

**Wydział Fizyki i Informatyki Stosowanej**

KATEDRA Układów Wbudowanych

Projekt dyplomowy

*Opracowanie serwisu WWW udostępniającego gry planszowe*

*Development of a website providing board games*

Autor: *Sławomir Krzysztof Klimowski*

Kierunek studiów: Informatyka Stosowana

Opiekun pracy: *dr inż. Antoni Dydejczyk*

Kraków, 2022

Spis treści

[1. Wstęp 4](#_Toc123237415)

[1.1. Cel pracy 5](#_Toc123237416)

[2. Rozwiązania technologiczne 6](#_Toc123237417)

[2.1. XMLHTTPRequest (XHR) 6](#_Toc123237418)

[2.2. WebSocket 6](#_Toc123237419)

[2.3. Funkcjonalność udostępniona przez serwis główny 7](#_Toc123237420)

[2.3.1. Node.js 7](#_Toc123237421)

[2.3.2. Funkcjonalność ogólna 8](#_Toc123237422)

[2.3.3. Funkcjonalność wewnątrz pokoju 10](#_Toc123237423)

[3. Udostępnione API 12](#_Toc123237424)

[3.1. Opis 12](#_Toc123237425)

[3.1.1. Funkcje interfejsu 12](#_Toc123237426)

[3.1.2. Dostępne rodzaje plansz 17](#_Toc123237427)

[3.2. Przykładowa implementacja 18](#_Toc123237428)

[3.2.1. Deklaracja gry 18](#_Toc123237429)

[3.2.2. Serwer HTTPS 19](#_Toc123237430)

[3.3. Testy interfejsu 21](#_Toc123237431)

[3.3.1. Testy serwera logiki 21](#_Toc123237432)

[3.3.2. Testy serwera głównego 21](#_Toc123237433)

[4. Opis aplikacji 22](#_Toc123237434)

[4.1. Architektura rozwiązania 22](#_Toc123237435)

[4.2. Model przepływu informacji 25](#_Toc123237436)

[5. Prezentacja 27](#_Toc123237437)

[5.1. Rejestracja gry 28](#_Toc123237438)

[5.2. Otwarcie pokoju i połączenie użytkownika 29](#_Toc123237439)

[5.3. Gra planszowa w Chińczyka 30](#_Toc123237440)

[5.4. Panel Gospodarza 31](#_Toc123237441)

[5.5. Statki 32](#_Toc123237442)

[6. Wdrożenie 36](#_Toc123237443)

[6.1. Rozwiązanie w chmurze 36](#_Toc123237444)

[6.2. Rozwiązanie w kontenerach 39](#_Toc123237445)

[7. Podsumowanie oraz wnioski 40](#_Toc123237446)

[8. Bibliografia 41](#_Toc123237447)

# Wstęp

Gry planszowe są jedną z najstarszych form rozrywki znaną człowiekowi, a historia gier takich jak Szachy, Go czy Królewska gra z Ur liczy sobie tysiące lat. Pomimo swojego wieku ta forma rozrywki cieszy się dużą popularnością nie tylko wśród młodzieży, ale również wśród osób dorosłych. Na rynku pojawiają się nowe tytuły, wprowadzające nową fabułę, nie zmieniając opracowanego przez tysiące lat sposobu prezentacji stanu rozgrywki. Znacząca część gier planszowych cechuje się ustandaryzowaną obsługą i formą prezentacji. 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 nowym graczom na szybkie zaznajomienie się z regułami nowego tytułu, korzystając z doświadczenia nabytego w innych tytułach. Porównując ze sobą kilka znanych tytułów tj. Szachy, Warcaby, Kółko i krzyżyk oraz Chińczyk jak również Królewską grę z Ur zauważyć można podobieństwa w aspekcie prezentacji. Najlepiej obrazuje to fakt, że plansza do gry w Szachy jest identyczna jak do gry w Warcaby,   
a jedyną widoczną różnicą jest kształt pionków. Większe różnice pojawiają się regułach gier, gdzie każdy tytuł niesie ze sobą nieraz całkiem odmienny zestaw zasad opisujących przebieg rozgrywki.

