filtrów do wyszukiwania.

W aplikacji brak modułów do zarządzania pracownikami i wolontariuszami, nie umożliwia też planowania zadań lub tworzenia harmonogramów np. dotyczących leczenia, szczepień lub spacerów. Jest to przede wszystkim program, który przechowuje dane i pozwala szybko odnaleźć potrzebne informacje. Do wad należy również fakt, że jest dostępny wyłącznie jako aplikacja desktopowa.

3.2. Pawlytics

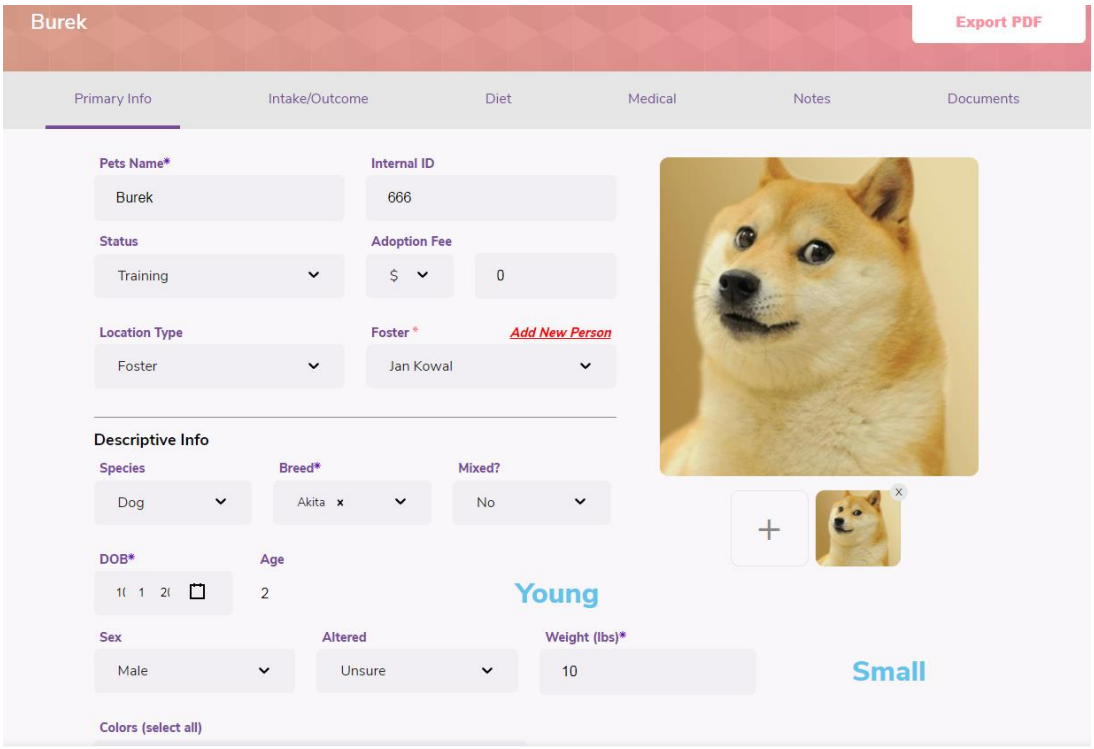
Pawlytics [12] to aplikacja webowa stworzona przez małą amerykańską firmę o tej samej nazwie. Program ten wspiera przede wszystkim zarządzanie adopcjami zwierząt, a nie schroniskiem jako całością. Producent deklaruje, że z pomocą jego oprogramowania organizacje zajmujące się ratowaniem i opieką nad zwierzętami będą w stanie łatwo i w sposób zautomatyzowany przeprowadzać procesy adopcyjne. Aplikacja nie ma stałej ilości modułów, tj. klient wybiera jakie elementy oprogramowania go interesują i tylko za nie płaci. Poniżej wymienione są moduły występujące obecnie w domyślnej wersji:

1. Zarządzanie zwierzętami;
2. Zarządzanie osobami;
3. Aplikacje/wnioski;
4. Użytkownicy - panel administratora.

Wkrótce zaś dostępne będą:

5. Tworzenie kompleksowych raportów;
6. Kalendarz medyczny;
7. Automatyczna integracja ze stroną adoptapet.com;
8. Zarządzanie zadaniami dla każdej adopcji.

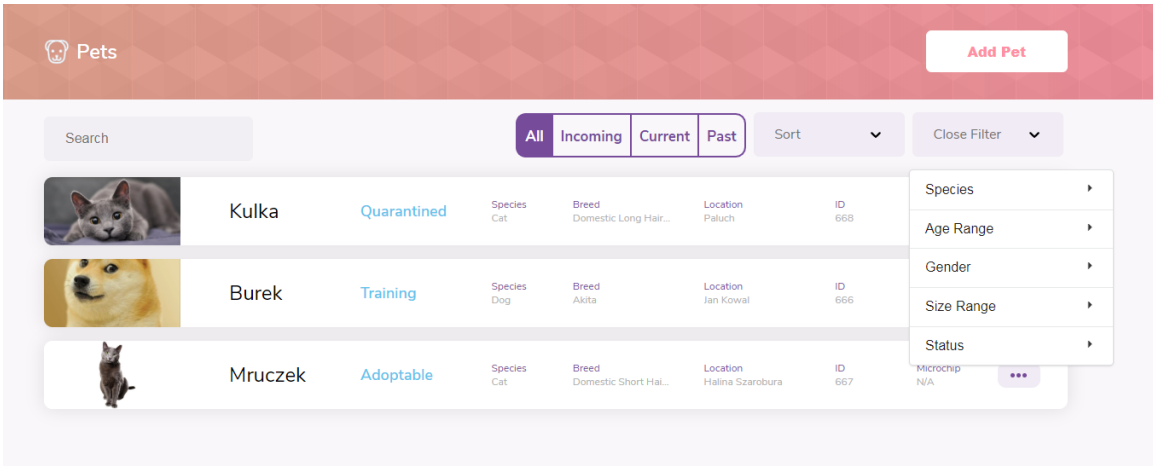
Podstawowym zadaniem programu jest przeprowadzanie procesu adopcyjnego poprzez udostępnienie e-aplikacji na stronie dla osób zainteresowanych oraz zarządzanie adopcjami w formie przechowywania danych osoby i powiązanego zwierzęcia. Dodatkowo oprogramowanie umożliwia tworzenie własnych wzorów podań o adopcję, zgłoszenie się na wolontariat, oddanie zwierzęcia czy stworzenie domu zastępczego. Dane z takich formularzy zapisywane są do bazy danych tworząc profil osoby.



Rysunek 4. Rejestracja zwierzęcia. Źródło: opracowanie własne.

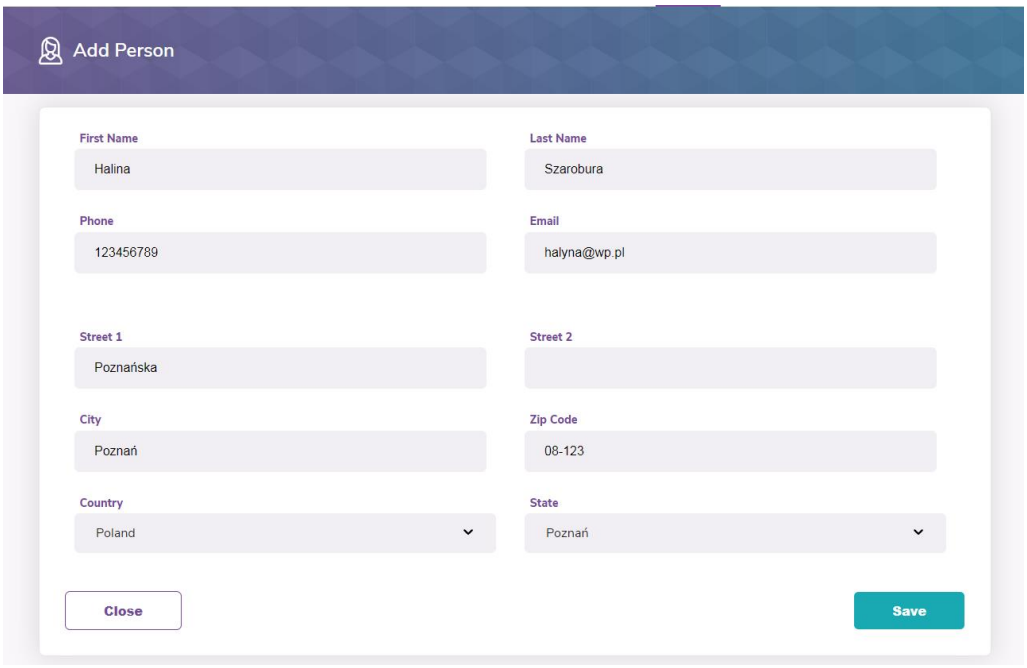
Rysunek 4 pokazuje przykład zakładania konta dla zwierzęcia. Poza podstawowymi danymi można przechowywać informacje o diecie lub historii medycznej, a także szczegółowe dane o cechach charakterystycznych dla danego zwierzęcia (np. czy jest agresywny, czy dobrze reaguje na dzieci lub inne zwierzęta).

Program umożliwia wygodne wyszukiwanie i przeglądanie listy osób oraz zwierząt. Można także sortować i filtrować listę, co zostało pokazane na Rysunku 5.



Rysunek 5. Filtrowanie listy zwierząt. Źródło: opracowanie własne.

Kolejną podstawową funkcjonalnością jest dodanie osoby pokazane na Rysunku 6. Aplikacja przechowuje wyłącznie podstawowe dane teleadresowe bez informacji szczegółowych np. dotyczących preferencji szukanego zwierzęcia lub charakteru miejsca jego ewentualnego przebywania.



Rysunek 6. Rejestracja osoby. Źródło: opracowanie własne.

Tabela 3 przedstawia moduły i elementy występujące w omawianej aplikacji.

Tabela 3. Obszary wspierane przez program Pawlytics. Źródło: Opracowanie własne.

Lp.	**Obszar**	**Opis podstawowych funkcjonalności**
1	Zarządzanie adopcją	- tworzenie własnego wniosku/aplikacji o adopcję;

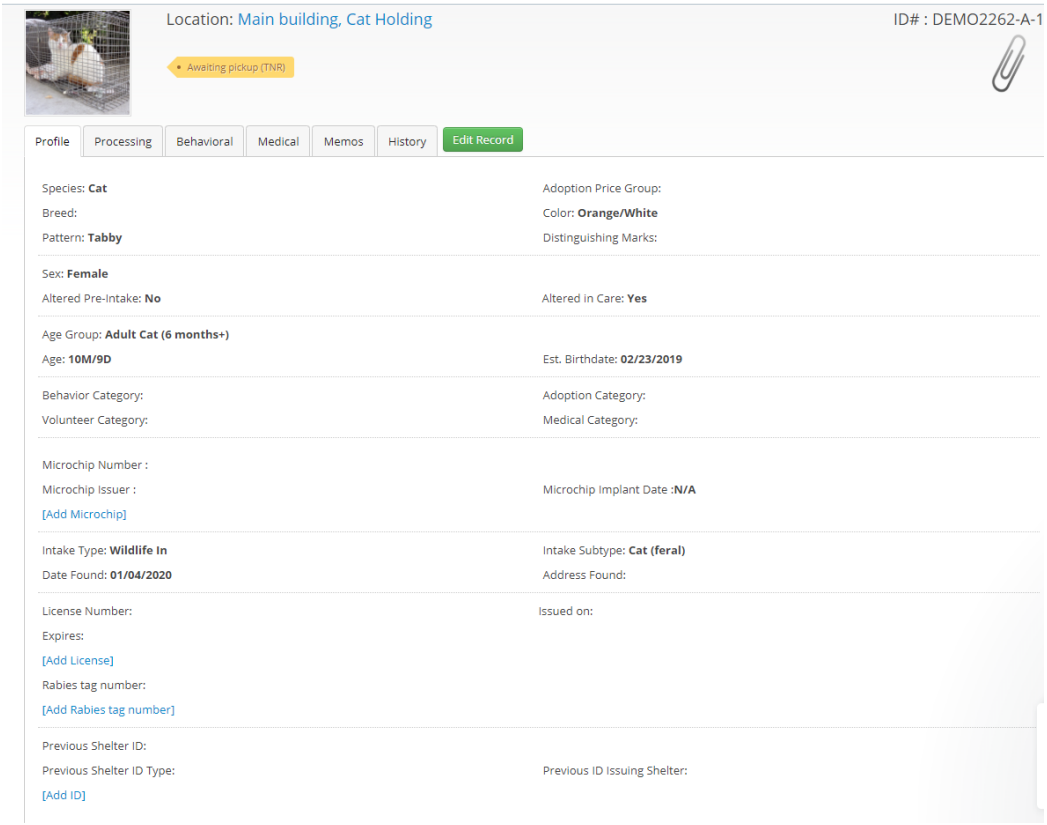
	Uwaga: Dodatkowo płatne rozszerzenie	- tworzenie własnego wniosku/aplikacji o zgłoszenie się do wolontariatu; - tworzenie własnego wniosku/aplikacji o utworzenie domu zastępczego; - tworzenie własnego wniosku/aplikacji o oddanie zwierzęcia.
2	Baza zwierząt	- wyszukiwanie (filtry, sortowanie); - dodanie zwierzęcia do bazy (nazwa, id, status (kwarantanna, uśpiony, do adopcji), gatunek, rasa, wiek, płeć data przyjęcia, kastracja, waga, opis, umaszczenie, link do filmiku youtube, mikrochip, czy może mieszkać z innymi psami, kotami, dziećmi, czy ma specjalne potrzeby, czy potrzebuje doświadczonego właściciela, zdjęcie); - dodanie informacji o diecie; - edycja wszystkich danych; - export pdf z danymi; - dodanie notatki; - dodanie pliku/załącznika.
3	Zarządzanie przyjęciami	- dodanie do bazy; - ustawienie statusu.
4	Zarządzanie historią medyczną	- dodanie procedury medycznej (data, lecznica, koszt, opis) w karcie zwierzęcia
5	Zarządzanie ludźmi Uwaga: Niektóre elementy dodatkowo płatne	- dodanie osoby do bazy; - wyszukiwanie (filtry, sortowanie); - zarządzanie wolontariuszami.

Podsumowując aplikacja firmy Pawlytics ma przejrzysty i nowoczesny layout oraz jest intuicyjna w użyciu. Posiada wygodny sposób tworzenia oraz przeglądania bazy danych zwierząt oraz osób. Ponadto umożliwia tworzenie własnych aplikacji oraz wniosków co zapewnia elastyczność i przyspiesza pracę.

Niemniej program ten ma bardzo okrojone funkcjonalności (kolejne są dopiero na etapie wdrażania) i skupia się przede wszystkim na samym procesie adopcyjnym a nie na zarządzaniu całym schroniskiem jako organizacją. Aplikacja przechowuje ponadto wyłącznie dane teleadresowe osób. Dostępna jest w angielskiej wersji językowej i dostosowana do amerykańskich realiów.

3.3. Schelterluv

Schelterluv [13] to aplikacja webowa umożliwiająca zarządzanie schroniskiem dla zwierząt lub innym podobnym obiektem. Zgodnie z opisem producenta oprogramowanie umożliwia efektywne i kompleksowe kierowanie instytucją tak, by zaoszczędzić jak najwięcej czasu oraz zminimalizować przechowywanie dokumentacji papierowej. Omawiany program ma bardzo szeroki i rozbudowany zakres dostępnych modułów. Wspiera on nie tylko samo prowadzenie schroniska jako domu dla zwierząt, lecz także procesy adopcyjne oraz rozbudowany e-sklep oraz kalendarz zadań.

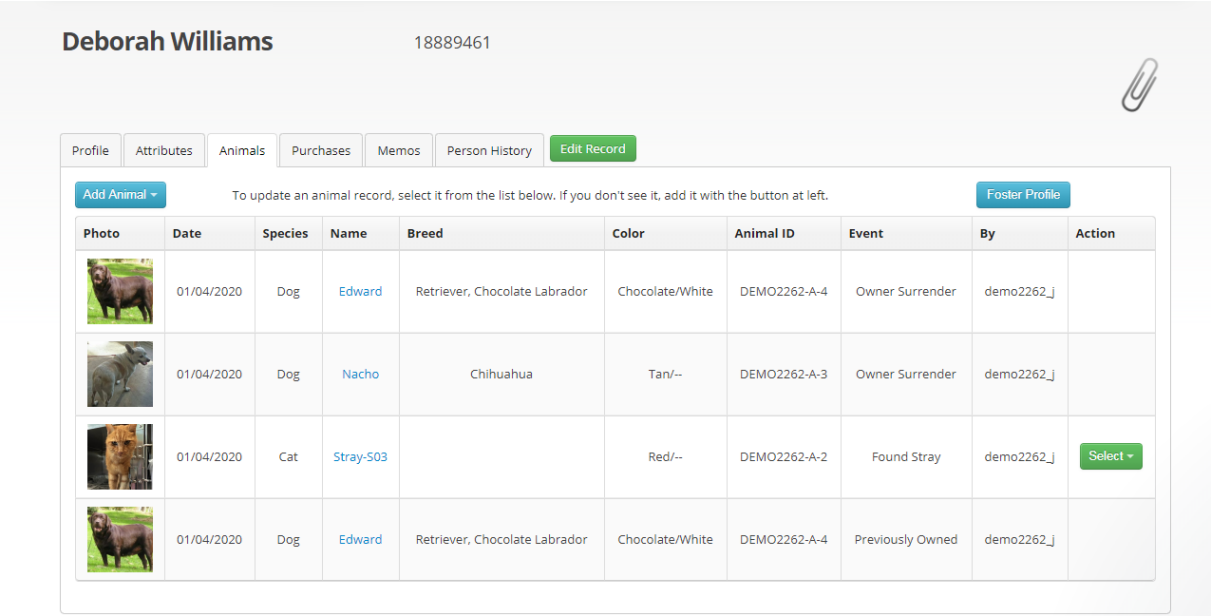


Rysunek 7. Kartoteka zwierzęcia uwzględniająca szeroki zakres danych i gotowych podpowiedzi. Źródło: opracowanie własne.

Pełna lista modułów głównych:

1. Rejestr zwierząt;
2. Rejestr osób;
3. Rejestr organizacji partnerskich;
4. Rejestr transakcji w sklepie;
5. Rejestr użytkowników;
6. Panel konfiguracyjny dla schroniska, medyczny oraz sklepu;
7. Kalendarz zadań;
8. Raportowanie;
9. Samouczek;
10. Zarządzanie adopcjami.

Podstawowym elementem aplikacji jest baza danych zwierząt oraz osób. Profile obu grup są mocno rozbudowane, tak by zawierały bardzo szeroki zakres informacji, np. w przypadku podopiecznych o zachowaniu, historii medycznej, historii pochodzenia; a w profilach osób dane o powiązanych zwierzętach, zakupach, informacjach dodatkowych, ostrzeżeniach i historii adopcji. Kartoteka zwierzęcia została pokazana na Rysunku 7, a jedna z zakładek kartoteki osoby na Rysunku 8.

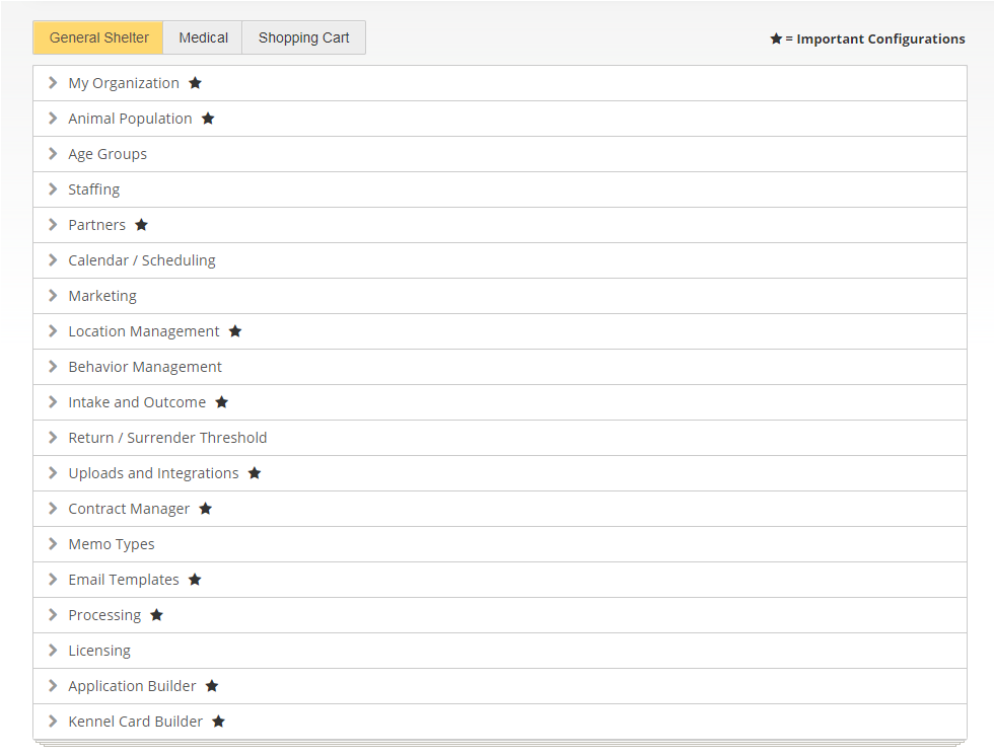


Rysunek 8. Kartoteka osoby, zakładka Animals. Źródło: opracowanie własne.

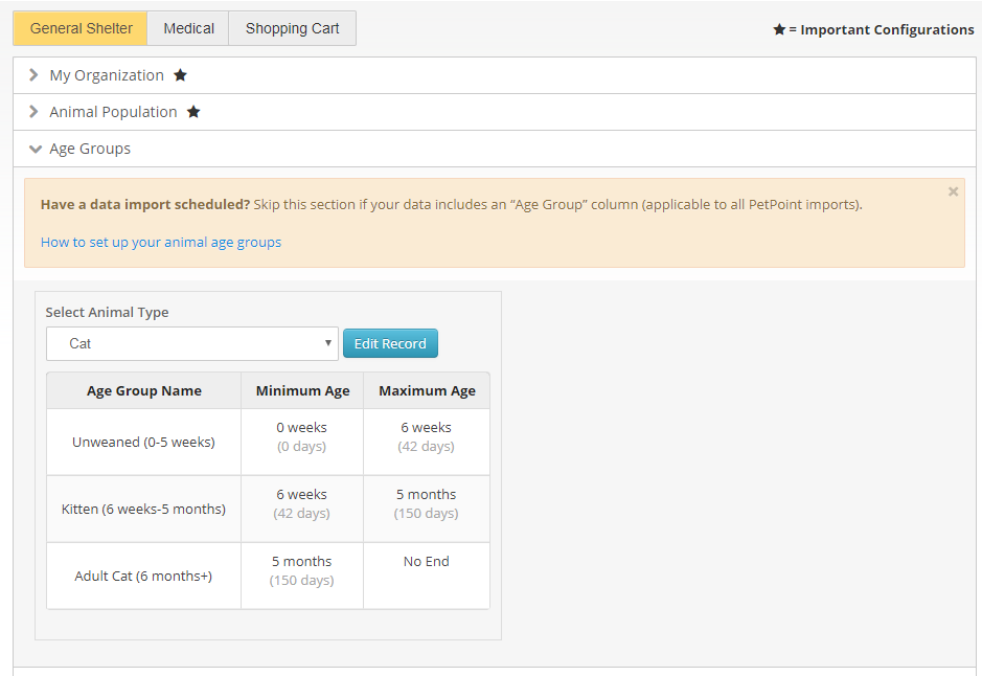
Bardzo rozbudowaną i zarazem niezwykle użyteczną częścią programu jest możliwość konfiguracji ustawień, dzięki czemu można utworzyć:

- gotowe wzorce,
- kategorie,
- opisy dotyczące wszelkich aspektów obejmujących aplikację, jak np. kategoryzacja wieku zwierząt, typów i opisów zachowania;
- wzorce wiadomości mailowych;
- rozróżnienia algorytmów i ścieżek zapytań w zależności od gatunku zwierzęcia;
- zadania;
- opisy medyczne;
- listę produktów udostępnianych w sklepie.

Tworzenie takich gotowych kategorii ułatwia filtrowanie, wyszukiwanie danych czy uzupełnianie formularzy. Listę konfigurowalnych modułów przedstawia Rysunek 9, a przykład konfiguracji modułu, w tym wypadku wieku kota, Rysunek 10.

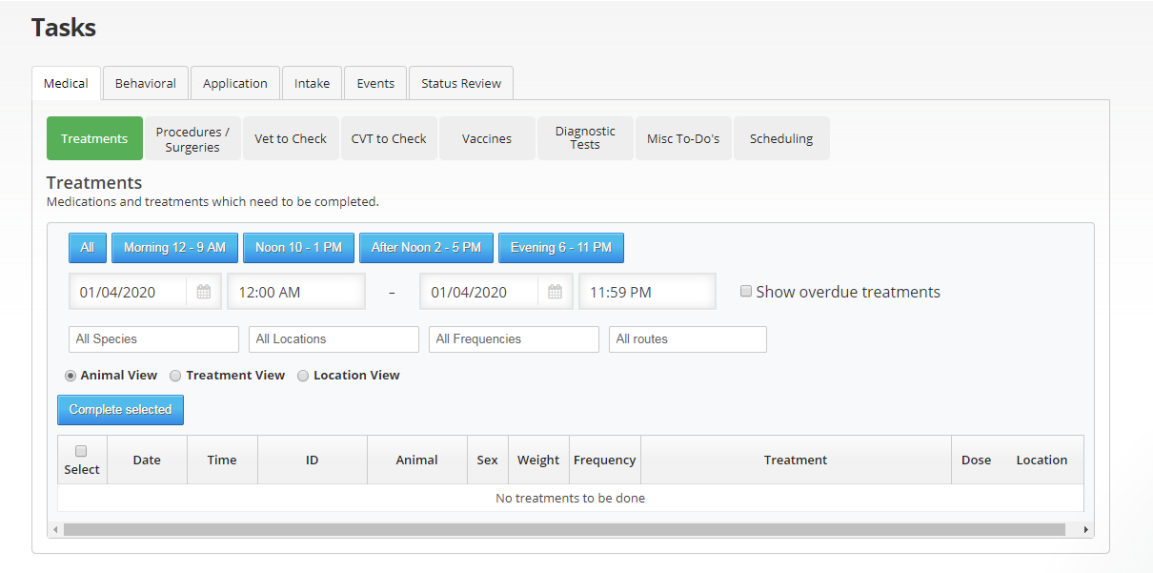


Rysunek 9. Ekran umożliwiający wybór modułu do konfiguracji. Źródło: opracowanie własne.



Rysunek 10. Zakładka umożliwiająca tworzenie wzorców i kategorii na przykładzie kategoryzacji wieku kota. Źródło: opracowanie własne.

Szczegółowo rozbudowano kalendarz zadań oraz zdarzeń. Podzielony jest na kategorie dotyczące zadań medycznych, ćwiczeń, zachowania, adopcji, przyjęć oraz podsumowania. Rysunek 11 pokazuje zakładkę zadań medycznych.



Rysunek 11. Zakładka zadań medycznych. Źródło: opracowanie własne.

Program umożliwia również generowanie bardzo wielu rodzajów raportów ułatwiających zarządzanie adopcjami, sklepem, zwierzętami, a także tworzenie własnych spersonalizowanych raportów. Oddzielny moduł odpowiedzialny jest za przyjmowanie oraz rozpatrywanie adopcji. W programie wyróżnia się e-sklep, w którym nabyć można produkty czy złożyć darowiznę dla konkretnego zwierzęcia. Moduł ten umożliwia płatność za pomocą karty, czeku oraz gotówki; paragon generowany jest automatycznie. Program Shelterluv wyróżnia się szerokim zakresem funkcjonalności. Zostały one zebrane w Tabeli 4.

Tabela 4. Obszary wspierane przez program Shelterluv. Źródło: opracowanie własne.

Lp.	**Obszar**	**Opis podstawowych funkcjonalności**
1	Zarządzanie adopcją	- profil osoby do adopcji/domu tymczasowego: dane personalne, - informacja o statusie: może być domem tymczasowym, ostrzeżenie, nie wydawać zwierząt; - przypisywanie zwierząt do profilu danej osoby; - przypisanie transakcji do osoby; - dodawanie notatek, historia;
2	Kontrola zwierząt	- rejestrowanie problemów z zachowaniem psa i “planu naprawczego”; - rejestrowanie aktywności psa; - rejestrowanie odpchlenia;

		- tworzenie miotów, rejestrowanie spokrewnienia ze sobą psów; - dodawanie notatek;
3	Baza zwierząt	- przechowywanie informacji o zwierzęciu: imię, rasa, id, status itd. - wyświetlanie pełnej informacji; - szczegółowe informacje na temat zachowania psa, czy może mieszkać z innymi zwierzętami, chip, poprzednie schronisko; - rejestrowanie statusów m.in.: możliwy do adopcji, w klinice, czeka na odbiór, ocena zachowania, w trakcie leczenia;
4	Zarządzanie zgłoszeniami/ incydentami	- dodawanie medycznych zleceń do wykonania lub spotkań z behawiorystą; - dodawanie incydentu z aplikacją; - tworzenie eventów;
5	Zarządzanie sponsorami/zbiórkami	- rejestrowanie partnerów;
6	Zarządzanie domami tymczasowymi	- razem z adopcją, ten sam profil;
7	Zarządzanie przyjęciami	- szczegółowe informacje na temat przyjęcia, jeden z powodów: dziko żyjące, zrzeczenie się właściciela, przeniesienie, bezdomny, urodzony w schronisku;
8	Zarządzanie boksami	- rejestrowanie boksów, - rejestrowanie budynków i sekcji.
9	Zarządzanie historią medyczną	- przechowywanie informacji na temat stanu zdrowotnego psa: wiek, kastracja; - dodawanie notatek szczegółowych i instrukcji; - dodawanie diagnozy spośród ogólnie dostępnych(screen); - dodawanie badań- do wykonania, oznaczanie jako wykonane;

		- rejestracja szczepień, obserwacji dziennej, badań fizycznych, leczenia, zabiegów i operacji oraz wizyt weterynarza;
10	Zarządzanie ludźmi	- tworzenie profili osób adoptujących/dających dom tymczasowy; - dostęp do aplikacji każdego z właściwymi uprawnieniami np. możliwość odczytu/edycji;

W ramach podsumowania należy uznać oprogramowanie Shelterluv za kompleksowy oraz rozbudowany system do zarządzania schroniskiem. Aplikacja ma przejrzysty i intuicyjny interfejs. Wyróżnia ją wysoka szczegółowość formularzy oraz pełna możliwość modyfikowania każdego z nich poprzez dodanie i usuwanie elementów, grupowanie, kategoryzowanie co znacznie przyspiesza pracę; ułatwia wyszukiwanie, umożliwia filtrowanie i zapobiega tworzeniu własnych treści przez nieuprawnionego użytkownika. Można także tworzyć własne formatki lub wzorce dokumentów oraz maili. Ponadto wygodny e-sklep ułatwia pozyskiwanie dodatkowych funduszy dla schroniska.

Omawiana aplikacja niestety nie zastępuje programu księgowego, toteż dokonane transakcje należy ręcznie przenosić do działu finansowego instytucji. Ponadto brakuje możliwości pełnego zarządzania wolontariuszami oraz planowania czasu zwierząt np. kontrolowania ilości odbytych spacerów.

3.4. Podsumowanie

Wszystkie przedstawione aplikacje mają na celu ułatwienie zbierania i zarządzania danymi zwierząt oraz innymi informacjami potrzebnymi do opieki nad nimi. Ich podstawowymi modułami są baza zwierząt, podobna w większości aplikacji oraz baza osób. Ta druga w zależności od aplikacji może zawierać dane pracowników, osób przyprowadzających/odbierających zwierzęta, adaptujących i wolontariuszy. Pozostałe moduły znacząco się różnią, a funkcjonalności realizowane są w odmienny sposób. Brak jest propozycji, która łączyłaby w sobie wszystkie użyteczne elementy. Co szczególnie istotne brak jest systemu, który umożliwiałby nie tylko zarządzanie danymi, ale także zarządzanie pracą całej organizacji: koordynowaniem pracy pracowników i wolontariuszy, zintegrowanym planowaniem działań zespołu weterynaryjnego, zarządzaniem zdarzeniami (np. spacer, nietypowe zdarzenia dotyczące zwierząt), kalendarzem. W przetestowanych systemach brak też możliwości wirtualnych adopcji i innych form wsparcia schroniska przez potencjalnych darczyńców. Dlatego aplikacja stworzona w toku pracy naszego zespołu ma wypełnić lukę i połączyć zalety istniejących systemów oraz, co najważniejsze, dostarczyć możliwości, których brakuje w pozostałych propozycjach.

4. Propozycja systemu zarządzania schroniskiem dla zwierząt: @zor

Na powstanie systemu @zor złożyły się fazy analizy problemu, projektowania, implementacji oraz faza testowania. W niniejszym rozdziale został opisany wynik prac faz analitycznej i projektowej. Rozdział rozpoczyna się wymaganiami funkcjonalnymi i niefunkcjonalnymi systemu, następnie zostały zaprezentowane diagramy przypadków użycia, ważniejsze scenariusze i diagramy aktywności, specyficzne dla omawianej problematyki, oraz diagramy stanu. W dalszej części można zapoznać się z etapami powstawania systemu wraz z szczegółowym opisem decyzji podjętych na etapie analizy oraz projektowania. Rozdział kończy opis aplikacji mobilnej i propozycja logo systemu.

4.1. Moduły systemu

System @zor został zaprojektowany jako narzędzie wspierające zarządzanie schroniskiem dla zwierząt. Umożliwia on zróżnicowany dostęp do danych zarówno pracownikom, jak i osobom z zewnątrz zainteresowanym działalnością konkretnej placówki. Powstał jako odpowiedź na wymagania prawne (opisane w rozdziale drugim), potrzeby pracowników wynikające z codziennego funkcjonowania oraz analizę istniejących systemów. Głównym celem jest umożliwienie realizacji podstawowych zadań schroniska, drugim – jak najpełniejszy dostęp do informacji osobom potencjalnie zainteresowanym przygarnięciem zwierzęcia lub wsparciem finansowym, co równocześnie wspiera cel pierwszy, jako że poszukiwanie domów i funduszy również należy do obowiązków regulowanych ustawą. W pierwszym etapie prac analitycznych największy nacisk został położony na zbieranie i dostęp do danych o zwierzętach oraz proces adopcyjny, jako realizacja podstawowych obowiązków prawnych, bez których codzienna działalność schroniska w Polsce nie jest możliwa. Analizując te potrzeby powstały trzy podstawowe moduły systemu:

1. Zarządzanie zwierzętami – obejmuje podstawowe dane dotyczące zwierzęcia, okoliczności jego przyjęcia i pobytu;
2. Zarządzanie potrzebami medycznymi – obejmuje dokumentację medyczną;
3. Zarządzanie adopcją – obejmuje czynności konieczne do wydania zwierzęcia nowemu właścicielowi.

W dalszym etapie analizy pomyślano o rozszerzeniu podstawowych modułów o funkcjonalności niezwiązane z wymogami prawa, ale użytecznych dla pracowników. W wyniku tego etapu prac powstały moduły:

4. Zarządzanie incydentami dotyczącymi zwierząt – uzupełnienie modułu zarządzania zwierzętami o informacje o ponadnormatywnych zdarzeniach w życiu zwierzęcia;
5. Kalendarz – moduł rozszerzający zarządzanie zwierzętami oraz ułatwiający organizację pracy poszczególnym pracownikom;

6. Raporty – moduł umożliwiający generowanie raportów na podstawie danych z trzech podstawowych modułów.
7. Zarządzanie użytkownikami systemu – moduł administracyjny.

Ostatnim elementem analizy było spojrzenie na system z punktu widzenia użytkownika, który trafił na stronę internetową schroniska i dodanie funkcjonalności, które pomagają w dostępie do informacji oraz zachęcają do mocniejszego związania się z konkretnym schroniskiem. Na tym etapie powstały moduły:

8. Zarządzanie sponsorami i wpływami – umożliwia osobom z zewnątrz zostanie sponsorem oraz ułatwia pracownikom schroniska kontrolę nad wsparciem finansowym od darczyńców. Zawiera w sobie podmoduł zarządzania e-adopcją, jako rozwinięcie modułu adopcyjnego, pozwalający na dofinansowanie placówki w powiązaniu z konkretnym zwierzęciem;
9. Ogłoszenia – możliwość dodawania ogłoszeń o zwierzętach zaginionych i szukających nowych właścicieli.

Twórcy systemu starali się, by wyodrębnione dziewięć obszarów w całości wypełniło założenia i spełniło oczekiwania pokładane w aplikacji. W kolejnych podrozdziałach zaprezentowano dokładny opis modułów.

4.2. Wymagania funkcjonalne

Opisane w poprzednim podrozdziale etapy analizy wyłoniły potrzeby użytkowników, które przez twórców systemu zostały przyporządkowane do modułów mających znaleźć się w systemie. W kolejnym etapie prac opis modułów został uszczegółowiony. W konsekwencji powstały wymagania funkcjonalne systemu. Ich opis został przedstawiony w kolejnych podrozdziałach.

4.2.1. Moduły podstawowe

Jak wspomniano powyżej osią systemu są: zarządzanie zwierzętami, zarządzanie potrzebami medycznymi, zarządzanie adopcją. Moduły te są dostępne wyłącznie dla zalogowanych użytkowników. W Tabeli 5 przedstawiono szczegółowe wymagania funkcjonalne dotyczące zarządzania zwierzętami. Moduł ten obsługuje:

- Przyjęcie zwierzęcia do schroniska;
- Dodawanie i modyfikowanie informacji w trakcie pobytu;
- Dostęp do listy zwierząt i szczegółowych informacji na temat poszczególnych zwierząt.

Tabela 5. Wymagania funkcjonalne: zarządzanie zwierzętami. Źródło: opracowanie własne.

Lp.	Nazwa	Opis	Aktorzy	Ograniczenia
1	Dodaj zwierzę	Umożliwi dodanie nowego zwierzęcia wraz podstawowymi	Pracownik Administrator	

		informacjami (gatunek, rasa, płeć, rozmiar, maść, nr chipa, nr obroży, opis, uwagi).		
2	Edytuj dane zwierzęcia	Umożliwia modyfikację danych zapisanych w systemie.	Pracownik Administrator	Zwierzę jest w bazie danych.
3	Usuń zwierzę	Umożliwia usunięcie błędnie dodanego zwierzęcia.	Pracownik Administrator	Zwierzę istnieje w bazie.
4	Ustaw status	Umożliwia ustawienie statusu zwierzęcia np.: do adopcji, czeka na odbiór, w trakcie leczenia.	Pracownik Administrator	Zwierzę istnieje w bazie.
5	Dodaj miejsce przebywania	Umożliwia określenie miejsca przebywania zwierzęcia: numeru boksu, domu tymczasowego.	Pracownik Administrator	Zwierzę istnieje w bazie.
6	Wyświetl listę zwierząt	Umożliwia wyświetlenie listy zwierząt (innej w zależności od uprawnień użytkownika).	Każdy użytkownik	W bazie istnieje przynajmniej jedno zwierzę.
7	Pokaż szczegółowe dane zwierzęcia	Umożliwia wyświetlenie szczegółowych danych na temat zwierzęcia.	Każdy użytkownik	Zwierzę istnieje w bazie.
8	Dodaj osobę przyprowadzającą	Umożliwia dodanie danych osoby, która przyprowadziła zwierzę (imię, nazwisko, PESEL, adres nr telefonu) lub danych instytucji przekazującej zwierzę.	Pracownik Administrator	Przyprowadzone zwierzę istnieje w bazie.
9	Dodaj okoliczności przyjęcia	Umożliwia dodanie szczegółów dotyczących przyjęcia zwierzęcia (bezdomny, zrzeczenie się właściciela, przyprowadzony przez osobę postronną).	Pracownik Administrator	Zwierzę istnieje w bazie.
10	Dodaj wizytę przedadopcyjną	Umożliwia dodanie informacji o wizycie przyszłego właściciela.	Pracownik Administrator	Zwierzę istnieje w bazie.
11	Dodaj multimedia	Umożliwia dodanie zdjęcia i linku do filmu ze zwierzęciem.	Pracownik Administrator	Zwierzę istnieje w bazie.

W Tabeli 6 przedstawiono szczegółowe wymagania funkcjonalne dotyczące zarządzania potrzebami medycznymi. Moduł ten obsługuje:

- Dodawanie i edycję informacji na temat wizyt weterynaryjnych, szczepień i innych aktywności medycznych;
- Wyświetlanie szczegółowych informacji o aktywnościach medycznych;
- Zarządzanie tagami aktywności medycznej.

Tabela 6. Wymagania funkcjonalne: zarządzanie potrzebami medycznymi. Źródło: opracowanie własne.

Lp.	Nazwa	Opis	Aktorzy	Ograniczenia
1	Wyświetlenie listy zwierząt	Umożliwia wyświetlenie listy zwierząt wraz z ogólnym stanem zdrowia (np. zdrowy, wymaga leczenia)	Pracownik Weterynarz	Istnienie przynajmniej jednego zwierzęcia w bazie.
2	Wyświetlenie listy aktywności medycznych zwierzęcia	Umożliwia wyświetlenie listy wszystkich aktywności medycznych zwierzęcia wraz z ich rodzajem (np. wizyta, szczepienie, podanie leku, inny zabieg)	Pracownik Weterynarz	Istnienie przynajmniej jednej aktywności medycznej powiązanej z danym zwierzęciem.
3	Wyświetlenie karty medycznej zwierzęcia	Umożliwia wyświetlenie informacji na temat ogólnego stanu zdrowia.	Pracownik Weterynarz	
4	Wyświetlenie szczegółów aktywności medycznej	Umożliwia wyświetlenie informacji z wykonanej aktywności medycznej.	Pracownik Weterynarz	
5	Rejestracja potrzeby medycznej zwierzęcia	Umożliwia zarejestrowanie wizyty u weterynarza z podaniem przyczyny.	Pracownik Weterynarz	
6	Dodanie aktywności medycznej	Umożliwia dodanie aktywności medycznej (np. wizyta u weterynarza, szczepienie, podanie leków) wraz z datą i opisem.	Weterynarz	

7	Edycja aktywności medycznej	Umożliwia modyfikację aktywności medycznej.	Weterynarz	Istnienie aktywności medycznej.
8	Usunięcie aktywności medycznej	Umożliwia usunięcie błędnie aktywności medycznej.	Weterynarz	Istnienie aktywności medycznej.
9	Dodanie tagu aktywności medycznej	Umożliwia dodanie tagu aktywności medycznej w zależności od potrzeb schroniska.	Pracownik uprawniony	
10	Edycja tagu aktywności medycznej	Umożliwia modyfikację tagu aktywności medycznej.	Pracownik uprawniony	Istnienie tagu w bazie.
11	Usunięcie tagu aktywności medycznej	Umożliwia usunięcie błędnie dodanego tagu aktywności medycznej.	Pracownik uprawniony	Istnienie tagu w bazie.

W Tabeli 7 przedstawiono szczegółowe wymagania funkcjonalne dotyczące zarządzania adopcją. Moduł ten obsługuje:

- Czynności wstępne konieczne do wykonania przez osobę zainteresowaną adopcją;
- Zarządzanie procesem adopcyjnym przez pracownika.

Tabela 7. Wymagania funkcjonalne: zrządzanie adopcją. Źródło: opracowanie własne.

Lp.	**Nazwa**	**Opis**	**Aktorzy**	**Ograniczenia**
1	Utworzenie profilu osoby zainteresowanej adopcją	Umożliwia stworzenie konta w systemie.	Gość	
2	Uzupełnienie formularza adopcyjnego	Umożliwia wypełnienie formularza adopcyjnego w systemie.	Osoba adoptująca	Użytkownik musi mieć założone konto i być zalogowany.
3	Dodanie preferencji adopcji osoby zainteresowanej	Umożliwia dodanie preferencji adopcji – oczekiwanych cech psa (np. lubi dzieci, może mieszkać z innymi zwierzętami).	Osoba adoptująca.	Użytkownik musi być zalogowany.

4	Utworzenie wniosku adopcyjnego	Umożliwia utworzenie wniosku adopcyjnego.	Pracownik	Osoba adoptująca musi istnieć w bazie.
5	Edycja wniosku adopcyjnego	Umożliwia modyfikację utworzonego wniosku adopcyjnego.	Pracownik	Wniosek musi istnieć w bazie.
6	Zmiana statusu adopcji	Umożliwia zmianę statusu adopcji (np. w trakcie, zakończona)	Pracownik	Wniosek adopcyjny musi istnieć w bazie.
7	Dodanie zwierzęcia do profilu osoby adoptującej	Umożliwia przypisanie zwierzęcia, które ma zostać adoptowane do profilu osoby adoptującej.	Pracownik	Osoba musi istnieć w bazie, zwierzę musi istnieć w bazie.
8	Wydruk wniosku adopcyjnego.	Umożliwia wydruk formularza adopcyjnego.	Pracownik	Wniosek musi istnieć w bazie.

4.2.2. Moduły uzupełniające

Moduły uzupełniające stanowią poszerzenie podstawowych funkcjonalności systemu. Należą do nich: zarządzanie incydentami, kalendarz, raporty, zarządzanie użytkownikami. Moduły te są dostępne wyłącznie dla zalogowanych pracowników schroniska.

W Tabeli 8 przedstawiono szczegółowe wymagania funkcjonalne dotyczące obsługi incydentów. Element ten jest rozszerzeniem modułu zarządzania zwierzętami i dotyczy informacji o ponadnormatywnych zdarzeniach w schronisku powiązanych z konkretnym zwierzęciem np. pogryzienie przez innego psa, agresywne zachowanie itp.

Tabela 8. Wymagania funkcjonalne: zarządzanie incydentami. Źródło: opracowanie własne.

Lp.	**Nazwa**	**Opis**	**Aktorzy**	**Ograniczenia**
1	Wyświetlenie listy incydentów	Umożliwia wyświetlenie listy incydentów w schronisku.	Pracownik	
2	Utworzenie incydentu	Umożliwia utworzenie incydentu powiązanego z konkretnym zwierzęciem.	Pracownik Weterynarz	
3	Edycja incydentu	Umożliwia modyfikację informacji dotyczących incydentu.	Pracownik Pracownik uprawniony	

4	Usunięcie incydentu	Umożliwia usunięcie błędnie utworzonego incydentu.	Pracownik Pracownik uprawniony	

W Tabeli 9 przedstawiono szczegółowe wymagania funkcjonalne dotyczące kalendarza. Kalendarz ma ułatwić pracownikom organizację pracy poprzez dodawanie zadań i zdarzeń np. wizyt weterynaryjnych, spacerów. Moduł ten obsługuje kalendarz pracownika oraz zwierzęcia.

Tabela 9. Wymagania funkcjonalne: kalendarz. Źródło: opracowanie własne.

Lp.	**Nazwa**	**Opis**	**Aktorzy**	**Ograniczenia**
1	Wyświetlenie kalendarza	Umożliwia wyświetlenie kalendarza wraz z wpisanymi do niego zdarzeniami.	Pracownik Weterynarz Wolontariusz	
2	Dodanie wizyty weterynaryjnej	Umożliwia dodanie wizyty weterynaryjnej w kalendarzu zwierzęcia lub pracownika.	Weterynarz Pracownik Wolontariusz	Istnienie zwierzęcia lub pracownika w bazie.
3	Usunięcie wizyty weterynaryjnej	Umożliwia usunięcie wizyty weterynaryjnej z kalendarza.	Weterynarz Pracownik Wolontariusz	Istnienie wizyty weterynaryjnej w bazie.
4	Dodanie wizyty adopcyjnej	Umożliwia dodanie wizyty adopcyjnej w kalendarzu zwierzęcia lub pracownika.	Pracownik Wolontariusz	Istnienie zwierzęcia lub pracownika w bazie.
5	Usunięcie wizyty adopcyjnej	Umożliwia usunięcie wizyty adopcyjnej z kalendarza.	Weterynarz Pracownik Wolontariusz	Istnienie wizyty adopcyjnej w bazie.
6	Dodanie incydentu	Umożliwia dodanie incydentu do kalendarza zwierzęcia.	Weterynarz Pracownik Wolontariusz	Istnienie zwierzęcia w bazie.
7	Usunięcie incydentu	Umożliwia usunięcie incydentu z kalendarza zwierzęcia.	Weterynarz Pracownik Wolontariusz	Istnienie incydentu w bazie.
8	Dodanie spaceru	Umożliwia dodanie konieczności wyjścia na spacer w kalendarzu zwierzęcia.	Pracownik Wolontariusz	Istnienie zwierzęcia w bazie.

9	Usunięcie spaceru	Umożliwia usunięcie spaceru w kalendarzu zwierzęcia.	Pracownik Wolontariusz	Istnienie spaceru w bazie.
10	Dodanie powiadomienia	Umożliwia ustawienie powiadomienia o wybranym wydarzeniu z kalendarza.	Weterynarz Pracownik Wolontariusz	Istnienie zdarzenia w bazie.
11	Usunięcie powiadomienia.	Umożliwia usunięcie powiadomienia o wybranym wydarzeniu.	Weterynarz Pracownik Wolontariusz	Istnienie powiadomienia w bazie.

W Tabeli 10 przedstawiono szczegółowe wymagania funkcjonalne dotyczące raportów. Moduł ten obsługuje pracę z wcześniej zdefiniowanymi raportami, które powstają na podstawie danych znajdujących się w systemie oraz stworzenie nowych typów raportów:

Tabela 10. Wymagania funkcjonalne: zarządzanie raportami. Źródło: opracowanie własne.

Lp.	**Nazwa**	**Opis**	**Aktorzy**	**Ograniczenia**
1	Stworzenie raportu	Umożliwia stworzenie wcześniej zdefiniowanego raportu, na podstawie danych znajdujących się w bazie	Pracownik	
2	Edycja raportu	Umożliwia modyfikację istniejącego raportu.	Pracownik	Istnienie raportu w bazie.
3	Podgląd raportu	Umożliwia wyświetlenie wcześniej stworzonego raportu.	Pracownik	Istnienie raportu w bazie.
4	Usunięcie raportu	Umożliwia usunięcie wcześniej stworzonego raportu.	Pracownik	Istnienie raportu w bazie.
5	Dodanie raportu nowego typu	Umożliwia zdefiniowanie nowego typu raportu.	Super-administrator	
6	Usunięcie definicji raportu	Umożliwia usunięcie raportu danego typu.	Super-administrator	Istnienie definicji raportu w bazie.
7	Edycja definicji raportu	Umożliwia edycję raportu danego typu.	Super-administrator	Istnienie definicji raportu w bazie.

W Tabeli 11 przedstawiono szczegółowe wymagania funkcjonalne dotyczące zarządzania użytkownikami. Moduł ten obsługuje:

- Dodawanie i edycję informacji na temat użytkowników;

- Wyświetlanie danych na temat użytkownika, w tym przypisanych mu zwierząt;
- Dodawanie zadań do kalendarza pracowników.

Tabela 11. Wymagania funkcjonalne: zarządzanie użytkownikami. Źródło opracowanie własne.

Lp.	Nazwa	Opis	Aktorzy	Ograniczenia
1	Dodanie użytkownika	Umożliwia dodanie danych o użytkowniku.	Pracownik	
2	Wyświetlenie użytkowników systemu	Umożliwia wyświetlenie listy osób istniejących w bazie.	Pracownik	Istnienie przynajmniej jednego użytkownika w bazie.
3	Edycja użytkownika	Umożliwia modyfikację danych o użytkowniku.	Pracownik	Istnienie użytkownika w bazie.
4	Odczyt użytkownika	Umożliwia wyświetlenie danych użytkownika.	Pracownik	Istnienie użytkownika w bazie.
5	Wyświetlenie listy przypisanych zwierząt	Umożliwia wyświetlenie listy zwierząt, które dany użytkownik ma pod opieką.	Pracownik	
6	Dodanie zadania	Umożliwia dodanie zadania do kalendarza wybranemu użytkownikowi (pracownikowi, wolontariuszowi, weterynarzowi).	Pracownik	Istnienie użytkownika w bazie.

