



UNIWERSYTET OPOLSKI

WYDZIAŁ MATEMATYKI, FIZYKI I INFORMATYKI

INSTYTUT INFORMATYKI

PRACA INŻYNIERSKA

Natalia Szymczak

Aplikacja bazodanowa dla klubów jeździeckich

Praca wykonana pod kierunkiem

dra Jacka Iwańskiego

OPOLE 2024

Streszczenie:

Abstract:

Keywords:

Klasyfikacja tematyczna wg MSC 2020:

Spis treści

1	Wstęp	1
2	Przegląd istniejących rozwiązań	2
2.1	Ridely	2
2.2	Equilab	3
2.3	Happie Horse	4
3	Technologie użyte w pracy	5
3.1	Microsoft Visual Studio 2022	5
3.2	Microsoft SQL Server 2019 Express	5
3.3	Microsoft SQL Server Management Studio	5
3.4	Structured Query Language	5
3.5	Windows Presentation Foundation	6
3.6	Xamarin	6
3.6.1	Xamarin.Android	7
3.7	Android Device Menager	8
3.8	NuGet	8
3.9	Git, Github, Sourcetree	9
3.10	Figma	10
3.11	Entity Framework	11
3.12	LiveChartsCore	12
3.13	ZXing	12
3.14	MVVM Toolkit	12
3.15	Xamarin Community Toolkit	12
3.16	ZXing.Net.Mobile.Forms	12

3.17	Xamarin.Essentials	12
3.18	Microsoft.Extensions.DependencyInjection	12
4	Specyfikacja wymagań	13
4.1	Opis wycinka rzeczywistości	13
4.2	Wymagania funkcjonalne	14
4.3	Wymagania niefunkcjonalne	21
5	Baza danych	25
5.1	Model konceptualny	25
5.2	Model logiczny	45
5.3	Model fizyczny	64
6	Projekt systemu	66
6.1	Model projektowanego systemu	66
6.1.1	Architektura aplikacji	66
6.1.2	Diagramy UML	68
6.2	Wybrane aspekty implementacyjne	70
7	Dokumentacja użytkownika	84
7.1	Aplikacja desktopowa	84
7.2	Aplikacja mobilna	96
8	Podsumowanie	105

Rozdział 1

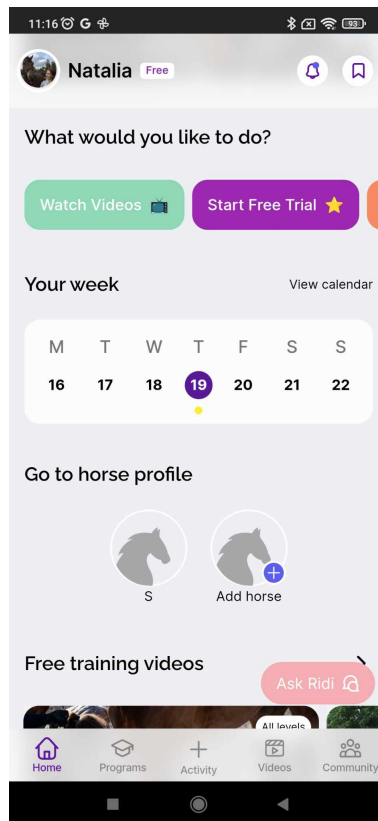
Wstęp

Rozdział 2

Przegląd istniejących rozwiązań

2.1 Ridely

Ridely to aplikacja pozwalająca na monitorowanie treningów. Dzięki niej możemy zapisać swoje treningi, oglądać poradniki i poprawiać swoje umiejętności jeździeckie. Aplikacja ma wiele mocnych stron. Skupia się głównie na monitorowaniu treningów i ten aspekt ma dopracowany bardzo dobrze. O samych aktywnościach można zbierać bardzo dużo informacji, jak także śledzić trening na bieżąco. Jednakże ma także wady. Kalendarz, w którym zaznaczone są aktywności jest mało czytelny i na pierwszy rzut oka nie widzimy co koń robił w poprzednie dni (patrz. rys. 2.1). Z zalet moż-



Rysunek 2.1: Widok główny aplikacji

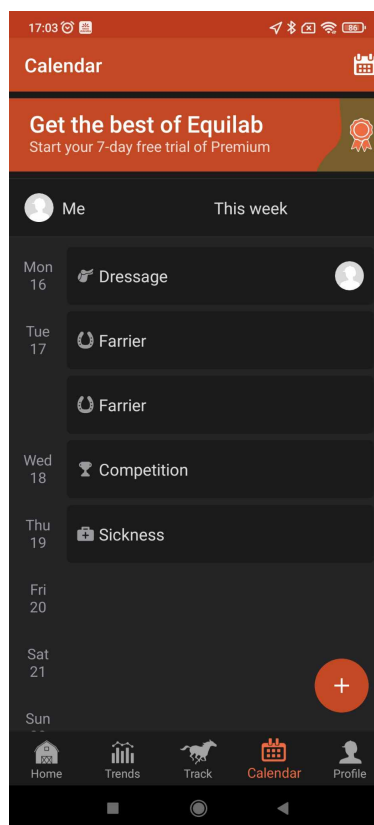
Źródło opracowanie własne

na wymienić jeszcze dostępność wielu materiałów wideo pozwalających na doszkolenie się jednakże materiały te są jedynie w języku angielskim. Porównując do stworzonej aplikacji nie ma możliwości zapisu wizyt lekarzy, planów żywienia ani nie ma informacji o zawodach. Jest ona mniej kompleksowa i przez dużą zawartość plików jest dość spora i wolno działa.

2.2 Equilab

Equilab także skupia się głównie na treningach. Nie ma na nim dostępnych materiałów wideo ani pouczających wpisów. Interfejs graficzny jest w miarę przyjazdy użytkownikowi, a sama aplikacja działa płynnie. Dzięki tej aplikacji można śledzić treningi i zapisywać ich trasy. Zapisywanie aktywności obejmuje także podstawowe wizyty lekarskie oraz kowali. Jednakże nie można o nich zapisać wielu informacji. Z minusów aplikacji można także wymienić mało przyjazdy UI w kalendarzu z zapisanymi aktywnościami. W porównaniu do stworzonej aplikacji nie ma możliwości za-

pisu planów żywienia. Minusem jest także połączenie w jednym widoku aktywności oraz wizyt przez co mamy mniej widocznych informacji i łatwiej się wśród nich zgubić (patrz rys. 2.2). W odróżnieniu od naszej aplikacji można tutaj zawierać przyjaźnie między użytkownikami i brać udział w wyzwaniach. Jednakże nie ma kluczowej funkcji udostępniania koni, która po rozmowach z osobami ze środowiska okazała się kluczowa.

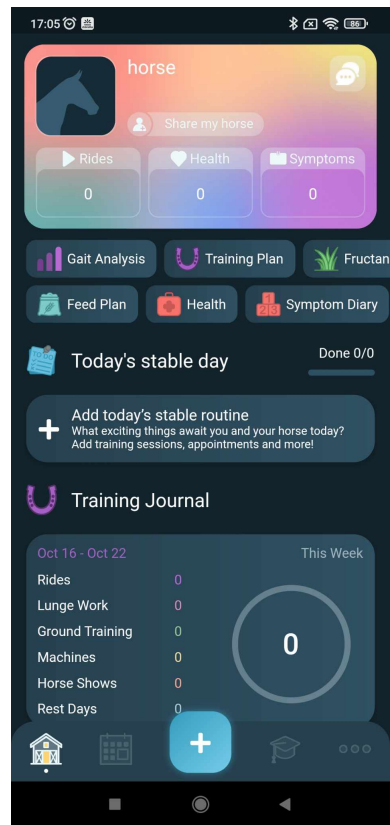


Rysunek 2.2: Widok kalendarza aplikacji

Źródło opracowanie własne

2.3 Happie Horse

Aplikacja Happie Horse posiada podobne funkcje do zaprojektowanego systemu. Na głównej stronie aplikacji mamy aktywności plan żywienia oraz wizyty u kowali i lekarzy. Aplikacja ma przyjazny choć dość ciemny interfejs. Zapis aktywności podobnie jak w poprzednich aplikacjach jest mało czytelny i na pierwszy rzut oka nie widać co koń robił przez ostatnie dni. Nie dostępne są także statystyki aktywności. W zamian za to aplikacja ma szereg kursów pozwalających na poprawę swoich umiejętności. To czego brakuje tej aplikacji to także funkcja udostępniania koni pomagająca kontrolować swoje konie w trakcie wyjazdów.



Rysunek 2.3: Widok kalendarza aplikacji

Źródło opracowanie własne

Rozdział 3

Technologie użyte w pracy

3.1 Microsoft Visual Studio 2022

Microsoft Visual Studio to środowisko IDE, za pomocą którego można edytować, debugować jak także kompilować kod. Po stworzeniu aplikacji można ją także opublikować w prost ze środowiska. Środowisko to zawiera wiele funkcji wzbogacających proces tworzenia takich jak narzędzia uzupełniania kodu (Intellisense). Dzięki temu środowisku możemy programować aplikacje na dowolną platformę oraz dowolne urządzenia.

3.2 Microsoft SQL Server 2019 Express

Microsoft SQL Server jest to system, wspomagający zarządzanie bazą danych stworzony oraz utrzymywany przez firmę Microsoft. MS SQL wykorzystuje język zapytań Transact-SQL, który jest rozwinięciem standardu języka zapytań ANSI/SQL.

3.3 Microsoft SQL Server Management Studio

3.4 Structured Query Language

SQL czyli Structured Query Language jest to język zapytań wykorzystywany w relacyjnych bazach danych. Umożliwia on tworzenie, modyfikowania oraz zarządzanie bazami danych. Dodatkowo dzięki SQL jesteśmy w stanie pobierać, dodawać, aktualizować oraz usuwać dane znajdujące się w naszej bazie danych. SQL wspiera również tworzenie skomplikowanych

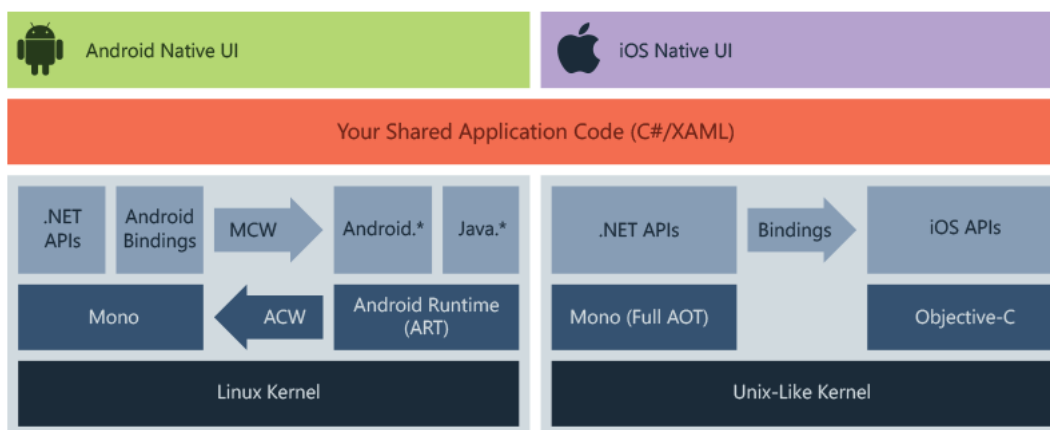
zapytań, dzięki czemu możemy wykonywać różne operacje na danych takie jak: filtrowanie, sortowanie, grupowanie oraz łączenie.

3.5 Windows Presentation Foundation

WPF - Windows Presentation Foundation, jest technologią opracowaną przez Microsoft. Służy ona do tworzenia aplikacji desktopowych na system Windows. Jest częścią .NET Framework i zapewnia on możliwość tworzenia zaawansowanych interfejsów użytkownika wykorzystując język XAML. Interfejs ten jest niezależny od rozdzielczości oraz używa aparatu renderowania opartego na wektorach, aby korzystać z nowoczesnego sprzętu graficznego. WPF dostarcza kontrolki, powiązanie danych, układ, grafiki 2D i 3D, animację, style, szablony, dokumenty, multimedia, tekst i typografię, jak także inne elementy interfejsu API platformy .NET. [8]

3.6 Xamarin

Xamarin jest to platforma do tworzenia aplikacji mobilnych za pomocą platformy .NET, która automatycznie obsługuje odzyskiwanie i alokowanie pamięci, jak także współdziałanie z platformami bazowymi. Dzięki Xamarinowi możemy pisać aplikacje na androida, iOS jak także na windows phone. Tworzy on warstwę abstrakcji komunikującą się za pomocą kodem aplikacji, a kodem bazowej platformy. Aplikacje wykorzystujące Xamarin możemy pisać nie tylko na komputerach PC z systemem Windows lub Linux, lecz także na urządzeniach z systemem MacOS. Architektura systemu Xamarin przedstawiona została na rysunku 3.1. Możemy na nim zobaczyć część architektury dotyczącą platformy android jak także, część dotyczącą platformy iOS.[2]

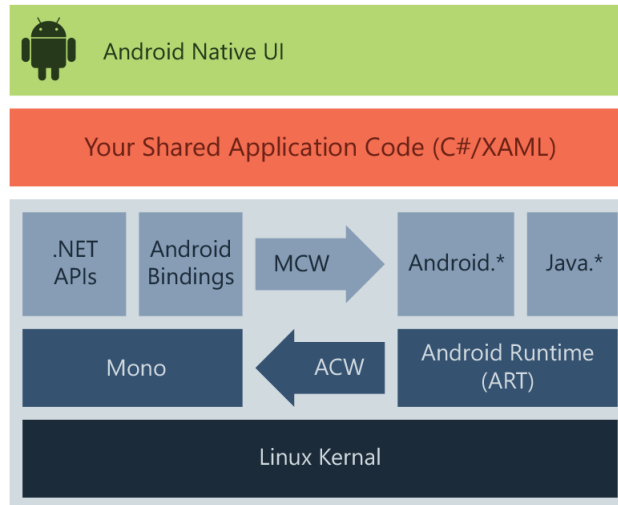


Rysunek 3.1: Architektura platformy Xamarin

Źródło: [2]

3.6.1 Xamarin.Android

Aplikacja HorseTracking dostępna będzie jedynie na platformę android. Aby dostosować ją do systemu iOS niezbędne było by urządzenie z systemem MacOS co nie było możliwe. Xamarin.Android kompilowany jest z języka C#, do języka pośredniego "just in time", często nazywanego JIT od pierwszych liter nazwy. JIT jest skompilowany do zestawu natywnego po uruchomieniu aplikacji. Xamarin.Android jest uruchamiany w środowisku mono obok maszyny wirtualnej środowiska Android Runtime. Dzięki platformie Xamarin możemy powiązać platformę .Net z przestrzeniami nazw Android.* i Java.*. Za pośrednictwem zarządzanych otok wywoływanych MCW środowisko mono może wywoływać przestrzenie nazw oraz udostępniać otoki wywoływane przez system Android(ACW). Dzięki temu oba środowiska mogą wywoływać kod nawzajem. Na rysunku 3.2 przedstawiona została architektura systemu Xamarin.Android[2].



Rysunek 3.2: Architektura platformy Xamarin.Android

Źródło: [2]

3.7 Android Device Menager

Dopisać

3.8 NuGet

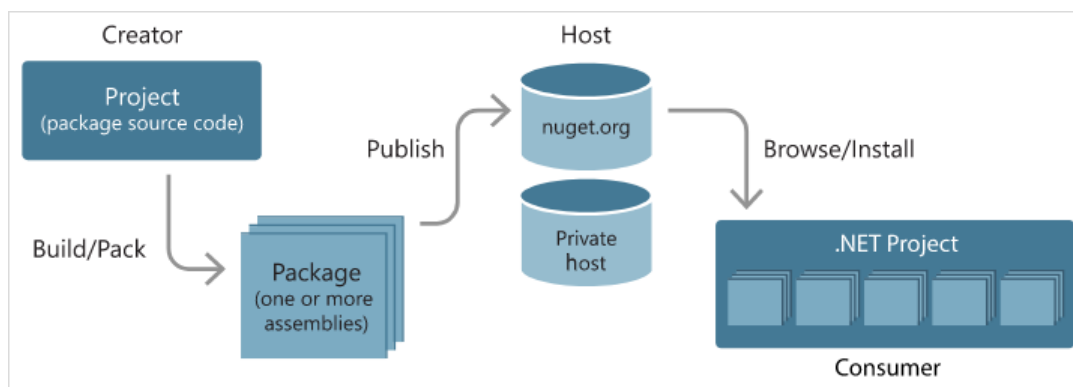
NuGet (logo systemu na rysunku 3.3) jest to mechanizm udostępniania kodu obsługiwany przez firmę Microsoft.



Rysunek 3.3: Logo systemu NuGet

Źródło: [3]

Służy on do współdzielenia kodu. Pakiet NuGet obsługuje hosty prywatne oraz publicznego hosta. Na hoście publicznym NuGet ma tysiące unikatowych pakietów dostępnych dla użytkowników .Net. Niezależnie od tego czy host jest prywatny czy publiczny jest on połączeniem między twórcami pakietów a ich konsumentami. Przepływ informacji między deweloperami pakietów, a ich konsumentami możemy zaobserwować na rysunku 3.4.



Rysunek 3.4: Przepływ informacji

Źródło: [3]

3.9 Git, Github, Sourcetree

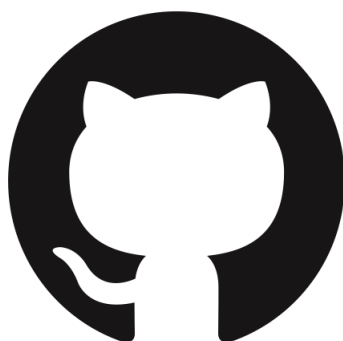
Git (logo na rysunku 3.5) to darmowy rozproszony system kontroli wersji typu open source. Zaprojektowany został do obsługi małych oraz bardzo dużych projektów. Dzięki niemu możliwe jest tworzenie backupów oraz zarządzanie nimi. Jak także rozwijanie aplikacji w wiele osób jednocześnie.[10]



Rysunek 3.5: Logo git

Źródło: [10]

Github (logo na rysunku 3.6) jest to nakładka na git, dzięki której w łatwiejszy sposób możemy zakładać repozytorium oraz zarządzać kodem.[11]



Rysunek 3.6: Logo github

Źródło: [11]

Sourcetree (logo na rysunku 3.7) to także nakładka na git, która pozwala na łatwiejsze tworzenie branchy, stashów, oraz szybkie przełączaniem się między gałęziami. Dzięki niemu możemy zarządzać branchami z wielu projektów na raz w sposób przyjaźniejszy niż przez Visual Studio.[12]



Rysunek 3.7: Logo sourcetree

Źródło: [12]

3.10 Figma

Figma jest to narzędzie do projektowania i prototypowania interfejsów aplikacji. Dzięki narzędziom takim jak figma możemy zaplanować cały interfejs jeszcze przed jego implementacją. Stworzony w ten sposób interfejs możemy przetestować dzięki funkcji prototypowania.

Jeszcze przed implementacją może on zostać udostępniony kilku osobą w celu sprawdzenia czy wszystkie funkcje aplikacji są dla użytkownika jasne i intuicyjne. Dzięki temu implementować będziemy interfejs już sprawdzony, więc będzie wymagał on mniej poprawek.

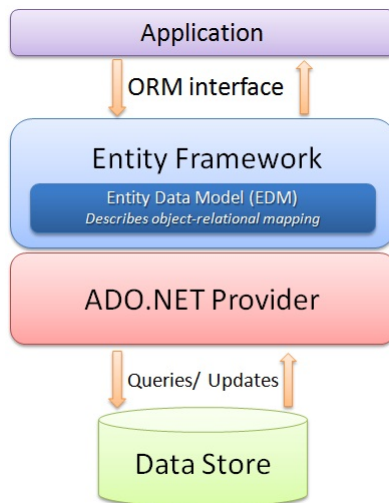


Rysunek 3.8: Logo Figmy

Źródło: [7]

3.11 Entity Framework

Entity Framework to narzędzie do mapowania obiektowo-relacyjnego, który umożliwia tworzenie przejrzystej, przenośnej i wysokopoziomowej warstwy dostępu do danych za pomocą platformy .NET (C#) dla wielu baz danych. Pozwala on na wykonywanie podstawowych operacji takich jak dodawanie, pobieranie, uaktualnianie i usuwanie danych. Możemy dzięki niemu także łatwiej zarządzać relacjami w bazie. Zasada działania tego narzędzia przedstawiona została na rysunku 3.9



Rysunek 3.9: przepływ informacji

Źródło: [4]

3.12 LiveChartsCore

LiveChartsCore to nugget pozwalający na łatwe i szybkie tworzenie wykresów.

3.13 ZXing

3.14 MVVM Toolkit

3.15 Xamarin Community Toolkit

3.16 ZXing.Net.Mobile.Forms

3.17 Xamarin.Essentials

3.18 Microsoft.Extensions.DependencyInjection

Rozdział 4

Specyfikacja wymagań

4.1 Opis wycinka rzeczywistości

Aplikacja przeznaczona jest dla klubów jeździeckich, czyli organizacji zrzeszających jeźdźców startujących w danej dziedzinie sportów konnych. Aplikacja skierowana jest do klubów, których zawodnicy startują w takich dziedzinach jak:

- Skoki przez przeszkody,
- WKKW (skrót od „Wszechstronny konkurs konia wierzchowego”),
- Ujeżdżenie.

W celu jak najlepszego określenia wymagań funkcjonalnych, przed napisaniem aplikacji przeprowadzono rozmowy z kilkoma osobami zaangażowanymi w to środowisko: pracownikami stadnin państwowych, właścicielami klubów, jak także z osobami prywatnie trzymającymi konie w stadninach. Po przeprowadzonych rozmowach zdecydowano się na dwie wersje aplikacji: desktopową oraz mobilną, które będą różnić się funkcjonalnościami.

Aplikacja ma na celu pomóc w gromadzeniu informacji o jeźdźcach przynależących do klubu oraz ich koniach. W aplikacji gromadzone są informacje o codziennych aktywnościach koni, ich chorobach, żywieniu oraz zawodach w których biorą udział. Naturalnie chcemy także zapisywać wyniki z tych zawodów, aby móc określić czy dany trening jest skuteczny. Z aplikacji będą korzystać zawodnicy, trenerzy, jacy i zarząd klubu.

