Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Sztuczna Inteligencja

Vetty – aplikacja umożliwiająca dzielenie się historią medyczną zwierząt domowych

Wiktor Kulesza
Numer albumu 304050

promotor
dr hab. inż. Mariusz Kamola

WARSZAWA 2023

Vetty – aplikacja umożliwiająca dzielenie się historią medyczną zwierząt domowych

Streszczenie. Praca polegała na opracowaniu oraz implementacji aplikacji, która umożliwiałaby przechowywanie danych medycznych zwierząt domowych. Dane medyczne zawierają historie przebytych chorób wraz z badaniami krwi. Zaimplementowana aplikacja internetowa Vetty oprócz przechowywania historii medycznej posiada również wbudowane forum stworzone w celu dzielenia się wiedzą oraz poszukiwania pomocy przez użytkowników. Celem systemu jest ułatwienie korzystania z usług wielu weterynarzy jednocześnie oraz codziennej opieki nad zwierzętami, a także umożliwienie szukania pomocy na forum. Aplikacja powstała z wykorzystaniem architektury klient-serwer. W jej skład wchodzi aplikacja kliencka, aplikacja serwerowa oraz baza danych. Do implementacji aplikacji klienckiej została wykorzystana biblioteka React.js dla języka JavaScript. Aplikacja serwerowa została stworzona przy użyciu platformy Spring Boot dla języka Java, a baza danych korzysta z relacyjnej bazy MySQL. Vetty spełnia postawione wymagania funkcjonalne i niefunkcjonalne, a ponadto ułatwia codzienne funkcjonowanie właścicieli zwierząt, którzy korzystając z aplikacji mogą w łatwy sposób przechowywać historie chorób swoich zwierząt, przeglądać ich statystyki na tle rasy oraz komunikować się z innymi użytkownikami poprzez forum.

Słowa kluczowe: ReactJS, Spring Boot, MySQL, aplikacja internetowa, dane medyczne zwierząt

Vetty - an app that allows you to share your pets' medical history

Abstract. The subject of the thesis is the development and implementation of an application that would enable the storage of medical data of pets. Medical data includes a history of past illnesses along with blood tests. Besides storing medical records, the implemented web application also has a built-in forum created for users to share experiences and seek help. The goal of the system is to facilitate consulting various veterinarians in the course of an illness and the day-to-day care of animals, as well as to allow people to seek help in a forum. The application was developed using client-server architecture. Its components include a client application, a server application and a database. The React.js library for JavaScript was used to implement the client application. The server application was developed using the Spring Boot platform for Java, and the database uses the MySQL management system. Vetty meets the set functional and non-functional requirements and facilitates the day-to-day functioning of pet owners, who, by using the application, can easily store their pets' medical histories, view their breed statistics and communicate with other users through a forum.

Keywords: ReactJS, Spring Boot, MySQL, web application, animal medical data

Spis treści

1. Wstęp

Według badania przeprowadzonego w 2020 roku, 62% mieszkańców wsi posiadało psa, podczas gdy w miastach co trzeci mieszkaniec posiadał czworonoga. Odsetek mieszkańców Polski posiadających kota wyniósł 37% [1]. Tak duża liczba skutkuje ogromną ilością danych na temat zwierząt domowych i historii ich chorób. Dane te najczęściej przechowywane są w książeczkach zdrowia zwierzęcia lub elektronicznych książkach klinicznych poszczególnych lecznic weterynaryjnych. Mimo zapisu elektronicznego informacji klient w przypadku zmiany weterynarza może najczęściej poprosić jedynie o papierowy wydruk historii chorób zwierzęcia, co utrudnia proces zmiany weterynarza oraz blokuje możliwość częstego podróżowania. Obecnie istnieje niewiele rozwiązań pozwalającym właścicielom zwierząt na łatwą transformację cyfrową z danych papierowych na elektroniczne, czy wymianę tych informacji z innymi lekarzami bądź właścicielami. Możliwość przechowywania danych medycznych zwierząt w jednym miejscu oraz dzielenia się informacją z innymi posiadaczami zwierząt ułatwiłaby znacząco funkcjonowanie społeczności jaką w tym przypadku jest grupa właścicieli zwierząt domowych. Dzięki takiemu rozwiązaniu posiadacze zwierząt nie byliby uzależnieni od jednego weterynarza, co w czasach charakteryzujących się dynamiką życia oraz częstymi zmianami zachęcałoby ludzi do podejmowania decyzji o zakupie, bądź adopcji zwierząt. Zbierane w jednym systemie dane pozwoliłyby na transformację cyfrową danych medycznych zwierząt i umożliwiłyby rozwój algorytmów sztucznej inteligencji w zakresie diagnostyki chorób.

1.1. Cel pracy

Celem pracy było stworzenie aplikacji internetowej, która umożliwi właścicielom zwierząt transformację cyfrową poprzez przechowywanie informacji oraz danych medycznych swoich zwierząt wraz z badaniami krwi w jednym systemie. Główną motywacją do implementacji rozwiązania była chęć ułatwienia procesu zmiany weterynarza. Dzięki wytworzonej aplikacji właściciele nie musieliby obawiać się braku potrzebnych informacji o przebytych chorobach zwierzęcia podczas wizyt w nowej klinice weterynaryjnej.

Ponadto aplikacja udostępnia szereg funkcjonalności ułatwiających codzienne funkcjonowanie właścicieli zwierząt, które dotyczą:

- dodawania zwierząt;
- dodawania historii medycznych z wynikami badań;
- dzielenia się problemami oraz informacjami poprzez publiczne forum;
- możliwości wyszukiwania zwierząt innych użytkowników;
- przeglądania statystyk dotyczących swoich zwierząt.

1.2. Struktura pracy

Rozdział drugi skupia się na założeniach oraz wymaganiach funkcjonalnych aplikacji webowej. W rozdziale trzecim przybliżona została architektura aplikacji oraz zarys projektu. W kolejnych rozdziałach przedstawiono szczegóły

implementacyjne elementów systemu z opisem wykorzystanych technologii oraz zostało zaprezentowane działanie aplikacji wraz z instrukcją obsługi.

2. Specyfikacja wymagań

W tym rozdziale przedstawione zostaną wymagania stawiane aplikacji Vetty służącej do zarządzania danymi medycznymi swoich Przeanalizowane zostaną wymagania funkcjonalne oraz niefunkcjonalne systemu, jak i jego niezbędne założenia.

2.1. Analiza wymagań

Po wnikliwym przeanalizowaniu aplikacji dostępnych na rynku została utworzona lista cech, którymi powinna charakteryzować się aplikacja Vetty:

- **responsywność**: Aplikacja powinna być responsywna, dzięki czemu interfejs użytkownika jest przejrzysty oraz czytelny niezależnie od rozmiaru ekranu.
- **bezpieczeństwo**: Aplikacja powinna zawierać mechanizmy uwierzytelniania i autoryzacji oraz powinna dbać o odpowiednie zabezpieczenie danych użytkowników, w tym ich haseł.
- **łatwość obsługi:** Aplikacja powinna posiadać intuicyjny oraz łatwy w obsłudze interfejs, nawet dla użytkowników niezaznajomionych z aplikacją.
- **łatwość utrzymania oraz rozbudowy**: Aplikacja powinna zostać stworzona według dobrych reguł programistycznych takich jak: KISS (keep it stupid simple) oraz DRY (don't repeat yourself), a także powinna przestrzegać zasad czystego kodu. Czysty i modułowy kod ułatwia prace nad rozwojem aplikacji i zwiększa jej skalowalność.
- **możliwość wymiany informacji z innymi użytkownikami:** użytkownicy powinni mieć możliwość komunikowania się między sobą oraz wymiany wiedzy o historii chorób swoich zwierząt.
- **zapis danych do analizy**: Aplikacja powinna przechowywać dane badań psów i kotów w celu ich dalszej analizy bądź wykorzystania w algorytmach sztucznej inteligencji.

Trzymając się wyżej wymienionych zasad oraz biorąc przykład z czołowych rozwiązań na rynku jesteśmy w stanie stworzyć listę wymagań funkcjonalnych oraz niefunkcjonalnych aplikacji, która wykorzysta najlepsze cechy aplikacji dostępnych na rynku i uniknie błędów występujących w niektórych produktach.

2.2. Dziedzinowy słownik pojęć

Zbiór pojęć wykorzystywanych w trakcie opisu implementacji aplikacji:

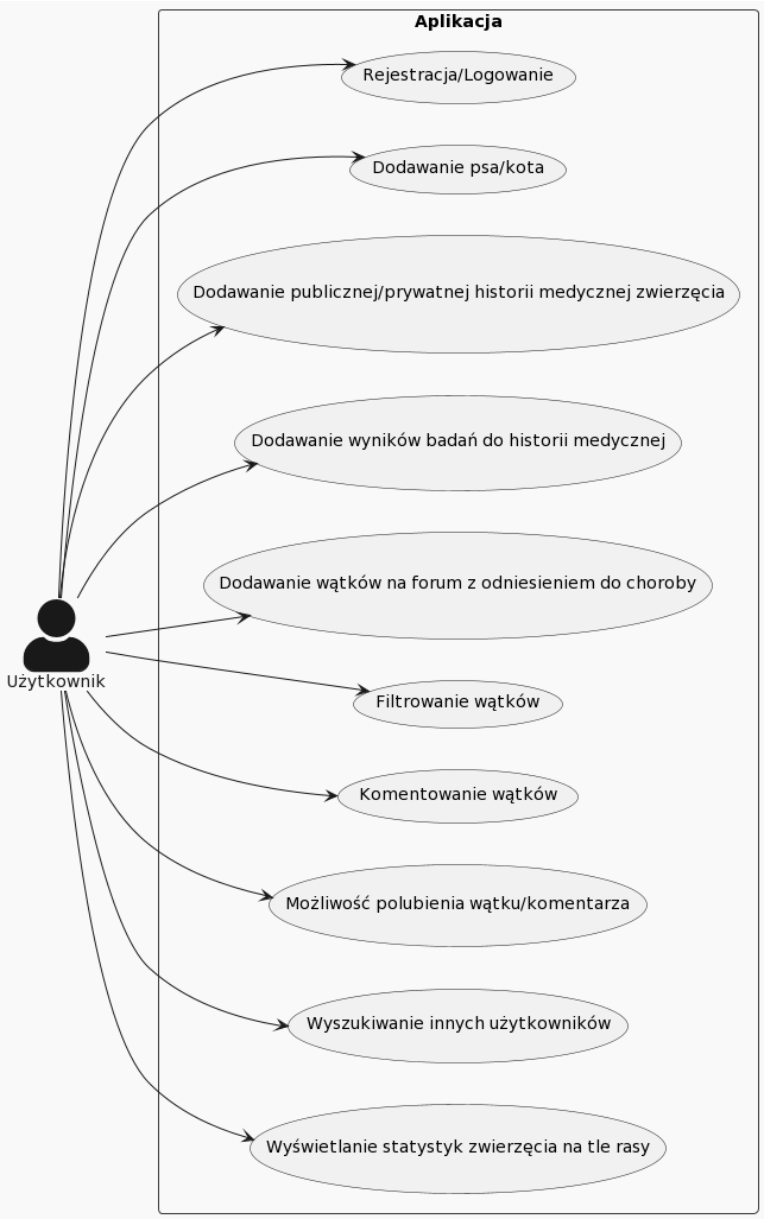
- Użytkownik – osoba korzystająca z aplikacji Vetty.
- Historia medyczna zwierzęcia - dane o przebytej chorobie zawierające diagnozę, opis przebytej choroby, datę rozpoczęcia choroby, informację czy historia jest publiczna, informacje o zwierzęciu, a także listę tagów.
- Publiczna historia medyczna – historia, którą mogą wyświetlić inni użytkownicy (niebędący właścicielami zwierzęcia).
- Właściciel zwierzęcia – użytkownik do którego konta przypisane jest dane zwierzę.
- Wyniki badań – lista czynników badań krwi z przypisanymi wartościami oraz informacją, czy dany czynnik jest w normie.
- Wątek – wpis na forum zawierający historię medyczną pojedynczego zwierzęcia, tytuł, opis napotkanego problemu oraz komentarze użytkowników.
- Filtrowanie wątków – wyszukiwanie wątków według kryteriów takich jak wiek oraz rasa zwierzęcia, czy tagi.
- Tagi – umowne znaczniki tekstowe opisujące niedoczynności oraz nadczynności badań krwi zwierzęcia zawarte w historii medycznej.
- Statystyki zwierzęcia – wykresy czynników badań krwi zwierzęcia ujęte w ramach czasowych na tle tychże wartości średnich dla rasy.
- Aplikacja internetowa – „program komputerowy, który pracuje na serwerze i komunikuje się poprzez sieć komputerową z hostem użytkownika komputera. W tym celu wykorzystuje się przeglądarkę internetową użytkownika, będącego interaktywnym klientem aplikacji internetowej pełniąc funkcję interfejsu użytkownika aplikacji internetowej. [2]
- Serwis – element aplikacji serwerowej przechowywany w module service, który odpowiada za przetwarzanie logiki biznesowej.

2.3. Wymagania funkcjonalne

Na podstawie przedstawionych wymagań przygotowana została lista wymagań funkcjonalnych aplikacji internetowej Vetty przedstawiona na **Rysunek 1**:

- użytkownik ma możliwość zarejestrowania się oraz logowania za pomocą e-mailu i hasła;
- użytkownik ma możliwość dodania zwierzęcia (pies, kot) do listy swoich zwierząt:
  - użytkownik ma możliwość dodania zdjęcia swojego zwierzęcia;
- użytkownik ma możliwość dodania historii chorób do każdego ze swoich zwierząt:

    - historia choroby zawiera: datę, informację o zwierzęciu której dotyczy, wyniki badań krwi, tagi pomagające w wyszukiwaniu, diagnozę oraz opis choroby;
- użytkownik ma możliwość dodania do każdej historii choroby wyników badań krwi , przepisując je z wydruku bądź przesyłając plik z wynikami badań:
    - plik z wynikami powinien być w formacie pdf, jpg lub png;
    - tagi opisujące historię choroby dodadzą się automatycznie po wczytaniu wyników badań;
- użytkownik ma możliwość decydowania o zakresie udostępniania każdej historii choroby swojego psa (opcje publiczna/prywatna);
- użytkownik ma możliwość założenia wątku na forum z możliwością podpięcia historii choroby swojego zwierzęcia;
- użytkownik ma możliwość wyszukiwania wątków innych użytkowników z opcją filtrowania z wyborem:
    - gatunku zwierzęcia;
    - rasy zwierzęcia;
    - wieku zwierzęcia;
    - tagów dołączonych do historii choroby;
- użytkownik ma możliwość dodawania komentarzy do wątków na forum;
- użytkownik ma możliwość polubienia wątków oraz komentarzy;
- użytkownik ma możliwość wyszukiwania innych użytkowników za pomocą ich adresu email;
- użytkownik ma możliwość przeglądania statystyk poszczególnych czynników krwi na przestrzeni czasu na tle średnich wartości dla danej rasy psa/kota;



Rysunek 1. Wymagania funkcjonalne aplikacji

2.4. Wymagania niefunkcjonalne

Na podstawie analizy wymagań, jak i listy wymagań funkcjonalnych, została stworzona lista wymagań niefunkcjonalnych:

- aplikacja będzie mogła być uruchomiana na przeglądarkach internetowych opartych w budowie o silnik Chromium.
- Aplikacja będzie mogła być uruchomiona na komputerze/laptopie;
- aplikacja będzie w stanie przetwarzać zapytania w czasie poniżej 4 sekund / zapytanie w 95% przypadków;
- aplikacja będzie używać standardu JWT (ang. JSON Web Token) przy wymianie danych pomiędzy użytkownikiem, a aplikacją w celu zapewnienia autentyczności danych;

- aplikacja będzie w bezpieczny sposób przechowywać hasła użytkowników, aby zapobiec wykradnięciu danych;
- komunikacja między komponentami aplikacji odbywa się za pomocą protokołu HTTP (ang. Hypertext Transfer Protocol) i stylu REST.