## Cel pracy

Duża popularność gier planszowych sprawia, że są one również jednym z najczęściej implementowanych rodzajów gier. Są one dobrze opisywalne za pomocą modelu maszyny stanowej, której zaprogramowanie nie stanowi dużej trudności nawet dla początkującego programisty. Ich oczywista schematyczność rodzi pytanie czy można ustandaryzować i ułatwić proces wytwórczy gier planszowych.

Celem niniejszej pracy jest opracowanie serwisu do udostępniania gier planszowych, który pozwoli na podział implementacji gry na dwie główne części, logikę i oprawę graficzną.

Logika przedstawia zbiór reguł, które opisują przebieg rozgrywki. Stanowi ona najbardziej zróżnicowaną pomiędzy różnymi tytułami część implementacji. Jest to również najbardziej formalna część, którą można opisać maszyną stanów. Nie narzuca ona żadnych znaczących obostrzeń na narzędzia programistyczne.

Druga część zawiera wszystkie aspekty związane z prezentacją danych użytkownikowi. Użytkownikiem jest osoba, która bierze czynny lub bierny udział w rozgrywce. Użytkownik   
o czynnym udziale nazywany jest graczem, natomiast użytkownik bierny przestawia obserwatora, który wyłącznie ogląda rozgrywaną partię. W skład oprawy graficznej wchodzi prezentacja planszy i pionków, obsługa kości do gry oraz komunikacja z użytkownikiem. Przedstawiana zawartość jest silnie uzależniona od logiki gry, gdyż reguły gry opisują kształt planszy i pionków, liczbę graczy jak również jaką część planszy dany użytkownik może widzieć. Jest to część stawiająca największy rygor na narzędzia programistyczne. Wymagana jest prosta i zrozumiała prezentacja graficzna planszy pozwalająca na intuicyjny i komfortowy sposób wykonywania ruchu.

W ramach pracy przygotowany został serwis, w dalszej części pracy nazywany serwisem lub serwerem głównym, implementujący oprawę graficzną wraz z ustandaryzowanym interfejsem do udostępniania gier planszowych. Serwis pozwala na rejestrację dowolnej gry planszowej spełniającej wymogi interfejsu opisane w rozdziale 3.1. Do zadań serwera należy zarządzanie listą gier jak również listą pokoi, zarządzaniem połączeniem z użytkownikami oraz ich rolą wewnątrz pokoju. Każdy pokój jest niezależną od innych instancją wybranej gry planszowej.

Dodatkowo powstał przykładowy serwis udostępniający logikę czterech gier planszowych, nazywany serwisem logiki w dalszej części pracy. Serwer ten implementuje interfejs spełniający wymogi serwisu głównego, dla czterech gier, które kolejno wykorzystują różne dostępne funkcjonalności.

Pierwszą zaimplementowaną grą jest Kółko i krzyżyk, która stanowi przykład minimalnej funkcjonalności wymaganej przez interfejs serwisu głównego. Następnym przykładem jest czteroosobowa gra połącz cztery w linii, która ilustruje możliwość zmiennej liczby graczy. Kolejno na podstawie gry w statki pokazana została możliwość obsługi wielu plansz oraz prezentacja różnej zawartości tych plansz w zależności od roli użytkownika. Dodatkowo jest to pierwsza gra wprowadzająca wielokrotny ruch jednego gracza. Ostatnią opracowaną grą jest Chińczyk, która jako jedyna wykorzystuje planszę o nieregularnych kształtach oraz wprowadza losowość w postaci rzutu kością do gry.

