Sławomir Klimowski

[Adres e-mail]

Streszczenie

[Przyciągnij uwagę czytelnika interesującym streszczeniem. Zazwyczaj jest to krótkie podsumowanie dokumentu.   
Aby dodać zawartość, wystarczy kliknąć tutaj i zacząć wpisywanie.]

Opracowanie serwisu WWW udostępniającego gry planszowe

Praca inżynierska

Spis treści

[1. Wstęp 2](#_Toc118299623)

[1.1. Cel pracy 2](#_Toc118299624)

[2. Funkcjonalność 2](#_Toc118299625)

[2.1. XMLHTTPRequest 2](#_Toc118299626)

[2.2. WebSocket 2](#_Toc118299627)

[2.3. Node.js 2](#_Toc118299628)

[3. Udostępnione API 3](#_Toc118299629)

[3.1. Opis 3](#_Toc118299630)

[3.2. Implementacja na podstawie Chińczyka 7](#_Toc118299631)

[4. Opis aplikacji 7](#_Toc118299632)

[Diagram funkcjonalności 7](#_Toc118299633)

[Funkcje wewnątrz pokoju 7](#_Toc118299634)

[Technologie 7](#_Toc118299635)

[Diagram komponentów 7](#_Toc118299636)

[Diagram przepływu danych 7](#_Toc118299637)

[5. Prezentacja 7](#_Toc118299638)

[Rejestracja gry 7](#_Toc118299639)

[Otwarcie pokoju i połączenie użytkownika 7](#_Toc118299640)

[Tworzenie gier 7](#_Toc118299641)

[6. Wdrożenie 7](#_Toc118299642)

[W chmurze 7](#_Toc118299643)

[Docker + reverse-proxy 7](#_Toc118299644)

[7. Podsumowanie 7](#_Toc118299645)

[8. Wnioski 7](#_Toc118299646)

[9. Bibliografia? 7](#_Toc118299647)

# Wstęp

Gry planszowe cechują się ustandaryzowaną obsługą. Gracze przemieszczają pionki po planszy w ściśle określony przez reguły gry sposób. Bieżący stan partii jest łatwo interpretowalny, dzięki czemu gracze nie tracą czasu na zrozumienie sytuacji i mogą się skupić na analizie i opracowaniu strategii wiodącej do zwycięstwa. Dodatkowo podobieństwa w prezentacji planszy pozwalają na szybkie nauczenie się podstaw, które stosowaną są w szerokiej gamie tytułów.

Schematyczność gier planszowych doprowadziła do powstania pomysłu opracowania systemu, który oddzieli oprawę graficzną (planszę z pionkami ) od logiki (reguły gry).

## Cel pracy

Celem pracy było przygotowanie serwisu do udostępniania gier planszowych obsługującego oprawę graficzną oraz udostępniającego API do łączenia z serwisami hostującymi logikę.

W ramach pracy powstał serwis internetowy zarządzający grami oraz pokojami, w których uczestnicy mogą grać w udostępniane gry. Serwis pozwala na rejestrację dowolnej gry planszowej spełniającej API. Każdy pokój zarządza rolami uczestników i połączeniem z każdym z nich.

Dodatkowo powstał przykładowy serwis hostujący logikę gier planszowych powstały w oparciu o API.

# Funkcjonalność

## XMLHTTPRequest

XMLHTTPRequest (XHR) jest interfejsem programowania aplikacji pozwalającym na komunikację między klientem a serwer w sposób asynchroniczny. Jest dostępny jako obiekt środowiska JavaScript przeglądarek internetowych. Umożliwia wysyłanie zapytań internetowych po załadowaniu strony HTML oraz obiór odpowiedzi bez potrzeby ponownego pobrania całości treści z serwera. Pozwala to na przesyłanie nie tylko dokumentów XML, ale również danych w formacie JSON i surowego tekstu. Dzięki czemu możliwe jest dynamiczne modyfikowanie zawartości w oparciu o odpowiedzi serwera.