3. Projekt aplikacji

W rozdziale tym przedstawiona zostanie struktura całej aplikacji Vetty oraz opis poszczególnych elementów systemu. Ponadto opisane zostaną wzorce architektoniczne oraz protokoły komunikacyjne wykorzystane podczas implementacji aplikacji.

3.1. Aplikacja internetowa

W dzisiejszych czasach większość aplikacji tworzona jest z myślą o dostępie za pomocą Internetu. Dzięki temu dostęp do wszelkich zasobów jest łatwy oraz powszechnie dostępny. Z roku na rok coraz popularniejsze stają się aplikacje internetowe, do których dostęp uzyskujemy bezpośrednio z poziomu przeglądarki internetowej. W działaniu aplikacji internetowej musi pośredniczyć serwer lub chmura obliczeniowa. W celu implementacji tego rodzaju aplikacji używa się różnych języków programowania (np. PHP, Java, C#), jak i bibliotek (np. Node, Spring, ASP.NET).

Dzięki takiemu rozwiązaniu mamy dostęp do aplikacji z dowolnego urządzenia, zarówno z komputera, telewizora, czy telefonu komórkowego. Dodatkowo są one niezależne od systemu operacyjnego urządzenia na którym działają. Innym aspektem przemawiającym za aplikacjami webowymi jest ich bezpieczeństwo, w którego skład wchodzą certyfikaty SSL/TLS, mechanizm autoryzacji oraz uwierzytelniania, system zarządzania uprawnieniami, a w niektórych przypadkach nawet audyty bezpieczeństwa. Dzięki łatwej dystrybucji aplikacji internetowych aktualizacje bezpieczeństwa mogą być dostarczane na bieżąco co zwiększa zaufanie, jak i komfort korzystania użytkowników.

Dlatego też w projekcie pracy zdecydowałem się na stworzenie aplikacji webowej, która umożliwia klientom korzystanie z aplikacji w dowolnym miejscu, na dowolnym urządzeniu z dostępem do przeglądarki internetowej. Dzięki temu klient może korzystać z produktu zarówno podczas wizyty weterynarza korzystając z telefonu komórkowego, jak i na komputerze osobistym w domu, czy na wakacjach.

3.2. Architektura aplikacji

Podczas tworzenia aplikacji wykorzystano architekturę klient-serwer, która umożliwia zorganizowanie struktury projektu w dwie komunikujące się ze sobą aplikacje. Struktura ta dzieli zadania między dostawcę zasobów, danych, bądź usług, zwanego serwerem, a żądającego usługę lub dane, zwanego klientem.

Serwer jest komponentem, który obsługuje żądania klientów i udostępnia wybrane zasoby, bądź też wykonuje określone operacje w odpowiedzi na przysłane żądania. Najczęściej odpowiedź na przychodzące żądanie składa się z ustalenia tożsamości autora żądania, przetworzenia go według ustalonej logiki biznesowej, udostępnienia odpowiednich zasobów, bądź też przechowania dostarczonych danych w bazie danych. W celu ustrukturyzowania komunikacji serwer wystawia interfejs programistyczny aplikacji – API (ang. application

programming interface), który opisuje jakie dane, typ zapytania oraz na jaki adres klient powinien wysyłać żądania [3].

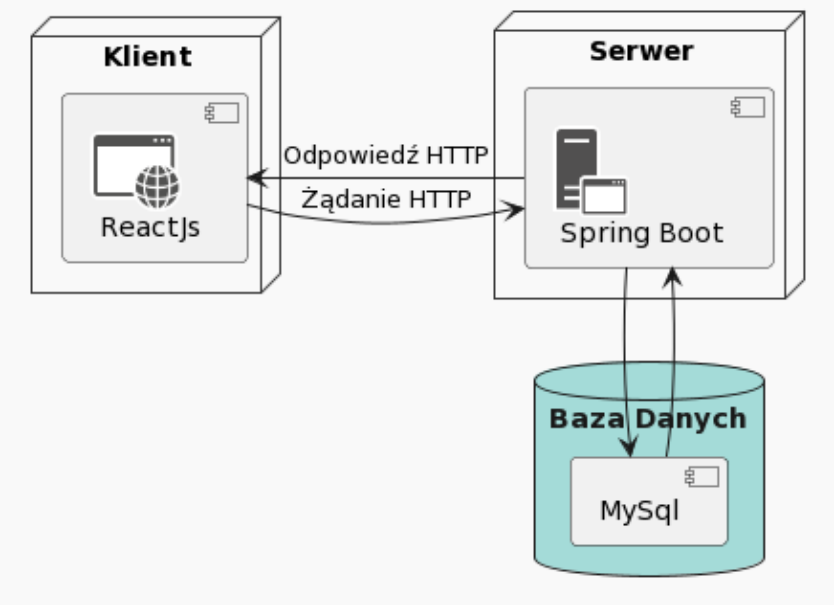
Zadaniem klienta jest dostarczenie użytkownikowi odpowiedniego interfejsu, który udostępni niezbędne informację oraz umożliwi komunikację z serwerem w postaci żądań pierwotnie zdefiniowanych przez serwer. Ostatecznie klient po otrzymaniu odpowiedzi na wysłane żądanie powinien aktualizować prezentowaną treść w przypadku zmiany danych zawartych w aplikacji klienckiej.

W takim modelu klient i serwer działają niezależnie, a komunikacja między nimi odbywa się z użyciem protokołu HTTP i działa na zasadzie żądanie-odpowiedź. Klient inicjuje żądanie, a serwer na nie odpowiada. Dzięki temu uzyskujemy skalowalność systemu, ponieważ wielu klientów może komunikować się z serwerem jednocześnie.

3.2.1. Struktura systemu

Struktura systemu Vetty przedstawiona na **Rysunek 2** składa się z poszczególnych komponentów:

1. Aplikacji klienckiej
2. Aplikacji serwerowej
3. Bazy danych



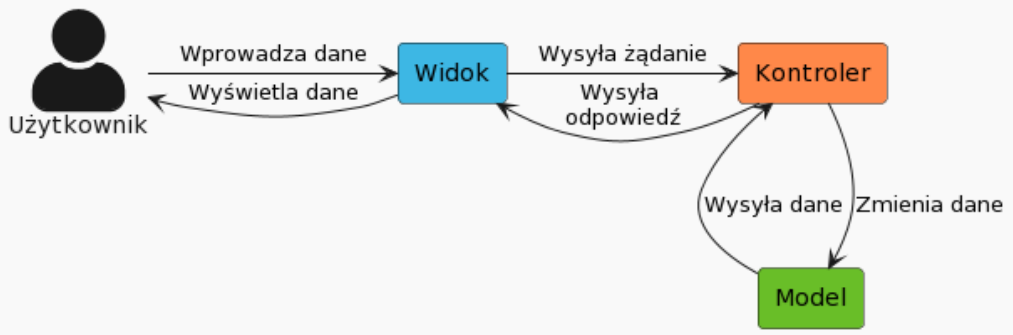
Rysunek 2. Architektura systemu

Aplikacja kliencka została zaprojektowana z użyciem biblioteki React.js języka JavaScript i udostępnia ona interfejs użytkownikowi, który poprzez interakcję z interfejsem może wysyłać zapytania do serwera.

Serwer odpowiedzialny jest za odbieranie żądań, przetwarzanie ich i przechowywanie, bądź otrzymywanie odpowiednich zasobów z bazy danych.

W projekcie aplikacji użyto bazy danych MySQL. Jest ona odpowiedzialna za przechowywanie szczegółowych informacji o użytkownikach, zwierzętach, ich historii medycznej, wynikach badań, a także wątkach.

3.2.2. Model MVC



Rysunek 3. Wzorzec architektoniczny MVC

Aplikacja została utworzona z użyciem wzorca architektonicznego MVC (ang. Model–View–Controller), którego model został przedstawiony na **Rysunek 3**. Jest to popularne podejście do projektowania i organizacji struktury aplikacji, którego głównym celem jest rozdzielenie odpowiedzialności logiki biznesowej od prezentacji danych. Składa się on z trzech komponentów: **modelu**, **widoku** oraz **kontrolera**, które odpowiedzialne są za różne zadania [4].

Model definiuje, jakie dane powinna zawierać aplikacja oraz odpowiada za część logiki biznesowej.

Widok definiuje sposób wyświetlania danych aplikacji oraz umożliwia interakcję z użytkownikiem.

Kontroler zarządza przepływem danych pomiędzy modelem, a widokiem.

Każdy z wyżej wymienionych elementów występuje w aplikacji Vetty. Komponenty aplikacji klienckiej, które zostały stworzone z wykorzystaniem języka JavaScript odzwierciedlają widoki z modelu MVC. Kontroler wraz z Modelem odzwierciedlone zostały w części odpowiedzialnej za przetwarzanie danych – aplikacji serwerowej. Model został umieszczony w poszczególnych serwisach aplikacji serwerowej, które odpowiedzialne są za realizowanie logiki biznesowej, pobieranie, modyfikowanie oraz zapisywanie danych. Kontroler natomiast został umieszczony w dostarczonych przez moduł Spring MVC kontrolerach. Opisują one sposób komunikacji z widokiem, obsługują żądania HTTP, a także wykonuje odpowiednie akcje na Modelu poprzez rozdzielenie pracy odpowiednim serwisom.

3.2.3. Protokół http

Aplikacja kliencka komunikuje się z aplikacją serwerową z wykorzystaniem protokołu http. Protokół ten charakteryzuje się bezstanowością oraz jest oparty na żądaniach i odpowiedziach, co umożliwia przesyłanie danych między klientem, a serwerem. Standardowy format żądania HTTP składa się z:

- metody – określa on rodzaj żądania taki jak:
  - GET – służy do pobierania zasobów z serwera;
  - POST – służy do przesyłania danych do serwera;
  - PUT – służy do modyfikacji zasobów z serwera;
  - DELETE – służy do usuwania zasobów z serwera;
- URL – określa on adres URL (ang. Uniform Resource Locator), czyli lokalizację zasobu na serwerze;
- nagłówków żądania – mogą zawierać dodatkowe informacje np. dane służące do uwierzytelnienia klienta przez serwer;
- treści żądania – zawiera dane przesyłane do serwera;

Po otrzymaniu żądania serwer przetwarza je i w wyniku operacji zwraca odpowiedź, która składa się z:

- statusu odpowiedzi – informuje on o powodzeniu operacji bądź o konkretnym błędzie w przypadku niepowodzenia;
- nagłówków odpowiedzi – zawierają one dodatkowe informacje o odpowiedzi;
- ciała odpowiedzi – zawiera ono dane przesyłane z serwera do klienta;

3.2.4. Komunikacja Serwer-Baza Danych

Komunikacja pomiędzy aplikacją serwerową, a bazą danych została zrealizowana z użyciem interfejsu JDBC (ang. Java Database Connectivity). Jest to standardowy interfejs programowania aplikacji API, który umożliwia aplikacjom Java połączenie do dowolnego systemu zarządzania bazą danych, o ile istnieje sterownik do tego konkretnego systemu. Składa się on ze standardowych interfejsów i klas dostarczających funkcjonalności potrzebnych do połączenia z bazą danych.

4. Implementacja

W rozdziale tym zostaną zaprezentowane szczegóły implementacyjne poszczególnych części systemu wraz z przedstawieniem technologii użytych w procesie rozwoju aplikacji.

4.1. Baza danych

W celu przechowywania danych w aplikacji użyto relacyjnej bazy MySQL. Relacyjne bazy danych składają się z tabel w których przechowywane są dane, a na same tabele składają się kolumny (atrybuty) oraz wiersze, które przedstawiają konkretne instancje danych. Taka struktura ułatwia organizację oraz przechowywanie danych.

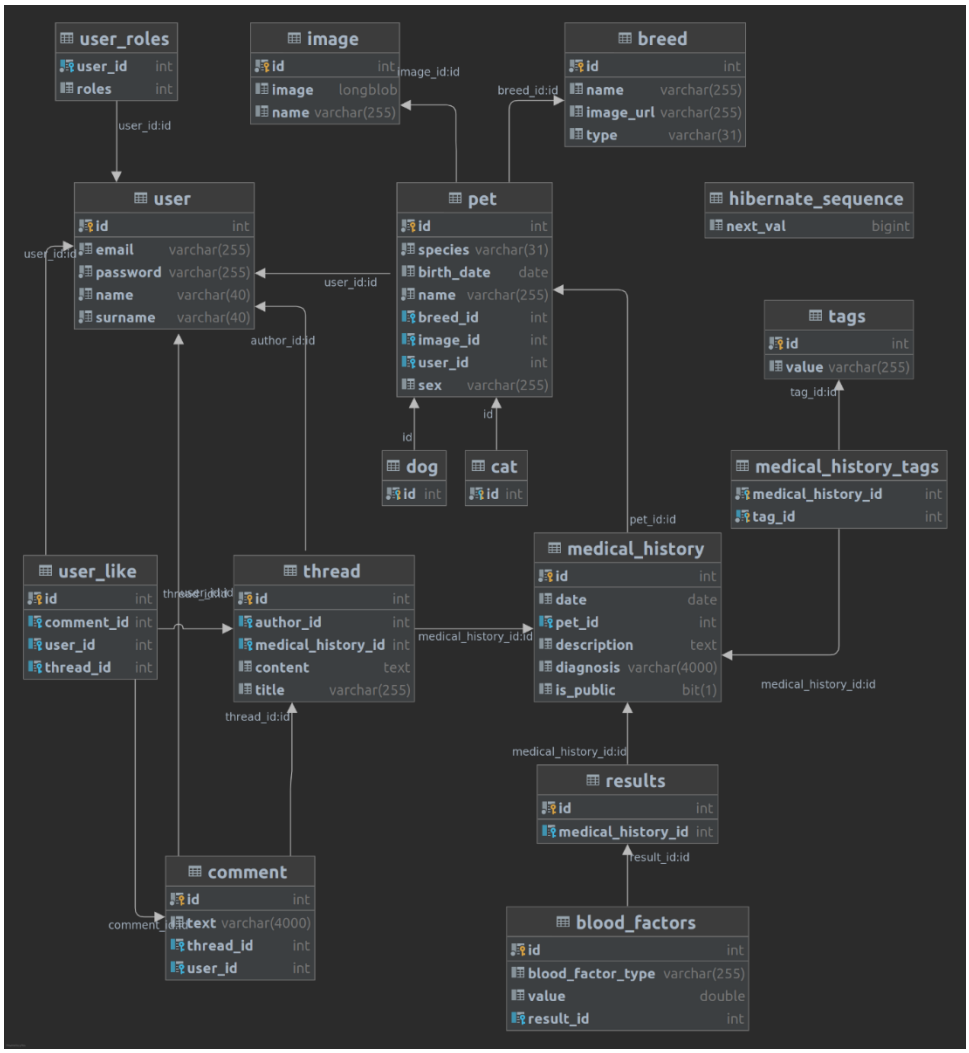
4.1.1. MySQL

MySQL to najpopularniejsza na świecie relacyjna baza danych typu open-source, która charakteryzuje się:

- strukturą tabelaryczną - Dane przechowywane są w postaci tabel, gdzie każda tabela składa się z kolumn (atrybutów) i wierszy (rekordów).
- relacjami między tabelami – Tabele są powiązane relacjami, które określają, jak dane są ze sobą powiązane. Relacje te są zdefiniowane przy użyciu kluczy głównych i obcych, które pełnią funkcję identyfikatorów oraz wskaźników.
- językiem zapytań SQL – MySql korzysta z języka zapytań SQL (ang. Structured Query Language) do manipulacji danymi.

MySQL został wybrany jako system zarządzający bazą danych w projekcie, ponieważ jest kompaktowy, łatwo skalowalny oraz ze względu na to, że sprawdza się bardzo dobrze w przypadku nieskomplikowanych schematów bazy danych.

