Marafone

### **Autorzy: Tomasz Jarząbek, Norbert Koziana, Szymon Gryz**

### **2025**

**Plan projektu**

Opis projektu – założenia, cele projektowe, funkcje

Analiza systemowa

a) Diagram przypadków użycia

b) Diagramy sekwencji

c) Diagram stanu

d) Podział prac w projekcie

Projekt architektury

a) Wybór technologii

b) Projekt architektury aplikacji

c) Projekt bazy danych

d) Diagram klas

Implementacja i testowanie aplikacji z użyciem wybranego narzędzia do projektowania zespołowego

## **Opis projektu**

### **Założenia**

Aplikacja ma za zadanie realizację funkcjonalności pozwalającej na:

* obsługę użytkowników (rejestracja, logowanie, profile graczy),
* zarządzanie grami (tworzenie rozgrywek, dołączanie do stołów, obserwowanie gier),
* grę wieloosobową w czasie rzeczywistym w karty Marafone (2v2),
* integrację z systemem sztucznej inteligencji (gracze AI zastępujący ludzi),
* wyświetlanie statystyk graczy,
* obsługę dźwięków i animacji kart w celu zwiększenia immersji,
* system rankingowy i punktowy,
* możliwość wyboru języka interfejsu,
* integrację z WebSocket do dynamicznej aktualizacji stanu gry.

### **Cele projektowe**

Celem projektu jest stworzenie funkcjonalnego i nowoczesnego systemu informatycznego, umożliwiającego użytkownikom rozgrywkę w grę karcianą Marafone zarówno przeciwko innym graczom, jak i przeciwko komputerowym przeciwnikom (AI).

Główne cele to:

* zapewnienie płynnej i realistycznej rozgrywki karcianej w trybie online i lokalnym,
* umożliwienie nauki zasad gry nowym graczom,
* rozwój algorytmu sztucznej inteligencji umożliwiającego strategiczną grę AI w modelu samouczącego się gracza (Reinforcement Learning),
* promowanie gry Marafone oraz integracja społeczności graczy poprzez rankingi, statystyki i rozgrywki publiczne,

### **Funkcje**

System umożliwia:

* rejestrację i logowanie użytkowników,
* tworzenie, dołączanie do gier online,
* rozgrywkę w czasie rzeczywistym 2v2, w tym także z graczami AI,
* przestrzeganie zasad gry Marafone, w tym reguł dotyczących koloru atutowego, punktacji i kolejności rozgrywki,
* wyświetlanie punktów, rund, wyniku końcowego i statystyk,
* dźwięki i animacje kart zwiększające jakość użytkowania,
* dynamiczne aktualizacje stanu gry dzięki WebSocket,
* zarządzanie rankingiem graczy na podstawie wyników.

### **Zasady gry**

Gra rozgrywana jest talią 40 kart włoskich, uporządkowanych według kolejności 3 2 A K C J 7 6 5 4 (najsilniejsza z lewej) Pod względem punktów asy są warte pełny punkt, podczas gdy dwójki, trójki i karty dworskie są warte ⅓ punktu; wszystkie inne karty nie są warte żadnych punktów.

Każde rozdanie składa się z 10 tur, pod koniec rundy zdobyte punkty są zaokrąglane w dół do liczby całkowitej, a zwycięzca ostatniej tury otrzymuje 1 dodatkowy punkt. Mecz trwa, aż jedna drużyna osiągnie 41 punktów (w wariancie klasycznym).