4.2.3. Moduły wspierające zarządzanie interakcją z użytkownikami zewnętrznymi

Dwa ostatnie moduły mają za zadanie umożliwić interakcję z użytkownikami niebędącymi pracownikami schroniska. W Tabeli 12 przedstawiono szczegółowe wymagania funkcjonalne dotyczące zarządzania sponsorami i wpływami. Moduł ten obsługuje:

- Zarządzanie danymi sponsorów;
- Zarządzanie wpływami;
- Wirtualną adopcję zwierzęcia.

Tabela 12. Wymagania funkcjonalne: zarządzanie sponsorami i wpływami. Źródło: opracowanie własne.

Lp.	Nazwa	Opis	Aktorzy	Ograniczenia
1	Wyświetlenie listy sponsorów	Umożliwia wyświetlenie listy sponsorów znajdujących się w systemie.	Pracownik Gość	Istnienie przynajmniej jednego sponsora w systemie.
2	Utworzenie sponsora	Umożliwia dodanie sponsora do bazy danych.	Pracownik uprawniony	
3	Edycja sponsora	Umożliwia modyfikację danych sponsora.	Pracownik uprawniony	Istnienie sponsora w bazie.
4	Usunięcie sponsora	Umożliwia usunięcie sponsora z bazy danych.	Pracownik uprawniony	Istnienie sponsora w bazie.
5	Wyświetlenie listy darowizn	Umożliwia wyświetlenie listy darowizn wraz z nazwą sponsora i kwotą.	Pracownik uprawniony	Istnienie przynajmniej jednej darowizny w bazie.
6	Dodanie darowizny	Umożliwi dodanie darowizny.	Adoptujący wirtualnie	
7	Wirtualna adopcja	Umożliwia wybranie zwierzęcia, które chciałby wspierać użytkownik.	Adoptujący wirtualnie	Użytkownik ma konto w bazie i jest zalogowany.
8	Deklaracja wpłaty	Umożliwia wpisanie kwoty jaką użytkownik chciałby wpłacać.	Adoptujący wirtualnie	Użytkownik wybrał zwierzę do wirtualnej adopcji.
9	Modyfikacja wpłaty	Umożliwia zmianę kwoty, jaką użytkownik chciałby wpłacać.	Adoptujący wirtualnie	Kwota została wcześniej zadeklarowana.

W Tabeli 13 przedstawiono szczegółowe wymagania funkcjonalne dotyczące ogłoszeń. Moduł ten obsługuje zarządzanie ogłoszeniami zarówno ze strony pracownika jak i gościa strony.

Tabela 13. Wymagania funkcjonalne: ogłoszenia. Źródło: opracowanie własne.

Lp.	Nazwa	Opis	Aktorzy	Ograniczenia
1	Dodanie ogłoszenia	Umożliwia umieszczenie ogłoszenia na stronie internetowej schroniska.	Pracownik Gość	
2	Edycja ogłoszenia	Umożliwia zmianę treści ogłoszenia na stronie schroniska.	Pracownik Gość	Istnienie ogłoszenia w bazie.
3	Usunięcie ogłoszenia	Umożliwia usunięcie ogłoszenia ze strony schroniska.	Pracownik Gość	Istnienie ogłoszenia na stronie.

4.3. Wymagania niefunkcjonalne.

Wymagania niefunkcjonalne aplikacji opisują obszar jakościowy systemu. Wynikają z założeń funkcjonalnych oraz architektonicznych. Źródłem wymagań niefunkcjonalnych są klienci, użytkownicy oraz osoby odpowiedzialne za integrację i utrzymanie aplikacji. Wymagania niefunkcjonalne dotyczą ograniczenia usług i funkcji tworzonego rozwiązania. Do ich lepszego opisu i późniejszej weryfikacji konieczne jest użycie precyzyjnych miar, które określą, kiedy dane wymaganie zostało spełnione. W kolejnych podrozdziałach zaprezentowane są wymagania naszego systemu.

4.3.1. Wymagania efektywności

- czas reakcji systemu na akcję użytkownika – poniżej 2 sekund;
- maksymalna liczba użytkowników jednocześnie używających systemu: 10 000;
- system powinien być dostępny na większości obecnie dostępnych przeglądarek internetowych tj. Google Chrome, Mozilla Firefox, Opera.

4.3.2. Wymagania niezawodności

- system dostępny dla użytkowników przez 99% czasu;
- średni czas bez awarii 60 dni;
- czas reakcji w przypadku awarii – poniżej 3 godzin;
- w przypadku aktualizacji systemu na stronie powinna znaleźć się informacja o planowanej przerwie w dostępności systemu co najmniej na 48 godzin przed planowaną aktualizacją;
- konserwacja systemu powinna być przeprowadzona w godzinach nocnych, pomiędzy 00:00 a 5:00;
- czas niedostępności usługi w przypadku aktualizacji systemu: do 10 minut.

4.3.3. Wymagania tworzenia kopii zapasowych

- kopie zapasowe danych powinny być tworzone nie rzadziej niż co 30 dni.

4.3.4. Wymagania zajętości pamięci

- w przypadku aplikacji webowej, system nie obciąża pamięci dysku twardego – wykorzystywana jest pamięć RAM w zależności od wykorzystywanej przeglądarki internetowej;

4.3.5. Wymagania organizacji procesu wytwórczej

- archiwizacja wersji systemu przy użyciu repozytorium plików np. GitLab;
- system napisany w języku programowania Java oraz JavaScript, baza danych mySQL;
- metoda wytwórcza oprogramowania z wykorzystaniem modelu kaskadowego z iteracjami.

4.3.6. Wymagania przenoszalności systemu

- system może być zainstalowany na serwerze z dowolnym systemem operacyjnym (Windows, Linux , MacOS).

4.3.7. Wymagania obsługi systemu

- system łatwy w obsłudze – czas szkolenia obsługi systemu poniżej 3 godzin;
- system powinien być responsywny i odpowiednio się skalować w zależności od rozmiaru okna.

4.3.8. Wymagania etyczne

- system nie wykorzystuje danych osobowych w celach komercyjnych – tylko do celu procesów administracyjnych i organizacyjnych.

4.3.9. Wymagania bezpieczeństwa

- system zapewnia dostęp do niejawnych danych po uprzedniej autentykacji i autoryzacji użytkownika. Dane autoryzacyjne przechowywane są w bazie danych w zaszyfrowanej formie.

4.4. Aktorzy systemu

System @zor jest przeznaczony zarówno dla pracowników schroniska, jak i innych osób zainteresowanych działalnością danej placówki. Dlatego aktorów systemu można podzielić na dwie grupy. Do grupy pracowników należą:

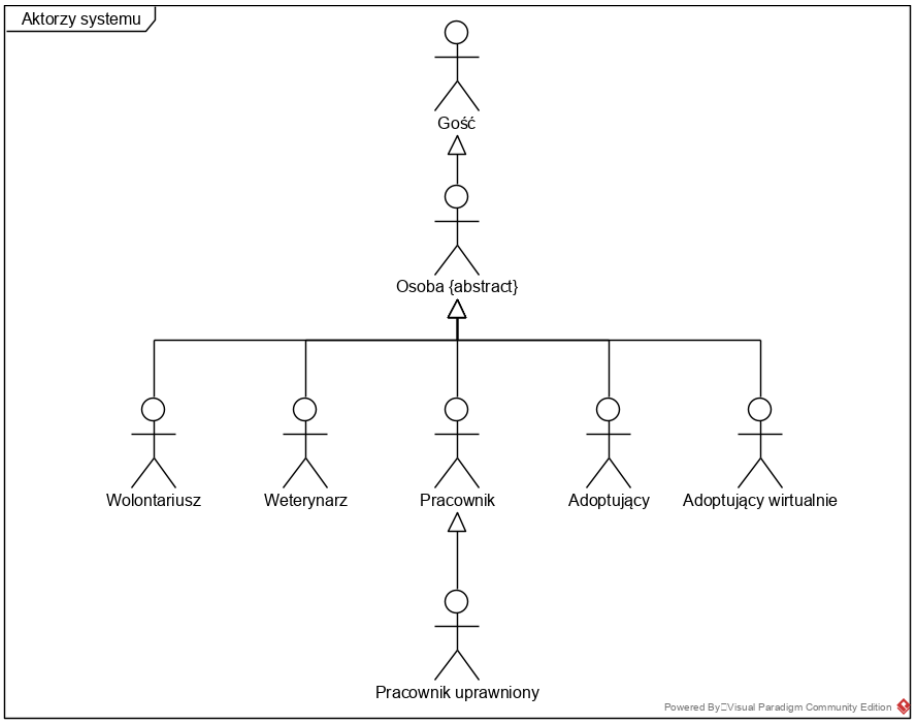
– pracownik,
– pracownik uprawniony (administrator),
– wolontariusz,
– weterynarz.

Pracownicy mają dostęp do modułów odpowiadających ich zakresowi obowiązków, umożliwiających wprowadzanie i edycje danych w systemie.

Grupa pozostałych użytkowników to:

– gość,
– sponsor,
– adoptujący,
– adoptujący wirtualnie.

Pozostali użytkownicy mogą przeglądać udostępnione im treści oraz wprowadzać dane bezpośrednio dotyczące ich osoby. Dokładny opis uprawnień znajduj się w podrozdziale dotyczącym przypadków użycia. Schemat aktorów systemu przedstawia Rysunek 12.



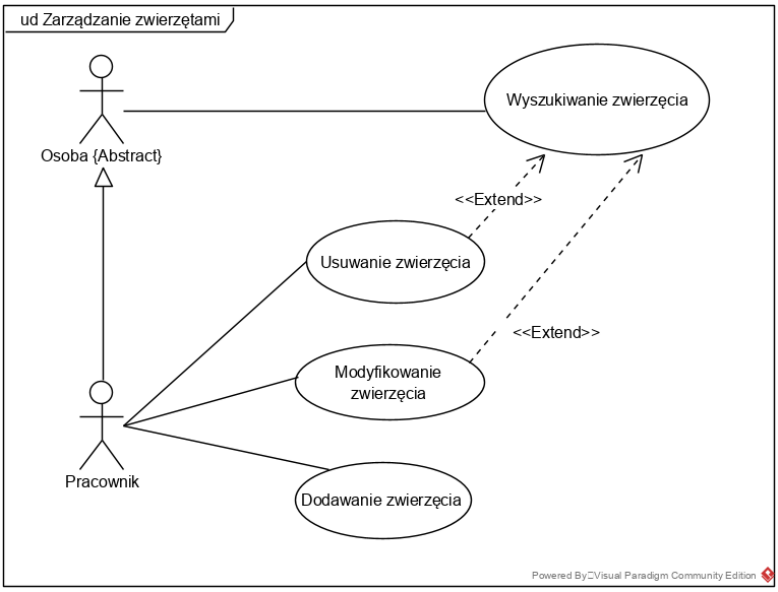
Rysunek 12. Aktorzy systemu – schemat dziedziczenia. Źródło: opracowanie własne.

Została także uwzględniona rola superadministratora, który nie jest związany z żadnym schroniskiem. Zarządza on systemem jako całością.

4.5. Przypadki użycia

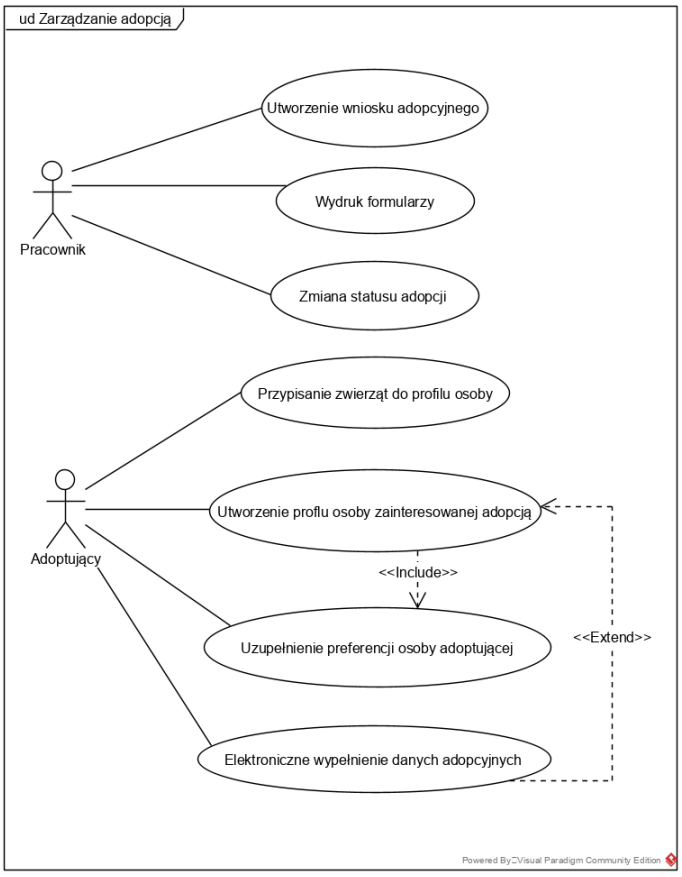
Poniżej znajdują się diagramy przypadków użycia – osobny dla każdego modułu.

Moduł zarządzanie zwierzętami przedstawia Rysunek 13.



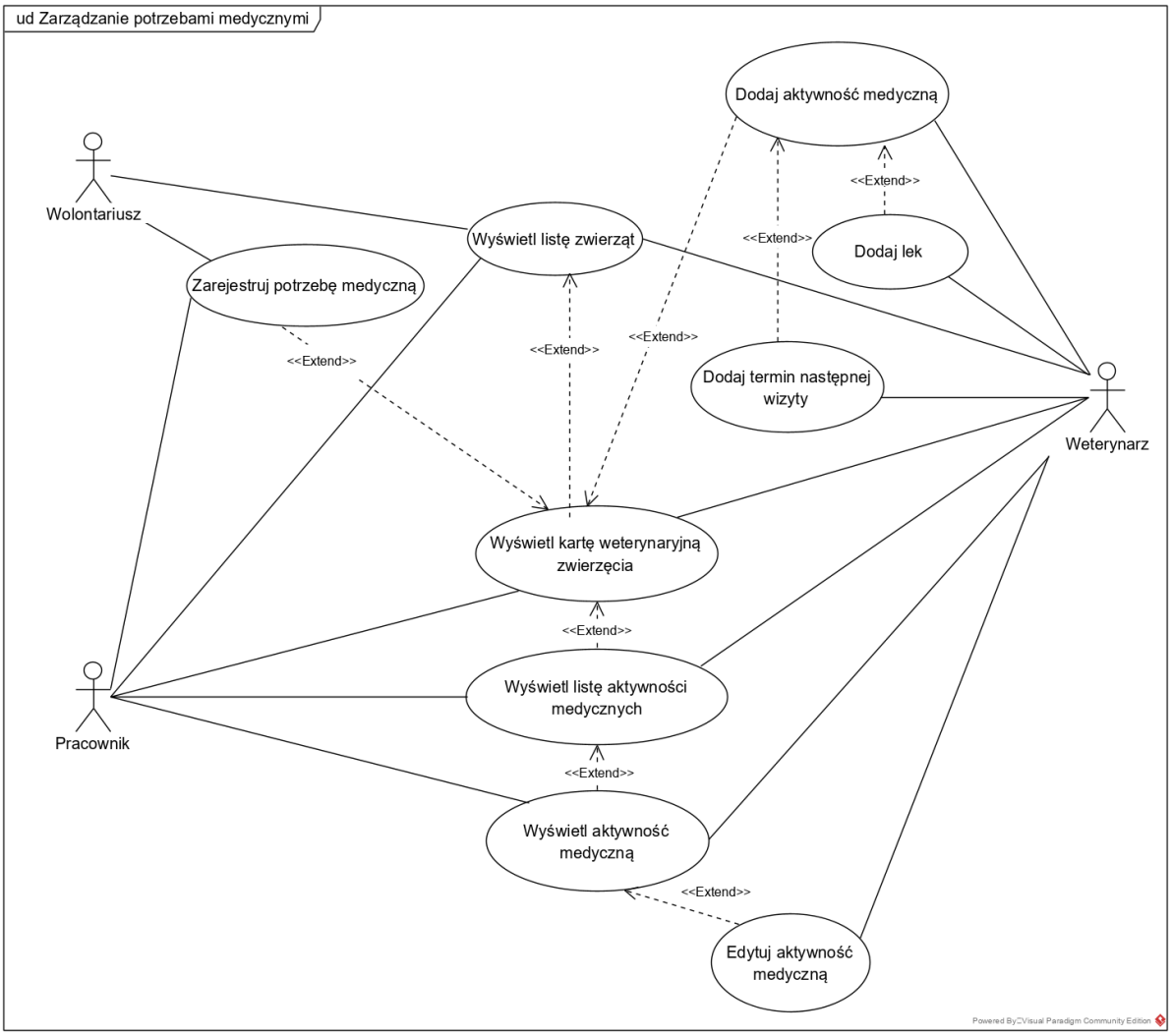
Rysunek 13. Diagram przypadków użycia: zarządzanie zwierzętami. Źródło: opracowanie własne.

Moduł zarządzanie adopcją przedstawia Rysunek 14.



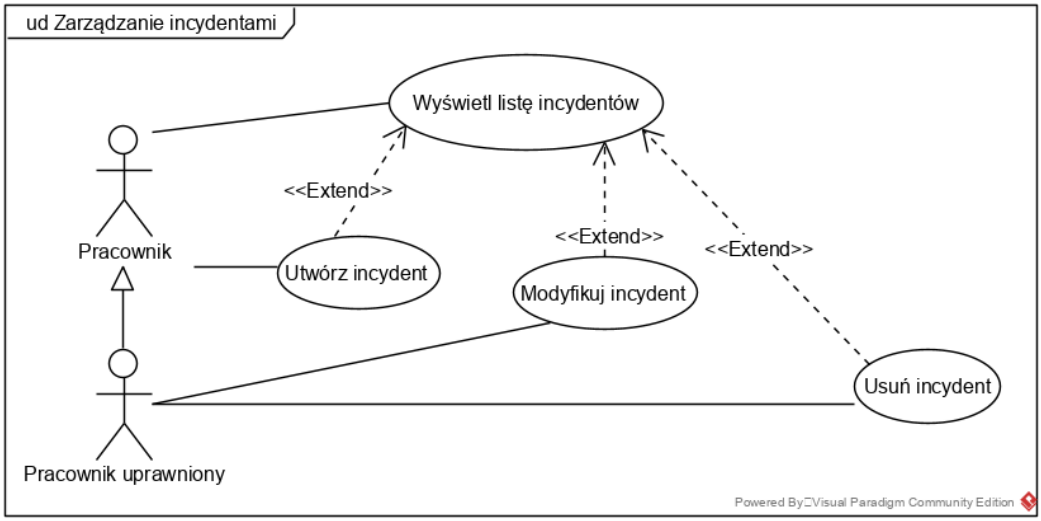
Rysunek 14. Diagram przypadków użycia: zarządzanie adopcją. Źródło: opracowanie własne.

Moduł zarządzanie potrzebami medycznymi przedstawia Rysunek 15.



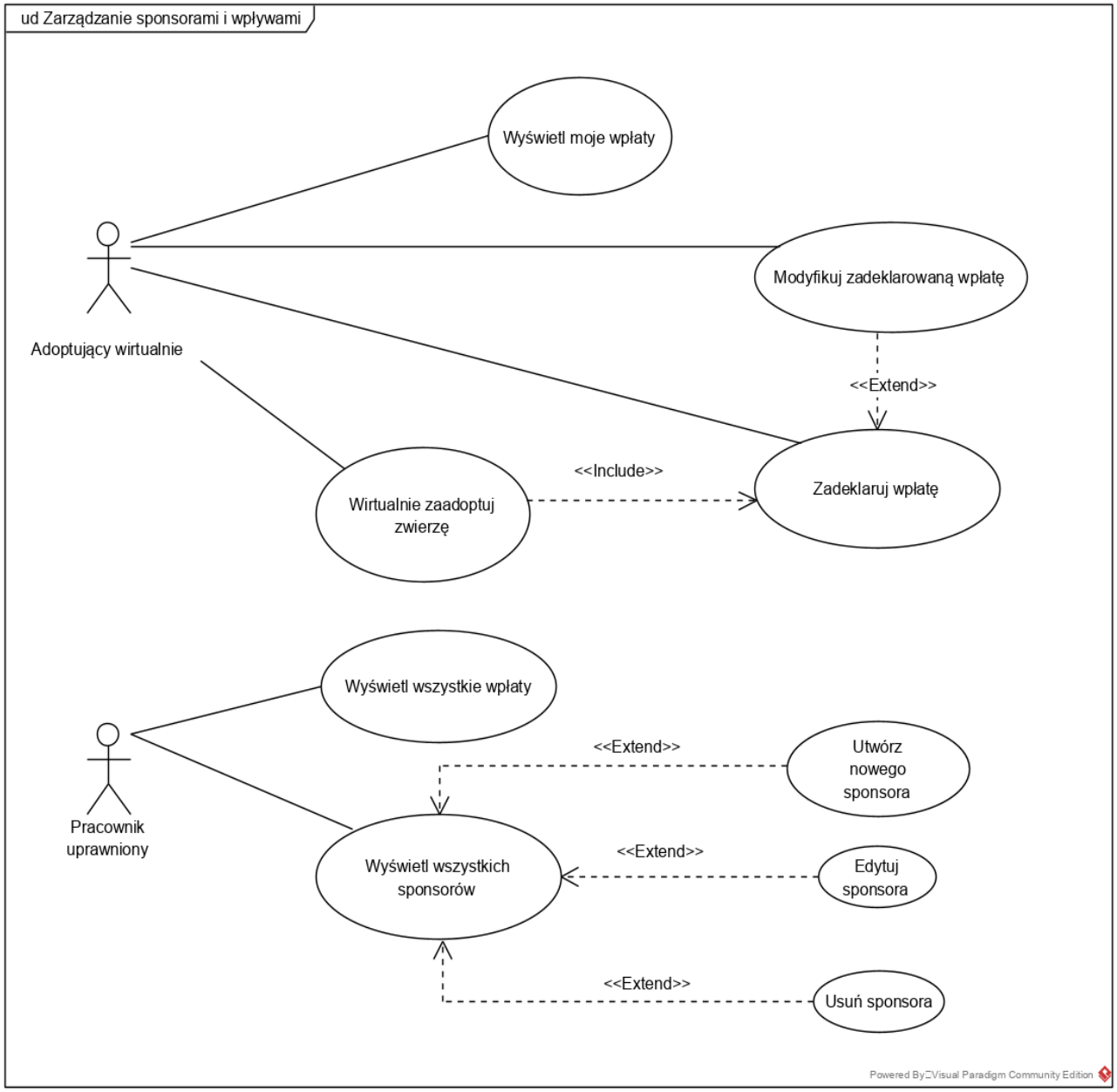
Rysunek 15. Diagram przypadków użycia: zarządzanie potrzebami medycznymi. Źródło: opracowanie własne.

Moduł zarządzania incydentami przedstawia Rysunek 16.



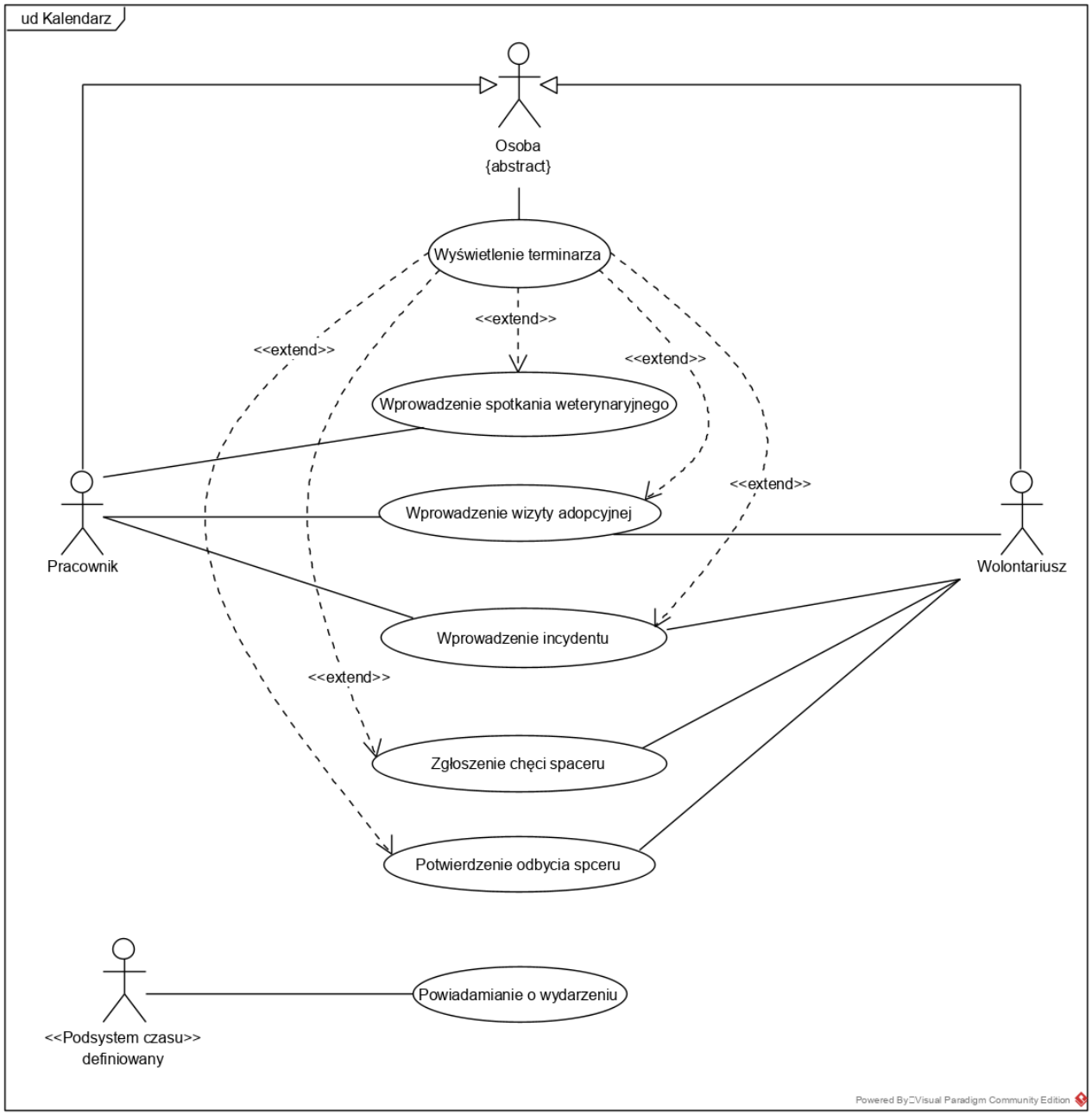
Rysunek 16. Diagram przypadków użycia: zarządzanie incydentami. Źródło: opracowanie własne.

Moduł zarządzanie sponsorami i wpływami przedstawia Rysunek 17.



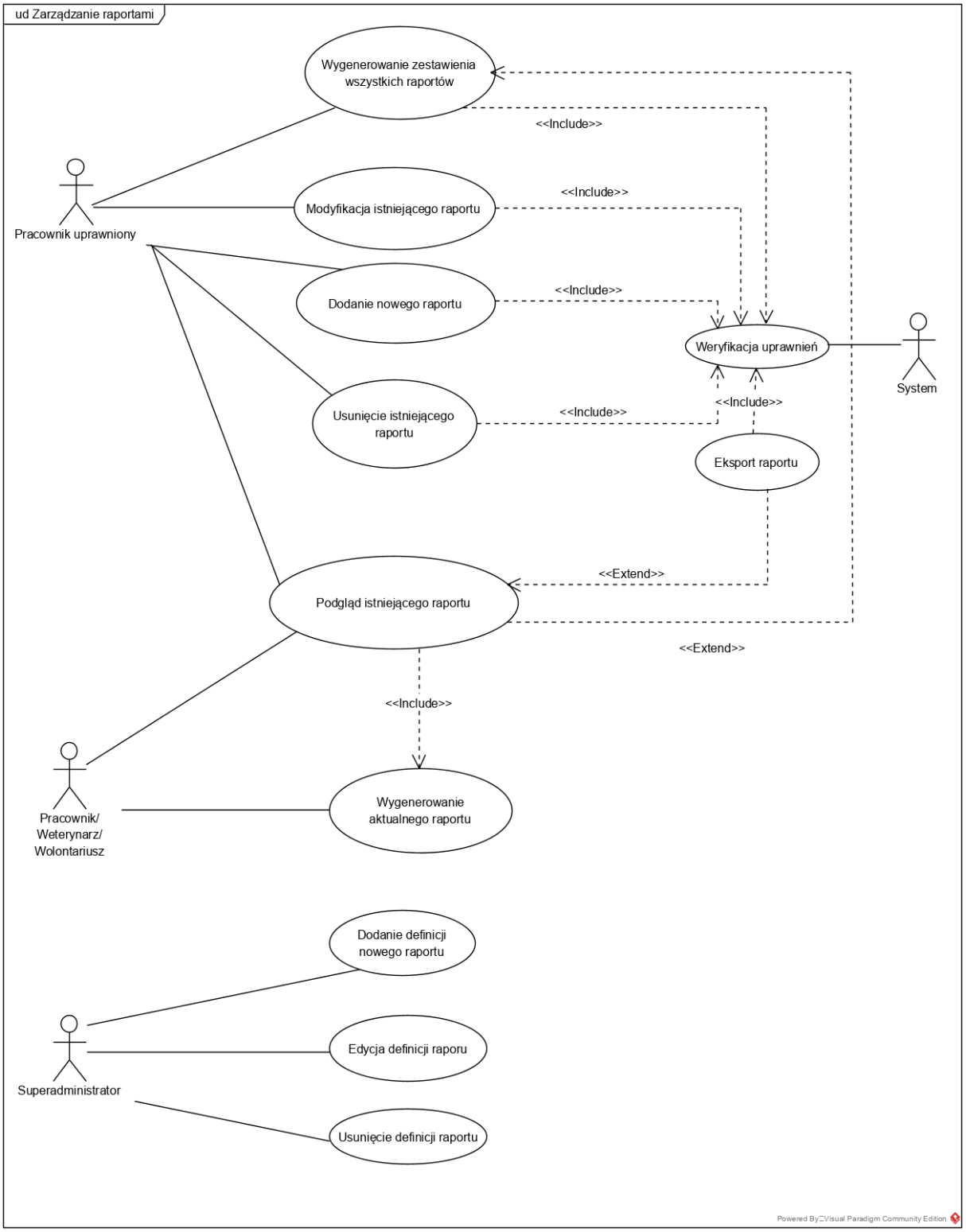
Rysunek 17. Diagram przypadków użycia: zarządzanie sponsorami i wpływami. Źródło: opracowanie własne.

Moduł kalendarz przedstawia Rysunek 18.



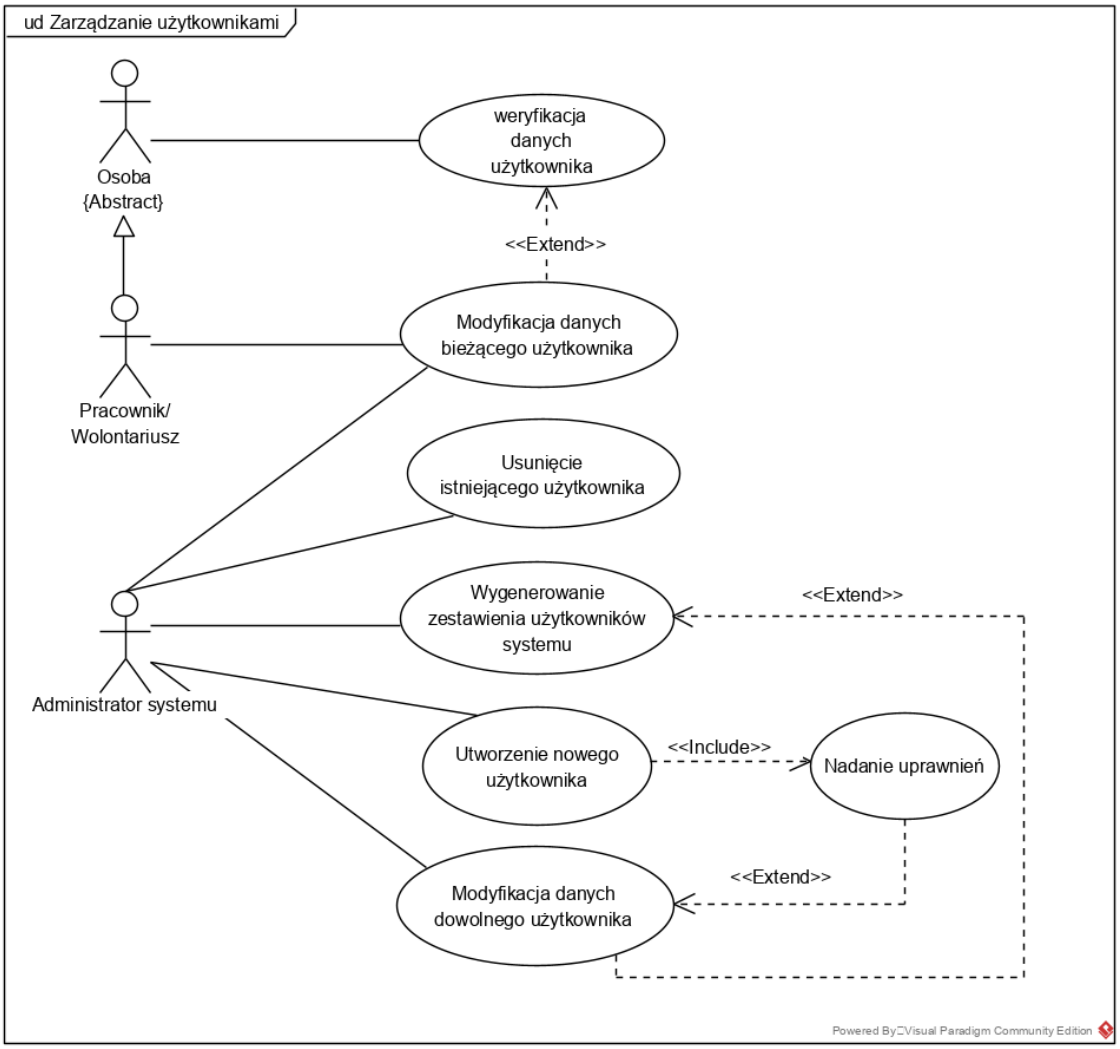
Rysunek 18. Diagram przypadków użycia: kalendarz. Źródło: opracowanie własne.

Moduł zarządzanie raportami przedstawia Rysunek 19.



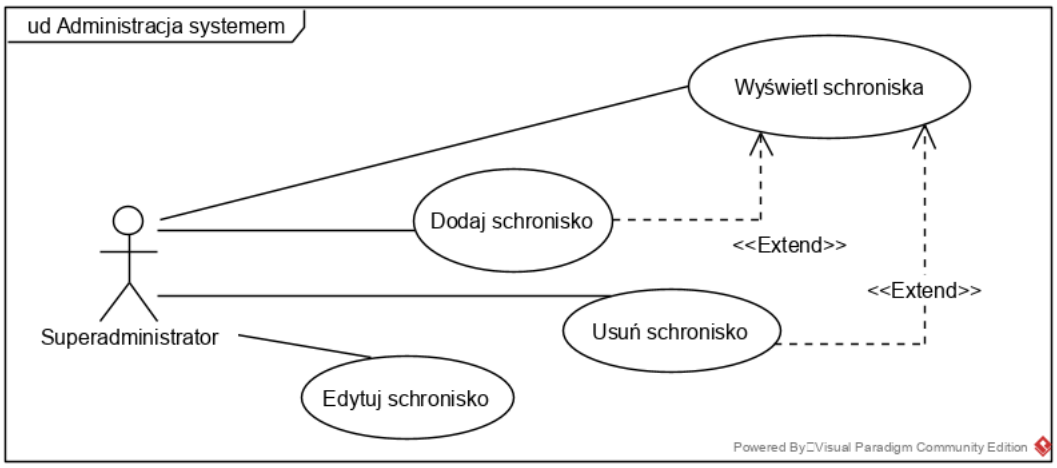
Rysunek 19. Diagram przypadków użycia: zarządzanie raportami. Źródło: opracowanie własne.

Moduł zarządzanie użytkownikami przedstawia Rysunek 20.



Rysunek 20. Diagram przypadków użycia: zarządzanie użytkownikami. Źródło: opracowanie własne.

Rysunek 21 przedstawia dodatkowe funkcje w ramach administracji systemem.



Rysunek 21. Diagram przypadków użycia: administracja systemem.

4.6. Omówienie wybranych przypadków użycia

Trzon aplikacji stanowią funkcjonalności związane z zarządzaniem danymi zwierząt, procesem adopcyjnym i zarządzaniem danymi weterynaryjnymi. Pierwszy moduł zawiera typowe przypadki użycia występujące w wielu aplikacjach, natomiast pozostałe dwa składają się z elementów specyficznych dla działalności schroniska dla zwierząt. Dlatego przypadki z tych modułów zostały wybrane do szczegółowej analizy. Pierwszy dotyczy złożenia wniosku adopcyjnego, drugi – dodania wizyty weterynaryjnej. Przykłady ilustrują też zmiany, jakie zostały dokonane w procesie transformacji projektu analitycznego projektu na projektowy, co zostanie opisane w kolejnym podrozdziale. Poniżej można zapoznać się ze scenariuszami i diagramami aktywności wybranych przypadków.

4.6.1. Dodaj aktywność medyczną: wizyta weterynaryjna

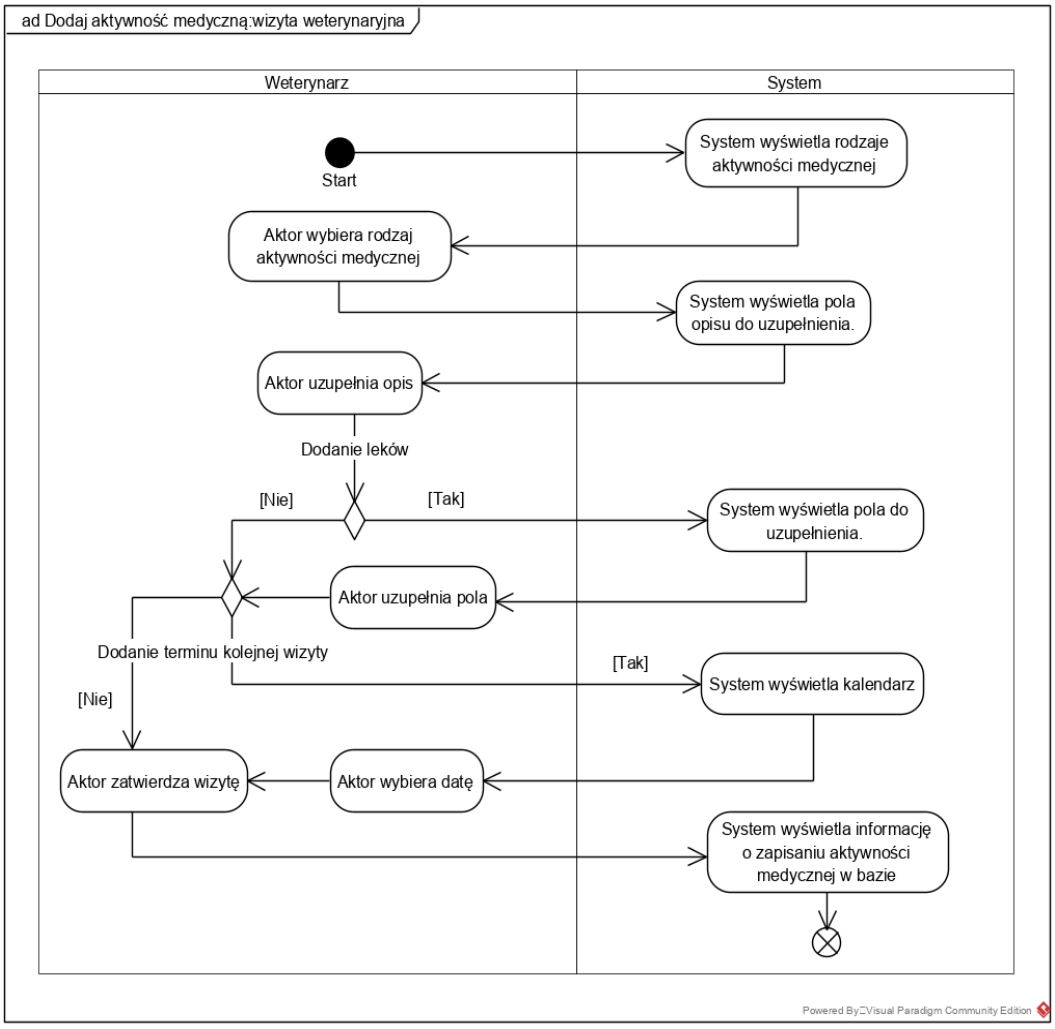
W module medycznym jednym z istotniejszych elementów jest tworzenie historii medycznej zwierzęcia. Dodawanie różnego typu czynności medycznych umożliwia przypadek: dodaj aktywność medyczną. Scenariusz dodawania wizyty weterynaryjnej przedstawia Tabela 14. Aktorem przypadku jest weterynarz.

Tabela 14. Scenariusz przypadku użycia: Dodaj aktywność medyczną: wizyta weterynaryjna. Źródło: opracowanie własne.

Dodaj aktywność medyczną: wizyta weterynaryjna	
Warunek początkowy	Istnienie zwierzęcia w bazie, istnienie weterynarza w bazie.
Główny przepływ zdarzeń	1. Aktor Weterynarz uruchamia przypadek użycia. 2. Aktor dodaje nową aktywność medyczną: wizytę weterynaryjną. 3. System wyświetla kartę wizyty z polami do uzupełnienia. 4. Aktor uzupełnia pola i zapisuje zmiany. 5.System kończy przypadek użycia.
Alternatywne przepływy zdarzeń	4a. Aktor uzupełnia pola i wybiera opcję *dodaj lek.* 5. System wyświetla pola dotyczące leku i sposobu jego podawania. 6. Aktor uzupełnia pola i zapisuje zmiany. 7. System kończy przypadek użycia. 4b. Aktor uzupełnia pola i wybiera *dodaj termin następnej wizyty*. 4. System wyświetla kalendarz. 5. Aktor wybiera datę, zatwierdza i zapisuje zmiany. 6. System kończy przypadek użycia.
Zakończenie	W dowolnym momencie

Warunek końcowy	W systemie zostają zapisane dane wizyty weterynaryjnej. Alternatywnie: w kalendarzu zwierzęcia zostaje dodany termin kolejnej wizyty.

Rysunek 22 przedstawia diagram aktywności ilustrujący proces dodawania wizyty weterynaryjnej.



Rysunek 22. Diagram aktywności dla przypadku użycia: Dodaj aktywność medyczną. Źródło: opracowanie własne.

4.6.2. Złóż wniosek adopcyjny

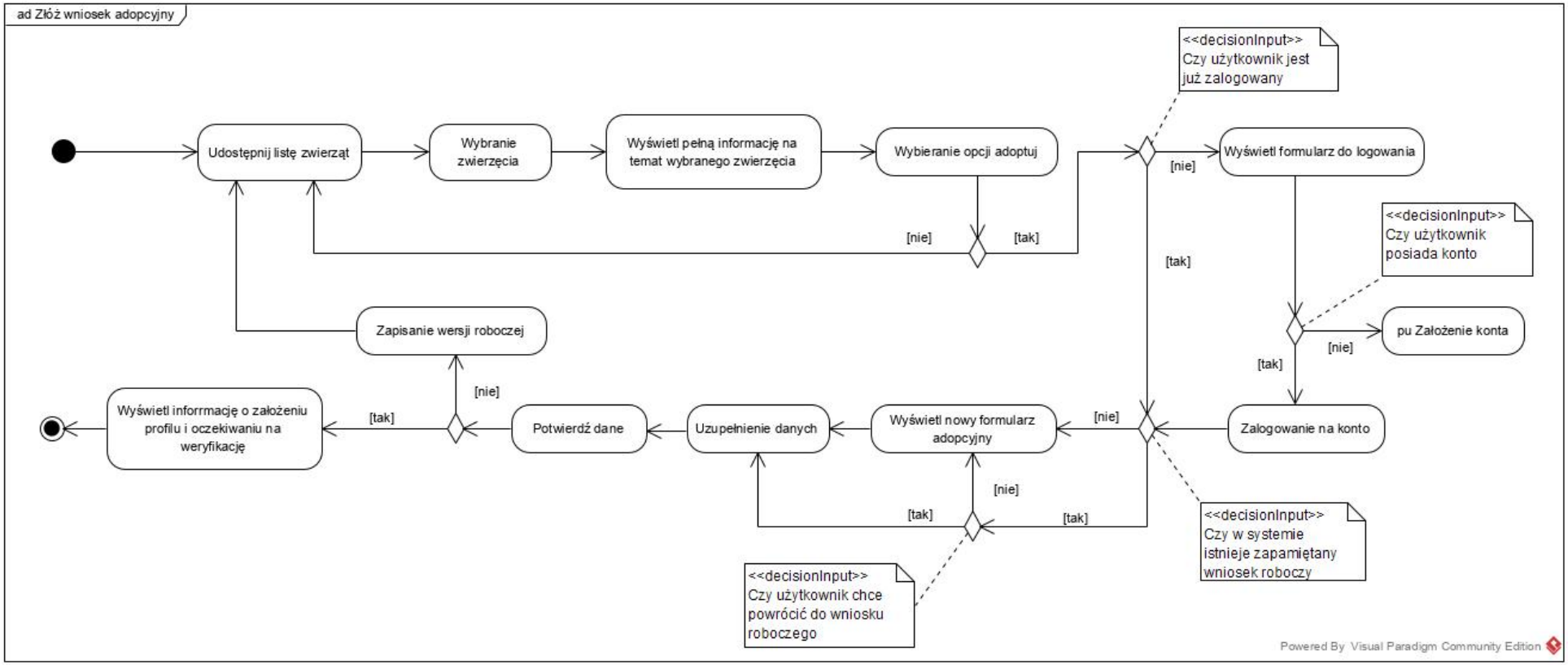
Jednym z pierwszych etapów adopcji zwierzęcia jest złożenie wniosku adopcyjnego. Tabela 15 przedstawia scenariusz dla tego przypadku użycia. Aktorem przypadku jest Adoptujący.

Tabela 15. Scenariusz przypadku użycia: Złóż wniosek adopcyjny. Źródło: opracowanie własne.

Złóż wniosek adopcyjny	
Warunek początkowy	Istnienie zwierzęcia w bazie

Główny przepływ zdarzeń	1. Aktor uruchamia przypadek użycia Złóż wniosek adopcyjny. 2. System wyświetla listę dostępnych zwierząt. Aktor przegląda listę i wybiera zwierzę. 3. System wyświetla pełną informację na temat zwierzęcia. Aktor klika na przycisk adoptuj. 4. System wyświetla formularz do logowania. Aktor loguje się na swoje konto. 5. System wyświetla nowy formularz adopcyjny. Aktor uzupełnia informacje i zatwierdza. 6. System wyświetla komunikat z potwierdzeniem uzupełnionych danych i pyta, czy wysłać wniosek. Aktor potwierdza informację. 7. System wyświetla informację na temat utworzenia profilu i oczekiwania na weryfikację.
Alternatywne przepływy zdarzeń	3a. Jeżeli aktor wybrał opcję Nie, to system wraca do punktu nr 2. 4a. Jeżeli aktor jest już zalogowany, to system przechodzi do punktu 5. 4b. Jeżeli aktor nie posiada konta to system uruchamia pu Założenie konta. 5a. Jeżeli istnieje zapisany w systemie wniosek roboczy, to system pyta czy aktor chce do niego powrócić, czy rozpocząć nowy z wybranym zwierzęciem. 5aa. Jeżeli aktor wybierze powrót do roboczego wniosku to system wczytuje wniosek roboczy. Aktor uzupełnia informacje i zatwierdza. 5ab. Jeżeli aktor wybierze nowy wniosek, to system wraca do punktu 5. 6a. Jeżeli aktor nie potwierdzi, wysłania wniosku system wyświetla informację o zapisaniu wniosku roboczego i przechodzi do punktu nr 2.
Zakończenie	W dowolnym momencie
Warunek końcowy	Jeżeli zostały zrealizowane wszystkie warunki złożenia wniosku adopcyjnego to w systemie zostają zapisane informacje na temat osoby, która chce adoptować zwierzę, zwierzęcia, które chce adoptować, a wypełniony formularz przekazywany jest do pracownika.

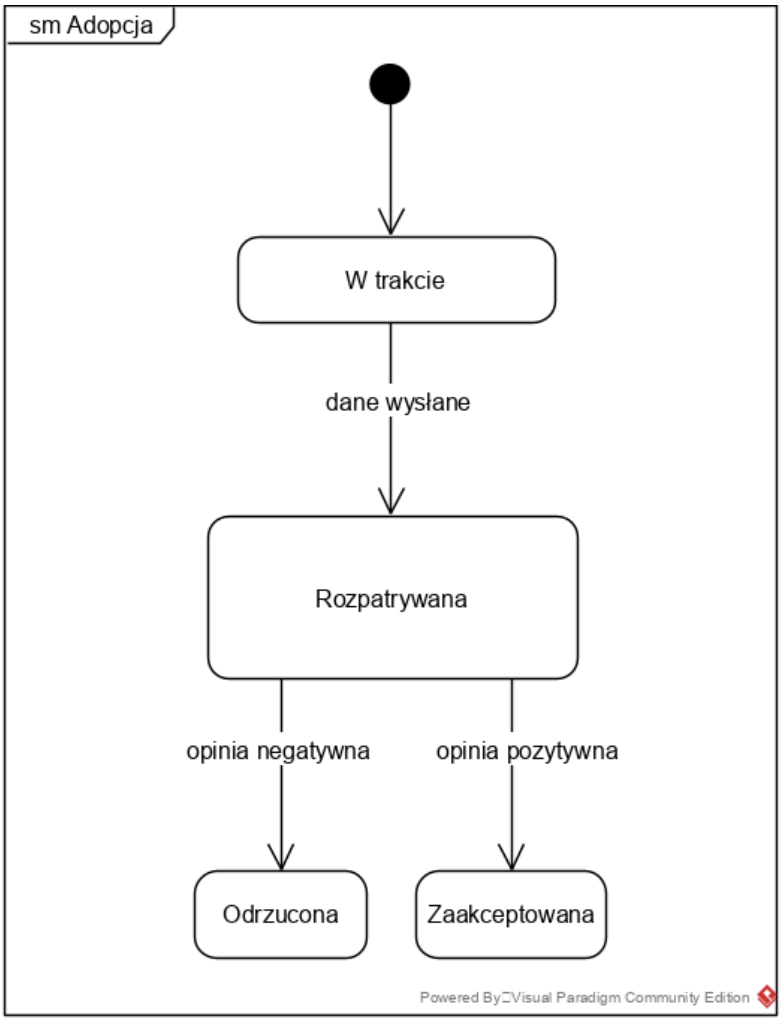
Rysunek 23 przedstawia diagram aktywności ilustrujący proces składania wniosku adopcyjnego.