## WebSocket

WebSocket jest protokołem komunikacyjnym, który zapewnia dwukierunkową komunikację wewnątrz jednego połączenia TCP. Protokół WebSocket został zaprojektowany do pracy w oparciu o protokół http, dzięki czemu może korzystać z portów 80 i 443, jak również obsługiwać serwery proxy. W celu zapewnienia kompatybilności dwóch protokołów, nawiązanie połączenia WebSocket odbywa się za pośrednictwem nagłówka http Upgrade. Połączenie WebSocket pozwala na komunikację między klientem i serwerem internetowym w znacznie zwięźlejszy sposób niż http, ponieważ nagłówek wysłany jest wyłącznie w czasie zestawiania połączenia. Dodatkowo połączenie jest dwukierunkowe i pozwala na wysyłanie w ustandaryzowany sposób informacji bezpośrednio z serwera do klienta, bez potrzeby ówczesnej inicjacji przez klienta.

## Node.js

Node.js jest wieloplatformowym środowiskiem uruchomieniowym dla aplikacji w języku JavaScript do zastosowań serwerowych. Wykorzystuje silnik JavaScript Google V8. Zaprojektowany zastał do umożliwienia tworzenia łatwo skalowalnych aplikacji internetowych stosując paradygmat „JavaScript everywhere” (JavaScript wszędzie). Pozwala na ujednolicenie procesu tworzenia aplikacji wykorzystując jeden język programowania zarówno po stronie klienta jak i serwera. Architektura Node.js oparta jest wokół pętli zdarzeń gwarantującej zdalność do równoległego wykonywania zadań asynchronicznych pomimo korzystania wyłącznie z jednego wątku procesora. Posiada bogate repozytorium pakietów, które rozszerzają podstawową funkcjonalność o m.in.:

* Tworzenie aplikacji w stylu REST
* Dynamiczne generacje stron HTML w oparciu o języki szablonów np. Embedded JavaScript (EJS)
* Obsługę połączeń stałych WebSocket

# Udostępnione API

## Opis

Serwis wystawia API za pomocą którego może komunikować się z serwerami logiki gry implementujące API zgodne z definicją openapi.

API opisuje minimalną liczbę stron? REST, które musi zawierać serwis.

Strony REST (w celu umożliwienia obsługi wielu gier planszowych przez jeden serwer ścieżka zawsze musi zaczynać się od nazwy identyfikującej grę oznaczoną jako {GameName}):

