Bazy danych 2
Sprawozdanie końcowe z projektu

Grupa 26 w składzie:
Jakub Bąba 325146
Adrian Murawski 325200
Angelika Ostrowska 325204
Kacper Straszak 325234
Aleksandra Szymańska 318733

1. Model pojęciowy

Aby zaprojektować bazę danych dla klubu Hobby Horse, musieliśmy dobrze zrozumieć, jakie informacje chcemy przechowywać i jak one ze sobą współpracują. Zaczęliśmy od stworzenia modelu logicznego. Dzięki temu mogliśmy dobrze przemyśleć i zrozumieć co będzie nam potrzebne w późniejszych etapach projektu i jak poszczególne encje powinny być ze sobą powiązane, żeby taka baza miała sens. Model ten jest bardzo prosty, dzięki czemu mogliśmy łatwo zmieniać koncepcje i konsultować ze sobą/zmieniać nowe rozwiązania do wprowadzenia w celu budowy późniejszego lepszego modelu. Nasz gotowy model logiczny idealnie przyczynił się jako fundament pod późniejsze przeniesienie danych do modelu relacyjnego.

Stworzone końcowe encje i ich powiązania potem przeniesione do modelu relacyjnego:

Konie - Przechowuje informacje o koniach, takie jak ich rasa, wiek, czy kolor. Pomaga w zarządzaniu katalogiem koni dostępnych w ośrodku.

Członkowie - Przechowuje podstawowe informacje o każdym człowieku należącym do klubu. Jest encją nadrzędną dla jeźdźców i pracowników.

Jeźdźcy - Przechowuje dane o jeźdźcach, takie jak informacje o przypisanych koniach oraz posiadanych licencjach.

Pracownicy - Przechowuje dane o pracownikach ośrodka, takie jak stanowisko, i data zatrudnienia. Pomaga w zarządzaniu personelem.

Turnieje - Zawiera informacje o turniejach, w tym nazwę, datę i miejsce. Umożliwia zarządzanie harmonogramem wydarzeń.

Uczestnicy turniejów - Przechowuje informacje o uczestnikach poszczególnych turniejów i miejscach na których ci uczestnicy ukończyli turnieje.

Grupy - Reprezentują grupy jeźdźców, z określoną maksymalną liczbą członków. Ułatwia

zarządzanie zajęciami grupowymi.

Zajęcia - Przechowuje informacje o zajęciach, takich jak typ, data i przypisany trener. Umożliwia planowanie zajęć.

Stajnie - Zawiera informacje o stajniach, takie jak nazwa i adres. Ułatwia zarządzanie lokalizacjami stajni.

Adresy - Przechowuje adresy używane przez różne encje (np. stajnie, turnieje).

Akcesoria koni - Przechowuje informacje o przypisaniu poszczególnych akcesoriów do koni.