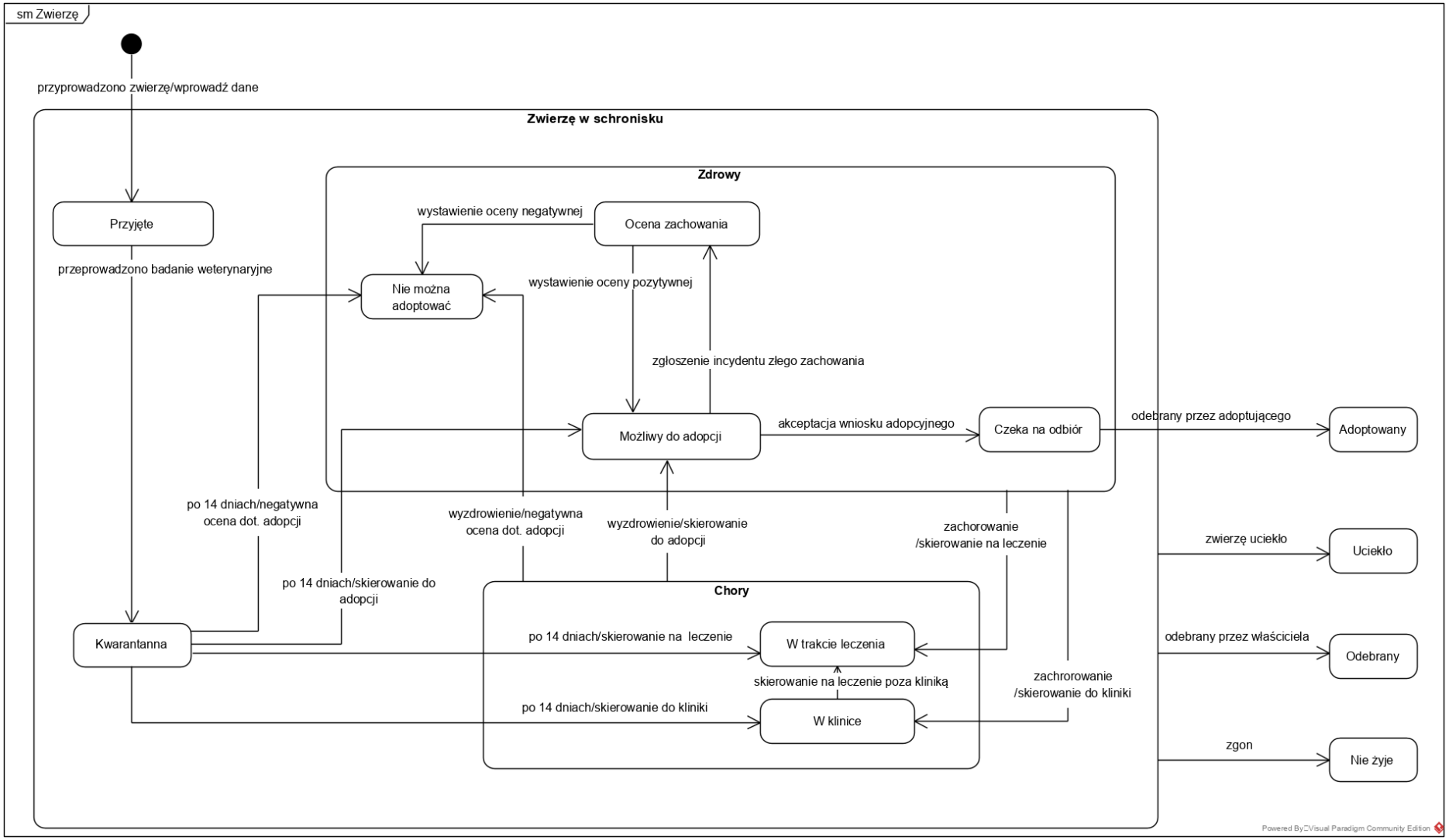
Rysunek 23. Diagram aktywności dla przypadku użycia: Złóż wniosek adopcyjny. Źródło: opracowanie własne.

4.7. Diagramy stanu

Poniżej zostały przedstawione diagramy stanu. Rysunek 24 przedstawia klasę Adopcja, a Rysunek 25 klasę Zwierzę.

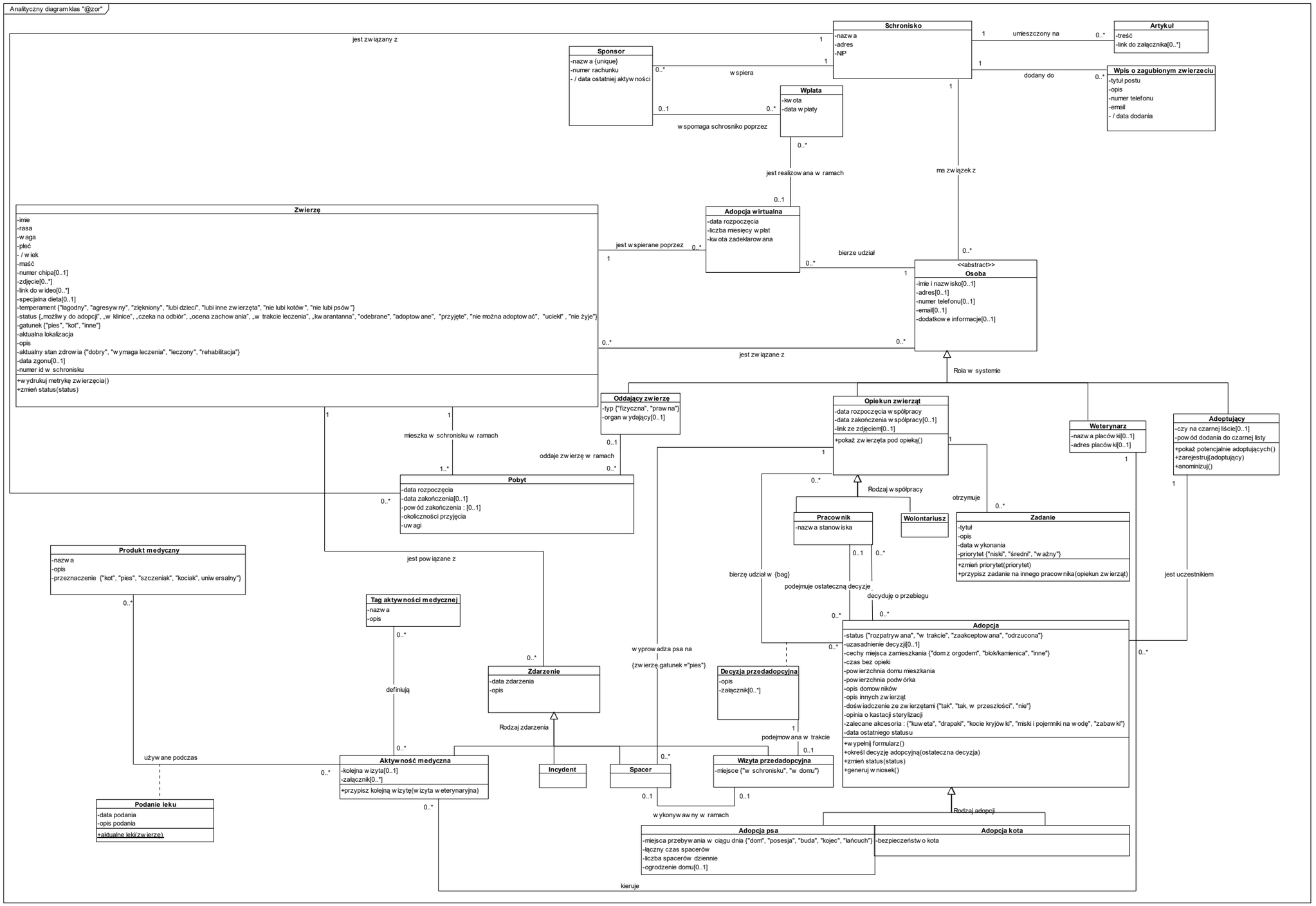


Rysunek 24. Diagram stanu dla klasy Adopcja. Źródło: opracowanie własne.



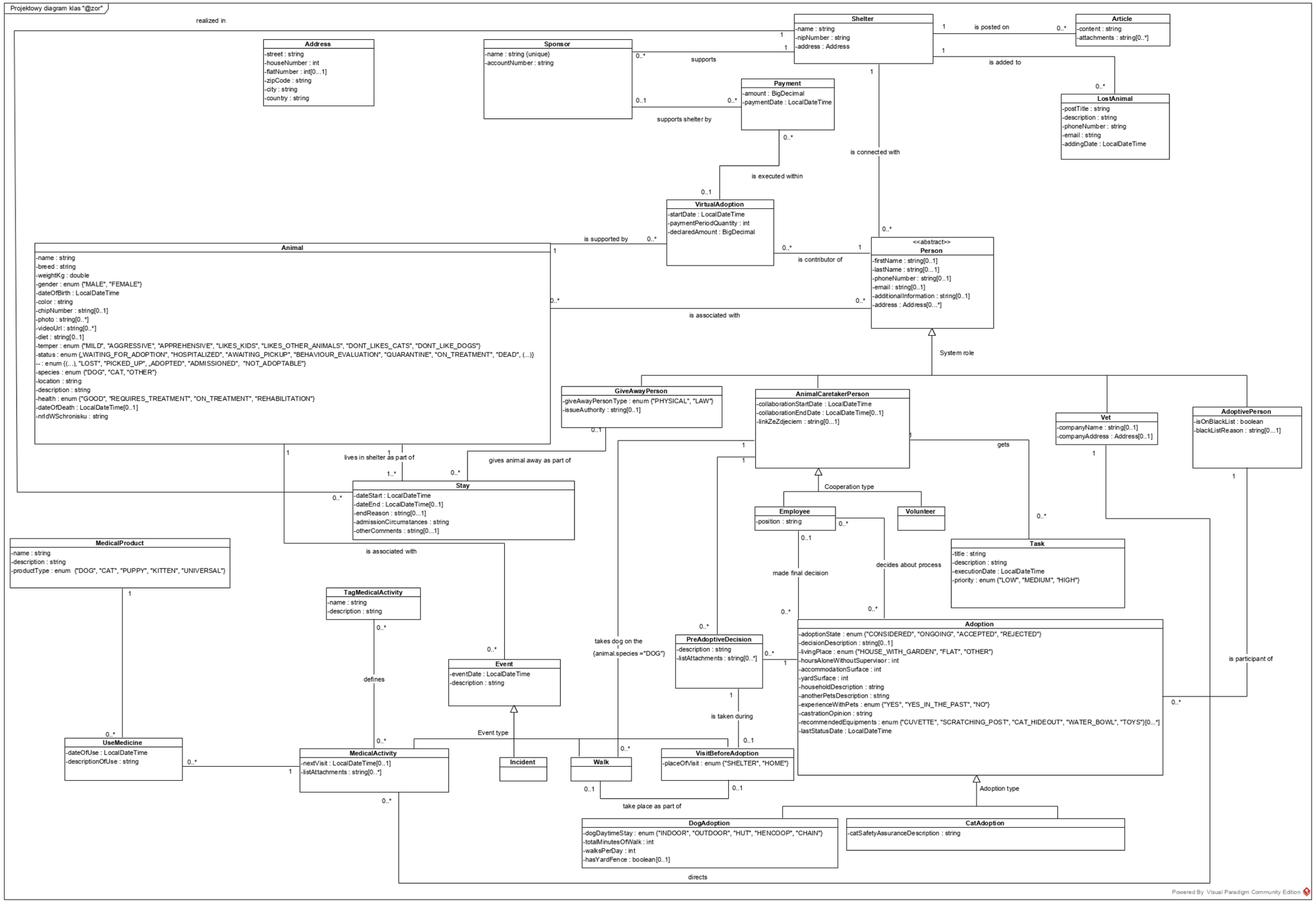
Rysunek 25. Diagram stanu dla klasy Zwierzę. Źródło: opracowanie własne.

4.8. Analityczny diagram klas



Rysunek 26. Analityczny diagram klas. Źródło: opracowanie własne.

4.9. Projektowy diagram klas



Rysunek 27. Projektowy diagram klas. Źródło: opracowanie własne.

4.10. Etapy powstawania projektu

Celem tego rozdziału jest zapoznanie czytelnika z historią tworzenia kluczowego elementu projektowego jakim jest diagram klas. Zawarte tu informacje opisują etapy powstawania analitycznego oraz projektowego diagramu klas wyjaśniając zamierzenia oraz pobudki jakimi kierowali się członkowie zespołów przy budowie, a następnie przy modyfikacji tych struktur.

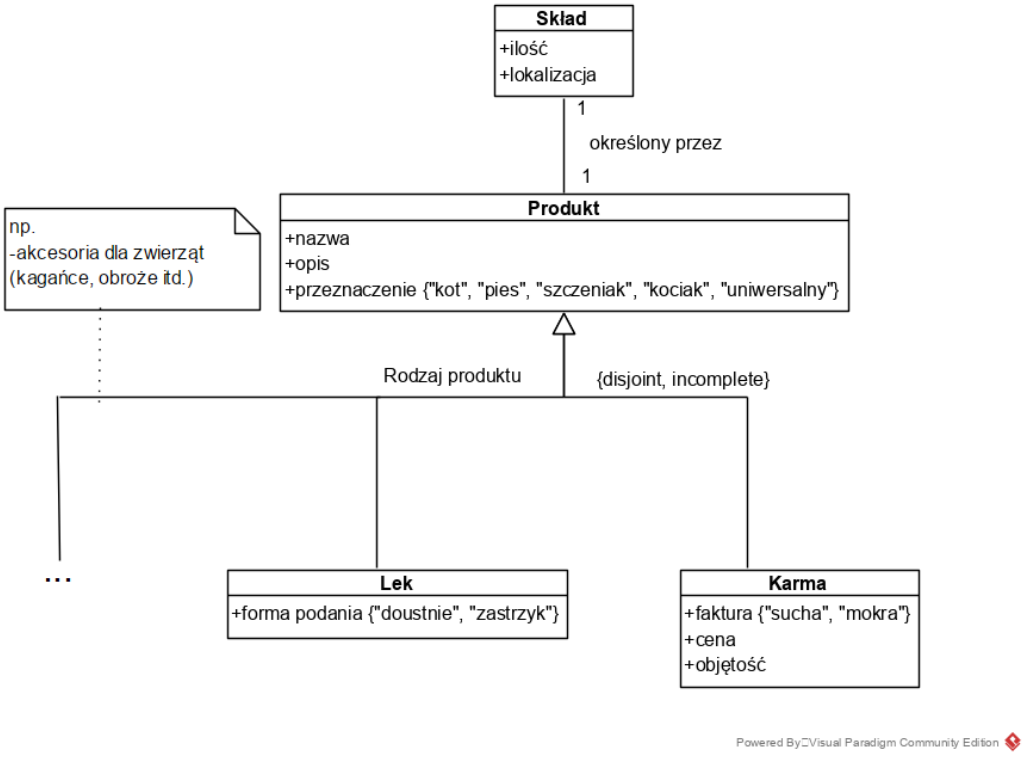
Zespół analityczny na podstawie badań rynku, przeprowadzonych wywiadów z pracownikami schroniska dla zwierząt oraz testów narzędzi konkurencyjnych zebrał listę wymagań funkcjonalnych oczekiwanego systemu do obsługi schronisk. Na tej podstawie przygotowano moduły z jakich składać ma się program i jakie funkcjonalności każdy z nich ma realizować. Na tej bazie zaproponowano pierwszy analityczny diagram klas, który w trakcie dalszych prac oraz cyklicznych konsultacji z zespołami projektowym oraz implementacyjnym był modyfikowany. Poprawki powodowane były przede wszystkim przez zmiany w zakresie planowanych funkcjonalności dla pierwszej wersji systemu, modyfikacje modułów, a także przez sposób realizacji poszczególnych elementów aplikacji. Brano również pod uwagę przyszły sposób przełożenia diagramu analitycznego na projektowy mając na uwadze możliwości wybranych technologii implementacyjnych. Ponieważ zadecydowano o użyciu języka obiektowego Java oraz relacyjnej bazy danych, więc do zamiany modelu obiektowego na relacyjny użyto mapowania obiektowo-relacyjnego (ORM). W tym celu wykorzystano bibliotekę Hibernate oraz bazę danych MySQL. Do pracy z Hibernate zastosowano adnotacje zgodne ze standardem JPA. Niniejszy rozdział opisuje istotne zmiany w diagramach i ich przyczyny wynikające z wymienionych wyżej powodów.

4.10.1. Modyfikacje analitycznego diagramu klas

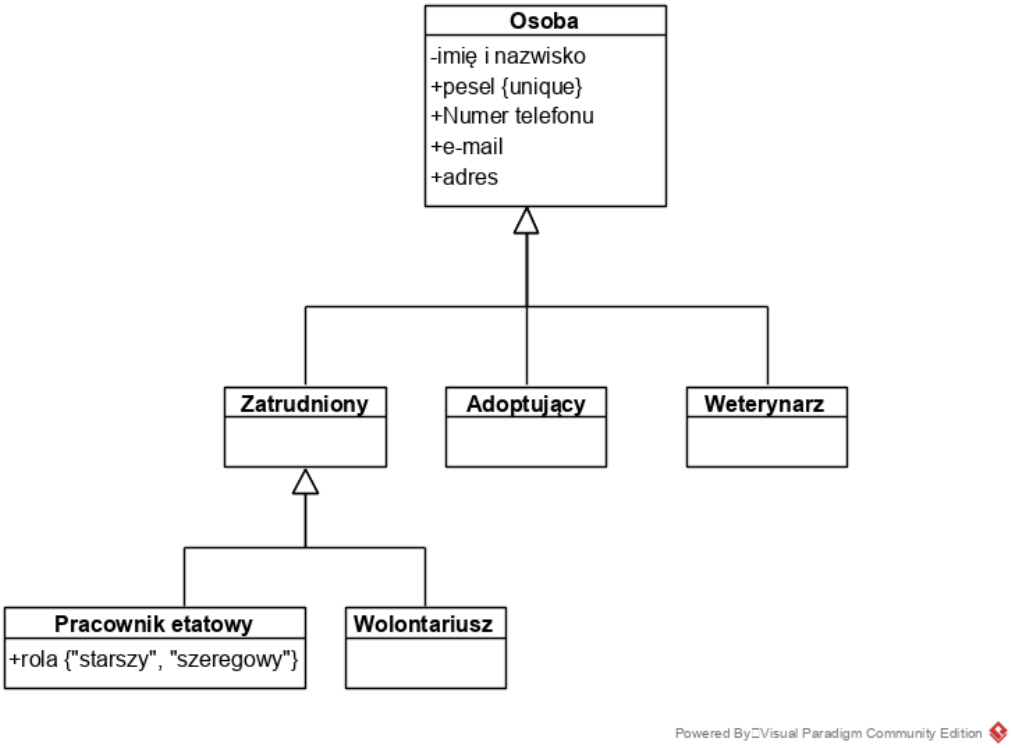
Poniższe punkty prezentują istotne zmiany dokonane w analitycznym diagramie klas.

4.10.1.1. Faza I. Budowa struktury systemu. Zmiany dokonane do 16.11.2019 roku

- Usunięto klasy "Produkt", "Skład", "Lek", "Karma" oraz elipsę dla kolejnych rodzajów produktów. Decyzją zespołu postanowiono, iż pierwsza wersja programu nie będzie obsługiwać stanów magazynowych/wyposażenia i akcesoriów dla zwierząt, a zaoszczędzone roboczogodziny przeznaczone zostaną przede wszystkim na rozwój elementów dotyczących stricte opieki nad zwierzętami. Rysunek 28 przedstawia usunięte klasy dotyczące stanów magazynowych.



Rysunek 28. Fragment analitycznego diagramu klas z fazy I: klasa Produkt. Źródło: opracowanie własne.

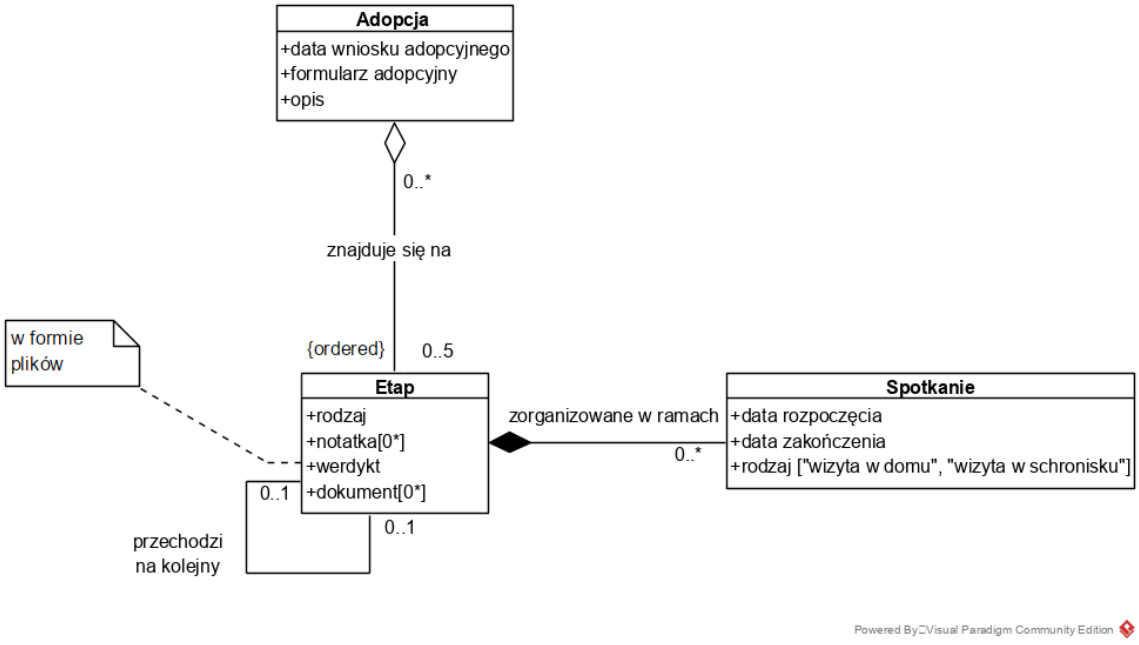


Rysunek 29. Fragment analitycznego diagramu klas z fazy I: klasa Zatrudniony. Źródło: opracowanie własne.

- Uproszczono strukturę dotyczącą aktorów powiązanych z systemem i usunięto klasy "Pracownik etatowy", "Zatrudniony". Decyzją zespołu postanowiono, iż pierwsza wersja

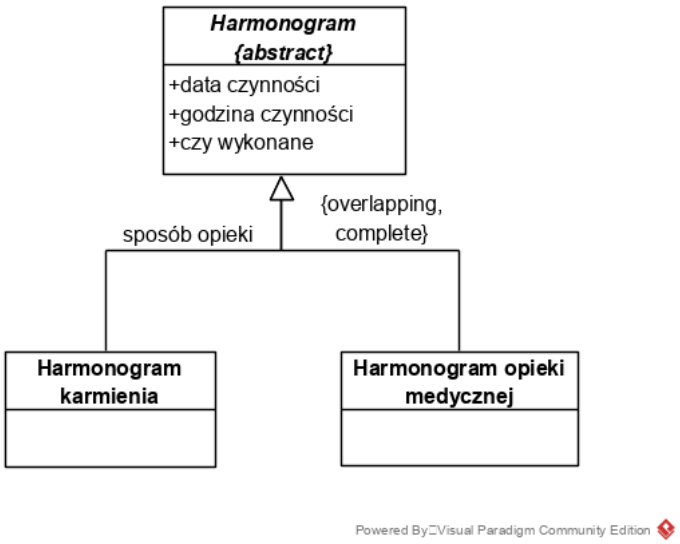
programu nie będzie obsługiwać zarządzania kadrami, a roboczogodziny przeznaczone zostaną przede wszystkim na rozwój elementów dotyczących stricte opieki nad zwierzętami. Oznacza to, że diagram prezentuje strukturę ról użytkowników pod kątem uprawnień systemowych, a nie uwzględnia struktury zatrudnienia i wszystkich stanowisk. Rysunek 29 przedstawia pierwotną wersję dotyczącą aktorów systemu.

- Zadecydowano o zmianie modelu dotyczącego adopcji zwierząt. Usunięto klasy "Spotkanie" oraz "Etap". Etapowanie procesu zostało umieszczone jako atrybut typu wyliczeniowego w klasie "Adopcja". Natomiast historia wizyt została powiązana ze zdarzeniami. Rysunek 30 przedstawia pierwotny moduł adopcyjny systemu na diagramie klas.



Rysunek 30. Fragment analitycznego diagramu klas z fazy I: klasy Etap i Spotkanie. Źródło: opracowanie własne.

- Obszar harmonogramowania opieki nad zwierzętami został zmieniony na formę kalendarzową; zmieniono także koncepcję tak, by wszelkie wydarzenia z datą ujednolicić tworząc nadklasę "Zdarzenie" (z podklasami: "Spacer", Wizyta weterynaryjna", "Incydent", "Wizyta przedadopcyjna"), aby kalendarz był czytelny, a także umożliwiał zamieszczanie dowolnych wpisów nie ograniczając ich wyłącznie do zamkniętej listy dotyczącej karmienia i opieki medycznej. Harmonogramowanie karmienia usunięto przewidując nadmiarowość danych i zbytnią szczegółowość. Rysunek 31 przedstawia pierwszą koncepcję budowy zdarzeń w systemie.

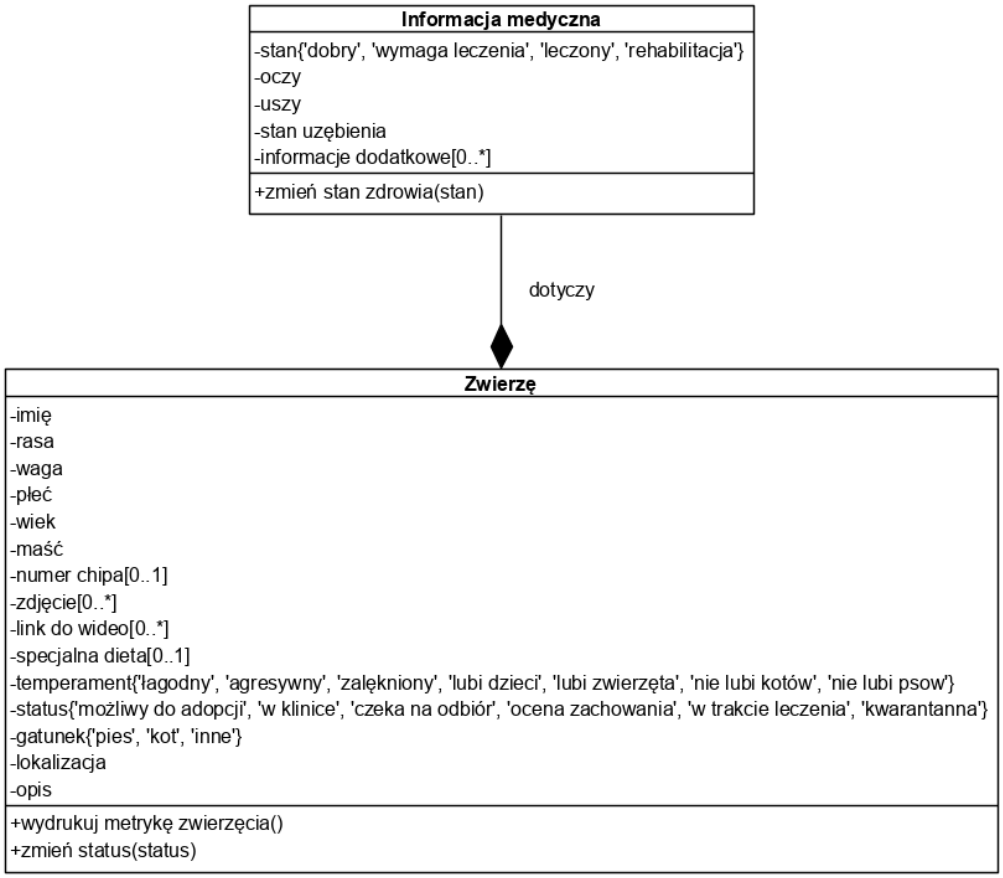


Rysunek 31. Fragment analitycznego diagramu klas z fazy I: klasa Harmonogram. Źródło: opracowanie własne.

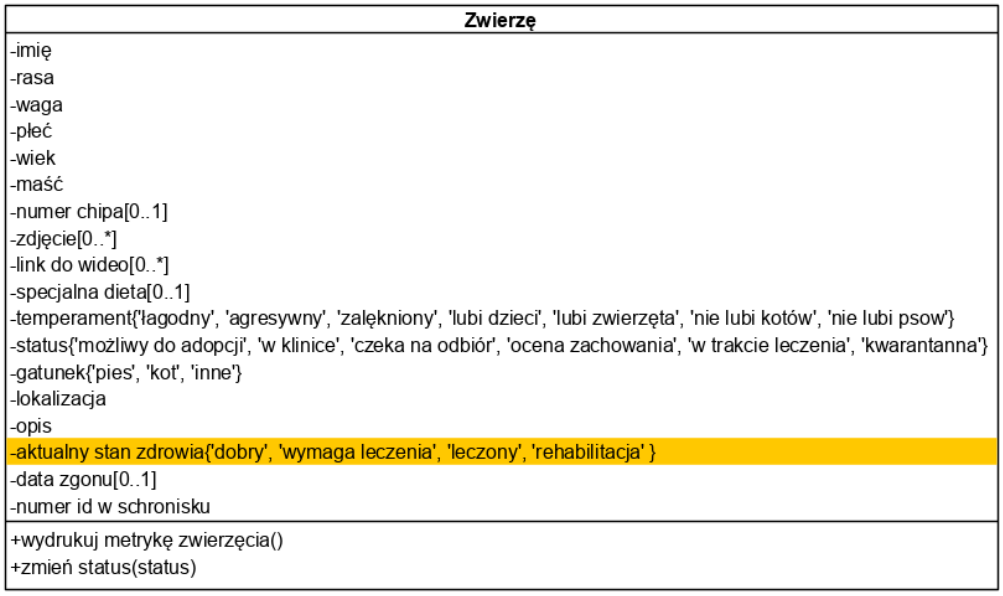
4.10.1.2. Faza II. Doskonalenie modelu. Zmiany dokonane po 16.11.2019 roku

- Dodano klasę "Schronisko", co odzwierciedla istotną decyzję projektową dotyczącą opracowywania systemu dla różnych placówek jako jednej aplikacji webowej, do której logują się zdalnie poszczególne schroniska.
- Przebudowano moduł medyczny. Wizyta weterynaryjna stała się elementem klasy "Aktywność medyczna" w ramach ujednolicania modelu. Rodzaje aktywności medycznej z atrybutu przeniesiono do oddzielnej klasy "Tag aktywności medycznej", by zwiększyć ich czytelność oraz dodano możliwość umieszczania opisów. Połączono klasy "Szczepionka" oraz "Produkt medyczny". Usunięto zbędną nadklasę "Elementy medyczne" przenosząc atrybuty do podklas.
- Klasa "Informacje medyczne" została usunięta, a istotne atrybuty zostały przeniesione do klasy "Zwierzę", co prezentuje Rysunek 32. Nie było potrzeby trzymania tych danych osobno.

Faza I

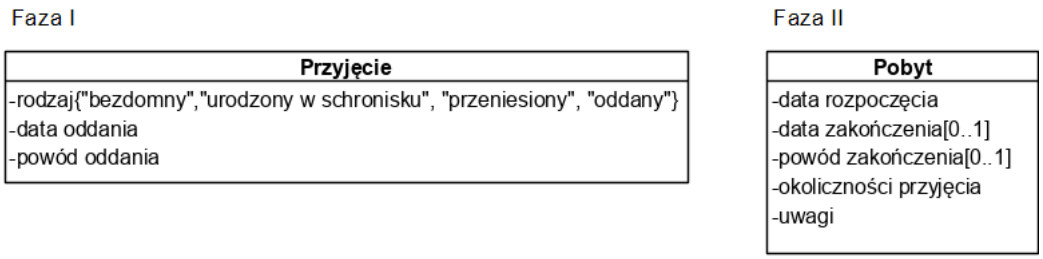


Faza II



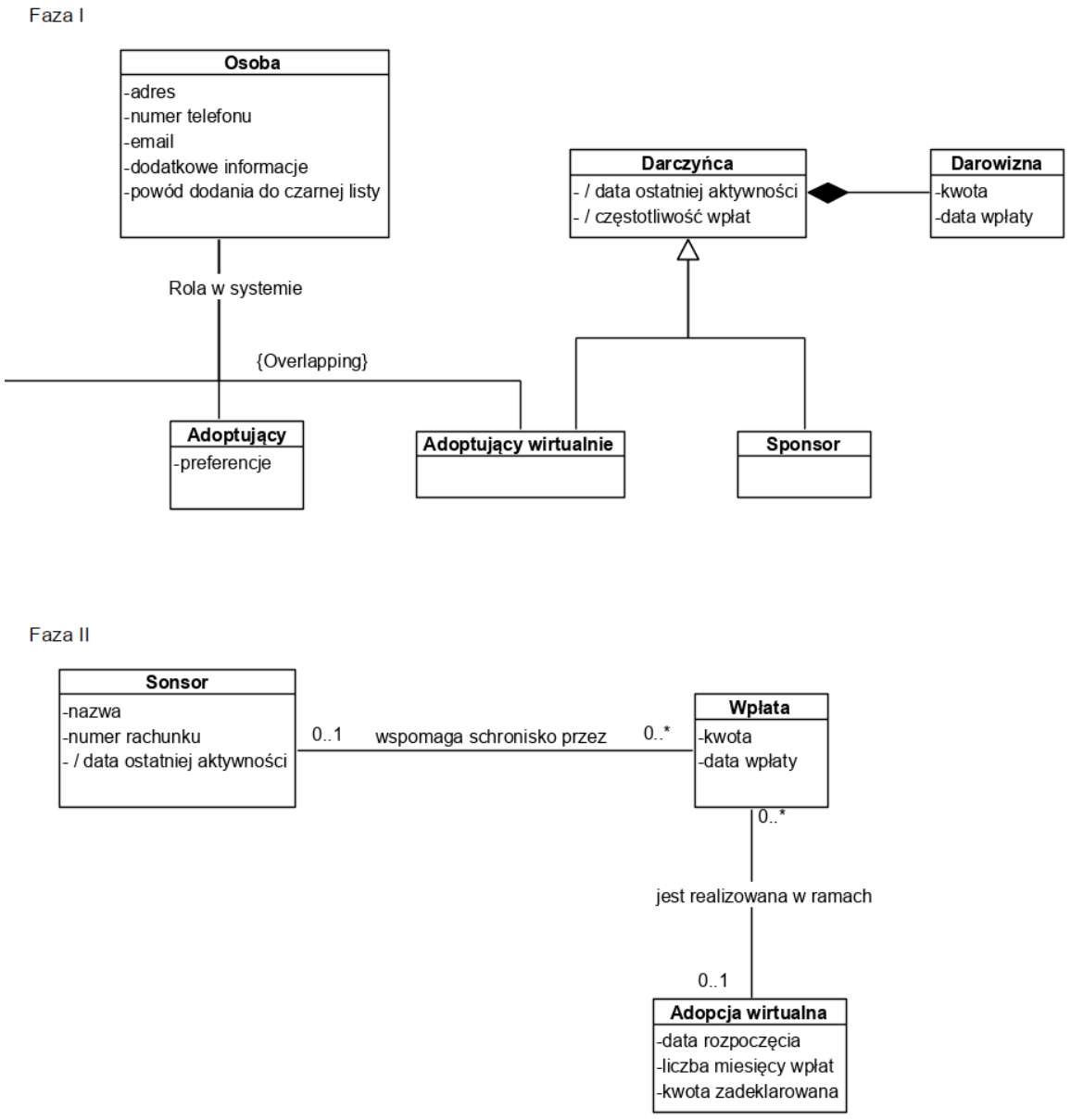
Rysunek 32. Porównanie fragmentu analitycznego diagramu klas z fazy I oraz II: klasa Zwierzę. Źródło: opracowanie własne.

- Klasę "Przyjęcie" zamieniono na klasę "Pobyt". Ten zabieg umożliwił logiczne przechowywanie informacji nie tylko o przyjęciu, ale i usunięciu ze stanu danego zwierzęcia, co prezentuje Rysunek 33



Rysunek 33. Porównanie fragmentu analitycznego diagramu klas z fazy I oraz II: klasy Przyjęcie i Pobyt. Źródło: opracowanie własne.

- Klasa "Zagubione zwierzę" została zamieniona na klasę "Wpis o zagubionym zwierzęciu", co odwzorowuje logikę działania systemu, gdzie te informacje stają się automatycznie wpisem na stronie schroniska w stosowej zakładce.
- Zmieniono logikę obsługi datków oraz adopcji wirtualnej z podejścia osobowego na podejście zdarzeniowe. Obecnie zarejestrowana wpłata pieniężna na konto schroniska jest zdarzeniem powodującym powstanie obiektu klasy "Wpłata", a powiązane asocjacjami klasy "Sponsor" oraz "Adopcja wirtualna" wskazują z jakim typem darowizny system ma do czynienia i jak ją obsłużyć. Ponadto w klasie abstrakcyjnej "Osoba" atrybuty stały się opcjonalne by zapewnić anonimowość adopcji wirtualnej. Powstały metody zapewniające konieczność nadania wartości tym atrybutom przy tworzeniu obiektów podklas klasy „Osoba” dla sytuacji innych niż anonimowa adopcja wirtualna. Dotychczasowe podejście, gdzie wpłata nowych środków powodowała powstanie obiektów z podklas nadklasy "Darczyńca", a w przypadku adopcji wirtualnej również z drugiej nadklasy "Osoba" byłoby trudniejsze w implementacji oraz wymagałoby systematycznej ręcznej obsługi każdej kolejnej wpłaty adoptującego wirtualnie. Obecne rozwiązanie zapewnia tę obsługę automatycznie poprzez zaharmonogramowanie przyszłych wpłat. W trakcie trwania fazy II próbowano rozwiązać skomplikowaną kwestię adopcji wirtualnej i wiążącą się z nią obsługą ograniczonych danych personalnych darczyńców za pomocą innego podejścia osobowego stosując anonimizację obiektów klasy "Osoba" poprzez utworzenie nadklasy "Osoba podstawowa", jednak sztuczność tego rozwiązania, brak oczekiwanych korzyści i ograniczenia jakie niosła ta koncepcja wpłynęły na decyzję o jej porzuceniu. Znaczące zmiany jakie zaszły w tym module prezentuje Rysunek 34 (nie uwzględniono okresu przejściowego z zastosowaniem klasy "Osoba podstawowa").



Rysunek 34. Porównanie fragmentu analitycznego diagramu klas z fazy I oraz II: klasy powiązane z modułem zarządzania sponsorami i wpłatami. Źródło: opracowanie własne.

- W celu zapełniania treścią strony głównej schroniska pojawiła się klasa "Artykuł". Newsy, ogłoszenia, informacje, którymi będzie chciało się podzielić schronisko ze swoimi czytelnikami będą pojawiały się odwiedzającym stronę internetową jako slidery.
- W trakcie prac w fazie II dodano ograniczenie *overlapping* pomiędzy podklasami klasy "Osoba". Decyzja ta odzwierciedlać miała sytuację, w której dowolny z aktorów systemu (np. weterynarz) stawał się również adoptującym zwierzę. Ponieważ jednak implementacja tego ograniczenia w języku Java jest skomplikowana, błędogenna i zasadniczo zaprzecza idei i korzyściom wynikającym z dziedziczenia, poddano rozwiązanie pod ponowną dyskusję. W jej trakcie podjęto biznesową decyzję, iż osoby istniejące już w systemie pod inną rolą niż adoptujący, będą zmuszone do założenia oddzielnego konta gdy podejmą decyzję o adopcji

zwierzęcia. Jako uzasadnienie stwierdzono, że będą to sytuacje sporadyczne, a także tacy użytkownicy będą już obeznani z systemem, a więc wykluczyć można problem uciążliwości przy zakładaniu nowego konta ze względu na brak umiejętności korzystania z aplikacji. Tym samym omawiane ograniczenie *overlapping* zostało usunięte z diagramu.

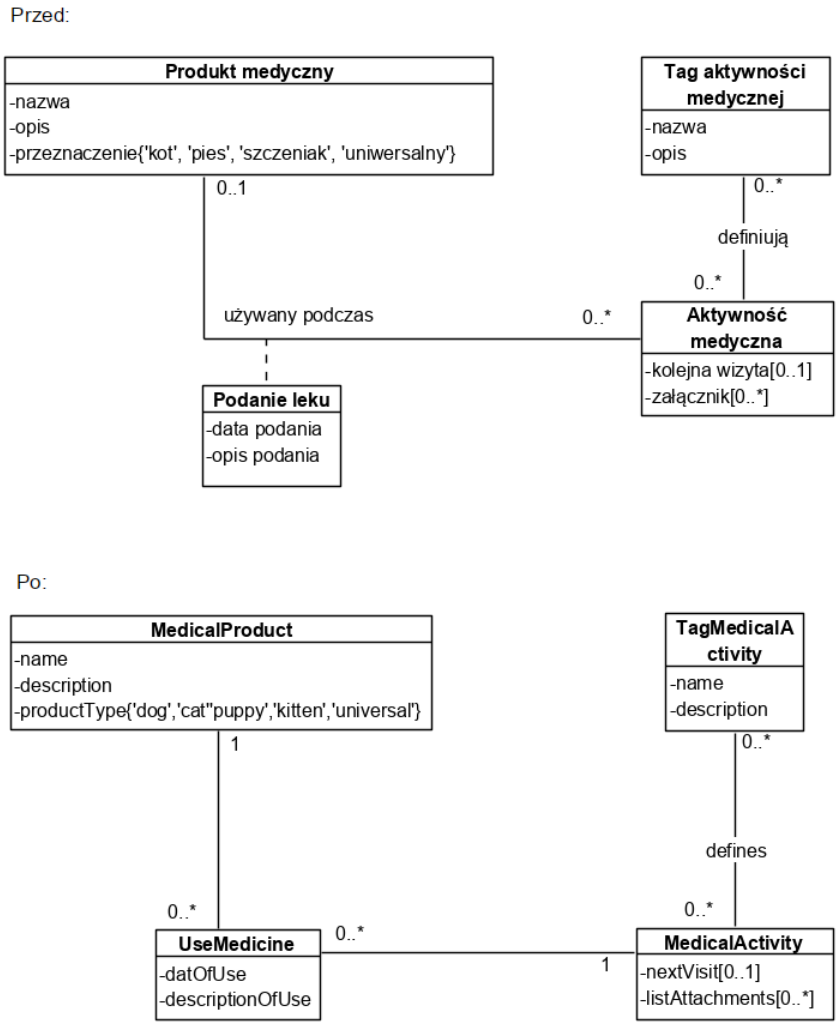
- W asocjacji pomiędzy klasami "Opiekun zwierząt" a "Adopcja" umieszczono ograniczenie *bag*. Umożliwia to wielokrotne wystawianie decyzji przedadopcyjnych dla jednej adopcji przez tego samego opiekuna zwierząt.
- W asocjacji pomiędzy klasami "Spacer" oraz "Wizyta przedadopcyjna" usunięto ograniczenie *overlapping* zastępując jego działanie biznesowe asocjacją typu 0-1. Znacząco uprości to implementację a zapewnia oznaczenie każdego spaceru jako wizyty zmierzającej do adopcji zwierzęcia.
- W klasie "Adoptujący" umieszczono metodę *anonimizuj()*. Celem tej metody jest najprostsze i bezpieczne rozwiązanie kwestii procedury adopcji, gdzie część danych potencjalnego adoptującego na pewnym etapie może pozostać nieznana. Tu ponownie wykorzystuje się omówione już rozwiązanie opcjonalnych atrybutów nadklasy "Osoba".
- Dla eleganckiego i wygodnego harmonogramowania podawania leków zmodyfikowano klasę asocjacyjną "Podanie leku" występującą pomiędzy klasami "Aktywność medyczna" oraz "Produkt medyczny". Z biznesowego punktu widzenia weterynarz będzie w trakcie wizyty ("Aktywności medycznej") zapisywał lek oraz opis podania (sposób, dawkowanie) jednocześnie wpisując datę oraz godzinę pierwszej dawki, a następnie częstotliwość dawkowania i zakończenie. Na tej podstawie system wygeneruje w kalendarzu obiekty klasy "Podanie leku" dla danego zwierzęcia jako zdarzenia przyszłe przypominając opiekunom o konieczności aplikacji leku.

4.10.2. Przemodelowanie analitycznego diagramu klas na projektowy

Na bazie analitycznego diagramu klas zamodelowano projektowy diagram klas, tj. taki gdzie uwzględnia się szczegóły implementacyjne oraz użyte technologie i związane z tym ograniczenia. Poniżej omówione zostały najważniejsze kwestie i decyzje podjęte w trakcie prac.

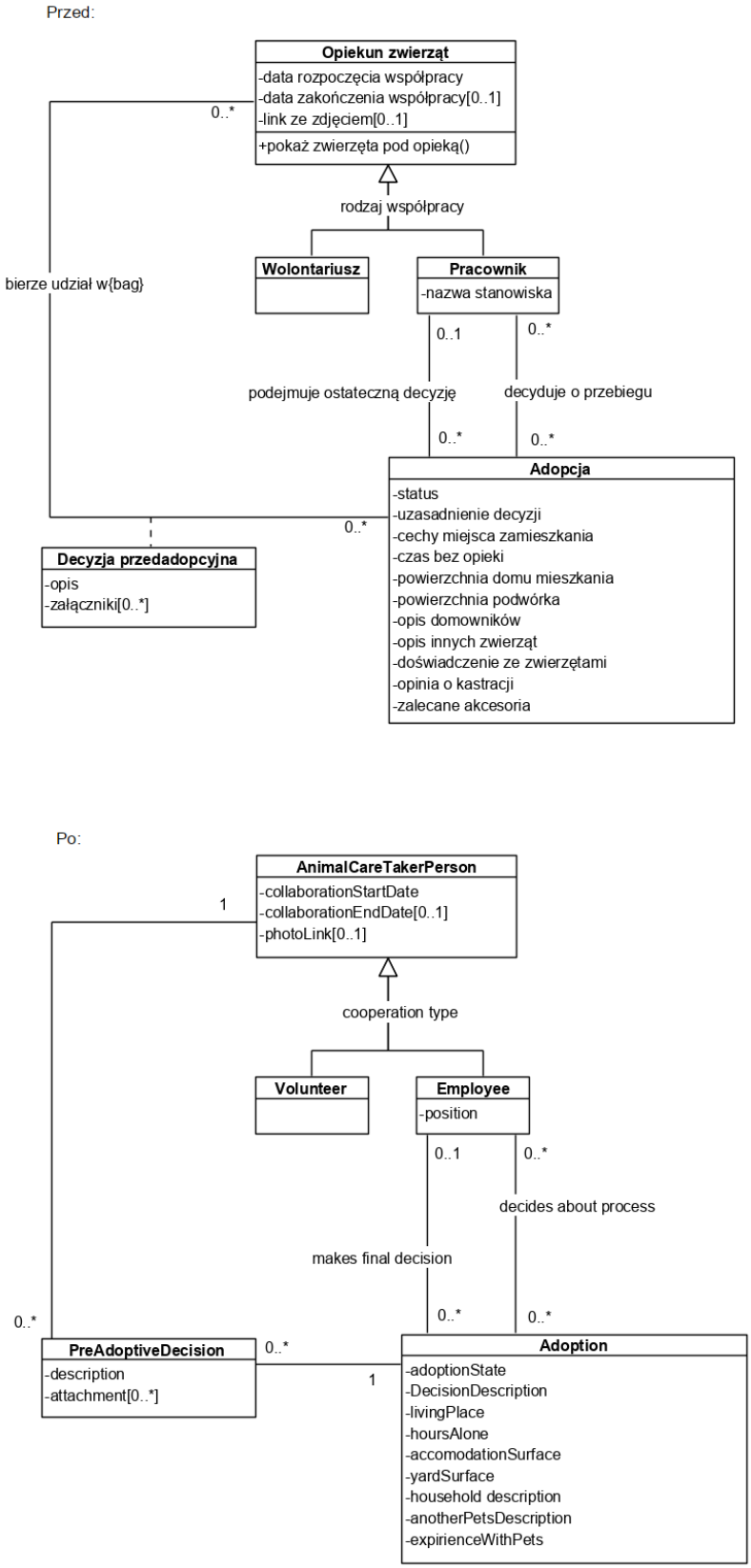
- Istotna zmiana dotycząca całego systemu wynikła z decyzji o implementacji z użyciem wyłącznie języka angielskiego. Spowodowało to przetłumaczenie całego diagramu z języka polskiego uwzględniając nazwy klas, atrybutów, asocjacji, metod itd.
- Utworzono nową klasę „Address”. Wydzielono ją na podstawie atrybutu złożonego o tej samej nazwie znajdującego się w klasie „Person”. Przyczyną tej decyzji był brak natywnego sposobu implementacji atrybutu złożonego w języku Java.
- Wprowadzono klasę pośredniczącą „UseMedicine”, której każdy obiekt oznacza podanie leku. Na diagramie analitycznym było to reprezentowane przez asocjację o liczności (0…*) – (0…*)

z klasą asocjacyjną. Zmiana doprowadziła także do modyfikacji liczności w asocjacjach. Przyczyną przemodelowania był brak natywnego sposobu implementacji asocjacji z klasą asocjacyjną w języku Java. Zmianę prezentuje Rysunek 35.



Rysunek 35. Porównanie fragmentu analitycznego oraz implementacyjnego diagramu klas: klasy Podanie Leku i UseMedicine. Źródło: opracowanie własne.

- Wprowadzono klasę pośredniczącą „PreAdoptiveDecision" której każdy obiekt oznacza ocenę pracownika o adoptującym wystawioną po odbytej wizycie. Na diagramie analitycznym była to asocjacja o liczności (0…*) – (0…*) z klasą asocjacyjną. Dodanie tej klasy doprowadziło również do zmian liczności w asocjacjach. Zmiana doprowadziła także do modyfikacji liczności w asocjacjach. Przyczyną przemodelowania był brak natywnego sposobu implementacji asocjacji z klasą asocjacyjną w języku Java. Zmianę prezentuje Rysunek 36.



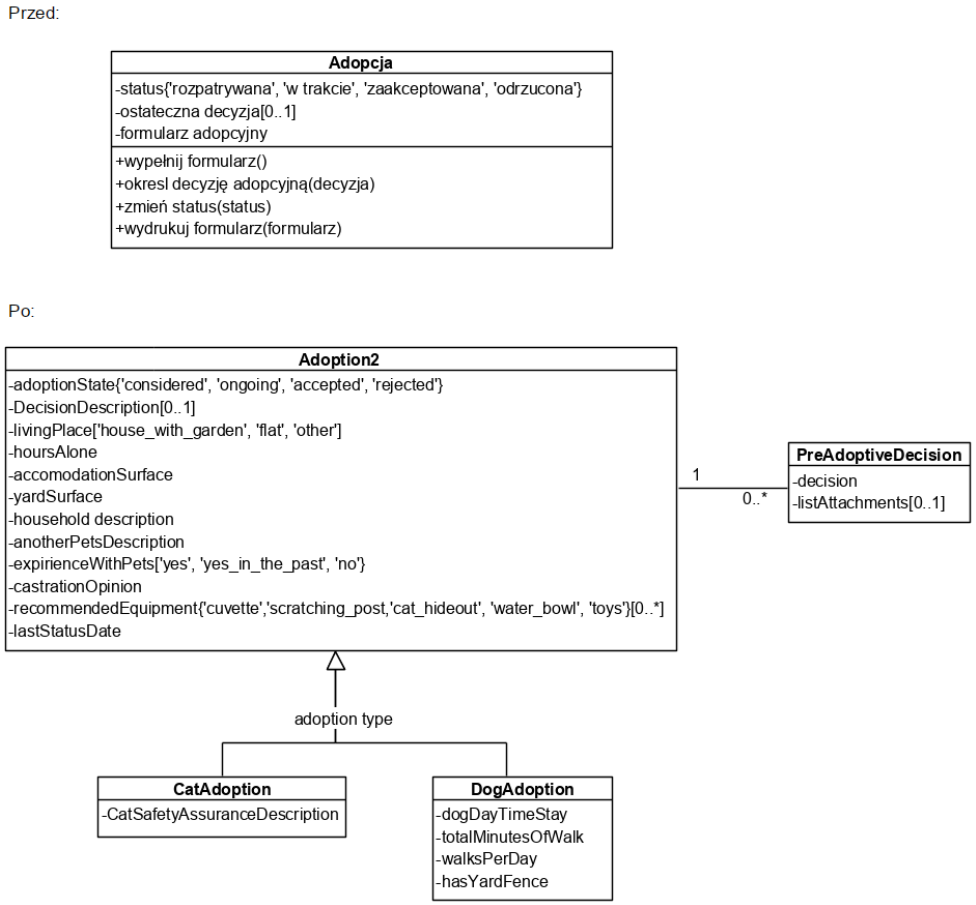
Rysunek 36. Porównanie fragmentu analitycznego oraz implementacyjnego diagramu klas: dodanie klasy PreAdoptiveDecision. Źródło: opracowanie własne.