* /{GameName}/api  
  Pod tym adresem znajduję się deklaracja gry dostępna metodą http get jak również post. Jest to krótki opis jakie środowisko ma zostać przygotowane, przez serwis by prawidłowo udostępnić grę.   
  Deklaracja gry:
  + Nazwa gry
  + Krótki opis
  + Adres internetowy, pod którym serwis jest dostępny
  + Numer niestandardowego portu na którym serwis nasłuchuje
  + Minimalna i maksymalna liczba graczy
  + Opis planszy(plansz) wykorzystywanej przez grę
    - Rodzaje plansz opisane w dalszej części
  + Lista ścieżek, pod którymi dostępne określone funkcje
    - API – adres, pod którym serwis udostępnia dokument opisu
    - NewGame – adres, pod którym serwis udostępnia polecenie rozpoczęcia nowej gry
    - NewRound – adres, pod którym serwis udostępnia polecenie rozpoczęcia newej rundy
    - Move – adres, pod którym serwis udostępnia polecenie wykonania ruchu
    - Update– adres, pod którym serwis udostępnia polecenie pobrania stanu gry przez użytkownika
    - Status– adres, pod którym serwis udostępnia polecenie pobrania stanu gry przez serwer
    - Open– adres, pod którym serwis udostępnia polecenie otwarcia nowego pokoju dla gry
    - Close– adres, pod którym serwis udostępnia polecenie zamknięcia otwartego pokoju gry
* /{GameName}/NewGame  
  Pod tym adresem serwis udostępnia polecenie rozpoczęcia nowej gry wewnątrz wybranego pokoju dostępne używając metody http post
  + Ciało zapytania:
    - room– nazwa identyfikująca instancję pokoju
    - players – liczba aktywnych graczy
  + Odpowiedź:
    - accepted – potwierdzenie zaakceptowania zapytania
    - message – opisowa odpowiedź na zapytanie
* /{GameName}/NewRound  
  Pod tym adresem serwis udostępnia polecenie rozpoczęcia nowej rundy wewnątrz wybranego pokoju dostępne używając metody http post
  + Ciało zapytania:
    - room – nazwa identyfikująca instancję pokoju
  + Odpowiedź:
    - accepted – potwierdzenie zaakceptowania zapytania
    - message – opisowa odpowiedź na zapytanie
* /{GameName}/Move  
  Pod tym adresem serwis udostępnia polecenie wykonania ruchu przez gracza wewnątrz wybranego pokoju dostępne używając metody http post
  + Ciało zapytania:
    - room – nazwa identyfikująca instancję pokoju
    - player – numer użytkownika wykonującego zapytanie
    - move – dane wykonanego ruchu  
      Serwis obsługuje trzy rodzaje ruchów:
      * choice – wybór jednego pola planszy  
        Przesłana wartość odpowiada nazwa wybranego pola
      * move – wybór pary dwóch pól planszy  
        Przesłana wartość odpowiada wzorcu:  
        „nazwa\_pierwszego\_pola|nazwa\_drugiego\_pola”
      * throw – rzut kością/kośćmi  
        Przesłana wartość to „throw”
    - board – numer planszy, na której dokonano ruchu
    - dices – lista ścianek rzuconych kości do gry
  + Odpowiedź:
    - accepted – potwierdzenie zaakceptowania zapytania
    - message – opisowa odpowiedź na zapytanie
* /{GameName}/Update  
  Pod tym adresem serwis udostępnia polecenie pobrania stanu gry przez użytkownika wewnątrz wybranego pokoju dostępne używając metody http post
  + Ciało zapytania:
    - room – nazwa identyfikująca instancję pokoju
    - player – numer użytkownika wykonującego zapytanie
  + Odpowiedź:
    - accepted – potwierdzenie zaakceptowania zapytania
    - message – opisowa odpowiedź na zapytanie
    - board/boards – lista stanów dla każdej planszy
    - score – punktacja wewnątrz pokoju
    - state – stan pokoju
      * gameActive – stan opisujący czy w pokoju rozgrywana jest partia gry
      * playerWon – wartość sygnalizująca, że bieżąca runda zakończyła się zwycięstwem gracza o podanym identyfikatorze
      * draw – flaga sygnalizująca, że bieżąca runda zakończyła się remisem
    - nextMove – rodzaj następnego ruchu, który gracz ma wykonać  
      Możliwe wartości: {„choice”, „move”, „throw”}
    - nextPlayer – identyfikator gracza, który ma wykonać następny ruch
* /{GameName}/Status  
  Pod tym adresem serwis udostępnia polecenie pobrania stanu gry przez serwis wewnątrz wybranego pokoju dostępne używając metody http post
  + Ciało zapytania:
    - room – nazwa identyfikująca instancję pokoju
  + Odpowiedź:
    - accepted – potwierdzenie zaakceptowania zapytania
    - message – opisowa odpowiedź na zapytanie
    - score – punktacja wewnątrz pokoju
    - state – stan pokoju
      * gameActive – stan opisujący czy w pokoju rozgrywana jest partia gry
      * playerWon – wartość sygnalizująca, że bieżąca runda zakończyła się zwycięstwem gracza o podanym identyfikatorze
      * draw – flaga sygnalizująca, że bieżąca runda zakończyła się remisem
* /{GameName}/Open  
  Pod tym adresem serwis udostępnia polecenie otwarcia nowej instancji pokoju dostępne używając metody http post
  + Ciało zapytania:
    - room – nazwa identyfikująca instancję pokoju
  + Odpowiedź:
    - accepted – potwierdzenie zaakceptowania zapytania
    - message – opisowa odpowiedź na zapytanie
* /{GameName}/Close  
  Pod tym adresem serwis udostępnia polecenie zamknięcie wybranej instacji pokoju dostępne używając metody http post
  + Ciało zapytania:
    - room – nazwa identyfikująca instancję pokoju
  + Odpowiedź:
    - accepted – potwierdzenie zaakceptowania zapytania
    - message – opisowa odpowiedź na zapytanie