4.1.2. Model relacyjny



Rysunek 4. Schemat bazy danych wygenerowany w IntellijIDEA

Model relacyjny (model opisujący encje występujące w bazie danych oraz relacje pomiędzy poszczególnymi encjami) został przedstawiony na **Rysunek 4**. Został on stworzony w projekcie z wykorzystaniem modułu Spring Data, którego opis znajduje się w rozdziale 4.2.4., a także dzięki narzędziu Hibernate. Hibernate jest narzędziem ORM (ang. Object-Relational Mapping), które umożliwia mapowanie obiektowo-relacyjne, dzięki któremu klasy stworzone w języku Java mogą być odzwierciedlone jako tabele w bazie danych. Do stworzenia takiego mapowania, jak na **Rysunek 5**, wykorzystałem adnotacje, które są alternatywą dla plików konfiguracyjnych w przypadku korzystania z narzędzia Hibernate. Są one specjalnymi znacznikami umieszczanymi w kodzie źródłowym klas, które wskazują narzędziu, jak dokonać mapowania. Przykładem może być adnotacja '@Entity', oznaczająca klasę jako encję, czy adnotacja '@Column', która definiuje mapowanie na kolumnę.

```java
@Getter
@Setter
@RequiredArgsConstructor
@Entity
@Table(name = "tags")
public class Tag {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id", nullable = false)
    private Integer id;

    private String value;

    @ManyToMany(mappedBy = "tags", fetch = FetchType.LAZY)
    @JsonIgnore
    private Set<MedicalHistory> medicalHistories = new
HashSet<>();
}
```

Rysunek 5. Mapowanie tabeli za pomocą klasy w języku Java

4.2. Aplikacja serwerowa

W celu przetwarzania danych, logiki biznesowej oraz komunikacji z bazą danych została stworzona aplikacja serwerowa. Zajmuje się ona obsługą żądań nadchodzących z aplikacji klienckiej oraz zarządzaniem danymi zwierząt, ich historiami medycznymi i danymi użytkowników.

4.2.1. Java

Do stworzenia aplikacji serwerowej został wykorzystany język Java. Jest to współbieżny, obiektowy język programowania szeroko wykorzystywany w projektach aplikacji webowych. Został on wykorzystany do implementacji aplikacji ze względu na mnogość dostępnych bibliotek, które ułatwiają implementację oraz codzienną pracę z kodem, a także ze względu na wsparcie dla programowania obiektowego. Biblioteki dostępne w środowisku Javy pozwalają programiście skupić się na rozwijaniu logiki biznesowej aplikacji, a nie na walce z problemami infrastrukturalnymi.

4.2.2. Spring Boot

Jedną z głównych bibliotek języka Java, która została wykorzystana podczas rozwoju aplikacji jest Spring Boot [5]. Jest to popularne narzędzie stworzone w oparciu o platformę Spring, które pozwala na szybsze i łatwiejsze tworzenie aplikacji webowych. W pracy biblioteka Spring Boot została wykorzystana w celu ułatwienia procesu tworzenia samodzielnej aplikacji, poprzez skorzystanie z wbudowanego serwera Tomcat. Dzięki niemu można uzyskać działający szkielet aplikacji serwerowej w kilka minut. Dodatkowo biblioteka dostarcza auto-konfigurację aplikacji oraz bazowego narzędzia Spring co ułatwiło rozpoczęcie pracy nad projektem.

4.2.3. Spring Security

Biblioteka Spring Security została wykorzystana w celu zapewnienia podstawowego bezpieczeństwa aplikacji [6]. Z jej użyciem został zaimplementowany mechanizm rejestracji użytkowników z wykorzystaniem adresu email oraz hasła, a także mechanizm uwierzytelniania z użyciem tokenów JWT (ang. JSON Web Token). W oparciu o funkcjonalność Spring Security stworzono wzorzec projektowy pełnomocnika (proxy), który przechwytuje przychodzące żądania z aplikacji klienckiej i uwierzytelnia właściciela żądania, przed przekazaniem go do kolejnej warstwy aplikacji serwerowej.

4.2.4. Spring Data

Aby ułatwić i przyśpieszyć proces tworzenia aplikacji wykorzystano moduł Spring Data. Zapewnia on możliwość mapowania obiektowo-relacyjnego ORM, czyli tworzy połączenia między obiektami w kodzie (klasami), a danymi przechowywanymi w bazie danych (tabelami). Biblioteka zapewnia również możliwość tworzenia operacji CRUD (Create, Read, Update, Delete) na obiektach, automatycznego generowania zapytań SQL na podstawie nazewnictwa metod, co pozwala na wysyłanie zapytań bez zbędnego nakładu czasowego. Dodatkowo Spring Data dostarcza możliwość filtracji oraz sortowania zapytań.

4.2.5. Maven

Narzędzie Maven [7] zostało użyte w celu umożliwienia zarządzania projektem oraz łatwego budowania aplikacji w środowisku Javy. Jest to platforma, która umożliwia zarządzanie zależnościami w projekcie poprzez definiowanie bibliotek oraz modułów w pliku konfiguracyjnym pom.xml, gdzie można zdefiniować etapy budowania, zależności, wersje kompilatora oraz inne ustawienia. Dzięki temu programista nie musi ręcznie pobierać wymaganych modułów, a także pamiętać o usuwaniu nieużywanych.

4.2.6. Tesseract

Aby umożliwić klientom aplikacji automatyczne wprowadzanie badań krwi swoich zwierząt, wykorzystano narzędzie Tesseract [8]. Biblioteka została użyta w celu zeskanowania wyników poszczególnych czynników oraz wprowadzania ich do bazy danych. Jest to silnik optycznego rozpoznawania znaków (OCR) stworzony dla różnych systemów operacyjnych. Jego działanie polega na odczytywaniu napisów, znaków zawartych w zdjęciu lub dokumencie przy użyciu sztucznej inteligencji, a następnie zwracaniu tablicy znaków.

4.2.7. Postman

Narzędzie Postman [9] zostało wykorzystane w celu przetestowania interfejsu API (ang. Application Programming Interface). Jest to platforma do tworzenia i korzystania z interfejsów API. Narzędzie to uprościło proces

testowania interfejsu poprzez automatyczne dodanie tokenów JWT do zapytań, które są wymagane w celu ich uwierzytelnienia.

4.2.8. IntelliJ IDEA

Środowisko programistyczne IntelliJ IDEA zostało wykorzystane w celu ustrukturyzowania projektu oraz ułatwienia procesu rozwoju aplikacji [10]. Jest to zaawansowane środowisko programistyczne IDE (ang. integrated development environment) stworzone przez firmę JetBrains. Oferuje ono wydajne i wszechstronne środowisko do tworzenia aplikacji w różnych językach programowania, w tym Java. Dostarcza również wbudowany debugger, który ułatwił znajdowanie błędów w aplikacji.

4.2.9. System kontroli wersji Git

GIT jest rozproszonym systemem kontroli wersji, który jest szeroko wykorzystywany w projektach informatycznych [11]. Jest to narzędzie, które umożliwia programistom śledzenie zmian w kodzie źródłowym oraz zarządzanie wersjami projektu. Został on wykorzystany w projekcie w celu zabezpiecza przed ewentualną utratą kodu w przypadku usterek pamięci komputera. Dodatkowo kontrola wersji dostarczana przez GIT ułatwiła odszukiwanie błędów oraz przywrócenie działającego kodu w przypadku krytycznych błędów.

4.2.10. Struktura projektu

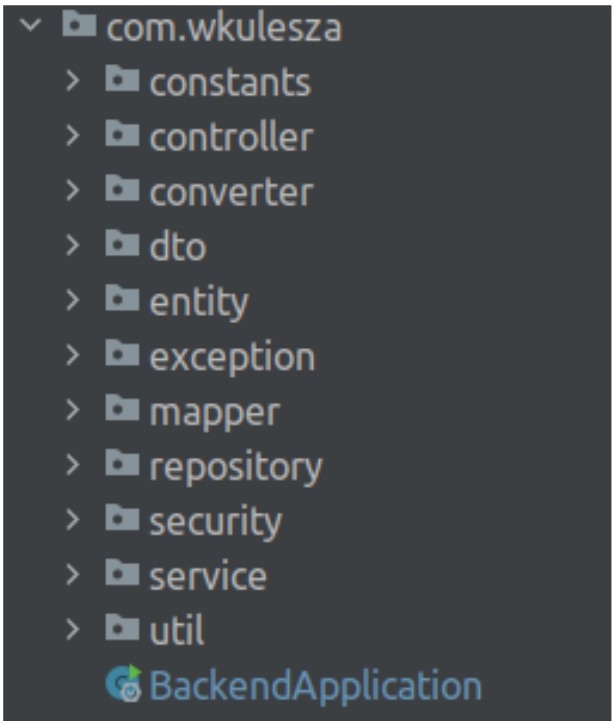
Aplikacja serwerowa została zaprojektowana z użyciem wzorca architektury warstwowej, znanej również jako wzorzec architektury n-warstwowej [12]. Wzorzec ten polega na organizowaniu komponentów w warstwy, gdzie każda z nich pełni określoną rolę oraz odpowiedzialność w aplikacji. Na przykład warstwa biznesowa, która została zaimplementowana z użyciem serwisów, odpowiada za wykonywanie określonych reguł biznesowych związanych z żądaniem. Dzięki takiemu podejściu uzyskujemy:

- Separację odpowiedzialności, dzięki której otrzymujemy lepszą czytelność kodu oraz łatwiejsze zrozumienie aplikacji.
- Modularność i skalowalność, dzięki której otrzymujemy łatwiejsze zarządzanie kodem, wprowadzanie do niego zmian oraz rozwijanie poszczególnych warstw niezależnie.
- Łatwość testowania, ponieważ każda warstwa może być testowana niezależnie od pozostałych. Warstwy mogą być testowane za pomocą testów jednostkowych, integracyjnych i funkcjonalnych, co ułatwia weryfikację poprawności działania każdej warstwy.

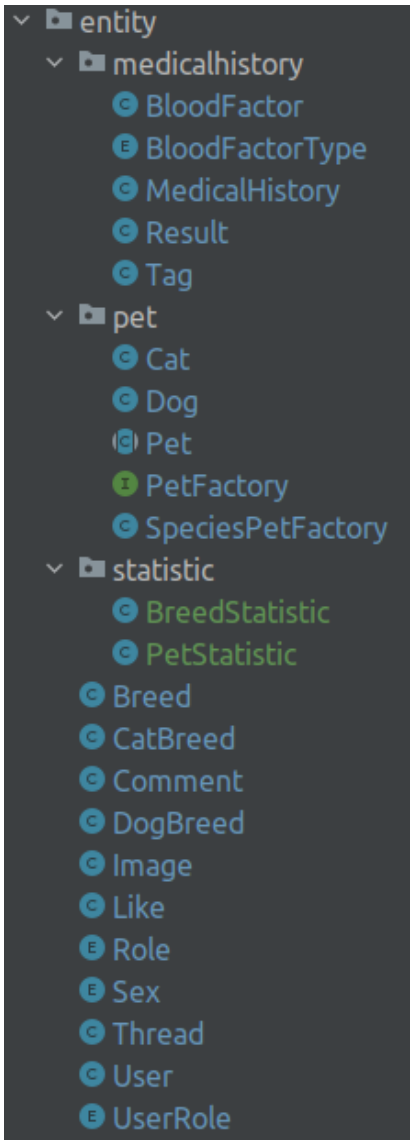
W oparciu o ten wzorzec kod został podzielony na pakiety przedstawione na **Rysunek 6**, które utworzyły architekturę z zamkniętymi warstwami. Oznacza to, że żądanie może przejść tylko do warstwy poniżej, bez możliwości pominięcia warstwy. Przykładem działania zamkniętych warstw może być żądanie przechodzące przez **warstwę prezentacji** (kontroler), następnie przez

warstwę biznesową (serwis) do **warstwy trwałości** (repozytorium), gdzie ostatecznie może zostać skierowane do warstwy bazy danych.

Na **Rysunek 7** przedstawiono strukturę pakietu odpowiedzialnego za strukturę danych. Została ona podzielona na mniejsze pakiety, które były stworzone w oparciu o przynależność do poszczególnych domen, dzięki czemu można było uzyskać lepszą organizację kodu, która przyczyniła się do zwiększonej czytelności kodu.



Rysunek 6. Struktura aplikacji serwerowej



Rysunek 7. Struktura pakietu entity

Warstwa prezentacji w aplikacji Vetty odpowiada za część kontrolera we wzorcu architektonicznym MVC opisanym w rozdziale 3.2.2. Odpowiada ona za obsługę zapytań napływających z aplikacji klienckiej, przekazywaniu ich do poszczególnych serwisów w celu przetworzenia, a także zwracania odpowiedzi z wykorzystaniem protokołu HTTP. Po uzyskaniu danych z serwisu kontroler dodatkowo przetwarza otrzymane obiekty do obiektów DTO (ang. Data Transfer Object). Jest to wzorzec projektowy, który służy do odseparowania warstw aplikacji oraz umożliwia przenoszenie danych w sposób niezależny od struktury wewnętrznej obiektów logiki biznesowej. Dzięki temu programista ma możliwość kontrolowania danych, które są udostępniane aplikacji klienckiej, co może zwiększać bezpieczeństwo aplikacji. W projekcie warstwa ta została umieszczona w pakiecie controller, a obiekty DTO przesyłane do aplikacji klienckiej

przechowywane są w pakiecie dto. Mapowanie obiektów przesyłanych z warstwy biznesowej do obiektów transferu danych zostało zaimplementowane z wykorzystaniem biblioteki ModelMapper. Ułatwia ona mapowanie danych między obiektami różnych klas. Plusem tej biblioteki jest ułatwienie pracy programisty przez automatyczne mapowanie pól, bez konieczności pisania dodatkowego kodu razem z możliwością konfiguracji bardziej złożonych mapowań [13]. Działanie przykładowego kontrolera zostało przedstawione na **Rysunek 8**.

```
@RestController
@CrossOrigin(maxAge = 3600)
@RequestMapping(path = "/api/statistics")
@RequiredArgsConstructor
public class StatisticController {

    private final StatisticService statisticService;


    @PostMapping(value = "/breed")
    @CrossOrigin
    public ResponseEntity<List<BreedStatistic>>
getStatisticsForBreed(@RequestParam String breedName) {
        List<BreedStatistic> statistics =
statisticService.getBreedStatistics(breedName);
        return ResponseEntity.ok(statistics);
    }

    @GetMapping(value = "/pet/{petId}")
    @CrossOrigin
    public ResponseEntity<List<PetStatisticDto>>
getStatisticsForPet(@PathVariable Integer petId) {
        List<PetStatistic> statistics =
statisticService.getPetStatisticsByPetId(petId);
        List<PetStatisticDto> petStatisticDtos =
PetStatisticConverter.convert(statistics);
        return ResponseEntity.ok(petStatisticDtos);
    }
}
```

Rysunek 8. Przykładowy kod kontrolera

Warstwa biznesowa znajduje się poniżej warstwy prezentacji w architekturze warstwowej. Wykonywana jest w niej logika biznesowa, a co najważniejsze łączy warstwę prezencji (kontrolerów) z warstwą dostępu do danych. W projekcie została zaimplementowana jako poszczególne serwisy znajdujące się w pakiecie service. Do ich głównych zadań oprócz implementacji logiki biznesowej należy, koordynacja działań poprzez wywoływanie innych serwisów, zewnętrznych usług bądź synchronizowania operacji. Dodatkowo serwisy mogą korzystać z repozytoriów poprzez wydobywanie z nich danych,

zapisywanie ich, aktualizowanie i usuwanie. Działanie przykładowego serwisu zostało przedstawione na **Rysunek 9**.

```
@RequiredArgsConstructor
@Service
public class CommentService {

    private final CommentRepository commentRepository;

    private final LikeRepository likeRepository;

    public Comment getCommentByThreadIdAndUserEmailAndText(Integer
threadId, String userEmail, String text) {
        return
commentRepository.getCommentByThreadIdAndUserEmailAndText(threadId,
userEmail, text);
    }

    public Comment likeComment(Like like) {
        Comment comment = like.getComment();
        if (comment.getLikes().contains(like)) {
            comment.getLikes().remove(like);
            likeRepository.delete(like);
        } else {
            comment.getLikes().add(like);
        }
        return commentRepository.save(comment);
    }

    public Comment getCommentById(Integer commentId) {
        return
commentRepository.findById(commentId).orElseThrow(IllegalArgumentException:
:new);
    }
}
```

