

UNIwersYTET RZESZOWSKI
WYDZIAŁ NAUK ŚCISŁYCH I TECHNICZNYCH
INSTYTUT INFORMATYKI



Jakub Wawro
134984

Informatyka

System zarządzania turniejami szachowymi

Praca projektowa

Praca wykonana pod kierunkiem
mgr inż. Ewa Żesławska

Rzeszów 2025

Spis treści

1. Streszczenie w języku polskim i angielskim.	7
1.1. Streszczenie w języku polskim.....	7
1.2. Streszczenie w języku angielskim	7
2. Opis założeń projektu	8
2.1. Wymagania funkcjonalne	8
2.2. Wymagania niefunkcjonalne	9
3. Opis struktury projektu	10
3.1. Baza danych.....	10
3.1.1. Tabela users.....	10
3.1.2. Tabela players	10
3.1.3. Tabela tournaments	11
3.1.4. Tabela tournaments_players.....	12
3.1.5. Tabela games.....	12
3.2. Aplikacja.....	13
3.2.1. Klasa Main	13
3.2.2. Klasa ENUM.....	13
3.2.3. Klasy modelowe.....	14
3.2.4. Klasa JDBC.....	14
3.2.5. Klasy DAO	14
3.2.6. Klasy autoryzacji	15
3.2.7. Klasy panelu administratora	15
3.2.8. Klasy panelu gracza	16
3.3. Wykorzystane technologie i narzędzia	16
3.4. Hierarchia klas.....	16
4. Harmonogram realizacji projektu	18
4.1. Trudności w projekcie	18
4.2. Repozytorium i kontrola wersji	19
5. Prezentacja warstwy użytkowej projektu	20
5.1. Ekran logowania	20
5.1.1. Logowanie komunikat - 1	21
5.1.2. Logowanie komunikat - 2	21
5.2. Ekran rejestracji.....	22
5.2.1. Rejestracja komunikat - 1	22
5.2.2. Rejestracja komunikat - 2	23
5.2.3. Rejestracja komunikat - 3	23
5.3. Ekran powitalny gracza	24

5.4.	Ekran profilu gracza	25
5.5.	Ekran turniejów	26
5.5.1.	Ekran turniejów komunikat - 1	26
5.6.	Ekran turniejów z szczegółami	27
5.6.1.	Ekran turniejów z szczegółami - wyniki poszczególnego gracza	28
5.7.	Ekran powitalny administratora	28
5.8.	Ekran zarządzania użytkownikami	29
5.8.1.	Ekran do zarządzania użytkownikami - Alerty	29
5.9.	Ekran zarządzania turniejami	30
5.9.1.	Ekran zarządzania turniejami - wyświetlenie wszystkich gier turnieju	31
6.	Podsumowanie	33
	Spis rysunków	34

1. Streszczenie w języku polskim i angielskim.

W tym rozdziale przedstawiono krótkie streszczenie projektu, w którym zaprezentowano cel aplikacji oraz technologie wykorzystane do jej stworzenia. Streszczenie zostało przygotowane w dwóch językach: polskim i angielskim.

1.1. Streszczenie w języku polskim

Dokumentacja jest poświęcona opisowi projektu „System zarządzania turniejami szachowymi”, który został stworzony, aby ułatwić i usprawnić turnieje odbywające się stacjonarnie. System ma na celu usprawnić przebieg turniejów stacjonarnych oraz zmniejszyć błąd obliczeniowy przy sumowaniu wyników turnieju. System ma również pełnić usługę dla graczy, którzy mogą sprawdzać swoje wyniki oraz statystyki.

System funkcjonuje na języku Java oraz przy pomocy biblioteki JavaFX, która jest odpowiedzialna za GUI. Style do GUI napisane są przy pomocy CSS-a, a baza danych jest oparta na MySQL, połączona przy pomocy technologii JDBC.

1.2. Streszczenie w języku angielskim

Documentation is a description of the project "Chess Tournament Management System", which was created to provide and improve stationary tournaments. The system aims to improve the course of stationary tournaments and to make a calculation error when summing up financial results. The system also has a full service for players who can send their data and statistics.

The system works on Java and with the help of the JavaFX library, which is responsible for the GUI. GUI style supporting CSS help, a database based on MySQL, connected with the help of JDBC technology.

2. Opis założeń projektu

Celem projektu jest usprawnienie zarządzania stacjonarnymi turniejami szachowymi poprzez stworzenie systemu, w którym administrator może zarządzać turniejami oraz użytkownikami, a użytkownik może śledzić wyniki i statystyki.

Problemem stacjonarnych turniejów szachowych jest ich czasochłonność oraz słaba organizacja. Źródłem problemu jest fakt, że wiele czynności, takich jak rejestracja do turnieju czy zliczanie wyników, odbywa się ręcznie i na miejscu.

Problem jest ważny, ponieważ w przypadku większych turniejów potęguje się, co może skutkować tym, że czas stracony na oczekiwanie wydłuży się do takiego stopnia, że turnieje szachowe tracą na swoim zainteresowaniu. Wszystkie czynności, które odbywają się ręcznie, takie jak zliczanie wyników, mogą również doprowadzić do błędów, przez które turnieje mogą być źle rozstrzygane.

Rozwiązaniem problemu jest system, który obsłuży wszystkie obliczenia oraz całą rejestrację, którą musiałby się zająć administrator. Również śledzenie wyników zostanie usprawnione, ponieważ będzie można sprawdzać je w systemie.

Problem zostanie rozwiązany poprzez stworzenie systemu zarządzania turniejami szachowymi przy użyciu JavaFX oraz bazy danych MySQL. Realizacja projektu rozpocznie się od planowania i zrozumienia wymagań. Następnie rozpoczną się prace nad przemyślaną bazą danych, która będzie zawierała wszystkie potrzebne tabele do funkcjonowania systemu. Kolejno, prace ruszą nad kodem w języku Java, zaczynając od poprawnego połączenia z bazą danych za pomocą JDBC. Później zajmiemy się segmentem odpowiedzialnym za logowanie i rejestrację użytkowników. Przechodząc dalej, stworzymy widok administratora, w którym będzie mógł zarządzać turniejami i użytkownikami oraz rozstrzygać mecze w turniejach. Następnym segmentem będzie widok gracza, który będzie mógł sprawdzić swój profil, a w nim statystyki i swoje mecze. Gracz będzie miał również możliwość dołączenia do turnieju oraz sprawdzenia swoich wyników w poszczególnych meczach turnieju. Kolejnym etapem będzie nadanie stylów naszemu interfejsowi użytkownika zaprojektowanemu w JavaFX za pomocą technologii CSS. Na sam koniec zostaną przeprowadzone testy sprawdzające działanie programu.

2.1. Wymagania funkcjonalne

Zarządzanie kontami użytkowników

- Użytkownik ma możliwość zarejestrowanie nowego konta
- Użytkownik ma możliwość zalogowania się do istniejącego konta
- Administrator ma możliwość zarejestrować nowego użytkownika
- Administrator ma możliwość edytować dane użytkownika
- Administrator ma możliwość usunąć zarejestrowanego użytkownika

Zarządzanie turniejami

- Gracz ma możliwość dołączenia do istniejącego turnieju pod warunkiem wolnych miejsc
- Administrator ma możliwość utworzenia turnieju

- Administrator ma możliwość edycji turnieju
- Administrator ma możliwość usunięcia turnieju
- Administrator ma możliwość zakończenia turnieju

Zarządzanie meczami

- Administrator ma możliwość rozstrzygania meczy

2.2. Wymagania niefunkcjonalne

Wydajność

- Wszystkie operacje przeprowadzane przez jednego użytkownika nie powinny przekraczać 1 sekundy. System nie powinien uruchamiać się dłużej niż 5 sekund.

Środowisko

- Środowisko uruchomieniowe powinno być kompatybilne z zainstalowanym JDK oraz bazą danych MySQL.

Użyteczność

- System powinien być użyteczny przy turniejach, w których bierze udział od 2 do 50 graczy. Przy mniejszych turniejach system nie ma sensu, a przy większych jest zbyt mało wydajny.

Niezawodność

- System powinien obsługiwać wszystkie możliwe wyjątki, które mogą występować podczas pracy z użytkownikiem.

Utrzymywalność

- Kod musi być zgodny ze standardami programowania obiektowego oraz musi być napisany w sposób czytelny i zrozumiały dla innych deweloperów.

Integralność danych

- System musi walidować dane wprowadzane przez użytkownika, aby sprawdzać ich poprawność. Dane usuwane przez administratora muszą być realizowane za pomocą usuwania kaskadowego, aby zachować spójność danych.

3. Opis struktury projektu

W tym rozdziale została opisana struktura projektu, która składa się z rozbudowanej bazy danych oraz aplikacji.

3.1. Baza danych

Baza danych została wykonana przy pomocy MySQL. Cała baza składa się z 5 tabel odpowiedzialnych za poprawne działanie systemu i przetwarzanie danych.

3.1.1. Tabela users

Tabela users odpowiada za przechowywanie danych o naszych użytkownikach, na których składają się administratorzy i gracze. Tabela ta zawiera 6 pól, które opisują użytkowników.