Aby skutecznie zbierać informacje o treningach i innych aktywnościach niezbędna jest aplikacja mobilna, ponieważ dane te muszą być wprowadzane na bieżąco. Informacje o wizytach różnorodnych lekarz oraz kowala także muszą być zapisywane na bieżąco podczas danej

wizyty. Dlatego funkcjonalności te dotyczą jedynie aplikacji mobilnej. W aplikacji mobilnej można również sprawdzić aktualny plan żywienia swojego konia. Do tej aplikacji będą mieć dostęp jedynie osoby posiadające konie.

W aplikacji desktopowej wyświetlane są statystyki aktywności koni danego użytkownika jak i szczegóły wizyt lekarzy i kowali. W tej aplikacji można zaplanować wyjazdy na zawody jak także szczegółowe plany żywienia swoich podopiecznych. W tej aplikacji tworzone będą także konta użytkowników, oraz ich koni. Dostęp do funkcji tworzenia kont będzie ograniczony i posiadać go będzie jedynie administrator aplikacji.

Każdy członek klubu będzie miał swoje konto z możliwością logowania zarówno do aplikacji mobilnej jak i desktopowej. Trenerzy, właściciele klubu i inne osoby związane z klubem będą miały dostęp jedynie do aplikacji desktopowej.

4.2 Wymagania funkcjonalne

Funkcjonalności aplikacji mobilnej oraz desktopowej nie są takie same mimo iż są połączone do jednej bazy, więc czerpią z tego samego źródła informacji. Pomimo znaczących różnic niektóre funkcjonalności pokrywają się w obu tych produktach. Wymagania funkcjonalne, które muszą spełniać obie aplikacje przedstawia tabela 4.1.

Wymaganie	Aktor	Opis wymagania
Logowanie do aplikacji	Trener, Członek klubu, Zarząd klubu	System pozwala na zalogowanie się po podaniu poprawnego loginu oraz hasła.
Resetowanie hasła przez email	Trener, Członek klubu, Zarząd klubu	System umożliwia resetowanie hasła przez adres e-mail.

Tabela 4.1: Wymagania funkcjonalne obu aplikacji

Aplikacja mobilna będzie służyć użytkownikom głównie do zapisu aktualnych wydarzeń z życia stajni. Jej głównym celem jest szybkie zapisanie informacji o aktywnościach koni i ich wizytach u lekarzy, bądź kowali. Można w niej także szybko sprawdzić przygotowany plan żywienia, oraz daty zbliżających się zawodów. Wymagania funkcjonalne dla aplikacji mobilnej zawierają poniższe tabele 4.2 i 4.3.

Wymaganie		Aktor	Opis wymagania
Zarządzanie aktywnościami	Dodawanie	Członek klubu	System umożliwia zapis danych wprowadzonych przez zalogowanego użytkownika do bazy danych.
	Edytowanie	Członek klubu	System umożliwia edytowanie dodanych wcześniej danych o aktywnościach.
	Usuwanie	Członek klubu	System pozwala na usuwanie dodanych wcześniej aktywności.
	Wyświetlanie	Trener, Członek klubu, Zarząd klubu	System umożliwia na przeglądanie wszystkich danych o aktywnościach danego konia zgromadzonych w bazie danych.
Zarządzanie wizytami	Dodawanie wizyt	Członek klubu	System pozwala na zapisanie danych z wizyty konia u lekarza/kowala do bazy danych.
	Edytowanie wizyt	Członek klubu	System powinien umożliwić zapis zaktualizowanych danych o wizycie do bazy.
	Usuwanie wizyt	Członek klubu	System powinien umożliwiać usuwanie danych o dodanych wcześniej wizytach.
	Wyświetlanie wizyt	Trener, Członek klubu, Zarząd klubu	System powinien umożliwić przeglądanie danych o wizytach zgromadzonych w bazie.
	Planowanie wizyt	Członek klubu	System powinien pozawalać użytkownikom na dodanie do bazy danych o następnej wizycie, czyli umożliwić zapis wizyt jedynie z datą i opisem.
	Zapisywanie zdjęć z wizyty	Członek klubu	System powinien pozwalać na zapisywanie zdjęć z wizyt.
	Przypomnienia o wizytach	Członek klubu	System powinien wysłać powiadomienie o zbliżającej się wizycie

Tabela 4.2: Wymagania funkcjonalne aplikacji mobilnej

Wymaganie		Aktor	Opis wymagania
Zarządzanie żywieniem	Przeglądanie planów żywienia	Członek klubu	System powinien umożliwiać przeglądanie planów żywienia umieszczonych w bazie.
	Wybór planu żywienia	Członek klubu	System powinien umożliwiać wybór jednego z planów żywienia umieszczonych w bazie jako tego aktualnie używanego.
Zarządzanie zawodami	Wyświetlanie najbliższych zawodów	Członek klubu	System powinien umożliwić wyświetlanie dat najbliższych zawodów umieszczonych w bazie.
	Potwierdzenie udziału w zawodach	Członek klubu	System powinien umożliwić użytkownikowi potwierdzenie swojego udziału w zawodach.
Udostępnianie koni		Członek klubu	System powinien umożliwić udostępnianie koni między użytkownikami.

Tabela 4.3: Wymagania funkcjonalne aplikacji mobilnej

Aplikacja desktop-owa przeznaczona jest zarówno dla użytkowników posiadających swoje konie jak i dla osób zarządzających klubem jeździeckim. W aplikacji desktop-owej posiadacze koni będą mogli obejrzeć zgromadzone informacje w przystępniejszej formie na dużym ekranie, stworzyć plan żywienia swojego konia, jak także przeanalizować statystyki swoich koni. Osoby zarządzające klubem będą miały możliwość dodawania nowych użytkowników i koni jak także sprawdzania statystyk wszystkich koni klubowych. Szczegółowe wymagania funkcjonalne dla aplikacji desktopowej zostały przedstawione w tabelach 4.4 oraz 4.5.

Wymaganie		Aktor	Opis wymagania
Zarządzanie planami żywienia	Tworzenie	Członek klubu	System umożliwia użytkownikowi stworzenie planu żywienia i zapisanie go do bazy.
	Edytowanie	Członek klubu	System pozwala aktualizować stworzone wcześniej plany żywienia.
	Usuwanie	Członek klubu	System umożliwia usuwanie danych o stworzonych wcześniej planach żywienia.
Zarządzanie końmi	Dodawanie	Zarząd klubu	System umożliwia wprowadzenie danych o koniach i dodanie ich do konkretnego użytkownika
	Usuwanie	Zarząd klubu	System umożliwia usuwanie koni
	Edytowanie	Zarząd klubu	System umożliwia edycję danych o koniach zgromadzonych już w bazie.
Zarządzanie użytkownikami	Dodawanie	Zarząd klubu	System umożliwia dodawanie danych o użytkownikach i tworzenie ich kont.
	Edytowanie	Zarząd klubu	System umożliwia edytowanie danych użytkownika
	Usuwanie	Zarząd klubu	System umożliwia usuwanie użytkowników
	Zmiana hasła	Zarząd klubu, Członek klubu, Trener	System umożliwia zmianę hasła przez użytkownika.
Zarządzanie zawodami	Dodawanie zawodów	Zarząd klubu	System pozwala na tworzenie zawodów, oraz zapraszanie do udziału w nich poszczególnych członków klubu
	Edytowanie zawodów	Zarząd klubu	System pozwala na edycję danych o dodanych wcześniej zawodach
	Usuwanie zawodów	Zarząd klubu	System pozwala na usuwanie danych o dodanych wcześniej zawodach.

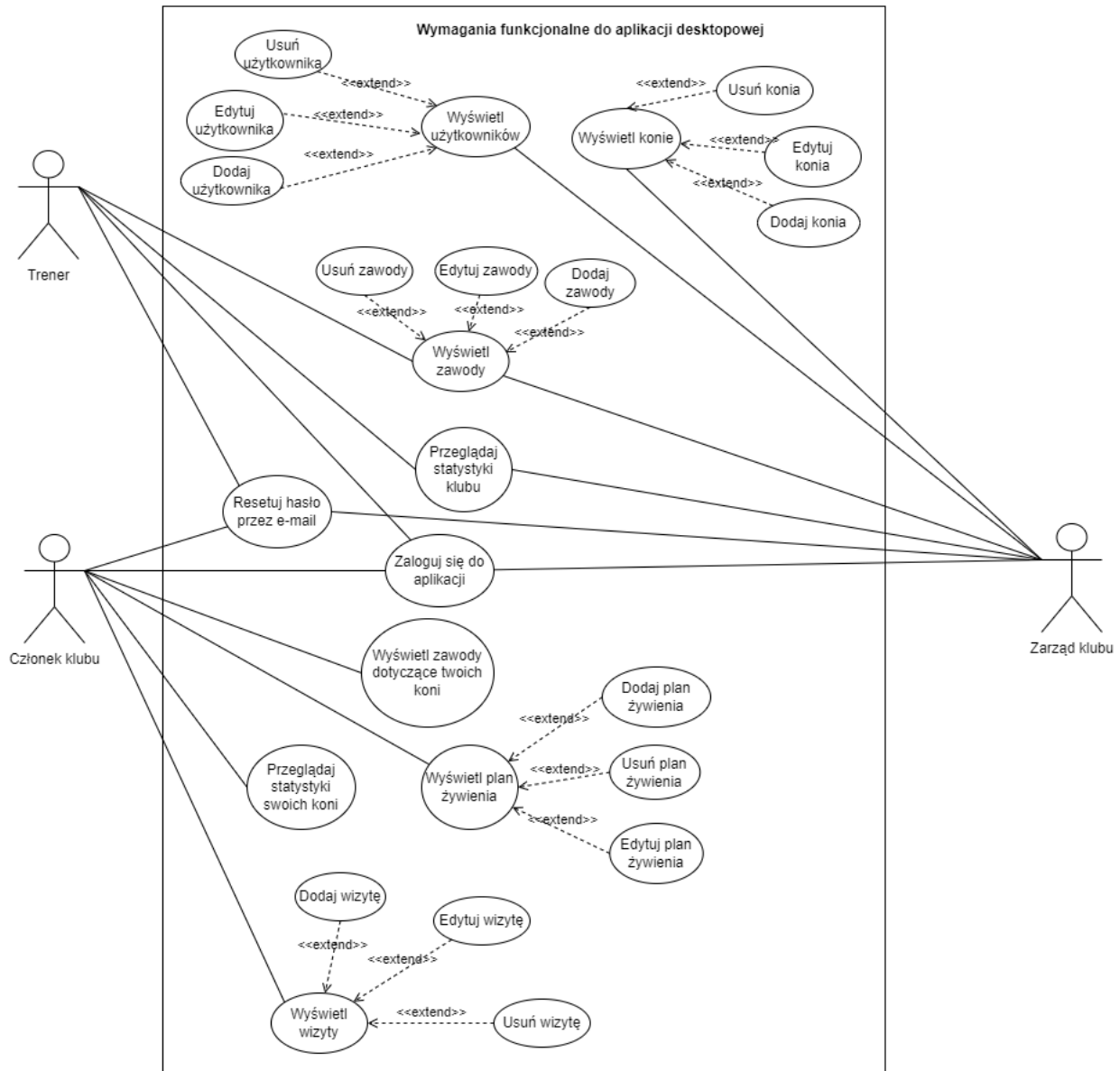
Tabela 4.4: Wymagania funkcjonalne aplikacji desktopowej

Wymaganie	Aktor	Opis wymagania
Przeglądanie historii wizyt	Członek klubu, Trener, Zarząd klubu	DODAĆ OPIS
Przeglądanie statystyk	Członek klubu, Trener, Zarząd klubu	DODAĆ OPIS

Tabela 4.5: Wymagania funkcjonalne aplikacji desktopowej

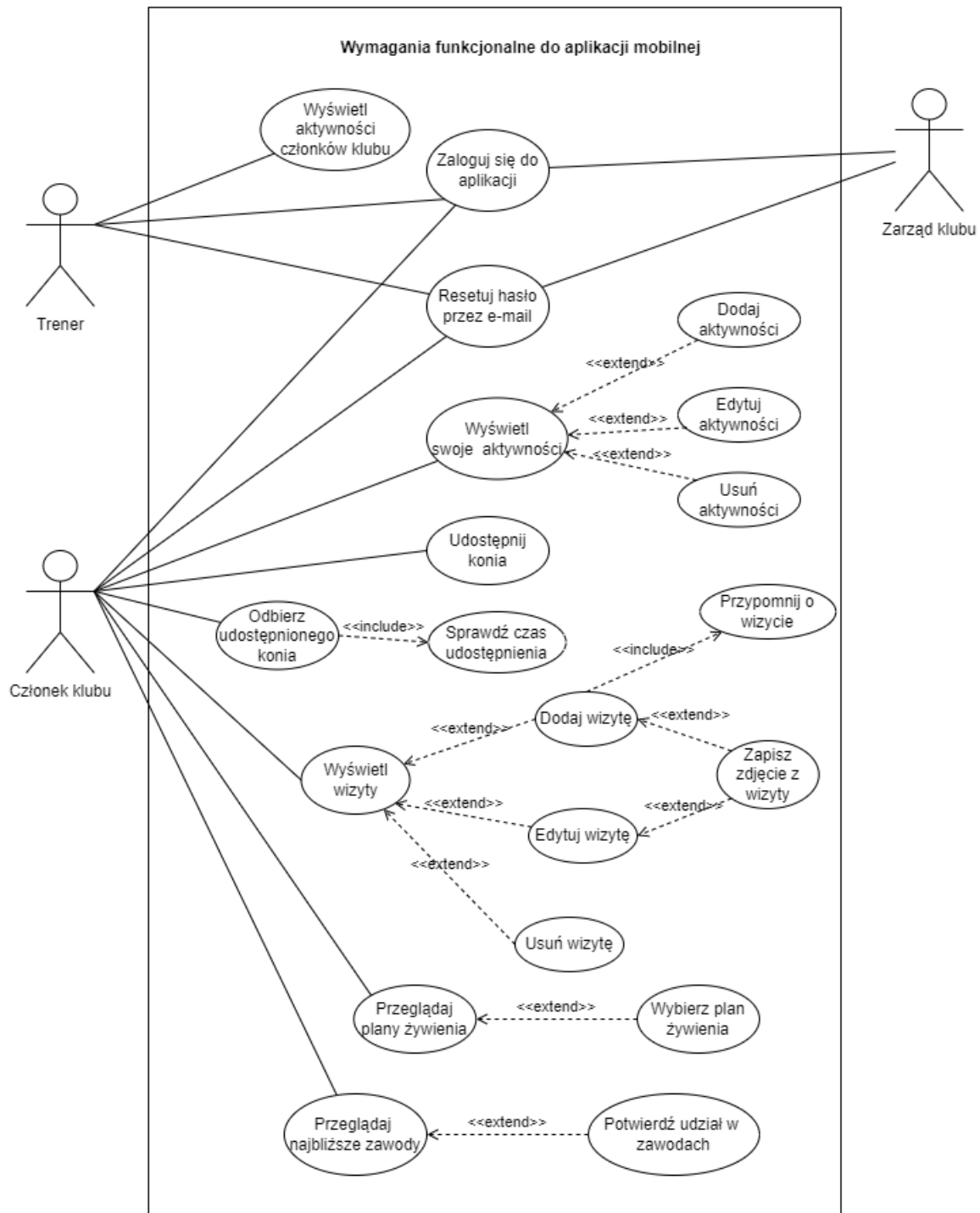
Przypadki użycia

Wszystkie wymagania funkcjonalne zgromadzone w powyższych tabelach, możemy przedstawić na diagramie przypadków użycia UML. Poniższe rysunki zostały sporządzone według zasad języka UML, opisanych w pozycji [odnieść się do bibliografii]. Na rysunku 4.1 przedstawione zostały przypadki użycia aplikacji desktopowej, zaś na rysunku 4.2 przedstawione zostały przypadki użycia aplikacji mobilnej.



Rysunek 4.1: Diagram Use Case dla aplikacji desktopowej

Źródło: Opracowanie własne



Rysunek 4.2: Diagram Use Case dla aplikacji mobilnej

Źródło: Opracowanie własne

4.3 Wymagania niefunkcjonalne

W tym rozdziale przedstawione zostaną wymagania niefunkcjonalne projektowanego systemu. W osobnych tabelach przedstawione zostaną wymagania dla aplikacji mobilnej (tabela 4.7) oraz desktopowej (tabela 4.6). Przedstawione wymagania zostały opracowane zgodnie ze standardem ISO 9126. Określone zostały atrybuty, takie jak: niezawodność (Reliability), obsługiwalność (Usability), wydajność (Efficiency), łatwość konserwacji (Maintainability) i przenośność (Portability). Na początek przedstawione zostały wymagania dla aplikacji desktopowej.

Tabela 4.6: Wymagania niefunkcjonalne aplikacji desktopowej

Źródło: Opracowanie własne

Nr	Nazwa wymagania	Opis wymagania
1	Interfejsy programowe	System operacyjny Microsoft Windows 7 lub nowszy. Baza danych zainstalowana na platformie Microsoft SQL Server 2019 lub nowszej oraz dostępna dla aplikacji zgodnie z zasadami określonymi w dokumentacji serwera.
2	Interfejsy sprzętowe	Komputer osobisty lub laptop, obsługujący system Windows 7 lub nowszy. Minimum 4 GB pamięci RAM i xGB wolnej przestrzeni na dysku, procesor 1GHz lub szybszy, 32 bitowy (x86) lub 64 bitowy (x64).
3	Obsługiwalność (Usability)	Aplikacja powinna mieć prosty i intuicyjny interfejs użytkownika. Interfejs powinien być dostosowany do pracy na monitorach o małej rozdzielczości i laptopach.
4	Niezawodność (Reliability)	Aplikacja powinna walidować wszystkie pola, do których wprowadzane są dane. W przypadku błędnych danych powinny wyświetlać się komunikaty informujące o nieprawidłowościach w klarowny sposób.

5	Język i narzędzia programowania	Aplikacja została napisana w C# na silniku graficznym Windows Presentation Foundation przy użyciu środowiska Microsoft Visual Studio Community 2022. Baza danych została napisana w języku SQL, przy pomocy środowiska Microsoft SQL Server Management Studio 18 i zainstalowana na serwerze bazy danych Microsoft SQL Server 2019
6	Aspekty prawne	W bazie danych przechowywane będą dane członków klubu, zarządu, specjalistów oraz których korzystają. Poza danymi osobowymi tych osób przechowywane będą także dane o koniach, ich aktywnościach oraz stanie zdrowia. Aplikacja nie będzie przekazywać danych osobowych poszczególnych członków innym użytkownikom. Dane specjalistów będą rozpowszechniane między użytkownikami, dlatego przed wpisaniem do bazy będą musieli wyrazić na to pisemną zgodę.
7	Wydajność (Efficiency)	Używanie aplikacji na sprzęcie o minimalnych wymaganiach powinno przebiegać w sposób płynny, to znaczy wyświetlanie, dodawanie, edytowanie, usuwanie, kategoryzowanie i filtrowanie danych powinno nie zajmować dłużej niż kilka sekund.
8	Łatwość konserwacji (Maintainability)	Działanie aplikacji będzie kontrolowane przy pomocy dziennika debugowania. Kolejne wersje systemu (kopie zapasowe) będą zapisywane za pomocą systemu kontroli wersji GIT.

9	Przenośność (Portability)	Aplikacja udostępniana będzie w pliku wykonywalnym .exe. Baza danych musi zostać zainstalowana i skonfigurowana na serwerze w danym klubie jeździeckim. Po zainstalowaniu aplikacja i baza danych powinny zostać skonfigurowane ze sobą.
---	---------------------------	--

Następnie w tabeli 4.7 przedstawiono wymagania dla aplikacji mobilnej. Zostały one podzielone na takie same sekcje jak wymagania aplikacji desktopowej. Mimo to wymagania przedstawione zostały w oddzielnej tabeli, ponieważ dotyczą one aplikacji na całkowicie inny system.

Tabela 4.7: Wymagania niefunkcjonalne aplikacji mobilnej

Źródło: Opracowanie własne

Nr	Nazwa wymagania	Opis wymagania
1	Interfejsy programowe	Android ?? lub nowszy
2	Interfejsy sprzętowe	Telefon lub tablet z systemem operacyjnym Android ?? lub nowszym
3	Obsługiwalność (Usability)	Aplikacja powinna dobrze skalować się na różnej wielkości ekrany. Na telefonie najważniejsze przyciski powinny znajdować się "w zasięgu kciuka". Ikony i przyciski powinny być widoczne i możliwe do kliknięcia nawet małych ekranach (minimalny ekran ??), w aplikacji dostępny jest jedynie tryb jasny.
4	Niezawodność (Reliability)	Aplikacja powinna walidować wszystkie pola, do których wprowadzane są dane. W przypadku błędnych danych powinny wyświetlać się komunikaty informujące o nieprawidłowościach w klarowny sposób.

5	Język i narzędzia programowania	Aplikacja została napisana na platformie Xamarin przy użyciu środowiska Microsoft Visual Studio Community 2022. Używa tej samej bazy co aplikacja desktopowa.
6	Aspekty prawne	Aspekty prawne aplikacji mobilnej i desktopowej pokrywają się, ponieważ korzystają one z tych samych danych.
7	Wydajność (Efficiency)	Używanie aplikacji na sprzęcie o minimalnych wymaganiach powinno przebiegać w sposób płynny, to znaczy wyświetlanie, dodawanie, edytowanie, usuwanie danych powinno nie zajmować dłużej niż kilka sekund.
8	Łatwość konserwacji (Maintainability)	Konserwacja aplikacji mobilnej będzie przebiegać analogicznie do desktopowej.
9	Przenośność (Portability)	Aplikacja udostępniana w postaci pakietu instalacyjnego .apk oraz powinna spełniać wszystkie wymagania do umieszczenia jej w sklepie Google Play.

Rozdział 5

Baza danych

W tym rozdziale przedstawimy model konceptualny, logiczny oraz fizyczny bazy danych. Rozdział ten został opracowany na podstawie [1].

5.1 Model konceptualny

Proces tworzenia bazy danych zaczynamy od modelu konceptualnego. W pierwszej fazie tworzenia go ważne jest określenie słownika pojęć, które będą następnie używane w projekcie bazy danych.

Słownik pojęć

- **Użytkownik** - wszyscy członkowie klubu, trenerzy oraz zarząd klubu.
- **Koń** - koń należący do któregoś z członków klubu jeździeckiego, lub dzierżawiony przez niego.
- **Atywności** - są to czynności wykonywane przez konia w ciągu dnia, należą do nich jazdy, skoki przez przeszkody, kross, ujeżdżenie, lonża, wyjazd w teren, karuzela, padok, wyjazd na zawody, spacer, skoki luzem, padok.
- **Wizyty** - to wizyty wszelkich lekarzy, jak także wizyty kowali.
- **Udostępnianie konia** - jest to przekazanie możliwości wprowadzania danych o danym koniu przez jego właściciela innemu członkowi klubu.