- Dodano metodę *getLastActivityDate*() w klasie „Sponsor” i usunięto atrybut wyliczalny „data ostatniej aktywności” z tej klasy. Nie ma możliwości implementacji takiego typu atrybutu w języku Java.
- Dodano metodę *getAnimalAge()* w klasie „Animal” i usunięto atrybut wyliczalny „wiek” z tej klasy. Nie ma możliwości implementacji takiego typu atrybutu w języku Java.

4.11. Skutki analizy dynamicznej

Przeprowadzona analiza dynamiczna doprowadziła do powstania diagramów stanu oraz aktywności dla przypadków użycia. Na ich podstawie stwierdzono konieczność pewnej przebudowy modelu. Poniżej zaprezentowane zostały zmiany dokonane na diagramie klas wynikające z powyższej analizy.

- Przepływ diagramu aktywności dla przypadku użycia „Złóż wniosek adopcyjny” wykazał, że koncepcja umieszczenia formularza adopcyjnego jako e-dokumentu jest niewłaściwa. Taki dokument byłby uzupełniany, a następnie przechowywany przez system. Za niepotrzebne uznano ewentualne powtórne ładowanie go do strony w przypadku powrotu do wniosku roboczego przez użytkownika.



Rysunek 37. Porównanie fragmentu analitycznego oraz implementacyjnego diagramu klas: klasa Adopcja. Źródło: opracowanie własne.

Postanowiono więc, że rolę formularza przejmą pola interfejsu, które pobrane informacje przekażą od razu jako wartości atrybutów nowo tworzonego obiektu klasy „Adopcja” bez potrzeby tworzenia dokumentu. W sytuacji odwrotnej: w przypadku powrotu do wniosku zapisanego w systemie, GUI pobierać będzie dane z obiektu. Ponieważ adopcja dotyczyć może psów jak i kotów utworzono stosowne dwie klasy dziedziczące po klasie „Adopcja” zawierające atrybuty charakterystyczne dla danego zwierzęcia. Przebudowany model prezentuje Rysunek 37.

- Na podstawie diagramu aktywności "Dodaj aktywność medyczną" przeprowadzono analizę klasycznej wizyty weterynaryjnej. Badanie ścieżki postępowania lekarza weterynarii, której odwzorowaniem jest omawiany diagram wykazało, że diagram klas nie uwzględniał możliwości ustalenia i zapisania w kalendarzu kolejnej wizyty. Ponieważ wyznaczenie następnego terminu konsultacji jest, poza zakończeniem leczenia, naturalnym następstwem badania, wprowadzono do diagramu metodę *przypiszKolejnąWizytę(wizytaWeterynaryjna)* w klasie "Aktywność medyczna" realizującą w sposób wygodny i naturalny dla użytkownika harmonogramowanie kolejnej konsultacji.
- Na podstawie diagramu stanów klasy „Adopcja” został zmodyfikowany atrybut wyliczeniowy "status" tej klasy. Do tej pory istniały następujące kroki: "rozpatrywana", "zaakceptowana", "odrzucona". Podczas przeprowadzanej analizy stwierdzono potrzebę rozdzielenia etapu rejestracji wniosku adopcyjnego zakładanego przez stronę schroniska od bezpośredniego przekazania go do rozpatrzenia pracownikowi. Wstępnie nie uwzględniono tego pierwszego stanu. Obecne rozwiązanie zakłada, że należy wyróżnić dwa stany: uzupełnianie danych przed użytkownika (przed zapisem do bazy danych) i poprawne uzupełnienie i potwierdzenie rejestracji wniosku, co skutkuje zapisaniem do bazy i przesłaniem do pracownika w celu merytorycznej weryfikacji. Ten pierwszy stan został wydzielony i oznaczony jako "W trakcie". Podsumowując dostępne obecnie statusy to: "w trakcie", "rozpatrywana", "zaakceptowana", "odrzucona".
- Po analizie utworzonego diagramu stanów klasy „Zwierzę” uzupełniono atrybut „status” o dodatkowe warianty. Dotychczasowe opcje uwzględniały: „możliwy do adopcji”, „w klinice”, „czeka na odbiór”, „ocena zachowania”, „w trakcie leczenia”, „kwarantanna”. Przegląd przepływu diagramu stanów doprowadził do wykrycia luk w logice dostępnych statusów. Przede wszystkim brak sytuacji gdy zwierzę po prostu znajduje się w schronisku. Nie uwzględniono również sytuacji gdzie uznano podopiecznego za niezdolnego do zaadoptowania lub z jakiegoś powodu już nie znajduje się w ośrodku. Co prawda w klasie „Pobyt” znajduje się atrybut „powód zakończenia pobytu”, ale atrybut „status” w obecnym kształcie zaprzeczałby zawsze takiej sytuacji. Na tej podstawie dodano następujące pozycje: „odebrany”, „adoptowany”, „przyjęty”, „nie można adoptować”, „uciekł”, „nie żyje”.

4.12. Aplikacja mobilna

System będzie dostępny również jako aplikacja mobilna. Będzie można z niej korzystać na dwa sposoby: jako użytkownik niezalogowany oraz jako użytkownik zalogowany. W trybie bez logowania dostępne będą:

1) Ekran startowy.
   Pozwala wybrać schronisko dostępne na liście.
2) Strona główna/aktualności.
   Prezentuje 3 najnowsze artykuły na stronie schroniska, pozwala przejść do wybranego artykułu.
3) Nasze zwierzęta.
   Prezentuje listę zwierząt w wybranym schronisku.
4) Zagubione zwierzęta.
   Prezentuje ogłoszenia o zaginionych zwierzętach.

Po zalogowaniu się przez pracownika dostępne są:

1) Zarządzanie.
2) Dodaj zwierzę.
3) Terminarz.

4.13. Logo aplikacji

Przy pomocy narzędzia FreeLogoDesign [20] zostało stworzone logo aplikacji. Zaprojektowane logo przedstawia Rysunek 38.

Rysunek 38. Logo aplikacji @zor.

5. Technologie i narzędzia wykorzystane w pracy

W trakcie tworzenia projektu wykorzystano różnorodne technologie i narzędzia. Zostały one opisane w tym rozdziale.

5.1. Java 11

Java [21] została stworzona przez Jamesa Goslinga z firmy Sun Microsystems. Prace nad tym językiem rozpoczęto już w roku 1990, a rok później powstała jego pierwsza wersja pod nazwą Oak. Nazwa Oak, została wykorzystana już wcześniej przez inna firmę – Oak Technology i ostatecznie James Gosling wybrał nazwę Java. W 1995 r. na konferencji SunWorld w San Francisco Sun Microsystems przedstawił pierwszą publiczną wersję Javy. Pierwsza wersja zawierała 500 klas (kolejna w 1998 r. już 2300) [22]. Od tego momentu jej rozwój przebiegał błyskawicznie, a przyczynił się do tego dynamiczny rozwój Internetu. Nowe wersje pojawiały się coraz szybciej, a od wersji 9, nawet co pół roku. Obecnie dostępna jest wersja 13, udostępniona we wrześniu 2019 r. W naszym projekcie korzystamy z Java SE 11, z września 2018 r. Podstawy Javy oparte są na wcześniej istniejących językach jak C++ oraz Smalltalk, jednak jej twórcy chcieli stworzyć nowy pozbawiony wad poprzedników. Java jest wszechstronnym językiem wysokiego poziomu posiada wiele cech, które znacząco ułatwiają pracę z nim w porównaniu z innymi językami [21] [23]:

- Zarządzanie pamięcią – w Javie nie ma arytmetyki wskaźnikowej, nie ma więc możliwości odwoływania się do dowolnego obszaru pamięci, co bywa przyczyną błędów, ponadto występuje automatyczne usuwanie nieużywanych obszarów pamięci poprzez „garbage collection”.
- Ścisła kontrola zgodności typów – już na poziomie kompilacji kodu typy są kontrolowane, a korzystanie ze zintegrowanego środowiska programistycznego (IDE) sprawia, że na etapie pisania kodu sprawowany jest nadzór i możliwe są podpowiedzi oraz automatyczna poprawa błędów. Ponadto podczas rzutowania typów nie ma możliwości niewłaściwej konwersji.
- Wymuszona obsługa niektórych wyjątków – kompilator zmusza programistę do obsługi podstawowych błędów.
- Współbieżność – w Javie w łatwy sposób można stworzyć i zsynchronizować wątki działające równolegle.
- Obiektowość.
- Przenośność – program pisany na jednej platformie można uruchomić na dowolnej innej platformie.
- Obsługa aplikacji sieciowych – łatwość tworzenia programów, które mogą być otwierane w przeglądarkach internetowych i korzystać z usług internetowych. W Javie istnieją również

aplikacje serwerowe mające szerokie spektrum możliwości. Java posiada liczne biblioteki wspierające programowanie sieciowe w tym protokoły TCP/IP, HTTP.

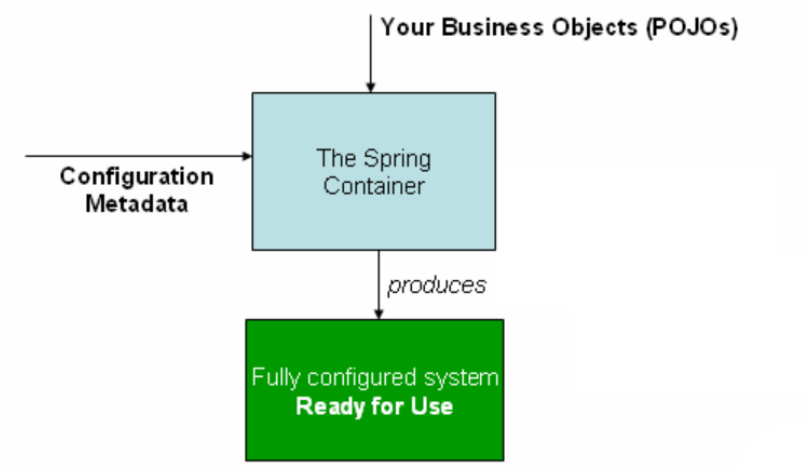
Java jest jednym z najpopularniejszych języków programowania. Od 2010 r. należy do firmy Oracle i, jak podaje ich strona, 97% komputerów lokalnych używanych w przedsiębiorstwach oraz 3 miliardy telefonów korzysta z oprogramowania Java. Jest ono rozwijane przez 9 milionów programistów [21] [24]. Powyższe cechy, a także doświadczenie naszego zespołu sprawiły, że wybraliśmy ten język do stworzenia naszej aplikacji. Ponadto Java posiada liczne frameworki, udostępniane na licencji GNU (podobnie jak Java) które umożliwiają szybsze i łatwiejsze tworzenie aplikacji. Opis wykorzystanych w niniejszej pracy frameworków znajduje się w kolejnych punktach.

5.2. Spring

Spring [25] jest platformą służącą do tworzenia aplikacji w języku Java. Za jego rozwój odpowiada firma Pivotal. Jego pierwsza wersja pojawiła się w 2003r., a w 2005r. został ściągnięty ponad 400 tys. razy [26]. Powstał jako odpowiedź na bardzo złożone standardy budowy aplikacji w J2EE. Spring udostępnia nam moduły, z których można wybrać sobie te, które są istotne do stworzenia konkretnej aplikacji. Do podstawowych elementów należą [26] [25]:

- Spring Framework,
- Spring Data,
- Spring MVC,
- Spring Security,
- Spring Boot,
- Spring Cloud.

Spring Framework dostarcza szerokiego zbioru funkcjonalności, które są podstawą większości współczesnych aplikacji. Jest to podstawowy moduł, w którym zawarte są gotowe rozwiązania, tak aby twórca aplikacji mógł skupić się na biznesowym aspekcie swojego projektu. Do bazowych funkcjonalności (Core Technolgies) należą: wstrzykiwanie zależności, obsługa zdarzeń, zarządzanie zasobami, lokalizacja językowa, walidacja, wiązanie danych, konwersja typów i Spring AOP (Aspect-Oriented Programming). Na szczególną uwagę zasługuje odwrócone sterowanie (IoC Container) w Springu występujące pod nazwą wstrzykiwanie zależności (dependency injection). Odwrócone sterowanie to przekazanie odpowiedzialności nad cyklem życia obiektów kontenerowi. Kontener tworzy obiekt (tzw. bean) ze wszystkimi zależnościami i konfiguracją korzystając z metadanych konfiguracyjnych zawartych w adnotacjach, pliku XML i kodzie. Wysokopoziomowy schemat działania kontenera IoC przedstawia Rysunek 39.



Rysunek 39. Kontener IoC. Źródło: [25]

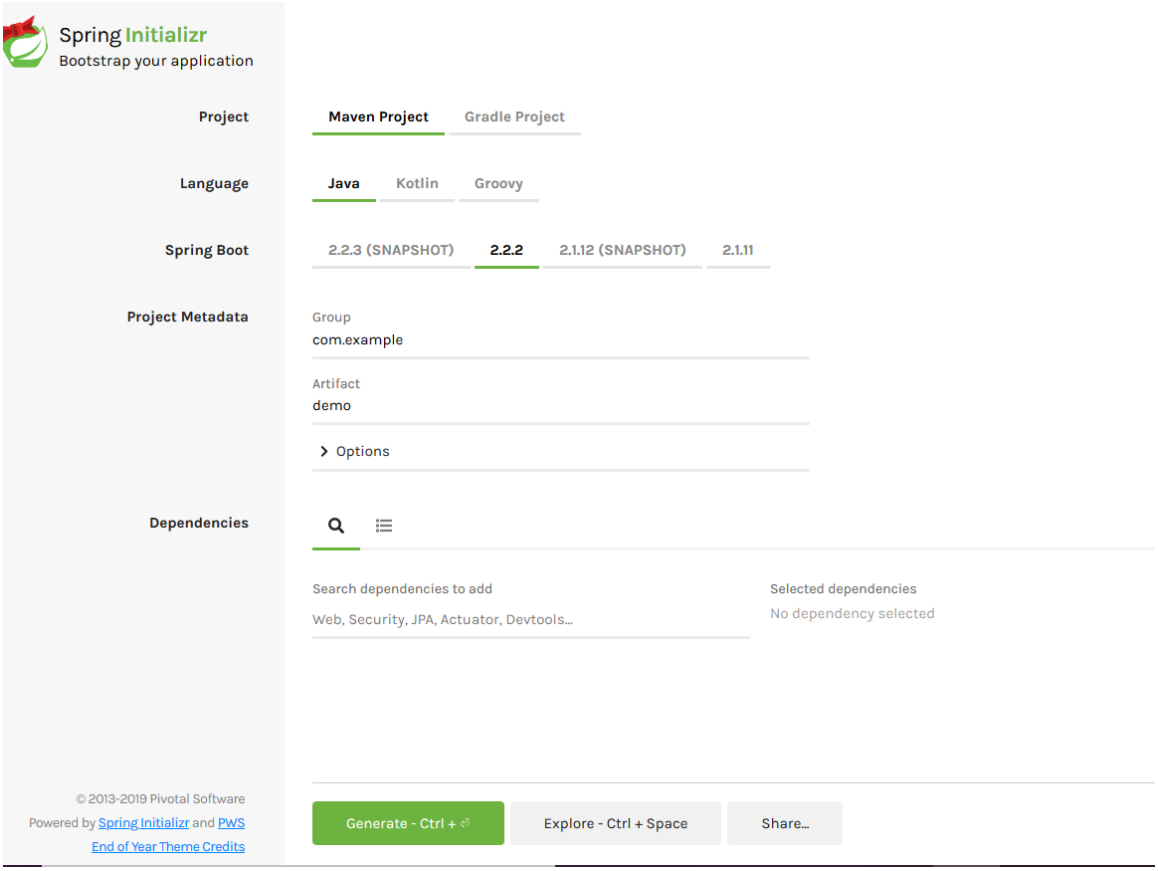
Spring Data ułatwia dostęp do danych znajdujących się w bazie danych, zarówno relacyjnej, jak i nie relacyjnej. Jego podstawowym założeniem jest zmniejszenie ilości powtarzającego się kodu. Podstawą każdego modułu typu Spring Data jest Spring Data Commons. Spośród licznych modułów przetwarzania danych mamy m.in. repozytorium wspierające standard JPA dostarczające gotowe interfejsy do wykonywania operacji bazodanowych (CrudRepository – metody dla operacji CRUD wraz z rozszerzeniami PagingAndSortingRepository i JPARepository).

Spring Web MVC (Model-View-Controller) jest używany przy tworzeniu aplikacji webowych. Jego centralną częścią jest Dyspozytor Servletu (DispatcherServlet), który dostarcza algorytm przetwarzający zapytania, podczas gdy właściwe zadania są wykonywane przez odpowiednio skonfigurowane komponenty. Servlet musi zostać zadeklarowany i odpowiednio zmapowany. Dzięki temu używając konfiguracji Springa wykrywa delegowane komponenty konieczne do realizacji różnych funkcjonalności. Spring MVC umożliwia mapowanie żądań poprzez korzystanie z odpowiednich adnotacji dedykowanych odpowiednim metodom: get, post, delete, put, patch oraz obsługę tych żądań i mogących wystąpić wyjątków. Alternatywa dla Spring MVC jest Spring WebFlux oparty na mechanizmach przetwarzania asynchronicznego i nieblokującego, w przeciwieństwie do technologii synchronicznego blokowania w MVC.

Spring Security to projekt służący zabezpieczeniu aplikacji. Pozwala na konfigurowanie kontroli dostępu: autentykacji i autoryzacji. Może zostać zintegrowane ze Spring MVC. Chroni przed atakami typu session fixation lub clickjacking [27].

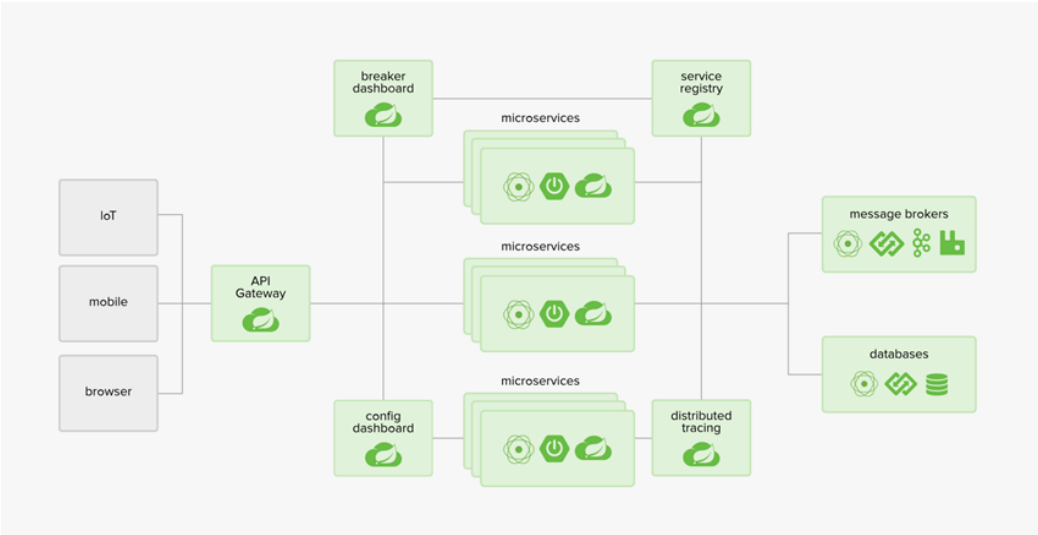
Spring Boot jest punktem startowym tworzenia aplikacji, to podstawowe narzędzie do szybkiej budowy projektu, posiada automatyczną konfigurację, a do wygenerowania szkieletu aplikacji wystarczy inicjalizator dostępny pod adresem https://start.spring.io/. Po wybraniu podstawowych danych dotyczących projektu (np. Maven/Gradle, język programowania, wersja) i podjęciu decyzji o zależnościach od razu mamy gotowy, działający projekt, który nie wymaga dodatkowej konfiguracji. Zależności podzielone są na kilka kategorii m.in.: Developer Tools, Web, SQL, NoSQL, Security,

Cloud, które umożliwiają szybki wybór potrzebnych nam funkcjonalności. Inicjalizator jest przedstawiony na Rysunku 40. W niniejszej pracy został użyty SpringBoot 2.2.



Rysunek 40. Inicjalizator Spring Boot. Źródło: [28]

Spring Cloud ma za zadanie uproszczenie budowy systemów rozproszonych. Celem jest zmniejszenie złożoności i mniejsze narażenie na błędy. Spring Cloud pomaga skoordynować działanie mikroserwisów. Dostarcza narzędzi do zarządzania serwisami i ich instancjami, obsługuje awarie, śledzi żądania i upraszcza odpytywanie zasobów w komunikacji opartej o http.



Rysunek 41. Architektura Spring Cloud. Źródło: [29]

Spring Cloud udostępnia także API Getaway, w którym tylko jeden serwis jest udostępniany, ale jego klientem mogą być zarówno strony web, aplikacje mobilne lub inne urządzenia. Ułatwia to sterowanie ruchem, jego monitoring, zarządzanie bezpieczeństwem i jest bardziej odporne na błędy poszczególnych serwisów. Rysunek 41 przedstawia schemat architektury Spring Cloud.

5.3. Hibernate

Hibernate ORM [30] jest mapperem obiektowo-relacyjnym, gotowym narzędziem pośredniczącym pomiędzy światem obiektowym a relacyjną bazą danych, służącym do transformacji danych z reprezentacji obiektowej na relacyjną. Pozwala na poradzenie sobie z wieloma problemami, jakie niesie za sobą utrwalenie danych, pierwotnie zdefiniowanych i zapisanych jako obiekty, w bazie relacyjnej przechowującej informacje w tabelach. Problem niezgodności impedancji powoduje konieczność brania pod uwagę różnic w obu światach, a w konsekwencji, nakłada ograniczenia na tworzony kod aplikacji. Jak pokazano w [31], [30], korzystając z relacyjnej bazy danych trzeba stale mieć świadomość, że:

- w bazie danych identyczność obiektów jest określana za pomocą klucza głównego, a w języku Java istnieją inne sposoby określania identyczności lub tożsamości obiektów;
- ilość klas w obiektowym modelu danych może być różna od ilości tabel, co jest związane m. in. ze stopniem szczegółowości (ziarnistości) danych w obu modelach lub brakiem dziedziczenia w modelu relacyjnym;
- asocjacje w modelu obiektowych są realizowane poprzez referencje, a w bazie danych poprzez klucze obce, co skutkuje innym zapisem relacji dwukierunkowych;
- nawigacja pomiędzy danymi w bazie jest oparta na łączeniu tabel poprzez polecenie JOIN, natomiast w Javie wymaga przechodzenia poprzez poszczególne obiekty.

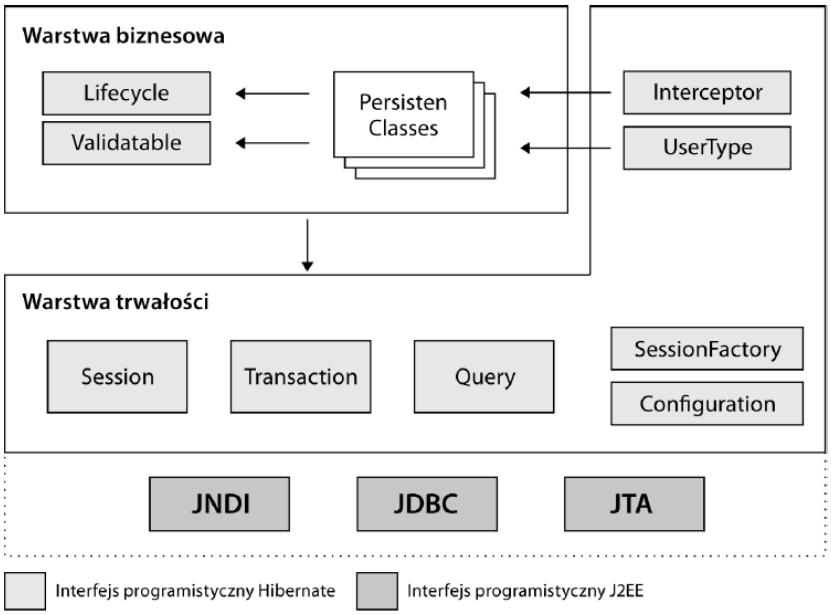
Mappery obiektowo-relacyjne mają na celu ułatwienie pracy z bazą danych i przede wszystkim są narzędziem, które tworzy i obsługuje komunikacje z bazą, minimalizując udział programisty w tym procesie.

Hibernate jest bezpłatnym narzędziem open-source, obecnie rozwijanym przez firmę Red Hat. Został stworzony przez Gavina Kinga w 2001 r., a prace były kontynuowane przez grupę jego współpracowników. W 2003r. wraz z firmą JBoss została wydana jego pierwsza komercyjna wersja [30]. Obecnie rozwijana jest wersja 6.0, a ostatnią stabilną jest 5.4.

Hibernate został napisany dla języka Java. W Javie istnieje interefejs JDBC umożliwiający bezpośrednią komunikację z bazą danych i choć daje on możliwość dużej kontroli nad komunikacją, to korzystanie mappera jest w wielu wypadkach wygodniejsze. Wykorzystanie Hibernate niesie za sobą wiele zalet [30] [32]. Dzięki niemu można zredukować ilość napisanego kodu i skupić się na biznesowej części aplikacji. Ponieważ warstwa aplikacji obsługująca cele biznesowe zostaje oddzielona od warstwy komunikującej się z bazą, kod jest czytelniejszy, a niezależność obu warstw sprawia, że wprowadzenie zmian w jednej z nich nie wymaga gruntownej przebudowy drugiej. Hibernate oferuje też mechanizmy

optymalizacji – dzięki temu ręczna optymalizacja jest konieczna tylko tam, gdzie wymaga tego specyfika aplikacji.

Założeniem Hibernate jest, aby biznesowa warstwa aplikacji i warstwa trwałości zostały od siebie oddzielone. Rysunek 42 przedstawia najważniejsze interfejsy z podziałem na warstwy.



Rysunek 42. Interfejsy programistyczne w Hibernate z uwzględnieniem warstwy biznesowej i warstwy trwałości. Źródło [33].

Konfiguracja Hibernate jest obsługiwana poprzez interfejs Configuration. Dane konfiguracyjne znajdują się w pliku *hibernate.properties* lub *hibernate.cfg.xml*. Plik XML poza parametrami konfiguracyjnymi zawiera także, opcjonalnie, lokalizację dokumentów odwzorowań.
Interfejs SessionFactory umożliwia utworzenie obiektu klasy Session, dla jednej bazy danych jest tylko jeden obiekt SessionFactory.

Najważniejszymi interfejsami z punktu widzenia komunikacji aplikacji z bazą danych, stanowiącymi przejście pomiędzy logiką biznesową a Hibernate są Session, Transaction i Query. Session tworzy sesję, a Query jest wykorzystywany do wykonywania operacji CRUD oraz wyszukiwania danych. Hibernate korzysta z HQL (Hibernate Query Language), podobnego do języka SQL, ale uwzględniającego obiektowość wraz jej specyfiką (np. dziedziczenie).
Lifecycle, Interceptor i Validatable mają na celu umożliwienie aplikacji reagowanie na zdarzenia, które zaszły wewnątrz Hibernate.

Mapowanie konstrukcji może być wykonane za pomocą adnotacji JPA (Java Persistence Annotations) [34]. Adnotacje mogą należeć do dwóch kategorii: opisujących model obiektowy oraz opisujących relacje między obiektami. Adnotacje są elementem kodu aplikacji i stanowią bardzo prosty sposób na zawarcie informacji o sposobie odwzorowania danych.

Hibernate stanowi skuteczne narzędzie umożliwiające programistom połączenie warstwy biznesowej i bazy danych. Ułatwia ono nie tylko komunikację i wykonywanie zapytań, również bardziej

złożonych, ale także stworzenie całej bazy danych na podstawie kodu zawartego w aplikacji. Jest to niezwykle przydatna cecha, zwłaszcza w projektach, w których programiści mają niewielkie doświadczenie i wolą skupić się na funkcjach, jakie ma spełniać aplikacja, niż rozwiązywać problemy związane z przechowywaniem danych lub np. optymalizacją komunikacji z bazą.

5.4. Maven

Apache Maven [35] jest narzędziem stworzonym w celu zarządzania projektem programistycznym. Jego historia sięga 2001 roku. Został stworzony przez Jasona van Zyl i był elementem projektu Jakarta Alexandria [36]. Znacząco ułatwia zarządzanie strukturą projektu, jego zależnościami, w tym również zależnościami przechodnimi, co sprawia, że nie ma konieczności samodzielnego ustalania, jakich bibliotek brakuje w projekcie, Maven automatycznie pobiera te, które są potrzebne i sprawdza ich wersje. Pilnuje właściwej kolejności budowania kolejnych modułów projektu oraz usuwa stare pliki pozostałe po poprzednim budowaniu aplikacji. Ponadto ułatwia tworzenie dokumentacji i jej testowanie. W niniejszym projekcie wykorzystywane jest środowisko programistyczne IntelliJ IDEA posiadające wbudowaną wersję Mavena, co zwalnia nawet z konieczności jego instalacji, a stworzenie projektu zarządzanego przez Mavena ogranicza się do kilku kliknięć. W ten sposób powstaje szkielet aplikacji z gotową strukturą katalogów. Typową strukturę przedstawia Rysunek 43.

```
|- src
|   \
|    |- main
|    |    \
|    |    |- java
|    |    |- resources
|    |- test
|          \
|          | - java
|- target
|- pom.xml
```

Rysunek 43. Typowa struktura katalogów w projekcie stworzonym w Maven. Źródło: opracowanie własne.

W katalogu main znajdują się dwa katalogi: java, w którym umieszcza się pliki z kodem źródłowym projektu i resources, zawierający pliki konfiguracyjne. W katalogu test znajdują się pliki związane z testami. Ponadto Maven tworzy jeszcze katalog target, w którym przechowuje pliki będące wynikiem jego działań. Ten folder pojawia się po pierwszym zbudowaniu projektu. Na samym końcu widoczny jest plik pom.xml (Project Object Model). Jest to plik pośredniczący między Mavenem a Javą. Znajdują się w nim wszystkie kluczowe informacje dotyczące projektu i jego konfiguracji. Rysunek 44

przedstawia plik pom.xml z minimalną konieczną ilością informacji. Znajdują się w nim informacje o wersji modelu, twórcy (groupId), nazwa aplikacji (artifactId) oraz wersji aplikacji (version).

```
1. <project>
2.   <modelVersion>4.0.0</modelVersion>
3.   <groupId>com.mycompany.app</groupId>
4.   <artifactId>my-app</artifactId>
5.   <version>1</version>
6. </project>
```

Rysunek 44. Plik pom.xml – wersja minimalna. Źródło: [37]

Praca Mavena opiera się na tzw. cyklu życia. Istnieją trzy cykle życia, podstawowy, to tzw. Build Lifecycle składający się z siedmiu faz [38].

- validate – walidacja poprawności kodu źródłowego i (opcjonalnie) innych elementów, sprawdzenie czy projekt można poprawnie skompilować,
- compile – kompilacja kodu źródłowego,
- test – automatyczne testowanie skompilowanego kodu,
- package – spakowanie skompilowanego kodu w np. w plik JAR lub
- verify – weryfikacja, czy nasza paczka spełnia kryteria jakościowe, jest poprzedzona testami integracyjnymi,
- install – instalacja paczki w lokalnym repozytorium, tak aby była dostępna dla innych modułów,
- deploy – umieszczenie w zdalnym repozytorium.

Pozostałe dwa cykle to:

- clean – usuwa poprzednie pliki z budowy aplikacji,
- site – budowanie dokumentacji.

Maven jest niezwykle użytecznym narzędziem, przyspieszającym pracę nad tworzenie aplikacji. Użycie go nawet w niewielkim projekcie, takim jak ten znacząco ułatwia jego stworzenie.

5.5. Serwer bazy danych MySQL

MySQL [39] jest systemem zarządzania relacyjnymi bazami danych (Relational Database Management System- RDBMS) rozwijanym przez firmę Oracle, a wcześniej przez Sun Microsystems i MySQL AB. Jest dostępny w wersji open source oraz komercyjnej. Obecnie dostępna jest wersja 8.0. Producent podaje, że jest to najpopularniejsza baza danych, z której korzystają takie firmy jak Facebook, Google, czy Adobe [40]. Serwer MySQL jest systemem wielowątkowym opartym na architekturze

klient – serwer, gdzie zadaniem serwera jest udostępnianie danych, wykonywanie instrukcji, zarządzanie dostępem do danych. Ma zapewnić jak najszybszy dostęp wielu użytkownikom jednocześnie. Połączenie z serwerem następuje poprzez protokoły TCP/IP, ODBC lub JDBC. Jest szybki w działaniu, niezawodny i bezpieczny, co istotne łatwo się go używa, więc nawet nie bardzo doświadczony użytkownik może z nim pracować [39].

5.5.1. Relacyjny model danych

MySQL został zaprojektowany w oparciu o relacyjny model danych, którego twórcą jest Edgar Codd [41]. Jego postulaty zostały opublikowane w 1970 r. i są podstawą, z której nadal czerpią współczesne, relacyjne bazy danych. Model relacyjny jest oparty na pojęciu relacji pochodzącym z matematyki oznaczający podzbiór iloczynu kartezjańskiego. W bardziej potocznym rozumieniu relacja jest reprezentowana przez tabelę, będącą podstawowym elementem bazy danych. W takiej bazie wszystko jest zapisane w tabeli i nie ma możliwości, zapisania informacji w inny sposób. Każda tabela (relacja) charakteryzuje się tym, że wiersze są unikalne, kolejność atrybutów (również unikalnych) jest dowolna, a kolumny mają niepodzielne, atomowe wartości. Tabele są identyfikowane poprzez niepowtarzalną nazwę, swoje nazwy posiadają także kolumny – unikalne w tabeli, ale mogące się powtarzać w innych tabelach. Każdy wiersz w jest identyfikowany poprzez identyfikator zwany kluczem głównym. W ten sposób poprzez nazwę tabeli, kolumny i klucz główny możemy dotrzeć do każdej pojedynczej danej w bazie. Klucz główny jest też używany do identyfikowania danego rekordu w innej tabeli – wtedy występuje pod nazwą klucz obcy.

MySQL jest oparty na języku SQL, służącym do kierowania zapytań i poleceń do bazy danych. Język został stworzony przez IBM Corporation w latach siedemdziesiątych pod nazwą SEQUEL (Structured English Query Language), potem zmieniono nazwę na SQL. Standardy języka SQL są wyznaczane przez American National Standards Institute (ANSI) oraz International Organization for Standardization (ISO), ich ostatnia wersja pochodzi z 2008r [41].

5.5.2. Język SQL

Zapytania w języku SQL mogą należeć do jednej z czterech grup: Data Manipulation Language (SQL DML), Data Definition Language (SQL DDL), Data Control Language (SQL DCL) i Data Query Language (SQL DQL).

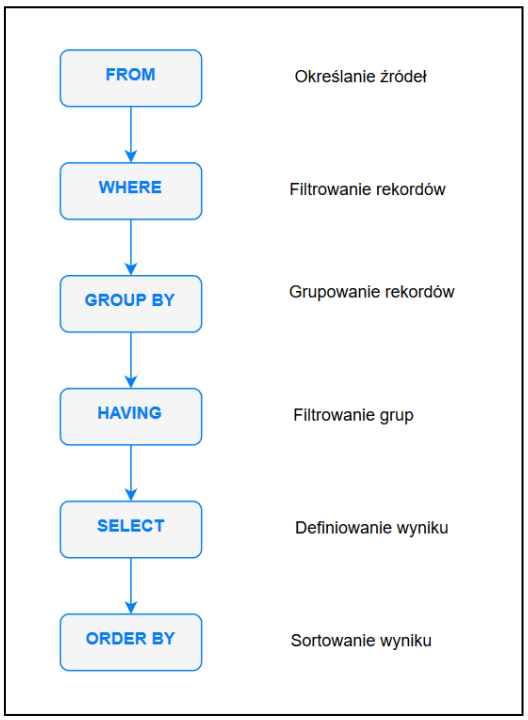
- Data Manipulation Language – służy do tworzenia rekordów w bazie oraz ich modyfikacji i usuwania. Podstawowe polecenia to INSERT, UPDATE i DELETE.
- Data Definition Language – służy do tworzenia, modyfikacji i usuwania obiektów w bazie (np. tabel). Podstawowe polecenia to CREATE, ALTER i DROP.
- Data Control Language – służy do zarządzania uprawnieniami. Podstawowe polecenia to GRANT, REVOKE i DENY.

- Data Query Language służy odczytywaniu danych z bazy, podstawowa składnia zapytań wymaga dwóch elementów: SELECT oraz FROM, inne klauzule są opcjonalne. Podstawowe klauzule używane w języku SQL są przedstawione w Tabeli 16.

Tabela 16. Podstawowe klauzule języka SQL. Źródło: opracowanie własne.

Klauzula	Cel użycia
FROM	Wskazuje źródłowe tabele, z jakich mają zostać pobrane dane.
WHERE	Określa kryteria według których zostają przefiltrowane wybrane wiersze z tabel.
ORDER BY	Sortuje wiersze na podstawie wybranych kolumn.
GROUP BY	Wskazuje sposób grupowania wyników.
HAVING	Wskazuje kryteria filtrowania pogrupowanych wierszy.

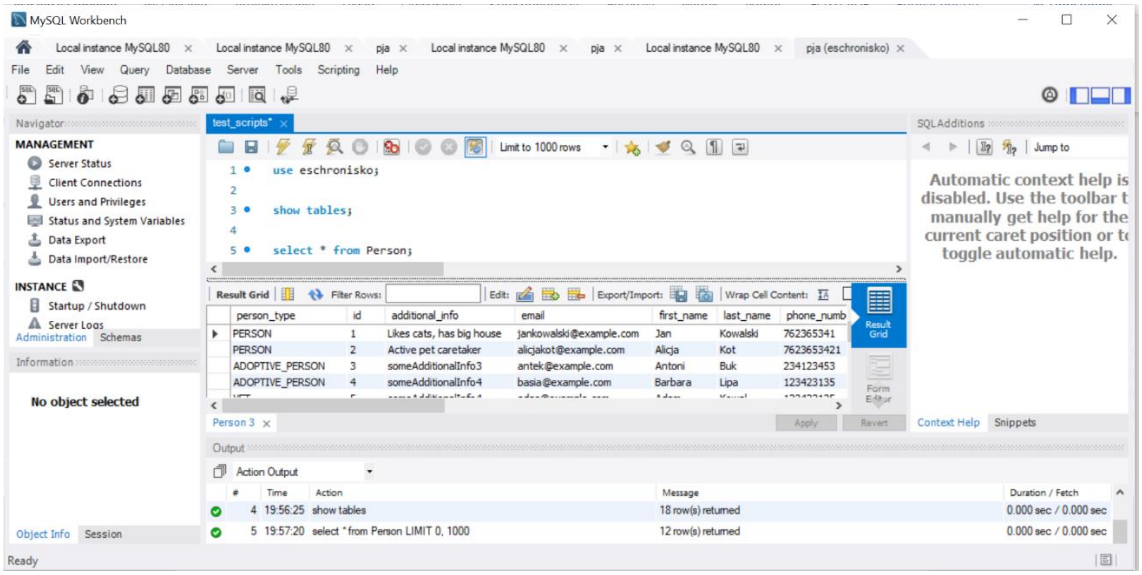
Składnia zapytania nie jest dowolna, klauzule muszą się pojawiać w odpowiedniej kolejności. Również wykonywanie klauzul odbywa się w ściśle określonej kolejności, identycznej dla wszystkich relacyjnych baz danych. Kolejność przedstawiona jest na Rysunku 45.



Rysunek 45. Kolejność wykonywania klauzul w zapytaniu SELECT. Opracowanie własne na podstawie [42].

5.5.3. MySQL Workbench

W pracy nad projektem inżynierskim zostało wykorzystane narzędzie MySQL Workbench [43]. Jest to narzędzie posiadające graficzny interfejs użytkownika umożliwiające modelowanie danych, tworzenie oraz zarządzanie bazą danych. Pozwala na tworzenie, wykonywanie i optymalizację zapytań SQL. W formie wizualnej pokazuje tabele, widoki i inne elementy zawarte w bazie danych. Wygląd edytora SQL w MySQL Workbench przedstawia Rysunek 46.



Rysunek 46. MySQL Workbench - edytor SQL. Źródło: opracowanie własne.

5.6. HTML5 oraz CSS

HTML oraz CSS to dwa odrębne języki służące do tworzenia stron internetowych, jednakże stosowane są razem, gdyż uzupełniają się i mają inną rolę do spełnienia. HTML odpowiada za strukturę treści i jej znaczenie np. czy jest to nagłówek, paragraf czy zdjęcie. CSS nadaje styl np. za pomocą kolorów i rodzajów czcionek. Ułatwia także adaptację sposobu wyświetlania strony do typu urządzenia, np. monitorów stacjonarnych czy smartfonów. Co do zasady kod pisany w tych językach, mimo że dotyczy tego samego dokumentu, powinien być trzymany oddzielnie, tj. nie należy umieszczać CSS wśród znaczników HTML i odwrotnie.

5.6.1. HTML

HTML (ang. HyperTextMarkup Language) [44] to hipertekstowy język znaczników używany obecnie do tworzenia stron internetowych. Jego początki sięgają lat osiemdziesiątych ubiegłego stulecia, gdy fizycy pracujący dla ośrodka naukowo-badawczego CERN rozważali sposoby udostępniania danych poprzez sieć Internet. Pierwszy szkic tego języka został opublikowany w 1993 roku. W późnych latach dziewięćdziesiątych wprowadzono Kaskadowe Arkusze Stylów (CSS), czyli język nadający treści styl, layout czy kolory. W tym samym okresie, gdy CSS został zaimplementowany w większości przeglądarek, zalecono by twórcy stron internetowych oddzielali kod odpowiadający za zawartość i

strukturę strony od tego odpowiedzialnego za jej wygląd. HTML służy do tego pierwszego celu. [45] Obecnie obowiązującą wersją, udostępnioną w październiku 2014 roku i rekomendowaną przez World Wide Web Consortium (W3C) jest HTML5. W3C jest organizacją zajmującą się rozwojem i standaryzacją pisania i przesyłania stron WWW [46].

Pomysł twórców HTML opierał się na wykorzystaniu hipertekstów, tj. organizacji danych w sposób niestrukturalny, do których dostęp uzyskuje się za pomocą hiperłączy. Język ten ponadto udostępnia m.in. znaczniki: za ich pomocą opisuje się strukturę i kształt informacji zawartych na stronie.
W skład języka HTML wchodzą następujące komponenty:

- znaczniki,
- typy danych,
- referencje znakowe,
- odwołania w postaci encji,
- deklaracje typu dokumentu.

Znaczniki służą do wskazania znaczenia poszczególnych elementów strony np.: tekstu, obrazu, menu czy listy. Definiuje to sposób ich wyświetlenia w przeglądarce internetowej. Początek znacznika oznacza się: <>, np. <body>, a koniec: </> np. </body>. Wszystko, co znajduje się w środku traktowane jest tak, jak zostało to dla danego elementu opisane. Cały dokument HTML składa się właśnie z takich komponentów, które zostały hierarchicznie zagnieżdżone [47]. Przykład zagnieżdżonego znacznika <title> prezentuje Rysunek 47.

```
<head>
  <title>The Title</title>
</head>
```

Rysunek 47. Przykład zagnieżdżonych znaczników HTML. Źródło: [48].

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <title>Sample page</title>
 </head>
 <body>
  <h1>Sample page</h1>
  <p>This is a <a href="demo.html">simple</a> sample.</p>
  <!-- this is a comment -->
 </body>
</html>
```

Rysunek 48. Przykład bardzo prostego dokumentu HTML przedstawiający strukturę strony. Źródło: [49]

Typy danych wskazują z jakim rodzajem elementu użytkownik ma do czynienia i co on opisuje, np. data, adres URI, kolor, wymiar strony. Typy wprowadzane są jako wartości elementów lub atrybutów.

Encje oraz referencje znakowe umożliwiają zapisanie określonych znaków za pomocą specjalnych stałych. Od HTML4 istnieją 252 encje, np. " oznacza cudzysłów. Deklaracja typu dokumentu (DTD) służy poprawnej walidacji utworzonej strony i zazwyczaj od tego zaczyna się dokument HTML. Jest to sekcja <DOCTYPE>. Rysunek 48 przedstawia kod prostej strony internetowej z podstawowymi znacznikami oraz DTD.

5.6.2. CSS

Kaskadowe Arkusze Stylów (ang. Cascading Style Sheets, CSS) [50] opisują wygląd dokumentu HTML. W3C przejęło pracę nad CSS od pierwotnego twórcy: Håkona Wium Lie i pod koniec 1996 roku została wydana pierwsza oficjalna dokumentacja. Jak wspomniano, CSS został stworzony by oddzielić prezentację strony WWW od jej zawartości. Pierwotnie to odpowiednie znaczniki w HTML pozwalały ustalać żądaną typografię czy kolory. Jednakże ze względu na brak jednego standardu, strony mogły wczytywać się w różny sposób, niekoniecznie prawidłowy, w różnych przeglądarkach internetowych. Obecnie najnowszą wersją CSS są moduły czwartego poziomu (czasem błędnie zwane CSS4), gdzie usunięto pewne ograniczenia dotyczące Selektorów, a także rozwinięto popularny od CSS3 moduł Flexbox [51].

W arkuszach stylów kod prezentowany jest w postaci: właściwość – dwukropek – wartość, co przedstawia Rysunek 49, gdzie przypisywane są cechy (np. kolor, zdjęcie w tle) elementowi strony o nazwie <body> w dokumencie html [50].

```
body {
   overflow: hidden;
   background-color: #000000;
   background-image: url(images/bg.gif);
   background-repeat: no-repeat;
   background-position: left top;
}
```

Rysunek 49. Przykład kodu CSS. Źródło: [52].

Używanie CSS ma wiele zalet. Poza oddzieleniem kodu opisującego widok strony od jej zawartości, CSS umożliwia ustawianie globalnych wartości dla różnych elementów dzięki dziedziczeniu i kaskadowości. Toteż zmieniając jedną cechę dla określonego znacznika można dokonać modyfikacji więcej niż jednego fragmentu strony, a nie powtarzać tę samą czynność w wielu miejscach w kodzie, co usprawnia pracę i ułatwia utrzymanie spójności oraz estetyki.

Decyzją naszego zespołu technologie HTML oraz CSS zostały wybrane jako podstawowe narzędzia tworzenia front-endu naszej aplikacji ze względu na ich dojrzałość, popularność i powszechne stosowanie przez twórców programów webowych. Istotny jest także wysoki poziom znajomości tych rozwiązań przez członków zespołu i wsparcie techniczne oferowane przez rozbudowaną dokumentację oraz duże środowisko web deweloperów.

5.7. JavaScript (JS)

Skryptowy język programowania JavaScript [53] jest trzecią, obok HTML i CSS, najbardziej popularną technologią służącą do budowy stron WWW [51]. Rysunek 50 przedstawia role HTML, CSS i JS w tworzeniu stron internetowych.

Rysunek 50. Schemat wykorzystywania HTML, CSS oraz JavaScript przy tworzeniu aplikacji webowych. Źródło: [54].

JS został zaproponowany przez Brendana Eicha w 1995 roku, a standardem uznanym przez European Association for Standardizing Information and Communication Systems stał się w roku 1997. ECMA Script to oficjalna nazwa tego języka.

JS to wysokopoziomowy, skryptowy (interpretowany lub kompilowany metodą JIT), pełnoprawny język posiadający wszystkie klasyczne konstrukcje (takie jak pętle, zmienne, klasy, warunki itd.). Główne jego cechy to dynamiczne typowanie, obiektowość, a także traktowanie funkcji jako obiektów pierwszej kategorii (ang. first-class functions), co umożliwia ich przechowywanie w zmiennych jako referencje czy przekazywanie do innych obiektów. Przykład użycia funkcji jako obiektów pierwszej kategorii przedstawia Rysunek 51 [55], [53].

```
function counter() {
    let count = 0;

    return function() {
        return ++count;
    };
}

let closure = counter();
closure(); // returns 1
closure(); // returns 2
closure(); // returns 3
```

Rysunek 51. Funkcje jako obiekty pierwszej kategorii/domknięcia [56].

Mnogość bibliotek JS wspiera twórcę udostępniając mu wygodne narzędzia służące do poprawy wyglądu strony, bezpieczeństwa czy tworzenia nowych ciekawych funkcjonalności. Jego ogromną zaletą jest możliwość zamiany elementu kodu HTML lub CSS bez konieczności ponownego łączenia się z serwerem [57]. Skrypty JS pozwalają na tworzenie interaktywnych stron, dynamicznych

podpowiedzi, a także walidację formularzy oraz generowanie alertów od strony przeglądarki. Przykładem wzbogacenia aplikacji dzięki użyciu JS mogą być ciekawe slidery, animacje, pop-upy czy karuzele. Kilka linijek kodu zapewnia wstępny stopień zabezpieczenia przed błędnymi danymi uzupełnionymi w formularzach oraz wyświetlenie ostrzeżeń czy informacji dla użytkownika. Rysunek 52 przedstawia kod dokumentu html wraz z użyciem JavaScript w celu zmiany treści strony internetowej.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <button id="hellobutton">Hello</button>
    <script>
        document.getElementById('hellobutton').onclick = function() {
            alert('Hello world!');                    // Show a dialog
            var myTextNode = document.createTextNode('Some new words.');
            document.body.appendChild(myTextNode);    // Append "Some new words" to the page
        };
    </script>
  </body>
</html>
```

Rysunek 52. Kod prostej strona z wykorzystaniem JS. Źródło: [56].

JavaScript, ze względu na swoją popularność, doczekał się wielu dodatkowych bibliotek, co jeszcze bardziej wzmocniło pozycję i możliwości tego języka. Poniżej omówiono kilka najbardziej popularnych:

- Angular – istnieje od 2009, gdy został wydany przez Google. Wspomaga szybkie i łatwe budowanie aplikacji jednocześnie na komputery stacjonarne jak i urządzenia mobilne;
- jQuery – wspiera tworzenie aplikacji webowych; dzięki niej zmniejsza się ilość linijek i upraszcza się kod, dba jednocześnie by był kompatybilny z różnymi przeglądarkami;
- Node.js – to platforma ułatwiająca napisanie aplikacji po stronie serwera w JS; bardzo szybko wykonuje działania i jest łatwo skalowalna.