# Rozwiązania technologiczne

Serwis do udostępniania gier planszowych jest przykładem rozwiązania problemu komunikacji pomiędzy serwerem aplikacji, a serwerami zewnętrznymi w sposób transparenty dla użytkownika. Rozwiązanie zapewnia ustandaryzowany sposób wymiany informacji, który nie naraża klientów na potencjalne zagrożenia ze strony aplikacji zewnętrznych. Serwis główny łączy ze sobą dwa rozwiązania technologiczne XHR oraz WebSocket, udostępniające komunikację asynchroniczną wewnątrz protokołu HTTP.

## XMLHTTPRequest (XHR)

XMLHTTPRequest jest to interfejs programowania aplikacji (API) pozwalający na asynchroniczną komunikację między klientem a serwer. Większość współczesnych przeglądarek internetowych udostępnia go jako obiekt środowiska JavaScript. Interfejs ten implementuje obsługę wysyłania zapytań internetowych nawet po załadowaniu strony HTML. Pozwala również na odbieranie odpowiedzi w sposób asynchroniczny za pośrednictwem pętli zdarzeń języka JavaScript. XHR został zaprojektowany do przesyłania dokumentów XML,   
w późniejszych wersjach dodano obsługę danych w formacie JSON, jak również wiadomości   
w formie surowego tekstu. Korzystając z tego interfejsu możliwe jest dynamiczne modyfikowanie zawartości strony HTML. Popularne w rozwiązaniach SPA.

Powyższe narzędzie znalazło zastosowanie w implementacji funkcjonalności odpowiedzialnej za nawiązywanie komunikacji pomiędzy serwerami. Proces ten jest zadaniem o bardzo niskiej częstotliwości występowania, natomiast może być czasochłonny w przypadku, gdy serwer zewnętrzny nie odpowiada. Czas oczekiwania na zakończenie tworzenia połączenia wynosi wtedy równowartość limitowi czasu oczekiwania na odpowiedź na żądanie HTTP. Wykonanie tego polecenia w sposób synchroniczny doprowadziłoby do zablokowania interfejsu użytkownikowi lub zerwania połączenia przez przeglądarkę z błędem 503 (Serwis niedostępy). Zastosowania narzędzia do komunikacji asynchronicznej gwarantuje użytkownikowi komfortowe korzystanie z serwisu.

## WebSocket

WebSocket jest to protokół komunikacyjny, który zapewnia dwukierunkową komunikację wewnątrz jednego połączenia TCP. Protokół WebSocket został zaprojektowany do współpracy z protokołem HTTP. Takie rozwiązanie gwarantuje pełną obsługę funkcjonalności oferowanej przez protokół HTTP, m. in. Serwery proxy. Dodatkowo połączenie WebSocket nie wymaga osobnego portu, tylko wykorzystuje ten sam, z którego korzysta usługa HTTP, standardowo 80 dla połączenia zwykłego oraz 443 w przypadku połączeń szyfrowanych. 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 zapytania przez klienta.

Cała funkcjonalność wewnątrz pokoju gier w serwisie głównym opiera się na połączeniu WebSocket nawiązanym automatycznie po dołączeniu. Dzięki takiemu rozwiązaniu każdy użytkownik otrzymuje powiadomienia o zmianach wywołanych przez innego członka. Pozwala to na znaczące ograniczenie ruchu w sieci, ponieważ nie ma potrzeby wysyłania periodycznych zapytań o stan pokoju, jak w przypadku rozwiązania opartego o technologię XHR.

## Funkcjonalność udostępniona przez serwis główny

Serwis główny, w celu sprostania narzuconym na niego wymaganiom wynikającym   
z założeń projektu, doje dostęp użytkownikowi do funkcjonalności niezbędnych do korzystania z gier dostępnych w serwisie logiki. Serwis udostępnia również funkcje, których zadaniem jest zwiększenie komfortu w czasie używania aplikacji. Interakcja z użytkownikiem obsłużona jest przez środowisko Node.js rozszerzone o dodatkowe pakiety.