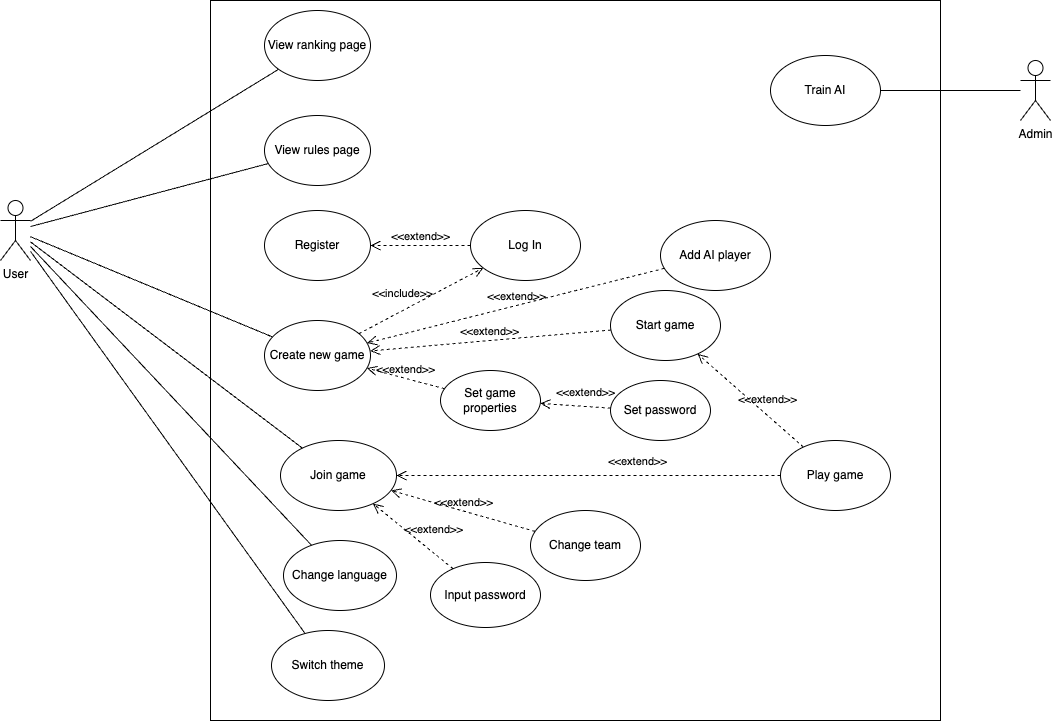
W pierwszym rozdaniu meczu gracz trzymający 4 monet chrzci kolor atutowy, zwany briscola, i rozpoczyna pierwszą turę. We wszystkich kolejnych rozdaniach gracz siedzący po prawej stronie rozdającego chrzci briscola.

Jeśli gracz trzyma maraffę (asa, dwójkę, trójkę) z koloru atutowego, może ją ogłosić, aby otrzymać 3 punkty bonusowe.

Gracze muszą dotrzymać koloru jeśli mogą, i dlatego mogą grać atut tylko wtedy, gdy nie posiadają żadnej karty w kolorze wiodącym. Tura jest przyznawana graczowi z najsilniejszą kartą w kolorze wiodącym, jeśli nie zagrano żadnego atutu, lub graczowi z najsilniejszym atutowym w przeciwnym wypadku. Zwycięzca tury musi rozpocząć następną turę. Pod koniec każdego rozdania punkty są podliczane, a gracz, który rozpoczął pierwszą turę, zostaje nowym rozdającym. Podczas gry nie wolno rozmawiać.

## **Analiza systemowa**

### **a) Diagram przypadków użycia**



### **b) Diagramy sekwencji**

Immagine che contiene testo, schermata, linea, diagramma

Descrizione generata automaticamente

*Rysunek 1 - Create Game*

Immagine che contiene testo, schermata, diagramma, linea

Descrizione generata automaticamente

*Rysunek 2 - Join Game*

Immagine che contiene testo, schermata, diagramma, numero

Descrizione generata automaticamente

*Rysunek 3 - Start Game*

### **c) Diagram stanu**

Immagine che contiene diagramma, testo, schizzo, disegno

Descrizione generata automaticamente

*Rysunek 4 - Stany klasy Game*

### **d) Podział pracy w projekcie**

Tomasz Jarząbek: design interfejsu, analiza wymagań, przygotowanie dokumentacji, implementacja frontendu (wersje językowe, tryb ciemny etc.)