Rysunek 9. przykładowy kod serwisu

Warstwa dostępu do danych jest odpowiedzialna za komunikację z bazą danych i jej głównym celem jest zapewnienie sposobu odczytu oraz zapisu danych w aplikacji. W projekcie została zrealizowana z wykorzystaniem modułu Spring Data JPA oraz repozytoriów dostarczanych przez ten moduł, których przykładowe działanie zostało przedstawione na **Rysunek 10**.

```
@Repository
public interface BreedRepository extends JpaRepository<Breed, Integer> {
    Boolean existsByName(String breedName);

    Optional<Breed> findByName(String name);

}
```

Rysunek 10. przykładowy kod repozytorium

4.2.11. REST API

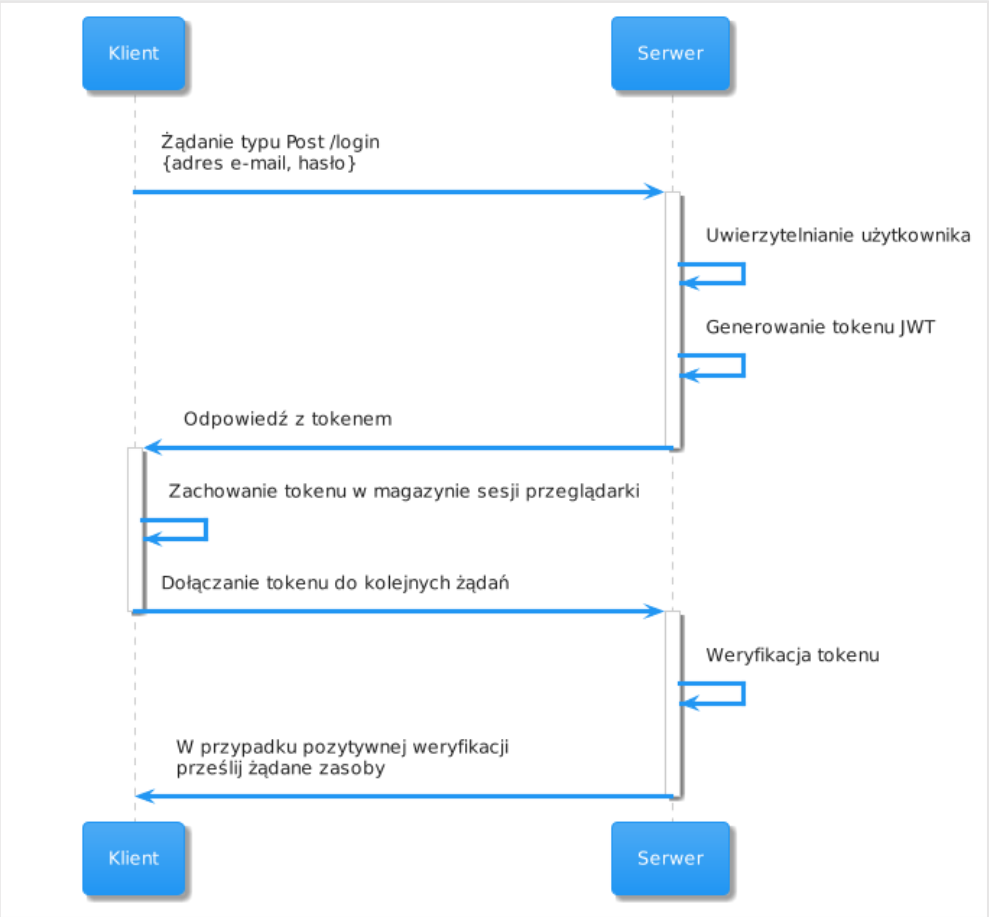
Aplikacja Serwerowa w celu komunikacji z aplikacją kliencką udostępnia interfejs programowania (API) z użyciem kontrolerów. Jest to zbiór reguł ściśle opisujący w jaki sposób aplikacja kliencka może komunikować się z serwerem.

W projekcie interfejs został zaprojektowany w oparciu o architekturę REST (ang. Representational State Transfer) [14], która opiera się na kilku zasadach oraz konwencjach określających sposób budowania interfejsów programistycznych. Spośród kilku zasad opisujących architekturę REST zostały zastosowane:

- łatwość użycia - REST API oparte jest na standardowych protokołach HTTP, takich jak GET (odczyt danych), POST (dostarczanie danych do serwera), PUT (aktualizacja danych), DELETE (usuwanie danych). Dzięki temu programiście łatwiej zrozumieć oraz rozwijać interfejs. Żądania mogą być wywoływane za pomocą standardowych żądań HTTP i mogą otrzymywać odpowiedzi w postaci danych w formacie np. JSON lub XML.
- jednolity interfejs - Określa on jednolity sposób interakcji z serwerem niezależnie od urządzenia. W projekcie został on oparty na zasobach, co oznacza, że poszczególne zasoby są identyfikowane w żądaniach, np. "api/users".
- bezstanowość - REST API jest bezstanowe, co oznacza, że każde żądanie jest niezależne od poprzednich. Serwer nie przechowuje żadnego stanu klienta, co upraszcza zarządzanie sesjami i skalowanie aplikacji.
- architektura klient-serwer;
- warstwowa architektura komponentów;

Dzięki zastosowaniu się do zasad REST otrzymujemy prostotę, elastyczność, skalowalność, niezależność od platformy oraz możliwość łatwej integracji z innymi systemami.

4.2.12. Zabezpieczenia JWT



Rysunek 11. Proces generacji oraz weryfikacji tokenu JWT

Aplikacja serwerowa dba o bezpieczeństwo zasobów przechowywanych w bazie danych oraz o dane użytkowników. W celu zabezpieczenia wrażliwych danych została zaimplementowana usługa uwierzytelnienia i autoryzacji użytkownika, a także mechanizm szyfrowania haseł użytkowników z wykorzystaniem funkcji haszującej Bcrypt. Proces uwierzytelniania został zrealizowany z wykorzystaniem tokenów JWT (ang. JSON Web Token) [15]. Jest to otwarty standard definiujący sposób reprezentacji bezpiecznej informacji pomiędzy aplikacją kliencką a serwerem w postaci obiektów JSON. Tokeny JWT służą do przesyłania informacji o uwierzytelnieniu użytkownika i składają się z trzech części:

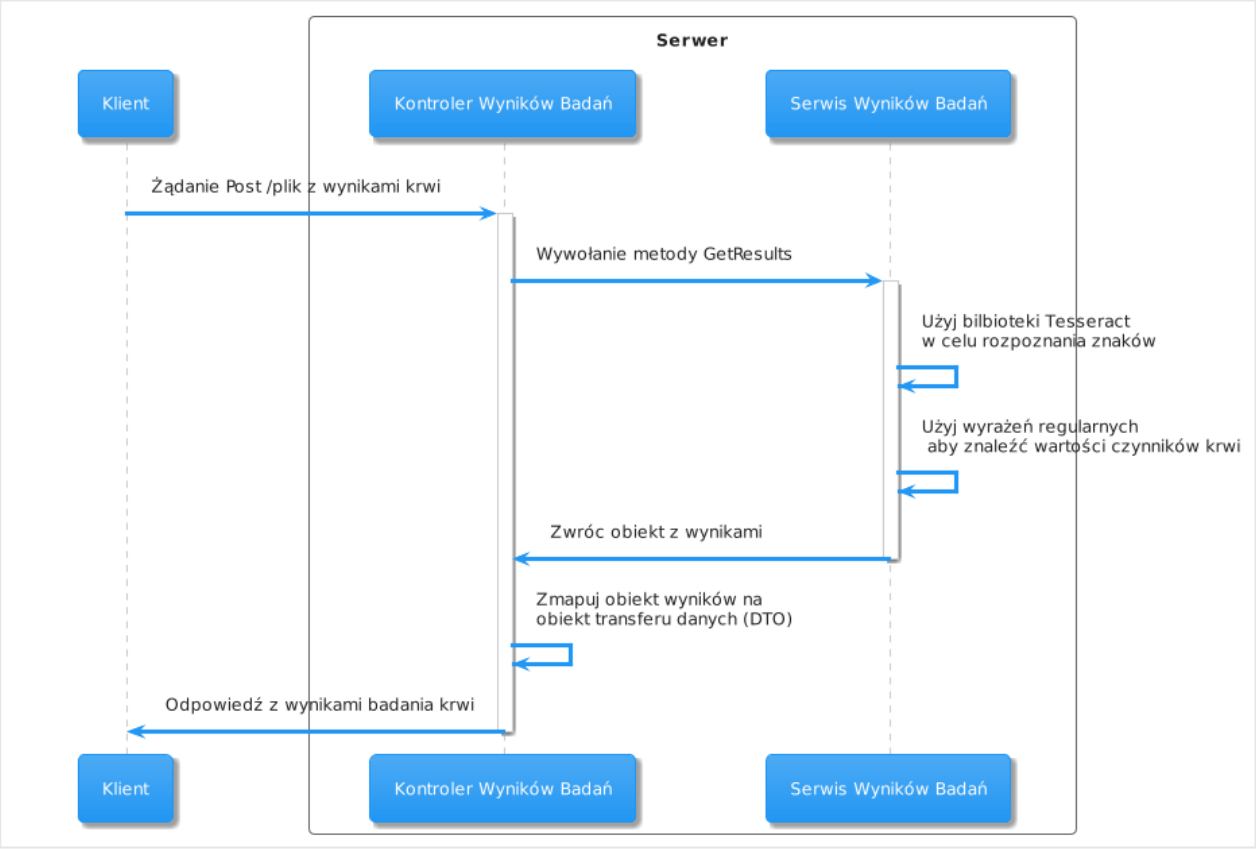
- nagłówku (header) - Zawiera on informacje o algorytmie szyfrowania używanym do podpisania tokena.
- zawartości (payload) - Zawiera on dane użytkownika, takie jak identyfikator, nazwa użytkownika, role, uprawnienia itp.
- podpisu (signature) - Jest to podpis cyfrowy, który zapewnia integralność i autentyczność tokenu. Jest generowany na podstawie nagłówka, zawartości i tajnego klucza.

Token jest szyfrowany z użyciem algorytmu HS256 (HMAC-SHA256), który jest jednym z dostępnych algorytmów szyfrowania symetrycznego tokenów

JWT. W aplikacji proces uwierzytelniania przedstawiony na **Rysunek 11** został zaimplementowany z użyciem modułu Spring Security 4.2.3, który umożliwia użycie pośrednika przed przetworzeniem żądania w celu weryfikacji tokenu.

4.2.13. Odczytywanie wyników badań krwi przy pomocy OCR

Aplikacja umożliwia również automatyczne odczytywanie wyników badań ze zdjęcia dostarczonego w formacie JPG, PNG lub dokumentu w formacie PDF. Klient po załadowaniu pliku w aplikacji klienckiej, wysyła żądanie z plikiem zakodowanym przy użyciu Base64. Serwer po otrzymaniu pliku w odpowiednim kontrolerze zleca serwisowi użycie biblioteki Tesseract, która rozpoznaje znaki w dokumencie i na podstawie tych znaków z użyciem wyrażeń regularnych odnajduje czynniki badań krwi. Następnie czynniki krwi zostają umieszczone w strukturze danych wyników badań i zostają zwrócone kontrolerowi. Kontroler następnie zamienia otrzymany obiekt na DTO (ang. Data Transfer Object), który to zwraca klientowi. Proces odczytywania wyników badań został przedstawiony na **Rysunek 12**.



Rysunek 12. Proces przetwarzania pliku z wynikami badań krwi

4.3. Aplikacja kliencka

Aplikacja kliencka została stworzona, aby umożliwić klientom wygodną interakcję z aplikacją, komunikację z serwerem i czytelną prezentację danych.

Jest ona aplikacja webową i została stworzona z użyciem biblioteki React.js, która służy do tworzenia interaktywnych interfejsów użytkownika.

4.3.1. JavaScript

Do stworzenia aplikacji klienckiej został wykorzystany język JavaScript, który jest szeroko wykorzystywany do tworzenia aplikacji webowych. Technologia ta została wykorzystana w projekcie ze względu na bogaty ekosystem bibliotek oraz narzędzi, takich jak: React, Bootstrap, czy Axios. Wszystkie z wymienionych bibliotek oferują gotowe komponenty, które zostały wielokrotnie wykorzystane podczas implementacji projektu w celu przyśpieszenia procesu tworzenia interfejsu użytkownika oraz zapewnienia jednolitego wyglądu aplikacji.

4.3.2. React.js

React jest popularną biblioteką języka JavaScript stworzoną przez firmę Facebook i wykorzystywaną do budowy interfejsów użytkownika o wysokiej wydajności i skalowalności [16]. Biblioteka React wprowadza koncepcję wirtualnego DOM (ang. Document Object Model), który jest efektywnym mechanizmem aktualizacji zawartości strony. Po zmianie jakiejkolwiek zawartości strony React porównuje wirtualny DOM (drzewiastą strukturę HTML strony) z rzeczywistą strukturą i dokonuje tylko niezbędnych zmian, przez co minimalizuje operacje na elementach strony przyśpieszając działanie aplikacji [17].

W projekcie biblioteka React została wykorzystana do optymalizacji szybkości działania aplikacji klienckiej, a także w celu wprowadzenia modułowości poprzez wykorzystanie komponentowej architektury. Dodatkowo wykorzystano bibliotekę React-bootstrap w celu poprawienia jakości interfejsu graficznego oraz dostarczenia responsywności aplikacji.

4.3.3. Bootstrap

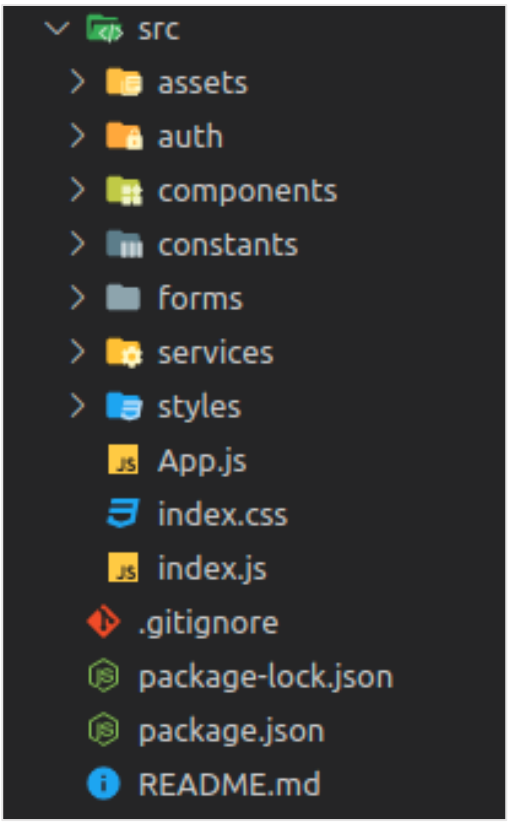
Bootstrap jest otwarto-źródłową biblioteką stworzoną przez firmę Twitter dostarczającą zestaw gotowych komponentów, stylów CSS (ang. Cascading Style Sheets), które mogą zostać dostosowane i wykorzystane przez programistę do tworzenia atrakcyjnych aplikacji internetowych [18]. Biblioteka została wykorzystana w projekcie, ponieważ dostarcza responsywne komponenty, które zapewniają elastyczność w dostosowywaniu wyglądu interfejsu do różnych rozmiarów ekranów. Dzięki temu zostały zaimplementowane elastyczne układy strony, które automatycznie dostosowują się do urządzeń mobilnych, tabletów i komputerów.