### Node.js

Node.js jest to wieloplatformowe środowisko uruchomieniowe dla aplikacji napisanych   
w języku JavaScript do zastosowań serwerowych. Aplikacje uruchamiane są korzystając   
z silnika JavaScript Google V8. Node.js zaprojektowany zastał do umożliwienia tworzenia łatwo skalowalnych aplikacji internetowych stosując paradygmat „JavaScript everywhere” (JavaScript wszędzie). Pozwala na ujednolicenie procesu tworzenia aplikacji wykorzystując wyłącznie jeden język programowania zarówno po stronie klienta jak i serwera. Architektura Node.js oparta jest wokół pętli zdarzeń gwarantującej zdolność do równoległego wykonywania zadań asynchronicznych pomimo korzystania wyłącznie z jednego wątku procesora. Posiada bogate repozytorium pakietów, które rozszerzają podstawową funkcjonalność.

Ważniejsze pakiety użyte w aplikacji:

* Express.js

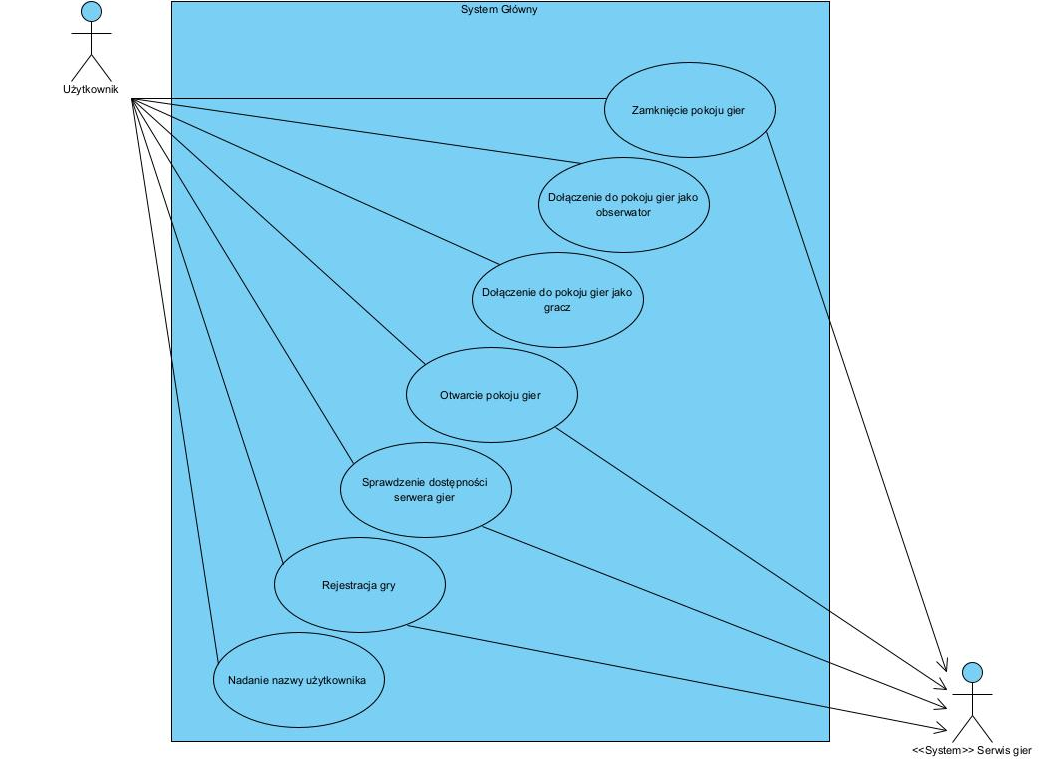
Odpowiedzialny za utworzenie aplikacji w stylu REST, pozwala na szybką implementację ścieżek. Do pakietu dołączona została również obsługa sesji i walidacja dokumentem OpenAPI oraz dynamiczna generacja stron HTML wykorzystując języki szablonów oparty na języku JavaScript .