Szymon Gryz: implementacja frontendu (Typescript), testowanie aplikacji

Norbert Koziana: implementacja backendu (Java Spring Boot)

## **Projekt architektury**

### **a) Wybór technologii**

Aplikacja korzysta z:

1. Bazy danych PostgreSql
2. Java (Spring Boot)
3. Typescript
4. Github, Github Actions, MockMvc, STOMP (Websockets), Docker

### **b) Projekt architektury aplikacji**

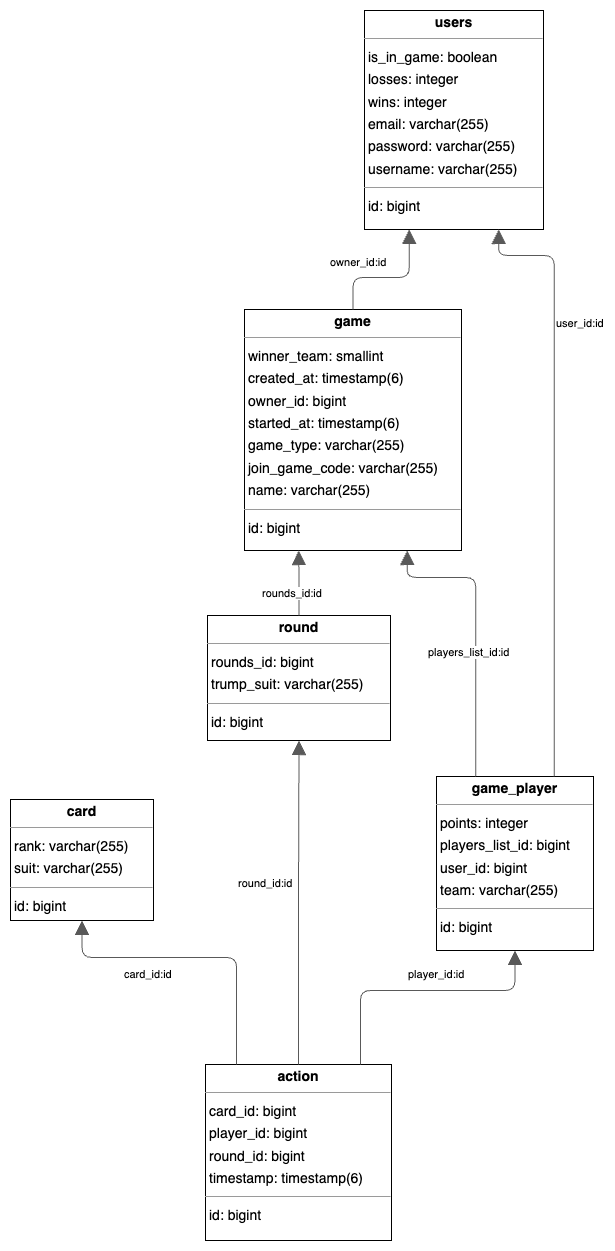
### Po wybraniu karty przez gracza, wszyscy uczestnicy rozgrywki muszą zostać natychmiast poinformowani o zmianie stanu gry. W tym celu zastosowano protokół STOMP, który umożliwia komunikację w czasie rzeczywistym. Alternatywne rozwiązanie, polegające na użyciu protokołu HTTP oraz ciągłym wysyłaniu zapytań w krótkich interwałach, byłoby nieefektywne i powodowałoby opóźnienie.

### Informacje o aktywnych grach, takie jak posiadane przez graczy karty, są przechowywane w strukturze ConcurrentHashMap, co pozwala na szybki i częsty dostęp do danych. Rozwiązanie to niesie ze sobą pewne wyzwania, m.in. konieczność zapewnienia bezpieczeństwa współbieżnego dostępu (by uniknąć race conditions) oraz ryzyko utraty danych o aktywnych grach w przypadku awarii serwera. Mimo to zapewnia lepszą wydajność w porównaniu z tradycyjnym podejściem opartym wyłącznie na bazie danych.

### Dane trwałe, takie jak informacje o użytkownikach (np. ranking) oraz historia zakończonych gier, są zapisywane w relacyjnej bazie danych PostgreSQL, co zapewnia ich trwałość oraz możliwość łatwego przetwarzania analitycznego.

### 

### **c) Projekt bazy danych**



### **d) Diagram klas**

Immagine che contiene testo, diagramma, Piano, schermata

Descrizione generata automaticamente