Dokumentacja projektowa

Zarządzanie Systemami Informatycznymi

Zarządzanie projektem i koordynacja zespołu developerskiego przy tworzeniu aplikacji webowej z użyciem Gita, Trello i Dockera.

Kierunek: Informatyka

Członkowie zespołu:
Paweł Lewandowski
Michał Kuta

Gliwice, 2024/2025

Spis treści

1 Informacje o zespole

Podział ról, zadań i wkładu każdego członka zespołu.

1.1 Role

- Paweł Lewandowski – organizator prac, projektant bazy danych, osoba odpowiedzialna za dokumentację i testowanie
- Michał Kuta – Główny programista, inżynier back-endu, architekt systemu

1.2 Zaplanowane zadania

- Paweł Lewandowski – zarządzanie Trello, planowanie sprintów, projektowanie schematu bazy danych, dokumentacja projektu, testowanie
- Michał Kuta – stworzenie szkieletu projektu, implementacja architektury MVC, logika back-endu, połączenie z bazą danych, testy wstępne

1.3 Zrealizowane zadania

- Paweł Lewandowski – dokumentacja projektu (techniczna i użytkowa), UML bazy danych, backlog sprintów, koordynacja, testowanie aplikacji
- Michał Kuta – szkielet aplikacji, pełna implementacja wzorca MVC, integracja PDO, logika back-endu, połączenie warstw, testowanie funkcjonalności

2 Wymagania projektowe

2.1 Wymagania techniczne i nietechniczne

Wymagania techniczne

- Aplikacja webowa oparta na wzorcu projektowym MVC.
- Backend aplikacji zaimplementowany w języku PHP.
- Uruchamianie aplikacji w kontenerach Docker za pomocą `docker-compose`.
- Warstwa prezentacji oparta na szablonach HTML i CSS.
- Wykorzystanie relacyjnej bazy danych MySQL.
- Kontrola wersji i współpraca zespołowa przez GitHub.

Wymagania nietechniczne

- Intuicyjny i prosty w obsłudze interfejs użytkownika, dostosowany do komputerów i tabletów.
- Możliwość rejestrowania wyników wielu graczy w różnych grach planszowych.
- Funkcjonalność przeglądania historii rozgrywek oraz statystyk.
- Łatwa rozbudowa o nowe funkcje w przyszłości.
- Niezawodność i łatwa konserwacja aplikacji.

2.2 TELOS

- **Techniczne** – Wykorzystanie sprawdzonych technologii (PHP, MySQL, Docker), zapewniających przenośność i zgodność.
- **Ekonomiczne** – Wszystkie używane narzędzia są darmowe i open-source, co obniża koszty projektu.
- **Prawne** – Projekt nie przetwarza danych osobowych, a używane oprogramowanie spełnia warunki licencji open-source.
- **Organizacyjne** – Aplikacja spełnia zakładane funkcje, jest łatwa w obsłudze i wdrożeniu.
- **Harmonogramowe** – Projekt został podzielony na pięć sprintów i zakończony zgodnie z założonym harmonogramem.

2.3 SWOT

- **Mocne strony (S)** – Modułowa architektura, konteneryzacja w Dockerze, system kontroli wersji Git.
- **Słabe strony (W)** – Brak zaawansowanego frontendu i funkcji logowania użytkowników.
- **Szanse (O)** – Możliwość łatwej rozbudowy o nowe funkcjonalności, np. logowanie, statystyki, wersję mobilną.
- **Zagrożenia (T)** – Możliwe problemy ze skalowalnością bazy danych, zmiany w obsługiwanych wersjach PHP lub Dockera.

2.4 Oczekiwane rezultaty projektu

- W pełni funkcjonalna aplikacja webowa umożliwiająca rejestrowanie wyników gier planszowych.
- Przejrzysty i łatwy w obsłudze interfejs użytkownika.
- Spójny kod aplikacji oparty na wzorcu MVC.
- Środowisko uruchomieniowe oparte na kontenerach Docker.
- Kompletna dokumentacja techniczna i użytkowa aplikacji.