- idusers jest polem typu INT oraz naszym kluczem głównym w tabeli. Zawiera również takie atrybuty jak NOT NULL, który wymusza wprowadzenie wartości w tym polu, oraz AUTO_INCREMENT, które automatycznie przydziela kolejną wartość o 1 większą od poprzedniej.
- login jest polem typu VARCHAR(45). Jest to pole odpowiedzialne za login naszego użytkownika, który nie może przekraczać 45 znaków. Zawiera również atrybut NOT NULL, który wymusza wprowadzenie wartości w tym polu.
- password jest polem typu VARCHAR(45). Jest to pole odpowiedzialne za hasło naszego użytkownika, które nie może przekraczać 45 znaków. Zawiera również atrybut NOT NULL, który wymusza wprowadzenie wartości w tym polu.
- firstname jest polem typu VARCHAR(45). Jest to pole odpowiedzialne za imię naszego użytkownika, które nie może przekroczyć 45 znaków. Zawiera również atrybut NOT NULL, który wymusza wprowadzenie wartości w tym polu.
- lastname jest polem typu VARCHAR(45). Jest to pole odpowiedzialne za nazwisko naszego użytkownika, które nie może przekroczyć 45 znaków. Zawiera również atrybut NOT NULL, który wymusza wprowadzenie wartości w tym polu.
- role jest polem typu ENUM('GRACZ', 'ADMINISTRATOR'). Jest to pole odpowiedzialne za rolę naszego użytkownika, która może być równa 'GRACZ' lub 'ADMINISTRATOR'. Zawiera również atrybut NOT NULL, który wymusza wprowadzenie wartości w tym polu, oraz atrybut DEFAULT, który ustawia wartość domyślną równą 'GRACZ'.

3.1.2. Tabela players

Tabela players odpowiada za gracza, który posiada bardziej specyficzne pola niż zwykły użytkownik. Tabela zawiera 7 pól, które opisują gracza.

- id jest polem typu INT oraz kluczem głównym w tabeli. Zawiera również takie atrybuty jak NOT NULL, który wymusza wprowadzenie wartości w tym polu, oraz AUTO_INCREMENT, które automatycznie przydziela kolejną wartość o 1 większą od poprzedniej.

- `user_id` jest polem typu `INT` i jest kluczem obcym, który łączy tabelę `players` z tabelą `users` relacją. Relacja wygląda tak, że każdy `user` może mieć co najwyżej jednego `playera`, a każdy `player` musi mieć jednego `usera`. Zawiera również atrybuty takie jak `NOT NULL`, który wymusza wprowadzenie wartości w tym polu, oraz atrybut `UNIQUE`, który wymusza, aby każda wartość była unikatowa.
- `ranking` jest polem typu `INT`. Jest to pole odpowiedzialne za ranking gracza. Zawiera również atrybut `DEFAULT`, który ustawia domyślny ranking na 1000.
- `wins` jest polem typu `INT`. Jest to pole odpowiedzialne za liczbę zwycięstw gracza. Zawiera również atrybut `DEFAULT`, który ustawia domyślną liczbę wygranych na 0.
- `draws` jest polem typu `INT`. Jest to pole odpowiedzialne za liczbę remisów gracza. Zawiera również atrybut `DEFAULT`, który ustawia domyślną liczbę remisów na 0.
- `losses` jest polem typu `INT`. Jest to pole odpowiedzialne za liczbę porażek gracza. Zawiera również atrybut `DEFAULT`, który ustawia domyślną liczbę porażek na 0.
- `games_played` jest polem typu `INT`. Jest to pole odpowiedzialne za liczbę rozegranych gier gracza. Zawiera również atrybut `DEFAULT`, który ustawia domyślną liczbę rozegranych gier na 0.

3.1.3. Tabela tournaments

Tabela `tournaments` odpowiada za nasze turnieje w systemie. Tabela zawiera 7 pól, które definiują strukturę turniejów.

- `id` jest polem typu `INT` oraz naszym kluczem głównym w tabeli. Zawiera również takie atrybuty jak `NOT NULL`, który wymusza wprowadzenie wartości w tym polu, oraz `AUTO_INCREMENT`, które automatycznie przydziela kolejną wartość o 1 większą od poprzedniej.
- `name` jest polem typu `VARCHAR(100)`. Jest to pole odpowiedzialne za nazwę naszego turnieju, która nie może przekraczać 100 znaków. Zawiera również atrybut `NOT NULL`, który wymusza wprowadzenie wartości w tym polu.
- `start_date` jest polem typu `DATE`. Jest to pole odpowiedzialne za datę startu naszego turnieju. Zawiera również atrybut `DEFAULT`, który ustawia domyślną datę startu turnieju na `NULL`.
- `end_date` jest polem typu `DATE`. Jest to pole odpowiedzialne za datę końca naszego turnieju. Zawiera również atrybut `DEFAULT`, który ustawia domyślną datę końca turnieju na `NULL`.
- `max_slots` jest polem typu `INT`. Jest to pole odpowiedzialne za maksymalną liczbę graczy w naszym turnieju. Zawiera również atrybut `DEFAULT`, który ustawia domyślną maksymalną liczbę graczy na 20, oraz atrybut `NOT NULL`, który wymusza wprowadzenie wartości w tym polu.
- `free_slots` jest polem typu `INT`. Jest to pole odpowiedzialne za liczbę wolnych miejsc w naszym turnieju. Zawiera również atrybut `DEFAULT`, który ustawia domyślną liczbę wolnych miejsc na 20, oraz atrybut `NOT NULL`, który wymusza wprowadzenie wartości w tym polu.
- `status` jest polem typu `VARCHAR(50)`. Jest to pole odpowiedzialne za status turnieju, który nie może przekraczać 50 znaków. Zawiera również atrybut `NOT NULL`, który wymusza wprowadzenie wartości w tym polu, oraz atrybut `DEFAULT`, który ustawia wartość domyślną na `'OTWARTY'`.

3.1.4. Tabela tournaments_players

Tabela tournaments_players jest tabelą łączącą tabele players z tabelą tournaments w relacji wiele do wielu.

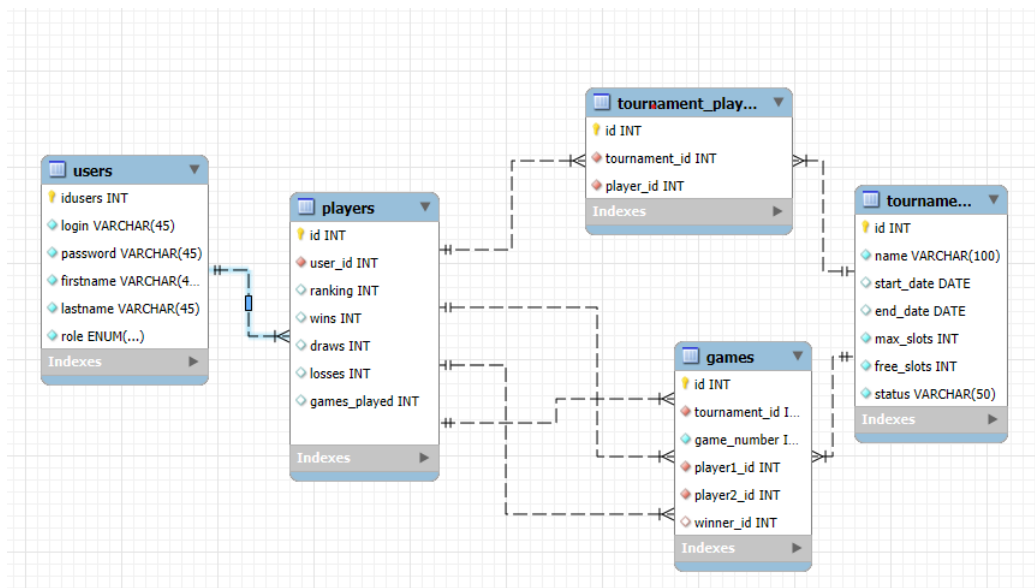
Tabela zawiera 3 pola, które służą do stworzenia relacji wiele do wielu.

- id jest polem typu INT oraz kluczem głównym w tabeli. Zawiera również atrybuty takie jak NOT NULL, który wymusza wprowadzenie wartości w tym polu, oraz AUTO_INCREMENT, które automatycznie przydziela kolejną wartość o 1 większą od poprzedniej.
- tournament_id oraz player_id są polami typu INT i są kluczami obcymi, które łączą tabelę players z tabelą tournaments relacją. Relacja wygląda tak, że wiele players może mieć wiele tournaments. Zawierają również atrybut NOT NULL, który wymusza wprowadzenie wartości w tych polach.

3.1.5. Tabela games

Tabela games przedstawia pojedyncze mecze, z których składają się turnieje. Każde pole w tej tabeli to dane o pojedynczym meczu. Tabela zawiera 6 pól.