4.3.4. Visual Studio Code

Visual Studio Code to nowoczesny i lekki edytor kodu źródłowego, stworzony przez firmę Microsoft. Jego głównym celem jest ułatwienie pisania kodu poprzez różnorodne funkcje, takie jak automatyczne uzupełnianie, podświetlanie składni

i słów kluczowych związanych z danym językiem programowania. W projekcie, edytor został użyty do tworzenia aplikacji klienckiej. Środowisko to wyróżnia się spośród innych narzędzi tego typu dzięki cechom charakterystycznym dla bardziej zaawansowanych środowisk programistycznych. Przykładowo, umożliwia debugowanie kodu, posiada wsparcie dla systemu kontroli wersji GIT oraz oferuje szerokie możliwości rozbudowy poprzez instalację dodatkowych rozszerzeń [19].

4.3.5. Struktura projektu



Rysunek 13. Struktura aplikacji klienckiej

Aplikacja kliencka została zaprojektowana z użyciem struktury widocznej na **Rysunek 13**, która składa się z:

- assets – Zawiera domyślne obrazy wyświetlane np. w przypadku braku obrazu psa.
- auth – Składa się z funkcji odpowiedzialnych za dostarczenie bezpieczeństwa do aplikacji klienckiej, w tym obsługę tokenów JWT oraz wylogowanie użytkownika w przypadku nieważności tokenu.
- components – Zawiera komponenty, które odpowiedzialne są za wyświetlanie aplikacji.
- constants – Obejmuje stałe wykorzystywane w projekcie, takie jak nazwy ścieżek interfejsu programistycznego.
- forms – Zawiera komponenty odpowiedzialne za formularze edycji oraz dodawania. Stworzone w celu zapewnienia przejrzystości projektu.
- services – Składa się z funkcji wykonujących żądania http.

4.3.6. Komponenty funkcyjne

Komponenty funkcyjne są podstawowymi jednostkami budującymi interfejs użytkownika w bibliotece React.js. Służą one do deklaratywnego definiowania interfejsu, poprzez określanie, jak dany komponent ma wyglądać w zależności od przekazanych mu danych. Mogą reprezentować zarówno małe elementy interfejsu nadające się do wielokrotnego użycia, jak i bardziej skomplikowane sekcje aplikacji. Nie przechowują one stanu wewnętrznego ani nie posiadają metod cyklu życia. Jest to zasada tzw. programowania reaktywnego, która sprawia, że komponenty są bardziej przewidywalne i łatwiejsze do testowania.

Dodatkowo komponenty funkcyjne pozwalają na wykorzystanie mechanizmu Hooks. Jest to funkcjonalność wprowadzona do biblioteki React w wersji 16.8 i umożliwia ona korzystanie z funkcjonalności klasowych komponentów w komponentach funkcyjnych. Pozwalają one na przechowywanie stanu oraz obsługę efektów ubocznych. Dozwolone jest również definiowanie własnych funkcji hook, które mogą być wykorzystywane do specyficznych zadań jak na przykład wysyłanie zapytania POST wraz z sygnalizowaniem, czy odpowiedź została już dostarczona w postaci zmiennej isPending.

Z powodu możliwości wykorzystywania funkcji hook oraz ich reaktywności w projekcie aplikacji zostały wykorzystane wyłącznie komponenty funkcyjne z pominięciem ich innej, starszej wersji – komponentów klasowych. Przykładami funkcji hook, które zostały wykorzystane w projekcie są:

- useState() – pozwala na deklarację oraz aktualizację stanu w komponentach funkcyjnych. Jako argument przyjmuje wartość początkową stanu, a zwraca parę: wartość stanu oraz funkcję, która pozwala na zmianę wartości stanu. Każda zmiana wartości stanu powoduje ponowne renderowanie komponentu z nową wartością stanu.
- useEffect() – pozwala na obsługę efektów ubocznych w komponentach funkcyjnych. Przyjmuje dwa argumenty: funkcję efektu oraz tablicę zależności. Przy każdej zmianie wartości elementu w tablicy zależności wywoływana jest funkcja efektu. Dzięki temu możemy czyścić zasoby, bądź też walidować token JWT.
- useParams() – funkcja będąca częścią biblioteki react-router-dom, która umożliwia dostęp do parametrów URL w komponentach funkcyjnych. Gdy zdefiniowane są dynamiczne segmenty adresu (np. „/pets/:id”), pozwala wyodrębnić podane parametry wewnątrz komponentu i przypisać je do zmiennej [20].
- useNavigate() – funkcja będąca częścią biblioteki react-router-dom, która pozwala na nawigowanie między ścieżkami w aplikacji.

5. Działanie aplikacji

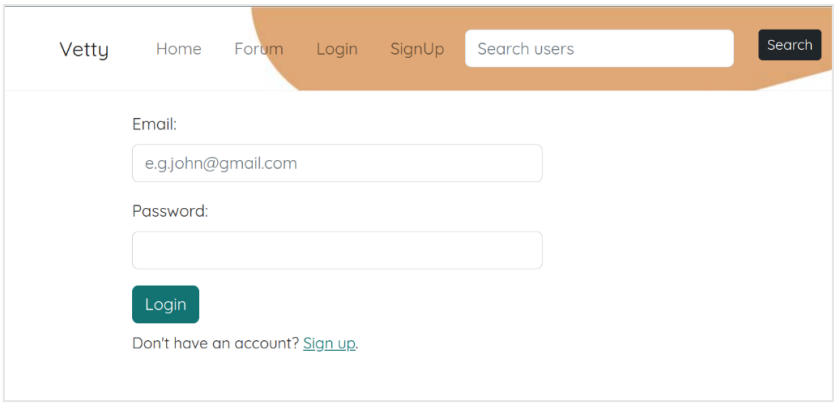
W rozdziale tym zostanie zaprezentowane działanie aplikacji Vetty. Przedstawione zostaną widoki dostępne w aplikacji webowej oraz dostarczona zostanie instrukcja obsługi funkcjonalności aplikacji.

5.1. Rejestracja oraz logowanie

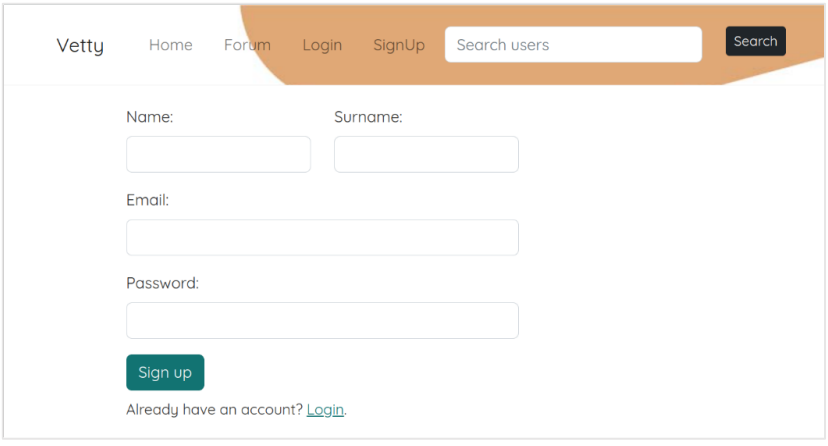
Po uruchomieniu aplikacji otworzone zostanie okno logowania przedstawione na **Rysunek 14**. W przypadku braku konta z okna logowania można bezpośrednio przejść do widoku rejestracji nowego użytkownika przedstawionego na **Rysunek 15** poprzez wciśnięcie przycisku **Sign up**. Aby utworzyć nowe konto, wymagane są następujące dane:

- imię użytkownika;
- nazwisko użytkownika;
- adres email użytkownika;
- hasło.

Następnie w celu dokończenia rejestracji wymagane jest wysłanie formularza poprzez wciśnięcie przycisku **Sign up**. Jeżeli podane są prawidłowe i w systemie nie istnieje użytkownik o podanym adresie email, konto zostanie założone, a my zostaniemy przeniesieni do głównej strony aplikacji.

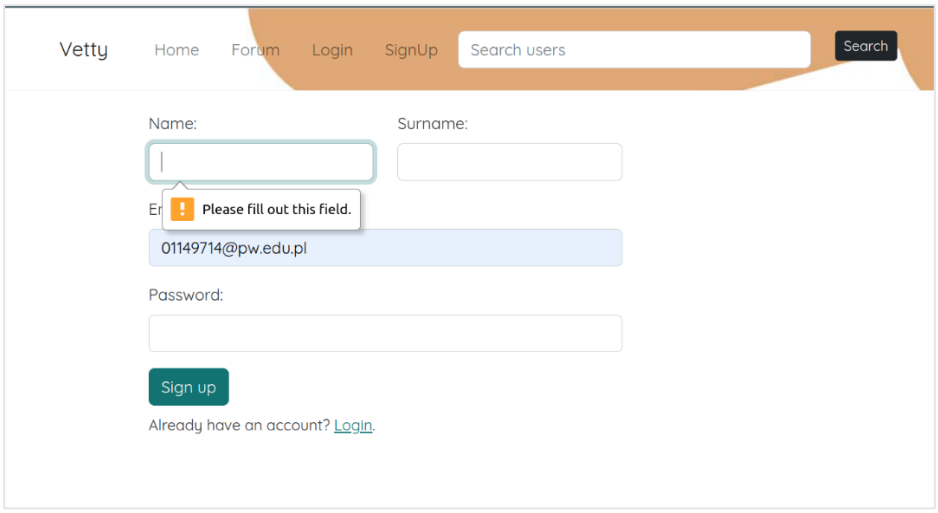


Rysunek 14. Okno logowania



Rysunek 15. Okno rejestracji

Dane zarówno w przypadku logowania jak i rejestracji poddawane są walidacji. W przypadku, gdy nie podamy wymaganej informacji i spróbujemy przetworzyć żądanie (przycisk Sign up) użytkownik zostanie poinformowany o brakujących danych co widać na **Rysunek 16**.

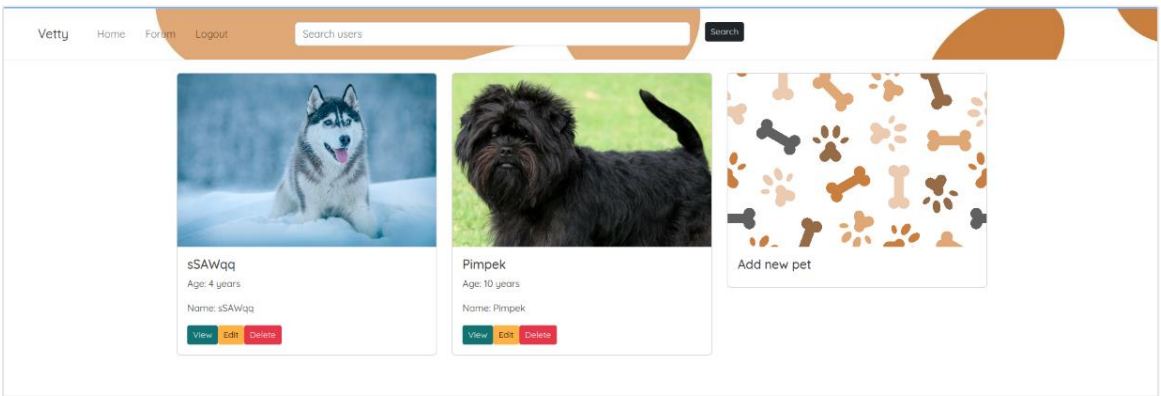


Rysunek 16. Okno rejestracji - walidacja danych

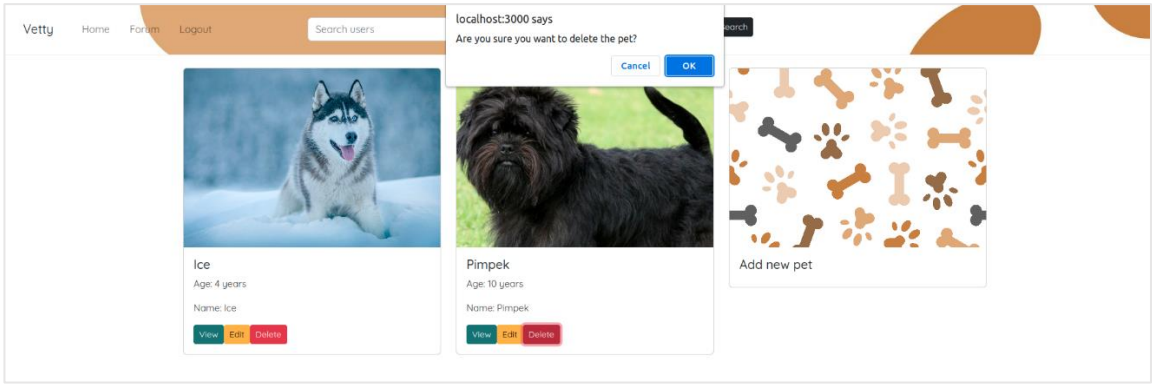
5.2. Główny ekran aplikacji

Po zalogowaniu, bądź rejestracji zakończonej sukcesem użytkownik zostaje skierowany do strony głównej aplikacji przedstawionej na **Rysunek 17**, która zawiera listę zwierząt, które użytkownik posiada oraz przycisk do dodawania nowych. W każdym małym panelu zwierzęcia znajdują się 3 przyciski:

- **view** – Po wciśnięciu kieruje użytkownika do widoku zwierzęcia.
- **modify** – Po wciśnięciu kieruje użytkownika do formularza modyfikacji danych zwierzęcia.
- **delete** – po wciśnięciu aplikacja wyświetli zapytanie, czy użytkownik jest pewny, że chcę usunąć zwierzę. Widok ten został przedstawiony na **Rysunek 18**. W przypadku zatwierdzenia zapytania, zwierzę zostanie usunięte.