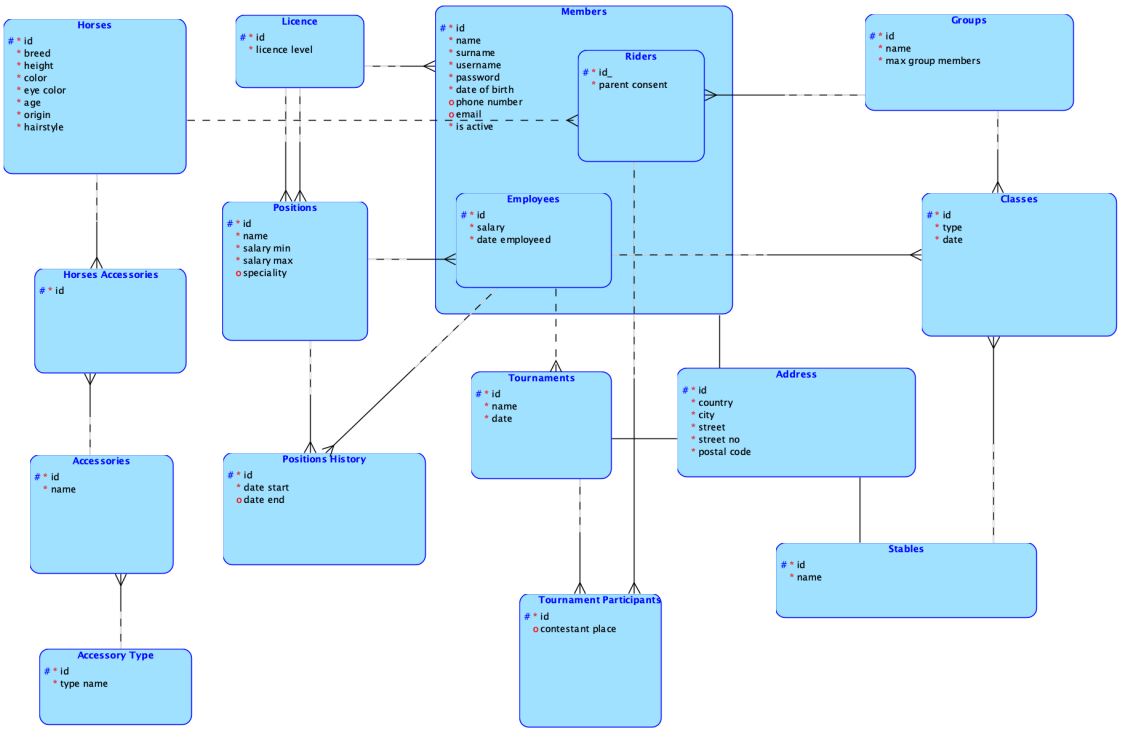
Akcesoria - Przechowuje dane o akcesoriach, takie jak nazwa i typ. Pomaga w zarządzaniu sprzętem jeździeckim.

Typ Akcesoriów - Zawiera informacje o typach akcesoriów, co ułatwia klasyfikację sprzętu.

Licencje - Przechowuje informacje o poziomach licencji dla jeźdźców.

Stanowiska - Przechowuje dane o stanowiskach pracy w ośrodku, takich jak nazwa i zakres pensji.

Historia Stanowisk - Przechowuje historię stanowisk pracowników, w tym daty rozpoczęcia i zakończenia pracy na danym stanowisku.