- id jest polem typu INT oraz kluczem głównym w tabeli. Zawiera również takie atrybuty jak NOT NULL, który wymusza wprowadzenie wartości w tym polu, oraz AUTO_INCREMENT, które automatycznie przydziela kolejną wartość o 1 większą od poprzedniej.
- tournament_id jest polem typu INT i jest kluczem obcym, który łączy tabelę games z tabelą tournaments. Relacja polega na tym, że jeden turniej może zawierać wiele meczów. Zawiera również atrybut NOT NULL, który wymusza wprowadzenie danych.
- game_number jest polem typu INT, które jest numerem meczu w turnieju. Zawiera atrybut NOT NULL, który wymusza wprowadzenie danych.
- player1_id jest polem typu INT i jest kluczem obcym, który łączy tabelę games z tabelą players. Relacja polega na tym, że jeden player może uczestniczyć w wielu meczach. Zawiera również atrybut NOT NULL, który wymusza wprowadzenie danych.
- player2_id jest polem typu INT i jest kluczem obcym, który łączy tabelę games z tabelą players. Relacja polega na tym, że jeden player może uczestniczyć w wielu meczach. Zawiera również atrybut NOT NULL, który wymusza wprowadzenie danych.
- winner_id jest polem typu INT i jest kluczem obcym, który łączy tabelę games z tabelą players. Relacja polega na tym, że jeden player może być zwycięzcą w wielu meczach. Pole to wyłania zwycięzcę spośród player1_id i player2_id, a NULL oznacza remis. Zawiera również atrybut DEFAULT, który ustawia winner_id na NULL.



Rys. 3.1. Diagram EER bazy danych systemu zarządzania turniejami szachowymi.

3.2. Aplikacja

Aplikacja została wykonana przy pomocy języka Java. GUI zostało wykonane z wykorzystaniem JavaFX, a style przy pomocy CSS-a. Pliki zostały poroździelane do pakietów. Aplikacja składa się z 29 klas, 16 plików FXML oraz 1 pliku CSS. W kolejnych podrozdziałach zostały również opisane wszystkie klasy oraz niektóre metody tych klas.

3.2.1. Klasa Main

Klasa Main jest miejscem, w którym program rozpoczyna swoje działanie. Klasa ta dziedziczy po Application i implementuje metodę start(), która jest niezbędna do uruchomienia aplikacji w JavaFX.

Metoda main jest minimalistyczna – wywołuje metodę launch(args), która z kolei wywołuje metodę start(). To właśnie dzięki niej aplikacja rozpoczyna swoje działanie.

Metoda start(), dziedziczona po Application, tworzy aplikację graficzną za pomocą JavaFX. Ustawia statyczną wielkość okna oraz ładuje ikonę aplikacji. Wczytuje scenę LoginController.fxml z początkowymi wymiarami 600x400px, przypisanymi do obiektu scene. Metoda ładuje również style CSS z pliku style.css. Nadaje aplikacji tytuł "System do zarządzania Turniejami Szachowymi", który będzie widoczny w lewym górnym rogu paska aplikacji. Na koniec metoda ustawia scenę i ją wyświetla.

3.2.2. Klasa ENUM

Klasa Role jest klasą ENUM i służy do reprezentowania roli użytkownika. ENUM składa się z 2 wybrów 'ADMINISTRATOR' oraz 'GRACZ'.

3.2.3. Klasy modelowe

Klasy User, Player, Admin, Tournament, Game są klasami modelowymi. Klasy modelowe mają przechowywać podstawowe dane, które są przypisane do poszczególnych klas. Klasa User przechowuje ogólne informacje o użytkowniku. Klasa Player dziedziczy po klasie User, pobiera jej pola oraz dodaje dodatkowe, które są szczególnymi danymi dla gracza. Klasa Admin dziedziczy po klasie User i pobiera jej pola. Klasa Tournament przechowuje informacje o turniejach. Klasa Game przechowuje informacje o poszczególnych meczach.

3.2.4. Klasa JDBC

Klasa JDBC jest klasą odpowiedzialną za połączenie aplikacji z bazą danych. Adres do bazy danych jest przechowywany pod stałą URL. Login i hasło do bazy danych znajdują się odpowiednio w stałych USER oraz PASSWORD.

3.2.5. Klasy DAO

Klasy DAO są odpowiedzialne za pobieranie danych z bazy danych, ich zapisywanie, aktualizowanie i usuwanie. W projekcie stworzono 3 klasy DAO, które są niezbędne do prawidłowego działania reszty systemu.

Klasa UserDao służy do operacji z tabelą users w bazie danych. Klasa posiada zdefiniowane stałe i zawiera 6 metod odpowiedzialnych za pracę z bazą danych. Metoda registerUser odpowiada za rejestrację użytkownika, który domyślnie zostanie dodany jako gracz do bazy danych. Metoda isUserExists sprawdza czy dany użytkownik już istnieje w bazie danych. Metoda loginUser służy do logowania istniejącego użytkownika do systemu. Metoda getAllUsers służy do pobrania wszystkich użytkowników z bazy danych. Metoda updateUser służy do aktualizowania danych usera. Metoda deleteUser służy do usuwania istniejącego użytkownika z bazy danych.

Klasa PlayerDAO służy do operacji z tabelą players w bazie danych. Klasa zawiera 6 metod odpowiedzialnych za pracę z bazą danych. Metoda addPlayerDetails służy za dodawanie danych gracza do bazy danych. Metody getPlayerTableIdByUserId, getPlayerRanking służą do pobierania danych gracza. Metoda deletePlayerDetails służy do usuwania gracza z bazy danych. Metoda getAllGamesForPlayer służy do pobrania wszystkich meczy konkretnego gracza z bazy danych.

Klasa TournamentDAO służy do operacji z tabelą tournaments w bazie danych. Klasa zawiera 6 metod odpowiedzialnych za pracę z bazą danych. Metoda getAllTournaments służy do pobrania wszystkich turniejów z bazy danych. Metoda addTournament służy za dodawanie turnieju do bazy danych. Metoda updateTournament służy do aktualizacji turnieju. Metoda deleteTournament służy za usuwanie turnieju. Metoda getTournamentById służy do pobierania turnieju z bazy danych. Metoda addPlayerToTournament służy za dodawanie gracza do turnieju. Metoda updateFreeSlotsAndCheckStatus służy do aktualizacji liczby wolnych miejsc w turnieju. Metoda updateTournamentStatus służy do aktualizacji statusu turnieju. Metoda generateGamesForTournament służy do wygenerowania/sparowania graczy w danym turnieju. Metoda isPlayerRegisteredForTournament służy do sprawdzania czy dany gracz jest już zarejestrowany w turnieju. Metoda getRegisteredPlayersWithRecordsForTournament służy do pobrania wszystkich zarejestrowanych graczy w danym turnieju. Metoda getAllGamesForTournament służy do pobrania wszystkich meczy danego turnieju. Metoda updateGameResult służy do rozstrzygania wyniku sparowanych par w turnieju. Metoda calculateAndApplyTournamentStats służy do obliczania statystyk gracza (W/D/L) w danym turnieju.

3.2.6. Klasy autoryzacji

Klasy autoryzacji służą do zarządzania dostępem użytkownika do systemu aplikacji. W projekcie istnieją 3 klasy służące do zarządzania autoryzacją użytkownika.

Klasa `LoginController` służy do logowania użytkownika do systemu. Metoda `handleLoginButtonAction` służy do walidacji wprowadzonych danych użytkownika oraz logowaniu jeżeli przejdzie walidację. Metoda `handleRegisterButtonAction` służy do przełączenia sceny na scenę rejestracji.

Klasa `RegisterController` służy do rejestracji użytkownika w bazie danych. Metoda `handleRegister` służy do walidacji wprowadzanych danych użytkownika, a następnie do jego rejestracji, jeśli dane przejdą walidację.

Klasa `Logout` służy za wylogowywanie użytkownika i przenoszenie go do ekranu logowania. Metoda `switchScene` służy za zmianę sceny na ekran logowania.

Klasa `BaseDashboardController` jest klasą abstrakcyjną dla kontrolerów ekranów głównych administratora oraz gracza. Zawiera abstrakcyjną metodę, której celem jest przekazanie danych o aktualnym użytkowniku.

3.2.7. Klasy panelu administratora

Klasy panelu administratora odpowiadają za obsługę większości mechanizm związanych z tym panelem. W projekcie znajduje się 7 klas ściśle powiązanych z panelem administratora.

Klasa `AdminDashboard` dziedziczy po klasie `BaseDashboardController` i jest kontrolerem głównego ekranu powitalnego administratora. Metoda `setLoggedInUser` służy do ustawiania zalogowanego użytkownika i tworzy dla niego tekst powitalny.

Klasa `NavigationPanel` służy za kontrolę nad częścią GUI odpowiedzialną za przechodzenie między scenami. Posiada tylko jedną metodę, która przenosi użytkownika do odpowiedniej sceny w zależności od kliknięcia.

Klasa `AdminTopBar` dziedziczy po `Logout` i służy jako kontroler górnej części GUI. Poprzez dziedziczenie klasa ma możliwość wylogowania użytkownika przyciskiem.