W aplikacji tworzonej przez nasz zespół JavaScript został zastosowany jako doskonałe uzupełnienie HTML oraz CSS, gdyż te technologie bardzo dobrze ze sobą współpracują. JS wzbogaca funkcjonalności naszej strony oraz wspomaga połączenie z serwerem [58].

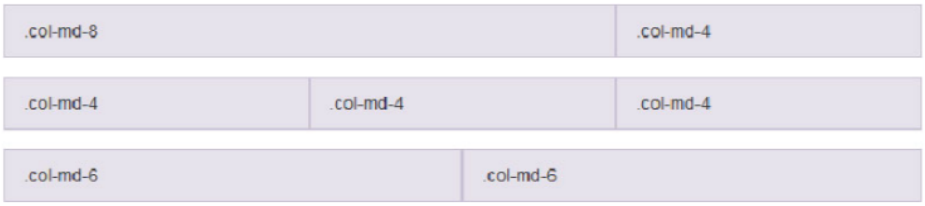
5.8. Bootstrap

Bootstrap [59] to jedna z najbardziej popularnych bibliotek CSS używanych obecnie do tworzenia responsywnych projektów w HTML, CSS oraz JavaScript. Powstała na Twitterze w 2011 roku. Działa obecnie na otwartej licencji oprogramowania. Dzięki temu frameworkowi deweloper oszczędza czas konieczny do zbudowania nowocześnie wyglądającej strony.

Wyróżniającym elementem Bootstrapa jest znaczące korzystanie z jQuery. Bez tego dodatku trudno byłoby zapewnić przenośność, a kod w JavaScript byłby zbyt rozbudowany. Bootstrap stosuje się w dowolnym IDE czy edytorze wraz z językami używanymi po stronie serwera jak ASP.NET, PHP czy Ruby [60].

Programista za pomocą Bootstrapa może korzystać z gotowych wzorów dla każdego elementu wyglądu strony: typografii, przycisków czy do pinów karuzelowych. Jest to narzędzie, które wspiera kreację zarówno prototypów projektów jak i całego programu od początku do końca. Omawiana biblioteka składa się z różnych narzędzi odciążających tworzącego aplikację od budowania każdego elementu samodzielnie od zera za pomocą gotowych struktur oraz powtarzalnego kodu. Dzięki niej prościej buduje się responsywne strony. Można także wykorzystać ją jedynie do tworzenia front-endu, co zapewni poprawnie działanie strony na różnych typach przeglądarek [59] [61].

Dla programisty jedną z najważniejszych korzyści stanowi intuicyjny web-designer. System gridów, które można dowolnie i wygodnie modelować, widoczny na Rysunku 53, ułatwia budowanie strony, a duża ilość gotowych komponentów zapewnia użyteczność i elastyczność wyboru.

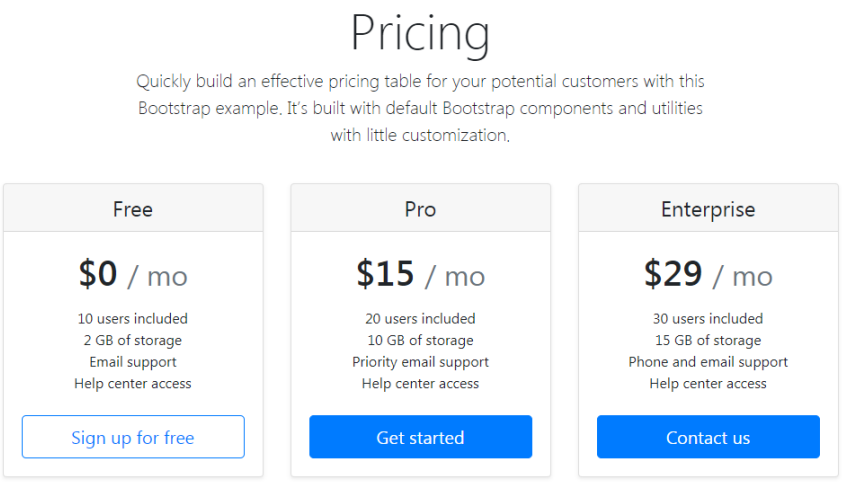


Rysunek 53. Przykład układu kolumn i rzędów w Bootstrap. Źródło: [62].

Kolejną zaletą Bootstrapa jest obsługa większości przeglądarek. Ponadto aplikacja zawiera kod automatycznie dopasowujący wielkość obrazów przy użyciu gotowych formuł CSS bez konieczności żmudnego zmieniania ich rozdzielczości. Deweloper może sam zadecydować, które z wielu dostępnych w bibliotece rozszerzeń będzie używał w swoim projekcie, dzięki czemu jest on szyty na miarę. Komponenty, które programista może wybrać do stworzenia aplikacji przedstawia Rysunek 54.

Rysunek 54. Komponenty Bootstrapa. Źródło: [62].

Bootstrapa łatwo zintegrować z innymi platformami i bibliotekami CSS, a ponadto zawiera wbudowane komponenty z gotowymi wzorcami różnych elementów strony jak menu, lista lub przygotowanych pod konkretny typ działalności. Przykład predefiniowanego wzoru strony z opcjami abonamentów przedstawia Rysunek 55. Dzięki temu projektowanie i budowa strony jest prostsza i szybsza [59].



Rysunek 55. Przykład gotowego komponentu w Bootstrap. Źródło: [63].

Powyższe zalety spowodowały, że Bootstrap wyróżnił się spośród innych dostępnych narzędzi i obecnie stosowany jest przez wielu web deweloperów, a przez to skupia wokół siebie liczne środowisko użytkowników. Obecnie Bootstrap ma ponad 10.000 commitów na platformie GitHub. Opisane wyżej zalety zdecydowały o wyborze tej biblioteki przy tworzeniu naszej aplikacji.

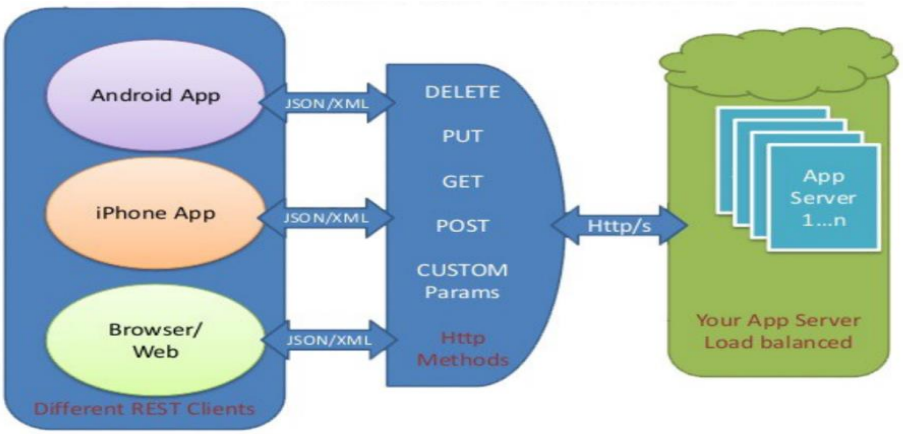
5.9. REST API

REST API [64] oraz metody jego implementacji zostały opisane w 2000 roku przez Roy'a Fieldinga w rozprawie doktorskiej. Idea projektu zakłada wykorzystywanie gotowych protokołów, bez dodawania kolejnych rozwiązań. Przykładowo korzystając z HTTP w komunikacji między urządzeniami nie ma konieczności instalacji i stosowania niczego nowego [64]. Rysunek 56 przedstawia ideę REST API w formie graficznej.

REST to nie kolejny framework, a zbiór zasad programowania oraz stylu architektury, który opiera się na pewnych dobrych praktykach dotyczących komunikacji pomiędzy programami. Składa się na to sześć elementów [65]:

- architektura typu Klient – Serwer,
- Cache,
- Stateless,
- Uniform Interface,

- Layered System,
- Code on Demand.



Rysunek 56. Architektura REST API. Źródło: [66].

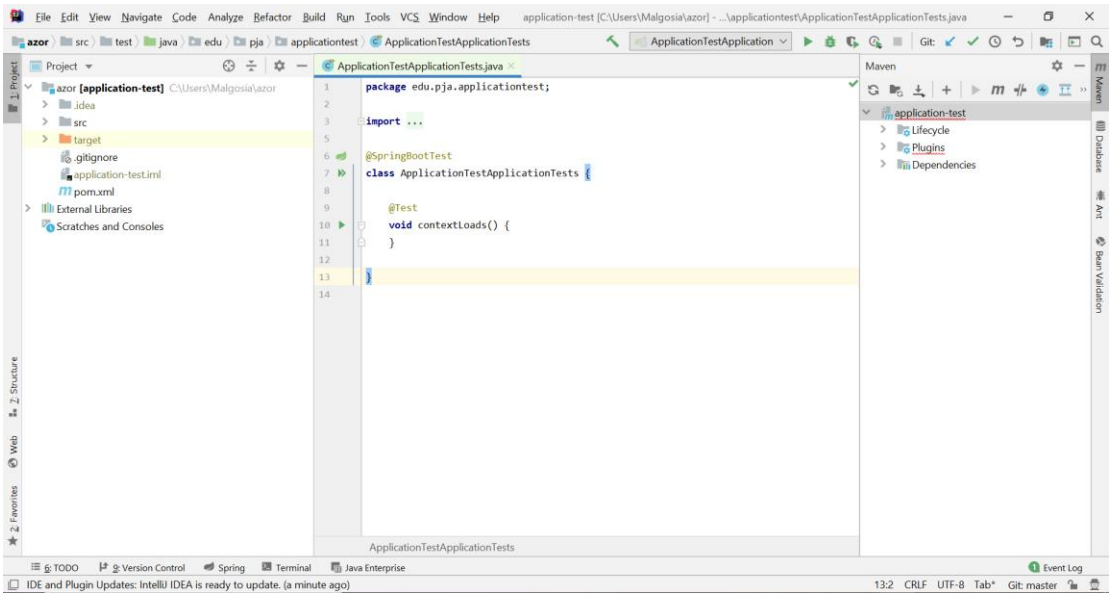
Spełniając je REST API cechuje się niezależnością strony klienckiej od strony serwera. Dzięki temu żadna nie może dokonywać ingerencji w drugą, a zmiany wprowadzone w jednej ze stron są dla przeciwnej bez znaczenia. Bezstanowość jest natomiast gwarancją, iż każde zapytanie od klienta jest kompletne i całkowicie niezależne od innego, a dane nie są przechowywane przez serwer. Wpływa to znacząco na poprawę niezawodności. Przewidziane jest także buforowanie po stronie klienta, a każda odpowiedź, którą otrzymuje ma wskazywać, czy informacje mają być cachowane czy też nie warto tego robić. Cachowanie informacji rzadko zmieniających stan zmniejsza ilość koniecznych połączeń z serwerem. By zwiększyć bezpieczeństwo REST API wprowadzono także koncepcję separacji warstw, tj. oddzielenia poszczególnych elementów oraz sposobu ich prezentacji. Dzięki temu atak na jedną warstwę nie oznacza dostępu do informacji zawartych gdzie indziej. Ostatni omawiany element: code on demand to nieobowiązkowe dodatkowe założenie, że można zmieniać czy uzupełniać kod aplikacji poprzez pobranie kolejnego pakietu z serwera.

Podsumowując, REST API zapewnia dużą uniwersalność i możliwość wykorzystania jednego API do różnych typów odbiorców mobilnych i stron internetowych. Dane otrzymywane są w powszechnie stosowanym i wygodnym formacie JSON, a opisane powyżej punkty zapewniają szybszą, bardziej niezawodną i bezpieczniejszą pracę. Łatwo także o testowanie endpointów, np. przy pomocy narzędzia o nazwie Postman. Dlatego też skorzystaliśmy z tego API przy produkcji naszej aplikacji.

5.10. IntelliJ IDEA

W pracy zostało wykorzystane środowisko programistyczne IntelliJ IDEA [67]. Jest to produkt czeskiej firmy JetBrains rozwijany od 2001r. dedykowany językowi Java. Występuje w dwóch wersjach: Ultimate (tę wersję można zobaczyć na Rysunku 57) oraz Community, która jest wersją darmową. Narzędzie wspiera różne języki oraz umożliwia korzystanie z popularnych technologii i

frameworków. IntelliJ IDEA wspiera zarówno języki zaprojektowane do wykonywania na Wirtualnej Maszynie Javy jak Java, Kotlin, Groovy, Scala, ale także i inne np. JavaScript, TypeScript lub SQL.



Rysunek 57. IntelliJ IDEA wersja Ultimate. Źródło: opracowanie własne.

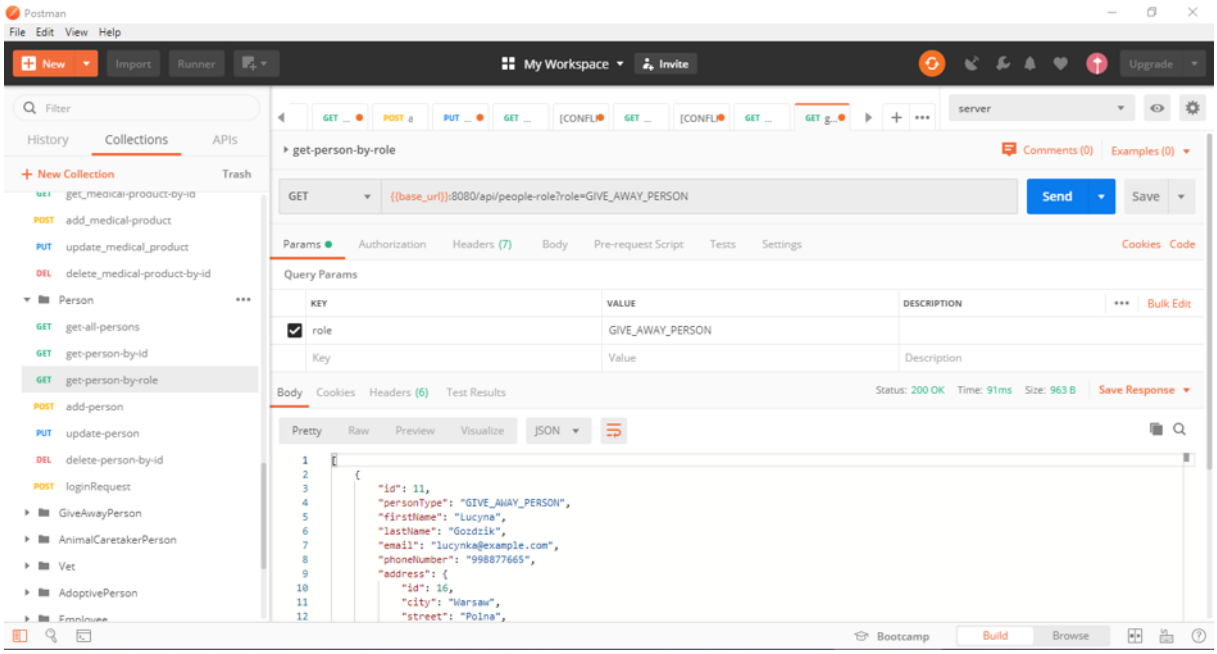
IntelliJ IDEA zawiera wiele wbudowanych funkcji ułatwiających rozwijanie aplikacji [67]:

- Kontrolowanie wersji. Program posiada jednolity interfejs do kontrolowania wersji za pomocą różnych narzędzi np. Git, SVN, Mercurial. Ułatwia też wyszukiwanie historii zmian i zarządzanie konfliktami pomiędzy wersjami kodu;
- Integracja z narzędziami do budowy aplikacji. Wspiera m.in. Maven, Gradle, Ant;
- Wykonywanie testów za pomocą np. JUnit, TestNG, Spock, ScalaTest;
- Dekompilator;
- Wsparcie narzędzi bazodanowych umożliwiające połączenie i pracę z bazą danych;
- Współpraca z serwerami aplikacji np. Tomcat, JBoss.

W niniejszej pracy szczególnie istotne jest obsługiwanie Javy i JavaScript, Springa, Hibernate oraz Maven, które stanowią podstawę opisywanego projektu inżynierskiego.

5.11. Postman

Postman [68] umożliwia wygodne testowanie API oraz wysyłanie żądań. Program powstał w 2012 roku i został udostępniony bezpłatnie w Chrome Web Store. Przewagę aplikacji stanowi zebranie wielu przydatnych funkcjonalności w ramach jednego środowiska. Dzięki niej łatwo tworzy się żądania http (GET, POST, PUT, PATCH), przechowuje środowiska i zapytania umożliwiając ich ponowne użycie czy konwersję API do kodu dla różnych języków programowania [69]. Obecnie z Postmana korzysta ponad dziesięć milionów użytkowników i pięćset tysięcy firm [68].



Rysunek 58. Testowanie metody w Postmanie. Źródło: opracowanie własne.

Zaletą wyróżniającą Postmana jest szybkie testowanie żądań REST oraz SOAP wprost z aplikacji. Ponadto program pozwala na zautomatyzowanie testów manualnych i integrację ich z własnymi przepływami CI/CD (zbiór zasad oraz wytycznych i dobrych praktyk dotyczących pracy nad projektami programistycznymi). Wygodnie również można symulować endpointy i ich odpowiedzi bez użycia serwera. Postman umożliwia zautomatyzowane generowanie dokumentacji stworzonego API. Poza podstawowym wysyłaniem żądań, można zapisywać je i dzielić się nimi z zespołem. Stworzone projekty synchronizują się automatycznie co umożliwia pracę na kilku komputerach jednocześnie. Dostępna jest również funkcja eksportowania środowiska oraz plików typu JSON, tak aby cały zespół miał do nich łatwy dostęp. Program jest pokazany na Rysunku 58.

5.12. Android SDK

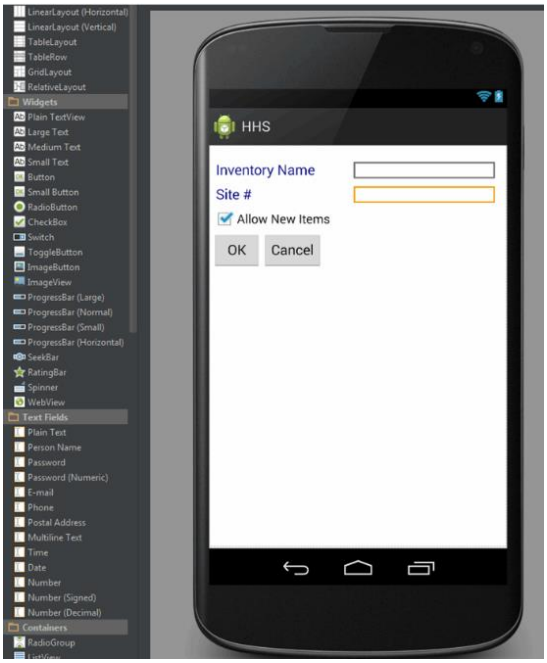
Android SDK (Android Software Development Kit) [70] to zestaw narzędzi przeznaczony do tworzenia oprogramowania z przeznaczeniem na urządzenia działające na platformie Android. Ten system operacyjny działa na jądrze Linux i przeznaczony jest do stosowania na urządzeniach mobilnych, tj. telefonach komórkowych, smartfonach, tabletach itp. Jest jednym z najpopularniejszych systemów mobilnych obecnie stosowanych na świecie. Właścicielem marki jest Google i on odpowiada także za wydawanie aktualnych wersji SDK zgodnie z nowo powstającymi wersjami Androida [70].

Wspierane platformy deweloperskie obejmują komputery działające pod Linuxem, Mac OS X i kolejnych, Windows 7 i kolejnych. Poprzednim oficjalnym IDE był Eclipse, jednak od paru lat jest to Android Studio stworzone przez Google przy współpracy z IntelliJ. SDK składa się z dwóch zasadniczych części: SDK Tools, czyli narzędzi do tworzenia aplikacji oraz Platform Tools, tj. bibliotek

przeznaczonych dla programów dostosowanych pod konkretne wersje Androida [71]. W skład zestawu narzędzi wchodzą [72]:

- debbuger,
- biblioteki,
- emulator,
- dokumentacja,
- przykładowy kod,
- treści instruktażowe.

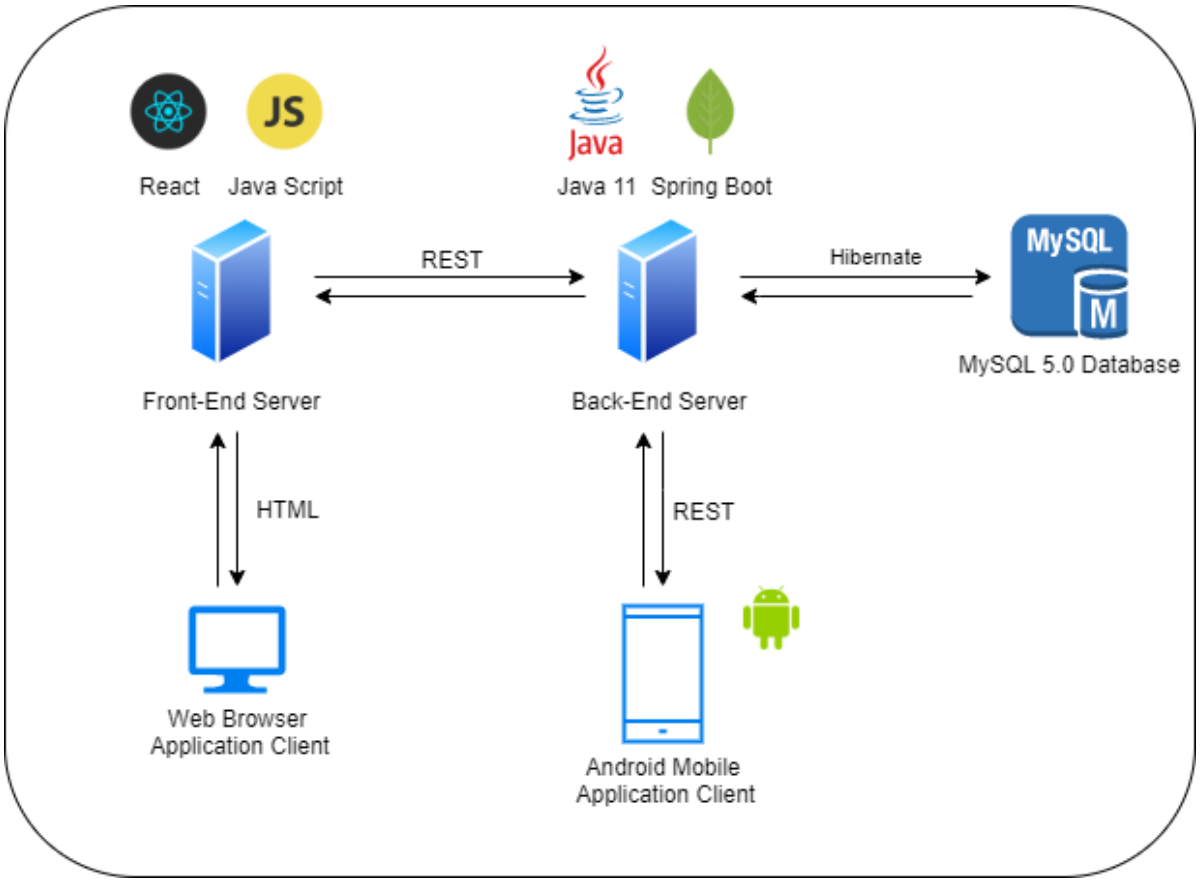
Obecnie planowanie wyglądu aplikacji oraz tworzenie widgetów może odbywać się w bardzo wygodny sposób dzięki Android Studio, gdzie zamiast pisać kod wystarczy „przeciągać" komponenty na widok ekranu "smartfona", co prezentuje Rysunek 59. Ponieważ nasza aplikacja jest przeznaczona także na smartfony z systemem operacyjnym Android, naszym naturalnym wyborem było środowisko Android SDK do stworzenia tej części oprogramowania, która odpowiada za obsługę urządzeń mobilnych.



Rysunek 59. Program Android Studio. Źródło: [73].

6. Implementacja systemu

Centralnym elementem systemu @zor jest trzon aplikacji napisany w języku Java [21]. Z użyciem technologii Spring [25] został utworzony rdzeń programu odpowiedzialny za obsługę procesów biznesowych i komunikację z bazą danych MySQL [39]. Projekt jest zarządzany przy pomocy narzędzia Apache Maven [35]. System posiada dwie wersje: dostępną w przeglądarce i mobilną. Wersja mobilna komunikuje się z serwerem bezpośrednio za pomocą REST API [64], natomiast dla wersji webowej została stworzona warstwa napisana w języku JavaScript [56] z wykorzystaniem biblioteki React [74]. Architekturę systemu przedstawia Rysunek 60.



Rysunek 60. Architektura systemu @zor. Źródło: opracowanie własne.

6.1. Wzorzec projektowy Model – Widok – Kontroler

Aplikacja została zaprojektowana w oparciu o wzorzec MVC (*Model View Controller*) [75]. MVC jest wzorcem wspieranym przez technologię Spring, która udostępnia odpowiednie mechanizmy i adnotacje do jego obsługi. Elementy wzorca w naszym projekcie zostały zaimplementowane w następujący sposób:

1) Model

   Warstwę modelu reprezentują klasy Javy zgodnie z projektem zawartym na diagramach klas analitycznym i projektowym.

2) Widok

   Warstwa aplikacji odpowiadająca za wyświetlanie danych i interakcję z użytkownikiem została napisana w języku JavaScript z użyciem bibliotek React oraz Redux.

3) Kontroler

   Na warstwę pośrednią, odpowiadającą za dostęp do danych składają się dwa typy klas:

   - klasy usługowe typu @Service udostępniające metody wyszukiwania, zapisu, modyfikacji i usuwania danych,
   - kontrolery oznaczane adnotacją @RestController, będące jednocześnie elementem architektury REST API. Udostępniają warstwie widoku zasoby poprzez adresy, pod które wysyłane są żądania.

6.1.1. Warstwa widoku

Warstwa widoku (*View*) jest reprezentowana przez front-end aplikacji napisany w języku JavaScript [53]. Jej główna część to uniwersalne komponenty napisane przy użyciu biblioteki React [74], które są wielokrotnie wykorzystywane w różnych częściach kodu. Stanowią szkielet, który może być następnie wypełniony konkretnymi danymi w zależności od potrzeb i miejsca, w którym się znajdują.

Przykładem takiego komponentu jest `BasicDataTable`. Dzięki zdefiniowaniu uniwersalnej tabeli, można jej użyć w każdym miejscu podając jako argumenty nazwę kolumn i źródło danych. Możemy w niej wyświetlić zarówno informacje na temat pracowników, zwierząt, jak i wizyt u weterynarza. Wykorzystanie uniwersalnej tabeli wyświetlającej listę wizyt przedadopcyjnych przedstawia Listing 1, na którym znajduje się kod komponentu `AdoptionVisit`. Ten komponent jest następnie jednym z elementów formularza adopcyjnego, co prezentuje Listing 2. Dzięki takiemu rozwiązaniu zespół uniknął powielania kodu, a te same elementy pojawiające się w różnych miejscach mają identyczną strukturę, niezależną od programisty implementującego dany element.

Listing 1. Funkcja `AdoptionVisit`. Źródło: opracowanie własne.

```
function AdoptionVisits(props)
//Nazwy kolumn - parametr w tabeli
    const columns = [
        { fieldName: "id", colName: "Id" },
        { fieldName: "employee.lastName", colName: "Nadzorca", link: "/people/",
         idName: "employee.id" },
        { fieldName: "description", colName: "Opis" }
    ];
// Źródło danych dla tabeli
    let data = props.adoption.listPreAdoptiveDecision;
    let table;
    if (data) {
// Komponent BasicDataTable
        table = <BasicDataTable data={data} columns={columns} />
```

```
    }
    return (
        <div class="tab-pane fade" id="nav-wizyty" role="tabpanel" aria-
        labelledby="nav-wizyty-tab">
            <div class="d-flex justify-content-end">
                <button type="button" class="btn btn-outline-secondary btn-sm"
                   data-toggle="modal" data-target="#exampleModal">Dodaj</button>
                <button type="button" class="btn btn-outline-secondary btn-sm"
                   data-toggle="modal" data-target="#exampleModal2">Usuń</button>
            </div>
            <div class="card mb-4">
                <div class="card-body">
                     // Wyświetlenie tabeli
                    {table}
                </div>
            </div>
        </div>
    )
}
```

Listing 2. Funkcja `AdoptionRecordForm()`. Źródło: opracowanie własne.

```
function AdoptionRecordForm(props) {
    return (
        <div class="tab-content" id="nav-tabContent">
        // Użycie komponentów jako elementów składowych
            <AdoptionBasicInformation adoption={props.adoption} />
            <AdoptionQuestionnaire adoption={props.adoption} />
            <AdoptionVisits adoption={props.adoption} />
        </div>
    )
}
```

Wypełnianie komponentów odbywa się za pomocą biblioteki Redux [76], która odpowiada za odczytywanie przychodzących plików JSON. Elementy na stronie są dynamicznie renderowane przez VirtualDOM.

Komunikacja z back-endem aplikacji odbywa się za pomocą interfejsu Fetch API [77]. Przykład zastosowania przedstawia Listing 3. Funkcja `getAdoption()` odpowiada za pobieranie danych o adopcjach. W pierwszym kroku zostaje zbudowany URL żądania, a następnie za pomocą interfejsu `Headers` dodawane są nagłówki, które zostają umieszczone wraz z nazwą metody i parametrem mode w danych konfiguracyjnych. Za pomocą funkcji `fetch()` zostaje wysyłane zapytanie, a po otrzymaniu odpowiedzi instrukcja w ciele metody `then()` wskazuje, co należy zrobić z danymi. Dzięki temu, że funkcja `fetch()` zwraca obiekt typu `Promise`, przetwarzanie danych odbywa się dopiero po ustawieniu odpowiedniej flagi oznaczającej, że dane zostały odebrane.

Listing 3. Funkcja `getAdoption()`. Źródło: opracowanie własne.

```
function getAdoption() {
        // Budowanie URL
        let api_get_adoption = SERVER_ADDRESS + GET_ADOPTIONS_URL + "/" +
        props.match.params.id + "?" + SHELTER_PARAM + shelterId;
        // console.log("GET request for adoption  ", api_get_adoption)
        // Budowanie nagłówka żadania
        const requestHeaders = new Headers();
        requestHeaders.append('Content-Type', 'application/json');
        requestHeaders.append('Authorization', authHeader());
        const config = {
            method: 'GET',
            headers: requestHeaders,
            mode: 'cors'
        }
        // Wysyłanie zapytania…
        fetch(api_get_adoption, config)
        //… i przetwarzanie odpowiedzi, gdy nadejdzie.
            .then(res => {
                if (!res.ok) { throw res }
                return res.json()
            })
            .then(data => {
                setAdoption(data);
            })
            .catch(err => {
                console.log(err)
            })
    }
```

6.1.2. Warstwa kontrolera

Warstwę kontrolera została zaimplementowana za pomocą dwóch typów klas:

- klas usługowych (adnotacja @Service),
- klas kontrolerów (adnotacja @RestController).

Klasy usługowe zawierają logikę biznesową i są odpowiedzialne za dostęp do danych. Możemy w nich znaleźć metody służące do wyszukiwania, zapisu i usuwania informacji z systemu. Listing 4 przedstawia kod interfejsu `AdoptionService`. Klasa implementująca ten interfejs umożliwia:

- Wyszukiwanie procesów adopcyjnych w zależności od podanego kryterium;
- Zapis procesu adopcyjnego lub jego modyfikację;
- Usunięcie procesu adopcyjnego;

- Rozróżnienie procesów adopcyjnych psa i kota. Podział na dwie dodatkowe kategorie został wprowadzony ze względu na występowanie w modelu danych dwóch klas dziedziczących po klasie `Adoption`: `DogAdoption` i `CatAdoption` użytecznych z biznesowego punktu widzenia;
- Paginację.

Listing 4. Interfejs `AdoptionService`. Źródło: opracowanie własne.

```
public interface AdoptionService {
	// Paginacja.
	public PageResponseDto<Adoption> findAll(Specification specification,
	Integer pageNo, Integer pageSize, String sortBy);

	// Wyszukiwanie wszystkich procesów adopcyjnych.
	public List<Adoption> findAllAdoptions(Integer pageNo, Integer
	pageSize, String sortBy);
	// Wyszukiwanie adopcji na podstawie id.
	public Optional<Adoption> findAdoptionById(int id);
	// Zapis lub modyfikacja danych.
	public void saveAdoption(Adoption adoption);
	// Usunięcie danych.
	public void deleteAdoptionById(int id);

	// Adopcja psa
	public List<DogAdoption> findAllDogAdoptions(Integer pageNo, Integer
	pageSize, String sortBy);
	public Optional<DogAdoption> findDogAdoptionById(int id);
	public void saveDogAdoption(DogAdoption dogAdoption);
	public void deleteDogAdoptionById(int id);

	// Adopcja kota
	public List<CatAdoption> findAllCatAdoptions(Integer pageNo, Integer
	pageSize, String sortBy);
	public Optional<CatAdoption> findCatAdoptionById(int id);
	public void saveCatAdoption(CatAdoption dogAdoption);
	public void deleteCatAdoptionById(int id);
}
```

Zakres funkcji przedstawionego interfejsu zawiera wszystkie operacje potrzebne do przeprowadzenia i udokumentowania procesu adopcji mające miejsce w rzeczywistości. Klasa `AdoptionServiceImpl` implementująca w/w interfejs nie działa bezpośrednio na obiektach POJO (Plain Old Java Object), lecz zgodnie z przyjętym wzorcem projektowym wykorzystuje obiekt DAO i jego metody, aby uzyskać dostęp do danych. Listing 5 prezentuje implementacje metod interfejsu `AdoptionService` dotyczące adopcji psa.

Listing 5. Metody dotyczące adopcji psa w klasie `AdoptionServiceImpl`. Źródło: opracowanie własne.

```
@Override
	@Transactional
	// Wyszukiwanie procesu adopcyjnego na podstawie id.
	public Optional<DogAdoption> findDogAdoptionById(int id) {
	  return dogAdoptionDAO.findById(id);
	}
	@Override
	@Transactional
	// Zapis procesu adopcyjnego do bazy danych.
	public void saveDogAdoption(DogAdoption dogAdoption) {
	  dogAdoptionDAO.save(dogAdoption);
```

```
    }

    @Override
    @Transactional
    // Usunięcie danych z bazy na podstawie id.
    public void deleteDogAdoptionById(int id) {

      dogAdoptionDAO.deleteById(id);

    }
```

Obiekty usługowe @Service są wykorzystywane przez inne klasy warstwy kontrolera, oznaczone adnotacją @RestController i będące elementem architektury REST API. Komponenty @RestController umożliwiają aplikacji webowej oraz mobilnej dostęp do zasobów, poprzez adresy (endpointy), pod które wysyłane są żądania. Dane przesyłane są w formacie JSON. Serializacja i deserializacja danych odbywa się przy użyciu biblioteki Jackson. Listing 6 przedstawia klasę `AdoptionRestController`. Ta klasa definiuje adresy pod jakie mają być wysyłane żądania, w celu uzyskania zasobów dotyczących procesów adopcyjnych. Wszystkie metody tej klasy zostały oznaczone adnotacją `@GetMapping` lub `@DeleteMapping` i posiadają odrębne adresy. Ponadto nad całą klasą znajduje się adnotacja `@RequestMapping('/api')`, która wskazuje, że każdy adres musi być poprzedzony podaną wartością.

Listing 6. Klasa `AdoptionRestController`. Źródło: opracowanie własne.

```
// Adnotacja dla kontrolera
@RestController
@CrossOrigin
// Adnotacja wskazująca endpoint i początek adresu dla metod klasy
@RequestMapping("/api")
public class AdoptionRestController {

        private AdoptionService adoptionService;


        @Autowired
        public AdoptionRestController(AdoptionService adoptionService) {
                this.adoptionService = adoptionService;
        }

        // Ścieżka dostępu do danych wszystkich adopcji
        @GetMapping("/adoptions")
        public ResponseEntity<List<Adoption>> findAllAdoptions(
                        @RequestParam(defaultValue = "0") Integer pageNo,
                        @RequestParam(defaultValue = "10") Integer pageSize,
                        @RequestParam(defaultValue = "id") String sortBy)
        {

                List<Adoption> resList = adoptionService.findAllAdoptions(pageNo,
                pageSize, sortBy);

                return new ResponseEntity<List<Adoption>>(resList, new
                HttpHeaders(), HttpStatus.OK);

        }

        // Ścieżka dostępu do danych adopcji o podanym id
        @GetMapping("/adoptions/{id}")
        public Adoption getAdoptionById(@PathVariable int id) {

                Optional<Adoption> adoption = adoptionService.findAdoptionById(id);
```

```
        if (adoption.get() == null) {
                throw new RuntimeException("adoption id not found - "+id);
        }

        return adoption.get();
}

// Ścieżka do usuwania danych adopcji o podanym id
@DeleteMapping("/adoptions/{id}")
public String deleteAdoptionById(@PathVariable int id) {

        Optional<Adoption> tempAdoption =
        adoptionService.findAdoptionById(id);

        // throw exception if null

        if (tempAdoption.get() == null) {
                throw new RuntimeException("Adoption id not found - "+id);
        }

        adoptionService.deleteAdoptionById(id);

        return "Deleted adoption with id: " + id;
}
```

Klasa kontrolera służy do komunikacji z warstwą widoku, natomiast właściwe operacje są wykonywane w klasie usługowej implementującej interfejs `AdoptionService`, udostępniający metody do obsługi całego procesu biznesowego, kod wspomnianego interfejsu został umieszczony wcześniej na Listingu 4.

6.2. Pozostałe wzorce projektowe

Pozostałe wzorce projektowe wykorzystane przy tworzeniu aplikacji to:

1. Odwrócenie sterownia (*Inversion of Control*) [25] – wzorzec wspierany przez Spring umożliwiający automatyczne zarządzanie instancjami klas i tworzenie obiektów w sposób niejawny poprzez przekazanie kontroli nad obiektami kontenerowi Spring. Adnotacje @Service i @Repository wskazują na klasę odpowiadającą za dostęp do danych;

2. Wstrzykiwanie zależności (*Dependency Ingestion*) [78] – w aplikacji został wykorzystany wzorzec wstrzykiwania zależności będący implementacją wzorca odwróconego sterowania. Zaimplementowano wstrzykiwanie poprzez konstruktor z zastosowaniem adnotacji @Autowired. Przykłady wstrzykiwania zależności przedstawiają Listing 7 i Listing 8. Listing 7 przedstawia klasę usługową `EventServiceImpl` do obsługi zdarzeń, korzystającą z obiektów dostępu do danych: `EventDAO`, `PersonDAO`, `AnimalDAO` i `PreAdoptiveDecisionDAO`. Natomiast Listing 8 przedstawia klasę repozytorium `ShelterDAOImpl`, gdzie został wstrzyknięty obiekt `EntityManager` zarządzający połączeniem z bazą danych.

Listing 7. Fragment klasy usługowej `EventServiceImpl`. Źródło: opracowanie własne.

```
//adnotacja oznaczająca klasę usługową
@Service
public class EventServiceImpl implements EventService  {

    private EventDAO eventDAO;
    private PersonDAO personDAO;
    private AnimalDAO animalDAO;
    private PreAdoptiveDecisionDAO preAdoptiveDecisionDAO;

    //wstrzykiwanie zależności w konstruktorze
    @Autowired
    public EventServiceImpl(EventDAO eventDAO, PersonDAO personDAO,EventDAO2
    eventDAO2,AnimalDAO animalDAO, PreAdoptiveDecisionDAO, preAdoptiveDecisionDAO){
            this.eventDAO = eventDAO;
            this.personDAO = personDAO;
            this.animalDAO = animalDAO;
            this.preAdoptiveDecisionDAO = preAdoptiveDecisionDAO;
            this.preAdoptiveDecisionDAO = preAdoptiveDecisionDAO;
    }
```

3. DAO (*Data Access Object*) [79] – wzorzec warstwy dostępu do danych, który uniezależnia model danych od sposobu dostępu do nich. W aplikacji każda encja posiada swoją klasę DAO, obsługującą dostęp do bazy danych. Przykład przedstawia Listing 8 pokazujący klasę, w której znajdują się metody do wyszukiwania, dodawania i usuwania danych schroniska w systemie;

Listing 8. Klasa `ShelterDAOImpl`. Źródło: opracowanie własne.

```
//adnotacja oznaczająca klasę odpowiadającą za dostęp do danych
@Repository
public class ShelterDAOImpl implements ShelterDAO{
    private EntityManager entityManager;


    @Autowired
    public ShelterDAOImpl(EntityManager theEntityManager) {
        entityManager = theEntityManager;
    }
    //metoda służąca do wyszukiwania schronisk
    @Override
    public List<Shelter> findAllShelter() {
        Session currentSession = entityManager.unwrap(Session.class);
        Query<Shelter> theQuery = currentSession.createQuery("from Shelter",
        Shelter.class);
        List<Shelter> shelters = theQuery.getResultList();
        return shelters;
    }
    //metoda służąca do wyszukiwania schroniska na podstawie id
    @Override
    public Shelter findShelterById(int id) {
        Session currentSession = entityManager.unwrap(Session.class);
        Shelter shelter =
                currentSession.get(Shelter.class, id);
```

```
        return shelter;
    }
    //metoda służąca do zapisu danych schroniska
    @Override
    public void saveShelter(Shelter shelter) {
        Session currentSession = entityManager.unwrap(Session.class);
        currentSession.saveOrUpdate(shelter);
    }
    //metoda służąca do usunięcia danych schroniska na podstawie id
    @Override
    public void deleteShelterById(int id) {
        Session currentSession = entityManager.unwrap(Session.class);
        Shelter shelter =
                currentSession.get(Shelter.class, id);
        entityManager.remove(shelter);
    }
}
```

4. DTO (*Data Transfer Object*) [80] – wzorzec warstwy dostępu do danych, który pozwala zredukować ilość wywołań wykonywanych w celu ich uzyskania. W opisywanej aplikacji do transferu danych zagregowanych z dwóch lub więcej klas zostały stworzone kontenery DTO. Przykładowy kontener służący do obsługi raportu przedstawia Listing 9. Kontener agreguje dane z dwóch klas: `Animal` i `Shelter`.

Listing 9. Klasa `ShelterReportDto`. Źródło: opracowanie własne.

```
public class ShelterReportDto {

    private String count;
    private String breed;
    private String shelter;

    public ShelterReportDto() { }
    public String getCount() {return count}
    public void setCount(String count) {this.count = count;}
    public String getBreed() {return breed;}
    public void setBreed(String breed) {this.breed = breed;}
    public String getShelter() {return shelter;}
    public void setShelter(String shelter) {this.shelter = shelter;}
}
```

5. Fasada (*Facade*) [81] – wzorzec strukturalny polegający na udostępnieniu systemu poprzez uproszczony interfejs (np. jedną klasę), umożliwiający korzystanie wyłącznie z potrzebnych funkcji. W aplikacji zostało to osiągnięte poprzez stworzenie klas usługowych służących do interakcji z systemem. Dzięki temu pozostałe warstwy nie mają dostępu do całego systemu (np.

encji, klas DAO), a jedynie do wybranych jego funkcji. Przykład interfejsu klasy-fasady pokazuje Listing 10. Implementacja tego interfejsu zawiera nie tylko podstawowe metody do wyszukiwania, dodawania lub modyfikacji i usuwania zwierzęcia z systemu, ale także np. funkcję związaną z powiadamianiem o przyjęciu nowego zwierzęcia do schroniska.

Listing 10. Klasa `ShelterReportDto`. Źródło: opracowanie własne.

```
public interface AnimalService {

	public List<Animal> findAll(Integer pageNo, Integer pageSize, String sortBy);

	public Optional<Animal> findById(int theId);

	PageResponseDto<Animal> findAllAnimalsByShelter(Integer pageNo, Integer
	pageSize, String sortBy, int shelterId);

	public List<Animal> findByHealth(String type, String health);

	public void save(Animal animal);

	public void deleteById(int theId);

	public List<Animal> getByType(String type);

	public Boolean notifyAboutNewAnimal(Integer animalId);

	List<Animal> findByStatus(Status... statuses);

	PageResponseDto<Animal> findByStatus(Integer pageNo, Integer pageSize, String
	sortBy, int shelterId, Status... statuses);

	PageResponseDto<Animal> findAnimalsToEadoptByShelter(Integer pageNo, Integer
	pageSize, String sortBy, int shelterId);
}
```

6.3. Komunikacja z bazą danych

W projekcie została wykorzystana relacyjna baza danych MySQL [39], a komunikacja i transformacja danych z reprezentacji obiektowej na relacyjną odbywa się poprzez maper obiektowo-relacyjny Hibernate [34]. Każda klasa znajdująca się na diagramie klas została zaimplementowana jako klasa Javy z odpowiednimi adnotacjami zgodnie ze standardem JPA (Java Persistance API). Obiekty są mapowane z uwzględnieniem atrybutów i asocjacji. Podstawowe adnotacje służące mapowaniu klas i atrybutów to:

- @Entity – oznacza, że klasa jest encją w bazie danych;
- @Table – nazwa tabeli odpowiadającej danej encji;
- @Id – identyfikator obiektu, dodatkowa adnotacja @GeneratedValue pozwala na ustawienie strategii generowania identyfikatora;
- @Column – oznacza nazwę kolumny i pozwala określić dodatkowe parametry np. nullable, unique, length;
- @Enumerated – oznacza enum, EnumType pozwala zdefiniować typ wartości;
- @Transient – oznacza atrybut wyliczalny, ignorowany przez Hibernate.

Adnotacje służące do mapowania asocjacji:

- @OneToOne –relacja jeden do jednego;
- @OneToMany –relacja jeden do wielu;
- @ManyToOne –relacja wiele do jednego;
- @ManyToMany –relacja wiele do wiele.

Asocjacje tabel w bazie danych są dwustronne (bi-directional), a implementacja w kodzie odpowiednich mechanizmów zapewnia bezpiecznie dodawanie i usuwanie asocjacji bez utraty spójności danych. Listing 11 przedstawia fragment kodu klasy `Animal`, wraz z adnotacjami wykorzystywanymi przez Hibernate.

Listing 11. Fragment klasy `Animal`. Źródło: opracowanie własne.

```
@Entity //adnotacja oznaczająca encję
@Table(name = "animal") //adnotacja nadająca nazwę tabeli w bazie danych
@JsonIgnoreProperties("medicalActivities")
public class Animal implements Serializable {

//adnotacja oznaczająca identyfikator obiektu, wymagana przez Hibernate
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;

//adnotacja nadająca nazwę kolumnie wraz z informacją, że wartość jest wymagana
    @Column(name = "name", nullable = false)
    private String name;

    @Column(name = "breed", nullable = false)
    private String breed;

    @Column(name = "weight_kg", nullable = false)
    private Double weightKg;

    @Column(name = "temper")
//adnotacja oznaczająca enum typy String
    @Enumerated(EnumType.STRING)
    private Temper temper;

//adnotacja nadająca nazwę kolumnie wraz z maksymalną ilością znaków
    @Column(name = "description", length = 3000)
    private String description;

//adnotacja mapująca dwukierunkową relację jeden do wielu wskazująca na drugą
stronę relacji
    @OneToMany(targetEntity = VirtualAdoption.class,
            mappedBy = "animal")
    @JsonIgnore
    private List<VirtualAdoption> virtualAdoptions = new ArrayList<>();

//adnotacja mapująca dwukierunkową relację wiele do wielu wskazująca strukturę
wiersza w tabeli asocjacyjnej
    @ManyToMany(fetch=FetchType.LAZY,
            cascade= {CascadeType.PERSIST, CascadeType.MERGE,
                    CascadeType.DETACH, CascadeType.REFRESH})
    @JoinTable(
            name="animal_person",
            joinColumns=@JoinColumn(name="animal_id"),
            inverseJoinColumns=@JoinColumn(name="person_id")
    )
```

```
    @JsonIgnoreProperties({"animals", "shelter", "virtualAdoptions",
    "medicalActivities"})
    private List<Person> people = new ArrayList<>();
```

Połączenie z bazą danych uzyskiwane jest poprzez sterownik MySQL Connector. Do komunikacji wykorzystywany jest Hibernate EntityManager - obiekt zarządzający sesją, a adnotacja @Transactional (umieszczona w klasach @Service) zapewnia transakcyjność połączeń. Dostęp do danych odbywa się poprzez obiekty DAO (Data Access Object), oznaczone adnotacją @Repository, wykorzystujące interfejsy Hibernate do operacji CRUD. Listing 12 przedstawia klasę `AnimalDAOImpl` zawierającą obiekt `EntityManager` do pobierania, zapisywania i usuwania danych z bazy.

Listing 12. Klasa `AnimalDAOImpl`. Źródło: opracowanie własne.

```
// Adnotacja wskazująca klasę służącą do przechowywania danych i wykonywania
operacji na nich.
@Repository
public class AnimalDAOImpl implements AnimalDAO {
// EntityManager - obiekt zarządzający sesją
    protected EntityManager entityManager;

    @Autowired
    public AnimalDAOImpl(EntityManager theEntityManager) {
        entityManager = theEntityManager;
    }
    private String getTableName() {
        return "Animal";
    }

// Metoda służąca do pobrania z bazy danych listy wszystkich zwierząt
    @Override
    public List<Animal> findAll() {
        // Utworzenie sesji za pomocą obiektu EntityManager
        Session currentSession = entityManager.unwrap(Session.class);
        // Utworzenie zapytania
        Query<Animal> theQuery =
                currentSession.createQuery("from " + getTableName(), Animal.class);
        // Uzyskanie listy zwierząt i przypisanie jej do zmiennej
        List<Animal> resultList = theQuery.getResultList();
        return resultList;
    }

// Metoda służąca do zapisu lub uaktualnienia danych zwierzęcia
    @Override
    public void save(Animal entity) {
        // Utworzenie sesji za pomocą obiektu EntityManager
        Session currentSession = entityManager.unwrap(Session.class);
```

```
        // Zapis lub uaktualnienie danych
        currentSession.saveOrUpdate(entity);
    }

    @Override
    public void delete(int id) {
        // Utworzenie sesji za pomocą obiektu EntityManager
        Session currentSession = entityManager.unwrap(Session.class);
        // Pobranie danych zwierzęcia na podstawie jego id
        Animal animal = currentSession.get(Animal.class, id);
        // Usunięcie zwierzęcia
        entityManager.remove(animal);
    }
}
```

6.4. Mechanizmy zapewniające bezpieczeństwo

W trakcie tworzenia aplikacji zostały zaimplementowane mechanizmy zapewniające bezpieczeństwo przechowywanych danych oraz dostęp do nich wyłącznie uprawnionym osobom. Zadbano także o obsługę wyjątków tak, by mimo ich wystąpienia aplikacja działała, a użytkownik miał poczucie ciągłości funkcjonowania mimo wystąpienia błędów. W kolejnych podrozdziałach przedstawiono zastosowane mechanizmy.

6.4.1. Logowanie do systemu

W aplikacji został zastosowany standard RFC 7617 [82]- uwierzytelnienie za pomocą Basic Authentication. Jest ono obsługiwane w `authentication.service.js` za pomocą funkcji `login()`, która koduje dane w formacie tekstowym za pomocą base64 i dodaje wymagany nagłówek. Funkcję `login()` przedstawia Listing 13.