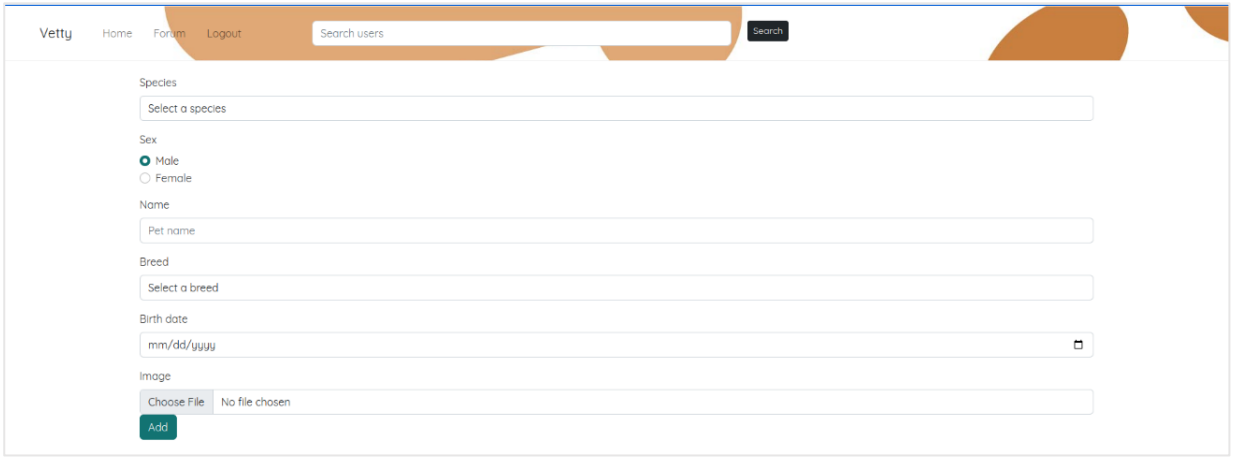


Rysunek 17. Okno główne - lista psów użytkownika

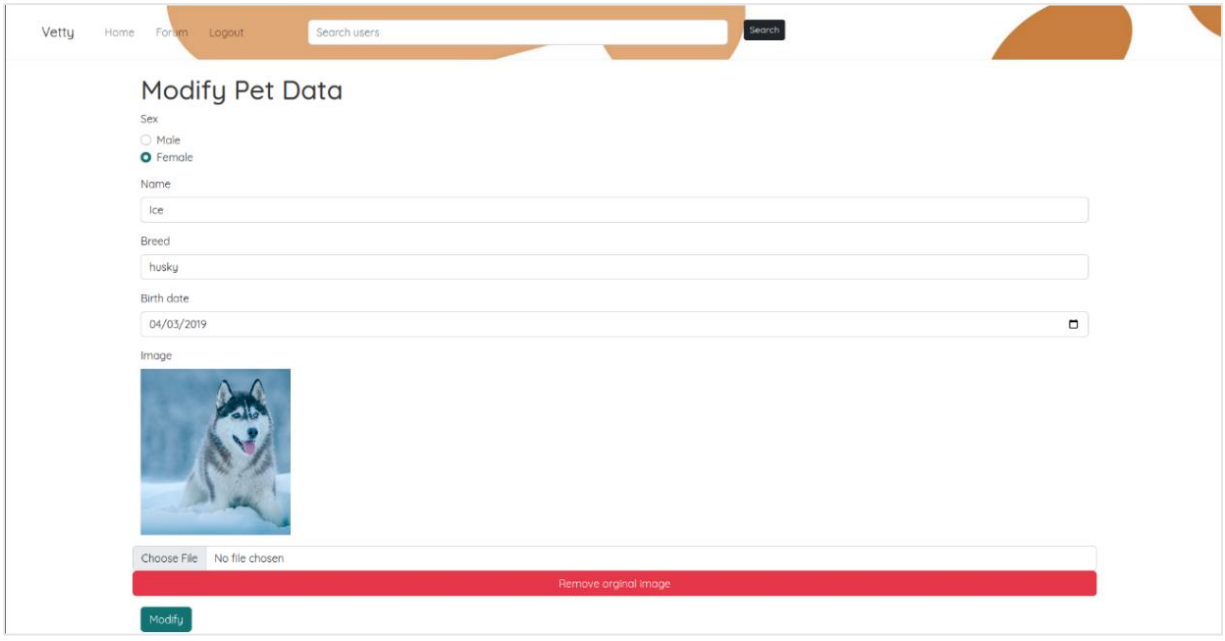


Rysunek 18. Okno główne - komunikat z potwierdzeniem usunięcia psa

5.3. Dodawanie i modyfikacja zwierząt



Rysunek 19. Okno dodawania zwierzęcia



Rysunek 20. Okno modyfikacji zwierzęcia

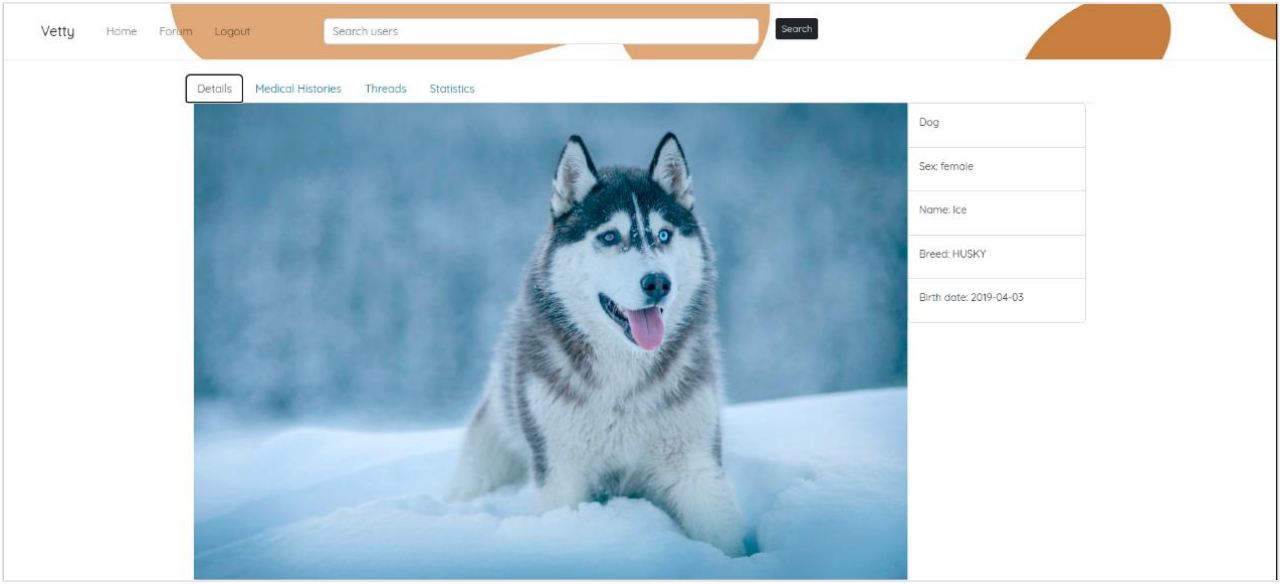
Po wciśnięciu przycisku dodania nowego zwierzęcia aplikacja przekieruje nas do formularza przedstawionego na **Rysunek 19**, który pozwoli nam dodać nowe zwierzę po wprowadzeniu informacji:

- gatunku zwierzęcia (pies, kot);
- płci zwierzęcia (samiec, samica);
- imienia zwierzęcia;
- rasy zwierzęcia;
- daty urodzenia zwierzęcia;
- (opcjonalnie) – zdjęcia zwierzęcia w formacie png, jpg;

W przypadku gdy wszystkie wymagane dane zostaną podane oraz formularz zostanie przetworzony bez błędów (operacja zakończy się sukcesem), użytkownik z powrotem zostanie przeniesiony do ekranu głównego aplikacji.

W przypadku wciśnięcia przycisku **Modify** użytkownik zostanie przekierowany do formularza modyfikacji zwierzęcia przedstawionego na **Rysunek 20**. Klient aplikacji ma możliwości usunięcia zdjęcia, bądź ustawienia nowego, a także modyfikacji każdej informacji z wykluczeniem gatunku zwierzęcia.

5.4. Panel zwierzęcia



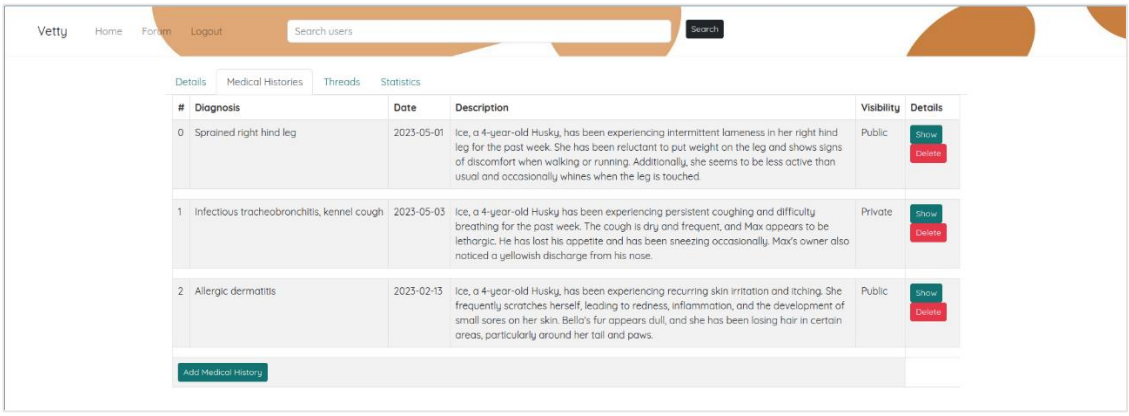
Rysunek 21. Panel zwierzęcia

Po wciśnięciu przycisku **View** w małym panelu zwierzęcia na stronie głównej strony użytkownik zostanie przeniesiony do panelu zwierzęcia przedstawionego na **Rysunek 21**. Z tego miejsca użytkownik może przejść do historii medycznej psa/kota poprzez wciśnięcie przycisku **Medical Histories**, wątków dotyczących tego zwierzęcia poprzez wciśnięcie przycisku **Threads** lub zobaczyć statystyki poszczególnych czynników krwi na tle swojej rasy poprzez wciśnięcie przycisku **Statistics**.

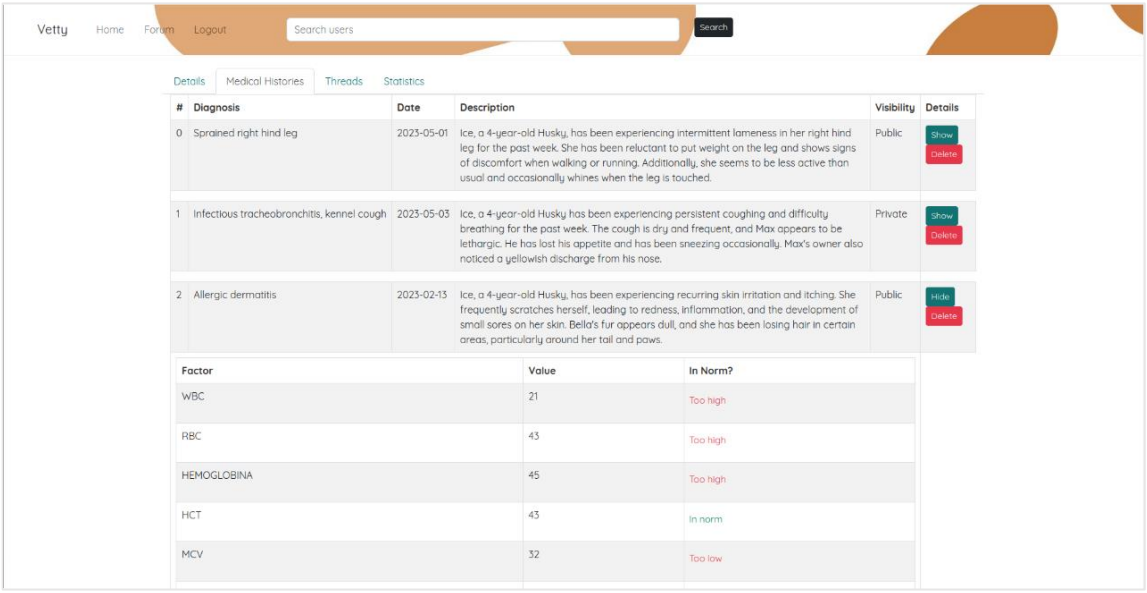
Na panelu zwierzęcia znajdują się jego dane takie jak:

- gatunek;
- płeć;
- imię;
- rasa;
- data urodzenia.

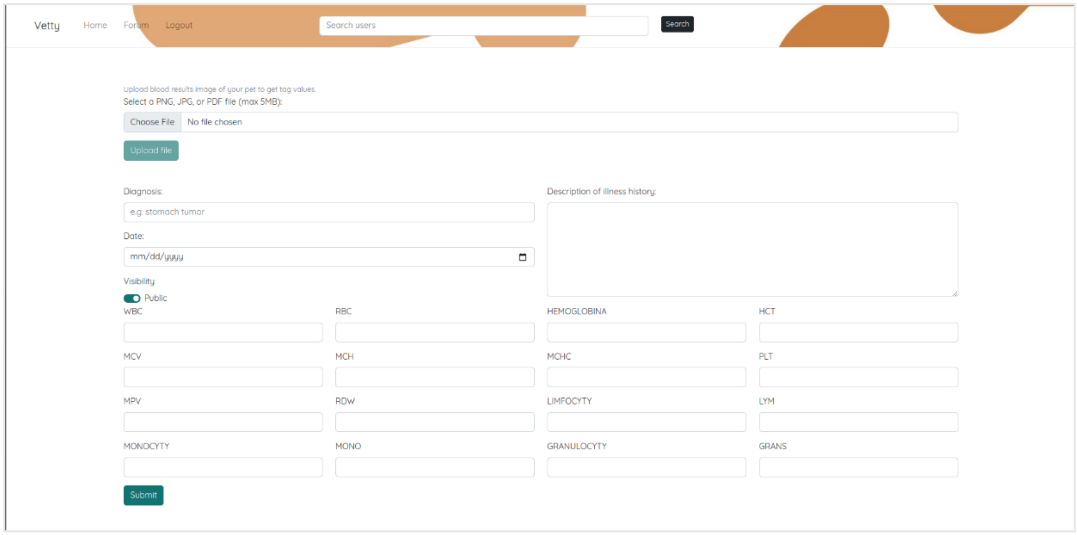
5.5. Panel historii medycznej zwierzęcia



Rysunek 22. Panel historii medycznej zwierzęcia



Rysunek 23. Panel historii medycznej zwierzęcia - wyniki badań



Rysunek 24. Panel dodawania nowej historii medycznej zwierzęcia

Panel historii medycznej psa/kota przedstawiony na **Rysunek 22** zawiera listę historii medycznych zwierzęcia które składają się z:

- diagnozy lekarza;
- daty rozpoczęcia choroby;
- opisu przebiegu choroby;
- flagi widoczności (definiującej, czy inni użytkownicy mogą zobaczyć historię tej choroby);
- panelu szczegółów (wyników badań) z przyciskiem **Show** pokazującym wyniki i **Delete** usuwającym historię medyczną.

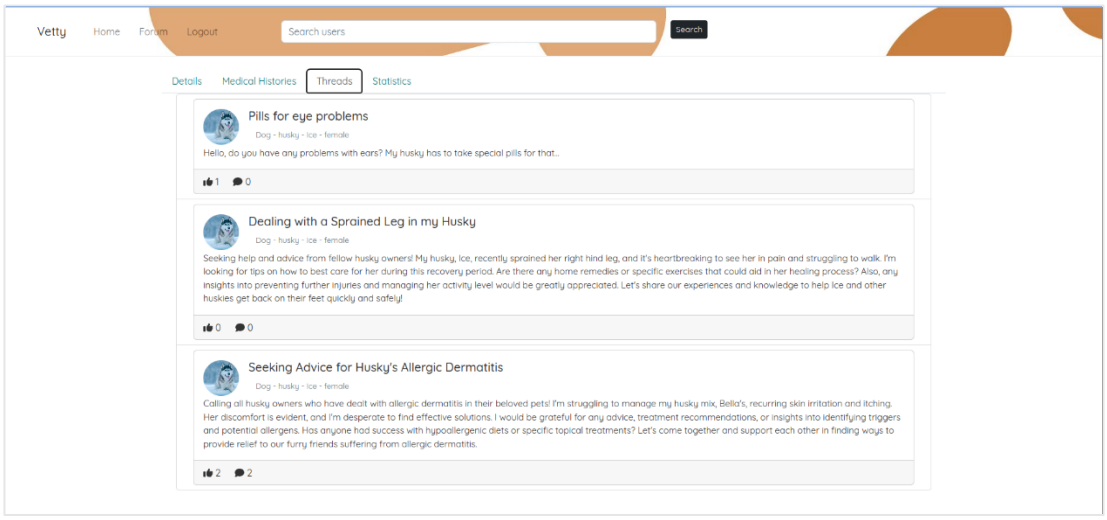
Na dole widoku znajduje się przycisk **Add Medical History**, który przenosi do formularza dodania historii medycznej widocznego na **Rysunek 24**. Użytkownik w przypadku istnienia wyników badań historii medycznej może rozwinąć wyniki poprzez wciśnięcie przycisku **Show**, co rozwinie panel wyników badań widoczny na **Rysunek 23**.

W przypadku dodawania nowej historii medycznej użytkownik musi podać:

- diagnozę;
- opis;
- datę rozpoczęcia choroby;
- flagę widoczności.