Po określeniu definicji poszczególnych pojęć używanych w projekcie możemy określić reguły funkcjonowania naszej aplikacji.

Reguły funkcjonowania

Reguły funkcjonowania określają zasady, procedury i wytyczne jakie musi spełniać projektowana aplikacja.

REG\001 Konta użytkowników tworzy jedynie użytkownik "administrator".

REG\002 Każdy użytkownik ma określony swój typ.

REG\003 Każdy użytkownik może zmienić swoje hasło.

REG\004 O każdym użytkowniku, jak także o lekarzu i kowalu zbieramy podstawowe dane personalne.

REG\005 Tylko użytkownik "administrator" dodaje konie do kont użytkowników.

REG\006 Każdy koń ma przypisaną płć.

REG\007 Każdy koń ma przypisany status.

REG\008 Jeden użytkownik może posiadać wiele koni.

REG\009 Aktywności konia może dodać jego właściciel lub osoba której właściciel udostępni konia.

REG\010 Koń może mieć wiele aktywności każdego dnia.

REG\011 Wizyty konia może dodawać tylko jego właściciel.

REG\012 Na wizycie jest jeden koń i jedne lekarz/kowal.

REG\013 Każdy lekarz ma określoną specjalizację.

REG\014 Plan żywienia konia może ustalać tylko właściciel.

REG\015 Koń może posiadać wiele planów żywienia, ale aktualnie może jeść tylko jeden.

REG\016 Plan żywienia zawiera wiele żywien.

REG\017 Żywnienie dotyczy konkretnego typu jedzenia, podawanego o konkretnej porze (rano, południe, wieczór), który swoją jednostkę miary.

REG\018 Użytkownicy, którym ktoś udostępnił konia mogą tylko wyświetlić plan żywienia.

REG\019 Statystyki mają być tworzone na podstawie aktywności.

REG\020 Użytkownik "członek klubu" może przeglądać statystyki tylko swoich koni.

REG\021 Użytkownik "trener" lub "administrator" może przeglądać statystyki wszystkich koni.

REG\022 Użytkownik "trener" lub "administrator" może dodawać wyjazd na zawody dla całego klubu i zapraszać poszczególnych użytkowników.

REG\023 Użytkownik "członek klubu" może dodawać swoje wyjazdy na zawody.

Ograniczenia dziedzinowe

Ograniczenia dziedzinowe to ograniczenia, które nakładane są na atrybuty w powyższych kategoriach. Wynikają one z analizy wycinka rzeczywistości i należy je uwzględnić podczas projektowania bazy danych oraz implementacji systemu.

OGR\001 Paszport konia składa się ze znaków i cyfr postaci xxx-aaa-bb-cccc-dd, gdzie

- xxx - określa kraj pochodzenia konia,
- aaa - oznacza kod hodowli konia,
- bb- oznacza rok urodzenia konia,
- ccccc - to numer paszportu konia,
- dd - to numer identyfikacyjny konia w ramach hodowli.

OGR\002 Data wizyty konia jest wcześniejsza niż data jego urodzenia.

Encje

Po określeniu kategorii, reguł funkcjonowania, ograniczeń dziedzinowych i transakcji należy przystąpić do tworzenia encji i relacji między nimi.

ENC\01 ACTIVITY

Semantyka encji - Encja zawierająca aktywności, które koń wykonuje w ciągu dnia. Każda aktywność oprócz typu, zawiera opis, czas trwania oraz ocenę satysfakcji i intensywności jej wykonania.

Opis atrybutów encji znajduje się w tabeli 5.1.

Tabela 5.1: Wykaz atrybutów encji typu Activity

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>activityID</i>	Numer identyfikujący aktywności	Liczba naturalna	+
<i>date</i>	Data wykonania aktywności	Data	+
<i>description</i>	Opis aktywności	Typ znakowy	-
<i>time</i>	Czas trwania aktywności	Czas	-
<i>intensity</i>	Intensywność wykonanej aktywności	Liczba naturalna	+
<i>satisfaction</i>	Satysfakcja wykonanej aktywności	Liczba naturalna	+
<i>activityType</i>	Typ wykonanej aktywności	Liczba naturalna	+

Klucze kandydujące: activityID

Klucz główny: activityID

Charakter encji: encja słaba

ENC\02 COMPETITION

Semantyka encji - encja zawierająca dane o zawodach jeździeckich.

Opis atrybutów znajduje się w tabeli 5.2.

Tabela 5.2: Wykaz atrybutów encji typu Competition

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>competitionID</i>	Numer identyfikujący zawody.	Liczba naturalna	+
<i>spot</i>	Miejsce, w którym odbywają się zawody.	Max. znaków 200	-
<i>date</i>	Data zawodów.	Data	+
<i>description</i>	Opis zawodów.	Typ znakowy	-
<i>rank</i>	Ranga zawodów.	Max. znaków 50	-

Klucze kandydujące: competitionID

Klucz główny: competitionID

Charakter encji: encja silna

ENC\03 NOTIFICATION*Semantyka encji* - Encja zawierająca powiadomienia utworzone przez użytkowników.

Opis atrybutów encji znajduje się w tabeli 5.3.

Tabela 5.3: Wykaz atrybutów encji typu Notification

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>notificationID</i>	Numer identyfikujący powiadomienie	Liczba naturalna	+
<i>title</i>	Tytuł powiadomienia	Max. znaków 30	+
<i>description</i>	Opis powiadomienia	Typ znakowy	-
<i>sendDate</i>	Data wysłania	Data	+
<i>createdDate</i>	Data stworzenia	Data	+
<i>turnOn</i>	Określenie, czy powiadomienie jest włączone	Prawda/Fałsz	+

Klucze kandydujące: notificationID

Klucz główny: notificationID

Charakter encji: encja słaba

ENC\04 PROFESSIONAL

Semantyka encji - encja opisująca specjalistów przyjeżdżających do konia takich jak lekarze (np. gastrolog, kardiolog, lekarz ogólny), fizjoterapeuci, kowale itp.

Opis atrybutów encji znajduje się w tabeli 5.4.

Tabela 5.4: Wykaz atrybutów encji typu PROFESSIONAL

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>professionalsID</i>	Numer identyfikujący specjalistę	Liczba naturalna	+
<i>degree</i>	Stopień naukowy doktora	Liczba naturalna	-

Klucze kandydujące: professionalsID

Klucz główny: professionalsID

Charakter encji: encja słaba

ENC\05 SPECIALISATION

Semantyka encji - encja słownikowa zawiera nazwy specjalizacji specjalistów.

Opis atrybutów encji znajduje się w tabeli 5.5.

Tabela 5.5: Wykaz atrybutów encji typu Specialization

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>specialisationID</i>	Numer identyfikujący specjalizacje	Liczba naturalna	+
<i>name</i>	Nazwa specjalizacji	Max. znaków 85	+

Klucze kandydujące: specializationID

Klucz główny: specializationID

Charakter encji: encja silna

ENC\06 DIET

Semantyka encji - Encja zawierająca informacje o aktywnej diecie konia.

Opis atrybutów encji znajduje się w tabeli 5.6.

Tabela 5.6: Wykaz atrybutów encji typu Diet

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>dietID</i>	Numer identyfikujący dietę	Liczba naturalna	+
<i>isActive</i>	Określenie czy plan jest w użyciu	Prawda/Fałsz	+

Klucze kandydujące: dietID

Klucz główny: dietID

Charakter encji: encja słaba

ENC\07 PORTION

Semantyka encji - Encja zawierająca informacje o porcji jedzenia.

Opis atrybutów encji znajduje się w tabeli 5.6.

Tabela 5.7: Wykaz atrybutów encji typu Portion

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>portionID</i>	Numer identyfikujący jedzenie	Liczba naturalna	+
<i>amount</i>	Ilość jedzenia w porcji	Liczba zmiennoprzecinkowa	+

Klucze kandydujące: portionID

Klucz główny: portionID

Charakter encji: encja słaba

ENC\08 FORAGE

Semantyka encji - Encja zawierająca informacje o paszy dla koni.

Opis atrybutów encji znajduje się w tabeli 5.8.

Tabela 5.8: Wykaz atrybutów encji typu Forage

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>forageID</i>	Numer identyfikujący paszy	Liczba naturalna	+
<i>name</i>	Nazwa paszy	Max. znaków 92	+
<i>producent</i>	Producent paszy	max. znaków 57	-
<i>capacity</i>	Ilość paszy w jednym worku	Liczba naturalna	-

Klucze kandydujące: forageID

Klucz główny: forageID

Charakter encji: encja słaba

ENC\09 HORSE

Semantyka encji - Encja zawierająca wszystkie informacje o koniach.

Opis atrybutów encji znajduje się w tabeli 5.9.

Tabela 5.9: Wykaz atrybutów encji typu Horse

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>horseID</i>	Numer identyfikujący konia	Liczba naturalna	+
<i>name</i>	Imie konia	max. znaków 60	+
<i>mother</i>	Imie klaczy	max. znaków 60	-
<i>father</i>	Imie ogiera	Max. znaków 60	-
<i>birthday</i>	Data urodzenia konia	Date	-
<i>race</i>	Rasa konia	Max. znaków 50	-
<i>breeder</i>	Hodowca koni	Max. znaków 60	-
<i>passport</i>	Paszport konia	Max. znaków 20	-
<i>photo</i>	Zdjęcie konia	Typ znakowy	-

Klucze kandydujące: horseID

Klucz główny: horseID

Charakter encji: encja słaba

ENC\10 HORSEGENDER

Semantyka encji - Encja słownikowa zawierająca płeć koni.

Opis atrybutów encji znajduje się w tabeli 5.10.

Tabela 5.10: Wykaz atrybutów encji typu HorseGender

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>genderID</i>	Numer identyfikujący płeć konia	Liczba naturalna	+
<i>gender</i>	Nazwa płci konia	Max. znaków 10	+

Klucze kandydujące: genderID

Klucz główny: genderID

Charakter encji: encja silna

ENC\11 STATUS

Semantyka encji - Encja słownikowa zawierająca statusy koni.

Opis atrybutów encji znajduje się w tabeli 5.11.

Tabela 5.11: Wykaz atrybutów encji typu Status

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>statusID</i>	Numer identyfikujący status konia	Liczba naturalna	+
<i>name</i>	Nazwa statusu konia	Max. znaków 20	+

Klucze kandydujące: statusID

Klucz główny: statusID

Charakter encji: encja silna

ENC\12 MEALNAME

Semantyka encji - Encja słownikowa zawierająca nazwy posiłków.

Opis atrybutów encji znajduje się w tabeli 5.12.

Tabela 5.12: Wykaz atrybutów encji typu MealName

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>mealNameID</i>	Numer identyfikujący posiłek	Liczba naturalna	+
<i>mealName</i>	Nazwa posiłku	Max. znaków 20	+

Klucze kandydujące: mealNameID

Klucz główny: mealNameID

Charakter encji: encja słaba

ENC\13 NUTRITIONPLAN

Semantyka encji - Encja zawierająca informacje o planie żywienia koni.

Opis atrybutów encji znajduje się w tabeli 5.13.

Tabela 5.13: Wykaz atrybutów encji typu NutrtrtionPlan

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>nutritionPlanID</i>	Numer identyfikujący plan żywienia	Liczba naturalna	+
<i>title</i>	Tytuł planu żywienia	Max. znaków 50	+
<i>desctription</i>	Ilość jedzenia w porcji	Typ znakowy	-
<i>icon</i>	Ikona dołączona do planu żywienia	Liczba naturalna	+
<i>color</i>	Color dołączona do planu żywienia	Max. 7 znaków	+

Klucze kandydujące: nutritionPlanID

Klucz główny: nutritionPlanID

Charakter encji: encja silna

ENC\14 PEOPLEDETAILS

Semantyka encji - encja zawiera szczegółowe dane użytkowników (członków klubu, trenerów i zarządu klubu) jak i lekarzy oraz kowali.

Opis atrybutów encji znajduje się w tabeli 5.14.

Tabela 5.14: Wykaz atrybutów encji typu PeopleDetails

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>detailsID</i>	Numer identyfikujący dane użytkowników	Liczba naturalna	+
<i>name</i>	Imie	Max. znaków 40	-
<i>surname</i>	Nazwisko	Max. znaków 40	+
<i>phonNumber</i>	Numer telefonu	Max. znaków 20	-
<i>email</i>	Adres e-mailowy	Max. znaków 320	-
<i>city</i>	Miasto zamieszkania	Max. znaków 200	-
<i>street</i>	Ulica zamieszkania	Max. znaków 90	-
<i>number</i>	Numer domu zamieszkania	Max. znaków 10	-

Klucze kandydujące: detailsID

Klucz główny: detailsID

Charakter encji: encja silna

ENC\15 PARTICIPATION*Semantyka encji* - encja zawierająca dane o uczestnictwie konia w zawodach.

Opis atrybutów encji znajduje się w tabeli 5.15.

Tabela 5.15: Wykaz atrybutów encji typu Participation

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>participationID</i>	Numer identyfikujący udział w zawodach	Liczba naturalna	+
<i>result</i>	Wynik zawodów	Typ znakowy	+
<i>place</i>	Zajęte miejsce	Liczba naturalna	+

Klucze kandydujące: participationID

Klucz główny: participationID

Charakter encji: encja słaba

ENC\16 SHARED

Semantyka encji - Encja zawierająca wpisy o udostępnianiu koni.

Opis atrybutów encji znajduje się w tabeli 5.16.

Tabela 5.16: Wykaz atrybutów encji typu Shared

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>sharedID</i>	Numer identyfikujący status konia	Liczba naturalna	+
<i>code</i>	Kod z kodu QR	Max. znaków 50	+
<i>endDate</i>	Data kończąca udostępnienie	Data	+
<i>startDate</i>	Data udostępnienia	Data	+

Klucze kandydujące: sharedID

Klucz główny: sharedID

Charakter encji: encja słaba

ENC\17 UNITOFMEASURE

Semantyka encji - Encja słownikowa zawierająca nazwy jednostek miary.

Opis atrybutów encji znajduje się w tabeli 5.17.

Tabela 5.17: Wykaz atrybutów encji typu UnitOfMeasure

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>unitID</i>	Numer identyfikujący jednostkę miary	Liczba naturalna	+
<i>unitName</i>	Nazwa jednostek miary	Max. znaków 30	+

Klucze kandydujące: unitID

Klucz główny: unitID

Charakter encji: encja silna

ENC\18 UserAccount

Semantyka encji - encja zawiera dane użytkownika (członków klubu, trenerów i zarządu klubu).

Opis atrybutów encji znajduje się w tabeli 5.18.

Tabela 5.18: Wykaz atrybutów encji typu UserAccount

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>userID</i>	Numer identyfikujący użytkownika	Liczba naturalna	+
<i>login</i>	Login użytkownika	max. znaków 50	+
<i>hash</i>	Hash hasła użytkownika	max. znaków 50	+
<i>salt</i>	Salt hasła użytkownika	max. znaków 50	+
<i>createdDateTime</i>	Data utworzenia konta	Data	+

Klucze kandydujące: userID

Klucz główny: userID

Charakter encji: encja słaba

ENC\19 USERTYPE

Semantyka encji - encja zawiera typy użytkowników: zwykły użytkownik (standard), trener (trainer), zarząd klubu (admin).

Opis atrybutów encji znajduje się w tabeli 5.19.

Tabela 5.19: Wykaz atrybutów encji typu UserType

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>userTypeID</i>	Numer identyfikujący typ użytkownika	Liczba naturalna	+
<i>typeName</i>	Nazwa typu użytkownika	max. znaków 20	+

Klucze kandydujące: *userTypeID*Klucz główny: *userTypeID*

Charakter encji: encja silna

ENC\20 VISIT*Semantyka encji* - encja zawierająca dane o wizytach.

Opis atrybutów encji znajduje się w tabeli 5.20.

Tabela 5.20: Wykaz atrybutów encji typu Visit

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>visitID</i>	Numer identyfikujący wizyte	Liczba naturalna	+
<i>cost</i>	Cena wizyty	Liczba rzeczywista dodatnia	-
<i>summary</i>	Opis podsumowujący wizytę	Typ znakowy	-
<i>artefactImage</i>	Zdjęcie z wizyty	Typ znakowy	-
<i>visitDate</i>	Data wizyty	Data	+

Klucze kandydujące: *visitID*Klucz główny: *visitID*

Charakter encji: encja słaba

ENC\21 MEAL

Semantyka encji - Encja słownikowa zawierająca posiłki.

Opis atrybutów encji znajduje się w tabeli 5.21.

Tabela 5.21: Wykaz atrybutów encji typu Meal

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>mealID</i>	Numer identyfikujący posiłek	Liczba naturalna	+
<i>hour</i>	Godzina podawania posiłku	Max. znaków 10	-

Klucze kandydujące: mealID

Klucz główny: mealID

Charakter encji: encja słaba

ENC\22 CONTEST

Semantyka encji - Encja zawierająca konkursy z zawodów.

Opis atrybutów encji znajduje się w tabeli 5.22.

Tabela 5.22: Wykaz atrybutów encji typu Contest

Źródło: Opracowanie własne

Nazwa atrybutu	Opis atrybutu	Typ	OBL(+) OPC(-)
<i>contestID</i>	Numer identyfikujący posiłek	Liczba naturalna	+
<i>level</i>	Poziom konkursu	Max. znaków 10	-
<i>name</i>	Nazwa konkursu	Max. znaków 50	-

Klucze kandydujące: contestID

Klucz główny: contestID

Charakter encji: encja słaba

Po zaprojektowaniu encji możemy zapisać predykatowe definicje typów encji:

ENC\01 **ACTIVITY**(activityID, date, description, time, intensivity, satisfaction, activityType)

ENC\02 **COMPETITION** (competitionID, spot, description, rank, date)

ENC\03 **NOTIFICATION** (notificationID, title, description, sendDate, createdDate, turnOn)

ENC\04 **PROFESSIONALS** (professionalsID, degree)

ENC\05 **SPECIALISATION** (specialisationID, name)

ENC\06 **DIET** (dietID, isActive)

ENC\07 **PORTION** (portionID, amount)

ENC\08 **FORAGE** (forageID, name, producent, capacity)

ENC\09 **HORSE** (horseID, name, mother, father, birthday, race, breeder, passport, photo)

ENC\10 **HORSEGENDER** (genderID, gender)

ENC\11 **STATUS** (statusID, name)

ENC\12 **MEALNAME** (mealNameID, mealName)

ENC\13 **NUTRITIONPLAN** (nutritionPlanID, title, description, icon, color)

ENC\14 **PEOPLEDETAILS** (detailsID, name, surname, phone, email, city, street, number)

ENC\15 **PARTICIPATION** (participationID, result, place)

ENC\16 **SHARED** (sharedID, code, endDate, startDate)

ENC\17 **UNITOFMEASURE** (unitID, unitName)

ENC\18 **USERACCOUNT** (userID, accountLogin, hash, salt, createdDateTime)

ENC\19 **USERTYPE** (userTypeID, typeName)

ENC\20 **VISIT** (careID, cost, summary, artefactImage, visitDate)

ENC\21 **MEAL** (mealID, hour)

ENC\22 **CONTEST** (contestID, level, name)

Predykatowe definicje związków encji:

ZWI\xx Związek (ENCJA1(min, max) ENCJA2(min, max))

ZWI\01 Has (HORSE(0,N) HORSEGENDER(1,1))

ZWI\02 Has (HORSE(0,N) HORSESTATUS(1,1))

ZWI\03 Take (HORSE(1,1) PARTICIPATION(0,N))

ZWI\04 Refers to (PARTICIPATION(0,N) COMPETITION(1,1))

ZWI\05 Performs (HORSE(1,1) ACTIVITY(0,N))

ZWI\06 Train (ACTIVITY(1,1) USERACOUNT(0,N))

ZWI\07 Ride (ACTIVITY(1,1) USERACOUNT(0,N))

ZWI\08 Refers to (NOTIFICATION(0,N) USERACOUNT(1,1))

ZWI\09 Refers to (NOTIFICATION(0,N) Horse(1,1))

ZWI\10 Gets (USERACOUNT(1,1) SHARE(0,N))

ZWI\11 Shares (USERACOUNT(1,1) SHARE(0,N))

ZWI\12 IsShared (HORSE(1,1) SHARE(0,N))

ZWI\13 Has (USERACOUNT(0,N), USERTYPE(1,1))

ZWI\14 Concern (PEOPLEDDETAILS(1,1) USERACOUNT(0,N))

ZWI\15 Attend (HORSE(1,1) VISIT(0,N))

ZWI\16 CareOn (VISIT(0,N) PROFESSIONALS(1,1))

ZWI\17 Has (PROFESSIONALS(0,N) SPECIALISATION(1,1))

ZWI\18 IsOn (HORSE(1,1) DIET(0,N))

ZWI\19 Contains (DIET(0,N) NUTRITIONPLAN(1,1))

ZWI\20 BelongsTo (MEAL(0,N) NUTRITIONPLAN(0,1))

ZWI\21 Has (MEALNAME(1,1) MEAL(0,N))

ZWI\22 ConsistOf (MEAL(1,1) PORTION(0,N))

ZWI\23 Contains (PORTION(0,N) FORAGE(1,1))

ZWI\24 Has (FORAGE(0,N) UNITOFMEASURE(1,1))

ZWI\25 Has (PORTION(0,N) UNITOFMEASURE(1,1))

ZWI\26 Add (NUTRITIONPLAN(0,N) USERACCOUNT(1,1))

ZWI\27 Has (HORSE(0,N) USERACCOUNT(1,1))

Model konceptualny przedstawiony został na diagramie ERD, który został przedstawiony na rysunku 5.1.

5.2 Model logiczny

Po stworzeniu modelu konceptualnego, należy przetransformować go do modelu logicznego. Poniżej przedstawiono tabele opisujące schematy relacji oraz znaczenia atrybutów tych relacji.

REL\01 Activities\ACTIVITY

Opis schematu relacji znajduje się w tabeli 5.23.