Dostępne rodzaje plansz:

* Prosta  
  Plansza przedstawiająca standardową prostokątną planszę  
  Deklaracja:
  + Type=”simple” – identyfikator rodzaju planszy
  + rowCount – liczba wierszy w zakresie <1, 9>
  + rowLabels – typ znaczników dla wierszy
    - ‘d’ – cyfry
    - ‘l’ – litery
  + columnCount – liczba kolumn w zakresie <1, 9>
  + columnLabels – typ znaczników dla kolumn
    - ‘d’ – cyfry
    - ‘l’ – litery
  + tileWidth – szerokość pola planszy
  + tileHeight – wysokość pola planszy
  + textures – lista tekstur wykorzystywana w czasie gry
* Własna  
  Plansza o dowolnym kształcie i liczbie pól  
  Deklaracja:
  + Type=”custom” - identyfikator rodzaju planszy
  + tileWidth – szerokość pola planszy
  + tileHeight – wysokość pola planszy
  + width – szerokość obszaru planszy
  + height – wysokość obszaru planszy
  + background – rodzaj tła
    - color – tło wypełnione jednolitym kolorem
    - image – adres URI grafiki, która ma pełnić rolę wypełnienia
  + tiles – lista pól planszy  
    Każde pola opisuje trójka wartości:
    - x – pozycja horyzontalna lewego górnego narożnika pola
    - y – pozycja wertykalna lewego górnego narożnika pola
    - name – nazwa identyfikująca pole (nie musi być unikalna)
  + textures – lista tekstur wykorzystywana w czasie gry
* Losowa   
  Plansza przedstawiająca obszar rzutu kością(kośćmi) do gry  
  Deklaracja:
  + Type=”random” – identyfikator rodzaju planszy
  + Real – flaga, czy kolejność ścianek ma odpowiadać fizycznej kostce sześciościennej
  + numberOfSides – liczba ścianek kostki
  + numberOfDices – liczba kostek rzucanych na raz
  + rotations – ile razy kostka ma się obrócić po rzucie

## Implementacja

Przykładowa implementacja serwera udostępniającego grę planszową na przykładzie gry Chińczyk.

Pierwszą część implementacji stanowi deklaracja gry zgodna z API:

* Podstawowe informacje:
  + name: 'ManDontGetAngry',
  + hostname: 'bgs-argen-game-server.herokuapp.com',
  + port: 443,
  + description: 'Cztero-osobowa gra planszowa w chińczyka',
  + maxNoPlayers: 4,
  + minNoPlayers: 2,
* Opis wykorzystywanych planszy:
  + Plansza własna, która przedstawia obszar gry:
    - type: 'custom',
    - tileWidth: 64,
    - tileHeight: 64,
    - width: 736,
    - height: 736,
    - background: { image: ‘https://i.pinimg.com/736x/0e/10/b5/0e10b5dee4f4d73f7facac1fac79a9c9.jpg’ } – tło będące grafiką pobraną z zewnętrznego serwera
    - textures: [
      * '/static/images/textures/Tile1.png',
      * '/static/images/textures/Tile1p1.png',
      * '/static/images/textures/Tile1p2.png',
      * '/static/images/textures/Tile1p3.png',
      * '/static/images/textures/Tile1p4.png',
      * '/static/images/textures/Tile1s.png',
      * '/static/images/textures/Tile1sp1.png',
      * '/static/images/textures/Tile1sp2.png',
      * '/static/images/textures/Tile1sp3.png',
      * '/static/images/textures/Tile1sp4.png',
      * '/static/images/textures/TileB.png',
      * '/static/images/textures/TileBp.png',
      * '/static/images/textures/TileG.png',
      * '/static/images/textures/TileGp.png',
      * '/static/images/textures/TileR.png',
      * '/static/images/textures/TileRp.png',
      * '/static/images/textures/TileY.png',
      * '/static/images/textures/TileYp.png',
    - ] – lista grafik wykorzystywana przez grę dostępnych na serwisie głównym
    - tiles: [
      * { x: 16, y: 272, name: 's0' },
      * { x: 80, y: 272, name: 'm1' },
      * { x: 144, y: 272, name: 's2' },
      * { x: 16, y: 656, name: 'b4' },
      * { x: 80, y: 656, name: 'b4' },
      * { x: 80, y: 336, name: 'f1' },
      * { x: 144, y: 336, name: 'f1' },
    - ] – siedem wybranych pól planszy z powtórzeniami nazwy
  + Plansza losowo do obsługi kostki sześciościennej
    - type: 'random',
    - real: true,
    - numberOfSides: 6,
    - numberOfDices: 1,
    - rotations: 10,
* Lista ścieżek pod którymi dostępne są odpowiednie funkcje
  + api: {
    - API: '/ManDontGetAngry/api',
    - NewGame: '/ManDontGetAngry/NewGame',
    - NewRound: '/ManDontGetAngry/NewRound',
    - Move: '/ManDontGetAngry/Move',
    - Update: '/ManDontGetAngry/Update',
    - Status: '/ManDontGetAngry/Status',
    - Open: '/ManDontGetAngry/Open',
    - Close: '/ManDontGetAngry/Close',
  + },