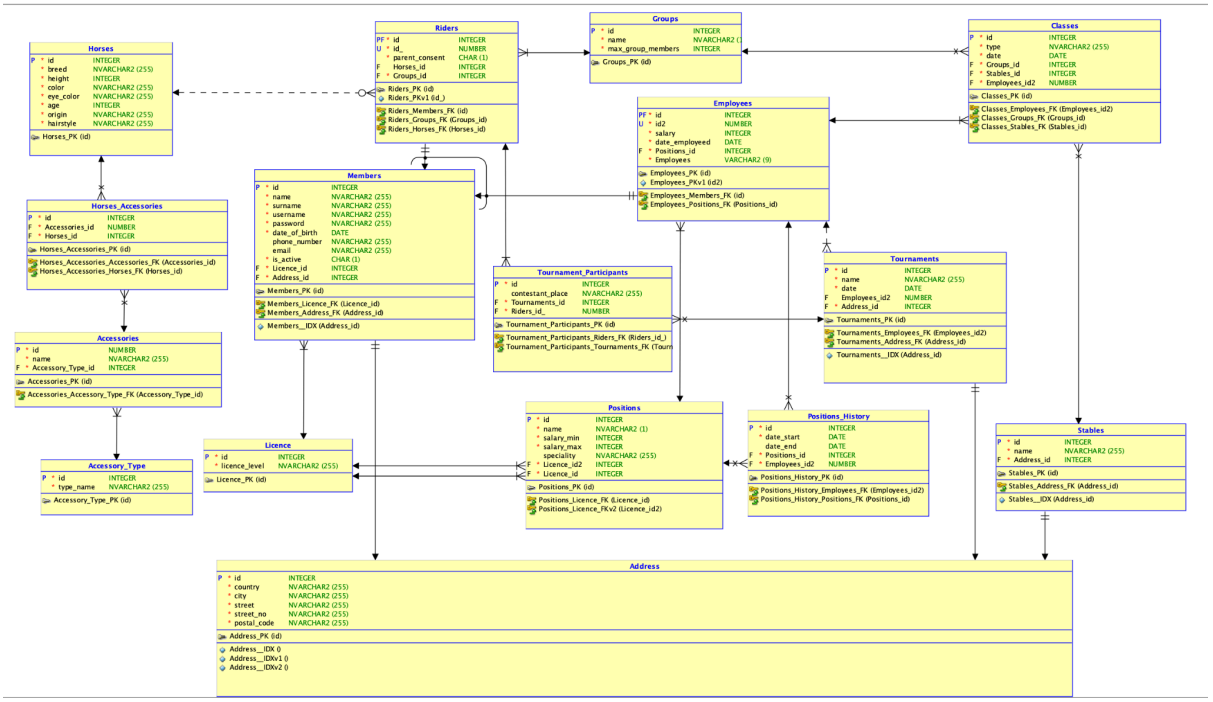
2. Model relacyjny

W celu bardziej precyzyjnego zarządzania klubem, opracowaliśmy przejście z modelu pojęciowego do relacyjnego. Model relacyjny jest szczegółową reprezentacją bazy danych. Przekształciliśmy wszystkie wyżej wymienione encje na konkretne tabele i kolumny. Dzięki temu po przekształceniu abstrakcyjnego modelu dostaliśmy rzeczywistą strukturę w systemie zarządzania bazą danych ze zdefiniowanymi atrybutami i w szczególności relacjami bazującymi na precyzyjnych połączeniach kluczy głównych i obcych. Dzięki takiemu modelowi baza jest bardziej spójna i dokładna, co skutkuje w mniejszej ilości anomalii i błędów, ale też lepszą organizacją i operacyjnością klubu.

Podczas tworzenia naszych modeli mieliśmy również do czynienia z normalizacją. Proces ten to podział tabeli na mniejsze w celu eliminacji redundancji i poprawy integralności danych. Dzięki temu mając na początku jedną tabelę określającą akcesoria rozszerzyliśmy ją na 3 różne, dzięki czemu możemy teraz bardziej swobodnie przeszukiwać bazę danych dla konkretnych parametrów, np znalezienie koni, które posiadają dany *typ akcesoriów* lub szybkie znalezienie akcesoriów dla danego konia. W naszym modelu normalizacja eliminuje także zbędne powielanie danych. Na przykład, zamiast trzymać adresy stajni i turniejów w każdej encji, przechowujemy je w oddzielnej tabeli Addresses, co zapobiega powielaniu tych samych informacji w różnych miejscach.

Podczas projektowania, rozważaliśmy bardziej rozbudowaną tabelę Tournaments zawierającą szczegółowe informacje o każdym turnieju, w tym historię wyników, sędziów, lokalizacje zmienne i inne. Taki projekt miał na celu umożliwienie szerokiej analizy i raportowania w kontekście całego środowiska jeździeckiego.
Po analizie, doszliśmy do wniosku, że taki szczegółowy poziom informacji jest zbędny z punktu widzenia zarządzania turniejami na poziomie klubowym. Klub Hobby Horse potrzebuje przede wszystkim podstawowych danych o turniejach i uczestnikach.
W związku z tym zdecydowaliśmy się na denormalizację, przez co struktura danych znacznie się uprościła. Końcowo utworzyliśmy dwie tabele: Tournaments, które zawierają podstawowe informacje o turniejach takie jak nazwa, data i adres, oraz Tournament_Participants, która przechowuje dane o uczestnikach turniejów, powiązując je z tabelą Jeźdźców. Dzięki temu podejściu klub uzyskał bazę dopasowaną do jego specyficznych potrzeb, umożliwiającą sprawne zarządzanie turniejami oraz uczestnikami bez nadmiaru niepotrzebnych danych.

Model pojęciowy jak i model relacyjny bazy danych zostały opracowane za pomocą aplikacji Oracle Data Modeler. To narzędzie umożliwiło wizualizację struktury bazy danych, definiowanie encji i relacji oraz transformację modelu pojęciowego na szczegółowy model relacyjny. Oracle Data Modeler zapewnia funkcje do zarządzania strukturą bazy danych, automatyzacji generowania skryptów DDL i optymalizacji projektu, co ułatwiło stworzenie wydajnej i spójnej bazy danych dla klubu.