Tabela 5.23: Opis schematu relacji Activities

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>activityID</i>	Integer+		+			+	PR		SZBD
<i>date</i>	Date	dd-mm-rrrr	+						USER
<i>description</i>	String		-						USER
<i>time</i>	Integer		+						USER
<i>intensivity</i>	Integer+		+						USER
<i>satisfaction</i>	Integer+		+						USER
<i>activityType</i>	Integer+		+						USER
<i>userID</i>	Integer+		+				FK	User	BD
<i>horseID</i>	Integer+		+				FK	Horse	BD
<i>trainerID</i>	Integer+		-				FK	User	BD

Tabela 5.24: Opis atrybutów relacji Activities

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>activityID</i>	Unikalne ID aktywności, generowane przez aplikację, klucz główny tabeli.
<i>date</i>	Data wykonywanej aktywności.
<i>description</i>	Opis aktywności, dowolne polskie znaki.
<i>time</i>	Liczba naturalna, oznaczająca czas trwania wykonywanej aktywności.
<i>intensivity</i>	Liczba naturalna od 0 do 5 oznaczająca poziom intensywności treningu.
<i>satisfaction</i>	Liczba naturalna od 0 do 5 oznaczająca poziom satysfakcji z treningu.
<i>activityType</i>	Liczba naturalna oznaczająca typ aktywności.
<i>userID</i>	Indentyfikator użytkownika, który wpisuje aktywność.
<i>horseID</i>	Identyfikator konia, którego dotyczy aktywność.
<i>trainerID</i>	Identyfikator użytkownika typu trener, który przeprowadzał trening.

REL\02 Competitions\COMPETITION

Opis schematu relacji znajduje się w tabeli 5.25.

Tabela 5.25: Opis schematu relacji Competitions

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>competitionID</i>	Integer+		+			+	PR		SZBD
<i>spot</i>	String		-						USER
<i>rank</i>	String		-						USER
<i>date</i>	Date		+						USER
<i>description</i>	String		+						USER

Tabela 5.26: Opis atrybutów relacji Competitions

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>competitionID</i>	Unikalny numer ID identyfikujący zawody, generowany przez aplikację.
<i>spot</i>	Miejsce, w którym odbędą się zawody.
<i>rank</i>	Ranga zawodów np. regionalne/międzynarodowe itp.
<i>date</i>	Dzień, w którym odbędą się zawody.
<i>description</i>	Opis zawodów.

REL\03 Notifications\NOTIFICATION

Tabela 5.27: Opis schematu relacji Notifications

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>notificationID</i>	Integer+		+			+	PR		SZBD
<i>title</i>	String		+						USER
<i>description</i>	String		+						USER
<i>sendDate</i>	Date		+						USER
<i>createdDate</i>	Date		+						USER
<i>userID</i>	Integer+		+				FK	User	BD

Tabela 5.28: Opis atrybutów relacji Notifiactions

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>notificationID</i>	Unikalny numer ID identyfikujący powiadomienie
<i>title</i>	Tytuł powiadomienia
<i>description</i>	Opis pojawiający się na powiadomieniu
<i>sendDate</i>	Data i godzina informująca kiedy ma zostać wysłane powiadomienie
<i>createdDate</i>	Data i godzina stworzenia powiadomienia
<i>userID</i>	Numer ID identyfikujący użytkownika wysyłającego powiadomienie

REL\04 Professionals\PROFESSIONAL

Tabela 5.29: Opis schematu relacji Professionals

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>professionalID</i>	Integer+		+			+	PR		SZBD
<i>degree</i>	String		-						USER
<i>detailsID</i>	Integer+		+				FK	Details	BD
<i>specialisationID</i>	Integer+		+				FK	Sepcialisation	BD

Tabela 5.30: Opis atrybutów relacji Professionals

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>professionalID</i>	Unikalny numer ID identyfikujący profesjonalistę
<i>degree</i>	Stopień naukowy profesjonalisty
<i>detailsID</i>	Numer ID identyfikujący dane personalne profesjonalistę
<i>specialisationID</i>	Numer ID identyfikujący specjalizację profesjonalisty

REL\05 Specialisations\SPECIALISATION

Tabela 5.31: Opis schematu relacji Specialisations

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>specialisationID</i>	Integer+		+			+	PK		SZBD
<i>name</i>	String		+						USER

Tabela 5.32: Opis atrybutów relacji Professionals

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>specialisationID</i>	Unikalny numer ID identyfikujący specjalizację.
<i>name</i>	Nazwa specjalizacji

REL\06 Diets\DIET

Tabela 5.33: Opis schematu relacji Diets

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>dietID</i>	Integer+		+			+	PR		BD
<i>isActive</i>	Boolean		+	true					
<i>horseID</i>	Integer+		+				FK	Horse	BD
<i>nutritionPlanID</i>	Integer+		+				FK	NutritionPlan	BD

Tabela 5.34: Opis atrybutów relacji Diets

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>dietID</i>	Unikalny numer ID identyfikujący daną dietę
<i>isActive</i>	Zmienna przyjmująca wartości true/false
<i>horseID</i>	Numer ID identyfikujący konia
<i>nutritionPlanID</i>	Numer ID identyfikujący plan żywienia

REL\07 Portions\PORTION

Tabela 5.35: Opis schematu relacji Portions

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>portionID</i>	Integer+		+			+	PR		SZBD
<i>amount</i>	Float+		+	1	0<				USER

Tabela 5.36: Opis atrybutów relacji Portions

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>portionID</i>	Unikalny numer ID identyfikujący porcję jedzenia dla konia
<i>amount</i>	Ilość jedzenia w porcji

REL\08 Forges\FORAGE (*forageID*, *name*, *producent*, *capacity*)

Tabela 5.37: Opis schematu relacji Forges

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>forageID</i>	Integer+		+			+	PR		SZBD
<i>name</i>	String		+						USER
<i>producent</i>	String		+						USER
<i>capacity</i>	String		+						USER
<i>unitID</i>	Integer+		+				FK	UnitOfMeasure	BD

Tabela 5.38: Opis atrybutów relacji Forges

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>forageID</i>	Unikalny numer ID identyfikujący paszę
<i>name</i>	Nazwa paszy
<i>producent</i>	Nazwa producenta paszy
<i>capacity</i>	Liczba naturalna oznaczająca ilość paszy w jednej paczce paszy
<i>unitID</i>	Numer ID identyfikujący jednostkę miary

Tabela 5.39: Opis schematu relacji Horses

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>horseID</i>	Integer+		+			+	PR		SZBD
<i>name</i>	Integer+		+						USER
<i>mother</i>	Integer+		+						USER
<i>father</i>	Integer+		+						USER
<i>birthday</i>	Date		+						USER
<i>race</i>	String		+						USER
<i>breeder</i>	String		+						USER
<i>passport</i>	String		+						USER
<i>photo</i>	String		+						USER
<i>statusID</i>	Integer+		+				FK	Status	USER
<i>genderID</i>	Integer+		+				FK	HorseGender	USER

Tabela 5.40: Opis atrybutów relacji Horses

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>horseID</i>	Unikalny numer ID identyfikujący konia
<i>name</i>	Imię konia
<i>mother</i>	Imię matki konia
<i>father</i>	Imię ojca konia
<i>birthday</i>	Data urodzenia
<i>race</i>	Rasa konia
<i>breeder</i>	Nazwa hodowli lub imię i nazwisko hodowcy
<i>passport</i>	Numer paszportu
<i>photo</i>	URL zdjęcia
<i>statusID</i>	Numer ID identyfikujący status konia
<i>genderID</i>	Numer ID identyfikujący płeć konia

REL\10 HorseGenders\HORSEGENDER

Tabela 5.41: Opis schematu relacji HorseGenders

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>genderID</i>	Integer+		+			+	PK		SZBD
<i>gender</i>	String								USER

Tabela 5.42: Opis atrybutów relacji HorseGenders

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>genderID</i>	Unikalny numer ID identyfikujący płeć konia
<i>gender</i>	Nazwa płci

REL\11 Status\STATUS

Tabela 5.43: Opis schematu relacji Specialisations

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>statusID</i>	Integer+		+			+	PK		SZBD
<i>name</i>	String		+						USER

Tabela 5.44: Opis atrybutów relacji Status

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>statusID</i>	Unikalny numer ID identyfikujący status konia bądź użytkownika
<i>name</i>	Nazwa statusu

REL\12 MealNames\MEALNAME

Tabela 5.45: Opis schematu relacji MealNames

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>mealNameID</i>	Integer+		+			+	PK		SZBD
<i>mealName</i>	String		+						USER

Tabela 5.46: Opis atrybutów relacji MealNames

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>mealNameID</i>	Unikatowy numer ID identyfikujący nazwę posiłku
<i>mealName</i>	Nazwa posiłku

REL\13 NutritionPlans\NUTRITIONPLAN

Tabela 5.47: Opis schematu relacji NutritionPlans

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>nutritionPlanID</i>	Integer+		+			+	PK		SZBD
<i>title</i>	String		+						USER
<i>description</i>	String		-						USER
<i>icon</i>	Integer+		+	1					USER

Tabela 5.48: Opis atrybutów relacji NutritionPlans

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>nutritionPlanID</i>	Unikatowy numer ID identyfikujący plan żywienia
<i>title</i>	Tytuł planu żywienia
<i>description</i>	Opis planu żywienia
<i>icon</i>	Id ikonki wyświetlanej koło planu żywienia

Tabela 5.49: Opis schematu relacji PeopleDetails

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>detailsID</i>	Integer+		+			+	PK		SZBD
<i>name</i>	String		+						USER
<i>surname</i>	String		-						USER
<i>phonNumber</i>	String	+_____	+						USER
<i>email</i>	Integer+	_____@_____	+						USER
<i>city</i>	Integer+		+						USER
<i>street</i>	Integer+		+						USER
<i>number</i>	Integer+		+						USER

Tabela 5.50: Opis atrybutów relacji PeopleDetails

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>detailsID</i>	Unikalny numer ID identyfikujący detale ludzi
<i>name</i>	Imie
<i>surname</i>	Nazwisko
<i>phonNumber</i>	Numer telefonu
<i>email</i>	Adres e-mail
<i>city</i>	Miasto
<i>street</i>	Ulica
<i>number</i>	Numer domu

Tabela 5.51: Opis schematu relacji Participations

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>ParticipationID</i>	Integer+		+			+	PK		SZBD
<i>level</i>	Integer+		+						USER
<i>result</i>	String		+						USER
<i>place</i>	String		+						USER
<i>competitionID</i>	Integer+		+						BD
<i>horseID</i>	Integer+		+						BD

Tabela 5.52: Opis atrybutów relacji Participations

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>ParticipationID</i>	Unikalny numer ID identyfikujący start w zawodach.
<i>level</i>	Poziom konkursu, w którym koń brał udział
<i>result</i>	Wynik z danego konkursu
<i>place</i>	Miejsce uzyskane w danym konkursie
<i>competitionID</i>	Numer ID zawodów, w których koń bierze udział
<i>horseID</i>	Numer ID konia biorącego udział w zawodach

Tabela 5.53: Opis schematu relacji Shareds

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>sharedID</i>	Integer+		+			+	PK		SZBD
<i>code</i>	String		+						USER
<i>endDate</i>	Date		+						USER
<i>startDate</i>	Date		+						USER
<i>horseID</i>	Integer+		+						USER
<i>userSharedID</i>	Integer+		+				FK	USER	BD
<i>userScanID</i>	Integer+		+				FK	USER	BD

Tabela 5.54: Opis atrybutów relacji Shareds

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>sharedID</i>	Unikalny numer ID identyfikujący pojedyncze udostępnienie konia między dwoma użytkownikami
<i>code</i>	Kod Qr dzięki któremu użytkownicy mogą udostępniać między sobą konie.
<i>endDate</i>	Data kończąca udostępnianie
<i>startDate</i>	Data od której koń będzie udostępniony
<i>horseID</i>	Numer ID identyfikujący udostępnianego konia
<i>userSharedID</i>	Numer ID identyfikujący użytkownika, który udostępnia konia
<i>userScanID</i>	Numer ID identyfikujący użytkownika, któremu zostanie udostępniony koń

REL\17 UnitOfmeasures\UNITOFMEASURE

Tabela 5.55: Opis schematu relacji UnitOfmeasures

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>unitID</i>	Integer+		+			+			SZBD
<i>unitName</i>	String		+						USER

Tabela 5.56: Opis atrybutów relacji UnitOfmeasures

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>unitID</i>	Unikalny numer ID identyfikujący jednostkę miary
<i>unitName</i>	Nazwa jednostki miary

REL\18 UserTypes\USERTYPE

Tabela 5.57: Opis schematu relacji UserTypes

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>userTypeID</i>	Integer+		+			+	PK		SZBD
<i>typeName</i>	String		+						USER

Tabela 5.58: Opis atrybutów relacji UserTypes

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>userTypeID</i>	Unikalny numer ID identyfikujący typ użytkownika
<i>typeName</i>	Nazwa typu użytkownika

REL\19 UserAccounts\USERACCOUNT

Tabela 5.59: Opis schematu relacji UserAccounts

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>userID</i>	Integer+		+			+	PK		SZBD
<i>accountLogin</i>	String		+			+			USER
<i>hash</i>	String		+						USER
<i>salt</i>	String		+						USER
<i>createdDateTime</i>	Datetime		+						USER
<i>typeID</i>	Integer+		+				FK	UserTypes	BD
<i>detailsID</i>	Integer+		+				FK	PeopleDetails	BD

Tabela 5.60: Opis atrybutów relacji UserAccounts

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>userID</i>	Unikalny numer ID identyfikujący użytkownika
<i>accountLogin</i>	Login użytkownika
<i>hash</i>	Hash powstający z hasła użytkownika
<i>salt</i>	Do hasła użytkownika
<i>createdDateTime</i>	Data utworzenia konta
<i>typeID</i>	Numer identyfikujący typ użytkownika
<i>detailsID</i>	Numer identyfikujący detale osobowe użytkownika

REL\20 Visits\VISIT

Tabela 5.61: Opis schematu relacji Visits

Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>careID</i>	Integer+		+			+	PK		SZBD
<i>cost</i>	Float		+						USER
<i>summary</i>	String		-						USER
<i>image</i>	String		-						USER
<i>visitDate</i>	Date		+						USER
<i>horseID</i>	Integer+		+				FK	Horse	DB
<i>professionalID</i>	Integer+		+				FK	Professional	DB

Tabela 5.62: Opis atrybutów relacji Visits

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>visitID</i>	Unikalny numer ID identyfikujący wizytę
<i>visitDate</i>	Data wizyty
<i>summary</i>	Podsumowanie wizyty, opis przepisanych leków i innych zaleceń
<i>image</i>	Obraz z wizyty
<i>cost</i>	Cena wizyty
<i>horseID</i>	Numer ID identyfikujący konia
<i>professionalID</i>	Numer ID identyfikujący profesjonalistę, który przeprowadza wizytę

REL\21 Meals\MEAL

Tabela 5.63: Opis schematu relacji Meals

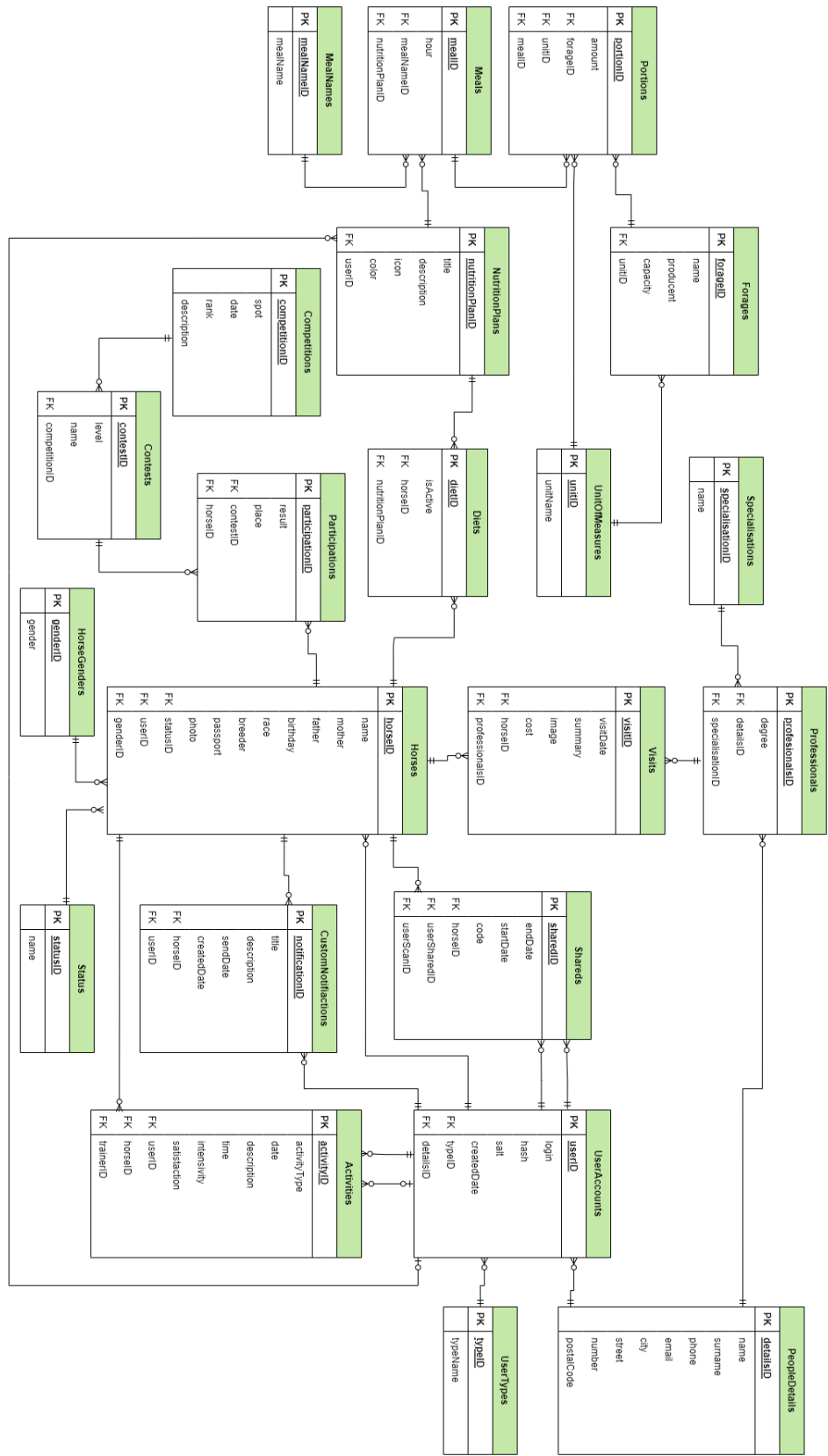
Źródło: Opracowanie własne

Nazwa atrybutu	Dziedzina	Maska	OBL(+) OPC(-)	Wartość domyślna	Ograniczenia	Unikalność	Klucz	Referencje	Źródło danych
<i>mealID</i>	Integer+		+			+			SZBD
<i>hour</i>	String		+						USER
<i>mealNameID</i>	Integer+		+				FK	MealName	BD
<i>nutritionPlanID</i>	Integer+		+				FK	NutritionPlan	BD

Tabela 5.64: Opis atrybutów relacji Meals

Źródło: Opracowanie własne

Nazwa atrybutu	Znaczenie
<i>mealID</i>	Unikalny numer ID identyfikujący posiłek
<i>hour</i>	Godzina, w której jedzony jest posiłek
<i>mealNameID</i>	Numer identyfikujący nazwę posiłku
<i>nutritionPlanID</i>	Numer identyfikujący plan żywienia



Rysunek 5.2: Logiczny schemat bazy danych

Źródło: Opracowanie własne

5.3 Model fizyczny

Po stworzeniu logicznego modelu bazy danych, należy przetransformować go w model fizyczny. Model fizyczny jest to model bazy danych dostosowany do konkretnej bazy. Aby z relacji stworzyć tabele należy dostosować nazwy tabel i atrybutów oraz dopasować do atrybutów odpowiedni typy danych charakterystyczny dla wybranej bazy. Diagram fizyczny bazy danych znajduje się na rysunku rys. 5.3. Po zaprojektowaniu tabel można sporządzić skrypty tworzące bazę danych i tabele widoczne na listingu 5.1.

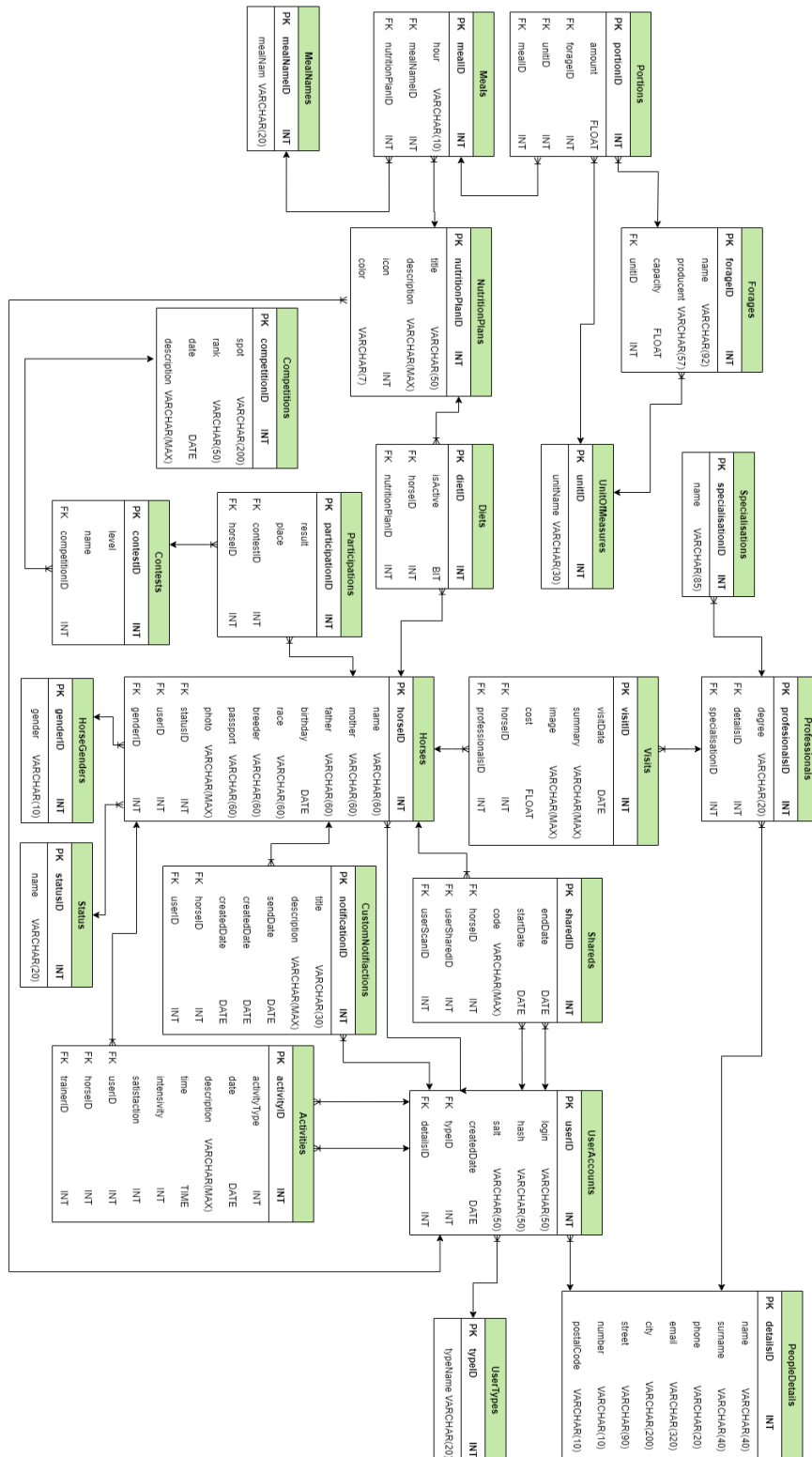
```
— Tworzenie bazy danych
2 CREATE DATABASE HorseTracking;

— Używanie nowo utworzonej bazy danych
4 USE HorseTracking;

— Tworzenie tabel
8 CREATE TABLE UserTypes (
10   typeID INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
   typeName varchar(20) NOT NULL)

12 CREATE TABLE PeopleDetails (
   detailID INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
14   name varchar(40) NULL,
   surname varchar(40) NOT NULL,
16   phone varchar(20) NULL,
   email varchar(320) NULL,
18   city varchar(200) NULL,
   street varchar(90) NULL,
20   number varchar(10) NULL,
   postalCode varchar(10) NULL)
```

Listing 5.1: Fragment skryptu tworzącego bazę danych i tabele



Rysunek 5.3: Fizyczny schemat bazy danych

Źródło: Opracowanie własne

Rozdział 6

Projekt systemu

6.1 Model projektowanego systemu

W tym rozdziale przedstawiony zostanie model projektowanego systemu. Zaprezentowane zostaną przykładowe diagramy aktywności, stanu, klas oraz sekwencji. Przedstawiona zostanie także architektura aplikacji oraz wykorzystane wzorce projektowe.