Listing 13. Funkcja `login()`. Źródło: opracowanie własne.

```
function login(email, password) {

    // Kodowanie za pomocą base64
    const basicToken = 'Basic ' + btoa(email + ":" + password);

    // Dodanie nagłówków
    const loginHeaders = new Headers();
    loginHeaders.append('Authorization', basicToken);
    loginHeaders.append('Content-Type', 'application/json');

    // Ustawienie parametrów
    const requestOptions = {
        method: 'POST',
        headers: loginHeaders,
```

```
        body: JSON.stringify({ email, password })
    };
    let url = azorProperties.SERVER_BASE_URL+ 'auth/basicauth';

    // Wysyłanie i odbiór danych za pomocą Fetch API
    return fetch(url, requestOptions)
        .then(handleResponse)
        .then(user => {
            user.basicToken = basicToken;
            localStorage.setItem('currentUser', JSON.stringify(user));
            currentUserSubject.next(user);
            return user;
        });
}
```

W celu zabezpieczenia przed nieuprawnionym dostępem hasło jest szyfrowane algorytmem MD5 od razu po wpisaniu go do formularza, tak więc funkcji `login()` zostaje przekazany hash hasła i na żadnym etapie komunikacji nie jest ono widoczne w formie niezaszyfrowanej. Ponieważ hasła przechowywane w bazie danych są zaszyfrowane podwójnie, najpierw wspomnianym już algorytmem MD5, a następnie algorytmem bcrypt, klasa BcryptGenerator należąca do back-endu aplikacji generuje kolejny hash i porównuje z danymi zapisanymi w bazie danych. Jeśli porównanie przebiegnie pozytywnie, sesji przydzielany jest token (czas ważności tokena został ustawiony na 15 minut). Dane dotyczące sesji są zapisywane w plikach cookies i na ich podstawie tworzony jest nagłówek autoryzacyjny w kolejnych wywołaniach wymagających autoryzacji.

W celu zapewnienia bezpieczeństwa przesyłanych danych został wdrożony protokół HTTPS [83] szyfrujący dane za pomocą protokołu SSL. Jednak ze względu na użycie certyfikatu z podpisem własnym (self-signed), a nie wydanym przez zaufany ośrodek, certyfikat był odrzucany przez przeglądarki. Dlatego ostatecznie w prototypie zrezygnowano z protokołu HTTPS na rzecz HTTP.

6.4.2. Dostęp do funkcjonalności wymagających logowania

W aplikacji został zastosowany mechanizm Role-based access control [84] polegający na przydzielaniu uprawnień rolom zdefiniowanym w systemie. Role zostają przypisane użytkownikom, którzy dzięki temu uzyskują dostęp do określonych czynności W systemie istnieje osiem ról:

- Adoptive person,
- Animal caretakier,
- Employee,
- Give away person,
- Vet,
- Volunteer,
- Admin,
- Superadmin.

Po stronie serwera front-end zastosowano PrivateRoute oparty na rolach. PrivateRoute to komponent Reacta kontrolujący dostęp do zasobów. Umożliwia sprawdzenie, czy dany zasób wymaga autoryzacji, a jeśli tak, to czy użytkownik jest zalogowany i czy przypisana mu rola ma uprawnienia do danego zasobu. Kod `PrivateRoute.js` przedstawia Listing 14.

Listing 14. `PrivateRoute.js`. Źródło: opracowanie własne.

```
import React from 'react';
import { Route, Redirect } from 'react-router-dom';

import { authenticationService } from './_services/authentication.service';

export const PrivateRoute = ({ component: Component, roles, ...rest }) => (
    <Route {...rest} render={props => {
        const currentUser = authenticationService.currentUserValue;
        if (!currentUser) {
        // Jeśli użytkownik nie jest zalogowany przekierowanie do strony logowania
            return <Redirect to={{ pathname: '/login', state: { from:
            props.location } }} />
        }
        // Sprawdzenie, czy strona jest dostępna dla roli użytkownika
        if (roles && roles.indexOf(currentUser.role) === -1) {
       // Jeśli rola nie posiada uprawnień następuje przekierowanie do strony
          głównej
            return <Redirect to={{ pathname: '/'}} />
        }
        // Rola posiada uprawnienia dostępu, więc zostaje zwrócony komponent
        return <Component {...props} />
    }} />
)

export default PrivateRoute;
```

Po stronie serwera back-end został zaimplementowany Spring Security. Konfiguracja znajduje się w klasie `SpringSecurityConfigurationBasicAuth` dziedziczącej klasę udostępnianą przez Spring – WebSecurityConfigurerAdapter. Za pomocą metody `configure()` dostępy do poszczególnych adresów url zostały przydzielone wyżej wymienionym rolom. Fragment klasy `SpringSecurityConfigurationBasicAuth` zawierający w/w metodę jest przedstawiony na Listingu 15

Listing 15. Metoda `configure()` w klasie `SpringSecurityConfigurationBasicAuth`. Źródło: opracowanie własne.

```
    protected void configure(HttpSecurity http) throws Exception {
        http
```

```
        .csrf().disable()
        .authorizeRequests()
        // Adresy URL dostępne dla wszystkich użytkowników
        .antMatchers(HttpMethod.OPTIONS,"/**").permitAll()
        .antMatchers(HttpMethod.GET, "/util/seed-db" ,"/api/animal*/**",
          "/api/articles*/**", "/api/email-subscription*/**",
          "/api/shelters*/**").permitAll()
        // Adresy URL dostępne dla poszczególnych ról w systemie
         .antMatchers(HttpMethod.POST, "/api/animal*/**", "/api/articles*/**",
           "/api/email-subscription*/**").hasRole(UserRole.EMPLOYEE)
        .antMatchers("/admin/**").hasRole(UserRole.ADMIN)
        .antMatchers("/api/**").hasRole(UserRole.EMPLOYEE)
        .antMatchers("/vet/**").hasRole(UserRole.VET)
        .antMatchers("/caretaker/**").hasAnyRole(UserRole.EMPLOYEE,
         UserRole.ANIMAL_CARETAKER, UserRole.VOLUNTEER)
        .antMatchers("/adoptive/**").hasRole(UserRole.ADOPTIVE_PERSON)
                .anyRequest().authenticated()
                .and()
            //.formLogin().and()
            .httpBasic();
    }
```

6.4.3. Inne mechanizmy zapewniające bezpieczeństwo aplikacji

Aplikacja została zaprojektowana i napisana z uwzględnieniem również innych mechanizmów bezpieczeństwa:

1. Brak dostępu do portu bazy danych z zewnątrz.
   Dzięki konfiguracji firewalla otwarte są wyłącznie porty 80 oraz 8080. Nie ma możliwości dostępu do portu bazy danych;
2. Rejestracja logów.
   W tym celu została użyta biblioteka slf4j [85] umożliwiająca pięć poziomów logowania: ERROR, WARN, INFO, DEBUG, TRACE. Przykład użycia poziomu info oraz error jest widoczny na Listingu 16;

Listing 16. Fragment klasy `UseMedicineRestController`. Źródło: opracowanie własne.

```
@RestController
@CrossOrigin
@RequestMapping("/api")
public class UseMedicineRestController {

    private UseMedicineService useMedicineService;
    // Stworzenie obiektu klasy Logger
    Logger logger = LoggerFactory.getLogger(UseMedicineRestController.class);

    @Autowired
    public UseMedicineRestController(UseMedicineService useMedicineService) {
        this.useMedicineService = useMedicineService;
    }
```

```java
@GetMapping("/use-medicines")
public List<UseMedicine> findAll() {
    return useMedicineService.findAll();
}

@GetMapping("/use-medicines/{useMedicineId}")
public UseMedicine getUseMedicine(@PathVariable int useMedicineId) {
    // Log typu INFO
    logger.info("START getUseMedicine");
    UseMedicine useMedicine = useMedicineService.findById(useMedicineId);
    // Log typu ERROR
    if (useMedicine == null) {
        logger.error("Record not found");
        throw new RuntimeException("Use Medicine id not found- +useMedicineId);
    }
    // Log typu INFO
    logger.info("END getUseMedicine");
    return useMedicine;
}
```

3. Obsługa błędów związanych z REST API.

   Zaimplementowano mechanizmy pozwalające na przechwytywanie nieprawidłowych żądań (np. zawierających nieprawidłową metodę lub brakujące parametry), które w odpowiedzi na błędne zapytanie zwracają plik typu JSON z kodem błędu i jego krótkim opisem.

7. Interfejs aplikacji

Wygląd interfejsu aplikacji oraz wygoda korzystania z oprogramowania mają kluczowe znaczenie dla odniesienia rynkowego sukcesu. Nierzadko może to przesądzić o wyborze programu przez potencjalnego klienta. Osoba dla której aplikacja ma być codziennym narzędziem pracy zwróci uwagę przede wszystkim na prostotę, czytelność i łatwość obsługi oraz na czas w jakim jest w stanie przyswoić umiejętność sprawnego wykorzystywania potrzebnych funkcji. Częsty problem stanowi też niechęć pracowników do nowych rozwiązań, gdyż wymaga to nauki i czasu na wdrożenie się. Toteż ważne jest aby już na pierwszy rzut oka system sprawiał wrażenie przyjaznego i zrozumiałego dla użytkownika.

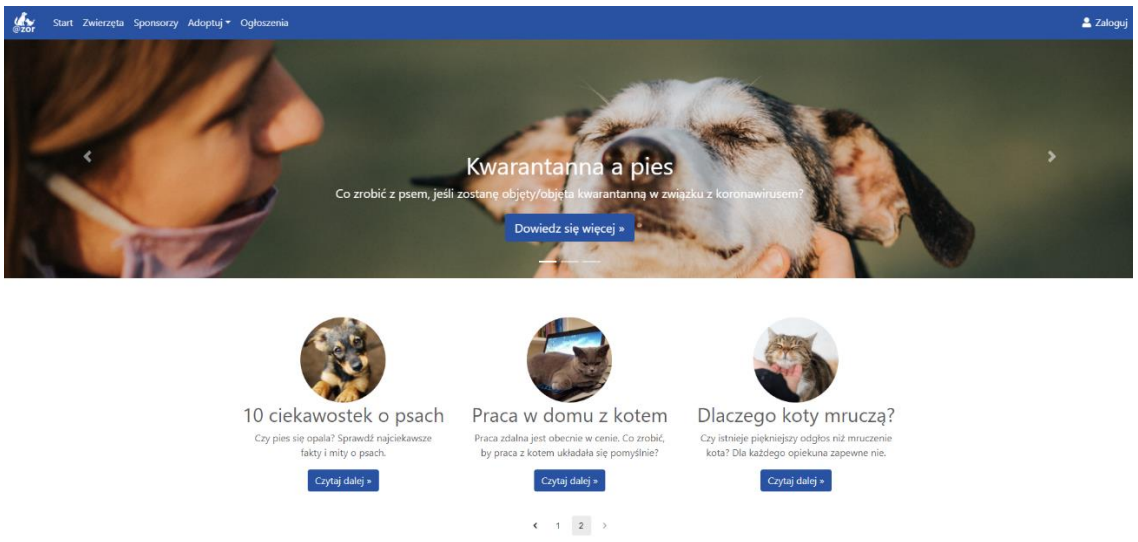
Tworząc interfejs aplikacji starano się zapewnić nowoczesny design oraz estetykę. Zadbano by ekrany były czytelne, spójne i powtarzalne. Zwracano również uwagę na responsywność oraz skalowalność tak by na każdym urządzeniu praca z aplikacją była możliwa i wygodna. Dla telefonów komórkowych zostało stworzone oddzielne GUI.

7.1. Nawigacja po aplikacji

Poniżej zaprezentowane zostały przykładowe elementy interfejsu omawianej aplikacji demonstrujące najważniejsze ekrany z punktu widzenia użytkowników oraz estetykę, ergonomiczność oraz powtarzalność zastosowanych rozwiązań.

7.1.1. Użytkownicy niezalogowani

Nasz program służy zarówno pracownikom jak i osobom spoza schroniska. Ekranem startowym widocznym dla osoby spoza organizacji jest główna strona internetowa danego schroniska, co przedstawia Rysunek 61.



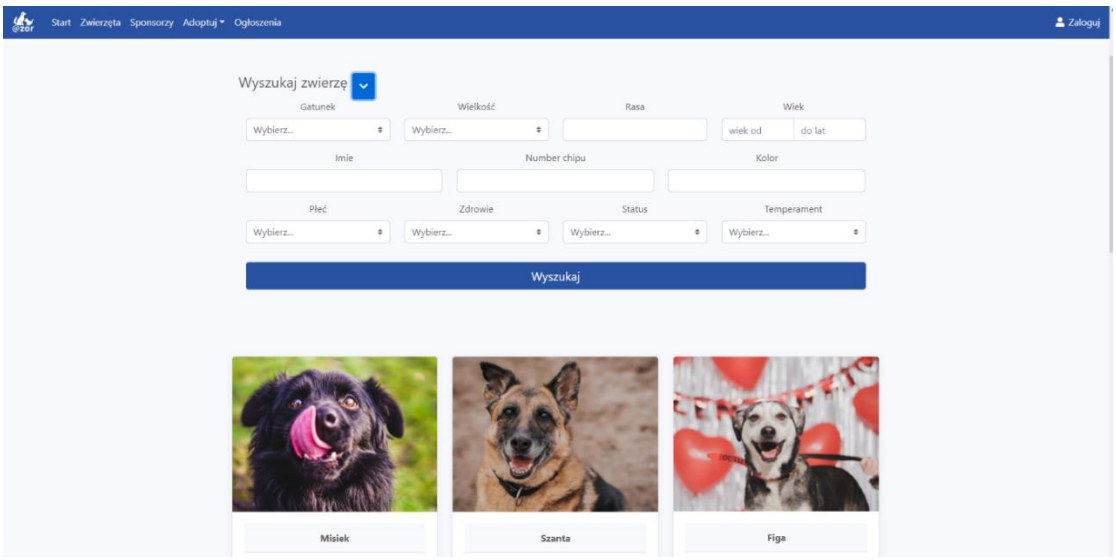
Rysunek 61. Ekran główny schroniska. Źródło: opracowanie własne.

Strona powitalna każdej placówki dla niezalogowanego użytkownika składa się z karuzeli wyświetlającej najnowsze informacje z życia schroniska, a także udostępnia linki do artykułów i ogłoszeń. Przykład takiego artykułu widać na Rysunku 62.

Rysunek 62. Ekran artykułów. Źródło: opracowanie własne.

Na górze ekranu znajduje się poziomy pasek nawigacji zastosowany jako główne menu aplikacji służące do poruszania się po stronie. Umożliwia on przejście do kolejnych stron/modułów programu bądź do logowania. Należy zwrócić uwagę, że pasek ten towarzyszy użytkownikowi przez cały czas pobytu na stronach schroniska udostępniając określone strony/moduły w zależności od nadanych uprawnień. Ta powtarzalność ułatwia szybkie przyswojenie obsługi wyboru dostępnych funkcjonalności.

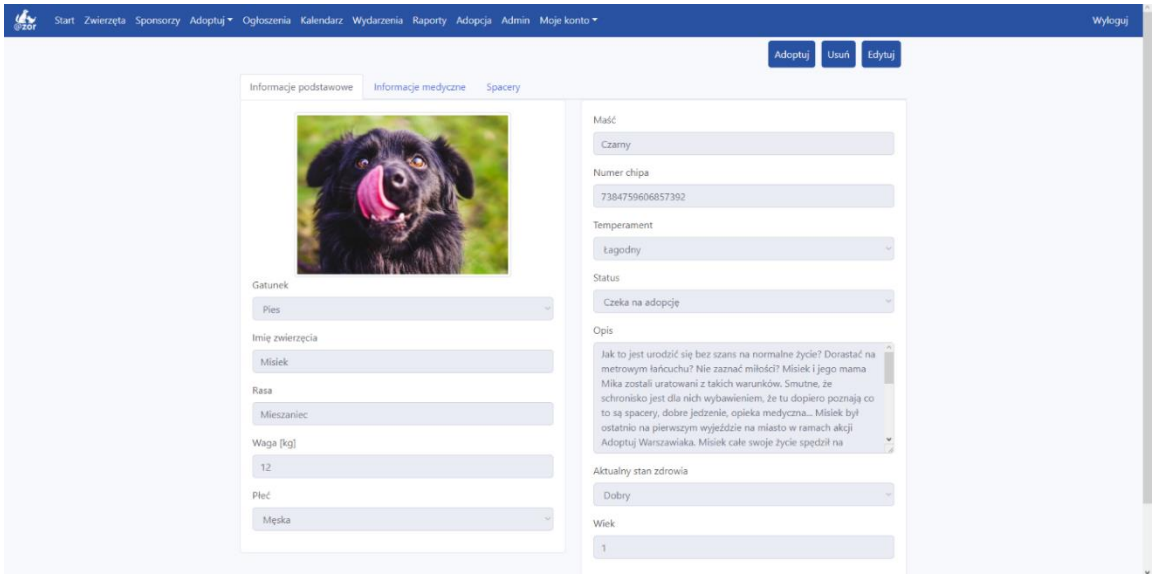
Poniżej omówione zostały przykładowe ekrany występujące w poszczególnych zakładkach aplikacji. Część z nich jest ogólnodostępna dla wszystkich odwiedzających serwis, a część tylko dla zalogowanych użytkowników. Wybrane ekrany różnią się w zależności od dostępnych uprawnień, co zostanie zademonstrowane na przykładach.



Rysunek 63. Ekran listy zwierząt. Źródło: opracowanie własne.

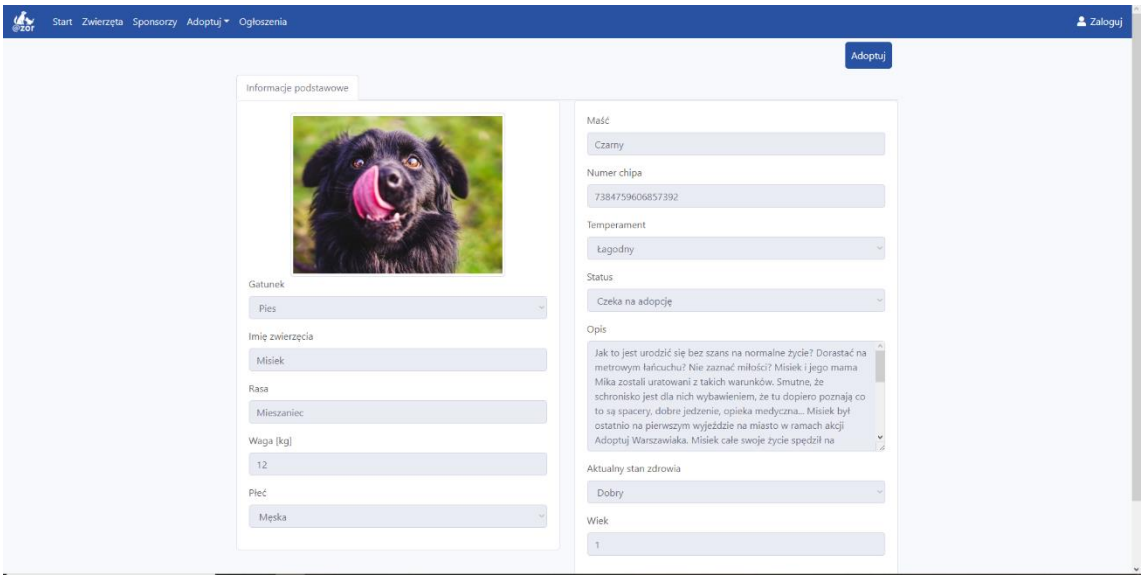
Moduł Zwierzęta widoczny w pasku nawigacji udostępnia listę wszystkich zwierząt obecnych w schronisku wraz z możliwością filtrowania po ich atrybutach, co ma ułatwić odnalezienie podopiecznego zgodnie z wybranym kryterium np. gatunkiem lub rasą. Następnie takie zwierzę może zostać zaadoptowane wirtualnie bezpośrednio z tego widoku. Pracownik zalogowany dodatkowo widzi przyciski umożliwiające dodanie, edycję czy usunięcie zwierzęcia. Podobnie jest na kolejnych prezentowanych ekranach. Dzięki temu nie ma znacznego rozróżnienia między widokiem stron dla zalogowanych jak i niezalogowanych osób. Zredukowało to ilość pracy programisty oraz gwarantuje spójność widoków niezależnie od roli. Rysunek 63 przedstawia ekran z listą zwierząt.

Należy ponadto zwrócić uwagę, że w celu uproszczenia pracy z aplikacją oraz zachowania spójnego wyglądu, taka struktura ekranu stanowi bazę także dla innych elementów interfejsu zawierających listę z obrazkami, jak np. okno zwierząt przeznaczonych do adopcji lub adopcji wirtualnej. Po wybraniu danego zwierzęcia strona przekierowuje użytkownika do jego indywidualnej kartoteki. Tak, jak w przypadku listy zwierząt schemat ekranu kartoteki jest wykorzystywany dla każdego rodzaju użytkownika, również do zarządzania pracownikami, udostępniając wybrane elementy kartoteki zgodnie z posiadanymi uprawnieniami, np. wspomniane przyciski umożliwiające edycję/usunięcie obiektu. Rysunek 64 prezentuje przykładową kartotekę zwierzęcia z perspektywy zalogowanego pracownika a Rysunek 65 niezalogowanego dla porównania.



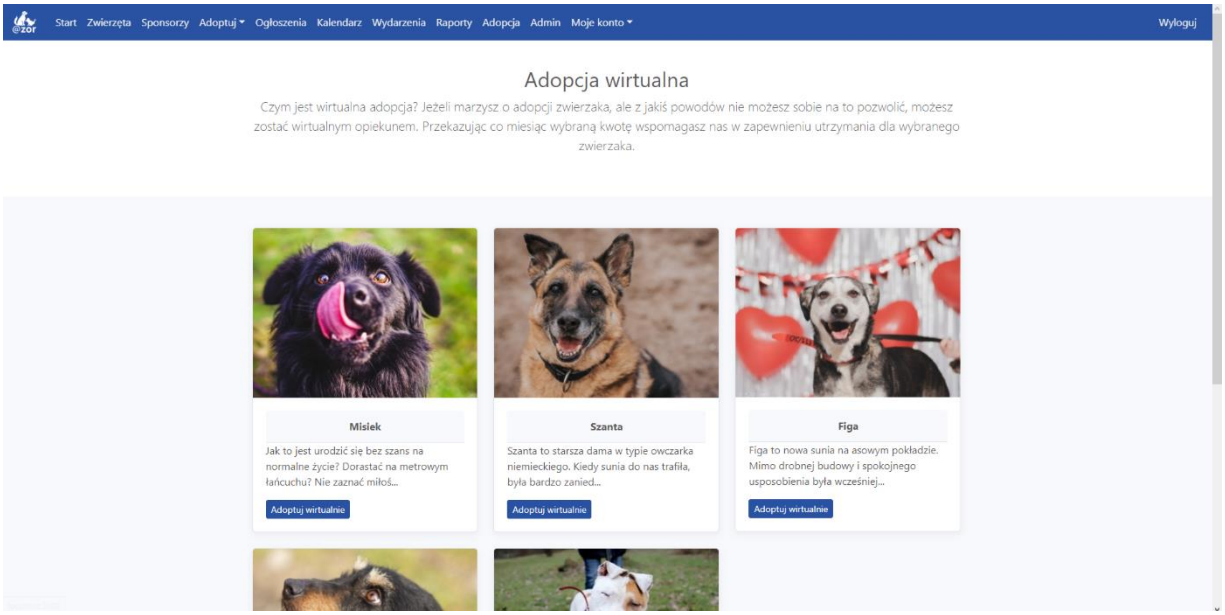
Rysunek 64. Ekran kartoteki zwierzęcia widoczny dla zalogowanego pracownika. Informacje podstawowe. Źródło: opracowanie własne.

Należy zwrócić uwagę, że zakładki „Informacje medyczne” oraz „Spacery” nie są dostępne dla osoby niebędącej pracownikiem lub wolontariuszem a przyciski „Edytuj”, „Usuń” widoczne są wyłącznie dla pracownika. O zakładkach będzie więcej w dalszej części pracy. Ten sam ekran przeznaczony jest także do dodania nowego zwierzęcia do bazy. Przykład ten doskonale prezentuje pożądaną powtarzalność schematów ekranów w celu zmniejszenia ich liczności, co z kolei wpływa na skrócenie czasu tworzenia aplikacji a także czasu wymaganego na naukę jej obsługi.



Rysunek 65. Ekran kartoteki zwierzęcia widoczny dla niezalogowanego użytkownika. Informacje podstawowe. Źródło: opracowanie własne.

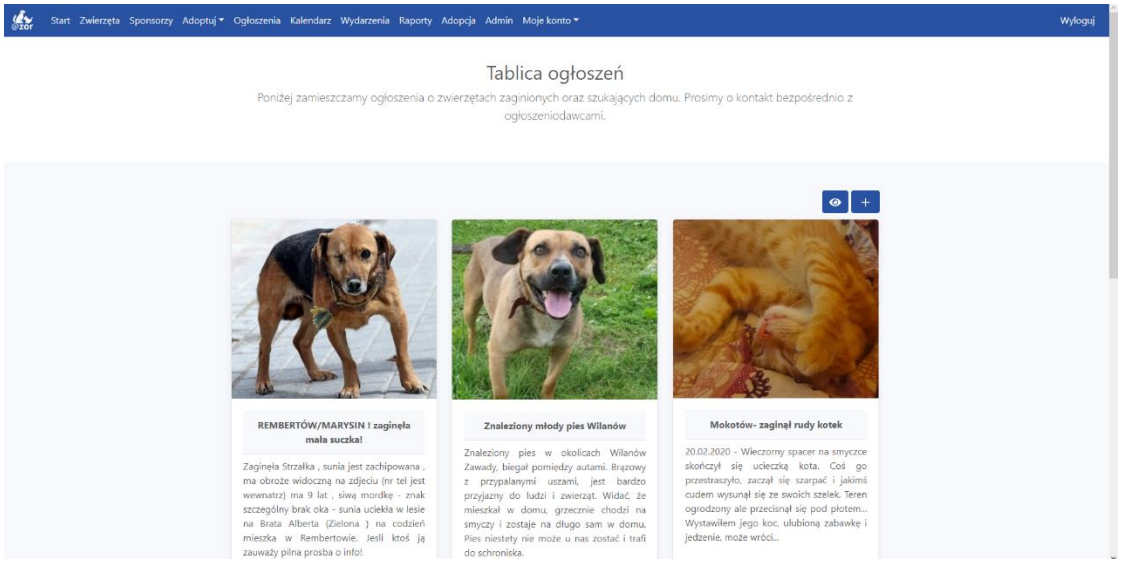
Drugim kluczowym modułem naszej aplikacji jest zarządzanie adopcjami. Dzielą się one na dwa podtypy. Po kliknięciu w przycisk „Adopcja" pokazuje się lista rozwijana (*drop-down menu*): „Zaadoptuj wirtualnie" oraz „Zabierz do domu". Po wybraniu opcji adopcji wirtualnej użytkownik przenoszony jest na stronę omawianej już listy zwierząt z dodatkowym nagłówkiem przybliżającym koncepcję tego typu adopcji. Prezentuje to Rysunek 66.



Rysunek 66. Ekran listy zwierząt do adopcji wirtualnej. Źródło: opracowanie własne.

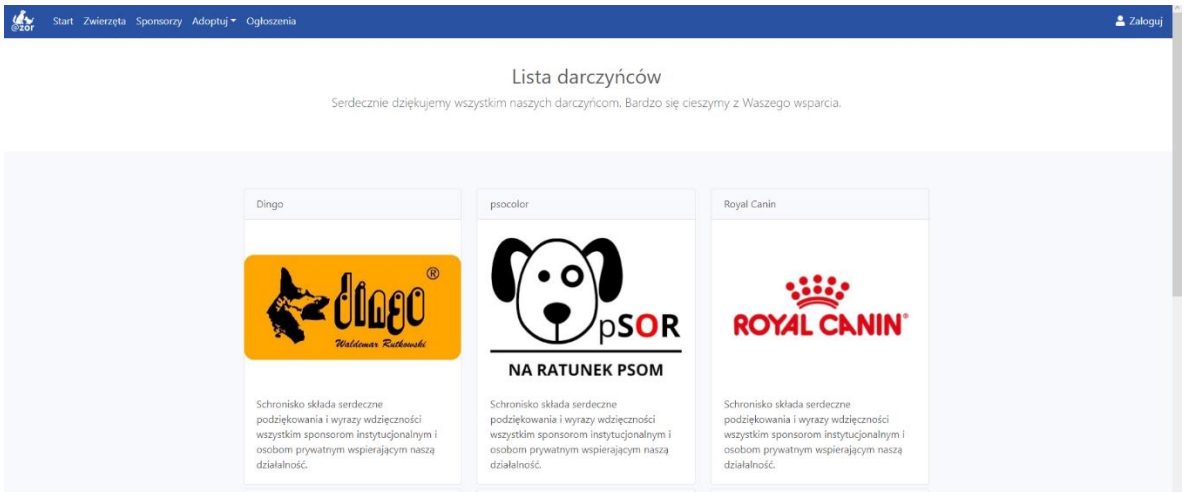
Wybór konkretnego zwierzęcia kieruje użytkownika do okna logowania/rejestracji a następnie do uzupełnienia wniosku. Natomiast w przypadku adopcji klasycznej, użytkownik po zalogowaniu proszony jest o wypełnienie e-formularza, który automatycznie trafia do pracowników schroniska w celu weryfikacji i podjęcia dalszych czynności. Zostanie to przybliżone w kolejnym podrozdziale.

Wyróżniającym nasze oprogramowanie elementem jest tablica ogłoszeń o zaginionych zwierzętach. Osoby postronne, niezwiązane ze schroniskiem na co dzień, mogą dodawać ogłoszenia o zaginionych pupilach, co poszerza krąg ludzi spoza organizacji, którzy zapoznają się z ogłoszeniem, a ponadto ułatwia pracownikom odnalezienie właściciela, jeśli szukane zwierzę trafiłoby do danego schroniska. Tablicę ogłoszeń prezentuje Rysunek 67.



Rysunek 67. Tablica ogłoszeń o zaginionych zwierzętach. Źródło: opracowanie własne.

Ogólnodostępnym ekranem jest także lista darczyńców/sponsorów schroniska, gdzie umieszczane są podziękowania dla każdego z nich. Widok ten prezentuje Rysunek 68.



Rysunek 68. Ekran z listą sponsorów. Źródło: opracowanie własne.

7.1.2. Użytkownicy zalogowani

Użytkownikami zalogowanymi mogą być:

- adoptujący,
- pracownicy,

- wolontariusze,
- administratorzy:
    - zwykły,
    - super.

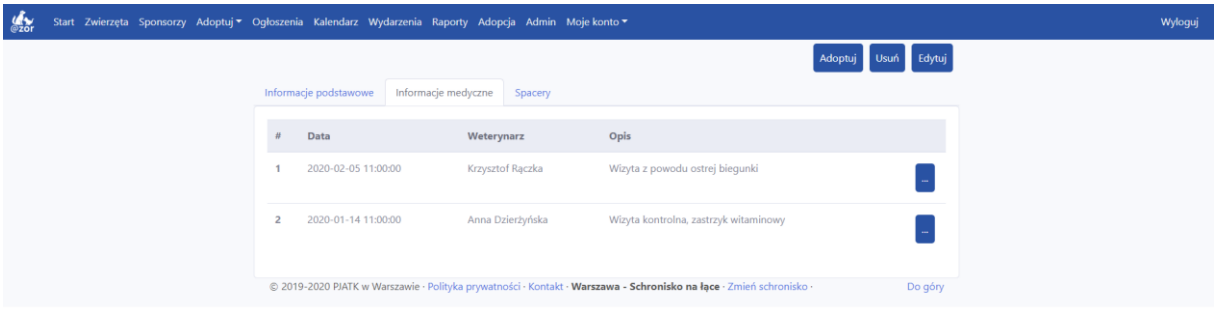
Adoptujący mogą podglądać swoje adopcje: zwykłe i wirtualne. Wolontariusze mają dostęp do modyfikacji kalendarza i wydarzeń oraz incydentów, pracownicy mogą zarządzać całym schroniskiem natomiast administrator rolami i użytkownikami systemu. Rola superadministratora dotyczy dodatkowo dostępu do zarządzania profilami wszystkich schronisk oraz zakładania kolejnych.
Pierwsza strona dla pracownika dotyczy wyboru schroniska z listy dostępnych placówek; po dokonaniu wyboru użytkownik zostaje przekierowany na stronę główną wskazanego miejsca. Ekran ten widoczny jest na Rysunku 69.

Rysunek 69. Ekran wyboru schroniska. Źródło: opracowanie własne.

W ramach powyższego podrozdziału omówione zostały ekrany list: artykułów, zwierząt, w tym zwierząt przeznaczonych do adopcji zwykłej oraz wirtualnej, a także sponsorów. Jak wspomniano dla zalogowanych pracowników różnią się jedynie możliwością dodawania, modyfikacji oraz usuwania obiektów.

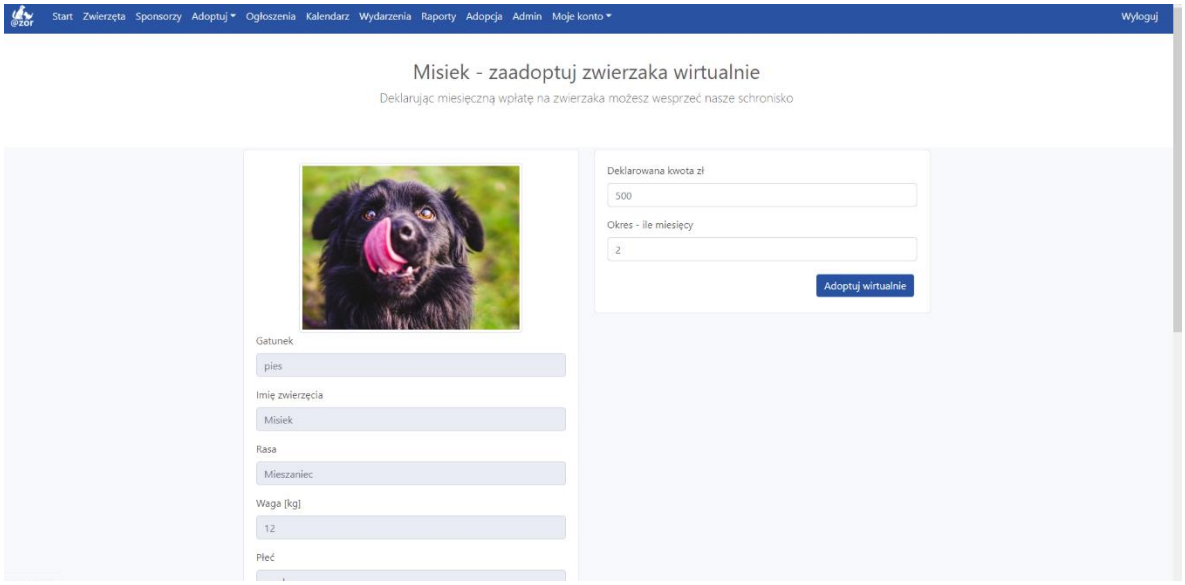
Moduł zwierząt rozbudowany jest na bazie kartoteki zwierzęcia o dodatkowe możliwości reprezentowane w interfejsie jako kolejne zakładki. Zakładki „Informacje medyczne” (przykład prezentuje Rysunek 70) oraz „Spacery” zrealizowane są w formie uproszczonej listy zdarzeń. Lista w tej formie wykorzystywana jest w każdym innym miejscu aplikacji, gdzie potrzebne jest wypunktowanie obiektów określonej klasy (np. incydenty, osoby, adopcje itd.). To kolejny przykład uspójniania

wyglądu programu dzięki czemu nauka stosowania narzędzia, a następnie codzienne użytkowanie stają się szybsze i powtarzalne.

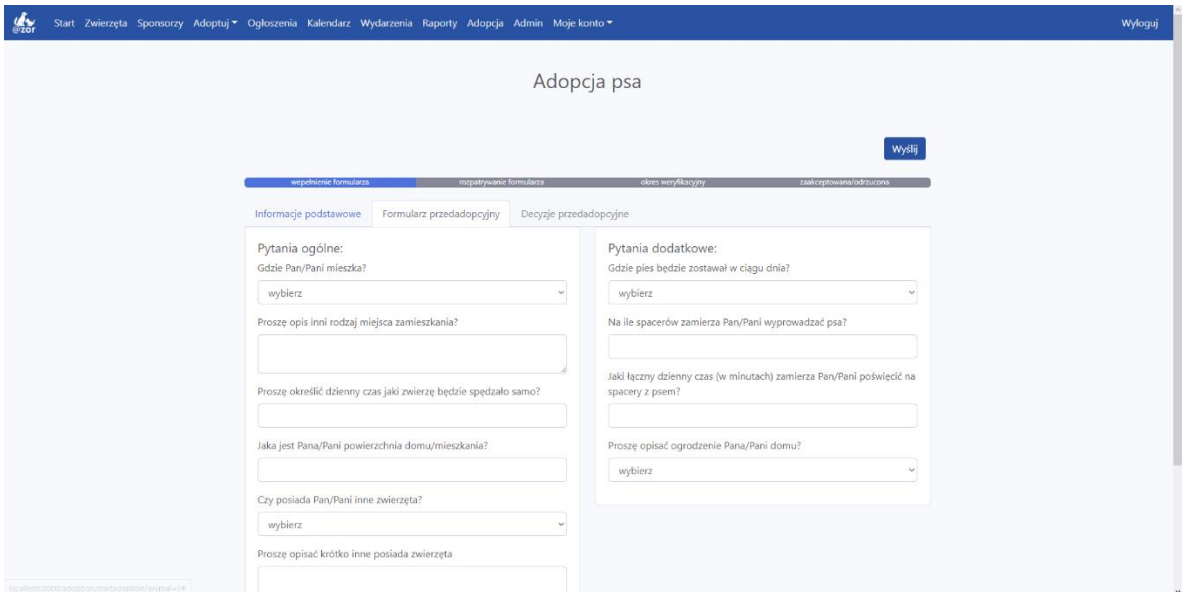


Rysunek 70. Ekran kartoteki zwierzęcia. Informacje medyczne. Źródło: opracowanie własne.

Moduł adopcji jest z kolei wyraźnie różny dla adoptujących i pracowników. W przypadku adopcji wirtualnej po wyborze zwierzęcia i zalogowaniu użytkownik wypełnia deklarację o wpłatach. Jest to uproszczony dokument, gdyż dotyczy jedynie wskazania kwoty oraz częstotliwości wpłat, co prezentuje Rysunek 71. Natomiast w przypadku adopcji klasycznej, użytkownik po zalogowaniu proszony jest o wypełnienie e-formularza (Rysunek 72), który automatycznie trafia do pracowników schroniska w celu weryfikacji i podjęcia dalszych czynności.

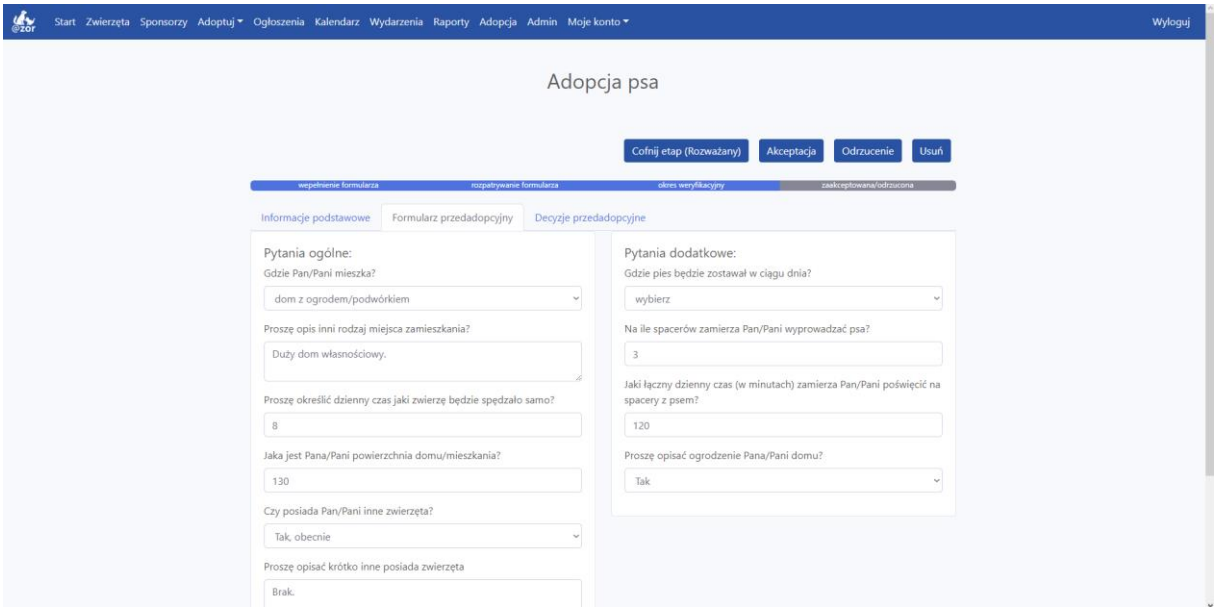


Rysunek 71. Kartoteka zwierzęcia z deklaracją adopcji wirtualnej. Źródło: opracowanie własne.

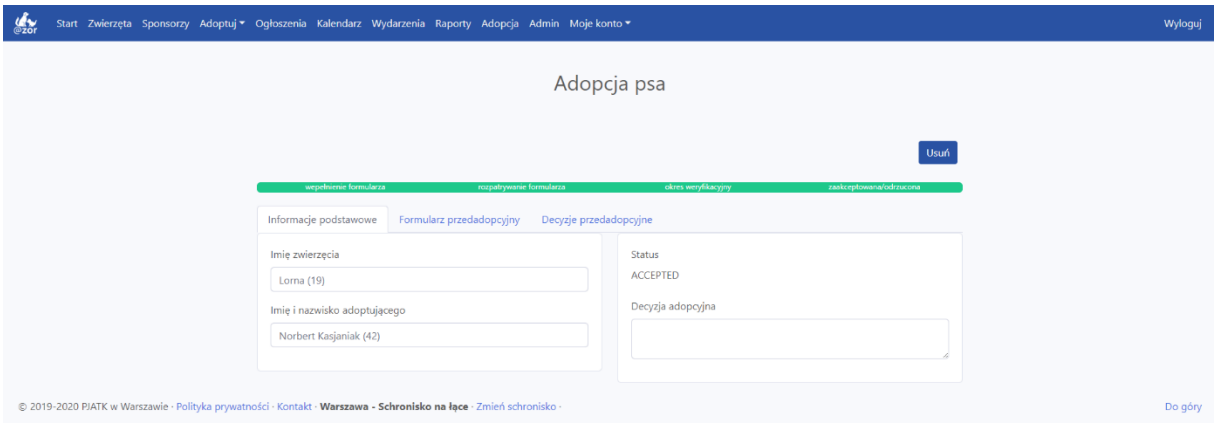


Rysunek 72. Ekran zarządzania adopcją. Formularz przedadopcyjny. Źródło: opracowanie własne.

Adoptujący w zakładce ”Zarządzanie adopcjami” może prześledzić historię swoich adopcji i ich status w formie listy, a pracownik w tej samej zakładce o identycznym sposobie prezentacji wszystkie adopcje wraz z wygodnym filtrem w celu szybkiego odnalezienia tej pożądanej. Ponownie zastosowano tu przedstawiony już format listy z nagłówkiem. Historia procesu adopcji: odbytych wizyt, opinii i podjętych decyzji została ujęta w intuicyjny timeline, gdzie każdy pracownik może wygodnie podejrzeć historię adopcji, jej status czy też ją uzupełnić. Ponownie wykorzystano „kartotekowy” wygląd ekranu z dwoma kolumnami oraz zakładkami. Rysunek 73 oraz Rysunek 74 demonstrują różne etapy adopcji.

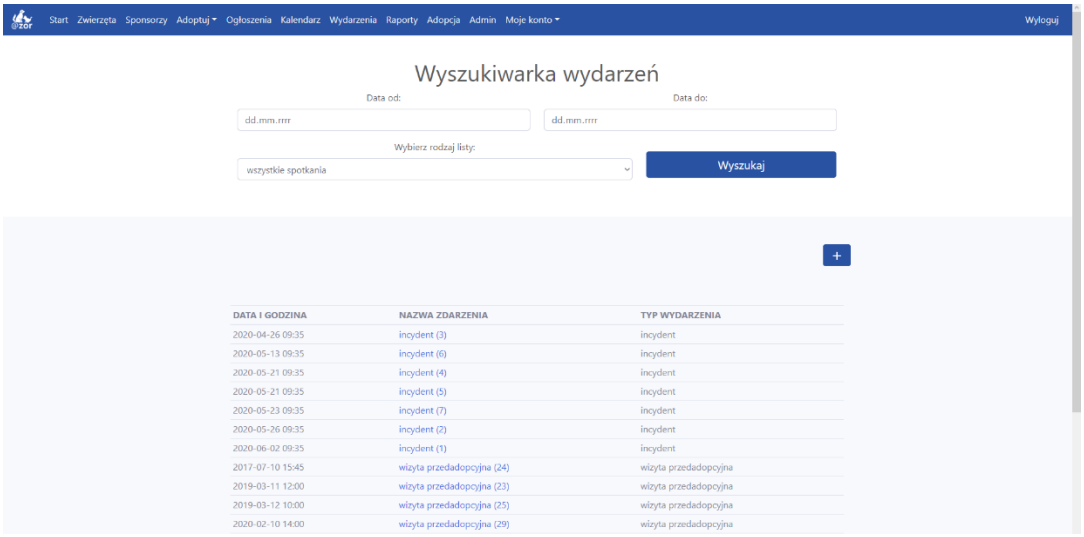


Rysunek 73. Ekran zarządzania adopcją. Etap weryfikacji. Źródło: opracowanie własne.

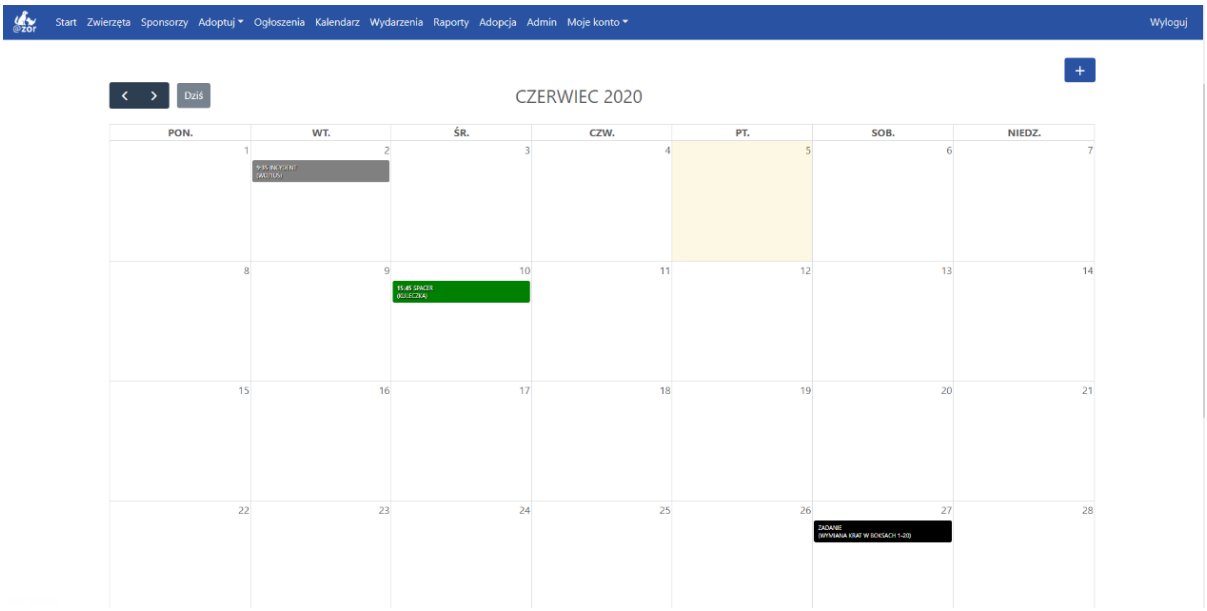


Rysunek 74. Ekran zarządzania adopcją. Etap zaakceptowania adopcji. Źródło: opracowanie własne.

Obsługa zdarzeń, które wymagają odnotowania w systemie jest zrealizowana w formie list lub w wygodnej formie kalendarza, gdzie uprawnieni użytkownicy mogą wyświetlić lub dodać zdarzenia takie jak spacery, wizyty weterynaryjne, przedadopcyjne czy podanie leków. Np. uzupełnienie formularza podawania leków w module medycznym generuje w kalendarzu zwierzęcia przyszłe zdarzenia informujące o konieczności aplikacji leku o określonych porach. Okno listy wydarzeń wraz z filtrami prezentuje Rysunek 75 natomiast formę kalendarzową Rysunek 76.

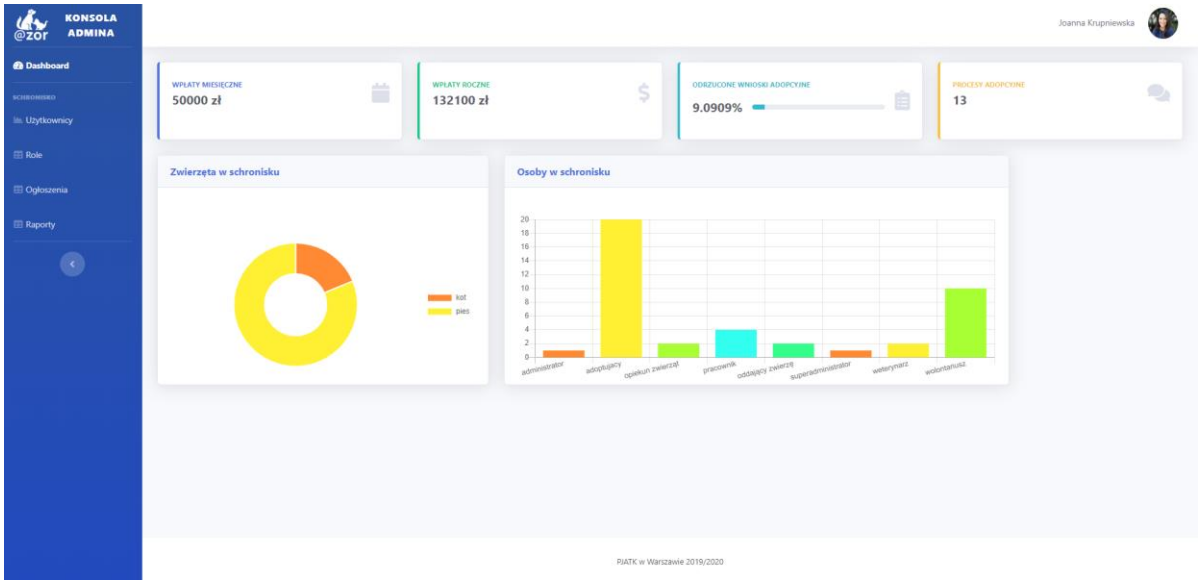


Rysunek 75. Ekran listy zdarzeń. Źródło: opracowanie własne.



Rysunek 76. Ekran kalendarza. Źródło: opracowanie własne.