2.5 Stos technologiczny

- Frontend: HTML, CSS.
- Backend: PHP z wykorzystaniem wzorca MVC.
- Baza danych: MySQL.
- System kontroli wersji: Git, GitHub.
- Zarządzanie: Trello.
- Konteneryzacja: Docker, Docker Compose.

2.6 Oczekiwane rezultaty projektu

- W pełni funkcjonalna aplikacja webowa umożliwiająca rejestrowanie wyników gier planszowych.
- Intuicyjny interfejs użytkownika umożliwiający łatwe dodawanie i przeglądanie rozgrywek.
- System zapisu danych z wykorzystaniem bazy danych (MySQL).
- Środowisko uruchomieniowe oparte na Dockerze.
- Dokumentacja techniczna i użytkowa aplikacji.

3 Przegląd projektu

Zestawienie narzędzi i systemów informatycznych wykorzystanych w projekcie. Każde narzędzie zostało dobrane zgodnie z wymaganiami projektu i kompetencjami zespołu. Poniżej przedstawiono cel ich wykorzystania.

3.1 Zarządzanie projektem

- **Trello** – wykorzystane do zarządzania zadaniami w modelu Kanban. Umożliwiało przejrzyste planowanie sprintów, monitorowanie postępów i priorytetyzację zadań.
- **Git i GitHub** – służyły do kontroli wersji, współdzielenia kodu i integracji pracy zespołowej. Umożliwiały śledzenie zmian, tworzenie branchy i przegląd kodu.

3.2 Metodologia

- **Agile** – elastyczne podejście do tworzenia oprogramowania, oparte na iteracyjnym rozwoju i regularnych retrospekcjach.
- **Scrum** – projekt podzielony został na sprinty, po których odbywały się przeglądy i aktualizacje planów.
- **Kanban** – wizualne przedstawienie zadań w Trello, ułatwiające zarządzanie przepływem pracy.

3.3 Back-End

- **PHP** – główny język użyty do tworzenia logiki aplikacji, oparty na wzorcu MVC.
- **MVC (Model-View-Controller)** – wzorzec projektowy umożliwiający oddzielenie logiki biznesowej od warstwy prezentacji.

3.4 Front-End

- **HTML i CSS** – wykorzystane do budowy i stylizacji interfejsu użytkownika, zapewniające prostotę i przejrzystość obsługi.

3.5 Baza danych

- **MySQL** – relacyjna baza danych, wykorzystywana do przechowywania danych o graczach, grach i wynikach rozgrywek.
- **PDO (PHP Data Objects)** – warstwa pośrednia do komunikacji z bazą danych, zapewniająca bezpieczeństwo i elastyczność zapytań.

3.6 Konteneryzacja i środowisko uruchomieniowe

- **Docker** – konteneryzacja aplikacji, zapewniająca spójne środowisko uruchomieniowe niezależne od systemu operacyjnego.
- **Docker Compose** – umożliwia jednoczesne uruchamianie aplikacji i bazy danych z poziomu jednego pliku konfiguracyjnego.

4 Realizacja projektu

Projekt został zrealizowany w następujących etapach:

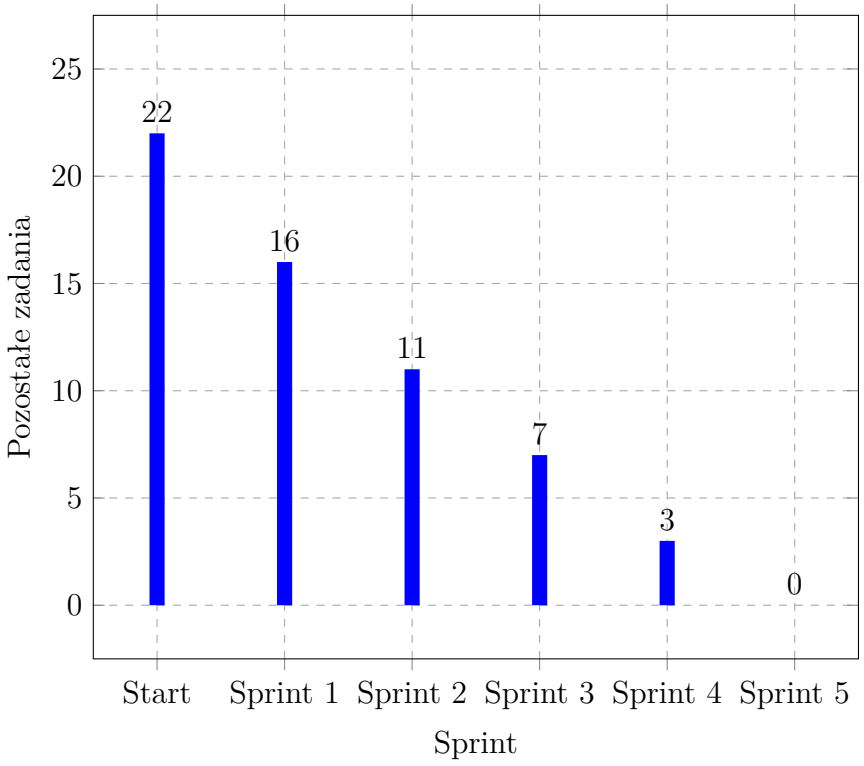
1. **Sprint 1 – Planowanie:**
    - Identyfikacja funkcjonalności aplikacji (dodawanie gier, graczy, przeglądanie rekordów).
    - Utworzenie tablicy Kanban w Trello do zarządzania zadaniami.
    - Wybór stosu technologicznego: PHP + MySQL + Docker, z wykorzystaniem wzorca MVC.
    - Zaplanowanie struktury katalogów i komponentów aplikacji.

- Opracowanie planu sprintów oraz roadmapy rozwoju.

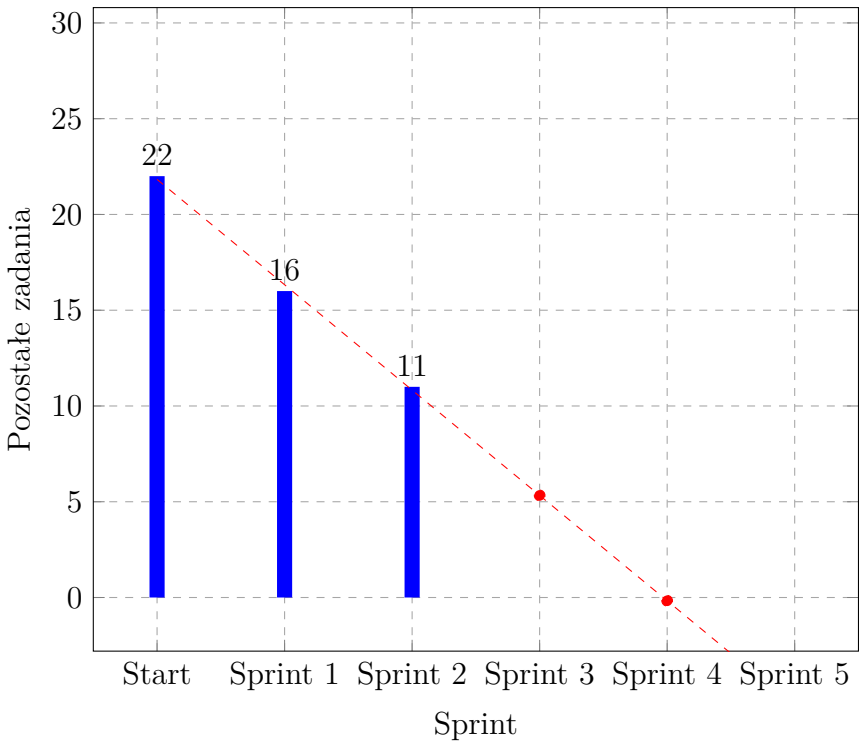
2. **Sprint 2 – Tworzenie podstaw:**
   - Utworzenie środowiska kontenerowego Docker (PHP + MySQL).
   - Implementacja szkieletu aplikacji zgodnie z wzorcem MVC.
   - Stworzenie warstwy modelu i połączenia z bazą danych.
   - Testowanie komunikacji z bazą MySQL za pomocą PDO.
   - Przygotowanie roboczych widoków do testowego wyświetlania danych.
3. **Sprint 3 – Główne programowanie:**
   - Implementacja logiki warstwy kontrolera.
   - Zastąpienie widoków roboczych finalnymi szablonami HTML i CSS.
   - Połączenie warstw modelu, widoku i kontrolera w spójną całość.
   - Wprowadzenie mechanizmów obsługi błędów.
4. **Sprint 4 – Testowanie:**
   - Intensywne testowanie aplikacji pod kątem poprawności działania.
   - Poprawa wykrytych błędów i usprawnienie mechanizmów walidacji.
   - Weryfikacja stabilności komunikacji między warstwami aplikacji.
   - Testy środowiska Docker (np. restart kontenerów, poprawność konfiguracji).
5. **Sprint 5 – Faza końcowa:**
   - Uzupełnienie i uporządkowanie dokumentacji technicznej projektu.
   - Weryfikacja wykonania zadań z tablicy Kanban.
   - Finalne poprawki w kodzie oraz przygotowanie aplikacji do prezentacji.