Dodatkowo właściciel zwierzęcia ma możliwość automatycznego wczytania wyników badań poprzez wczytanie pliku z wynikami badań w panelu na górze okna. Po odczycie wyników zostanie wyświetlony komunikat przez stronę i odpowiednie komórki wartości czynników badania krwi zostaną wypełnione odczytanymi wartościami. W przypadku, gdy aplikacja nie odczytała części wartości, użytkownik ma możliwość wprowadzenia brakujących wartości ręcznie.

5.6. Panel wątków zwierzęcia



Rysunek 25. Panel wątków dotyczących danego zwierzęcia

Okno wątków zwierzęcia widoczne na **Rysunek 25** pozwala w łatwy sposób znaleźć wątki stworzone przez właściciela dotyczące konkretnego psa/kota. Dzięki temu możemy szybciej przejść do widoku konkretnego wątku w celu przejrzenia, bądź dodania komentarza.

5.7. Panel statystyk zwierzęcia

Rysunek 26. Panel statystyk

Panel statystyk widoczny na **Rysunek 26** pozwala użytkownikowi zobaczyć jak zmieniały się wartości poszczególnych czynników krwi na przestrzeni czasu razem z porównaniem do średniej wartości dla rasy zwierzęcia. Użytkownik może wybrać czynnik krwi, którego statystykę chce zobaczyć. W skład dostępnych czynników wchodzą:

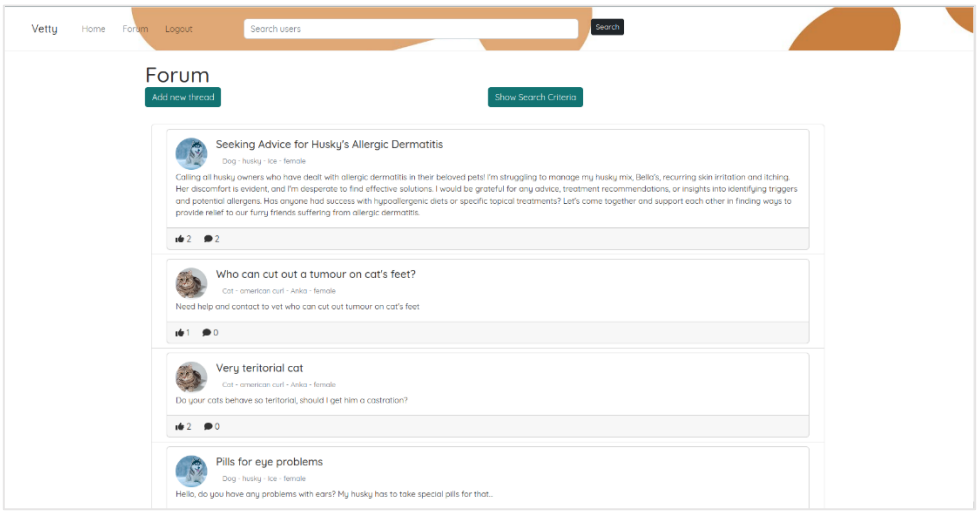
- **LYM** – Liczba limfocytów / procent (%) objętości.
- **MCV** (ang. Mean Corpuscular Volume) – Średnia objętość czerwonych krwinek. Jest to miara średniego rozmiaru czerwonych

krwinek w organizmie. MCV jest obliczane jako stosunek objętości czerwonych krwinek do ich liczby. Wyrażane jest w femtolitrach (fL).

- **RDW** (ang. Red Cell Distribution Width) – Szerokość rozkładu objętości czerwonych krwinek. Jest to miara zróżnicowania rozmiarów czerwonych krwinek. Wyrażana jest w procentach (%).
- **LIMFOCYTY** – Liczba limfocytów / liczba absolutna tysięcy na mikrolitr (K/µL).
- **MONO** – Liczba monocytów / procent (%) objętości.
- **MONOCYTY** – Liczba monocytów / liczba absolutna tysięcy na mikrolitr (K/µL).
- **GRANULOCYTY** - Liczba granulocytów, czyli liczba neutrofilów, eozynofilów i bazofilów / liczba absolutna tysięcy na mikrolitr (K/µL).
- **MPV** (ang. Mean Platelet Volume) – Średnia objętość płytek krwi / femtolitry (fL).
- **RBC** - Liczba erytrocytów (czerwone krwinki) / miliony na mikrolitr (M/µL).
- **MCH** (ang. Mean Corpuscular Hemoglobin) – Średnia masa hemoglobiny w jednej czerwonej krwince / pikogramy (pg).
- **GRANS** – Liczba granulocytów, czyli liczba neutrofilów, eozynofilów i bazofilów / procent (%) w stosunku do liczby leukocytów.
- **MCHC** (ang. Mean Corpuscular Hemoglobin Concetration) – Średnie stężenie hemoglobiny w jednej czerwonej krwince / gram na decylitr (g/dL).
- **HEMOGLOBINA** – Białko w czerwonych krwinkach transportujące tlen / gram hemoglobiny na decylitr krwi (g/dL).
- **HCT** – Hematokryt / procent (%) objętości.
- **PLT** – Liczba płytek krwi / tysiące na mikrolitr (K/µL).
- **WBC** – Liczba leukocytów (białe krwinki) / tysiące na mikrolitr (K/µL).

Wyniki pokazują potencjał zbieranych danych i choć obecnie możliwości statystyk są niewielkie, to w przyszłości aplikacja ma duże możliwości na rozwój w tym kierunku. Przykładem potencjalnych zmian może być opcja wybierania masy, wieku, płci oraz potencjalnych chorób psów na których tle chcemy porównywać wyniki.

5.8. Forum

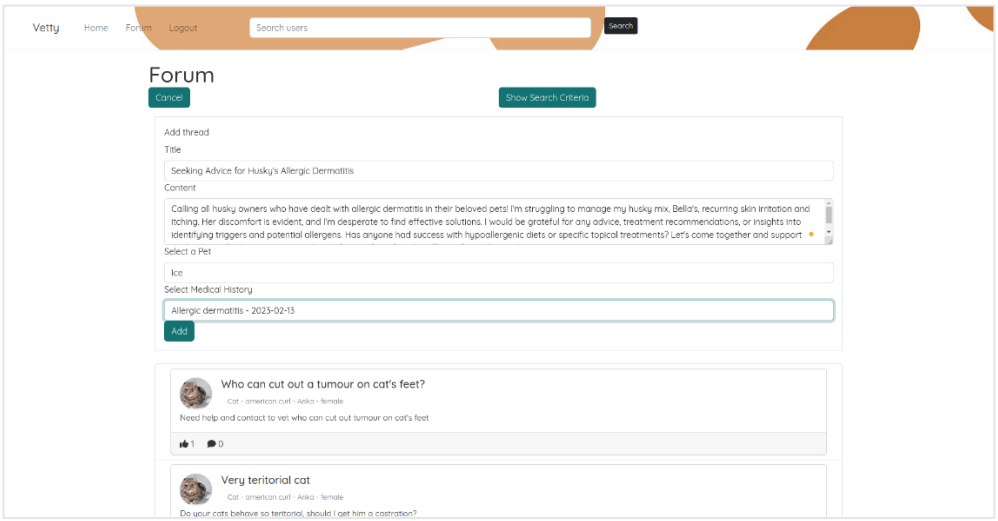


Rysunek 27. Panel forum

Do panelu forum przedstawionego na **Rysunek 27** użytkownik może przejść poprzez wciśnięcie przycisku **Forum** w panelu nawigacyjnym strony. Po przekierowaniu aplikacja udostępni użytkownikowi listę wszystkich wątków, gdzie każdy panel małego wątku zawiera:

- tytuł wątku;
- Dane zwierzęcia którego dotyczy wątek (miniatura zdjęcia, gatunek, rasa, imię, płeć);
- opis wątku;
- liczba polubień oraz komentarzy.

Dodatkowo użytkownik ma możliwość dodania nowego wątku poprzez wciśnięcie przycisku **Add new thread** oraz wybierania kryteriów wyszukiwania wątków poprzez wciśnięcie przycisku **Show search criteria**. Użytkownik może polubić każdy z wątków znajdujących się na forum oraz przenieść się do dedykowanego panelu wątku poprzez wciśnięcie okna z małym wątkiem.

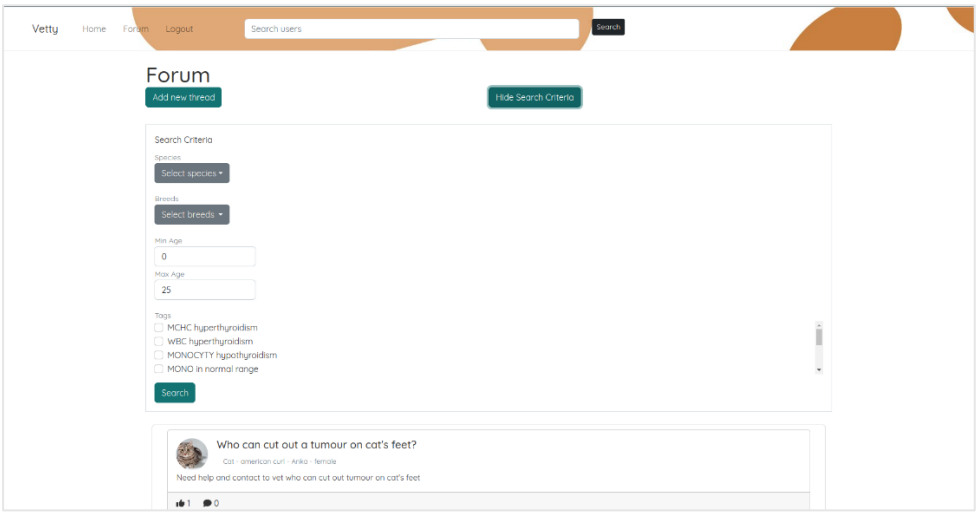


Rysunek 28. Okno dodawania wątku na forum

Po wciśnięciu przycisku służącego do dodawania nowego wątku pojawi się dedykowany panel widoczny na **Rysunek 28**. Użytkownik ma możliwość dodania wątku poprzez:

- Wpisanie tytułu wątku.
- Wpisanie opisu wątku.
- Wybranie zwierzęcia, którego dotyczy wątek.
- Historię medyczną zwierzęcia, które wybraliśmy (dostępne są tylko publiczne historie medyczne).

Po wprowadzeniu wszystkich danych użytkownik jest w stanie dodać nowy wątek poprzez wciśnięcie przycisku **Add**. Jeżeli wątek zostanie poprawnie dodany użytkownik będzie mógł zobaczyć dodany wątek na liście dostępnych.



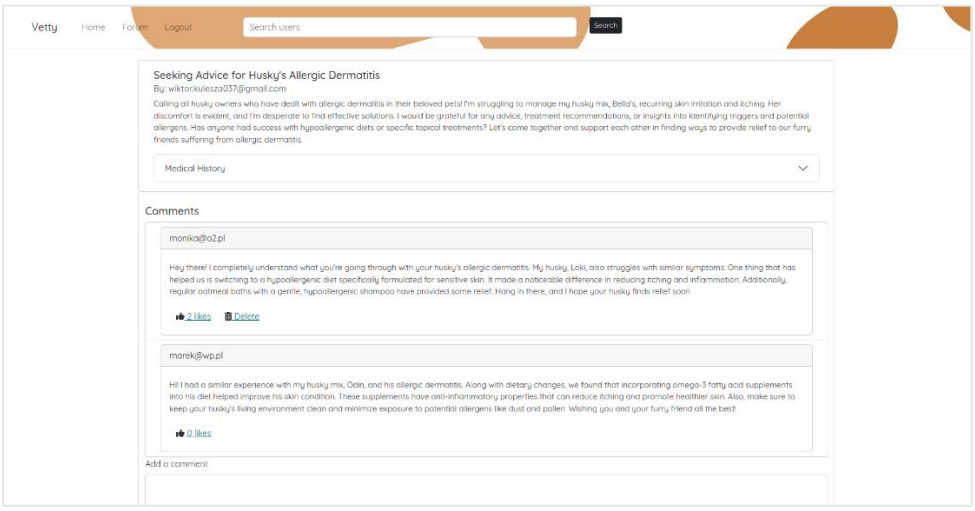
Rysunek 29. Okno wybierania kryteriów wyszukiwania wątków

Po wciśnięciu przycisku służącego do wyświetlenia kryteriów wyszukiwania wątków pojawi się dedykowany panel przedstawiony na **Rysunek 29**. Użytkownik ma możliwość wybrania kryteriów wyszukiwania spośród:

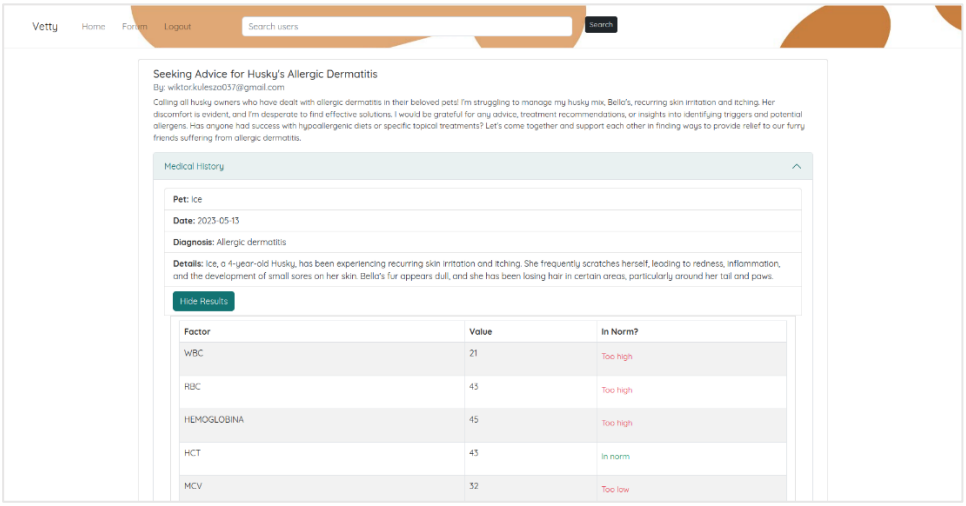
- Gatunku zwierzęcia, którego dotyczą wątki (pies/kot).
- Rasy zwierzęcia, którego dotyczą wątki (odpowiednio psich ras dla psów, kocich dla kotów oraz wszystkich w przypadku braku wybrania konkretnego gatunku).
- Minimalnego oraz maksymalnego wieku wyrażonego w latach (gdzie pies mający mniej niż 12 miesięcy ma 0 lat).
- Tagów, które generowane są automatycznie przy dodawaniu wyników badań krwi do historii choroby. Jeżeli wątek, którego podpięta historia choroby posiada wymagane tagi zostanie wyświetlona.

Po wprowadzeniu wszystkich danych użytkownik jest w stanie dodać nowy wątek poprzez wciśnięcie przycisku **Add**. Jeżeli wątek zostanie poprawnie dodany użytkownik będzie mógł zobaczyć dodany wątek na liście dostępnych.