3. Realizacja bazy danych (warstwa logiczna systemu)

Powyższą bazę danych zrealizowano w następujący sposób.
Użyto systemu zarządzania **MySQL**, który został uruchomiony w **Docker'owym** kontenerze. Rozwiązanie to zapewnia rozdzielenie bazy od reszty systemu, tak aby nie można było w bezpośredni sposób na nią wpływać (a to byłoby problemem na przykład gdyby bazę danych była przechowywana w pliku, który mógłby zostać przez przypadek usunięty lub w inny sposób naruszony). Jednocześnie zachowana jest możliwość dość łatwej komunikacji z takim kontenerem przez określony port.

Do komunikacji z bazą użyto serwera stworzonego przy pomocy framework'a **Django**.
Django oferuje *Object-Relational Mapping* (ORM), co ułatwia tworzenie, modyfikowanie i usuwanie rekordów w bazie, dzięki czemu możliwe było w wygodny i efektywny sposób przygotować operacje zmiany danych w bazie. Aby było to możliwe, stworzono modele, które są odwzorowaniem tabel tworzonej bazy. Wywołując odpowiednie metody na tych modelach, wykonywane są odpowiednie operacje w bazie danych.

Framework ten oferuje stworzenie **serwera HTTP**, który udostępnia pod ustalonymi adresami konkretne operacje. Konfiguracja serwera znajduje się w pliku */database/settings.py.* Zarówno adresy, jak i operacje, można dowolnie dostosować. Pozwoliło to na stworzenie wielu URLi, które są następnie wywoływane przez właściwą aplikację, aby uzyskać lub w odpowiedni sposób zmodyfikować konkretną tabelę. Serwer ten realizuje część logiczną naszego systemu. Stworzono ponad sto adresów (dostępne są one w pliku */database/urls.py)*, które pokrywają każdy potrzebny obszar modyfikacji danych. Zapewniają one możliwość analizowania, czy raportowania danych zawartych w bazie (wybierania odpowiednich rekordów z bazy), a także operacyjne możliwości związane z bazą danych (tworzenie nowych danych, zmiana istniejących, usuwanie starych, niepotrzebnych).

Wywołanie URLa powoduje dopasowanie odpowiedniej, przypisanej do niego metody. Te metody realizują główne operacje na bazie danych, a także wstępnie sprawdzają przesłane dane. Każda z metod ma *doc-stringa*, który informuje o wymaganym przez metodę formacie danych i pokazuje przykładową odpowiedź serwera. W przypadku poprawnych danych, metody przesyłają odpowiedź o sukcesie operacji. W przypadku niepowodzenia metody przechwytują błędy, które się pojawiły (lub które baza danych sygnalizuje za pomocą *triggerów* - dokładne informacje o nich znajdują się poniżej w tej dokumentacji). Informacje o tych błędach są przesyłane do użytkownika.

Do warstwy logicznej należą także testy (pliki w katalogu */tests)*, które stworzono w celu zbadania, czy stworzone URLe zachowują się zgodnie z oczekiwaniami. Framework wspiera testowanie, dzięki czemu nie jest modyfikowana faktyczna baza, tylko tworzona jest identyczna baza testowa, na której dowolnie możemy wykonywać operacje, uzyskując możliwość odpowiedniego testowania. Na koniec ta tymczasowa kopia oryginalnej bazy jest usuwana.

Sporą część logiki umieszczono w samej bazie danych, zgodnie z założeniami. W formie składowanych tam procedur umieszczono ***triggery***, które odpowiadają za poprawność danych wprowadzanych do systemu. Jest to dobre rozwiązanie, gdyż dane przed wprowadzeniem do bazy muszą być weryfikowane, a *triggery* to najlepsza możliwość walidacji danych przy wprowadzaniu i modyfikowaniu danych do bazy. Jest to słuszna opcja, ze względu na łatwość ich tworzenia i możliwość przekazywania komunikatów o błędach. W przypadku wystąpienia takowych, serwer Django przechwyci ten błąd i poinformuje w odpowiedzi aplikację o zajściu nietypowego zdarzenia, która następnie obsłuży wyświetlenie tego błędu użytkownikowi (tak, aby wiedział, jak powinien odpowiednio wprowadzić informacje). Wszystkie *triggery* umieszczono w pliku *sql_scripts/triggers.sql*.