5 Wykresy postępu projektu

Poniżej przedstawiono dwa wykresy ilustrujące postęp prac nad projektem w kolejnych sprintach.



Rysunek 1: Wykres liczby pozostałych zadań w kolejnych sprintach.



Rysunek 2: Wykres regresji liniowej i przewidywań na przyszłe sprinty.

Według tego wykresu powinniśmy zakończyć projekt przy sprincie 4.

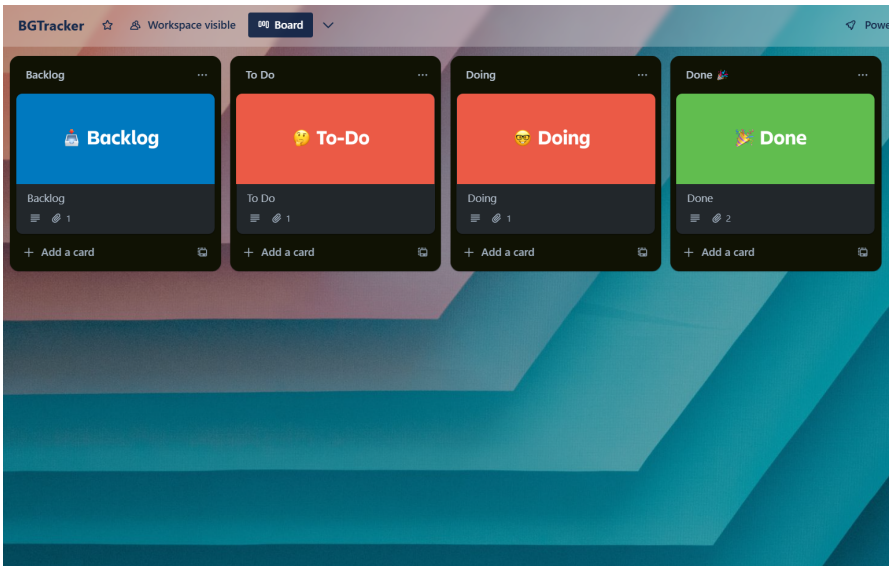
6 Trello – zarządzanie zadaniami

Do zarządzania zadaniami projektowymi wykorzystano narzędzie **Trello**, zgodnie z podejściem Kanban i metodologią Scrum. Na początku projektu utworzono kolumnę **Backlog**, zawierającą wszystkie zaplanowane zadania, zarówno techniczne, jak i organizacyjne. Następnie zadania z backlogu zostały podzielone na sprinty.