Kolejnym elementem jest opracowanie serwera https w stylu REST, który udostępni wymagane funkcjonalności. W tym celu można wykorzystać pakiet Express.js.

Propozycja implementacji ścieżki dla funkcji API

app.route('/:GameName/api')

.get((req, res) => {

if (!description[req.params.GameName]) {

res.json({ accepted: false, message: 'Gra nie istnieje!' });

return;

}

res.json({ accepted: true, api: description[req.params.GameName] });

})

.post((req, res) => {

if (!description[req.params.GameName]) {

res.json({ accepted: false, message: 'Gra nie istnieje!' });

return;

}

res.json({ accepted: true, api: description[req.params.GameName] });

});

Wszystkie elementy związane z logiką gry są w pełni uniezależnione od serwisu głównego. Dzięki czemu nie ma narzuconej kolejności graczy, ani kolejności wykonywanych ruchów. W głównym serwisie nie ma również walidacji informacji przesyłanych przez użytkowników z wartościami oczekiwanymi.

Logikę gry można zaprojektować jako klasę lub pakiet, którego odpowiednie funkcje wywoływane są w zależności od ścieżki.

Przykład obsłużenia funkcji Update oraz odpowiadającej jej części logiki gry:

app.post('/:GameName/Update', (req, res) => {

const gameName = req.params.GameName;

const game = games[gameName];

if (!game) res.json({ accepted: false, message: "Game doesn't exists!" });

const rid = req.body.room;

const roomName = '#' + rid;

const room = rooms.get(roomName);

if (room) {

const player = req.body.player;

const data = **game.getUpdate(room, player);**

if (Object.keys(data).length) {

res.json({

...{ accepted: true, message: 'Request successful' },

...data,

});

} else res.json({ accepted: false, message: 'Request unsuccessful' });

} else res.json({ accepted: false, message: "Room doesn't exists!" });

});

function getUpdate(room, playerId) {

const board = room.board.concat(room.bases).concat(room.finishes);

saveZones.concat(startZones).forEach((zoneId) => {

board[zoneId] += 5;

});

for (var pid = 0; pid < 4; ++pid) {

for (var zid = 0; zid < 4; ++zid) {

board[4 \* pid + zid + 40 + 16] =

10 + 2 \* pid + (board[4 \* pid + zid + 40 + 16] > 0 ? 1 : 0);