Klasa `AdminUserController` służy do kontroli ekranu odpowiedzialnego za obsługę i manipulację danymi użytkownika. Metoda `setupActionsColumn` służy za kontrolę przycisków, które znajdują się w tabeli w kolumnie Operacje. Przyciski te odpowiadają za edycję i usuwanie wierszy. Metoda `handleEditRequest` służy za wypełnienie danymi użytkownika, którego edytujemy, pól przeznaczonych do edycji. Metoda `handleAddUser` służy za dodawanie nowego użytkownika do bazy danych. Metoda `handleUpdateUser` służy za aktualizację danych użytkownika w bazie danych na te wprowadzone w polach, jeśli przeszły walidację. Metoda `handleDeleteUser` służy za usuwanie użytkownika z bazy danych. Metoda `resetForm` służy za czyszczenie formularza. Metoda `loadAllUsers` służy za ładowanie listy wszystkich użytkowników, którzy zostaną wyświetleni. Metoda `loadUserByRole` służy za pobieranie wszystkich użytkowników, a następnie filtruje ich według roli. Przefiltrowana lista wyświetlana jest w tabeli. Metoda `refreshTable` służy za zmienianie wyświetlanych użytkowników zależnie od ustawionego filtra.

Klasa `AdminTournamentController` służy za kontrolowanie i zarządzanie ekranem Turniejowym. Metoda `handleShowTournamentMatches` służy za przełączanie się na widok meczów w turnieju. Metoda `setupActionsColumn` służy za tworzenie przycisków służących do manipulacji danymi. Metoda `handleEditRequest` służy za wypełnienie danymi turnieju którego chcemy edytować pola w których go edytujemy. Metoda `handleEditRequest` służy za wypełnienie danymi turnieju, którego edytujemy pola. Metoda `handleUpdateTournament` służy za aktualizację turnieju w bazie danych po wcześniejszej walidacji. Metoda `handleDeleteUser` służy za usuwanie turniejów z bazy danych. Metoda `handleAddTournament` służy za

dodawanie nowego turnieju do bazy danych. Metody `handleClearForm`, `resetForm` służy za czyszczenie formularza. Metoda `loadTournamentDataFromDatabase` służy za ładowanie turniejów do aplikacji. Metoda `handleEndTournament` służy do zakończenia turnieju.

Klasa `AdminResolveGame` odpowiedzialna za kontrolowanie obszaru związanego z rozstrzyganiem meczy. Metody `handlePlayer1Win`, `handlePlayer2Win`, `handleDraw` służą za rozstrzyganie wyniku spotkania.

3.2.8. Klasy panelu gracza

Klasy panelu gracza odpowiadają za obsługę większości mechanik związanych z tym panelem. W projekcie znajduje się 7 klas ściśle powiązanych z panelem gracza.

Klasa `PlayerDashboard` dziedziczy po `BaseDashboardController`. Klasa służy za kontrolę głównego ekranu użytkownika po zalogowaniu. Metoda `setLoggedInUser` służy za ustawianie zalogowanego użytkownika oraz tworzy dla niego tekst powitalny. Metoda `handleShowTournaments` służy za przełączanie na widok ekranu turniejów. Metoda `handleShowTournaments` służy za przełączanie na widok ekranu turniejów ze szczegółami. Metoda `handleShowProfile` służy za przełączanie na widok na ekranu Profilu użytkownika

Klasa `PlayerNavigation` służy za kontrolowanie obszaru nawigacji aplikacji. Metoda `handleProfileClicked` służy do kontroli przycisku przełączającego na profil. Metoda `handleTournamentsClicked` służy za kontrolę przycisku przełączającego na turnieje.

Klasa `PlayerTopBar` dziedziczy po `Logout`. Klasa kontroluje górną część interfejsu gracza. Metoda `handleLogout` służy za wylogowywanie z systemu.

Klasa `PlayerTournaments` służy za kontrolę nad widokiem związanym z turniejami. Metoda `loadTournaments` służy za pobranie turniejów z bazy danych. Metoda `handleRegisterForTournament` służy za rejestrowanie się do danego turnieju.

Klasa `PlayerTournamentInfo` służy za kontrolę sceny ze szczegółami turnieju. Metoda `loadTournamentParticipants` służy za pobieranie listy wszystkich użytkowników zarejestrowanych do turnieju. Metoda `handleRegisterForTournament` służy do otwierania nowego okna, które wyświetla widok meczów danego użytkownika.

Klasa `PlayerIndividualMatches` obsługuje widok meczów poszczególnego zawodnika. Metoda `loadPlayerMatches` służy do pobierania listy wszystkich meczów danego użytkownika w turnieju. Metoda `handleClose` służy za zamykanie okna.

3.3. Wykorzystane technologie i narzędzia

W projekcie wykorzystano technologie, które umożliwiły stworzenie systemu opartego na języku Java. Cały system został napisany w Javie przy pomocy środowiska IntelliJ IDEA 2024.3.3 Ultimate.

Interfejs graficzny użytkownika zrealizowano za pomocą technologii JavaFX. Sceny były głównie tworzone przy użyciu narzędzia Scene Builder. Style do scen tworzone za pomocą technologii CSS.

Baza danych projektu powstała w MySQL, z wykorzystaniem narzędzia MySQL Workbench 8.0 CE.

3.4. Hierarchia klas

Projekt został zaprogramowany z wykorzystaniem programowania obiektowego, co potwierdzają dziedziczenia, klasa abstrakcyjna oraz hermetyzacja.

Struktura projektu opiera się na klasie `User`, która jest podstawą dla każdego, kto loguje się do systemu. `User` rozdziela się na dwie pomniejsze klasy: `Admin` oraz `Player`. Klasa `Player` dziedziczy po `Userze`, dodając unikatowe dla gracza pola.

4. Harmonogram realizacji projektu

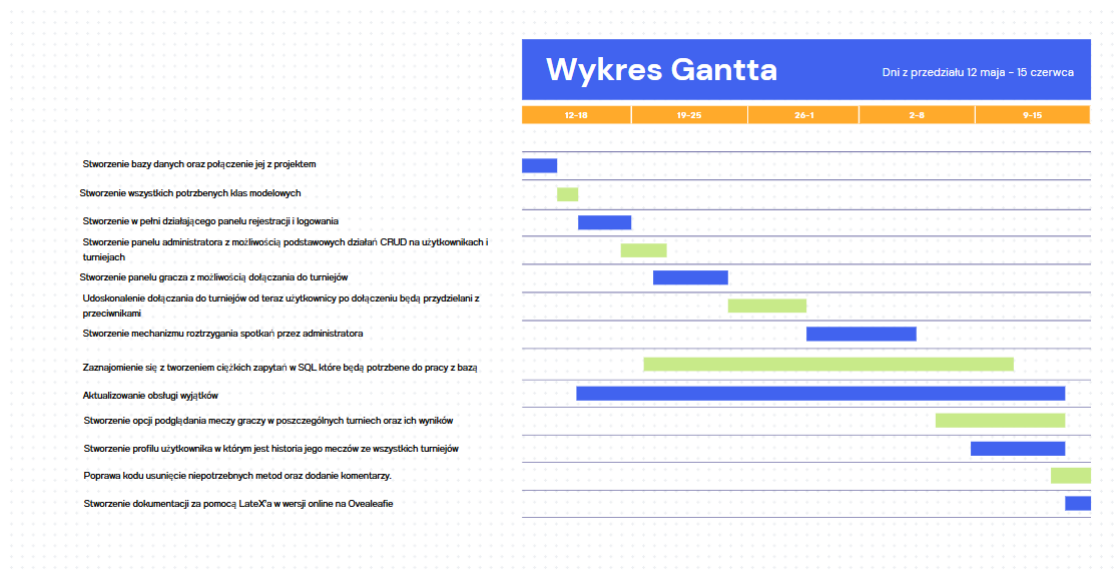
W tym rozdziale przedstawiono harmonogram realizacji projektu. Znajdziesz tu wizualizację w postaci Wykresu Gantta, prezentującą etapy tworzenia projektu wraz z poświęconym na nie czasem. Omówione zostaną również trudności, jakie wystąpiły podczas realizacji, oraz Repozytorium, na którym znajduje się projekt.

4.1. Trudności w projekcie

Projekt potrafił przysporzyć wiele problemów, a największym z nich była obszerność projektu, którą początkowo źle oszacowano. Projekt miał początkowo zakładać, że turnieje szachowe będą odbywać się w postaci drabinki. Jednak na początkowym etapie tworzenia koncepcja została zmieniona na turnieje "każdy z każdym", ponieważ wydawało się to łatwiejszym podejściem.

Zmiana podejścia wymagała modyfikacji, gdyż na tym etapie była już stworzona baza danych, którą trzeba było odpowiednio edytować, aby pasowała do nowej koncepcji projektu. Prace nad projektem po tych zmianach szły bardzo dobrze, aż do momentu, gdy trzeba było stworzyć zaawansowane zapytania w języku SQL, aby wyświetlać potrzebne dane w tabelach. Problem był na tyle kłopotliwy, ponieważ podczas pisania zapytań wyniki zawsze były nie do końca poprawne. Lecz po dłuższym czasie zdecydowano się na napisanie zaawansowanych zapytań SELECT przy użyciu wielu `**JOIN**`ów, co rozwiązało problem i pomogło przy późniejszym tworzeniu projektu.

Ostatnim problemem było wyrobienie się z deadline, ponieważ na dwa dni przed końcem brakowało całej dokumentacji. Jednak przy wykorzystaniu całego dostępnego czasu udało się wykonać projekt przed terminem.