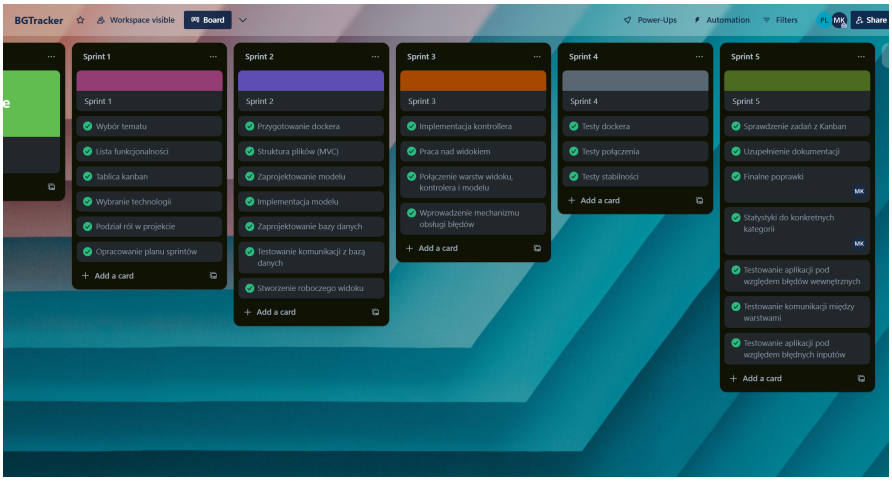
W ramach każdego sprintu zadania były przenoszone kolejno przez kolumny: **To do**, **Doing** oraz **Done**, co umożliwiało pełną wizualizację postępu prac oraz bieżące monitorowanie realizacji zadań.

Po zakończeniu danego sprintu, wszystkie zadania z kolumny **Done** danego sprintu wracały do kolumny sprintu, a cała kolumna była przenoszona na prawą część tablicy. Taki zabieg pozwalał zachować czytelność tablicy oraz śledzić historię postępów w czasie całego cyklu życia projektu.

Poniżej przedstawiono przykładowe zrzuty ekranu z tablicy Trello wykorzystywanej w naszym projekcie:



Rysunek 3: kolumny: **Backlog**, **To do**, **Doing** oraz **Done**



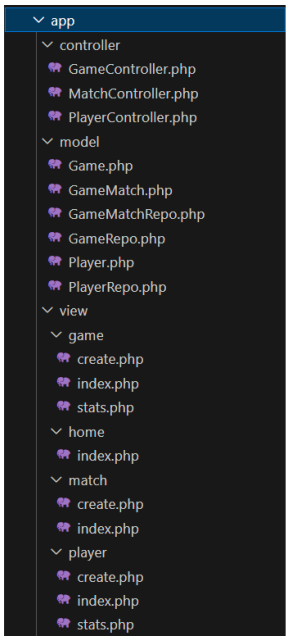
Rysunek 4: kolumny sprintów

7 Architektura MVC

Projekt został zaprojektowany zgodnie z wzorcem architektonicznym **Model-View-Controller (MVC)**, który pozwala na przejrzyste oddzielenie logiki biznesowej od warstwy prezentacji oraz obsługi żądań.

- **Model** – znajduje się w folderze `model` i zawiera klasy reprezentujące dane (`Game.php`, `Player.php`, `GameMatch.php`) oraz klasy obsługujące statystyki (`GameRepo.php`, `PlayerRepo.php`, `GameMatchRepo.php`) odpowiedzialne za interakcję z bazą danych.
- **Controller** – folder `controller` zawiera klasy odpowiedzialne za przetwarzanie logiki aplikacji i komunikację między modelem a widokiem: `GameController.php`, `MatchController.php` `PlayerController.php`.
- **View** – folder `view` przechowuje pliki `.php`, które odpowiadają za generowanie interfejsu użytkownika. Widoki są pogrupowane tematycznie w podfolderach: `game`, `match`, `player` oraz `home`.