* WebSocket

Pakiet implementuje obsługę protokołu WebSocket wykorzystywanego przez połączenia wewnątrz pokoi gier.

### Funkcjonalność ogólna

Pierwsza grupa udostępnianych funkcjonalności odpowiada za przygotowanie środowiska dla pokoi gier. Rysunek 1 przedstawia wszystkie funkcje dostępne dla użytkownika poza instancją pokoju gier. Rysunek sygnalizuje również, które funkcje wymagają aktywnego połącznie z serwisem logiki, a które wykonują wyłącznie operacje wewnętrzne.



Rysunek Funkcjonalność ogólna

* Sprawdzenie dostępności serwisu logiki wybranej gry

Fundamentalna funkcja, którą serwis musi udostępniać. Wywołuje ona próbę nawiązania lub sprawdzenie stanu połączenia między serwisami. Pozwala to stwierdzić czy serwis logiki jest osiągalny od strony serwisu głównego oraz czy wcześniej oferowana przez niego gra nadal jest dostępna. Tę funkcję użytkownik może wykonać na dwa sposoby. Po pierwsze może wysłać zapytanie na dowolny adres internetowy   
w procesie rejestracji gry, jak również na adres zapisany w pamięci serwisu dla zarejestrowanej gry. Żądania HTTP wysyłane są asynchronicznie za pomocą narzędzia XHR, co umożliwia sprawdzenie wielu gier naraz. Wszystkie pozostałe funkcjonalności wykonywane są synchronicznie.

* Rejestracja gry planszowej

Druga podstawowa funkcja, która odpowiada za rejestrację dowolnej gry planszowej spełniającej wymogi interfejsu. Wysłane zostaje przez serwis główny żądanie HTTP   
o deklarację interfejsu gry na adres wskazany przez użytkownika. Jeżeli walidacja odpowiedzi zakończy się sukcesem otrzymany dokument zostanie zapisywany   
w pamięci. Dalsza komunikacja między serwisami opiera się o informacje zawarte wewnątrz deklaracji.

* Otwarcie i zamknięcie pokoju gier

Każda dopuszczona do rejestracji gra musi posiadać możliwość obsługi co najmniej jednej instancji pokoju gier. Serwis główny udostępnia użytkownikowi funkcje Open służącą do zainicjowania próby otwarcia nowego pokoju oraz Close, która wysyła prośbę o zamknięcie wybranego pokoju. Ich dokładne działanie jest uzależnione od implementacji serwisu logiki. Serwis główny narzuca wyłącznie ograniczenie od dołu na dokładną liczbę obsługiwanych pokoi na grę. Nie jest wymaga również walidacja nieaktywności pokoju w czasie jego zamknięcia, wszyscy aktywni użytkownicy zostaną automatycznie rozłączeni.