Rys. 4.1. Wykres Gantta

4.2. Repozytorium i kontrola wersji

Projekt wraz z pełną dokumentacją oraz plikami źródłowymi jest przechowywany na publicznym repozytorium. Kontrola wersji przy pomocy Git pomogła przy projekcie w ten sposób, że umożliwiła wracanie do starszych rozwiązań, jeżeli nowe podejścia do problemów nie były wystarczające.

- GitHub.<https://github.com/JakubWawroUR/ChessTournamentManagementSystem>.

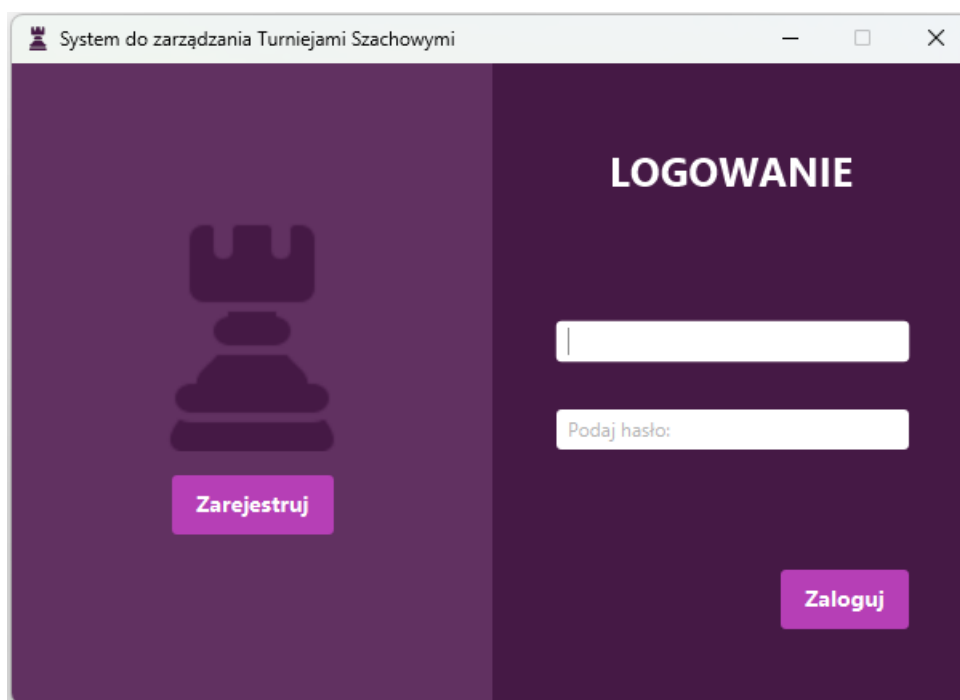
5. Prezentacja warstwy użytkowej projektu

W tym rozdziale zostanie przedstawione GUI systemu. Zaprezentujemy je poprzez zrzuty ekranu oraz szczegółowe opisy wszystkich scen w projekcie. Funkcjonalności systemu zostaną przedstawione w sposób intuicyjny i dokładny.

5.1. Ekran logowania

Pierwszą sceną, jaka wyświetla się użytkownikowi po uruchomieniu aplikacji, jest ekran logowania. Na scenie logowania można zauważyć:

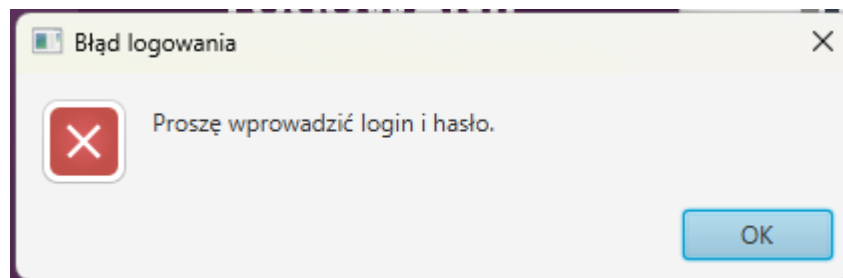
- Przycisk Zarejestruj jest odpowiedzialny za zmianę sceny na scenę rejestracji.
- Dwa pola tekstowe służące do logowania, w których podaje się odpowiednio login i hasło.
- Przycisk odpowiedzialny za zatwierdzenie danych z pól tekstowych do próby zalogowania.



Rys. 5.1. Ekran logowania.

5.1.1. Logowanie komunikat - 1

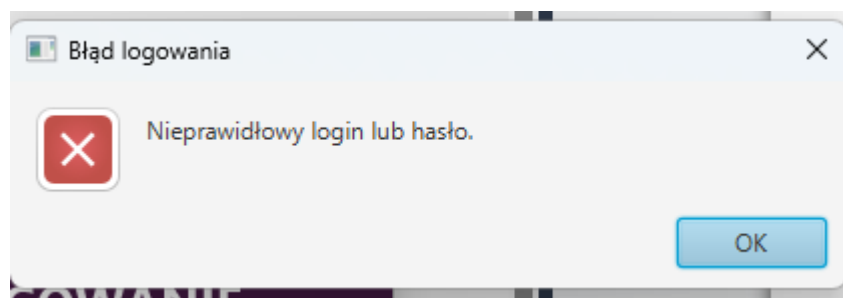
Komunikat przy próbie zalogowania się przez użytkownika gdy wprowadzi danych wyświetli się komunikat zwrotny:



Rys. 5.2. Ekran logowania - komunikat o braku wprowadzenie danych

5.1.2. Logowanie komunikat - 2

Komunikat wyświetlany przy próbie zalogowania się przez użytkownika, gdy wprowadzi dane niepasujące do żadnego istniejącego użytkownika.

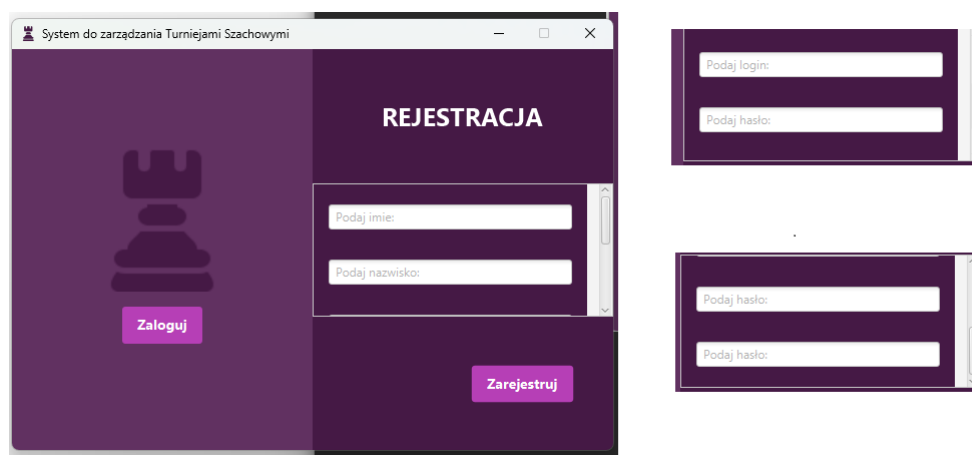


Rys. 5.3. Ekran logowania - komunikat o wprowadzenie błędnych danych

5.2. Ekran rejestracji

Scena, która uruchamia się z ekranu logowania poprzez kliknięcie przycisku Zarejestruj, to ekran rejestracji. Służy on do rejestracji użytkownika w bazie danych. Na scenie znajdują się:

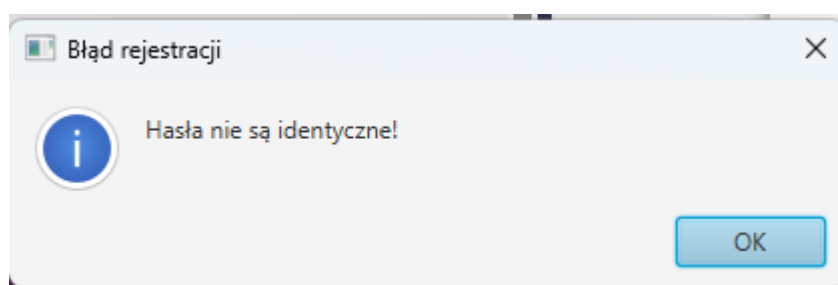
- Przycisk Zaloguj odpowiedzialny za zmianę sceny na scenę logowania.
- Pięć pól tekstowych służących do rejestracji. Pola te odpowiadają odpowiednio za imię, nazwisko, login, hasło oraz potwierdzenie hasła.
- Przycisk odpowiedzialny za zatwierdzenie danych z pól tekstowych do próby rejestracji.



Rys. 5.4. Ekran logowania - komunikat o wprowadzenie błędnych danych

5.2.1. Rejestracja komunikat - 1

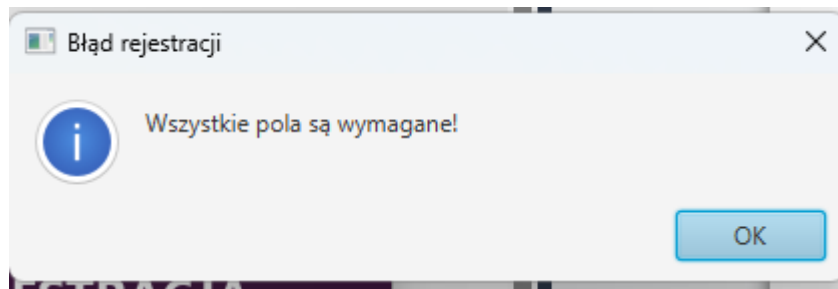
Komunikat wyświetlany przy próbie zarejestrowania się przez użytkownika, gdy wprowadzi różne hasła.