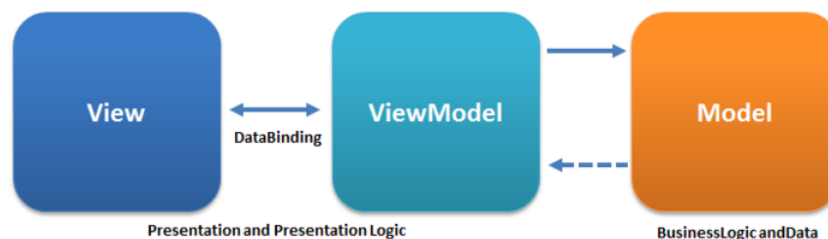
6.1.1 Architektura aplikacji

Jaka baza jakie połączenie itp.

Model architektoniczny MVVM

Jako model architektoniczny aplikacji mobilnej wybrano model MVVM, ponieważ jest on dedykowany aplikacją tego typu. W aplikacji desktopowej również został wybrany ten model ze względu na łatwość jego obsługi oraz wygodną implementację w WPF, który wspiera data binding.

Wzorzec Model-View-ViewModel (MVVM) oddziela logikę i prezentację aplikacji od interfejsu przeznaczonego dla użytkownika. Dzięki tej separacji można rozwiązać wiele problemów z zaprogramowaniem aplikacji. Pomaga ona także w testowaniu, konserwacji oraz rozwoju.[14]



Rysunek 6.1: Diagram przedstawiający architekturę MVVM

Źródło: [13]

Jak można zobaczyć na rysunku 6.1 "View", czyli widok naszej aplikacji odpowiada za to co widzi użytkownik, za wszelkie struktury, układ elementów na ekranie oraz ich wygląd. "ViewModel", czyli model naszego widoku odpowiada za logikę aplikacji. To w nim zapisane są wszystkie komendy oraz metody. "ViewModel" powiadamia widok o wszelkich zmianach jest on odpowiedzialny za aktualizacje modeli. "Model" to klasa, która hermetyzuje dane aplikacji. Zawiera on dane reprezentujące logikę biznesową oraz obiekty transferu danych (DTO). Dzięki zastosowaniu MVVM model i widok są od siebie całkowicie niezależne. [14]

Zalety korzystania z MVVM:

- ułatwia tworzenie testów jednostkowych niezależnych od widoku.
- interfejs użytkownika może być implementowany niezależnie od modelu.
- wiele programistów może pracować na jednej aplikacji niezależnie. Widok może być tworzony niezależnie od modeli i modeli widoku.

Wykorzystane wzorce projektowe

Podczas implementacji aplikacji desktopowej oraz mobilnej wykorzystano wzorce projektowe:

- **Dependency injection** - jest to wzorzec opierający się na paradygmacie "Odwróconego Sterowania". Polega on na tym, że zamiast wywoływania bibliotek przez kod programisty, to framoworki wywołują kod. Przez to że zachowanie między bibliotekami a kodem zostało odwrócone paradygmat nosi nazwę "Odwróconego sterowani". Dzięki wzorcu projektowemu dependency injection możemy uniknąć ciągłego tworzenia nowych instancji serwisów oraz ciągłych zmian kodu w przypadku zmian w serwisach. Zajmuje się on wstrzykiwaniem potrzebnych zależności dzięki specjalnemu serwisowi.[15]

- **Singleton** - jest to wzorzec dzięki któremu wiemy, że istnieje tylko jeden obiekt z danego rodzaju. Korzystając z tego wzorca wiemy, że np. dany serwis ma tylko jedną instancję z której stale korzystamy. Wzorzec ten ma zalety ale także wady, ponieważ ogranicza nam tworzenie testów jednostkowych oraz modularność kodu.

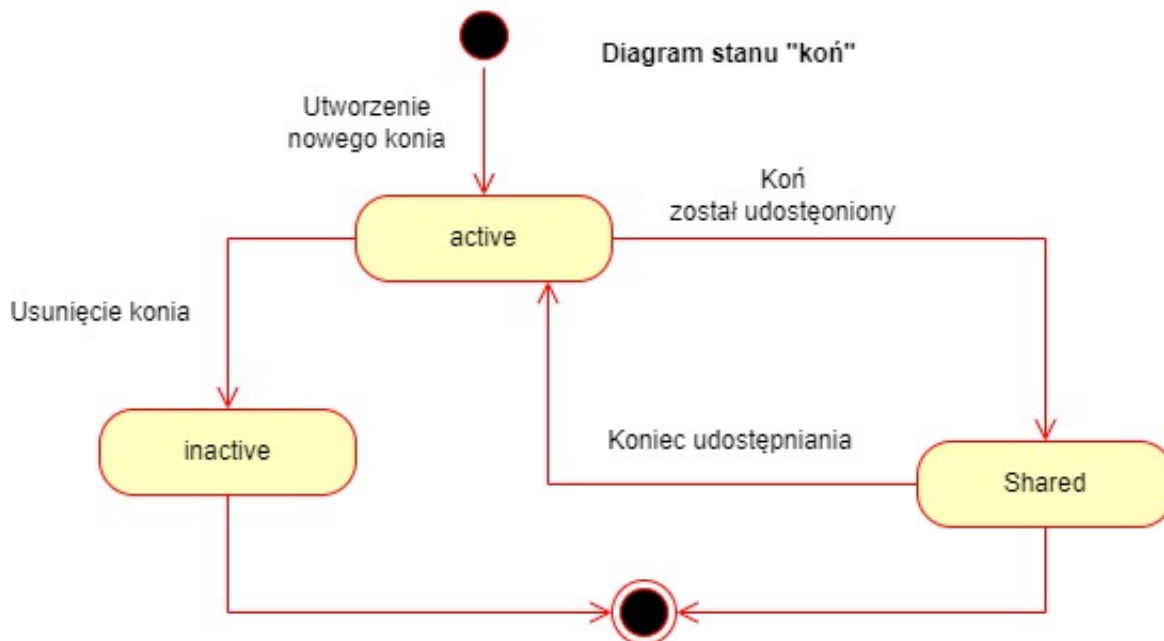
Entity Framework

Entity Framework to maper relacji obiektów, dzięki któremu można utworzyć warstwę dostępu do danych za pomocą .Net i dowolnej bazy danych działającej lokalnie bądź w chmurze. Obsługuje ono zapytania LINQ, aktualizacje, migracje oraz śledzenie zmian. Dzięki niemu możemy z łatwością łączyć aplikację z bazą danych i wykonywać na niej wszelkie operacje. [16]

6.1.2 Diagramy UML

Diagramy UML stanowią kluczowy element procesu projektowania aplikacji, pozwalając na graficzną reprezentację struktury, zachowań oraz relacji między różnymi częściami systemu.

Diagramy stanów



Rysunek 6.2: Diagram stanu - koń

Źródło: Opracowanie własne

Diagramy aktywności

Rysunek 6.3: Diagram aktywności logowanie

Źródło: Opracowanie własne

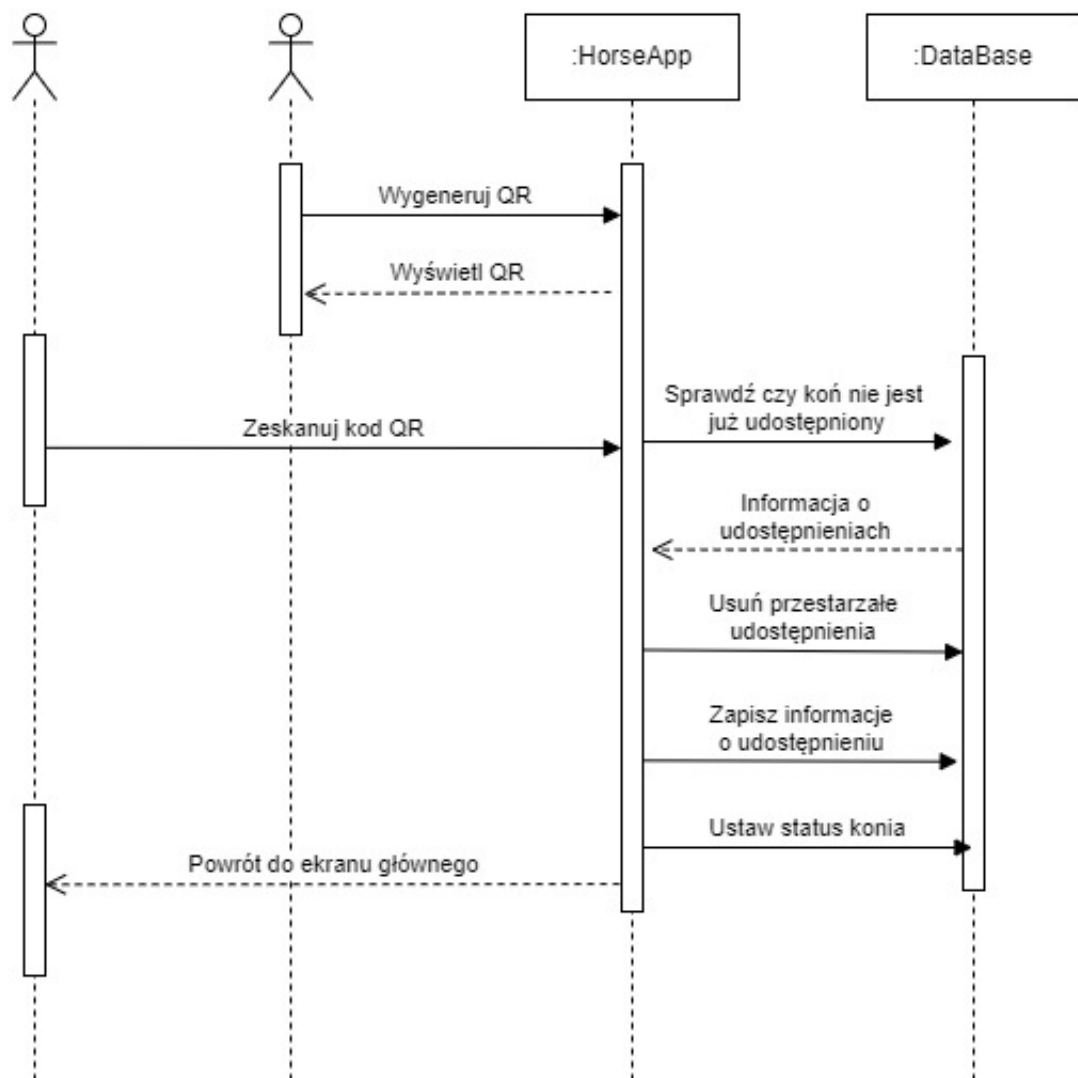
Diagram klas

Rysunek 6.4: Fragment diagramu klas

Źródło: Opracowanie własne

Diagram sekwencji

Diagram sekwencji pozwala na zaprezentowanie w sposób graficzny interakcji między obiektami, przy czym każda interakcja jest umieszczona w czasie. W aplikacji wiele interakcji można zobrazować za pomocą diagramu sekwencji. Jako przykład na poniższym diagramie (rys. 6.5) przedstawione zostało udostępnianie konia.



Rysunek 6.5: Diagram sekwencji - udostępnianie koni

Źródło: Opracowanie własne

6.2 Wybrane aspekty implementacyjne

W tym rozdziale przedstawione zostały wybrane fragmenty implementacji aplikacji mobilnej oraz desktopowej. Zaprezentowana została implementacja architektury aplikacji oraz wzorce projektowe przedstawione wcześniej. Przedstawione zostaną także niektóre z nuggetów opisane wcześniej oraz inne ciekawe aspekty implementacji.

Implementacja architektury MVVM

W aplikacji mobilnej oraz desktopowej zastosowano model architektoniczny MVVM. Implementacja modelu odbywa się poprzez zdefiniowanie DataContext jako odpowiedni view model co możemy zauważyć w listingu 6.1. Listing ten przedstawia fragment pliku xaml, w którym definiujemy paczki i zależności. W 8 linii listingu widzimy zdefiniowanie DataContext jako NutritionPageModel.

```
<Page x:Class="HorseTrackingDesktop.Pages.MainPage.NutritionPage"
2 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
6 xmlns:main="clr-namespace:HorseTrackingDesktop.PageModel.Main"
  xmlns:converter="clr-namespace:HorseTrackingDesktop.Converters"
8 d:DataContext="{d:DesignInstance Type=main:NutritionPageModel}"
  mc:Ignorable="d"
10 d:DesignHeight="450" d:DesignWidth="800"
  Title="NutritionPage">
```

Listing 6.1: Fragment kodu xaml

Dodatkowo, aby przypisać odpowiednie metody do eventu ładowania podpięto także DataContext w pliku .cs widoku (patrz listing 6.3). View model został wstrzyknięty za pomocą Dependency Injection, a następnie przypisany pod DataContext.

```
namespace HorseTrackingDesktop.Pages.MainPage
2 {
    public partial class NutritionPage : Page
4 {
```

```

6      private NutritionPageModel? pageModel;

8      public NutritionPage()
9      {
10         InitializeComponent();
11         pageModel = Startup.ServiceProvider?.GetService<NutritionPageModel>();
12         DataContext = pageModel;
13         if (pageModel != null)
14         {
15             Loaded += async (s, e) => await pageModel.GetPlans();
16         }
17     }
18 }

```

Listing 6.2: Podpięcie DataContext w pliku .cs

Można zauważyć, że w pliku widocznym na listingu 6.3 nie ma żadnych metod dotyczących funkcjonalności aplikacji jest to część implementacji MVVM. Dzięki temu, że podpięty został ViewModel możemy tam umieścić wszystkie te metody a tym samym oddzielić je od widoku.

```

2      namespace HorseTrackingDesktop.Pages.MainPage
3      {
4          public partial class NutritionPage : Page
5          {
6              private NutritionPageModel? pageModel;

7              public NutritionPage()
8              {
9                  InitializeComponent();
10                 pageModel = Startup.ServiceProvider?.GetService<NutritionPageModel>();
11                 DataContext = pageModel;
12                 if (pageModel != null)
13                 {
14                     Loaded += async (s, e) => await pageModel.GetPlans();
15                 }
16             }
17         }
18     }

```

Listing 6.3: Podpięcie DataContext w pliku .cs

Dzięki użyciu tej architektury można uniknąć powtarzania tego samego kodu. Możemy używać jednego ViewModelu do obsługi wielu widoków. W aplikacji zostało to użyte wiele razy, większość widoków dodawania, edytowania oraz szczegółów danego komponentu implementują ten sam ViewModel.

INotifyPropertyChanged

ViewModel musi informować widok o zmianach w różnych zmiennych w nim zaimplementowanych, aby mógł to robić zaimplementowany został interfejs `INotifyPropertyChanged`. Każdy ViewModel musi implementować ten interfejs, dlatego utworzono klasę bazową (patrz listnig 6.4), w której został on zaimplementowany, a wszystkie view modele mogą po niej dziedziczyć.

```
public class BaseViewModel : INotifyPropertyChanged
2 {
    protected bool SetProperty<T>(ref T backingStore, T value,
4     [CallerMemberName] string propertyName = "",
    Action onChanged = null)
6     {
        if (EqualityComparer<T>.Default.Equals(backingStore, value))
8         return false;

        backingStore = value;
        onChanged?.Invoke();
        OnPropertyChanged(propertyName);
        return true;
14    }

    public event PropertyChangedEventHandler PropertyChanged;

18    protected void OnPropertyChanged([CallerMemberName] string propertyName = ""
    )
    {
20        var changed = PropertyChanged;
        if (changed == null)
```

```

22     return;

24     changed.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }

26 }

```

Listing 6.4: Klasa bazowa view modeli

W klasie tej zaimplementowano łatwiejszy dostęp do wywołania eventu **PropertyChangedEventArgs** poprzez wywołanie metody **OnPropertyChanged**(argument), gdzie jako argument podajemy nazwę zmiennej. Zaimplementowano także metodę SetProperty która pozwala na łatwe przypisywanie wartości do zmiennych z jednoczesną aktualizacją ich w widoku użytkownika.

Dependency Injection

W aplikacji użyto Dependency Injection do wstrzykiwania zależności do viewModeli i serwisów. W celu implementacji użyty został nugget Microsoft.Extensions.DependencyInjection w aplikacji mobilnej oraz desktopowej. Utworzono klasę statyczną Startup.cs (patrz listing 6.5) w której zaimplementowano serwis obsługujący wstrzykiwanie.

```

public static class Startup
2 {
    public static IServiceProvider ServiceProvider { get; set; }

4

    public static IServiceProvider Init()
6 {
    var serviceProvider = new ServiceCollection()
8     .ConfigureServices()
    .ConfigureViewModels()
10    .BuildServiceProvider();

12    ServiceProvider = serviceProvider;

14    return serviceProvider;
    }

16 }

```

Listing 6.5: Klasa Startup

Metody konfigurujące serwisy oraz view modele zostały zaimplementowane w klasie DependencyInjectionContainer widoczną na listingu 6.6.

```
1 public static class DependencyInjectionContainer
2 {
3     public static IServiceCollection ConfigureServices(this IServiceCollection
4         services)
5     {
6         services.AddSingleton<IUserService, UserService>();
7         services.AddSingleton<IAppState, AppState>();
8         :
9         return services;
10    }
11
12    public static IServiceCollection ConfigureViewModels(this IServiceCollection
13        services)
14    {
15        services.AddTransient<LoginViewModel>();
16        services.AddSingleton<BaseViewModel>();
17        :
18        return services;
19    }
20 }
```

Listing 6.6: Klasa DependencyInjectionContainer

Można zauważyć, że serwisy zostały zainicjowane jako Singletons, ponieważ potrzebujemy tylko jedną ich instancję, natomiast viewModele jako transient, ponieważ przy każdym ich użyciu chcemy mieć ich nową instancję. Dzięki dependency injection nie trzeba ręcznie tworzyć wszystkich zależności i zależności do ich zależności. Można w łatwy sposób w konstruktorze wstrzyknąć wszystko co jest potrzebne (patrz. listing 6.7).

```
1 public ActivityViewModel(IActivityService activityServices,
2     IAppShellRoutingService appShellRoutingService,
3     IAppState appState, IHorseService horseService)
4 {
5     _activityServices = activityServices;
6     _appShellRoutingService = appShellRoutingService;
7     _appState = appState;
```

```

    _horseService = horseService;
8   :
   }

```

Listing 6.7: Wstrzykiwanie zależności w konstruktorze

Command i RelayCommand

Większość funkcjonalności z aplikacji mobilnej została zaimplementowana jako komendy używając klasy Command, co można zobaczyć na poniższym listingu 6.8.

```

public LoginViewModel(IUserService userService, IAppState appState,
    IHorseService horseService, IShareHorseServices shareHorseServices)
2 {
    _userService = userService;
4   _appState = appState;
    _horseService = horseService;
6   _shareHorseServices = shareHorseServices;

8   LoginCommand = new Command(() =>
    {
10      if (string.IsNullOrEmpty(ReadedLogin) && string.IsNullOrEmpty(
        ReadedPassword))
        return;
12      var hashedPassword = ReadedPassword;
        var user = _userService.GetUser(ReadedLogin, hashedPassword);
14      if (user == null)
        {
16          IncorrectData();
            return;
18      }
        _appState.CurrentUser = user;
20      GoToTheApp();
    });
22 }

```

Listing 6.8: Użycie komend

Zaś w aplikacji desktopowej użyto RelayCommand z paczki nuggetowej (patrz listing 6.9)

```

[RelayCommand]

```



```

2 public async Task GoToAddPlan()
3 {
4     new AddNutritionView().ShowDialog();
5     await GetPlans();
6 }

```

Listing 6.9: Użycie RelayCommand

QRCode

W aplikacji jest możliwość udostępniania koni między użytkownikami. Można to zrobić przez zwykłą wyszukiwarke lub szybciej za pomocą kodu QR. Do udostępniania przez QR zaprojektowano dwa widoki, jeden ze skanerem (patrz listing 6.10), oraz drugi umożliwiający generację kodu.

Scanner zdefiniowany jest w 12 linijce listingu 6.10. Skaner skanuje tylko jeśli zmienna `IsScanning` ustalona jest na `true`. Po zeskanowaniu odpala się event **OnScanResult**, gdzie dostępny jest zeskanowany tekst.