}

}

return {

board: board,

score: room.score,

state: room.state,

nextMove: room.nextMove,

nextPlayer: room.player,

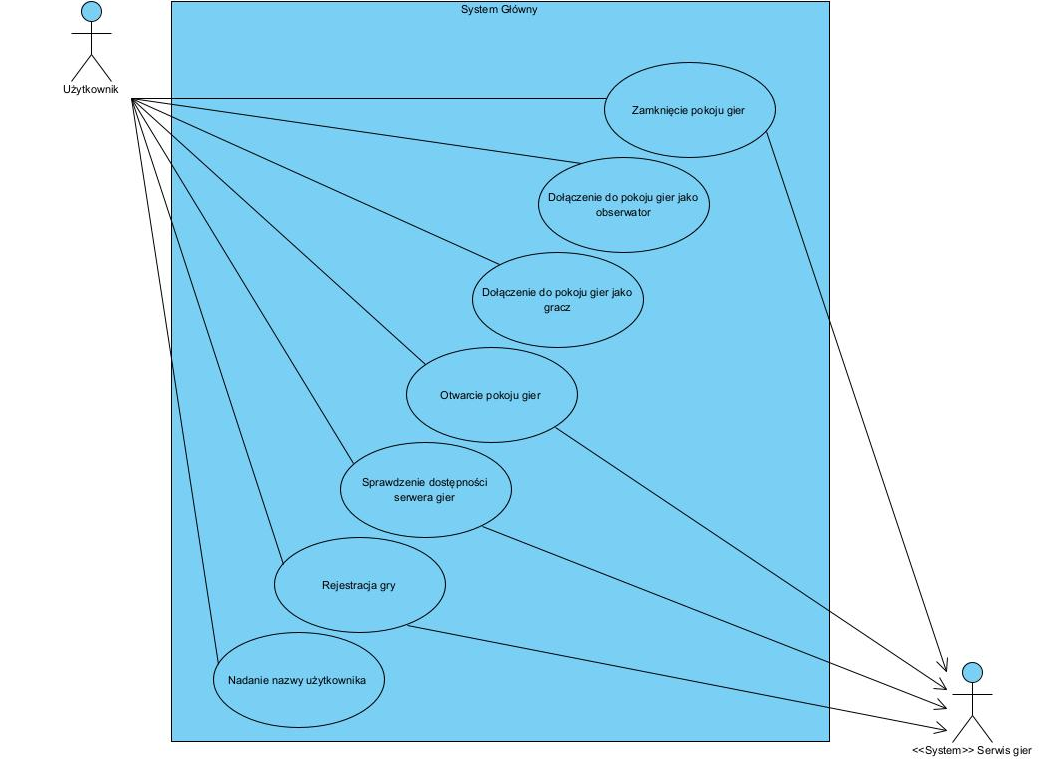
};