Rys. 5.5. Ekran rejestracji - komunikat o wprowadzeniu różnych haseł

5.2.2. Rejestracja komunikat - 2

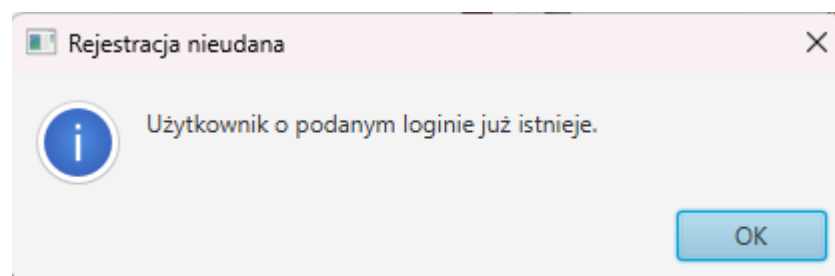
Komunikat wyświetlany przy próbie zarejestrowania się przez użytkownika, gdy nie uzupełni całego formularza.



Rys. 5.6. Ekran rejestracji - komunikat o niepełnym wypełnieniu formularzu

5.2.3. Rejestracja komunikat - 3

Komunikat wyświetlany przy próbie zarejestrowania się przez użytkownika, gdy ustali taki sam login, jak istniejący w bazie danych.

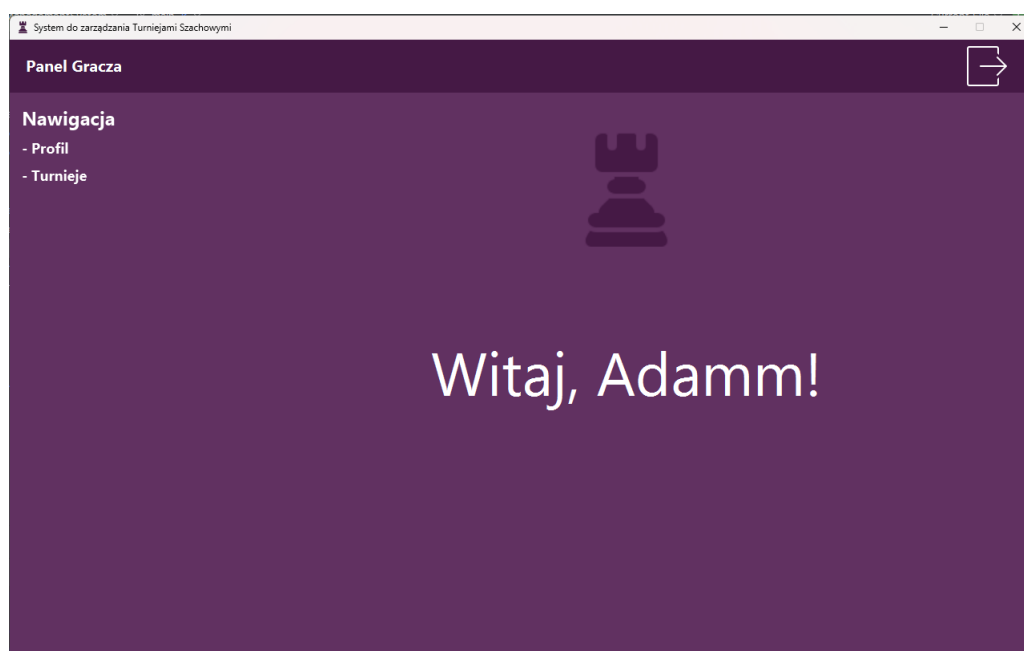


Rys. 5.7. Ekran rejestracji - komunikat o próbie zarejestrowaniu użytkownika o istniejącym loginie

5.3. Ekran powitalny gracza

Scena, która uruchamia się po zalogowaniu się przez użytkownika z rolą 'Gracz', to PlayerDashboard. Jest to scena powitalna użytkownika, zawierająca kilka elementów, z którymi można wejść w interakcję, takich jak:

- Przycisk w prawym górnym rogu, przedstawiony za pomocą ikony, jest odpowiedzialny za wylogowanie się z systemu i tym samym przenosi użytkownika do ekranu logowania.
- Dwa odnośniki pod napisem Nawigacja po lewej stronie aplikacji. Przenoszą one odpowiednio na scenę własnego profilu gracza oraz na scenę, na której znajdują się turnieje.

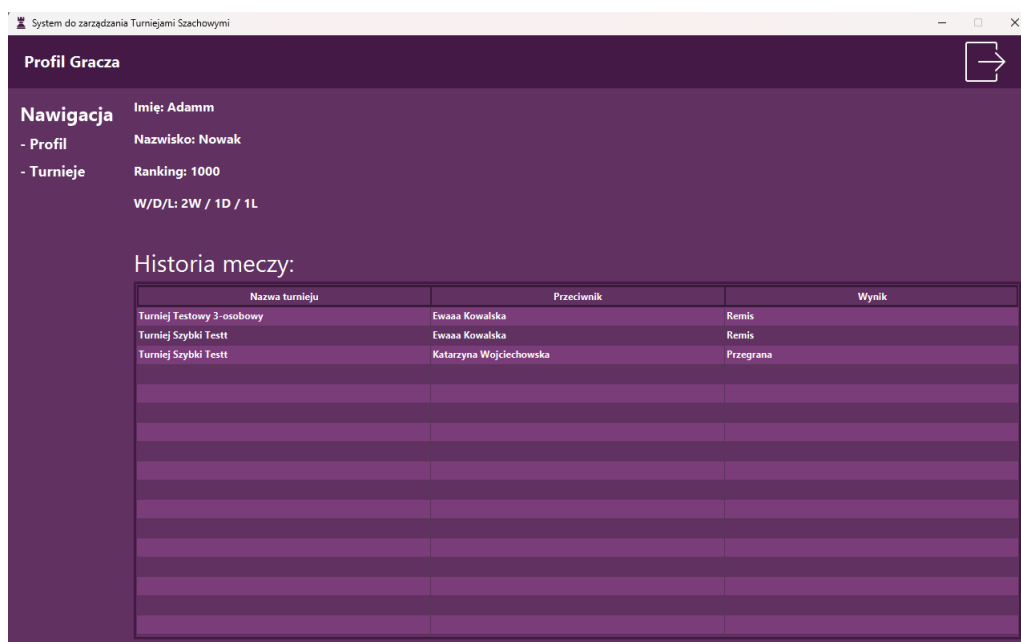


Rys. 5.8. Ekran powitalny użytkownika

5.4. Ekran profilu gracza

Scena, która uruchamia się po kliknięciu na odnośnik "Profil", to profil użytkownika. Przedstawia on tabelę będącą historią meczów w turniejach. Nad historią meczów wypisane są:

- Imię naszego użytkownika.
- Nazwisko naszego użytkownika.
- Ranking naszego użytkownika.
- W/D/L czyli łączną liczbę naszych: Wygranych/Remisów/Porażek (Nie zwracaj uwagi na dane, ponieważ zostały zmodyfikowane w bazie; w normalnych warunkach będą wyświetlać poprawne wartości.)

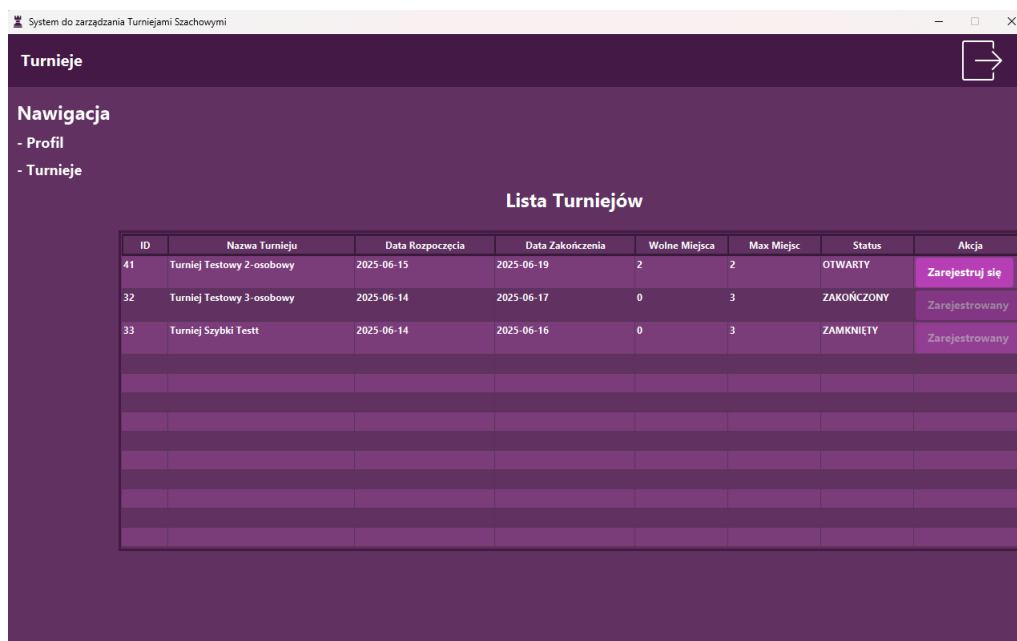


Nazwa turnieju	Przeciwnik	Wynik
Turniej Testowy 3-osobowy	Ewaaa Kowalska	Remis
Turniej Szybki Testt	Ewaaa Kowalska	Remis
Turniej Szybki Testt	Katarzyna Wojciechowska	Przegrana

Rys. 5.9. Profil gracza

5.5. Ekran turniejów

Scena, która uruchamia się po kliknięciu na odnośnik "Turnieje", przedstawia tabelę z danymi turniejów oraz przyciskami, z którymi można wejść w interakcję. Interakcję z przyciskami można podjąć, jeśli są wolne miejsca do dołączenia i użytkownik jeszcze do niego nie dołączył; w przeciwnym razie przycisk będzie zdezaktywowany. Można również podjąć interakcję z każdym turniejem, klikając na niego dwukrotnie, co przeniesie użytkownika do ekranu ze szczegółowymi informacjami o turnieju.