Wraz z tworzeniem bazy, są w niej umieszczane niniejsze *triggery*, a także **przykładowe dane**, które służą prezentacji możliwości wykorzystania systemu. Dane przygotowano w sposób, który umożliwia pełne sprawdzenie możliwości systemu, w szczególności możliwości uruchomienia aplikacji klienckiej, za pośrednictwem której użytkownik w wygodny, przyjemny dla człowieka sposób może prowadzić interakcje z bazą. Przykładowe dane wprowadzane na początku do bazy są dostępne w pliku *sql_scripts/insert_data.sql*.

Cała warstwa logiczna odpowiada za odpowiednie przetworzenie i wykonanie żądania wysłanego przez użytkownika, tak aby nie musiał on zajmować się skomplikowanymi wywołaniami, a jedynie obsługiwał je za pomocą prostego, wygodnego interfejsu.

4. Aplikacja kliencka (warstwa interaktywna systemu)

Aplikacja ilustrująca działanie bazy danych napisana jest w Javie z użyciem biblioteki graficznej Swing. Umożliwia ona operacje wyświetlania, dodawania, modyfikacji i usuwania danych z tabel bazy. Aplikacja Javowa realizująca graficzny interfejs użytkownika łączy się z bazą danych poprzez serwer Django i dzięki temu umożliwia obsługiwanie bazy.

Jest to aplikacja desktopowa, umożliwiająca interakcję poprzez naciskanie przycisków i wpisywanie danych w pola tekstowe. Składa się z wielu okienek, widoków różnego typu, pomiędzy którymi można się nawigować. Główne typy widoków to:

- **Formularz** składający się z określonej liczby i typów pól. Mogą być to pola tekstowe, rozwijane listy, checkboxy do zaznaczania. Stworzony został uniwersalny szablon widoku formularza, który w prosty sposób umożliwia stworzenie nowych, różnorodnych, dostosowanych do potrzeb formularzy. Przykładowe formularze to “RiderFormGUI” - tworzenie konta Jeźdźca, “AddHorseGUI” - dodawanie nowego Konia, “AddTrainingGUI” - dodawanie nowego Treningu. Formularze używane są do Dodawania wierszy do bazy danych oraz do Modyfikacji już istniejących wierszy.
- **Przewijany widok** listy obiektów, umożliwiający przeglądanie istniejących danych z tabeli. Możliwe jest wyświetlenie wszystkich pasujących danych, informacji o nich - w zależności od tabeli, wypisywane są różne ważne informacje dotyczącego danego obiektu. Z poziomu tego widoku można też przenieść się do okna Dodawania, Usuwania lub Modyfikacji danych z aktualnej tabeli. Przykładowe pliki wykorzystujące ten szablon to “GroupScrollGUI” - przeglądanie Grup, “RidersScrollGUI” - przeglądanie Jeźdźców i “AccessoriesScrollGUI” - przeglądanie Akcesoriów.
- **Ekran logowania** oraz zakładania nowego konta, pozwalający wybrać czy konto, którego chcemy używać jest Jeźdźcem czy Pracownikiem. Z ekranu logowania odbywa się identyfikacja i uwierzytelnianie użytkownika, poprzez podanie loginu i hasła do naszej aplikacji. Są to widoki “ChooseAccountTypeGUI” i “LogInGUI”.
- Widok **Strony Głównej**, pełniącej funkcję Menu, nawigującego do dalszych ekranów. W zależności od typu użytkownika (Jeździec/Pracownik) i ewentualnie Pozycji Pracownika wyświetlany jest odpowiedni podzbiór możliwych przycisków. Prawa dostępu do widoków określone są w osobnym pliku w programie. Jeźdźcy mają przykładowo dostęp do swoich Treningów (tylko odczyt), do Grup i Turniejów, do których mogą się zapisać lub wypisać i do Stajni (również tylko odczyt). Pracownicy na różnych Pozycjach mogą A. Nie mieć dostępu, B. Mieć dostęp do odczytu i C. Mieć dostęp do zapisu (dodawanie, modyfikacja, usuwanie) danych z tabel. Np. Owner ma najwyższe poziomy dostępów do każdej z tabel, co umożliwia mu np. dodawać i usuwać Pracowników. Za to Trener nie ma dostępu do modyfikacji tabeli Pracowników, ponieważ wykraczałoby to poza jego kompetencje. Jest to ekran “HomePageGUI”.