5.9. Panel wątku



Rysunek 30. Okno wątku



Rysunek 31. Okno wątku – historia badań z wynikami

Po wciśnięciu małego panelu z wątkiem użytkownik zostanie przekierowany do widoku wątku, który został przedstawiony na **Rysunek 30**, gdzie może zobaczyć szczegóły tematu. W skład okna wchodzą:

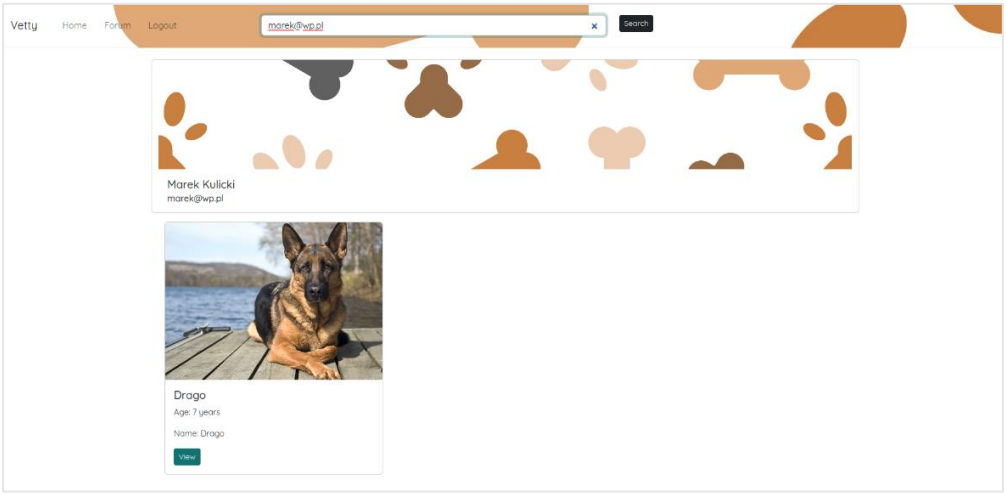
- temat wątku;
- adres email autora;
- opis wątku;
- historia medyczna, która zawiera:
  - imię zwierzęcia;
  - datę historii medycznej;
  - diagnozę;
  - opis choroby;
  - panel z wynikami badań;
- sekcja komentarzy.

Użytkownik ma możliwość zobaczenia szczegółów historii medycznej poprzez wciśnięcie przycisku **Medical History**. Po otworzeniu się panelu z historią medyczną użytkownik ma możliwość przejrzenia wyników badań przedstawionych na **Rysunek 31**, które składają się z listy czynników, ich wartości oraz wskaźników opisujących czy dana wartość jest w normie.

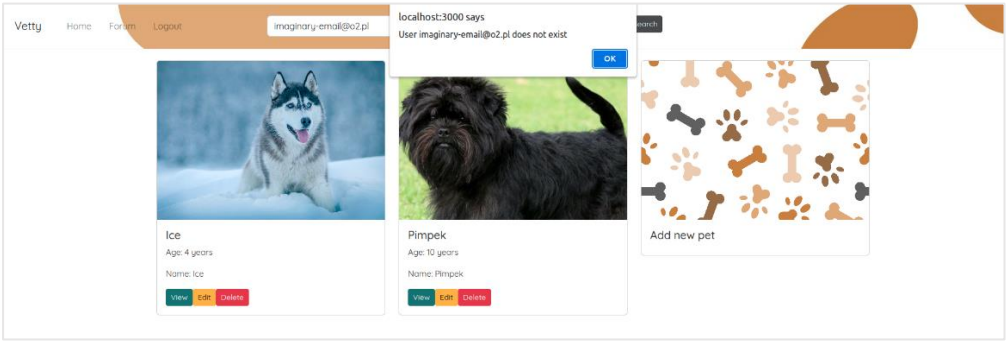
Dodatkowo okno wątku zawiera również sekcję komentarzy znajdującą się poniżej szczegółów wątku. Składa się on z: komentarzy innych użytkowników oraz z panelu do wprowadzenia własnego komentarza. Każdy z komentarzy można polubić poprzez wciśnięcie przycisku kciuka znajdującego się przy każdym komentarzu. W przypadku chęci anulowania polubienia, użytkownik powinien ponownie nacisnąć przycisk kciuka.

Wprowadzenie komentarza odbywa się poprzez wprowadzenie treści komentarza w polu tekstowym na dole stronie oraz wciśnięcie przycisku **Add**. Następnie komentarz jest dodawany wraz z adresem email autora do listy komentarzy.

5.10. Wyszukiwanie użytkowników



Rysunek 32. Widok wyniku wyszukiwania istniejącego użytkownika



Rysunek 33. Widok wyniku wyszukiwania nieistniejącego użytkownika

Aplikacja dostarcza również możliwość wyszukiwania innych użytkowników za pomocą ich adresu email poprzez panel wyszukiwania znajdujący się w sekcji

nawigacyjnej w górnej części aplikacji. Po wprowadzeniu adresu email użytkownik może wcisnąć przycisk **Search**. Następnie aplikacja sprawdzi, czy użytkownik o podanym adresie email istnieje. Jeżeli walidacja przebiegnie pomyślnie użytkownik zostanie przeniesiony do widoku użytkownika widocznego na **Rysunek 32**. W przeciwnym przypadku zostanie wyświetlony odpowiedni komunikat informujący o nieistniejącym profilu przedstawiony na **Rysunek 33**.

Widok profilu składa się z informacji o wyszukiwanym użytkowniku, w tym jego imienia, nazwiska oraz adresu email. Dodatkowo w skład panelu wchodzi lista zwierząt wyszukiwanego użytkownika, która w przeciwieństwie do ekranu głównego nie posiada przycisków do modyfikacji oraz usuwania zwierząt. Z panelu tego możemy przejść do widoku konkretnego zwierzęcia poprzez wciśnięcie przycisku **View**. Widok zwierzęcia, jak i jego historii medycznych oraz wyników badań będzie również ograniczony o możliwość edycji, usuwania, a także będziemy w stanie zobaczyć tylko publiczne historie chorób.

5.11. Analiza wydajności aplikacji

Aplikacja została przetestowana pod względem wymagań niefunkcjonalnych. Działanie aplikacji zostało sprawdzone manualnie w dwóch przeglądarkach internetowych opartych na silniku Chromium: Google Chrome oraz Operze. W obydwu przypadkach interfejs użytkownika był responsywny oraz wyświetlał się w przejrzysty sposób. Dodatkowo sprawdzony został średni czas potrzebny aplikacji serwerowej na przetworzenie żądania. W tym celu wykorzystano narzędzie Postman w którym utworzono kolekcję zapytań składających się z operacji wysyłanych podczas korzystania z aplikacji. Każde zapytanie zostało przetworzone dziesięciokrotnie w celu uśrednienia wyników. Ostatecznie średni czas potrzebny na przetworzenie zapytania wyniósł 1.87 sekundy, natomiast w przypadku gdy pod uwagę wzięte zostały tylko zapytania których przetworzenie nie wymagało użycia narzędzia Tesseract wartość ta była mniejsza i wyniosła średnio 1.35 sekundy na zapytanie.

6. Podsumowanie

Celem pracy było stworzenie aplikacji webowej umożliwiającej łatwe przechowywanie, przeglądanie i udostępnianie danych medycznych psów oraz kotów, a także interakcję z innymi użytkownikami za pomocą forum. Aplikacja znacząco ułatwia komunikację między właścicielem zwierzęcia, a weterynarzem w przypadku braku dostępu lekarza do historii medycznej zwierzęcia, a także umożliwia szukanie pomocy na jednym globalnym forum weterynaryjnym. Ponadto system ten poprzez utrzymanie struktury danych chorób oraz wyników pozwala na przyszłą analizę tych danych co może przyczynić się do rozwoju diagnostyki chorób psów i kotów.

6.1. Uzyskane rezultaty

Projekt spełnił postawione cele oraz zrealizował postawione wymagania funkcjonalne oraz niefunkcjonalnych w tym:

- dodawanie zwierząt;
- dodawanie rekordów medycznych do historii medycznych zwierząt;
- dodawanie wyników badań z możliwością automatycznego wczytywania wyników;
- dodawanie wątków na forum z możliwością komentowania;
- wyszukiwanie innych użytkowników oraz ich zwierząt;
- zarządzanie prywatnością tworzonych rekordów medycznych;
- wyświetlanie statystyk czynników krwi na tle rasy zwierzęcia;

Rozwiązanie zostało zaprojektowane z wykorzystaniem rozwiązań webowych takich jak Spring Boot, React.js oraz relacyjna baza danych z systemem zarządzania MySQL.

Platforma Spring Boot umożliwiła łatwiejszą implementację projektu poprzez zarządzanie zależnościami, dostarczenie wbudowanego serwera oraz automatyczną konfigurację co pozwoliło na skupienie się na logice biznesowej aplikacji.

Platforma React.js umożliwiła stworzenie prostego, wygodnego oraz interaktywnego interfejsu użytkownika. Wraz z użyciem biblioteki Bootstrap pozwoliła na stworzenie responsywnych widoków, które są czytelne oraz proste w obsłudze dla użytkowników niezależnie od wielkości ekranu oraz urządzenia końcowego. Modułowość projektów poprzez użycie komponentów pozwala na łatwy dalszy rozwój aplikacji.

MySQL jako relacyjna baza danych oferuje skalowalność oraz wydajność systemu dzięki któremu łatwo możemy rozwijać aplikację, a także łatwą analizę przechowanych danych, które mogą zostać wykorzystane w potencjalnej diagnostyce chorób psów i kotów.

6.2. Możliwe kierunki rozwoju

Przechowywanie danych medycznych zwierząt domowych jest rozległym tematem, który posiada możliwości analizy oraz wykorzystania ich do wymiany informacji pomiędzy właścicielami oraz lekarzami weterynarii. Dodatkowo

aplikacja webowa stworzona w ramach pracy dyplomowej posiada wiele możliwości rozwoju, a także udoskonaleń. Do możliwych kierunków rozwoju należą:

- Zwiększenie bezpieczeństwa aplikacji poprzez wykorzystanie szyfrowanego protokołu HTTP – HTTPS w komunikacji pomiędzy aplikacją kliencką, a serwerową. Pozwoliłoby to na bezpieczną wymianę danych oraz utrudnienie prób wykradnięcia haseł użytkowników.
- Dodanie dedykowanego forum dla weterynarzy wraz z dodaniem osobnego rodzaju profilu, który wyróżniałby się w komentarzach za pomocą specjalnej odznaki. Dzięki temu lekarze mieliby platformę do wymiany informacji o trudnych przypadkach medycznych w ścisłym gronie wykwalifikowanych osób. Dodatkowo pozostali użytkownicy widzieliby, które komentarze pochodzą od lekarzy i mieliby większą pewność co do prawdziwości przekazywanych informacji.
- Rozwinięcie przedstawianych statystyk o możliwość filtracji wieku, rasy oraz płci, a także dodanie zwiększonej ilości danych jak: mediana i pożądane wartości dla zdrowego zwierzęcia.
- Rozwinięcie analizy przechowywanych chorób poprzez wprowadzenie półautomatycznej diagnostyki (np. proponowanie diagnozy na podstawie przebytych chorób zwierzęcia oraz wprowadzonych badań krwi). Ułatwiłoby to korzystanie z aplikacji, a także miałoby ogromne zastosowanie w diagnostyce chorób jako asysta dla lekarzy weterynaryjnych.
- Zmienienie sposobu wskazywania, czy wartości czynników badania krwi są w normie. Aktualnie jest to rozwiązanie dość powierzchowne i nie bierze pod uwagi rozmiaru rasy oraz płci zwierzęcia przez co jest niedokładnym wskaźnikiem. Dokładniejsze wskaźniki zwiększyłyby wiarygodność aplikacji.
- Dodanie listy chorób do wyboru przy wybieraniu diagnozy historii choroby. Dzięki temu rekordy były lepiej ustrukturyzowane co pozwoliłoby na lepszą analizę rekordów oraz lepszą potencjalną diagnostykę chorób zwierząt.
- Usprawnienie algorytmu odczytywania wyników z pliku badań krwi co zwiększy jego dokładność oraz zmniejszy wymagany nakład pracy użytkownika potrzebny do wprowadzenia wyników badań.
- Dodanie możliwości wprowadzenia innych wyników badań, takich jak: badania moczu, zdjęcia rentgenowskie, tomografia komputerowa, rezonans magnetyczny czy badania EKG (elektrokardiografia). Dzięki temu użytkownicy oraz weterynarze będą mieli możliwość posiadania większej ilości informacji, co ułatwi diagnostykę oraz zwiększy możliwości analizy danych.

- Dodanie komunikatora, dzięki któremu moglibyśmy komunikować się bezpośrednio z innymi użytkownikami. Wpłynęłoby to pozytywnie na odczucia użytkowników podczas korzystania z aplikacji.
- Dodanie możliwości zmiany hasła oraz innych informacji profilowych, które przełożyłyby się na pozytywniejszy odbiór aplikacji.
- Rozwinięcie profilu zwierząt o dodatkowe dane takie jak masa, ubarwienie, cechy charakterystyczne, czy rodowód. Pozwoliłoby to na dokładniejszą diagnostykę oraz wpływałoby pozytywnie na jakość wartości wskazującej czy czynniki badań są w normie.

Bibliografia

[1] P. Woźnicki i J. Wilczyńska, „Koty i psy w Polsce”, *Koty i psy w Polsce oraz charakterystyka właścicieli i ich postaw względem zwierząt*. https://vetkompleksowo.pl/kategorie-tematyczne/koty-i-psy-w-polsce-oraz-charakterystyka-wlascicieli-i-ich-postaw-wzgledem-zwierzat/ (dostęp 14 maj 2023).

[2] „Aplikacja Internetowa”. https://pl.wikipedia.org/wiki/Aplikacja_internetowa (dostęp 15 maj 2023).

[3] „Client Server Architecture - CIO Wiki”. https://cio-wiki.org/wiki/Client_Server_Architecture (dostęp 15 maj 2023).

[4] „MVC - Glossary”. https://developer.mozilla.org/en-US/docs/Glossary/MVC (dostęp 21 maj 2023).

[5] „Java Spring Boot”. https://www.ibm.com/topics/java-spring-boot (dostęp 17 maj 2023).

[6] „Spring Security”, *Spring Security*. https://spring.io/projects/spring-security (dostęp 5 czerwiec 2023).

[7] „What is Maven?” https://maven.apache.org/what-is-maven.html (dostęp 24 maj 2023).

[8] „Tesseract (software)”. https://en.wikipedia.org/wiki/Tesseract_(software) (dostęp 31 maj 2023).

[9] „Postman - About”. https://www.postman.com/company/about-postman/ (dostęp 2 czerwiec 2023).

[10] „IntelliJ IDEA overview | IntelliJ IDEA”, *IntelliJ IDEA Help*. https://www.jetbrains.com/help/idea/discover-intellij-idea.html (dostęp 5 czerwiec 2023).

[11] „GIT - About Version Control”. https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control (dostęp 3 czerwiec 2023).

[12] „Architecture Patterns - Layered Architecture”. https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html (dostęp 15 maj 2023).

[13] „ModelMapper”. https://modelmapper.org/ (dostęp 3 czerwiec 2023).

[14] „What is REST”, *REST API Tutorial*, 7 kwiecień 2022. https://restfulapi.net/ (dostęp 5 czerwiec 2023).

[15] S. Raina, „Symmetric vs Asymmetric JWTs”, *Medium*, 27 marzec 2019. https://medium.com/@swayamraina/symmetric-vs-asymmetric-jwts-bd5d1a9567f6 (dostęp 5 czerwiec 2023).

[16] „React.js”. https://react.dev/ (dostęp 29 maj 2023).

[17]„What is DOM in React?” https://www.javatpoint.com/what-is-dom-in-react

[18]„React Bootstrap”. https://react-bootstrap.github.io/ (dostęp 26 maj 2023).

[19]„Visual Studio Code - Overview”. https://code.visualstudio.com/docs (dostęp 2 czerwiec 2023).

[20]„React Router - Concepts”. https://reactrouter.com/en/main/start/concepts (dostęp 30 maj 2023).

Wykaz symboli i skrótów

JSON – ang. JavaScript Object Notation
MVC – ang. Model-View-Controller
ORM – ang. Object-Relational Mapping
API – ang. Application Programming Interface
HTTP – ang. Hypertext Transfer Protocol
URL – ang. Uniform Resource Locator
JDBC – ang. Java DataBase Connectivity
SQL – ang. Structured Query Language
CSRF – ang. Cross-Site Request Forgery
XSS – ang. Cross-site Scripting
OCR – ang. Optical Character Recognition
DTO – ang. Data Transfer Object
REST – ang. Representational State Transfer
DOM – ang. Document Object Model
CSS – ang. Cascading Style Sheets

Spis rysunków