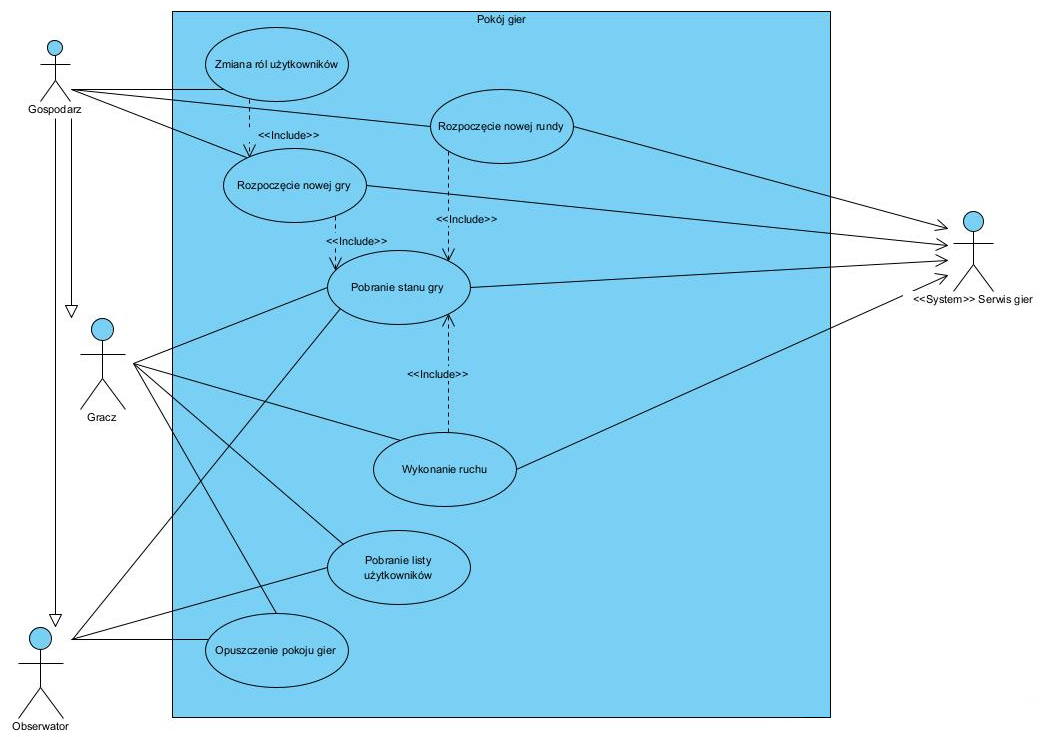
}

Opuścić

# Opis aplikacji

## Diagram przypadków użycia

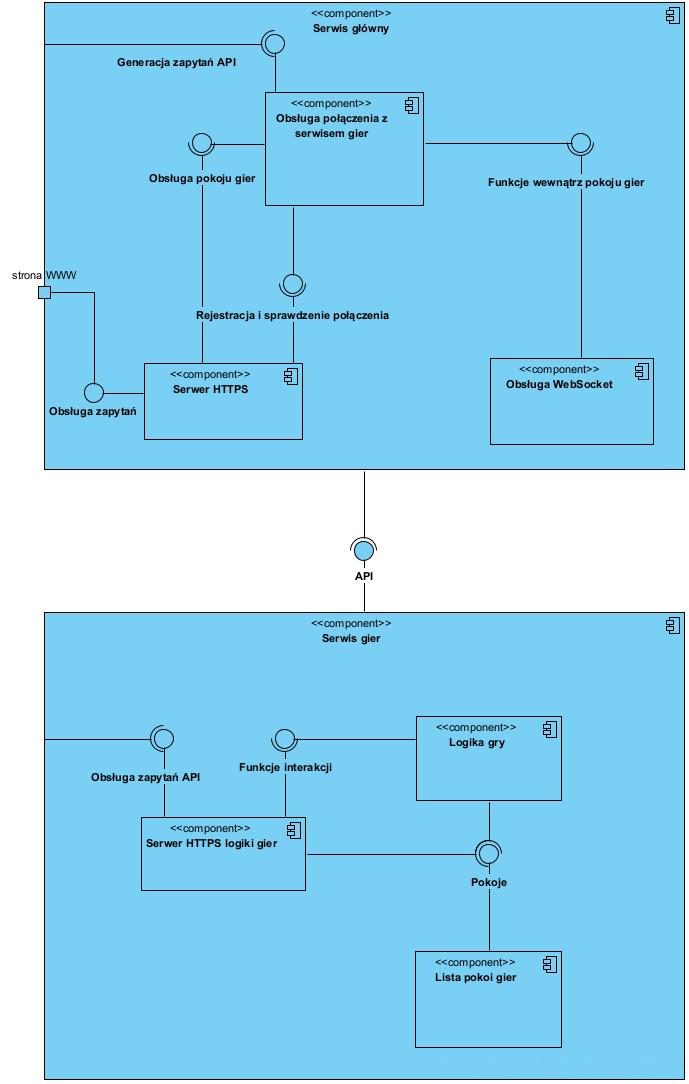




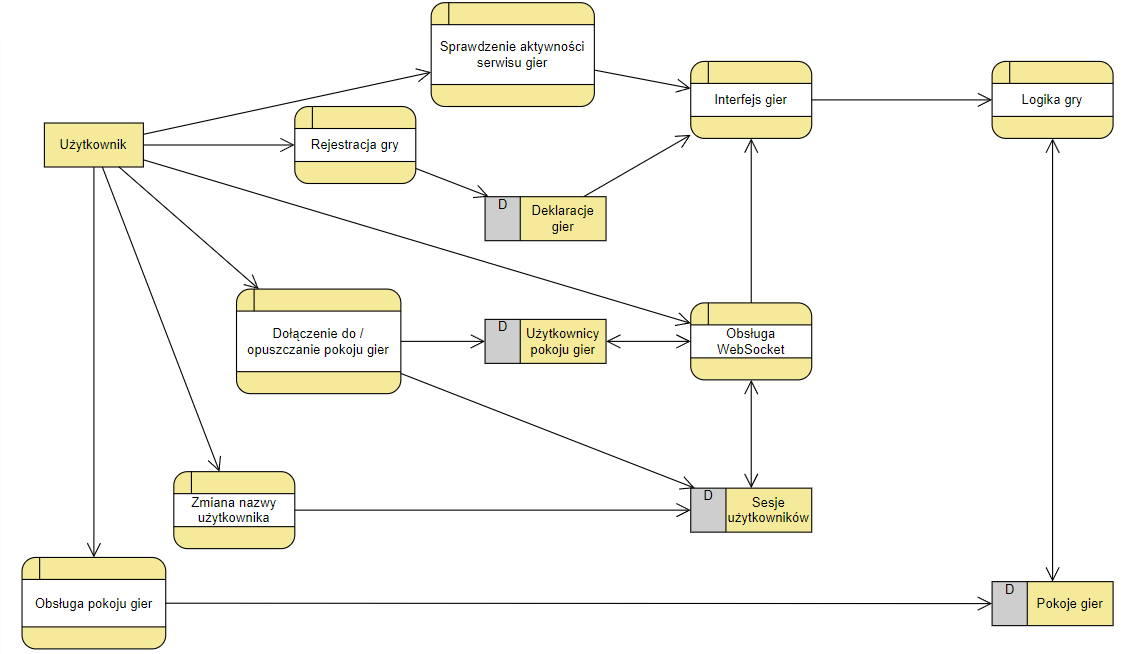
## Funkcje wewnątrz pokoju

## Technologie

## Diagram komponentów

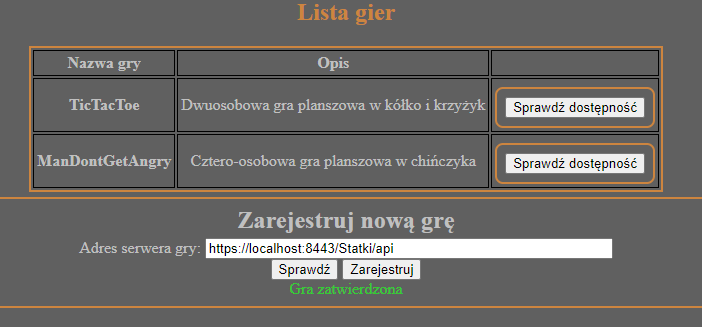