Funkcje interfejsu komunikują się z funkcjami logicznymi, które z kolei umożliwiają komunikację z bazą danych.

Postarano się o czytelność i intuicyjność aplikacji, a także o efektywne zaimplementowanie jej - użycie dziedziczenia, klas abstrakcyjnych - aby podobne widoki jak najbardziej uogólnić, znaleźć ich wspólny mianownik.

Przy projektowaniu interfejsu wykorzystano ikony i obrazy przechowywane w folderze resources. Stworzono paletę kolorów, aby widoki były estetyczne i spójne.

5. Uruchomienie

Wraz z projektem dostarczony został skrypt *setup.sh* (w wersji dla systemów Linux i MacOS). Skrypt ten uruchamia kontener z bazą danych, wprowadza całą strukturę bazy i tworzy w niej rekordy (stworzone przez nas na potrzeby prezentacji możliwości działania systemu). Został dostarczony także skrypt, który uruchamia serwer Django. Serwer jest uruchamiany na adresie *loopback*. Jest to konfiguracja służąca pokazaniu działania, stąd adres *localhost* jest wystarczający.
Docelowo, w przypadku potrzeby wdrożenia takiego systemu do faktycznego działania, serwer taki bez problemu można uruchomić tak, aby przyjmował zapytania przez sieć internetową, dzięki czemu umożliwi działanie wielu użytkownikom.
Następnie użytkownik może uruchomić faktyczną aplikację, która dalej kieruje go odpowiednimi przyciskami umożliwiając korzystanie z niej. Robione jest to przez uruchomienie pliku **DefaultApp.java**.

6. Użytkowanie

Przykładowym procesem użytkowym aplikacji ze strony Pracownika byłoby:

1. Otworzenie aplikacji - uruchomienie pliku **DefaultApp.java**, który przenosi nas do ekranu Logowania.
2. Zalogowanie się jako już istniejący użytkownik - Pracownik, podając nazwę użytkownika i hasło.
3. Aplikacja zaloguje nas na odpowiednie konto należące do Pracownika. Pracownik ma przypisane Stanowisko i w zależności od niego ma dostęp do różnych akcji. Załóżmy, że nasz Pracownik pracuje aktualnie na Stanowisku Trainer.
4. Otwarte zostanie okno Strony Głównej, z przyciskami Stajnie, Konie, Akcesoria, Jeźdźcy, Treningi, Turnieje.
5. Kliknięcie przycisku Stajnie. Aplikacja przeniesie nas do odpowiedniego widoku.
6. Obejrzenie dostępnych Stajni. Nie ma możliwości dodawania, usuwania ani modyfikacji danych.
7. Powrót do Strony Głównej poprzez kliknięcie strzałki w lewym dolnym rogu ekranu.
8. Kliknięcie przycisku Treningi. Aplikacja przeniesie nas do odpowiedniego widoku.
9. Obejrzenie dostępnych Treningów.
10. Chęć dodania nowego Treningu. Żeby to osiągnąć, kliknięcie przycisku Dodaj w prawym górnym rogu ekranu. Aplikacja przeniesie nas do odpowiedniego formularza.
11. Wypełnienie pól formularza dodania Treningu zgodnie z wizją Pracownika.
12. Kliknięcie przycisku Prześlij, aby przesłać formularz. Zostanie sprawdzona poprawność wprowadzonych danych. System pozwoli dodać nowy Trening lub wyświetli komunikat o błędzie.
13. Ewentualna zmiana danych Treningu i dodanie go do bazy z sukcesem.
14. Powrót do widoku wszystkich Treningów, zauważenie że utworzony przez nas przed chwilą obiekt jest widoczny wśród innych.
15. Powrót do Strony Głównej poprzez kliknięcie strzałki w lewym dolnym rogu ekranu.
16. Wylogowanie się używając przycisku w prawym górnym rogu ekranu.
17. Zamknięcie aplikacji