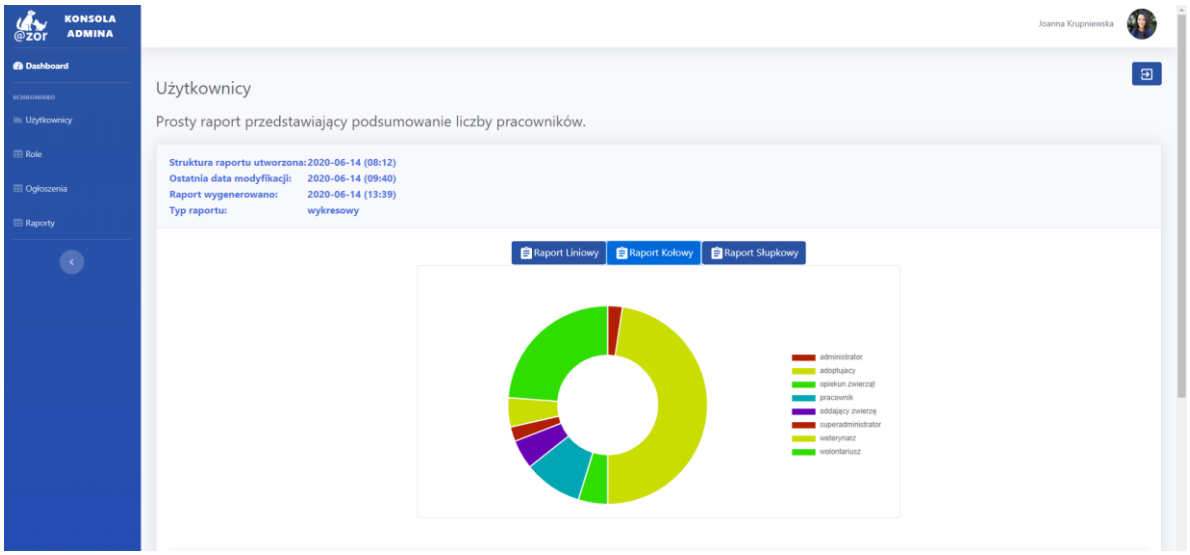
Kolejnym istotnym elementem aplikacji jest zarządzanie osobami. Kokpit administracyjny jest dostępny wyłącznie dla pracowników i administratorów. Ekran kokpitu powitalnego składa się z generowanych automatycznie nowoczesnych w wyglądzie statystyk co prezentuje Rysunek 77. Sam ekran kokpitu różni się od pozostałych stron serwisu rodzajem menu. W celu wygodniejszego korzystania z panelu administracyjnego zastosowano większy i bardziej czytelny pasek boczny. Użytkownicy systemu, ich role oraz raporty dotyczące schroniska są prezentowane w formie już omówionej listy. Dodawanie, edycja czy podgląd realizowane są natomiast przedstawioną przy okazji zarządzania zwierzętami dwukolumnową kartoteką z zakładkami.



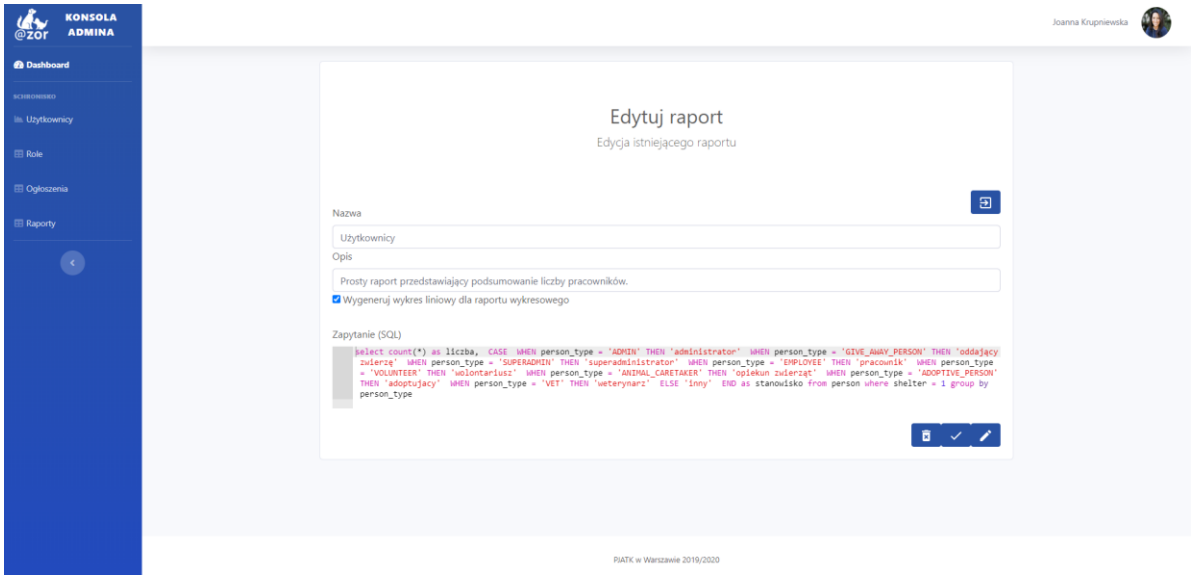
Rysunek 77. Ekran kokpitu. Źródło: opracowanie własne.

Omawiany program oferuje ponadto generowanie raportów dotyczących stanu inwentarza, pracowników, zdarzeń i statystyk. Raporty te, w zależności od potrzeb i uzyskiwanych danych, mogą

być wyświetlane w formie tabelarycznej, atomowej wartości lub jako wykres (kołowy, słupkowy i liniowy). Ponadto zapewniono ich generyczność, dzięki czemu szybko i wygodnie można tworzyć nowe wersje. Specjalny panel administratora umożliwia wprowadzanie zapytań SQL, sprawdzający przy tym poprawność składni i nazw kolumn oraz tabel. System na podstawie zapytania sam rozpozna w jakiej formie może przedstawić wyniki oraz sprawdzi i poprawi ewentualny wybór dokonany uprzednio przez administratora. Rysunek 78 prezentuje przykładowy wygenerowany raport w formie wykresu kołowego, a Rysunek 79 panel umożliwiający wprowadzanie i edycję raportów.



Rysunek 78. Wykres na podstawie wygenerowanego raportu. Źródło: opracowanie własne.

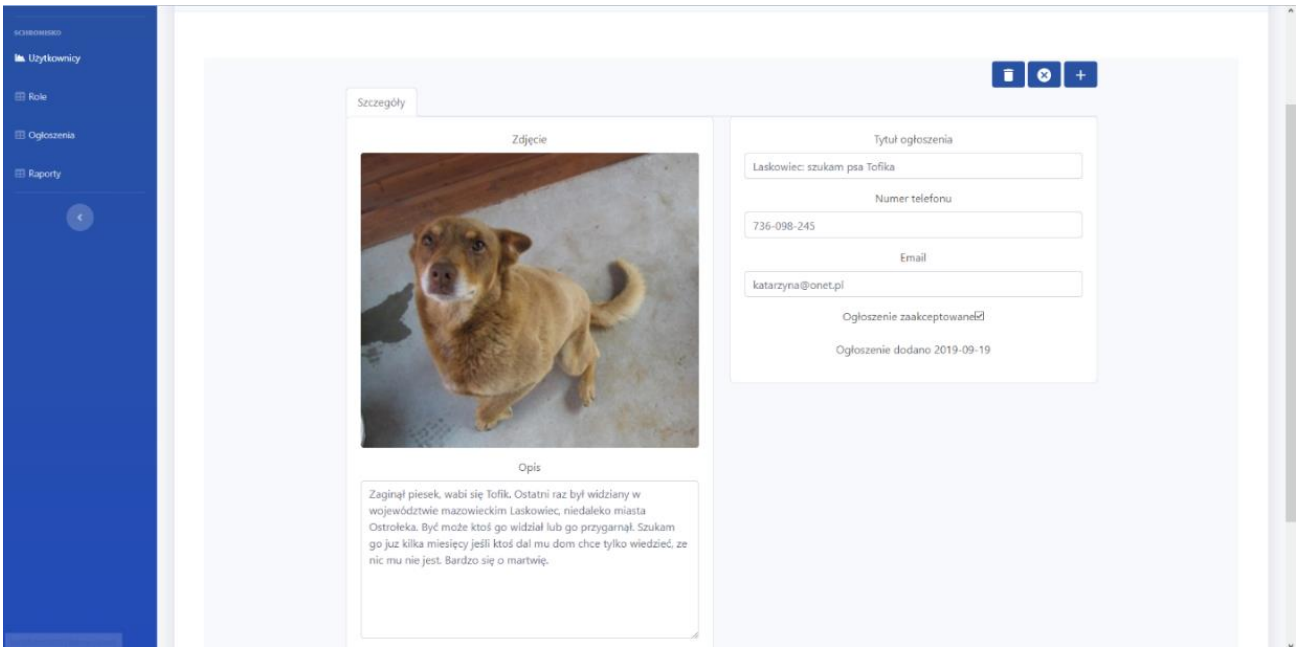


Rysunek 79. Panel umożliwiający wprowadzanie i edycję raportów. Źródło: opracowanie własne.

Poza koncepcją adopcji wirtualnej aplikacja umożliwia pracownikom łatwe wyróżnienie klasycznych darczyńców w formie listy udostępnionej na stronie schroniska w zakładce Sponsorzy, co zaprezentowane zostało w poprzednim podrozdziale [zob. 7.1.1]. Zakładka każdego sponsora pozwala

pracownikowi zalogowanemu do systemu na edycję danych, dodanie kolejnej darowizny oraz prześledzenie historii wpłat.

Omówione wcześniej ogłoszenia o zaginionych zwierzętach dodawane przez osoby postronne nie trafiają prosto na stronę internetową lecz najpierw przekazywane są do pracowników w celu akceptacji lub odrzucenia. Ma to na celu eliminację błędnych lub niepożądanych treści. Ekran weryfikacji ogłoszenia w formie kartoteki prezentuje Rysunek 80.



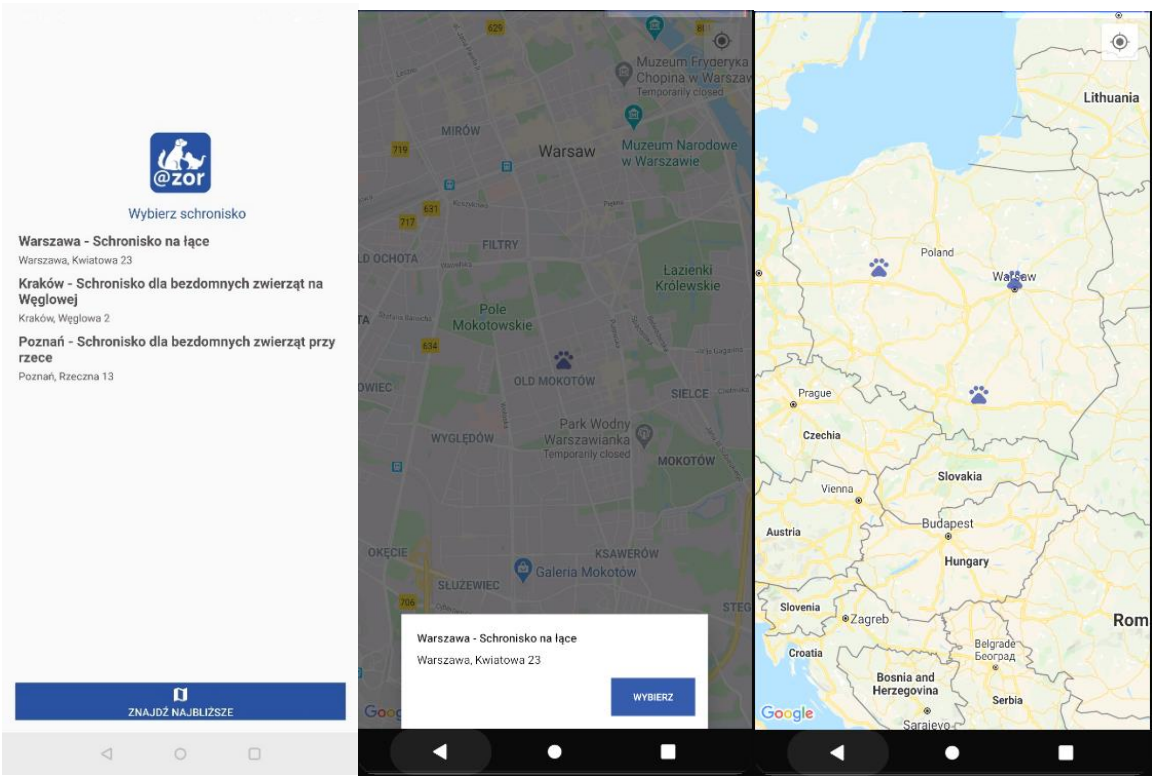
Rysunek 80. Ekran weryfikacji ogłoszenia. Źródło: opracowanie własne.

7.2 Aplikacja mobilna

Poza klasyczną aplikacją webową postanowiono, zgodnie ze współczesnymi trendami, udostępnić program również na smartfony. Aby zagwarantować prawidłową skalowalność, proporcje oraz wygodę obsługi zaprogramowano całkowicie niezależny interfejs użytkownika za pomocą narzędzia Android SDK. Zakres dostępnych funkcjonalności został dostosowany do tych, które rzeczywiście mogą przydać się użytkownikowi poza siedzibą schroniska. Ograniczenie elementów sprzyja także czytelności stron wyświetlanych na ekranach o niewielkiej przekątnej.

W aplikacji mobilnej udostępniono następujące elementy:

1. Ekran powitalny, gdzie mamy możliwość wyboru schroniska z listy lub mapy, co demonstruje Rysunek 81.

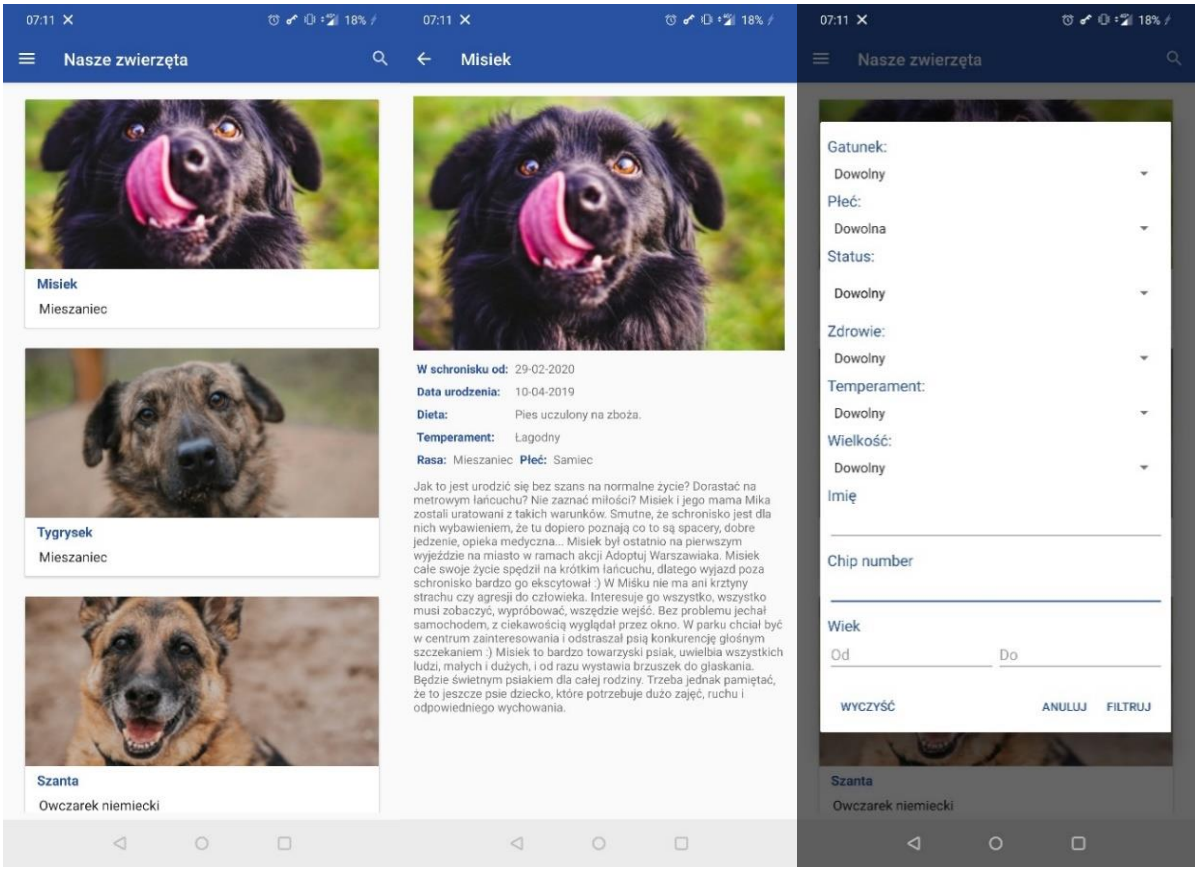


Rysunek 81. Ekran powitalny aplikacji mobilnej. Źródło: opracowanie własne.

2. Strona główna, na której wyświetlane są najnowsze informacje oraz artykuły.
   Po dokonaniu wyboru użytkownik przenoszony jest do konkretnego artykułu. Zastosowano tu slider: aby oglądać kolejne informacje należy przesunąć palcem w bok. Przykład takiego artykułu prezentuje Rysunek 82.

Rysunek 82. Artykuł wyświetlony w aplikacji mobilnej. Źródło: opracowanie własne.

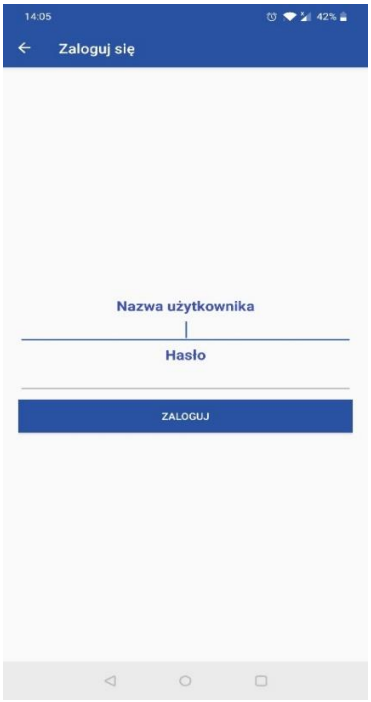
3. Zwierzęta – lista zwierząt przebywających w schronisku.
   Zachowana jest możliwość filtrowania po atrybutach. Wybór konkretnego zwierzęcia przenosi użytkownika do szczegółowej kartoteki. Rysunek 83 prezentuje listę oraz kartotekę wybranego zwierzęcia.



Rysunek 83. Ekrany listy zwierząt oraz kartoteka zwierzęcia. Źródło: opracowanie własne.

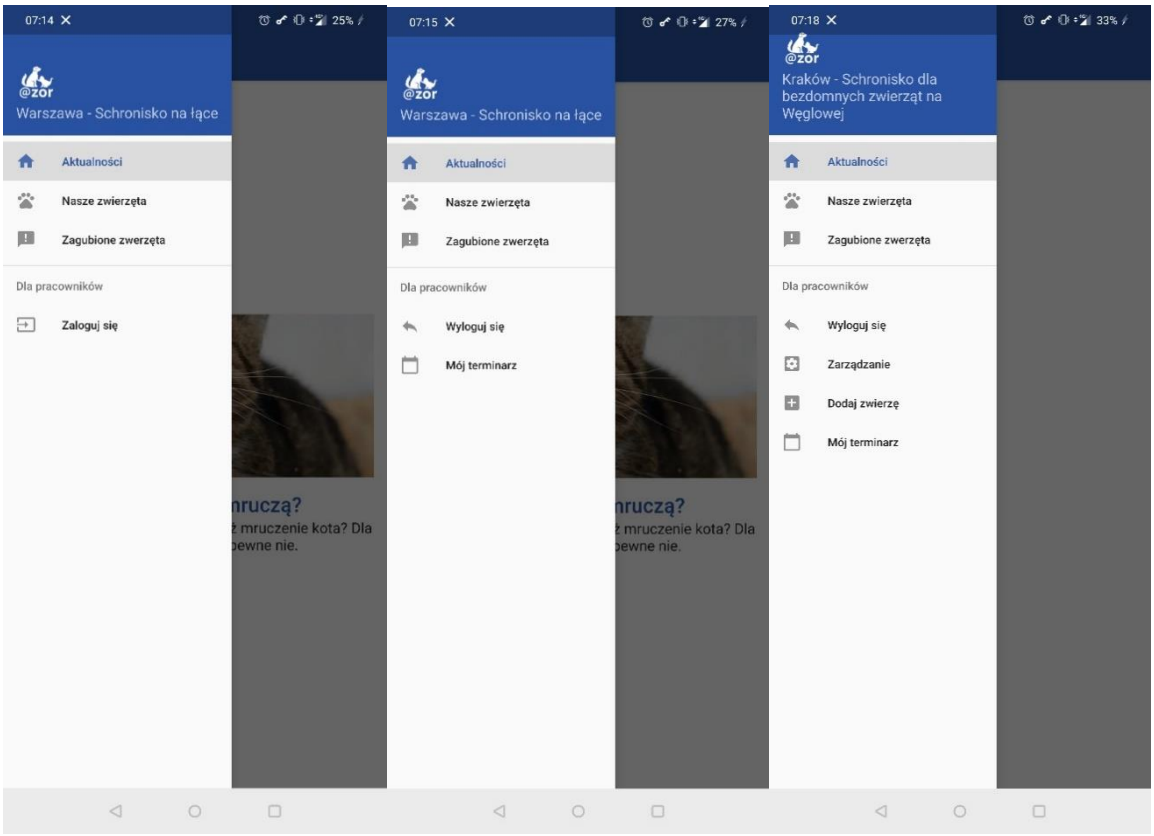
4. Ogłoszenia o zaginionych zwierzętach.
   Tu wyświetlana jest lista ogłoszeń o zaginionych zwierzętach. Można także dodać nowe ogłoszenie do listy wraz ze zdjęciem pupila, co prezentuje Rysunek 84.

Rysunek 84. Ekrany dodawania ogłoszenia o zaginionym zwierzęciu. Źródło: opracowanie własne.



Rysunek 85. Ekrany logowania. Źródło: opracowanie własne.

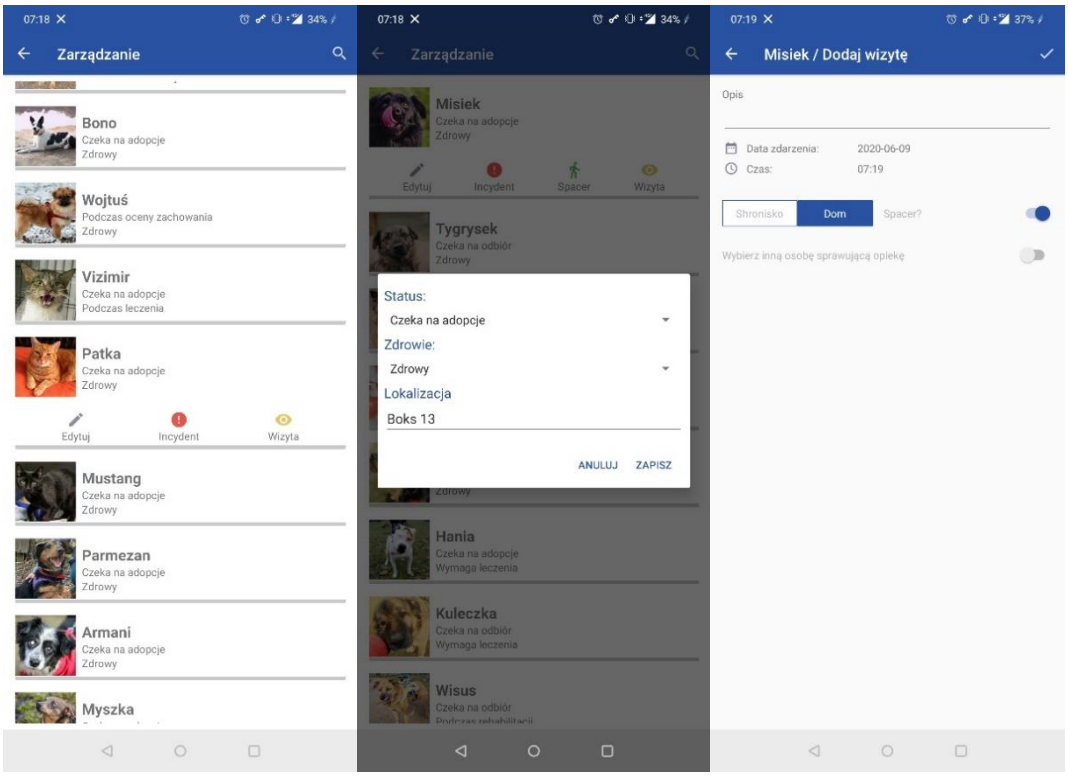
5. Ekran logowania dedykowany dla pracowników schroniska prezentuje Rysunek 85.
6. Wygląd menu został przemodelowany pod system Android. Rysunek 86 prezentuje menu dostosowane do uprawnień zarejestrowanego użytkownika.

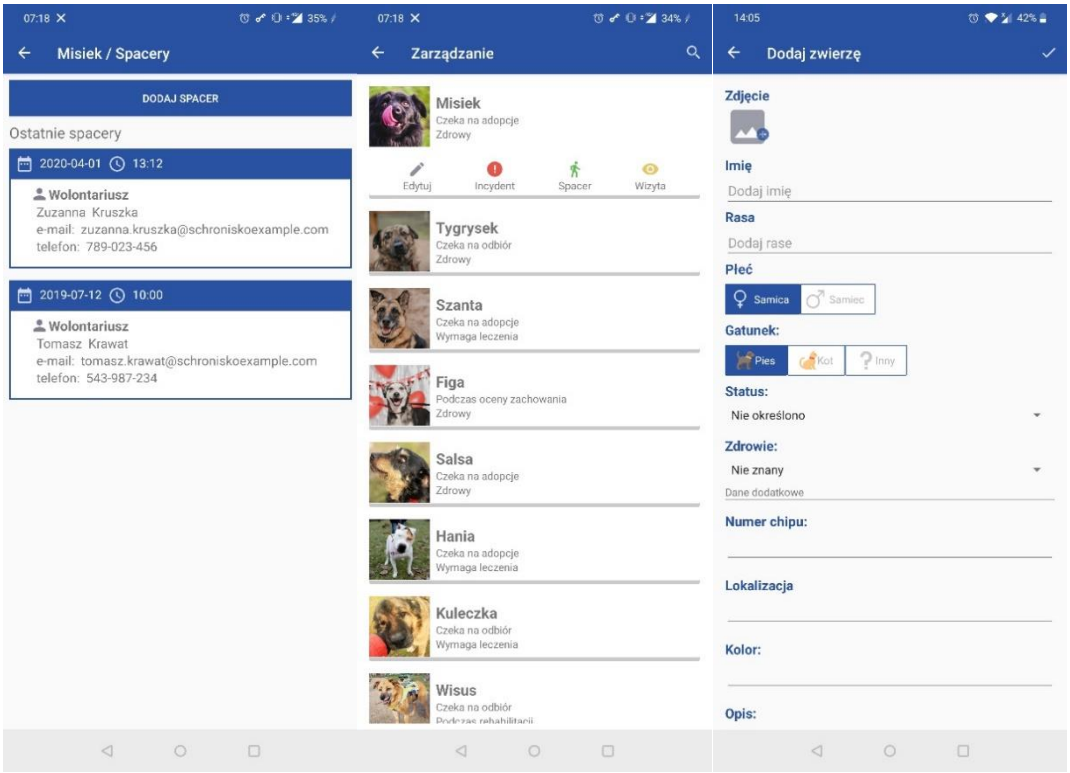


Rysunek 86. Ekrany menu. Źródło: opracowanie własne.

7. Główne funkcjonalności zarządzania schroniskiem po zalogowaniu.
   Udostępnione zostały tu
   - lista zwierząt w schronisku,
   - podgląd kartoteki konkretnego zwierzęcia (z uwzględnieniem gatunku),
   - edycja wybranych atrybutów,
   - dodanie incydentu,
   - wylistowanie wszystkich spacerów,
   - dodanie spaceru,
   - wylistowanie wszystkich wizyt przedadopcyjnych,
   - dodanie kolejnej wizyty do kalendarza.

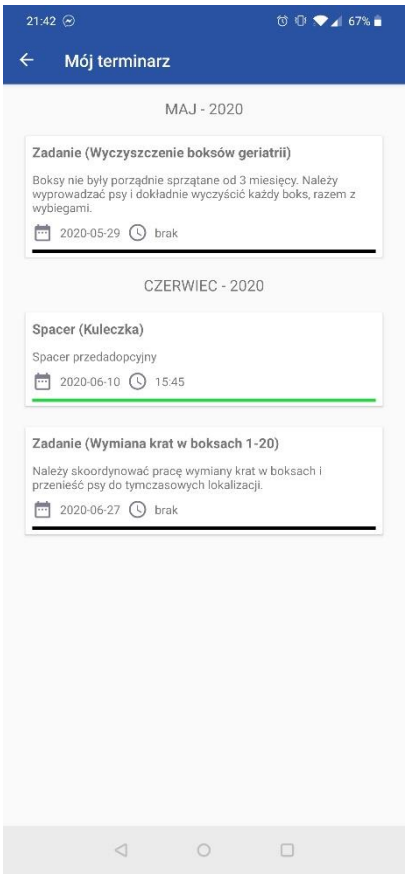
   Ekrany dla wymienionych funkcjonalności prezentuje Rysunek 87.





Rysunek 87. Ekrany funkcjonalności zarządzania schroniskiem. Źródło: opracowanie własne.

8. Kalendarz z wydarzeniami/zadaniami przypisanymi konkretnemu pracownikowi, co prezentuje Rysunek 88.



Rysunek 88. Ekran zarządzania kalendarzem. Źródło: opracowanie własne.

8. Testowanie aplikacji

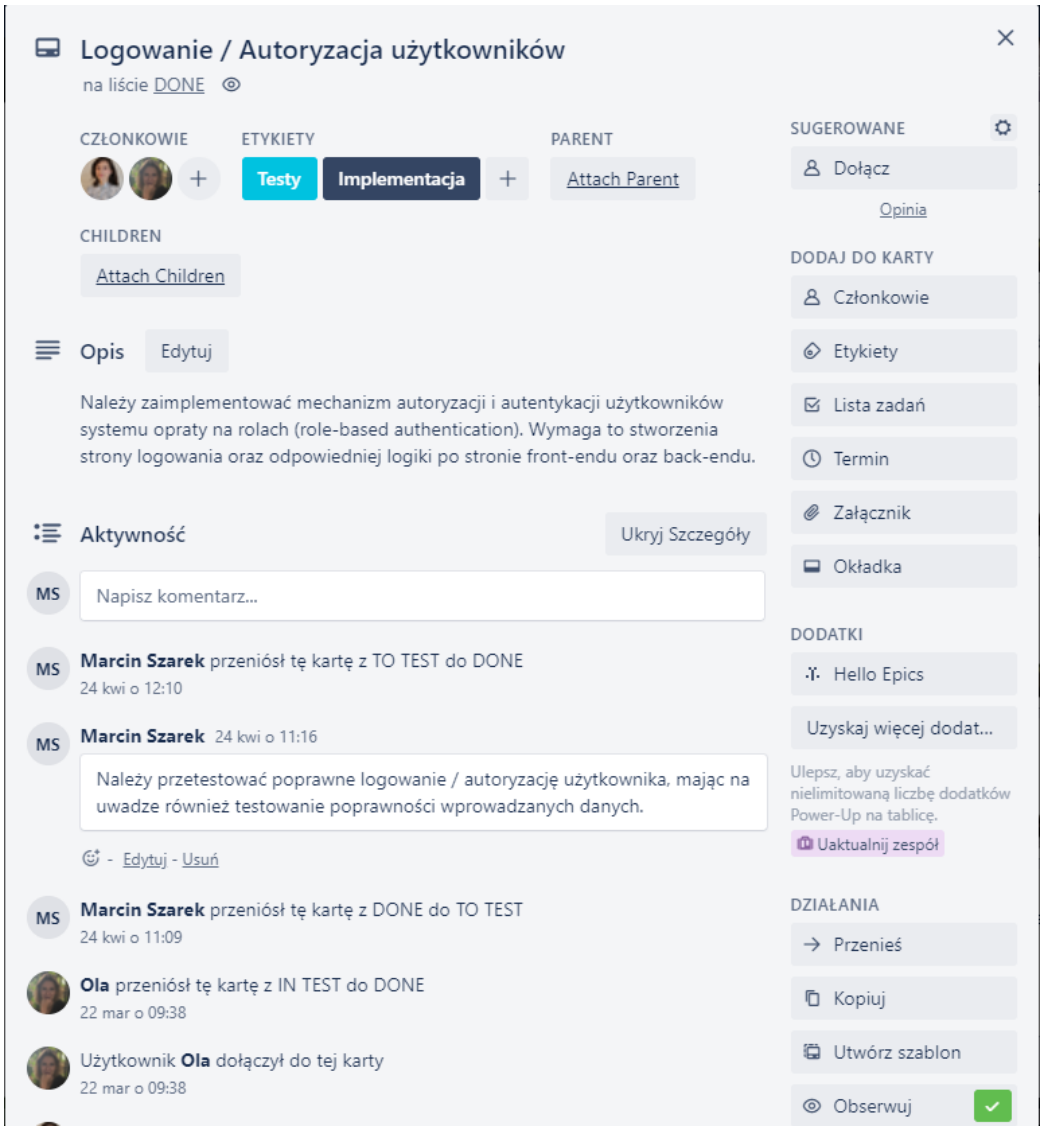
W dotychczasowej historii informatyki można spotkać znaczną ilość przykładów wytworzonych programów i systemów, które ze względu na brak przeprowadzonych testów lub zastosowanie niewłaściwych, nie spełniły oczekiwań użytkowników i zostały porzucone lub po prostu ulegały awariom. Zastosowanie odpowiednich technik testowania dobranych pod konkretne oprogramowanie oraz potrzeby klienta a także testowanie we właściwych fazach cyklu życia programu mogą znacząco zredukować liczbę tego rodzaju problemów. Podsumowując, testowanie ma na celu weryfikację oraz walidację oprogramowania [86].

Testy aplikacji rozpoczęto niezwłocznie po powstaniu jej pierwszych elementów oraz funkcjonalności by zapobiec dalszemu rozwijaniu programu na wadliwym kodzie lub niewłaściwie realizowanych przypadkach użycia. Testowanie przeprowadzone zostało na poziomach integracyjnym i systemowym z zastosowaniem testów manualnych i automatycznych. Każdy test manualny dokumentowany był w postaci karty na ścieżce przepływu w aplikacji Trello. Przykład karty omówiony i zaprezentowany został poniżej na Rysunku 89.

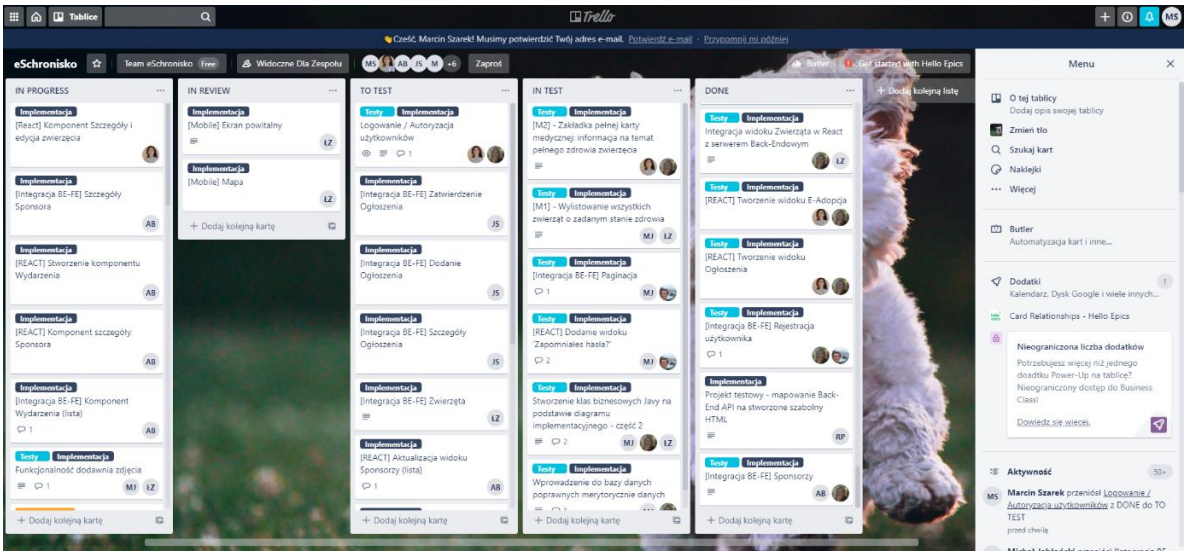
8.1. Manualne testy integracyjne i systemowe

Integracyjne testy manualne użyte zostały do sprawdzania poszczególnych fragmentów aplikacji poprzez realizację zaplanowanych scenariuszy testowych powstałych na bazie oczekiwanych funkcjonalności. Testowano zgodność produktu z wymaganiami oraz jednocześnie, gdzie było to możliwe, integrację front-endu/back-endu/bazy danych. Zastosowano metodę czarnej skrzynki, tj. testerzy nie znali wytworzonego kodu stojącego za danym zadaniem a jedynie realizowali dostarczony scenariusz. Poniżej zaprezentowany został przykład testowania funkcjonalności „Logowanie/Autoryzacja użytkowników”.

1. Karty zrealizowanych funkcjonalności oczekujących na test znajdowały się w zakładce „To Test” w aplikacji Trello. Oznaczało to, że kod dotyczący tej funkcjonalności został wystawiony przez osobę implementującą do zatwierdzenia przez kierownika zespołu implementacji, zaakceptowany, scalony oraz wdrożony na serwer. Przypisany zadaniu tester pobierał kartę funkcjonalności i zapoznawał się z jej treścią, gdzie podane były założenia funkcjonalności oraz elementy wymagające przetestowania (Rysunek 89); tester następnie umieszczał kartę w zakładce „In Test”, co prezentuje Rysunek 90.

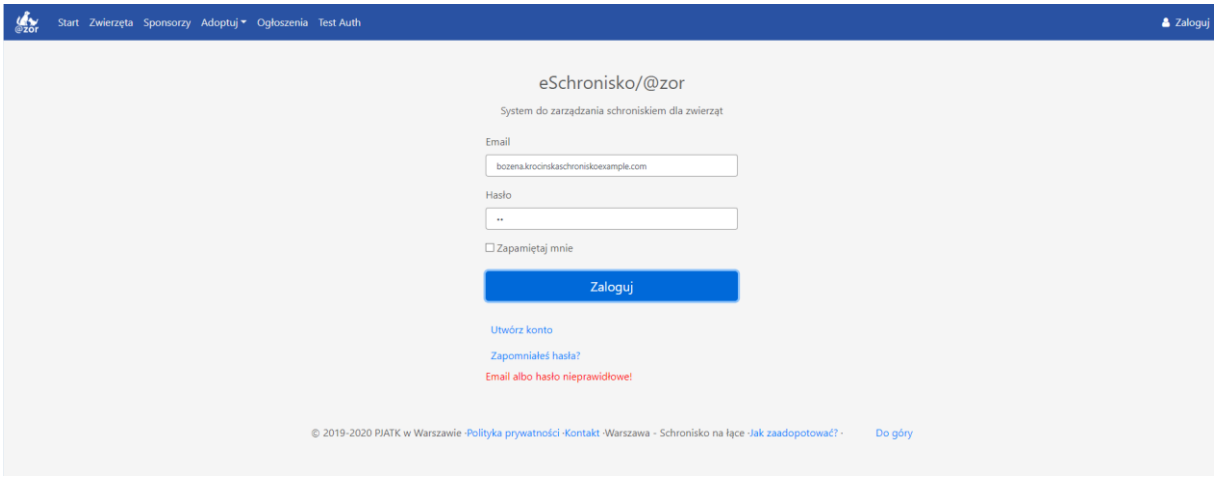


Rysunek 89. Widok karty w aplikacji Trello. Źródło: opracowanie własne.



Rysunek 90. Ekran główny aplikacji Trello. Widok na zakładki. Źródło: opracowanie własne.

2. Tester realizował założony scenariusz w uruchomionej aplikacji, co prezentuje Rysunek 91. Na załączonym przykładzie tester sprawdzał czy aplikacja waliduje poprawność wprowadzonych danych przez klienta. Aplikacja informuje użytkownika, iż został użyty niepoprawny adres email.



Rysunek 91. Ekran logowania testowanej aplikacji. Źródło: opracowanie własne.

3. Jeżeli przeprowadzony test nie wykazywał nieprawidłowości tester umieszczał kartę funkcjonalności w zakładce „Done”. Jeżeli test wypadł niepomyślnie tester umieszczał adnotację o odnalezionej usterce i przekazywał kartę z powrotem do zakładki „To Do”.

8.2. Manualne testy API za pomocą narzędzia Postman

Kolejny przykład prezentuje integracyjny test manualny przeprowadzany za pomocą narzędzia Postman. Operacje przeprowadzane w tej aplikacji służyły przede wszystkim do testowania API, a jednocześnie sprawdzano integrację z bazą i poprawność zwracanych danych. Przykład dotyczy przetestowania zapytania zwracającego listę wszystkich zarejestrowanych użytkowników systemu. Realizuje to metoda `get-all-persons` uwzględniająca paginację okien w aplikacji.

1. Tester podobnie jak poprzednio pobierał kartę z zakładki „To test” w aplikacji Trello, zapoznawał się z jej treścią, gdzie podane były metody wymagające przetestowania, a następnie umieszczał kartę w zakładce „In Test”.
2. Tester sprawdzał jaka powinna zostać otrzymana odpowiedź zwrotna z serwera.
3. Tester wysyłał zapytanie do bazy, a zwracaną odpowiedź weryfikował. Przykład prezentuje Rysunek 92, gdzie tester wysłał zapytanie *get-all-persons* i otrzymał zwrotnie listę w formacie JSON.

Rysunek 92. Ekran aplikacji Postman. Wysyłanie zapytania na serwer. Źródło: opracowanie własne.

4. Podobnie jak poprzednio, jeżeli przeprowadzony test nie wykazywał nieprawidłowości, tester umieszczał kartę w zakładce „Done”. Jeżeli test wypadł niepomyślnie tester umieszczał adnotację o odnalezionej usterce i przekazywał kartę z powrotem do zakładki „To Do”.

8.3. Automatyczne testy integracyjne

Poza testami manualnymi przeprowadzono także szereg testów automatycznych. Dotyczyły one integracji bazy danych z back-endem aplikacji. Zostały zastosowane w celu sprawdzenia stanu aplikacji po wprowadzeniu zmian w kodzie (tzw. testy regresji), dzięki temu szybko można było sprawdzić czy nadal działają kluczowe metody API aplikacji. Testy te przeprowadzano za pomocą narzędzia IDE Intellij IDEA przy użyciu języka Groovy [87], frameworka Rest-assured [88] oraz Spock [89]. Ten ostatni jest biblioteką służącą do testowania i tworzenia specyfikacji, natomiast Rest-assured umożliwia łatwe i czytelne wykonywanie metod http w kodzie Groovy oraz wbudowane asercje wykonywane bezpośrednio na otrzymywanych odpowiedziach od serwera. Zdecydowano się na te narzędzia ze względu prostotę implementacji i szybkość wykonywania testów (te korzyści uzyskano dzięki językowi Groovy) oraz możliwość tworzenia scenariuszy czytelnych z biznesowego punktu widzenia dzięki bibliotece Spock. Korzystano także z dostępnej w tej bibliotece konwencji „Given, When, Then”, która umożliwia podział testu na czytelne grupy kodu a także z testowania sterowanego danymi (ang. data driven testing) za pomocą tabel danych. W tabelach danych pierwszy rząd deklaruje zmienne zaś kolejne zawierają wartości przypisywane do tych zmiennych. Metoda dla której istnieje taka tablica wykona się raz dla każdego rzędu zawierającego wartości przypisując je po kolei do zmiennych w każdej iteracji. Przykład przygotowania oraz wykonania takich testów opisany został poniżej. Dotyczy on przetestowania metod CRUD dla klasy `Animal`.

1. Listing 17 prezentuje klasę `AnimalService` przeznaczoną do testowania zapytań http dla klasy `Animal`. Fragment kodu przedstawia metodę dodającą obiekt klasy `Animal` do bazy danych. Dzięki użyciu frameworka REST-assured osiągnięcie tego celu upraszczało pracę implementującego oraz było wygodne, jakkolwiek wymagało uprzednio przeszkolenia zespołu w tej dziedzinie. Konfiguracja adresu url wykonywana była raz w metodzie `setup()` uruchamianej przed wszystkimi testami. W teście podawało się końcówkę adresu do endpointu jako string. Metoda jako parametr przyjmowała obiekt `Animal`, który był od razu parsowany

na format JSON i przekazywany w ciele zapytania. Możliwe było wykonanie walidacji oraz parsowania uzyskanej odpowiedzi na obiekt.

Listing 17. Fragment kodu klasy `AnimalService`. Źródło: opracowanie własne.

```
Animal updateAnimal(Animal animalWithNewProperties) {
        given()
                .header('Authorization', "Basic $token")
                .header('content-type', 'application/json')
                .body(animalWithNewProperties)
                .when()
                .put('/api/animals')
                .then()
                .statusCode(200)
                .log().ifError()
                .extract().response().as(Animal.class)
    }
```

2. Tester implementował na bazie wzorca projektowego „Builder" klasy „Budowniczych" za pomocą których możliwe było tworzenie i modyfikowanie obiektów testowanej klasy. Przykład kodu dla klasy `AnimalBuilder` prezentuje Listing 18. Fragment kodu zawiera konstruktor tworzący domyślny obiekt klasy `Zwierzę` oraz metody umożliwiające modyfikacje wybranych atrybutów obiektu.

Listing 18. Fragment kodu klasy `AnimalBuilder`. Źródło: opracowanie własne.

```
AnimalBuilder(){
        animal = new Animal(
                name: 'Azor',
                breed: 'owczarek',
                weightKg: 7.5,
                gender: 'MALE',
                dateOfBirth: '2010-01-10T10:10:00',
                dateOfDeath: null,
                chipNumber: '789798',
                photo: 'https://www.google.com/urlToPhoto',
                videoUrl: null,
                diet: 'Pescowegetarianizm',
                temper: 'LIKES_KIDS',
                status: 'BEHAVIOUR_EVALUATION',
                species: 'DOG',
                location: 'pokoj 2',
                description: 'lubi sie bawic',
                listEvent: [],
                age: 10)
    }

    static AnimalBuilder aAnimal(){
        return new AnimalBuilder()
    }

    AnimalBuilder withName(String name){
        animal.name = name
        return this
    }

    AnimalBuilder withBreed(String breed){
        animal.breed = breed
        return this
    }
```

3. Dla każdej testowanej klasy stworzona została odpowiednia klasa testująca. Listing 19 prezentuje fragment kodu klasy `AnimalSpec`, która testuje CRUD klasy `Animal`. Przedstawiony fragment sprawdza poprawność zapisania w bazie danych zmiany obiektu. Widoczny jest podział na sekcje: „Given" – przygotowanie danych do testu, „When" – wykonanie testowanej akcji, „Then" – wykonanie asercji i walidacja poprawności testowanej metody. We fragmencie „Given" tworzony jest obiekt klasy „Animal" z domyślnymi atrybutami z konstruktora oraz pobierane jest id tego obiektu z bazy. „When" zawiera kod odpowiadający za modyfikację obiektu. Tworzony jest nowy obiekt ze zmienionymi atrybutami *`name`*, *`weight`* oraz *`location`*, a następnie ten obiekt zastępuje uprzednio stworzony. Kod ograniczony „Then" prezentuje asercję. Użytkownik otrzymywał wynik testu. Dodatkowy warunek „Where" na Listingu 19 zawiera omówioną tabelę danych, gdzie pierwszy rząd to nazwy zmiennych a kolejne przypisywane zmiennym wartości. W podanym przykładzie kod testu wykonałby się dwa razy: za pierwszym razem podmieniane zwierzę miałby na imię Czarek zaś w drugiej iteracji Tola.

   Tester miał możliwość uruchomienia wszystkich testów na raz, a po ich wykonaniu otrzymywał informację o rezultacie dla każdej badanej metody. Rysunek 93 prezentuje wynik przeprowadzonego testu dla klas `Animal`, `Employee` i `Adoption`.

Listing 19. Fragment kodu klasy „AnimalSpec". Źródło: opracowanie własne.

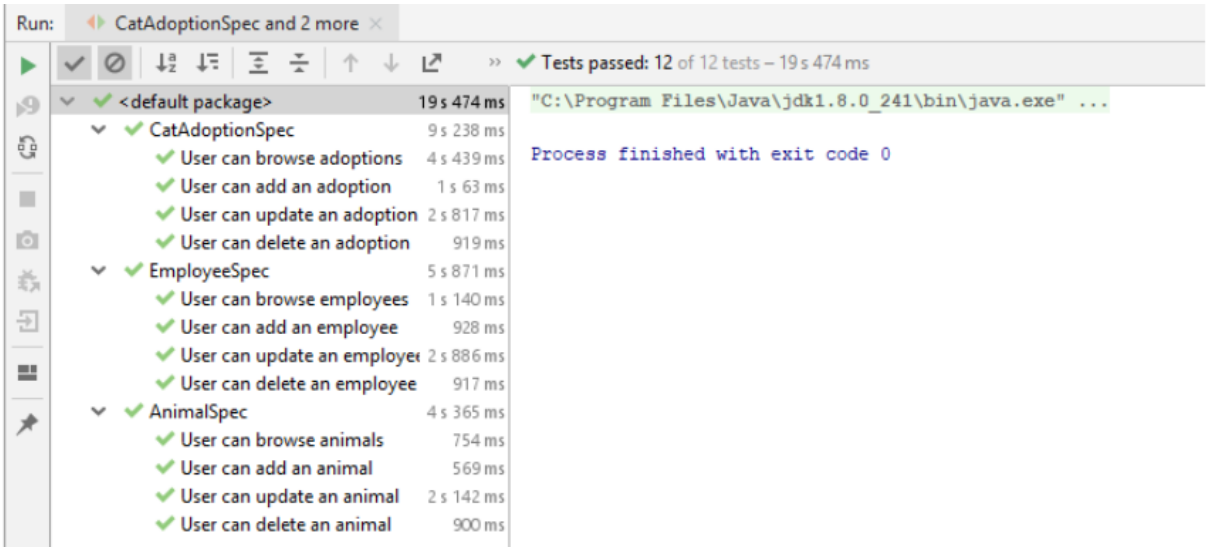
```
def 'User can update an animal' (){

        given: 'User has an animal'
            Animal animal = new AnimalBuilder().build()
            int id = animalService.addAnimal(animal).id

        when: 'User updates an animal'
            Animal animalWithNewProperties = new AnimalBuilder()
                    .withName(name)
                    .withWeightKg(weight)
                    .withLocation(location).build()
            animalWithNewProperties.setId(id)
            animalService.updateAnimal(animalWithNewProperties)

        then: 'Animal has new properties'
            Animal updatedAnimal = animalService.getAnimalById(id)
            expectAnimalWithNewProperties(updatedAnimal, name, weight, location)

        where:
            name     |   weight |    location
            'Czarek'|   30.5   |    'nowy dom'
            'Tola'   |   12.0   |    'box 7'
    }
```



Rysunek 93. Rezultat testów. Źródło: opracowanie własne.

Przykłady scenariuszy przeprowadzonych testów znajdują się w Dodatku B do niniejszej pracy.

9. Podsumowanie prac nad aplikacją

Proces tworzenia aplikacji od postaw, zawierający cały cykl powstawania oprogramowania, okazał się zadaniem skomplikowanym i czasochłonnym. Mimo, iż projektem zajmowało się jedenaście osób przez dziewięć miesięcy, to prace trwały do ostatniego dnia. Niemniej było to ciekawe i rozwijające doświadczenie pozwalające wszystkim członkom działać w każdym aspekcie z jakim mają do czynienia prawdziwe zespoły informatyczne: od badania rynku czy analizy konkurencji, przez etapy powstawania funkcjonalności, implementację aż po wykorzystanie miękkich umiejętności jak szeroko rozumiana praca w zespole czy organizacja czasu. W kolejnych podrozdziałach opisane zostały zebrane doświadczenia oraz pomysły związane z przyszłym rozwojem projektu.