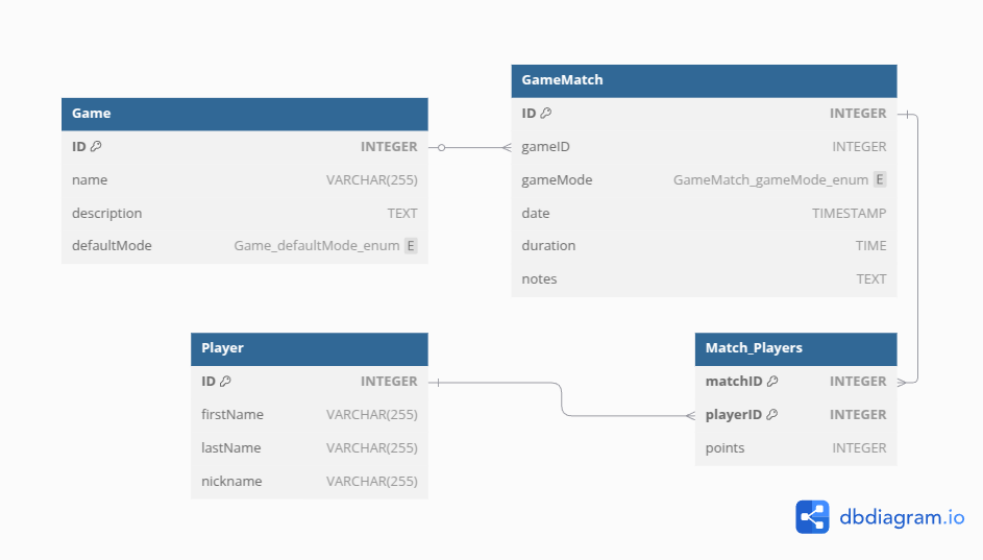
Poniższy zrzut ekranu przedstawia strukturę katalogów w projekcie, odzwierciedlającą implementację wzorca MVC:



Rysunek 5: Struktura katalogów aplikacji oparta na wzorcu MVC

8 Schemat bazy danych

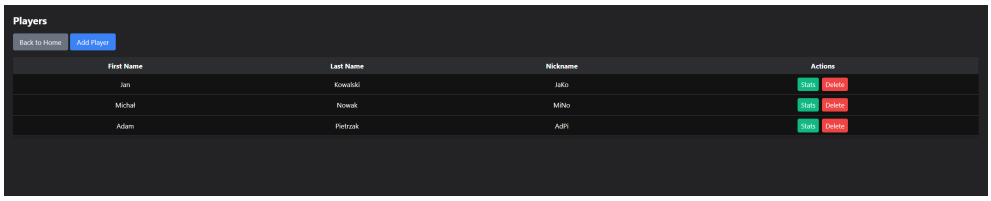
W celu odwzorowania relacji pomiędzy encjami w naszym systemie, przygotowano diagram bazy danych w notacji UML. Baza danych została zaprojektowana w sposób umożliwiający przechowywanie informacji o grach, graczach oraz rozegranych meczach. Diagram uwzględnia kluczowe encje: **Game**, **Player**, **GameMatch** oraz tabelę pośredniczącą **MatchPlayers**, która realizuje relację wiele-do-wielu pomiędzy meczami a graczami. Diagram ten stanowi podstawę dalszego projektowania warstwy logicznej aplikacji oraz implementacji komunikacji z bazą danych.