## Diagram przepływu danych

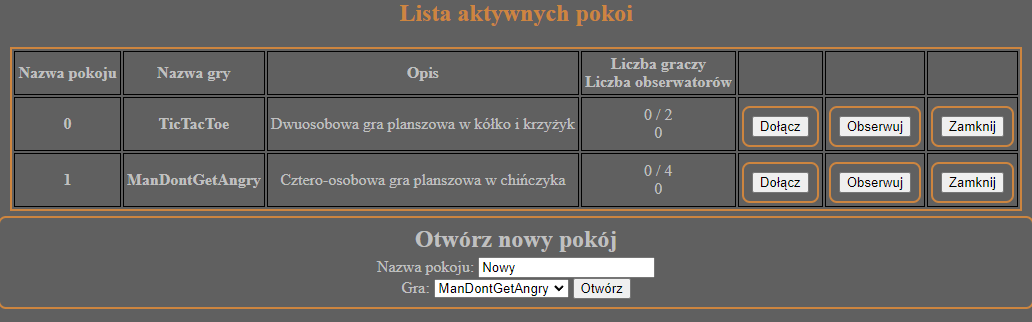


# Prezentacja

## Rejestracja gry



## Otwarcie pokoju i połączenie użytkownika



## Tworzenie gier

# Wdrożenie

## W chmurze

## Docker + reverse-proxy

# Podsumowanie

# Wnioski

# Bibliografia?