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
2 xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  x:Class="HorseTrackingMobile.Views.ScanHorseView"
4 xmlns:zxing="clr-namespace:ZXing.Net.Mobile.Forms;assembly=ZXing.Net.Mobile.
    Forms"
  xmlns:zxingcommon="clr-namespace:ZXing.Common;assembly=zxing.portable"
6 xmlns:viewmodels="clr-namespace:HorseTrackingMobile.ViewModels"
  x:DataType="viewmodels:ScanHorseViewModel"
8 Shell.TabBarIsVisible="False"
  Shell.FlyoutBehavior="Disabled">
10 <ContentPage.Content>
  <StackLayout>
12     <zxing:ZXingScannerView OnScanResult='ZXingScannerView_OnScanResult'
        IsScanning="{Binding IsScanning}" />
14     </StackLayout>
  </ContentPage.Content>
16 </ContentPage>

```

Listing 6.10: Skaner kodów QR

Aby użytkownik mógł zeskanować kod QR, wcześniej inną osobą musi wygenerować go w aplikacji. Aby to zrobić przygotowany został widok z generatorem kodów QR przedstawiony na listingu 6.11. W widoku tym możliwe jest nie tylko same generowanie, lecz także wybranie udostępnianego konia oraz daty ważności udostępnienia. Po wybraniu obu tych wartości widoczny jest QR kod gotowy do zeskanowania przez innych użytkowników. Generator kodów QR podobnie jak skaner pochodzi z paczki ZXing, zadeklarowanej w 3 linijsce listingu 6.11. Koń na raz może być udostępniony tylko jednej osobie, więc po zeskanowaniu kod staje się ważny.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
2 xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:zxing="clr-namespace:ZXing.Net.Mobile.Forms;assembly=ZXing.Net.Mobile.
    Forms"
4 xmlns:viewmodels="clr-namespace:HorseTrackingMobile.ViewModels"
  xmlns:zxingcommon="clr-namespace:ZXing.Common;assembly=zxing.portable"
6 x:DataType="viewmodels:ShareHorseViewModel"
  x:Class="HorseTrackingMobile.Views.ShareHorseView"
8 Shell.TabBarIsVisible="False"
  Shell.FlyoutBehavior="Disabled">
10 <ContentPage.Content>
  <StackLayout Margin="30,50">
12 <Label Text="Wybierz konia: " />
  <Picker SelectedItem="{Binding SelectedHorse}"
14 ItemsSource="{Binding Horses}"
    ItemDisplayBinding="{Binding Name, Mode=TwoWay}" />
16 <Label Text="Wybierz date: " />
  <DatePicker Date="{Binding EndDate, Mode=TwoWay}" MinimumDate="{Binding
    DateNow}" />
18 <zxing:ZXingBarcodeImageView BarcodeFormat="QR_CODE" BarcodeValue="{Binding
    BarcodeValue}" WidthRequest="500" HeightRequest="500">
  <zxing:ZXingBarcodeImageView.BarcodeOptions>
20 <zxingcommon:EncodingOptions Height="500" Width="500" />
  </zxing:ZXingBarcodeImageView.BarcodeOptions>
22 </zxing:ZXingBarcodeImageView>
  </StackLayout>
24 </ContentPage.Content>
</ContentPage>
```

Kodowanie i odkodowanie

Informacje o udostępnieniu przekazywane w kodach QR są szyfrowane, aby nie możliwe było podrobienie kodu i przejęcie czyjś koni. Informacja przekazywana w kodzie zawiera id konia oraz datę ważności udostępnienia. Gdyby nie była ona zaszyfrowana inny użytkownik mógłby z łatwością wygenerować kody QR pozwalające na udostępnianie różnych koni. Aby zapobiec takiemu nie pożądanemu zjawisku wprowadzono szyfrowanie i odszyfrowanie tekstu z kodu QR (patrz listing 6.12).

```
private byte[] EncryptStringToBytes_Aes(string text, byte[] key, byte[] iv)
2 {
    using (var aesAlg = Aes.Create())
4 {
    aesAlg.Key = key;    aesAlg.IV = iv;
6    var encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);
    using (var msEncrypt = new MemoryStream())
8 {
        using (var csEncrypt = new CryptoStream(msEncrypt, encryptor,
10 CryptoStreamMode.Write))
        {
            using (var swEncrypt = new StreamWriter(csEncrypt))
12 {
                swEncrypt.Write(text);
14            }
        }
16    return msEncrypt.ToArray();
    }
}

18 private string DecryptStringFromBytes_Aes(byte[] text, byte[] key, byte[] iv)
{
20    using (var aesAlg = Aes.Create())
    {
22        aesAlg.Key = key;    aesAlg.IV = iv;
        var decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);
24        using (var msDecrypt = new MemoryStream(text))
```

```

26      {
        using (var csDecrypt = new CryptoStream(msDecrypt, decryptor,
CryptoStreamMode.Read))
        {
28            using (var srDecrypt = new StreamReader(csDecrypt))
                {
30                return srDecrypt.ReadToEnd();
            }
        }
    }
}

```

Listing 6.12: Kodowanie i odkodowanie

Własne kontrolki

Menu boczne dostępne w aplikacji desktopowej zostało stworzone z customowych kontrollek. Przyciski dostępne w menu to przyciski zapewniające dodatkowy routing do określonej strony oraz obrazek. Klasa w której została stworzona kontrolka (patrz listing 6.13) dziedziczy po klasie Button i wykorzystuje wiele jego funkcjonalności.

```

public class NavigationButton : Button
2 {
    static NavigationButton() {
4        DefaultStyleKeyProperty.OverrideMetadata(typeof(NavigationButton), new
FrameworkPropertyMetadata(typeof(NavigationButton))); }

6    public NavigationButton() {
        this.Template = (ControlTemplate)Application.Current.Resources["Template"];
    }

8    public ImageSource ImageSource {
10        get { return (ImageSource)GetValue(ImageSourceProperty); }
        set { SetValue(ImageSourceProperty, value); } }

12    public static readonly DependencyProperty ImageSourceProperty =
        DependencyProperty.Register("ImageSource", typeof(ImageSource), typeof(
        NavigationButton), new PropertyMetadata(null));

14    public Brush Highlight {
16        get { return (Brush)GetValue(HighlightProperty); }
    }
}

```

```

18         set { SetValue(HighlightProperty, value); } }

public static readonly DependencyProperty HighlightProperty =
    DependencyProperty.Register("Highlight", typeof(Brush), typeof(
        NavigationButton), new PropertyMetadata(null));

20

public Uri Routing {
22     get { return (Uri)GetValue(RoutingProperty); }
    set { SetValue(RoutingProperty, value); } }

24

public static readonly DependencyProperty RoutingProperty =
    DependencyProperty.Register("Routing", typeof(Uri), typeof(NavigationButton
    ), new PropertyMetadata(null));

26 }

```

Listing 6.13: Kontrolka NavigationButton

Konwertery

W aplikacji desktopowej zastosowano dwa konwertery dotyczące widoczności obiektów w UI użytkownika. Jeden z nich konwertuje zmienną boolowską prawdę/fałsz na odpowiedni enumerator widoczności, zaś drugi datę. Pierwszy z nich został przedstawiony na listingu 6.14.

```

public class BoolToVisibilityConverter : IValueConverter
2 {
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
4 {
    if (value is bool boolValue && boolValue)
6 {
        return Visibility.Visible;
8     }
    else
10 {
        return Visibility.Collapsed;
12     }
    }
14 }

```

```

    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
16    {
        throw new NotSupportedException();
18    }
    }

```

Listing 6.14: Konwerter

Hash i salt

Aby bezpiecznie przechowywać hasła w bazie danych należy je zaszyfrować. Hasła powinny być zapisane w taki sposób, aby po ich wycieku nie dało się ich odszyfrować. Odpowiednim algorytmem jest algorytm hashujący, ponieważ po zakodowaniu nim nie da się już odszyfrować danych. Paczka wybrana do implementacji używa hash i salt, zapisuje je w jednej zmiennej, aby następnie móc sprawdzić czy podane przez użytkownika hasło jest prawidłowe. Hasło hashujemy po jego utworzeniu oraz po zmianie (patrz listing 6.15)

```

public bool ChangePassword(int id, string newPassword, string oldPassword)
2 {
    var hashNew = PasswordHasher.Hash(newPassword);
4    var user = GetUser(id);
    if (PasswordHasher.Validate(oldPassword, user.Hash))
6    {
        var query = $"UPDATE UserAccounts " +
8        $"SET hash = '{hashNew}' " +
        $"WHERE userID = {id}";

10        var cmd = new SqlCommand(query, _connectionService.GetConnection());
12        cmd.ExecuteReader();
        return true;
14    }
    return false;
16 }

```

Listing 6.15: Hashowanie po zmianie hasła

Po za hasho'waniu haseł są one bezpieczne w bazie. Jednakże sprawdzenie poprawności logowania nie polega teraz jedynie na porównaniu dwóch stringów. Aby sprawdzić poprawność

hasła używamy specjalnej metody pokazanej w linii 8 listingu 6.16.

```
LoginCommand = new Command(() =>
2 {
    if (string.IsNullOrEmpty(ReadedLogin) && string.IsNullOrEmpty(
        ReadedPassword))
4         return;
    var user = _userService.GetUser(ReadedLogin);
6    if (user != null)
    {
8        if (PasswordHasher.Validate(ReadedPassword, user.Hash))
        {
10            _appState.CurrentUser = user;
            GoToTheApp();
12            return;
        }
14    }

16    IncorrectData();
    return;
18 });
```

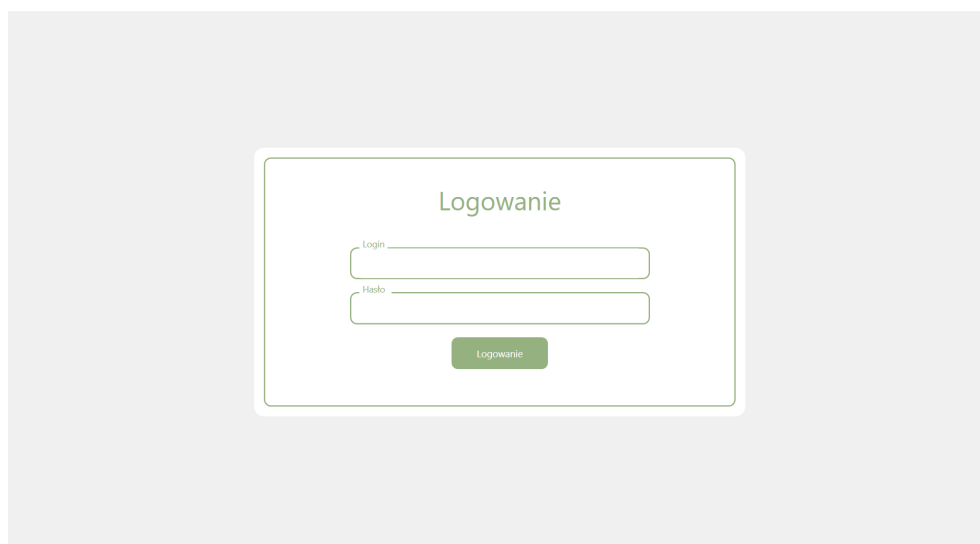
Listing 6.16: Sprawdzenie poprawności hasła

Rozdział 7

Dokumentacja użytkownika

7.1 Aplikacja desktopowa

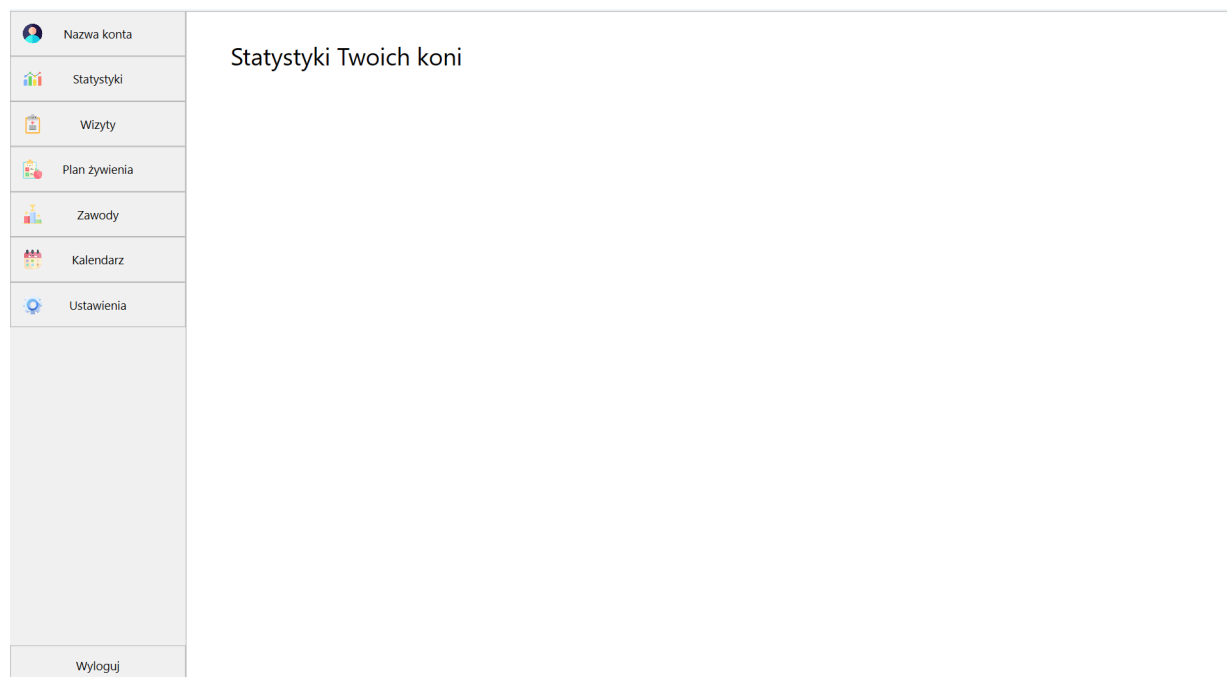
Aplikacja desktopowa "HorseTracking" służy do zarządzania wizytami, żywieniem i zawodami. Można w niej obejrzeć także statystyki z aktywności oraz zawodów. W aplikacji tej dostępny jest także kalendarz z zapisanymi wszystkimi wydarzeniami. Po uruchomieniu użytkownikowi wyświetlony zostaje panel logowania. Każdy użytkownik, posiadający konto w aplikacji może się zalogować. Stworzenie konta, samodzielnie przez użytkownika nie jest możliwe.



Rysunek 7.1: Panel logowania

Źródło: Opracowanie własne

Po zalogowaniu widoczna jest strona statystyk wraz z bocznym menu.



Rysunek 7.2: Strona statystyk

Źródło: Opracowanie własne

Wśród statystyk w lewym górnym rogu widoczne jest miesięczne zestawienie aktywności jednego konia oraz w lewym dolnym wszystkich koni.(Dokończyć jak będzie więcej wiadomo)

Nazwa konta

Statystyki

Wizyty

Plan żywienia

Zawody

Kalendarz

Ustawienia

Wyloguj

Wizyty Twoich koni

Wichawa

Data wizyty	Imię konia	Nazwisko lekarza/kowala	Specjalizacja	Cena
01.03.2023	Wichawa	Tomczyk	Kowal	100
28.08.2023	Wichawa	Dąb	Weterynarz	800
24.08.2023	Wichawa	Dąb	Weterynarz	0
28.08.2023	Wichawa	Tomczyk	Kowal	300

Dodaj

Edytuj

Usuń

Rysunek 7.3: Strona wizyt

Źródło: Opracowanie własne

Po wybraniu wizyt z menu bocznego można obejrzeć wizyty koni. Na górze ekranu widoczny jest pole, dzięki któremu możemy wybrać dla jakiego konia mają być wyświetlane wizyty. Wizyty wyświetlane są w tabeli, posegregowane według malejąco według daty jej odbycia. Na głównym ekranie występują jedynie skrócone informacje na temat wizyty. Dostępna jest data wizyty, imię konia który odbywał wizytę, nazwisko oraz specjalizacja lekarza lub kowala oraz cena wizyty.

Szczegóły konkretnej wizyty można obejrzeć klikając dwa razy na wiersz w tabeli z informacjami o niej. Po kliknięciu pojawi się nowe okno (patrz rys. 7.4) na którym znajduje się nie tylko szczegółowy opis przebiegu wizyty, ale także zdjęcie z wizyty, wszelkie szczegółowe informacje dotyczące konia oraz osoby przeprowadzającej. Po obejrzeniu szczegółów można powrócić do okna głównego klikając przycisk "Wróć".

Na dole głównego ekranu znajdują się trzy przyciski pozwalające na zarządzanie wizytami. Po kliknięciu przycisku dodaj otworzy się nowe okno pozwalające dodać nową wizytę (patrz rys.7.5). W oknie tym możemy wybrać konia, lekarza oraz dodać opis wizyty i zapisać koszt z nią związane. Data wizyty automatycznie ustawiona jest na dzisiejszą, jednakże można ją dowolnie modyfikować. Oprócz przycisku "dodaj" dostępny jest także przycisk "edytuj".

Wizyta 8.28.2023 10:11 PM

Podsumowanie

Lekarz zalecił odpoczynek 2 tygodnie, smarowanie ekyflogylem 2 razy dziennie po jednej pompce oprócz tego spacery po twardym 30 min.

Wichawa
klacz
ur. 3.21.2018
r. małopolak
m. Wiecha
o. Wiewat
hod. SK Prudnik

Cena 300


Zdjęcie z wizyty

Paweł Tomczyk
Kowal
+48789567321
paweltomczyk@gmail.com

ul. Krakowska 15
Opole

Wróć

Dodaj wizytę

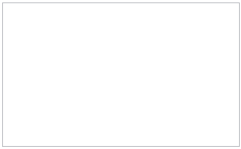
Data wizyty: 28.08.2023 


Specjalista: **Paweł Tomczyk** ▼

Koń: **Wichawa** ▼

Koszt:

Opis (zalecenia, ważne informacje itp.)



Zdjęcie z wizyty 


Dodaj Wróć

Rysunek 7.4: Szczegóły wizyty

Źródło: Opracowanie własne

Rysunek 7.5: Dodawanie wizyt

Źródło: Opracowanie własne



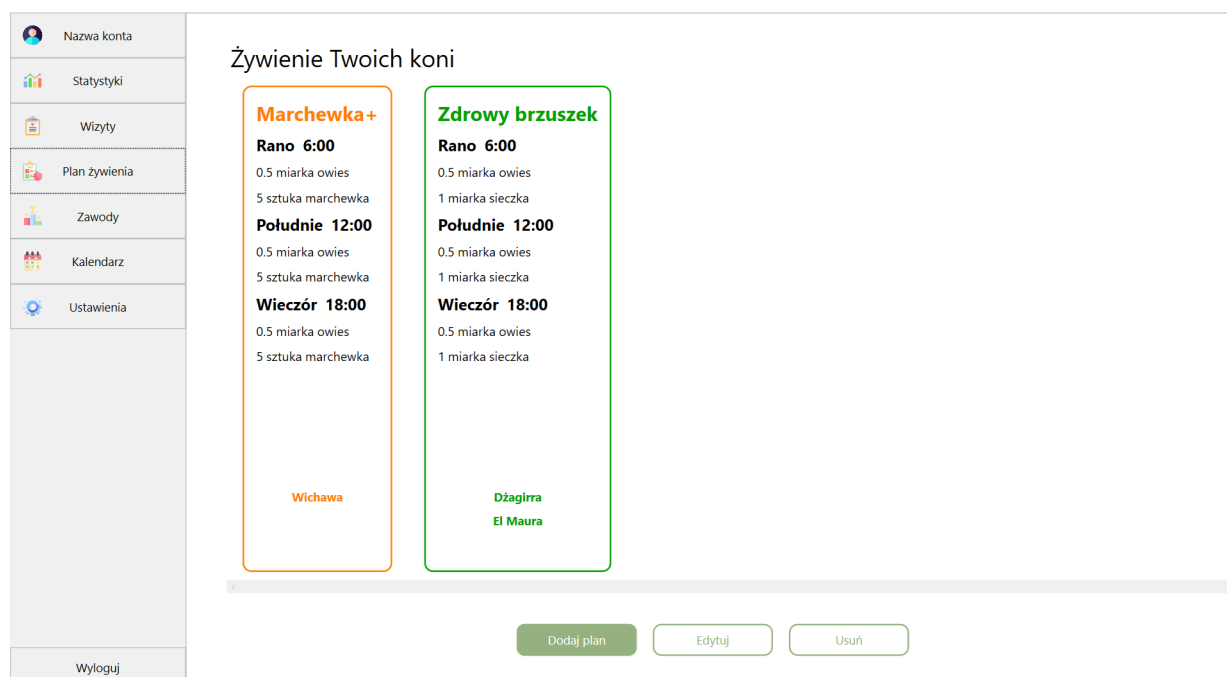
Dodaj Usuń

Potwierdź

Rysunek 7.6: Dodawanie zdjęcia do wizyty

Źródło: Opracowanie własne

Przycisk ten pozwala na edycje obecnie zaznaczonej wizyty. Otwiera on to samo okno co przy dodawaniu, jednakże tym razem widok jest już wypełniony danymi. Ostatni z przycisków, przycisk "usuń" pozwala na usunięcie obecnie zaznaczonej wizyty. Na oknie dodawania wizyt dostępny jest przycisk dodania zdjęcia. Przycisk ten przenosi użytkownika na nowe okno, w którym możliwe jest wgranie zdjęcia z komputera i zapisanie go do bazy danych (patrz rys. 7.6).



Rysunek 7.7: Strona żywienia

Źródło: Opracowanie własne

Na menu bocznym dostępna jest także zakładka "Żywienie" pozwalająca na zarządzanie planami żywienia. Po kliknięciu zostajemy przeniesieni na stronę widoczną na rysunku 7.7. Dla każdego z koni można ustawić jego indywidualny plan, poprzez kliknięcie przycisku „dodaj plan”. Jest on całkowicie modyfikowalny, może zawierać dowolną ilość posiłków podawanych o dowolnej porze dnia, każdy posiłek może zawierać wiele składników w różnych ilościach. Pory posiłków oraz ich nazwy można ustawić po kliknięciu przycisku ustawienia. W tym miejscu można także dodać nowe rodzaje składników posiłku takich jak pasze, warzywa itp.. Wiele koni może być aktualnie na tej samej diecie i stosować ten sam plan żywienia. Na dole planu wypisane są imiona koni, które aktualnie są na tej diecie co pozwala na łatwą identyfikację. Każdy z planów ma swój kolor dzięki czemu użytkownik od razu wie na który

plan spojrzeć. Pory posiłków zostały napisane wytłuszczoną czcionką, co poprawienia czytelności. Na tej stronie można zapisać dowolnie wiele planów żywienia, nie każdy z nich musi być obecnie używany. Jeśli zostanie zapisane więcej danych niż może zmieścić ekran pojawia się suwaki dzięki, którym można obserwować wszystkie zapisane informacje.