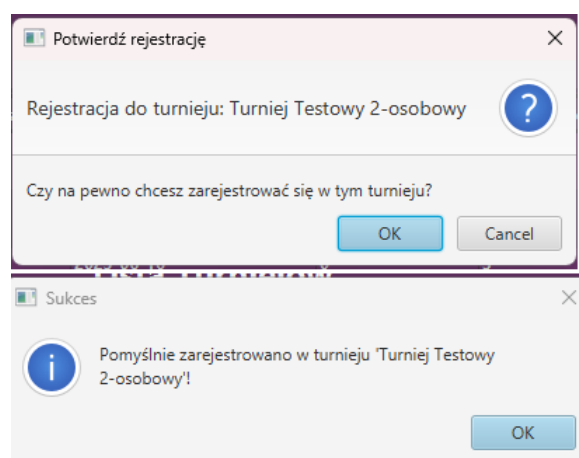


ID	Nazwa Turnieju	Data Rozpoczęcia	Data Zakończenia	Wolne Miejsca	Max Miejsc	Status	Akcja
41	Turniej Testowy 2-osobowy	2025-06-15	2025-06-19	2	2	OTWARTY	Zarejestruj się
32	Turniej Testowy 3-osobowy	2025-06-14	2025-06-17	0	3	ZAKOŃCZONY	Zarejestrowany
33	Turniej Szybki Test	2025-06-14	2025-06-16	0	3	ZAMKNIĘTY	Zarejestrowany

Rys. 5.10. Turnieje

5.5.1. Ekran turniejów komunikat - 1

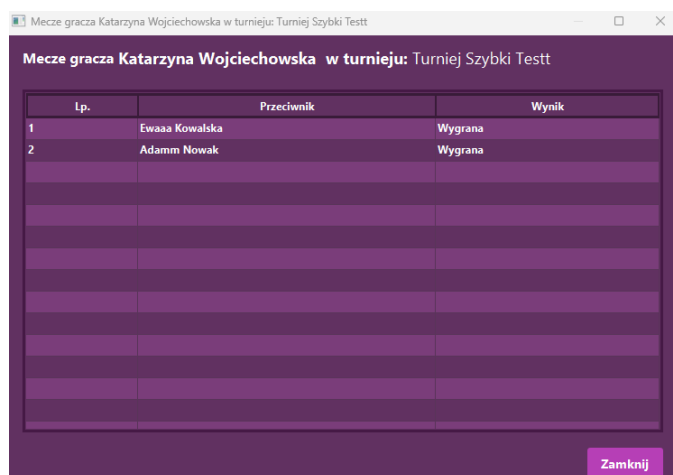
Komunikat wyświetlany przy próbie dołączenia do turnieju, gdy przycisk wskazuje na możliwość dołączenia do turnieju.



Rys. 5.11. Turnieje komunikat o dołączeniu do turnieju

5.6.1. Ekran turniejów z szczegółami - wyniki poszczególnego gracza

Scena, która uruchamia się po kliknięciu przycisku "Pokaż Mecze", przedstawia wszystkie mecze danego gracza w konkretnym turnieju.

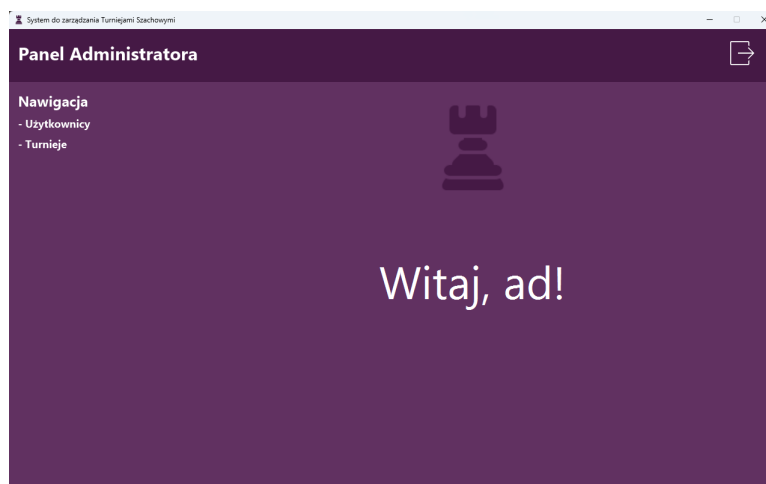


Lp.	Przeciwnik	Wynik
1	Ewaaa Kowalska	Wygrana
2	Adam Nowak	Wygrana

Rys. 5.13. Wszystkie mecze zawodnika w danym turnieju

5.7. Ekran powitalny administratora

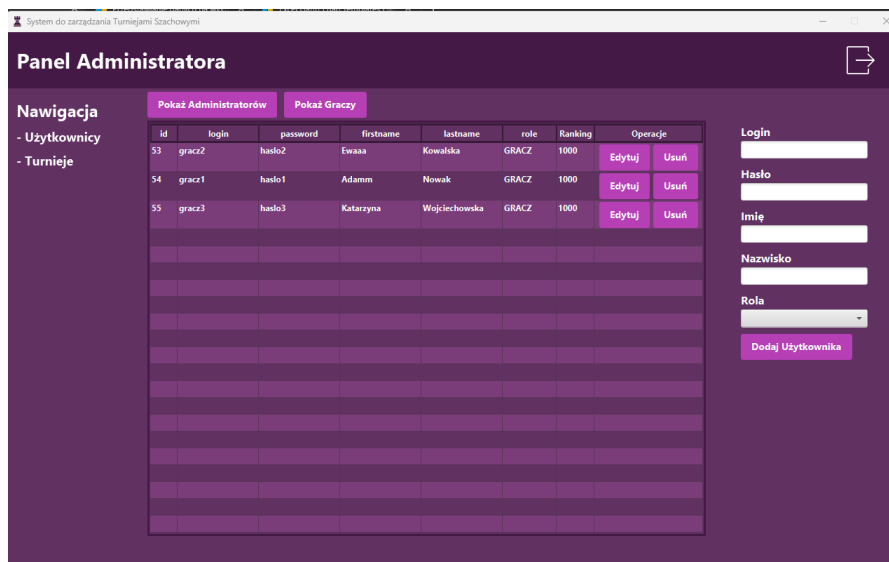
Ekran powitalny administratora jest niemal identyczny z panelem gracza; różni się tym, że odnośniki są inne i kierują w inne miejsca.



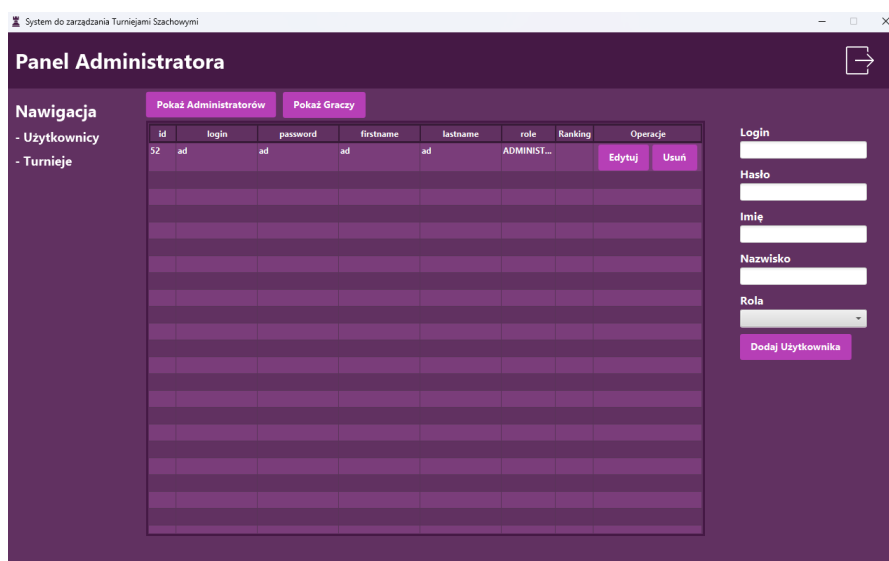
Rys. 5.14. Ekran powitalny administratora

5.8. Ekran zarządzania użytkownikami

Ekran do zarządzania użytkownikami włącza się poprzez kliknięcie "Użytkownicy" pod nawigacją. Domyślnie ekran pokazuje tylko Graczy, natomiast nad tabelą można przełączyć widok na Administratorów. Po prawej stronie znajdują się pola służące do operacji CRUD.