9.1. Wprowadzenie

Zespół projektowy składał się z jedenastu studentów informatyki, którzy posiadali różny stopień wiedzy dotyczący tworzenia oprogramowania. Nikt z zespołu nie miał jednak wystarczających umiejętności programistycznych by móc pracować bez dodatkowego wdrażania się w użytkowane narzędzia i technologie. Kilku członków miało pewne zawodowe doświadczenie deweloperskie w języku Java i bibliotece Hibernate, jednak reszta posiadała wyłącznie wiedzę zdobytą podczas studiów. Natomiast nikt z zespołu nie pracował do tej pory w wielu zastosowanych technologiach i dziedzinach jak JavaScript i jej biblioteki (przede wszystkim React.js czy Node.js), HTML, CSS, administrowanie bazą danych, REST API, Android SDK. Jako tester pracowały zawodowo dwie osoby. Dlatego kilka pierwszych zajęć poświęconych było na zapoznanie się zespołu (większość osób nie znała się wcześniej), rozpoznaniu mocnych i słabych stron, wstępnym podziale obowiązków, wybraniu technologii oraz tematyki projektu. Następnie równocześnie z analizą i badaniem stanu sztuki rozpoczął się szereg szkoleń realizowanych grupowo i indywidualnie by poznać technologie, w których pracować mieli poszczególni członkowie. Jednocześnie rozpoczęto pracę nad interfejsem w HTML/CSS oraz dokumentacją i nad przygotowaniem stacku technologicznego oraz konfigurowaniem narzędzi. Kolejne miesiące aż do końca projektu zajęła implementacja, wypełnianie bazy danych, wdrażanie interfejsu, równocześnie powstawała dokumentacja oraz przeprowadzano testy oddawanych funkcjonalności systemu.

9.2. Organizacja pracy

Wszyscy uczestnicy projektu zostali podzieleni na kilka zespołów: analizy, projektowania, implementacji, testów oraz dokumentacji, a na czele stanął kierownik odpowiadający za całość i raportujący przebieg prac. Zastosowano prowadzenie projektu w trybie kaskadowym. Zespoły zazębiały się, tj. każdy był członkiem więcej niż jednego zespołu. Największy był ten odpowiedzialny za implementację. Zasadniczo skład poszczególnych działów nie ulegał zmianie, niemniej na pewnym etapie prac okazało się, że kluczowe funkcjonalności nie są rozwijane w bezpiecznym dla

harmonogramu tempie i konieczne stało się, by osoby nie zajmujące się wcześniej implementacją zostały zaangażowane w rozwój aplikacji. Intensyfikacja działań deweloperskich po zakończeniu poprzednich faz, a w konsekwencji równoległa praca wielu programistów nad projektem, wymagała większej koordynacji, wykorzystania narzędzia do wersjonowania kodu git [90] i pracy na osobnych gałęziach, a mimo to nie uniknięto problemów z rozwiązywaniem konfliktów w kodzie, co czasami skutkowało powstawaniem błędów w aplikacji. W związku z brakiem wystarczających kompetencji programistów w wybranych obszarach wpływających na opóźnienia w dostarczaniu przydzielonych fragmentów prac, doszło do niewielkich zmian w składzie zespołów i przydziale zadań. Ponadto zastosowany tryb kaskadowy okazał się mało odporny na wprowadzanie zmian. Niemniej zespół pracował sprawnie, nie dochodziło do konfliktów czy przypadków unikania odpowiedzialności. Należy zwrócić uwagę, iż aplikacja powstawała w unikalnych okolicznościach gdy Światowa Organizacja Zdrowia ogłosiła pandemię wirusa COVID-19, a Rząd RP wprowadził daleko idące obostrzenia w grupowaniu się społeczeństwa, a także zakazał prowadzenia zajęć na uczelniach wyższych inaczej niż w sposób zdalny. Zespół nie miał wiele czasu by zacieśnić więzy, a następnie regularnie widywać się osobiście celem wymiany doświadczeń i przydzielania kolejnych zadań. Zrealizowanych zostało pięć spotkań stacjonarnie w dwutygodniowych odstępach, a kolejne trzy odbyły się bardzo nieregularnie ze względu na przerwę świąteczną oraz ferie. Ostatni zjazd zespołu miał miejsce na początku marca. Dalej prace toczyły się wyłącznie zdalnie. Cała komunikacja odbywała się przez telefon, i przede wszystkim internet, poprzez takie aplikacje jak Slack, MS Teams czy Facebook. Można odnotować, iż tryb ten nie miał znaczącego negatywnego wpływu na rozwój programu. Zespół pracował bez przerw, a opóźnienia lub błędy w systemie były identyfikowane i korygowane.

9.3. Uwagi o technologiach

Największą trudność stanowił brak doświadczenia w zastosowanych technologiach. Z tego powodu wiele czasu spędzono na doszkalaniu się i wzajemnej pomocy w realizacji zadań. Pomimo nauki i uzyskiwania stopniowej biegłości w implementacji napotykano trudności, np. działający lokalnie kod nie sprawiał kłopotów, a po scaleniu i wdrożeniu na serwer przestawał działać lub powodował nieprawidłowości w zaimplementowanych wcześniej fragmentach. Poniżej wypunktowano główne problemy związane z implementacją:

1. Brak wystarczającego doświadczenia przy analizie projektu spowodował, że na etapie implementacji pewne rozwiązania na diagramie klas okazywały się niewłaściwe oraz niepełne, co wymagało ponownego rozpatrzenia i nanoszenia poprawek powodując opóźnienia w pracy deweloperskiej.
2. Znaczącą ilość czasu zajęła instalacja aplikacji na serwer produkcyjny. Serwer był długo niedostępny, pojawił się problem z połączeniem się z bazą danych, a także skonfigurowaniem

maszyny Linux, firewalla oraz samej bazy, przez co zespół testowy nie mógł początkowo pracować na umieszczonym na serwerze programie.

3. Wycofano się ze stosowania protokołu https, gdyż brak certyfikatów ssl podpisanych przez zaufany ośrodek (korzystano jedynie z wystawionych samodzielnie, tzw. *self-signed*) powodował problemy z pracą w przeglądarkach i uniemożliwiał testy automatyczne.
4. Zespół miał trudności z implementacją interfejsu graficznego spowodowane przez kaskadowe arkusze stylów biblioteki React, których opanowanie zajęło dużo czasu. Problem sprawiało właściwe i zgodne z projektem rozmieszczenie elementów na stronie, zagnieżdżanie obiektów oraz dostosowywanie zdjęć i obrazków do rozdzielczości lub zadanej przekątnej. Zdarzało się, że raz opanowane błędy występowały ponownie po wdrożeniu kolejnych, czasem nieznacznych zmian. Jednak nikt z zespołu nie miał zawodowego doświadczenia w tej technologii.
5. Członkowie zespołu postawili sobie ambitny plan: naukę zupełnie dla nich nowej biblioteki React i użycie jej w produkcji programu. Skutkiem tego pojawiły się problemy związane ze specyfiką tej technologii (nadpisywanie CSS, zrozumienie funkcjonowania komponentów klasowych i funkcyjnych, przesyłanie danych do renderowania).
6. Dużo czasu zabrało wdrożenie się członków w technologie JavaScript, które odpowiadały za front-end.
7. Wprowadzono częściową zmianę interfejsu DAO do komunikacji z bazą danych już podczas trwania projektu. Początkowo była to ręczna implementacja metod CRUD z wykorzystaniem API JPA. Jednak później, aby rozwiązać bardziej złożone problemy, jak np. stworzenie API do wyszukiwarki zwierząt i osób lub zaimplementowanie paginacji, wykorzystano interfejs Spring Data JPA. Wymagało to kosztownej czasowo refaktoryzacji.
8. W testowaniu największą trudność sprawiały niejasne opisy wykonanej przez implementację pracy, tj. tester nie miał czasami wystarczającej informacji jaki obszar i w jakim zakresie powinien testować.

Ponadto jeden z pracowników implementacji oddelegowany był na stałe do pracy nad aplikacją mobilną i tylko w pewnym stopniu mógł wspierać prace nad back-endem. Wiele drobniejszych błędów i nieścisłości pojawiało się w trakcie całego procesu; nierzadko usunięcie ich wymagało większych nakładów pracy, np. dokształcenia, zmiany technologii, lub modyfikacji diagramu klas systemu, a w konsekwencji, przebudowy kodu.

9.4. Rozwój aplikacji

Ze względu na ograniczone możliwości nie zdołano zaimplementować wszystkich funkcjonalności i pomysłów związanych z projektem. Niektóre od razu zostały odrzucone, gdyż nie miały stanowić bazowej struktury aplikacji jaką jest zarządzanie zwierzętami w schronisku, a część trafiła na listę oczekujących jako wartościowe lecz nie niezbędne. Nieprzekraczalny termin, wielkość

oraz doświadczenie zespołu czy brak środków finansowych były przyczynami dla których poniższe pomysły nie zostały wdrożone lecz należałoby zaimplementować je w kolejnych wersjach programu:

1. Dostosowanie do innych rynków zbytu

   Chociaż pierwsze diagramy powstawały w języku polskim to na pewnym etapie uzgodniono, iż kolejne projekty oraz kod aplikacji powstawać będą w języku angielskim, jako powszechnie stosowanym w programowaniu. Brano również pod uwagę wydanie interfejsu aplikacji w tym języku, ponieważ jednak sama struktura programu oraz formularze dostosowane były do rynku polskiego zarzucono ten pomysł. Niemniej z racji ograniczenia ilości schronisk działających w Polsce, a co za tym idzie i rynku zbytu, warto byłoby zaproponować omawiany system również schroniskom w innych krajach dostosowując język oraz konieczne formularze czy klasy do danej specyfiki i regulacji prawnych.
2. Aplikacja na urządzenia pracujące z systemem iOS

   W ramach pierwszej wersji zadecydowano o powstaniu wersji mobilnej na najpopularniejszą platformę jaką jest system Android. Naturalną konsekwencją byłoby także utworzenie wersji na urządzenia mobilne pracujące na systemie drugiego gracza na rynku tj. produkty firmy Apple. Bez tego w sposób naturalny ogranicza się ilość użytkowników, którzy mobilnie mogą korzystać z takich funkcji jak dodanie spaceru czy zamieszczenie/przeglądanie ogłoszeń o zaginionych zwierzętach.
3. Certyfikaty ssl

   Brak funduszy spowodował, że zrezygnowano z zaimplementowanego już rozwiązania szyfrowania przy pomocy protokołu https. Okazało się bowiem, że certyfikaty ssl wystawione przez zespół (*self-signed*) traktowane były przez przeglądarki jako potencjalnie niebezpieczne, ponadto niepoprawnie działało wywoływanie API blokując między innymi możliwość przeprowadzania testów. W celu zapewnienia najwyżej jakości należałoby powrócić do tego rozwiązania inwestując środki w certyfikaty poświadczone przez stosowne organizacje odpłatnie.
4. Mapa schronisk

   Jednym z odrzuconych pomysłów była mapa schronisk, tj. wykorzystanie map Google jako wariantu wyboru schroniska z mapy zamiast zwykłej listy. Niemniej API Google udostępnia bezpłatnie jedynie określoną liczbę zapytań, co powodowałoby problemy przy korzystaniu z tego rozwiązania bez poniesienia kolejnych kosztów.
5. Raporty

   Funkcjonalność generowania raportów przygotowano w formie modułu umożliwiającego elastyczne i generyczne tworzenie kolejnych, dowolnych wzorów w przyszłości, bez zmiany kodu aplikacji. Na potrzeby wersji pierwszej powstało kilka podstawowych typów, jednak w

prosty sposób można by opracować kolejne, przydatne dla organizacji, choćby w przypadku rozwoju systemu o zarządzanie kolejnymi elementami w schronisku jak kadry lub magazyn.

6. Dodatkowe moduły aplikacji

   Podstawowym zadaniem stworzonego programu jest zarządzanie zwierzętami w schronisku. Moduł zarządzania użytkownikami dotyczy głównie ich ról, praw dostępu oraz adopcji i zarządzania zadaniami z punktu widzenia zwierząt. Natomiast w celu wzrostu znaczenia i konkurencyjności tego systemu należałoby wdrożyć kolejne przydatne dla użytkownika moduły. W ramach prac analitycznych powstały pomysły implementacji zarządzania stanami magazynowymi i zasobami rzeczowymi schroniska takimi jak: boksy, karma, środki czystości itp. Taki moduł można by rozwinąć w przyszłości o sklep internetowy, gdzie w ramach generowania dodatkowych przychodów dla schroniska użytkownicy mogliby zakupić karmę, boksy czy zabawki dla wybranych zwierząt. Byłby to rozszerzony i uatrakcyjniony moduł adopcji wirtualnej.

   Można także umożliwić zarządzanie pracownikami z księgowego punku widzenia, czyli zaproponować moduł kadrowo-płacowy jako uzupełnienie dla działu kadr oraz finansowego. Wiązałoby się to ze znacznymi nakładami, ale jednocześnie uwalniałoby to schroniska z użytkowania jakiegokolwiek innego oprogramowania.

7. Płatności elektroniczne

   Ułatwienie adopcji wirtualnej poprzez wprowadzenie wpłat on-line mogłoby zwiększyć bieżące przychody schroniska. Użytkownicy Internetu przyzwyczajeni są już do wygody wpłat niewymagających logowania do banku czy wypełniania formularzy z danymi do przelewów. Aby wytworzony program postrzegany był jako nowoczesny należało by umożliwić wpłaty współczesnymi narzędziami jak e-przelewy, płatności kartą czy BLIKiem.

8. Kolejne gatunki zwierząt

   Naturalnym z punktu widzenia rozwoju oprogramowania i niezbyt skomplikowanym we wdrożeniu rozwiązaniem byłoby wprowadzenie obsługi kolejnych gatunków zwierząt. Skupiono się na dwóch, których opieką schroniska zajmują się zdecydowanie najczęściej: kotach i psach. Wprowadzenie kolejnych podklas klasy „Zwierzę” posiadających unikalne cechy czy atrybuty nie wiązałoby się ze znacznymi nakładami, a podkreśliłoby wyjątkowość tego rozwiązania na rynku; ponadto być może w przyszłości, przy już zdecydowanie większym nakładzie pracy, można by zaoferować udoskonalony produkt hodowlom czy małym ogrodom zoologicznym.

9.5. Podsumowanie

Zaplanowane w ramach projektu @zor prace implementacyjne zostały w większości wykonane. Wszystkie zakładane moduły wykonano i wdrożono. Zrealizowano również elementy istotne z punktu

widzenia bezpieczeństwa: rejestrację oraz logowanie użytkowników wraz z przypisaniem im ról i właściwych uprawnień. Dane logowania zostały zaszyfrowane.
Aplikacja realizuje cele postawione na początku jej powstawania:

- umożliwia wypełnianie narzucanych przez prawo zadań schroniska,
- wspiera pracowników w codziennym funkcjonowaniu,
- zapewnia dostęp do informacji osobom z zewnątrz.

Dwa pierwsze cele są realizowane przez moduły dostępne po zalogowaniu. Zalogowani użytkownicy mają udostępnione funkcjonalności zależne od udzielonych uprawnień. Pracownicy oraz administratorzy za pomocą estetycznego menu i przyjaznego kokpitu zarządzają całym schroniskiem: mogą nadzorować spacery, przeprowadzać procesy adopcyjne, mieć podgląd na wszystkie zwierzęta oraz osoby oraz dodawać kolejne dane do bazy lub je edytować. System umożliwia również zarządzanie stroną internetową: artykułami, informacjami, ogłoszeniami, listą darczyńców. Z poziomu kokpitu udostępniony został moduł raportowy, gdzie uzyskać można dowolną informację o sytuacji w schronisku. W każdej chwili bowiem, pracownicy mogą dodać kolejny rodzaj raportu wyłuskujący potrzebne dane z bazy i wyświetlić go na kilka sposobów: liczbowo, tabelarycznie lub wygenerować wykres: linowy, kołowy i słupkowy, w zależności od celu i potrzeb.

Zalogowani weterynarze mogą prowadzić całą historię stanu zdrowia zwierząt on-line. System oferuje wygodne kartoteki medyczne, gdzie pamiętane są wszelkie procedury medyczne, operacje, wyniki badań oraz dawkowanie leków. Natomiast wolontariusze w kilka chwil ustalą harmonogram spacerów.

Realizacja ostatniego celu – dostępu dla osób z zewnątrz, stanowiącego pozornie poboczny element systemu, jest jednym z jego największych atutów. Każdy użytkownik ma nie tylko możliwość biernego przeglądania ogólnodostępnych modułów: listy zwierząt w schronisku, listy darczyńców, ogłoszeń o zaginionych zwierzętach lub aktualności z życia schroniska, ale także w sposób aktywny może pomagać wybranej placówce i jej pracownikom. Aplikacja zachęca do działania i je ułatwia. Osoby spoza organizacji poprzez deklarację wysokości wpłat oraz ich ilości, mogą w kilku prostych krokach wirtualnie adoptować zwierzę. Ponadto mają możliwość wyszukania wymarzonego zwierzęcia za pomocą wygodnego filtra i przeglądania szczegółowych informacji na jego temat. Dzięki aplikacji pierwszy krok adopcji jest wykonywany bez wychodzenia z domu. Uzupełnienie czytelnego formularza jest intuicyjne, a wniosek od razu trafia bezpośrednio do pracowników. Obie strony oszczędzają czas, a aplikujący mogą podejrzeć status procesu na swoim profilu.

W życiu schroniska uczestniczą też właściciele zaginionych pupili. Dodając ogłoszenie o zaginięciu mają możliwość poinformowania innych o poszukiwaniach a zarazem dostarczają informację schronisku, do którego może trafić zgubione zwierzę.

Do wielu z tych funkcjonalności zapewniony jest dostęp z dowolnego miejsca na świecie – niekoniecznie miejsca przy biurku i komputerze. Umożliwia to mobilna wersja aplikacji na system Android.

Zarzadzanie tak złożoną organizacją jak schronisko dla zwierząt, gdzie modyfikacja danych dotyczących podopiecznych następuje wiele razy dziennie, wymaga elastycznego i efektywnego narzędzia, by szybko móc reagować na zmiany. @zor – system ułatwiający organizację pracy w schronisku dla zwierząt jest właśnie takim narzędziem.

10. Bibliografia

[1] Atlassian, „Trello,” [Online]. Available: https://trello.com. [Data uzyskania dostępu: 2 6 2020].

[2] „Ustawa z dnia 21 sierpnia 1997 r. o ochronie zwierząt,” [Online]. Available: http://isap.sejm.gov.pl/isap.nsf/download.xsp/WDU19971110724/U/D19970724Lj.pdf. [Data uzyskania dostępu: 1 12 2019].

[3] Biuro Ochrony Zwierząt, „Raport o problemie bezdomnych zwierzat,” [Online]. Available: http://www.boz.org.pl/raport/2016/1.htm. [Data uzyskania dostępu: 1 12 2019].

[4] Schronisko dla zwierząt "NaPaluchu", „O schronisku,” [Online]. Available: https://www.napaluchu.waw.pl/o_schronisku/. [Data uzyskania dostępu: 1 12 2019].

[5] Schronisko dla zwierząt w Olsztynie, „Zasady przyjmowania i wydawania zwierząt,” [Online]. Available: http://www.schronisko.olsztyn.pl/pl/jak-dzialamy/zasady-przyjmowania-i-wydawania-zwierzat.html. [Data uzyskania dostępu: 5 12 2019].

[6] „Rozporządzeniu Ministra Rolnictwa i Rozwoju Wsi z dnia 23 czerwca 2004 r. w sprawie szczegółowych wymagań weterynaryjnych dla prowadzenia schronisk dla zwierząt.,” [Online]. Available: http://prawo.sejm.gov.pl/isap.nsf/download.xsp/WDU20041581657/O/D20041657.pdf. [Data uzyskania dostępu: 20 2 2020].

[7] „Ustawa z dnia 11 marca 2004 r. o ochronie zdrowia zwierząt oraz zwalczaniu chorób zakaźnych zwierząt,” [Online]. Available: http://prawo.sejm.gov.pl/isap.nsf/download.xsp/WDU20040690625/U/D20040625Lj.pdf. [Data uzyskania dostępu: 12 3 2020].

[8] Schronisko dla zwierząt w Poznaniu, „Wolontariat,” [Online]. Available: https://schronisko.com/jak-pomoc/#wolontariat. [Data uzyskania dostępu: 5 12 2019].

[9] Schronisko dla zwierząt Ciapkowo, „Regulamin adopcji wirtualnej,” [Online]. Available: http://www.ciapkowo.pl/regulamin-adopcji-wirtualnej. [Data uzyskania dostępu: 5 12 2019].

[10] Fundacja Niechciane Zapomniane, „Adopcje wirtualne,” [Online]. Available: http://niechcianeizapomniane.org/adopcje-wirtualne/. [Data uzyskania dostępu: 5 12 2019].

[11] Datum Software, „Program Schronisko,” [Online]. Available: http://www.datum.com.pl/schronisko.html. [Data uzyskania dostępu: 10 11 2019].

[12] Pawlytics, 20 12 2019. [Online]. Available: https://app.pawlytics.com.

[13] Shelterluv, 1 12 2019. [Online]. Available: https://www.shelterluv.com.

[14] RescueConnection Software, „RescueConnection Software,” [Online]. Available: https://www.rescueconnectionsoftware.com. [Data uzyskania dostępu: 2 6 2020].

[15] The Pet Friend, „ Pet Friend,” [Online]. Available: https://www.thepetfriend.com. [Data uzyskania dostępu: 2 6 2020].

[16] Sheltermanager Ltd., „About sheltermanager.com,” [Online]. Available: https://sheltermanager.com. [Data uzyskania dostępu: 2 6 2020].

[17] PetPal Manager, „Pet and Rescue Management Tools,” [Online]. Available: http://petpalmanager.com/. [Data uzyskania dostępu: 2 6 2020].

[18] BARRK, „Get your animal rescue organized with BARRK,” [Online]. Available: https://www.barrk.net/. [Data uzyskania dostępu: 2 6 2020].

[19] Shelter Pro Software, [Online]. Available: https://www.shelterpro.com/. [Data uzyskania dostępu: 2 6 2020].

[20] FreeLogoDesign, „FreeLogoDesign,” [Online]. Available: https://www.freelogodesign.org/. [Data uzyskania dostępu: 6 3 2020].

[21] Oracle, „Dowiedz się więcej o technologii Java,” [Online]. Available: https://www.java.com/pl/about/. [Data uzyskania dostępu: 4 01 2020].

[22] T. Woliński, „Java Historia,” [Online]. Available: https://stormit.pl/historia-java/. [Data uzyskania dostępu: 4 01 2020].

[23] B. Joy, G. Steel, G. Bracha, A. Buckley, D. Smith i J. Gosling, „The Java® Language Specification. Java SE 11 Edition,” [Online]. Available: https://docs.oracle.com/javase/specs/jls/se11/jls11.pdf. [Data uzyskania dostępu: 4 01 2020].

[24] Oracle, „Jak uzyskać oprogramowanie Java dla urządzeń mobilnych?,” [Online]. Available: https://www.java.com/pl/download/faq/java_mobile.xml. [Data uzyskania dostępu: 4 01 2020].

[25] Pivotal Software, „Spring Framework Documentation.,” [Online]. Available: https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#spring-core. [Data uzyskania dostępu: 29 12 2019].

[26] Javastart, „Spring,” [Online]. Available: https://javastart.pl/baza-wiedzy/frameworki/spring. [Data uzyskania dostępu: 29 12 2019].

[27] Pivotal Software, „Spring Security.,” [Online]. Available: https://spring.io/projects/spring-security. [Data uzyskania dostępu: 29 12 2019].

[28] VMware Inc., „Spring Initializr,” [Online]. Available: https://start.spring.io/. [Data uzyskania dostępu: 13 3 2020].

[29] Pivotal, „Spring,” [Online]. Available: https://spring.io/. [Data uzyskania dostępu: 29 12 2019].

[30] G. King i C. Bauer, „Hibernate in action. Maning,,” 2015. [Online]. Available: https://archive.org/details/hibernateinactio00baue/page/n15. [Data uzyskania dostępu: 22 12 2019].

[31] „Hibernate ORM. What is an Object/Relational Mapping,” [Online]. Available: https://hibernate.org/orm/what-is-an-orm/. [Data uzyskania dostępu: 22 12 2019].

[32] G. King i C. Bauer, „Java Persistence with Hibernate, Second Edition.,” Maning, 2015. [Online]. Available: https://www.manning.com/books/java-persistence-with-hibernate-second-edition. [Data uzyskania dostępu: 22 12 2019].

[33] G. King i C. Bauer, „Hibernate w akcji.," Helion, 2007. [Online]. Available: ftp://ftp.helion.pl/online/hibakc/hibakc-2.pdf. [Data uzyskania dostępu: 22 12 2019].

[34] JBoss, „Hibernate Documentation," [Online]. Available: https://docs.jboss.org/hibernate/stable/annotations/reference/en/html_single/#entity-mapping. [Data uzyskania dostępu: 22 12 2019].

[35] The Apache Maven Foundation, „Welcome to Apache Maven," [Online]. Available: https://maven.apache.org/index.html. [Data uzyskania dostępu: 13 3 2020].

[36] J. v. Zyl, „History of Maven," [Online]. Available: https://maven.apache.org/background/history-of-maven.html. [Data uzyskania dostępu: 2 1 2020].

[37] T. A. M. Foundation, „Introduction to the POM," [Online]. Available: http://maven.apache.org/guides/introduction/introduction-to-the-pom.html. [Data uzyskania dostępu: 13 3 2020].

[38] The Apache Software Foundation, „Introduction to the Build Lifecycle.," [Online]. Available: https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html. [Data uzyskania dostępu: 2 1 2020].

[39] Oracle, „MySQL 5.0 Reference Manula.," [Online]. Available: https://docs.oracle.com/cd/E17952_01/mysql-5.0-en/what-is-mysql.html. [Data uzyskania dostępu: 2 1 2020].

[40] Oracle, „ Why MySql?," [Online]. Available: https://www.mysql.com/why-mysql/. [Data uzyskania dostępu: 2 1 2020].

[41] Oracle, „Database SQL Language References.," [Online]. Available: https://docs.oracle.com/cd/E11882_01/server.112/e41084/intro001.htm#SQLRF50932. [Data uzyskania dostępu: 4 1 2020].

[42] J. Kasprzak, „Logiczne przetwarzanie zapytań SQL," [Online]. Available: https://www.sqlpedia.pl/logiczne-przetwarzanie-zapytan-sql. [Data uzyskania dostępu: 13 3 2020].

[43] Oracle, „MySQL Workbench," [Online]. Available: https://www.mysql.com/products/workbench/. [Data uzyskania dostępu: 13 3 2020].

[44] Mozilla, „HTML 5," [Online]. Available: https://developer.mozilla.org/pl/docs/HTML/HTML5. [Data uzyskania dostępu: 13 3 2020].

[45] „HTML 4.0 Specification.," [Online]. Available: https://www.w3.org/TR/REC-html40-971218/conform.html. [Data uzyskania dostępu: 5 1 2020].

[46] H. Delgado, „HTML History - Origin and evolution of the Web hypertext.," [Online]. Available: https://disenowebakus.net/en/html-history/. [Data uzyskania dostępu: 5 1 2020].

[47] J. Duckett, HTML i CSS. Zaprojektuj i zbuduj witrynę WWW. Podręcznik Front-End Developera, Gliwice: Helion, 2018.

[48] Wikipedia, „HTML,” [Online]. Available: https://en.wikipedia.org/wiki/HTML. [Data uzyskania dostępu: 13 3 2020].

[49] WHATWG, „HTML,” [Online]. Available: https://html.spec.whatwg.org/multipage/introduction.html#is-this-html5?. [Data uzyskania dostępu: 11 3 2020].

[50] H. W. Lie i B. Bos, Cascading Style Sheets, designing for the Web, Addison Wesley, 1999.

[51] W3C, „ Selectors Level 4,” [Online]. Available: https://www.w3.org/TR/REC-html40-971218/conform.html. [Data uzyskania dostępu: 5 1 2020].

[52] Wikipedia, „Cascading Style Sheets,” [Online]. Available: https://en.wikipedia.org/wiki/Cascading_Style_Sheets. [Data uzyskania dostępu: 13 3 2020].

[53] MDN web docs, „JavaScript,” [Online]. Available: https://developer.mozilla.org/pl/docs/Web/JavaScript. [Data uzyskania dostępu: 5 1 2020].

[54] Internetingishard, „Introduction,” [Online]. Available: https://internetingishard.com/html-and-css/introduction/. [Data uzyskania dostępu: 11 1 2020].

[55] D. Flanagan, JavaScript: The Definitive Guide. Activate Your Web Pages. 6th Edition (ebook), O’Reilly, 2011.

[56] Wikipedia, „JavaScript,” [Online]. Available: . Źródło: https://en.wikipedia.org/wiki/JavaScript. [Data uzyskania dostępu: 13 3 2020].

[57] D. Goodman i B. Eich, „JavaScript bible, fourth edition.,” [Online]. Available: https://archive.org/details/javascriptbible00dann. [Data uzyskania dostępu: 10 1 2020].

[58] A. Goel, „10 Best JavaScript Frameworks to Use in 2020,” [Online]. Available: https://hackr.io/blog/best-javascript-frameworks. [Data uzyskania dostępu: 10 1 2020].

[59] Bootstrap, „Bootstrap Documentation: Introduction,” [Online]. Available: https://getbootstrap.com/docs/4.4/getting-started/introduction/. [Data uzyskania dostępu: 11 1 2020].

[60] Wikipedia, „Bootstrap (front-end framework),” [Online]. Available: https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework). [Data uzyskania dostępu: 12 1 2020].

[61] Bootstrap, „Bootstrap Documentation: JavaScript,” [Online]. Available: https://getbootstrap.com/docs/3.4/javascript/. [Data uzyskania dostępu: 11 1 2020].

[62] Amelia, „What Is Bootstrap,” [Online]. Available: https://wpamelia.com/what-is-bootstrap/. [Data uzyskania dostępu: 13 3 2020].

[63] Bootstrap, „Bootstrap,” [Online]. Available: https://getbootstrap.com/. [Data uzyskania dostępu: 13 3 2020].

[64] D. Szczepaniak, „Wstęp do REST API,” [Online]. Available: https://devszczepaniak.pl/wstep-do-rest-api/. [Data uzyskania dostępu: 11 1 2020].

[65] G. Kwaśniewski, „ Rest-api od podstaw.,” [Online]. Available: https://www.devmobile.pl/rest-api-od-podstaw/ . [Data uzyskania dostępu: 10 1 2020].

[66] G. James, „Creating a simple REST API in PHP,” [Online]. Available: https://shareurcodes.com/blog/creating%20a%20simple%20rest%20api%20in%20php. [Data uzyskania dostępu: 13 3 2020].

[67] JetBrains, „Oficjalna strona IntelliJ IDEA.,” [Online]. Available: https://www.jetbrains.com/idea/. [Data uzyskania dostępu: 29 12 2019].

[68] Postman, „Abot Postman,” [Online]. Available: https://www.postman.com/about-postman. [Data uzyskania dostępu: 10 1 2020].

[69] D. Plawgo, „Postman - testowanie API,” [Online]. Available: https://plawgo.pl/2019/01/22/postman-testowanie-api. [Data uzyskania dostępu: 10 1 2020].

[70] E. Burnette, „Hello, Android : introducing Google's mobile development platform,” [Online]. Available: https://archive.org/details/isbn_9781934356562. [Data uzyskania dostępu: 12 1 2020].

[71] Techopedia, „Android SDK,” [Online]. Available: https://www.techopedia.com/definition/4220/android-sdk. [Data uzyskania dostępu: 10 1 2020].

[72] Wikipedia, „Android Software Development,” [Online]. Available: https://en.wikipedia.org/wiki/Android_software_development. [Data uzyskania dostępu: 12 1 2020].

[73] B. Shannon, „How to Use an Android Layout Mockup Utility that Generates Usable XML and Java,” [Online]. Available: https://www.codeproject.com/Articles/782206/How-to-Use-an-Android-Layout-Mockup-Utility-that-G. [Data uzyskania dostępu: 13 3 2020].

[74] Facebook Inc., „React,” [Online]. Available: https://reactjs.org/. [Data uzyskania dostępu: 28 5 2020].

[75] R. Eckstein, „Java SE Application Design With MVC,” Oracle, 3 2007. [Online]. Available: https://www.oracle.com/technical-resources/articles/javase/application-design-with-mvc.html. [Data uzyskania dostępu: 28 5 2020].

[76] D. Abramov, „React Redux,” [Online]. Available: https://react-redux.js.org/. [Data uzyskania dostępu: 28 5 2020].

[77] „Fetch API,” 1 02 2020. [Online]. Available: https://kursjs.pl/kurs/ajax/fetch.php. [Data uzyskania dostępu: 28 5 2020].

[78] Wikipedia, „Dependency Injection,” [Online]. Available: https://en.wikipedia.org/wiki/Dependency_injection. [Data uzyskania dostępu: 28 5 2020].

[79] Wikipedia, „Data Acces Object,” [Online]. Available: https://pl.wikipedia.org/wiki/Data_Access_Object. [Data uzyskania dostępu: 28 5 2929].

[80] Wikipedia, „Data Transfer Object,” [Online]. Available: https://en.wikipedia.org/wiki/Data_transfer_object. [Data uzyskania dostępu: 28 5 2020].

[81] Wikipedia, „Fasada (wzorzec projektowy),” [Online]. Available: https://pl.wikipedia.org/wiki/Fasada_(wzorzec_projektowy). [Data uzyskania dostępu: 28 5 2020].

[82] J. Reschke, „The ’Basic’ HTTP Authentication Scheme,” Internet Engineering Task Force, 9 2015. [Online]. Available: https://www.rfc-editor.org/pdfrfc/rfc7617.txt.pdf. [Data uzyskania dostępu: 28 5 2020].

[83] Wikipedia, „Https,” [Online]. Available: https://pl.wikipedia.org/wiki/HTTPS. [Data uzyskania dostępu: 28 5 2020].

[84] National Institute of Standarsd and Technology, „Role Based Access Control,” [Online]. Available: https://csrc.nist.gov/projects/role-based-access-control. [Data uzyskania dostępu: 28 05 2020].

[85] Quality Open Software, „Simple Logging Facade for Java (SLF4J),” [Online]. Available: http://www.slf4j.org/. [Data uzyskania dostępu: 29 04 2020].

[86] Stowarzyszenie Jakości Systemów Informatycznych, „Certyfikowany tester. Sylabus poziomu podstawowego ISTQB. Wersja 2018 V 3.1.,” 2020.

[87] Apache Software Foundation, „Apache Groovy,” [Online]. Available: https://groovy-lang.org/. [Data uzyskania dostępu: 13 6 2020].

[88] Rest ASsured, [Online]. Available: http://rest-assured.io/. [Data uzyskania dostępu: 13 6 2020].

[89] Spock, „Spock the enterprise ready specification framework,” [Online]. Available: http://spockframework.org/. [Data uzyskania dostępu: 13 6 2020].

[90] Git, „Git,” [Online]. Available: https://git-scm.com/. [Data uzyskania dostępu: 25 5 2020].

[91] G. James, „Creating a simple REST API in PHP,” [Online]. Available: https://shareurcodes.com/blog/creating%20a%20simple%20rest%20api%20in%20php. [Data uzyskania dostępu: 13 3 2020].

11. Spis tabel

12. Spis listingów

Dodatek A

Organizacja pracy – podział ze względu na osoby

Poniżej znajduje się zakres prac każdego z uczestników projektu. Uczestnicy są wymienieni w kolejności alfabetycznej.

Małgorzata Bieniek

1. Analiza stanu sztuki;
2. Funkcjonalności systemu;
3. Przypadki użycia;
4. Diagramy aktywności;
5. Scenariusze;
6. Diagramy stanu;
7. Projektowanie raportów;
8. Logo aplikacji;
9. Dokumentacja:
    - Wstęp;
    - Rozdział 2;
    - Rozdział 3: podrozdziały 3.1 i 3.4;
    - Rozdział 4: podrozdziały 4.1 – 4.2, 4.4 – 4.6;
    - Rozdział 5: podrozdziały 5.1 – 5.5, 5.11;
    - Rozdział 6.

Adrian Borkowski

1. Analiza stanu sztuki;
2. Funkcjonalności systemu;
3. Analityczny diagram klas;
4. Diagramy stanu;
5. Projektowanie ekranów;
6. Projektowanie kalendarza;
7. Implementacja klas z diagramu projektowego;
8. Mapowanie Hibernate;
9. CRUD API;
10. Integracja Front-end – Back-End;
11. Implementacja modułów systemu.

Paweł Celejewski

1. Szkolenie SpringBoot i Hibernate;
2. Przygotowanie do konfiguracji stacji roboczych i stack technologiczny;
3. Konfiguracja serwera uczelnianego oraz wdrożenie aplikacji;
4. Implementacja klas z diagramu projektowego;
5. mapowanie Hibernate;
6. CRUD API
7. Implementacja generycznej obsługi błędów zapytań REST;
8. Implementacja skryptu wypełniającego bazę danych;
9. Tworzenie widoków w REACT;
10. Integracja Front-end – Back-End;
11. Szyfrowanie danych logowania;
12. Rejestracja użytkowników;
13. Implementacja modułów systemu.

Aleksandra Grabowska

1. Analiza stanu sztuki;
2. Funkcjonalności systemu;
3. Przypadki użycia;
4. Diagramy aktywności;
5. Scenariusze;
6. Tworzenie widoków w REACT;
7. Integracja Front-end - Back-End;
8. Wypełnianie bazy danych;
9. Poprawa błędów UX;
10. Testy automatyczne;
11. Testy API;
12. Funkcjonalne testy manualne;
13. Implementacja modułów systemu.

Michał Jabłoński

1. Projektowy diagram klas;
2. Tworzenie widoków w REACT;
3. Integracja Front-end - Back-End;
4. Szkolenie Unit testy;
5. Testy automatyczne;
6. Funkcjonalne testy manualne;

7. Implementacja modułów systemu.

Rafał Pajączkowski

1. Szkolenie JavaScript Fetch API, React, Angular;
2. Tworzenie widoków w REACT;
3. Integracja Front-end - Back-End;
4. Implementacja modułów systemu.

Renata Pigoń

1. Funkcjonalności systemu;
2. Przypadki użycia;
3. Scenariusze;
4. Projektowanie ekranów;
5. Walidacja wymagań i funkcjonalności;
6. Analiza UX;
7. Poprawa błędów UX.

Marcin Szarek

1. Koordynacja prac całego zespołu;
2. Harmonogramy;
3. Konspekt;
4. Funkcjonalności systemu;
5. Przypadki użycia;
6. Diagramy aktywności;
7. Scenariusze;
8. Dokumentacja:
    - Rozdział 3: podrozdziały 3.2 i 3.3;
    - Rozdział 4: podrozdziały 4.3, 4.10 – 4.11;
    - Rozdział 5: podrozdziały 5.6 – 5.10, 5.12 – 5.13;
    - Rozdział 7;
    - Rozdział 8;
    - Podsumowanie.

Jan Szczawiński

1. Funkcjonalności systemu;
2. Przypadki użycia;
3. Diagramy aktywności;

4. Tworzenie widoków w REACT;
5. Integracja Front-end - Back-End;
6. Generowanie i wykresy raportów;
7. Wysyłka mailingów;
8. Implementacja modułów systemu.

Justyna Tomaszewska

1. Szkolenie REST i Git;
2. Konfiguracja konta i tablicy Trello;
3. Implementacja klas z diagramu projektowego;
4. Mapowanie Hibernate;
5. CRUD API;
6. Tworzenie widoków w REACT;
7. Integracja Front-end – Back-End;
8. Szyfrowanie danych logowania;
9. Poprawa błędów UX;
10. Utworzenie wspólnego konta Postman;
11. Testy automatyczne;
12. Testy API;
13. Funkcjonalne testy manualne;
14. Implementacja modułów systemu.

Łukasz Zajkowski

1. Projektowanie aplikacji mobilnej;
2. Szkolenie Android Studio;
3. Implementacja klas z diagramu projektowego;
4. Mapowanie Hibernate;
5. CRUD API;
6. Budowa aplikacji mobilnej;
7. API dla aplikacji mobilnej;
8. Ekrany aplikacji mobilnej.

Dodatek B

Przykłady scenariuszy testowych

Poniżej w tabelach 17 – 25 zaprezentowane zostały przykładowe scenariusze testowe przygotowane dla zespołu testowego.

Tabela 17. Scenariusz testowy: Wysłanie zgłoszenia adopcyjnego. Źródło: opracowanie własne.

Nazwa scenariusza	Wysłanie zgłoszenia adopcyjnego	
Warunki początkowe	Użytkownik jest zalogowany	
Realizacja scenariusza testowego		
Krok	**Akcja użytkownika**	**Odpowiedź systemu**
1	Użytkownik wchodzi w zakładkę „Adoptuj” > ”Zabierz do domu”.	System wyświetla listę zwierząt przeznaczonych do adopcji wraz z filtrem.
2	Użytkownik wybiera zwierzę z listy i klika na przycisk „Więcej”.	System wyświetla kartę zwierzęcia.
3	Użytkownik klika w przycisk „Adoptuj”.	System wyświetla okno „Rozpoczęcie procesu adopcji”.
4	Użytkownik potwierdza swoje dane oraz zwierzę.	System wyświetla kwestionariusz do wypełnienia.
5	Użytkownik uzupełnia wszystkie pola kwestionariusza i klika „Wyślij”.	System potwierdza wysłanie formularza.
Oczekiwany rezultat	System potwierdza wysłanie formularza i zapisuje adopcję w bazie.	
Uzyskany rezultat	System potwierdza wysłanie formularza i zapisuje adopcję w bazie.	
Wynik testu	Pozytywny	

Tabela 18. Scenariusz testowy: Dodanie decyzji przedadopcyjnej. Źródło: opracowanie własne.

Nazwa scenariusza	Dodanie decyzji przedadopcyjnej	
Warunki początkowe	Pracownik jest zalogowany	
Realizacja scenariusza testowego		
Krok	**Akcja użytkownika**	**Odpowiedź systemu**
1	Użytkownik wchodzi w zakładkę „Adopcja”.	System wyświetla listę adopcji wraz z filtrem.

2	Użytkownik wybiera adopcję z listy i klika na przycisk numeru wybranej adopcji.	System wyświetla kartę adopcji z informacjami dotyczącymi adopcji oraz jej bieżącym statusem.
3	Użytkownik klika w zakładkę „Decyzje przedadopcyjne”.	System wyświetla zakładkę „Decyzje przedadopcyjne” z listą dotychczasowych decyzji.
4	Użytkownik klika w przycisk „Dodaj”.	System wyświetla kwestionariusz do wypełnienia.
5	Użytkownik uzupełnia wszystkie pola kwestionariusza i klika „Dodaj”.	System wyświetla listę decyzji wraz z nowo dodaną decyzją.
Oczekiwany rezultat	System wyświetla listę decyzji wraz z nowo dodaną decyzją i zapisuje decyzję w bazie.	
Uzyskany rezultat	System wyświetla listę decyzji wraz z nowo dodaną decyzją i zapisuje decyzję w bazie.	
Wynik testu	Pozytywny	

Tabela 19. Scenariusz testowy: Zgłoszenie do adopcji wirtualnej. Źródło: opracowanie własne.

Nazwa scenariusza	Zgłoszenie do adopcji wirtualnej	
Warunki początkowe	Użytkownik jest zalogowany	
Realizacja scenariusza testowego		
Krok	**Akcja użytkownika**	**Odpowiedź systemu**
1	Użytkownik wchodzi w zakładkę „Adoptuj” > ”Wirtualnie”.	System wyświetla listę zwierząt wraz z filtrem oraz nagłówek z informacją czym jest adopcja wirtualna.
2	Użytkownik wybiera zwierzę z listy i klika przycisk „Adoptuj wirtualnie”.	System wyświetla kartę zwierzęcia wraz z dodatkowymi polami „Deklarowana kwota” oraz „Okres”.
3	Użytkownik uzupełnia pola i klika przycisk „Adoptuj wirtualnie”.	System wyświetla informacje o przyjętej adopcji wirtualnej podając dane do przelewu i prosi o potwierdzenie wysłania formularza.
4	Użytkownik klika w przycisk „Potwierdź”.	System wyświetla komunikat o przyjęciu adopcji wirtualnej i zapisuje adopcję w bazie.
Oczekiwany rezultat	System wyświetla komunikat o przyjęciu adopcji wirtualnej i zapisuje adopcję w bazie.	

Uzyskany rezultat	System wyświetla komunikat o przyjęciu adopcji wirtualnej i zapisuje adopcję w bazie.
Wynik testu	Pozytywny

Tabela 20. Scenariusz testowy: Rezygnacja z adopcji wirtualnej. Źródło: opracowanie własne.

Nazwa scenariusza	Rezygnacja z adopcji wirtualnej	
Warunki początkowe	Użytkownik jest zalogowany	
Realizacja scenariusza testowego		
Krok	**Akcja użytkownika**	**Odpowiedź systemu**
1	Użytkownik wchodzi w zakładkę „Moje konto” > ”Moje adopcje wirtualne”.	System wyświetla listę zwierząt wirtualnie zaadoptowanych.
2	Użytkownik wybiera zwierzę z listy i klika przycisk „Szczegóły”.	System wyświetla kartę zwierzęcia wraz z przyciskiem „Anuluj e-adopcję”.
3	Użytkownik klika przycisk „Anuluj e-adopcję”.	System pyta o potwierdzenie rezygnacji.
4	Użytkownik klika w przycisk „OK”.	System wyświetla listę zwierząt przeznaczonych do adopcji wirtualnej i usuwa adopcję z listy.
Oczekiwany rezultat	System wyświetla listę zwierząt przeznaczonych do adopcji wirtualnej i usuwa adopcję z listy.	
Uzyskany rezultat	System wyświetla listę zwierząt przeznaczonych do adopcji wirtualnej i usuwa adopcję z listy.	
Wynik testu	Pozytywny	

Tabela 21. Scenariusz testowy: Dodanie aktywności medycznej do kalendarza. Źródło: opracowanie własne.

Nazwa scenariusza	Dodanie aktywności medycznej do kalendarza	
Warunki początkowe	Pracownik jest zalogowany	
Realizacja scenariusza testowego		
Krok	**Akcja użytkownika**	**Odpowiedź systemu**
1	Użytkownik wchodzi w zakładkę „Wydarzenia”	System wyświetla listę wydarzeń wraz z ich typami.
2	Użytkownik naciska przycisk „+”	System wyświetla kartę wydarzenia.
3	Użytkownik uzupełnia pola i klika przycisk „Ikonka dyskietki”.	System wyświetla informacje o dodanej aktywności wraz z dodatkowymi polami do uzupełnienia

		dotyczącymi zwierzęcia, kolejnej wizyty oraz z zakładką „Leki”.
4	Użytkownik klika w zakładkę „Leki”.	System wyświetla listę podanych leków.
5	Użytkownik klika w ikonkę „+”.	System wyświetla formularz podania leków.
6	Użytkownik uzupełnia pola i klika „Dodaj”.	System powraca do listy leków wraz z wyświetloną nową pozycją.
7	Użytkownik wraca do zakładki szczegóły i klika przycisk „Ikonka dyskietki”.	System wyświetla listę wydarzeń wraz z nową pozycją oraz dodaje ją do kalendarza.

Oczekiwany rezultat	System wyświetla listę wydarzeń wraz z nową pozycją oraz dodaje ją do kalendarza
Uzyskany rezultat	System wyświetla listę wydarzeń wraz z nową pozycją lecz nie dodaje jej do kalendarza. W trakcie dodawania leku oraz aktywności medycznej system wyświetla komunikat w formacie JSON.
Wynik testu	Negatywny

Tabela 22. Scenariusz testowy: Edycja wydarzenia w kalendarzu. Źródło: opracowanie własne.

Nazwa scenariusza	Edycja wydarzenia w kalendarzu	
Warunki początkowe	Pracownik jest zalogowany	
Realizacja scenariusza testowego		
Krok	**Akcja użytkownika**	**Odpowiedź systemu**
1	Użytkownik wchodzi w zakładkę „Kalendarz”.	System wyświetla kalendarz wraz z wydarzeniami różnego typu.
2	Użytkownik naciska wybrane wydarzenie.	System wyświetla kartę wydarzenia.
3	Użytkownik edytuje pola i klika przycisk „Zapisz”.	System wyświetla kalendarz wraz z edytowanym wydarzeniem.
Oczekiwany rezultat	System wyświetla kalendarz wraz z edytowanym wydarzeniem.	
Uzyskany rezultat	System wyświetla błąd Unhandled Rejection: err.json is not a function.	
Wynik testu	Negatywny	

Tabela 23. Scenariusz testowy: Dodanie nowego typu raportu. Źródło: opracowanie własne.

Nazwa scenariusza	Dodanie nowego typu raportu
Warunki początkowe	Pracownik jest zalogowany

Realizacja scenariusza testowego		
Krok	**Akcja użytkownika**	**Odpowiedź systemu**
1	Użytkownik wchodzi w zakładkę „Admin”.	System wyświetla kokpit administratora.
2	Użytkownik naciska zakładkę „Raporty”.	System wyświetla listę raportów.
3	Użytkownik klika w przycisk „Dodaj raport”.	System wyświetla formularz do wypełnienia.
4	Użytkownik wypełnia pola i wciska przycisk ”Waliduj”.	System wyświetla informację o poprawnym kodzie SQL.
5	Użytkownik klika w ikonkę „Dodaj”.	System wyświetla listę raportów z nowo dodanym raportem.
Oczekiwany rezultat	System wyświetla listę raportów z nowo dodanym raportem.	
Uzyskany rezultat	System wyświetla listę raportów z nowo dodanym raportem.	
Wynik testu	Pozytywny	

Tabela 24. Scenariusz testowy: Wygenerowanie raportu. Źródło: opracowanie własne.

Nazwa scenariusza	Wygenerowanie raportu	
Warunki początkowe	Pracownik jest zalogowany	
Realizacja scenariusza testowego		
Krok	**Akcja użytkownika**	**Odpowiedź systemu**
1	Użytkownik wchodzi w zakładkę „Admin”.	System wyświetla kokpit administratora.
2	Użytkownik naciska zakładkę „Raporty”.	System wyświetla listę raportów.
3	Użytkownik wybiera raport z listy klika w przycisk „Wygeneruj raport” oraz wybiera z listy opcję typu raportu.	System wyświetla raport w żądanej formie o ile to możliwe lub informuje że wybrana forma była niemożliwa i wyświetla w tabelarycznej.
Oczekiwany rezultat	System wyświetla raport w żądanej formie o ile to możliwe lub informuje że wybrana forma była niemożliwa i wyświetla w tabelarycznej.	
Uzyskany rezultat	System wyświetla raport w żądanej formie o ile to możliwe lub informuje że wybrana forma była niemożliwa i wyświetla w tabelarycznej.	
Wynik testu	Pozytywny	

Tabela 25. Scenariusz testowy: Dodanie zwierzęcia do bazy. Źródło: opracowanie własne.

Nazwa scenariusza	Dodanie zwierzęcia do bazy.	
Warunki początkowe	Pracownik jest zalogowany	
Realizacja scenariusza testowego		
Krok	**Akcja użytkownika**	**Odpowiedź systemu**
1	Użytkownik wchodzi w zakładkę „Zwierzęta”.	System wyświetla listę zwierząt.
2	Użytkownik naciska przycisk „+”	System wyświetla kartotekę zwierzęcia do uzupełnienia.
3	Użytkownik uzupełnia dane wraz ze statusem i zdjęciem a następnie klika „Dodaj”.	System wyświetla listę zwierząt wraz z nowym zwierzęciem.
Oczekiwany rezultat	System wyświetla listę zwierząt wraz z nowym zwierzęciem.	
Uzyskany rezultat	System wyświetla listę zwierząt wraz z nowym zwierzęciem.	
Wynik testu	Pozytywny	