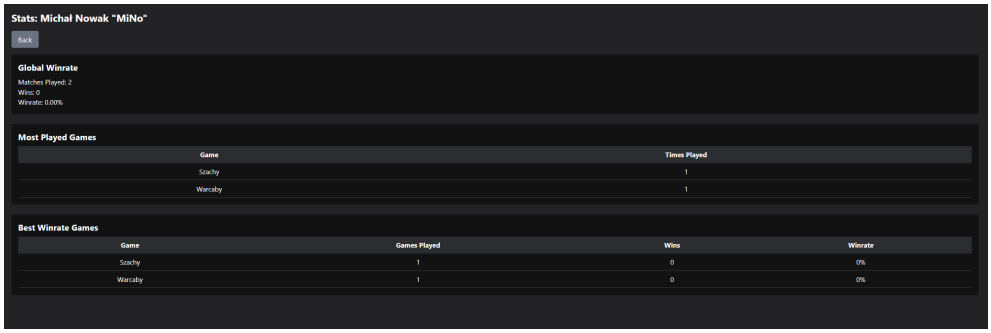
Rysunek 6: Diagram bazy danych

9 Ukończona aplikacja

9.1 Gracze

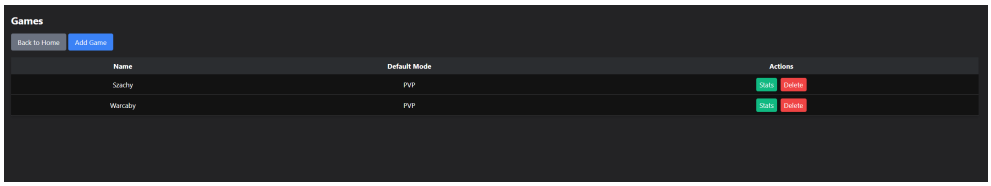


Rysunek 7: Tabela graczy

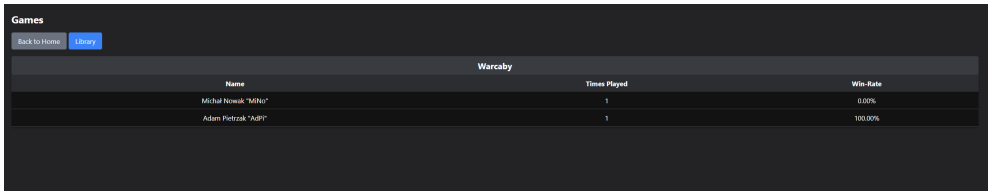


Rysunek 8: Statystyki gracza

9.2 Gry

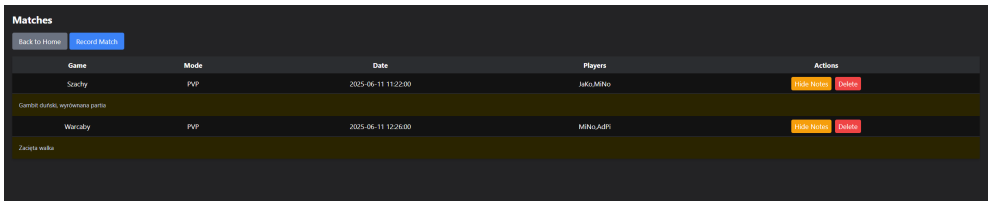


Rysunek 9: Tabela gier



Rysunek 10: Statystyki gry

9.3 Mecze



Rysunek 11: Tabela meczy

10 Wnioski

- **Spostrzeżenia:** Praca nad aplikacją umożliwiła pogłębienie wiedzy z zakresu wzorca MVC, konteneryzacji za pomocą Dockera oraz pracy zespołowej z wykorzystaniem narzędzi takich jak Trello i GitHub. Projekt unaocznił także znaczenie jasnego podziału obowiązków oraz dobrej dokumentacji.
- **Osiągnięcia:** Udało się zbudować w pełni działającą aplikację webową, umożliwiającą rejestrowanie wyników gier planszowych w przejrzysty i funkcjonalny sposób. Stworzono modułowy kod z zachowaniem zasad separacji warstw, a całość została uruchomiona w środowisku Docker, co ułatwia wdrażanie i testowanie.
- **Potencjał rozwoju:** Projekt może być w przyszłości rozwijany o nowe funkcjonalności, takie jak: logowanie użytkowników, historia rozgrywek z filtrowaniem, generowanie statystyk i wykresów, eksport danych do plików (CSV/PDF), integracja z kontami Google lub Facebook, czy też wersja mobilna aplikacji.

11 Bibliografia

(a) Oficjalna dokumentacja PHP – https://www.php.net/docs.php

(b) Dokumentacja Docker – https://docs.docker.com/

(c) Dokumentacja MySQL – https://dev.mysql.com/doc/

(d) Artykuł: „Understanding MVC Architecture" – https://www.geeksforgeeks.org/mvc-design-pattern/

(e) Oficjalna dokumentacja PDO – https://www.php.net/manual/en/book.pdo.php

(f) Dokumentacja Git – https://git-scm.com/doc

(g) Trello – https://trello.com

(h) dbdiagram.io – https://dbdiagram.io