Przykładowe informacje do logowania na konto Pracownika:
login: dwlodar, hasło: wlopass

Przykładowe informacje do logowania na konto Pracownika, który jest Właścicielem (więcej opcji modyfikacji i dostep do większej ilości danych ze względu na wyższe stanowisko)**:**
login: pjanko, hasło: piotrpass

Przykładowym procesem użytkowym aplikacji ze strony Jeźdźca byłoby:

1. Otworzenie aplikacji - uruchomienie pliku **DefaultApp.java**, który przenosi nas do ekranu Logowania.
2. Załóżmy, że nie mamy jeszcze konta. Klikamy więc w przycisk Załóż nowe konto w prawym dolnym rogu ekranu. Aplikacja przeniesie nas do odpowiedniego widoku.
3. Wybór przycisku Jeździec, ponieważ chcemy utworzyć nowe konto Jeźdźca. Aplikacja przeniesie nas do odpowiedniego formularza.
4. Wypełnienie pól formularza dodania nowego Jeźdźca zgodnie z wizją nowego ucznia.
5. Kliknięcie przycisku Prześlij, aby przesłać formularz. Zostanie sprawdzona poprawność wprowadzonych danych. System pozwoli założyć konto w takiej formie lub wyświetli komunikat o błędzie.
6. Ewentualna zmiana danych Jeźdźca i dodanie go do bazy z sukcesem.
7. Powrót do widoku logowania.
8. Zalogowanie się jako nowo utworzony użytkownik, podając nazwę użytkownika i hasło.
9. Aplikacja zaloguje nas na odpowiednie konto.
10. Otwarte zostanie okno Strony Głównej, z przyciskami Treningi, Turnieje, Grupy, Stajnie, Konie, Dezaktywacja konta.
11. Kliknięcie przycisku Treningi. Aplikacja przeniesie nas do odpowiedniego widoku.
12. Nie powinny być widoczne żadne Treningi, chyba że Jeździec na etapie rejestracji zapisał się do Grupy. W takim wypadku zobaczymy listę Treningów przypisanych do danej Grupy. Nie ma możliwości dodawania, usuwania ani modyfikacji danych.
13. Powrót do Strony Głównej poprzez kliknięcie strzałki w lewym dolnym rogu ekranu.
14. Chęć uczęszczania na więcej treningów. W tym celu należy zapisać się do dodatkowej Grupy.
15. Kliknięcie więc przycisku Grupy. Aplikacja przeniesie nas do odpowiedniego widoku.
16. Obejrzenie dostępnych Grup.
17. Kliknięcie przycisku Dołącz przy wybranej Grupie. Jeśli limit miejsc nie będzie wyczerpany, dołączymy z sukcesem.
18. Powrót do Strony Głównej poprzez kliknięcie strzałki w lewym dolnym rogu ekranu.
19. Wylogowanie się używając przycisku w prawym górnym rogu ekranu.
20. Zamknięcie aplikacji

Przykładowe informacje do logowania na konto Ridera:
login: jlewan, hasło: qwerty

7. Wnioski

Udało nam się stworzyć działający system, który jest w stanie obsłużyć organizację pracy w sportowym klubie. Baza danych przeznaczona do tego systemu uwzględnia między innymi pracowników (ich pozycję, pensję), pozycje i historie pozycji pracowników, zajęcia (kiedy i

gdzie się odbywają, kto będzie uczestnikiem zajęć, a kto je prowadził), grupy zajęciowe (i należących do nich uczniów), a także turnieje (ich uczestników i wyniki), licencje (które służą do reprezentacji poziomu uczniów oraz pracowników, a także możliwości nauczania) oraz konie i akcesoria dla nich (z podziałem na typy akcesoriów). Zmiana danych w bazie odbywa się bezproblemowo, a także na drodze do wprowadzenia danych występuje ich walidacja, aby uniknąć sytuacji, gdzie nieprawidłowe dane trafią do bazy danych.

Z kolei warstwa aplikacyjna, interfejsu użytkownika, pozwala na przyjemną interakcję z serwerem, pozwalającą na wygodne raportowanie, analizowanie i modyfikowanie (działania operacyjne) na bazie danych.

Oba elementy tworzą łącznie cały system.