Dodaj plan żywienia

Tytuł planu:

Kolor:
Ikona:

Opis:

Nazwa posiłku

▼

Pasza

▼

Miara

▼

Ilość

Pasza

▼

Miara

▼

Ilość

Dodaj kolejny składnik
Usuń ostatni składnik

Dodaj posiłek
Usuń ostatni posiłek

Dodaj

Dodaj plan żywienia

Tytuł planu:

Kolor:
Ikona:

Opis:

Nazwa posiłku

▼

Pasza

▼

Miara

▼

Ilość

Pasza

▼

Miara

▼

Ilość

Dodaj kolejny składnik
Usuń ostatni składnik

Nazwa posiłku

▼

Pasza

▼

Miara

▼

Ilość

Dodaj

Rysunek 7.8: Dodawanie żywienia

Źródło: Opracowanie własne

Rysunek 7.9: Edycja żywienia

Źródło: Opracowanie własne

Jak zostało wcześniej wspomniane, aby dodać kolejne plany należy kliknąć przycisk "dodaj plan". Kliknięcie go powoduje otwarcie nowego okna widocznego na rysunku 7.8. W oknie tym możemy nadać tytuł naszemu planu wybrać kolor, dodać opis oraz stworzyć plan poprzez dodawanie kolejnych składników oraz posiłków. Każdy wybrany przez nas składnik musi

podawany być w odpowiedniej ilości, aby konie zdrowo się odżywiały w tym celu przy każdej paszy którą mamy do wyboru dostępne jest pole do wpisania ilości oraz miejsce w którym możemy wybrać w jakiej miarze została podana ilość.

Możliwa jest także późniejsza edycja planów poprzez kliknięcie przycisku edytuj na stronie głównej. W tym przypadku otwiera się to samo okno co przy dodawaniu planu jednakże jest ono już uzupełnione danymi (patrz. rys 7.9). Po dodaniu bądź edycji planu użytkownik zostanie przekierowany na stronę główną. Jeśli dodany plan go nie satysfakcjonuje można go usunąć klikając przycisk usuń.

Na menu bocznym dostępną mamy także zakładkę zawody. Po kliknięciu w nią użytkownik przeniesiony zostanie na stronę gdzie możliwe jest zarządzanie zawodami. Po wejściu widoczna jest lista zawodów oraz puste miejsce po prawej stronie. Kliknięciu w dowolne zawody dostępne na liście spowoduje pojawienie się po prawej stronie od listy, spisu konkursów dostępnych na tych zawodach. Jeśli jakiś koń jest już zapisany na te zawody jego imię pojawi się pod każdym z konkursów w których bierze udział. Jeśli zawody są już zakończone wyniki konia powinny zostać uzupełnione. Aby dodać nowego konia do zawodów należy kliknąć w

Rysunek 7.10: Strona zawodów

Źródło: Opracowanie własne

nazwę konkursu. Poskutkuje to otwarciem nowego okna z wyborem konia, którego do do-

dania. W oknie tym widoczne będą wszystkie konie danego użytkownika. Po zatwierdzeniu koń zostanie dodany do tych zawodów. Wyniki konia będą puste, ponieważ aplikacja umożliwia także planowanie przyszłych startów. Dopiero zakończonych zawodach można dodać wyniki dowolnego konia uzupełniając pola w wybranych konkursach przy imieniu odpowiedniego konia. W razie pomyłki i nieprawidłowego zapisania wystarczy edytować pole z błędem. Zmiany zapiszą się w bazie automatycznie. aby usunąć start konia w zawodach wystarczy kliknąć przycisk usuń przy jego imieniu.

Dodaj zawody

Miejsce:

Data:

Ranga zawodów:

Opis:

Nazwa konkursu

Poziom konkursu

Dodaj konkurs
Usuń ostatni konkurs

Dodaj

Rysunek 7.11: Dodawanie zawodów

Źródło: Opracowanie własne

Edytuj zawody

Miejsce:

Data:

Ranga zawodów:

Opis:

Nazwa konkursu

Poziom konkursu

Nazwa konkursu

Poziom konkursu

Nazwa konkursu

Poziom konkursu

Nazwa konkursu

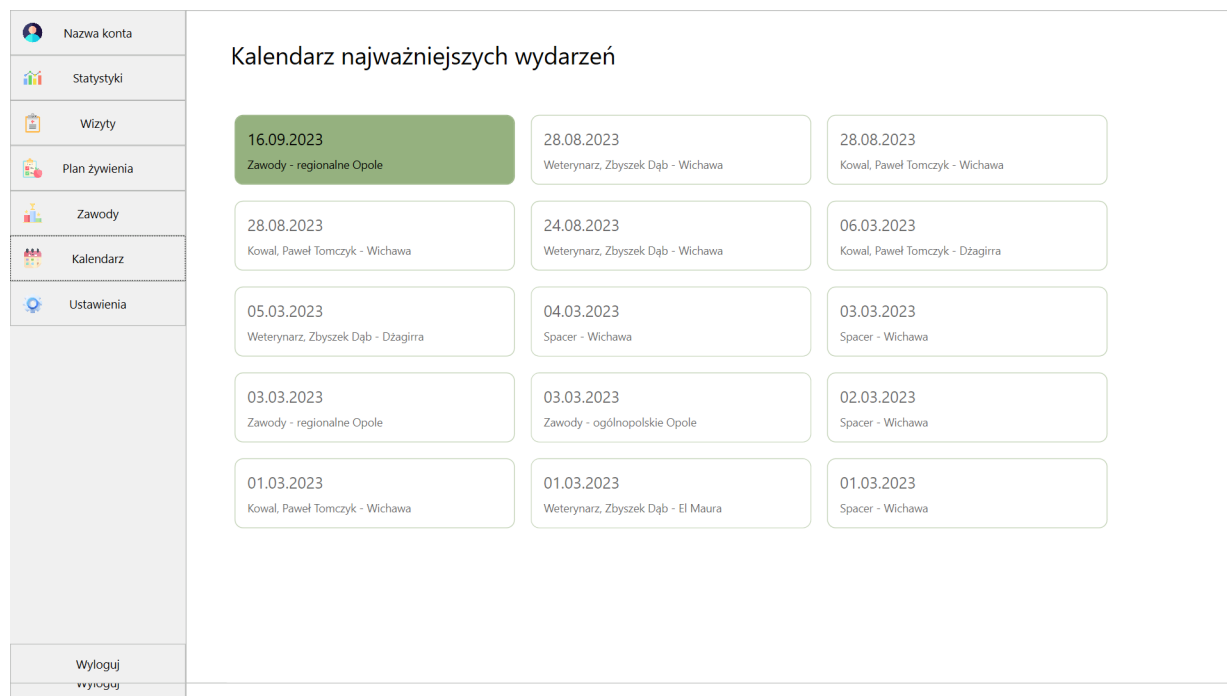
Poziom konkursu

Dodaj

Rysunek 7.12: Edycja zawodów

Źródło: Opracowanie własne

Możliwe jest także dodawanie, edytowanie oraz usuwanie całych zawodów. Dodawanie odbywa się poprzez nowe okno widoczne na rys.7.11. W oknie tym można uzupełnić wszystkie niezbędne informacje, dodać dowolną ilość konkursów oraz zapisać stworzone zawody. Okno edycji widoczne na rys. 7.12 jest praktycznie identyczne co okno dodawania zawodów jednakże przy edycji uzupełnione jest ono wcześniej podanymi danymi.



Rysunek 7.13: Strona z kalendarzem

Źródło: Opracowanie własne

W menu bocznym dostępna jest także ikona kalendarza. Po kliknięciu tej ikony użytkownik zostaje przeniesiony na stronę z widokiem różnych wydarzeń wprowadzonych do aplikacji. W tym miejscu można spojrzeć kiedy koń miał ostatnio wizytę, jakie aktywności wykonywał kiedy są jakie zawody. Wszystkie informacje z różnych zakładki aplikacji dostępne w skróconej formie na jednym oknie. Kalendarz jest przejrzysty zawiera nie wiele szczegółów, a wydarzenia w nim dostępne posegregowane są od najnowszego do najstarszego. Po naciśnięciu myszką na konkretny kafelek z listy staje się on wyraźniejszy oraz jego tło zmienia się na zielone. Z tego widoku nie możemy edytować żadnych wizyty bądź aktywności. Jest to widok przeznaczony tylko i wyłącznie do wyświetlania informacji.

Nazwa konta

Statystyki

Wizyty

Plan żywienia

Zawody

Kalendarz

Konta

Ustawienia

Wyloguj

Zarządzaj:

Użytkownikami

Końmi

Specjalistami

Login	Email	Data utworze	Imię	Nazwisko	Typ	Ulica	Numer	Kod pocztowy	Miasto	Tel. num
admin	asianowak@gm	1/1/2023 12:00	Asia	Nowak	admin	Oleska	12		Opole	+48123890123
appowner	karolinakowalsk	1/1/2023 12:00	Karolina	Kowalska	appOwner	Katowicka	22A		Opole	+48234890567
horseowner	kasiagrab@gm	1/1/2023 12:00	Kasia	Grab	horseOwner		12		Moszna	+48678123098
trener1	tomek11@gma	1/1/2023 12:00	Tomek	Kowalik	trainer	Opolska	12		Leszno	+48456789321
trener2	paweltomczyk@	1/1/2023 12:00	Paweł	Tomczyk	trainer	Krakowska	15		Opole	+48789567321
horseowner2	zbyszekdab@gr	9/24/2023 12:00	Zbyszek	Dąb	horseOwner	Dąbrowskiego	22		Prudnik	+48876543876

Resetuj hasło

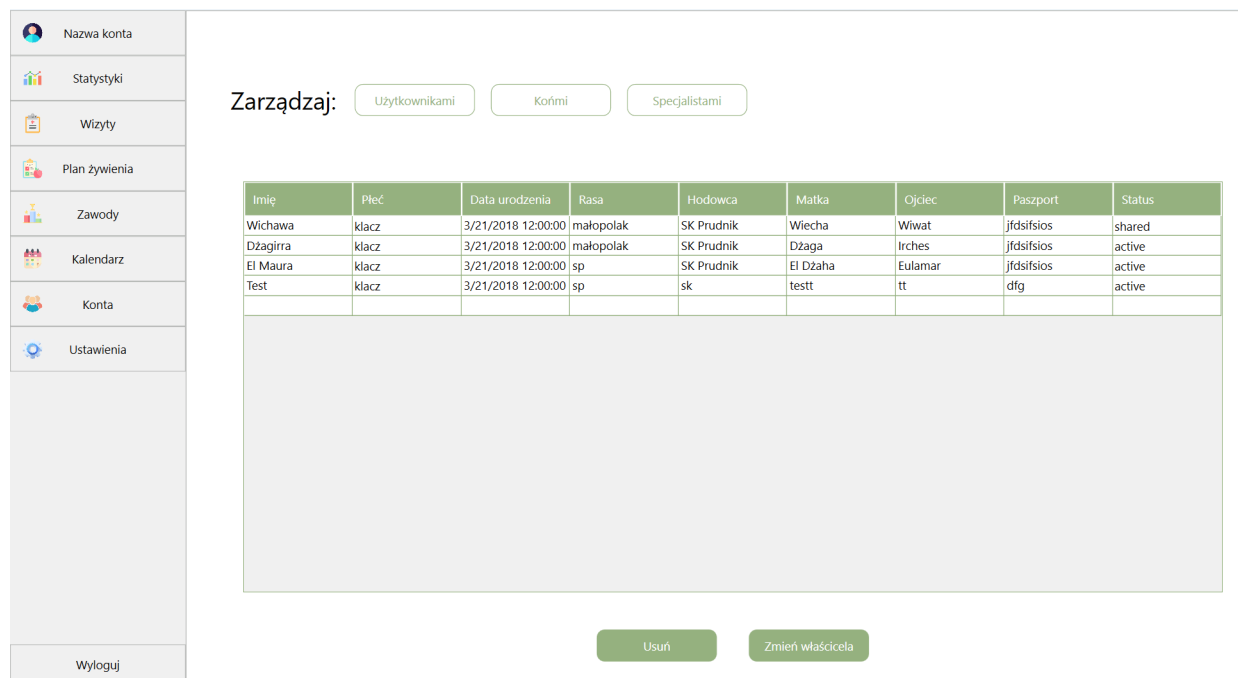
Rysunek 7.14: Zarządzanie użytkownikami

Źródło: Opracowanie własne

Zakładka konta jest dostępna jedynie dla użytkownika o prawach administratora. Po wejściu w tą zakładkę możliwe jest zarządzanie użytkownikami (patrz rys.7.14). Dodawanie nowych użytkowników poprzez wpisanie ich danych do tabeli, edycja tych danych, bądź usunięcie poprzez naciśnięcie klawisza delete. Dostępny jest także przycisk pozwalający na zresetowanie hasła użytkownika widoczny poniżej tabeli.

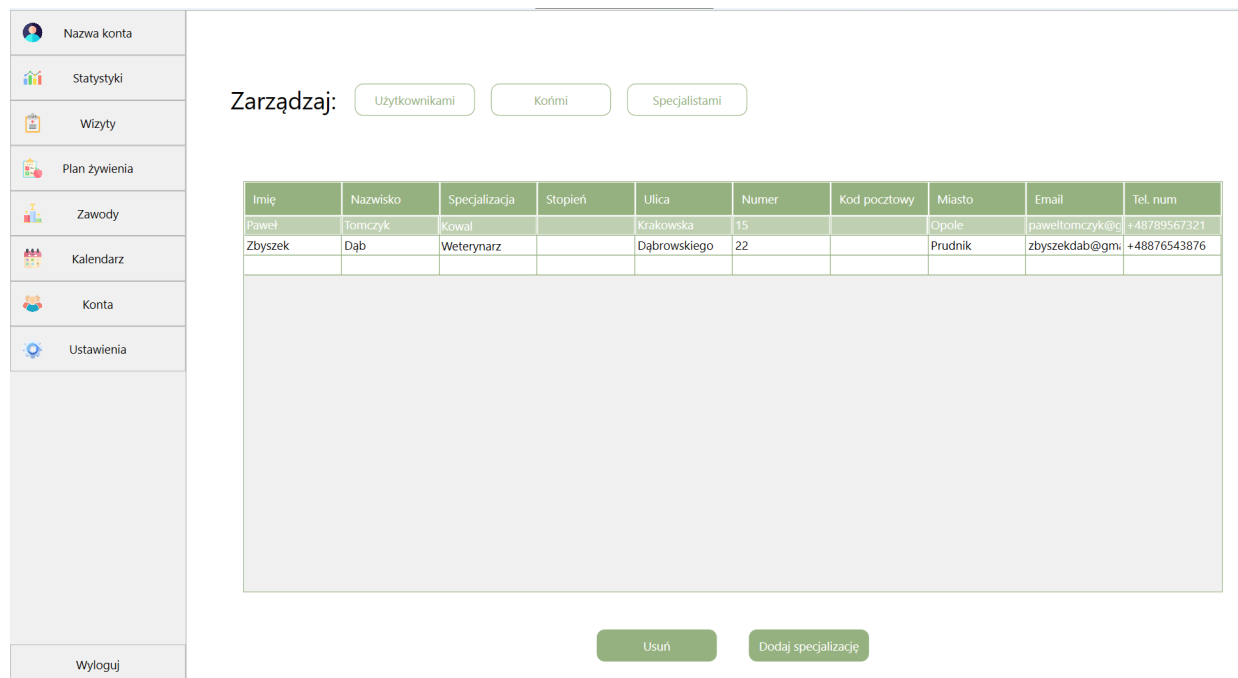
Z tej zakładki możemy także zarządzać końmi (patrz rys.7.15) oraz lekarzami, kowalami (patrz rys.7.16) dostępnymi w naszej aplikacji. W przypadku koni usunięcie konia możliwe jest poprzez kliknięcie przycisku usuń po niżej tabeli. Obok tego przycisku dostępny jest także przycisk zmień właściciela, pozwalający na przypisanie konia do innego użytkownika poprzez małe okno dialogowe. W zarządzaniu specjalistami jakimi są lekarze oraz kowale także możliwe jest dodawanie, edytowanie oraz usuwanie danych. Dwie pierwsze dostępne są poprzez uzupełnienie tabeli, zaś usuwanie odbywa się po naciśnięciu przycisku. Możemy także dodawać nowe specjalizacje poprzez kliknięcie przycisku "dodaj specjalizacje".

Wszystkie tabele są sformatowane i dostosowane do danych przechowywanych w aplikacji. Podczas dodawania nowych obiektów część pól tabeli jest zwykły polem tekstowym, a część jest dostępna jako rozwijane listy z predefiniowanymi elementami.



Rysunek 7.15: Zarządzanie końmi

Źródło: Opracowanie własne



Rysunek 7.16: Zarządzanie lekarzami, kowalami itp

Źródło: Opracowanie własne

Ustawienia

7.2 Aplikacja mobilna

Aplikacja mobilna "HorseTracking" służy do zapisywania dziennych aktywności koni, ich wizyt u lekarzy, kowali jak także do zarządzania zawodami.

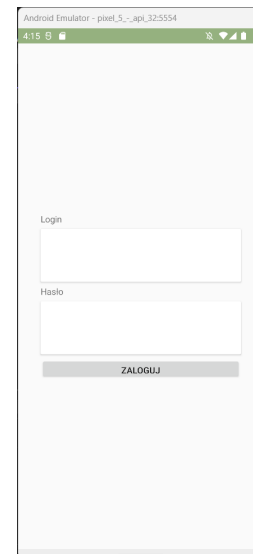
Po pierwszym otwarciu aplikacji użytkownikowi ukaże się okno logowania przedstawione na rysunku 7.17.

Na tym ekranie użytkownik może zalogować się do aplikacji. Możliwa jest także opcja zresetowania swojego hasła, w przypadku zapomnienia hasła poprzez zgłoszenie tego administratorowi. Rejestracja do aplikacji nie jest możliwa, ponieważ ta funkcja jest dostępna jedynie dla administratora w aplikacji desktopowej. Po wprowadzeniu loginu i hasła, a następnie kliknięciu przycisku "Zaloguj" sprawdzana jest poprawność wprowadzonych danych.

Jeśli przy próbie zalogowania podane zostaną nieprawidłowe dane logowania, bądź użytkownika nie ma w systemie, zostanie on o tym poinformowany (patrz. rys 7.18) Użytkownik nieposiadający koni (użytkownik typu appOwner) nie może zalogować się do aplikacji mobilnej, ponieważ służy ona tylko do wpisywania danych o swoich koniach.

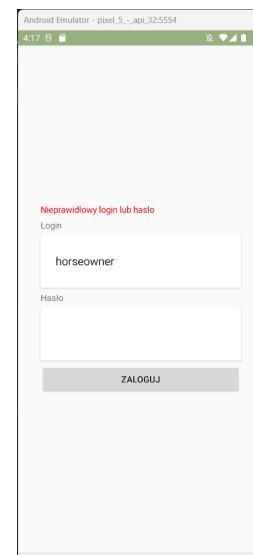
Po poprawnym zalogowaniu się dane użytkownika zostają zapamiętane, więc przy kolejnym otwarciu aplikacji użytkownik będzie już zalogowany. Aby wylogować się z aplikacji użytkownik musi otworzyć menu boczne i wybrać opcję "Wyloguj".

Po zalogowaniu do aplikacji użytkownik



Rysunek 7.17: Logowanie do aplikacji.

Źródło: Opracowanie własne



Rysunek 7.18: Błędne dane logowania.

Źródło: Opracowanie własne

zostaje przeniesiony na okno główne. Zawiera ono menu dolne pozwalające na nawigację pomiędzy czterema głównymi sekcjami aplikacji: aktywności, wizyty, żywienie, zawody. Na początek wyświetlona zostaje strona dotycząca aktywności. W tym widoku można przeglądać informacje o wszystkich aktywnościach koni posiadanych lub tych które trenujemy.

Aktywności dotyczą konia, którego imię podane jest w polu powyżej. Aby zmienić konia wystarczy kliknąć w to pole i wybrać innego konia. Strzałki lewo-prawo widoczne na dole ekranu umożliwiają nawigację między kolejnymi tygodniami. W przypadku braku aktywności w danym dniu, wyświetlony jest napis informujący o braku aktywności tego dnia. Każdy typ aktywności ma inny kolor i ikonę, aby ułatwić identyfikację. Po kliknięciu w daną aktywność możemy przejść do detali dotyczących tej aktywności. Ekran szczegółów został przedstawiony na rysunku 7.22 i zostanie omówiony później.

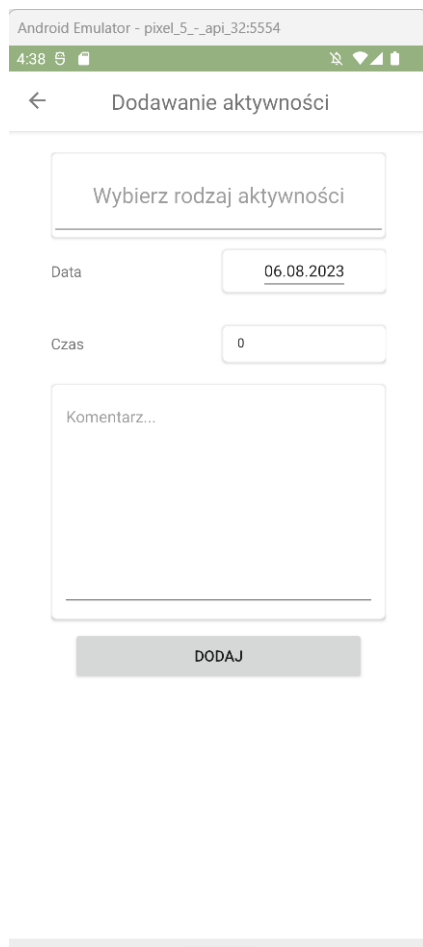
Pomiędzy strzałkami nawigującymi między tygodniami znajduje się okrągły przycisk z ikoną "+". Umożliwia on dodawanie aktywności dla aktualnie wybranego konia. Przycisk ten dostępny jest jedynie dla właściciela konia oraz osób którym dany koń został udostępniony. Oznacza to, że jeśli użytkownik jest trenerem ta opcja jest dla niego zablokowana, a przycisk nie jest widoczny.