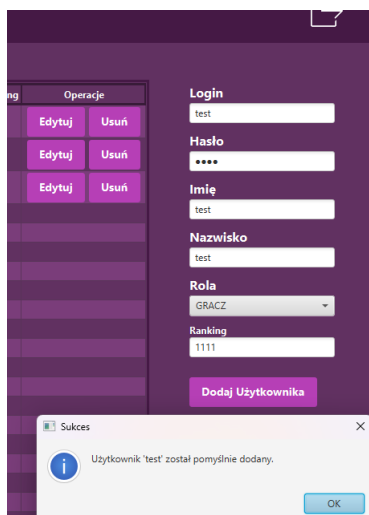
Rys. 5.15. Ekran do zarządzania graczami



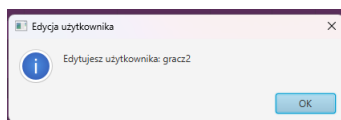
Rys. 5.16. Ekran do zarządzania administratorami

5.8.1. Ekran do zarządzania użytkownikami - Alerty

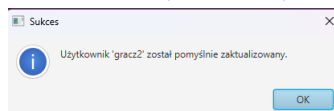
Alerty występujące przy manipulacji danymi użytkowników.



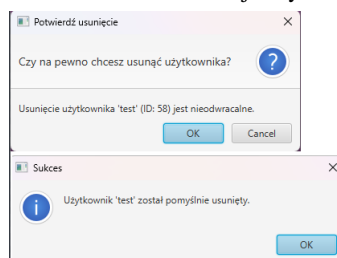
Rys. 5.17. Ekran do zarządzania administratorami



Rys. 5.18. Alert - Edytowanie użytkownika



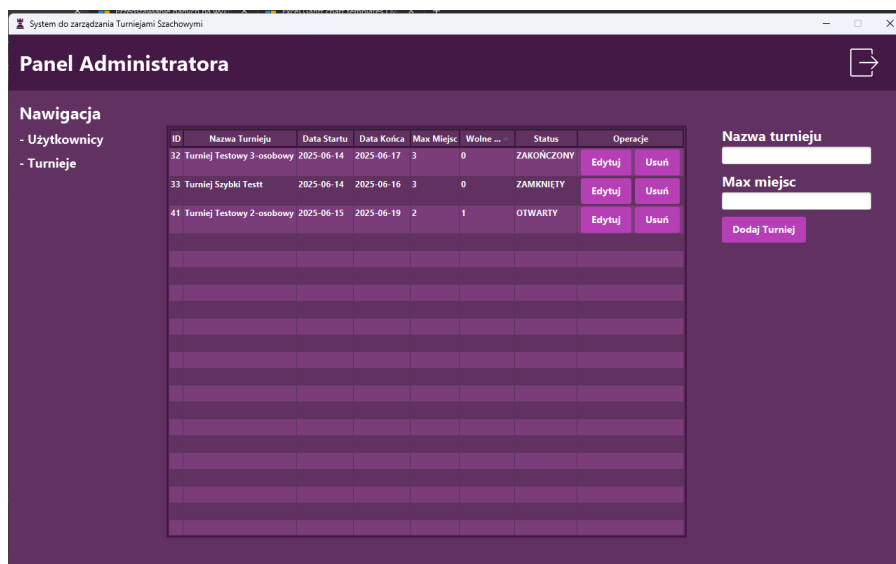
Rys. 5.19. Alert - Aktualizacja użytkownika



Rys. 5.20. Alert - Usuwanie użytkownika

5.9. Ekran zarządzania turniejami

Ekran do zarządzania turniejami włącza się poprzez kliknięcie "Turnieje" pod nawigacją. Administrator może wykonywać operacje CRUD na tabeli turniejów za pośrednictwem GUI. Podwójne kliknięcie na turniej uruchamia nowe okno.



Rys. 5.21. Ekran do zarządzania turniejami

5.9.1. Ekran zarządzania turniejami - wyświetlenie wszystkich gier turnieju

Ekran wyświetla wszystkie gry, a obok nich przycisk "Rozstrzygnij". Gdy administrator go naciśnie, włącza się nowe okno.



Rys. 5.22. Ekran do zarządzania administratorami

5.9.1.1. Ekran zarządzania turniejami - wyświetlenie wszystkich gier turnieju - rozstrzygnięcie spotkań

Ekran służy do rozstrzygnięcia poszczególnych meczów między dwoma graczami.



Rys. 5.23. Ekran do rozstrzygnięcia meczy

6. Podsumowanie

Projekt to system do zarządzania turniejami szachowymi, idealnie sprawdzający się w turniejach odbywających się stacjonarnie. System oferuje zarówno funkcje administracyjne, jak i użytkowe dla graczy.

Administratorzy mogą samodzielnie tworzyć, edytować i usuwać turnieje szachowe, a także rozstrzygać poszczególne spotkania w turniejach. Mają również możliwość dodawania, usuwania i edycji użytkowników. Dla graczy dostępne są funkcje takie jak dołączanie do turniejów oraz sprawdzanie swojego profilu, w tym meczów oraz statystyk. System posiada w pełni działające funkcje autoryzacji zarówno dla administratorów, jak i graczy.

Projekt można rozwinąć pod kątem dodania nowych typów turniejów, np. drabinkowych. Kolejnym dobrym pomysłem jest umożliwienie graczom tworzenia pokoi do pojedynczych meczów. Świetnym usprawnieniem byłaby również automatyzacja rozstrzygania meczów poprzez API, co pozwoliłoby systemowi działać także jako hub gier online.

Spis rysunków

3.1	Diagram EER bazy danych systemu zarządzania turniejami szachowymi.	13
3.2	DIAGRAM UML - Logout	17
3.3	DIAGRAM UML - BaseDashboardController	17
3.4	DIAGRAM UML - User	17
4.1	Wykres Gantta	18
5.1	Ekran logowania.	20
5.2	Ekran logowania - komunikat o braku wprowadzenie danych	21
5.3	Ekran logowania - komunikat o wprowadzenie błędnych danych	21
5.4	Ekran logowania - komunikat o wprowadzenie błędnych danych	22
5.5	Ekran rejestracji - komunikat o wprowadzeniu różnych haseł	22
5.6	Ekran rejestracji - komunikat o niepełnym wypełnieniu formularzu	23
5.7	Ekran rejestracji - komunikat o próbie zarejestrowaniu użytkownika o istniejącym loginie .	23
5.8	Ekran powitalny użytkownika	24
5.9	Profil gracza	25
5.10	Turnieje	26
5.11	Turnieje komunikat o dołączeniu do turnieju	26
5.12	Turnieje szczegóły	27
5.13	Wszystkie mecze zawodnika w danym turnieju	28
5.14	Ekran powitalny administratora	28
5.15	Ekran do zarządzania graczami	29
5.16	Ekran do zarządzania administratorami	29
5.17	Ekran do zarządzania administratorami	30
5.18	Alert - Edytowanie użytkownika	30
5.19	Alert - Aktualizacja użytkownika	30
5.20	Alert - Usuwanie użytkownika	30
5.21	Ekran do zarządzania turniejami	31
5.22	Ekran do zarządzania administratorami	31
5.23	Ekran do rozstrzygania meczy	32

Załącznik nr 2 do Zarządzenia nr 228/2021 Rektora Uniwersytetu Rzeszowskiego z dnia 1 grudnia 2021 roku w sprawie ustalenia procedury antyplagiatowej w Uniwersytecie Rzeszowskim

OŚWIADCZENIE STUDENTA O SAMODZIELNOŚCI PRACY

..... Jakub Wawro
Imię (imiona) i nazwisko studenta

Wydział Nauk Ścisłych i Technicznych

..... Informatyka
Nazwa kierunku

..... 134984
Numer albumu

1. Oświadczam, że moja praca projektowa pt.: System zarządzania turniejami szachowymi
 - 1) została przygotowana przeze mnie samodzielnie*,
 - 2) nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2021 r., poz. 1062) oraz dóbr osobistych chronionych prawem cywilnym,
 - 3) nie zawiera danych i informacji, które uzyskałem/am w sposób niedozwolony,
 - 4) nie była podstawą otrzymania oceny z innego przedmiotu na uczelni wyższej ani mnie, ani innej osobie.
2. Jednocześnie wyrażam zgodę/~~nie wyrażam zgody~~** na udostępnienie mojej pracy projektowej do celów naukowo-badawczych z poszanowaniem przepisów ustawy o prawie autorskim i prawach pokrewnych.

Gorliczyna 15.06.2025
(miejscowość, data)

Jakub Wawro
(czytelny podpis studenta)

* Uwzględniając merytoryczny wkład prowadzącego przedmiot

** – niepotrzebne skreślić