Rysunek 7.19: Ekran aktywności.

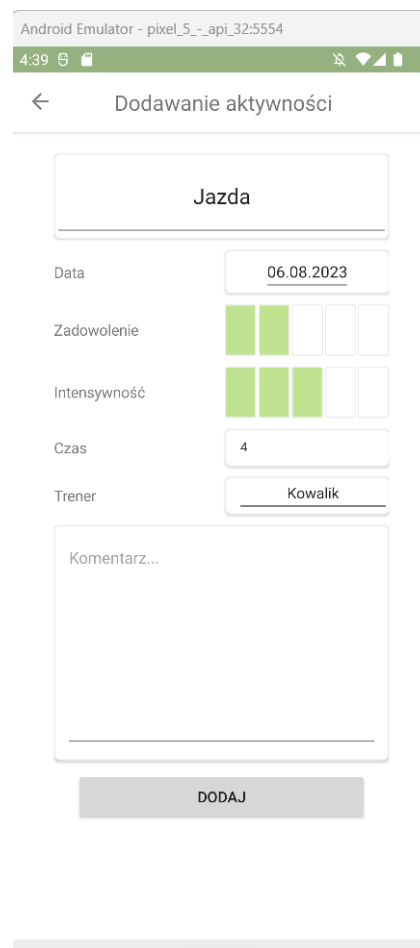
Źródło: Opracowanie własne

Po kliknięciu w przycisk "+" użytkownik zostaje przeniesiony na okno "Dodawanie aktywności".



Rysunek 7.20: Okno dodawania aktywności.

Źródło: Opracowanie własne



Rysunek 7.21: Okno dodawania aktywności rozszerzone.

Źródło: Opracowanie własne

Okno to wygląda różnie w zależności od tego jaki typ aktywności chcemy dodać (patrz rys. 7.20 i rys. 7.21). Na początku wyświetlony jest prostszy model okna, a po wybraniu typu aktywności dostosowuje się. Dla aktywności: jazda, skoki, zawody, kros czy skoki w oknie dochodzą nowe opcje takie jak satysfakcja, intensywność oraz wybór trenera (patrz. rys 7.21). Po uzupełnieniu wszystkich niezbędnych informacji aktywność zostaje dodana i pojawia się na

ekranie głównym. Jeśli któraś z niezbędnych informacji nie zostanie uzupełniona aktywność nie doda się, użytkownik zostanie poinformowany o nieprawidłowościach i będzie mógł je poprawić.

Po dodaniu aktywności użytkownik zostaje przeniesiony z powrotem na okno główne, gdzie po kliknięciu w wybraną aktywność może zobaczyć jej szczegóły (patrz rys.7.22).

W oknie szczegółów oprócz przeczytania wszystkich informacji dotyczących aktywności można także przejść do edycji lub usunąć daną aktywność. Przy edycji otwiera się to samo okno co przy dodawaniu aktywności jednakże tym razem jest ono wypełnione aktualnymi danymi wybranej aktywności. Po zakończonej edycji użytkownik zostaje przeniesiony na okno główne. Po kliknięciu przycisku usuń, wyświetla się komunikat proszący o potwierdzenie wykonania akcji. Jeśli użytkownik potwierdzi, że akcje, to aktywność zostanie usunięta, a użytkownik zostanie przeniesiony na ekran główny. **Usunięcie aktywności jest także możliwe poprzez długie przytrzymanie wybranej aktywności na ekranie głównym, a następnie potwierdzenie akcji na pojawiającym się komunikacie.**

The screenshot shows an Android emulator window titled 'Android Emulator - pixel_5_-_api_32:5554'. The app interface has an orange header with a back arrow, the title 'Skoki', and a silhouette of a horse and rider. Below the header, the date '26.06.2023' is displayed. The main content area contains four data boxes: 'Satysfakcja: 3' and 'Intensywność: 0'; 'Czas trwania: 3'; 'Trener: Kowalik'; and 'Opis:'. At the bottom, there are two orange buttons: 'EDYTUJ' and 'USUŃ'.

26.06.2023	
Satysfakcja:	3
Intensywność:	0
Czas trwania:	3
Trener:	Kowalik
Opis:	

EDYTUJ USUŃ

Rysunek 7.22: Szczegóły aktywności.

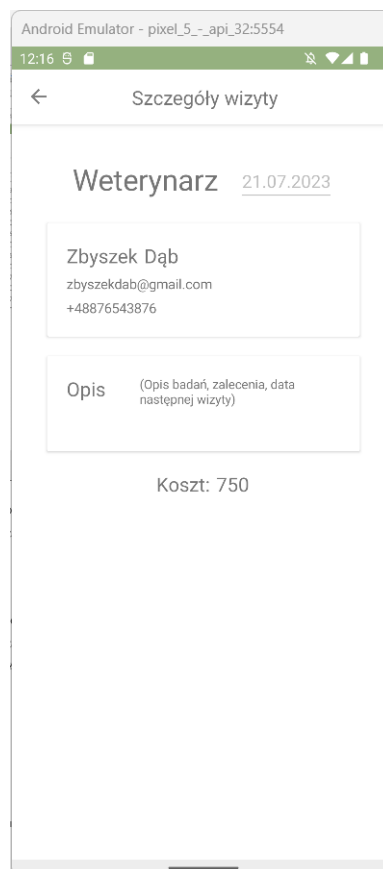
Źródło: Opracowanie własne

Kolejną opcją w menu dolnym są wizyty. Na tym oknie podobnie jak w oknie aktywności mamy pole pozwalające wybrać konia o którym informacje chcemy obejrzeć. Koń wybrany na oknie aktywności przenosi się na okno wizyt i na odwrót. W tym oknie można sprawdzić jakie wizyty odbył ostatnio wybrany koń.



Rysunek 7.23: Okno wizyt.

Źródło: Opracowanie własne



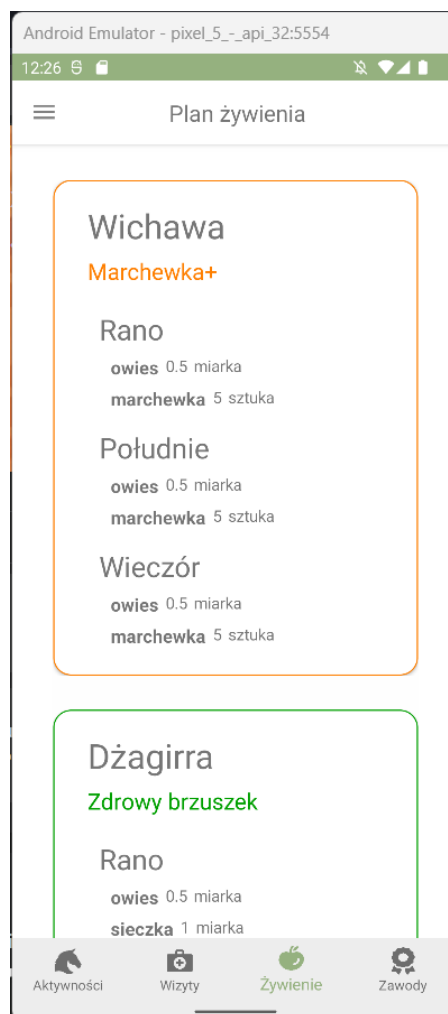
Rysunek 7.24: Szczegóły wizyty.

Źródło: Opracowanie własne

Na rysunku 7.23 widzimy listę wizyt wybranego konia. Po kliknięciu w wizytę zostaniemy przeniesieni do szczegółów wizyty, gdzie możemy znaleźć dane kontaktowe do lekarza/kowala, który przeprowadził wizytę oraz szczegóły takie jak opis wizyty i jej koszt. Dzięki temu widokowi użytkownik może sprawdzić jakie zalecenia były na poprzednich wizytach, jaki był ich koszt i kiedy dokładnie się odbyły.

Kolejną opcją dostępną w menu dolnym są plany żywienia. Po kliknięciu w ikonę jabłka użytkownik zostanie przeniesiony na stronę z planami żywienia wszystkich jego koni. Plany żywienia przedstawione zostały na liście, którą możemy scrollować, aby zobaczyć wszystkie plany dotyczące naszych koni. W widoku tym można jedynie obejrzeć plany żywienia (patrz 7.25), nie jest możliwe ich dodanie, edycja bądź usunięcie. Całość zarządzania planami żywienia została zaimplementowana w aplikacji desktopowej. Dzięki tej zakładce użytkownicy przebywający w stajni mogą sprawdzić dietę konia i przygotować dla niego posiłki. Plany zostały przedstawione w przejrzysty sposób, gwarantujący możliwość szybkiego sprawdzenia diety konia. Kolory oraz tytuły zastosowane w poszczególnych planach są ściśle z nimi związane i można je ustalić przy tworzeniu planu, dzięki nim po jednym spojrzeniu wiemy, który plan dotyczy którego konia.

Ostatnia zakładka dostępna w menu dolnym z ikoną nagrody "flo" rozety rozdawanej na zawodach, przenosi nas do strony, na której możemy obejrzeć informacje dotyczące zawodów. Na tej stronie widzimy listę startów naszych koni w różnych zawodach. Starty te zostały posegregowane według daty od najwcześniejszego do najbardziej odległego w czasie. Jeśli zawody już się zakończyły i uzupełnione zostały wyniki z tych zawodów to one także pojawią się na tej liście. Okno zawodów można zobaczyć na rysunku 7.26.



Rysunek 7.25: Plany żywienia.

Źródło: Opracowanie własne



Rysunek 7.26: Okno przedstawiające starty w zawodach.

Źródło: Opracowanie własne

W aplikacji mobilnej oprócz menu dolnego, które prowadzi nas przez najważniejsze funkcjonalności aplikacji mamy także menu boczne. W menu bocznym mamy możliwość wylogowania się z aplikacji. Wyjście z aplikacji nie powoduje wylogowania, użytkownik po wpisaniu swoich danych zostaje zapamiętany i po wejściu do aplikacji będzie już zalogowany na swoje konto. Dlatego w menu bocznym dodany został przycisk powodujący wylogowanie, dzięki czemu inna osoba może zalogować się na swoje konto. Oprócz tego w menu bocznym znajdują się przycisk pozwalający na przejście do panelu udostępniania (patrz rys. 7.27).



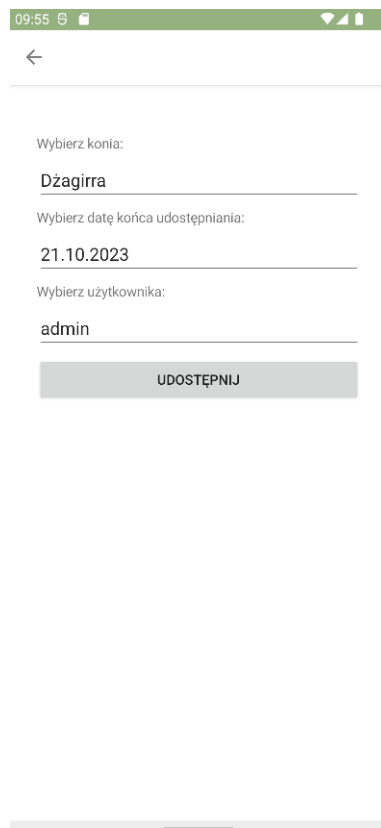
Rysunek 7.27: Panel udostępniania.

Źródło: Opracowanie własne

W tym miejscu można przejść do udostępniania przez kod QR. Gdzie po wybraniu konia do udostępnienia oraz daty do której ma być udostępniony możemy wygenerować kod QR (patrz rys. 7.28). Dzięki któremu inna osoba posiadająca aplikację będzie mogła go zeskanować i dodać do swojej puli koni. Można także udostępnić konia danemu użytkownikowi poprzez wyszukanie go na liście wszystkich użytkowników (patrz rys. 7.29). Po udostępnieniu koń znika z listy koni właściciela, powróci na nią dopiero po zakończeniu daty udostępniania, bądź po kliknięciu przycisku "Usuń udostępnienia". Osoba która dostała konia nie może go usunąć samodzielnie ze swojej listy.



Rysunek 7.28:
 Udostępnianie koni przez
 QR.
*Źródło: Opracowanie
 własne*



Rysunek 7.29:
 Udostępnianie koni przez
 wyszukiwarkę.
*Źródło: Opracowanie
 własne*

Rozdział 8

Podsumowanie

Bibliografia

- [1] Hanna Mazur, Zygmunt Mazur, *Projektowanie relacyjnych baz danych*. Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2004.
- [2] profexorgeek, alexbuckgit, v-hearya, davidbritch, conceptdev *Co to jest środowisko Xamarin?* <https://learn.microsoft.com/pl-pl/xamarin/get-started/what-is-xamarin> [Dostęp: 06.08.2023]
- [3] JonDouglas, alexbuckgit, Mikejo5000, v-hearya, zivkan, chrisraygill, loic-sharma, karannmsft, NickKruger, mairaw, kraigb, alfredmyers, *Wprowadzenie do narzędzia NuGet* <https://learn.microsoft.com/pl-pl/nuget/what-is-nuget> [Dostęp: 06.08.2023]
- [4] Paweł Łukasiewicz *C# - Entity Framework* <https://www.plukasiewicz.net/Artykuly/EntityFramework> [Dostęp: 06.08.2023]
- [5] Juris Lavrinovics, *Figma - narzędzie do projektowania interfejsu użytkownika* <https://blog.consdata.tech/2023/02/15/uiux-tools.html> [Dostęp 05.05.2023]
- [6] <https://www.figma.com/about/> [Dostęp 05.05.2023]
- [7] Figma *Figma-logo* <https://commons.wikimedia.org/wiki/File:Figma-logo.svg> [Dostęp 05.05.2023]
- [8] adegeo, ihsansfd, alexbuckgit, v-trisshores, DCtheGeek, *Przewodnik dotyczący aplikacji klasycznych (WPF .NET)* <https://learn.microsoft.com/pl-pl/dotnet/desktop/wpf/overview/?view=netdesktop-7.0> [Dostęp: 06.08.2023]
- [9] Sharon Allen, Evan Terry, *Beginning Relational Data Modeling*, Apress Berkeley, CA, 2005

- [10] GIT *Git –local-branching-on-the-cheap* <https://git-scm.com/> [Dostęp 10.10.2023]
- [11] GitHub - <https://github.com/> [Dostęp 10.10.2023]
- [12] Sourcetree - <https://www.sourcetreeapp.com/> [Dostęp 10.10.2023]
- [13] MVVM Pattern <https://en.wikipedia.org/wiki/Model-view-viewmodel> [Dostęp 12.10.2023]
- [14] davidbritch, nschonni, conceptdev, JohnCOsborne *The Model-View-ViewModel Pattern* <https://learn.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm> [Dostęp 14.10.2023]
- [15] Michał Miszczyszyn *Wzorce Projektowe: Dependency Injection* <https://typeofweb.com/wzorce-projektowe-dependency-injection> [Dostęp 14.10.2023]
- [16] Microsoft Learn *Dokumentacja EntityFramework* <https://learn.microsoft.com/pl-pl/ef/> [Dostęp 14.10.2023]

Spis rysunków

2.1	Widok główny aplikacji	2
2.2	Widok kalendarza aplikacji	3
2.3	Widok kalendarza aplikacji	4
3.1	Architektura platformy Xamarin	7
3.2	Architektura platformy Xamarin.Android	8
3.3	Logo systemu NuGet	8
3.4	Przepływ informacji	9
3.5	Logo git	9
3.6	Logo github	10
3.7	Logo sourcetree	10
3.8	Logo Figmy	11
3.9	przepływ informacji	11
4.1	Diagram Use Case dla aplikacji desktopowej	19
4.2	Diagram Use Case dla aplikacji mobilnej	20
5.1	Diagram ERD	44
5.2	Logiczny schemat bazy danych	63
5.3	Fizyczny schemat bazy danych	65
6.1	Diagram przedstawiający architekturę MVVM	67
6.2	Diagram stanu - koń	69
6.3	Diagram aktywności logowanie	69
6.4	Fragment diagramu klas	69
6.5	Diagram sekwencji - udostępnianie koni	70
7.1	Panel logowania	84

7.2	Strona statystyk	85
7.3	Strona wizyt	86
7.4	Szczegóły wizyty	87
7.5	Dodawanie wizyt	87
7.6	Dodawanie zdjęcia do wizyty	87
7.7	Strona żywienia	88
7.8	Dodawanie żywienia	89
7.9	Edycja żywienia	89
7.10	Strona zawodów	90
7.11	Dodawanie zawodów	91
7.12	Edycja zawodów	91
7.13	Strona z kalendarzem	92
7.14	Zarządzanie użytkownikami	93
7.15	Zarządzanie końmi	94
7.16	Zarządzanie lekarzami, kowalami itp	94
7.17	Logowanie do aplikacji.	96
7.18	Błędne dane logowania.	96
7.19	Ekran aktywności.	97
7.20	Okno dodawania aktywności.	98
7.21	Okno dodawania aktywności rozszerzone.	98
7.22	Szczegóły aktywności.	99
7.23	Okno wizyt.	100
7.24	Szczegóły wizyty.	100
7.25	Plany żywienia.	101
7.26	Okno przedstawiające starty w zawodach.	102
7.27	Panel udostępniania.	103
7.28	Udostępnianie koni przez QR.	104
7.29	Udostępnianie koni przez wyszukiwarę.	104

Listings

5.1	Fragment skryptu tworzącego bazę danych i tabele	64
6.1	Fragment kodu xaml	71
6.2	Podpięcie DataContext w pliku .cs	71
6.3	Podpięcie DataContext w pliku .cs	72
6.4	Klasa bazowa view modeli	73
6.5	Klasa StartUp	74
6.6	Klasa DependencyInjectionContainer	74
6.7	Wstrzykiwanie zależności w konstruktorze	75
6.8	Użycie komend	76
6.9	Użycie RelayCommand	76
6.10	Skaner kodów QR	77
6.11	Wyświetlanie kodów QR	78
6.12	Kodowanie i odkodowanie	79
6.13	Kontrolka NavigationButton	80
6.14	Konwerter	81
6.15	Hashowanie po zmianie hasła	82
6.16	Sprawdzenie poprawności hasła	82

Spis tabel

4.1	Wymagania funkcjonalne obu aplikacji	14
4.2	Wymagania funkcjonalne aplikacji mobilnej	15
4.3	Wymagania funkcjonalne aplikacji mobilnej	16
4.4	Wymagania funkcjonalne aplikacji desktopowej	17
4.5	Wymagania funkcjonalne aplikacji desktopowej	18
4.6	Wymagania niefunkcjonalne aplikacji desktopowej	21
4.7	Wymagania niefunkcjonalne aplikacji mobilnej	23
5.1	Wykaz atrybutów encji typu Activity	28
5.2	Wykaz atrybutów encji typu Competition	29
5.3	Wykaz atrybutów encji typu Notification	29
5.4	Wykaz atrybutów encji typu PROFESSIONAL	30
5.5	Wykaz atrybutów encji typu Specialization	30
5.6	Wykaz atrybutów encji typu Diet	31
5.7	Wykaz atrybutów encji typu Portion	31
5.8	Wykaz atrybutów encji typu Forage	32
5.9	Wykaz atrybutów encji typu Horse	33
5.10	Wykaz atrybutów encji typu HorseGender	33
5.11	Wykaz atrybutów encji typu Status	34
5.12	Wykaz atrybutów encji typu MealName	34
5.13	Wykaz atrybutów encji typu NutrtrionPlan	35
5.14	Wykaz atrybutów encji typu PeopleDetails	36
5.15	Wykaz atrybutów encji typu Participation	36
5.16	Wykaz atrybutów encji typu Shared	37
5.17	Wykaz atrybutów encji typu UnitOfMeasure	37
5.18	Wykaz atrybutów encji typu UserAcount	38

5.19 Wykaz atrybutów encji typu UserType	39
5.20 Wykaz atrybutów encji typu Visit	39
5.21 Wykaz atrybutów encji typu Meal	40
5.22 Wykaz atrybutów encji typu Contest	40
5.23 Opis schematu relacji Activities	45
5.24 Opis atrybutów relacji Activities	46
5.25 Opis schematu relacji Competitions	46
5.26 Opis atrybutów relacji Competitions	47
5.27 Opis schematu relacji Notifications	47
5.28 Opis atrybutów relacji Notifiactions	48
5.29 Opis schematu relacji Professionals	48
5.30 Opis atrybutów relacji Professionals	48
5.31 Opis schematu relacji Specialisations	49
5.32 Opis atrybutów relacji Professionals	49
5.33 Opis schematu relacji Diets	49
5.34 Opis atrybutów relacji Diets	50
5.35 Opis schematu relacji Portions	50
5.36 Opis atrybutów relacji Portions	50
5.37 Opis schematu relacji Forges	51
5.38 Opis atrybutów relacji Forges	51
5.39 Opis schematu relacji Horses	52
5.40 Opis atrybutów relacji Horses	53
5.41 Opis schematu relacji HorseGenders	53
5.42 Opis atrybutów relacji HorseGenders	53
5.43 Opis schematu relacji Specialisations	54
5.44 Opis atrybutów relacji Status	54
5.45 Opis schematu relacji MealNames	54
5.46 Opis atrybutów relacji MealNames	55
5.47 Opis schematu relacji NutritionPlans	55
5.48 Opis atrybutów relacji NutritionPlans	55
5.49 Opis schematu relacji PeopleDetails	56
5.50 Opis atrybutów relacji PeopleDetails	56
5.51 Opis schematu relacji Participations	57

5.52	Opis atrybutów relacji Participations	57
5.53	Opis schematu relacji Shareds	58
5.54	Opis atrybutów relacji Shareds	58
5.55	Opis schematu relacji UnitOfmeasures	59
5.56	Opis atrybutów relacji UnitOfmeasures	59
5.57	Opis schematu relacji UserTypes	59
5.58	Opis atrybutów relacji UserTypes	60
5.59	Opis schematu relacji UserAccounts	60
5.60	Opis atrybutów relacji UserAccounts	61
5.61	Opis schematu relacji Visits	61
5.62	Opis atrybutów relacji Visits	62
5.63	Opis schematu relacji Meals	62
5.64	Opis atrybutów relacji Meals	62

Opis zawartości APD