* Dołączenie użytkownika do pokoju gier z preferowaną rolą

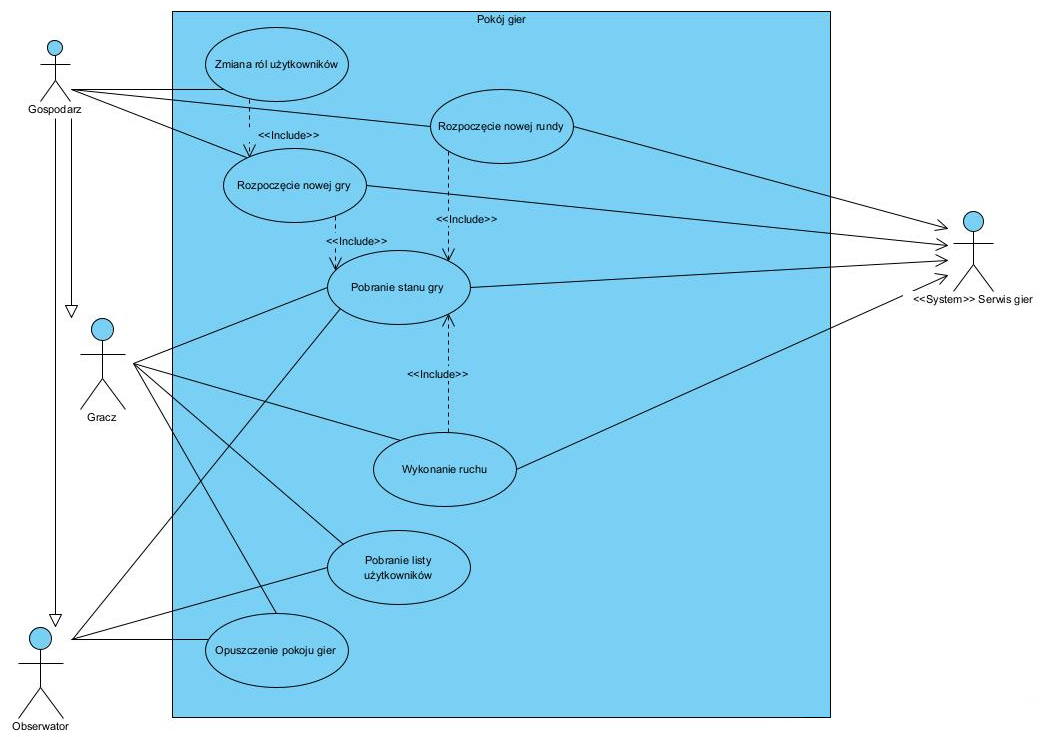
Każdy użytkownik ma umożliwione dołączenie do dowolnego ówcześnie otwartego pokoju. Dodatkowo może sprecyzować jaką rolę wewnątrz danego pokoju chciałby objąć. Dołączenie jako gracz jest możliwe wyłącznie, gdy nie przekroczona została jeszcze maksymalna liczba graczy dla wybranego pokoju. Jeżeli dojdzie do sytuacji,   
że dołączenie jako gracz będzie nie dostępne, wtedy użytkownikowi automatycznie zostanie przydzielona rola obserwatora.

* Nadanie nazwy użytkownika

Ostatnią ogólną funkcjonalnością udostępnianą przez serwis główny jest możliwość wybrania własnej nazwy użytkownika. Nazwa ta jest wykorzystywana jako identyfikator na liście członków pokoju gier. Nadawanie unikalnej nazwy nie jest wymagane, służy ona wyłącznie użytkownikom.

### Funkcjonalność wewnątrz pokoju

Osobną grupą funkcjonalności są funkcje wewnątrz pokoju. Część funkcji dostępne są wyłącznie dla wybranej roli użytkownika. Każdy użytkownik pokoju gier ma przydzieloną jedną z dwóch ról, gracz lub obserwator. Gracz to użytkownik, który bierze czynny udział   
w rozgrywce. Oznacza to, że ma prawo do wykonywania ruchów w czasie trwania partii. Obserwatorem nazywa się użytkownika, który bierze wyłącznie bierny udział w rozgrywce i nie może wykonywać żadnego ruchu. Dodatkowo jednemu użytkownikowi przedzielana jest specjalna dodatkowa rola gospodarza pokoju. Rola ta rozszerza możliwości członka pokoju   
o zmianę przydzielonych ról z wyjątkiem roli gospodarza, rozpoczęcie nowej partii oraz nowej rundy w czasie partii.



Rysunek Funkcjonalności wewnątrz pokoju

Na rysunku 2 przedstawione zostały funkcje z podziałem na role członków. Poniżej znajdują szczegółowe opisy każdej funkcjonalności.

* Funkcje dostępne wyłącznie dla gospodarza:
  + Zmiana ról użytkowników

Odpowiada za udostępnienie możliwości zmiany przydzielonych użytkownikom ról, pozwalając przeprowadzić rotację graczy wewnątrz pokoju. Jest alternatywną metodą rozpoczęcia nowej gry, gdzie w ciele zapytania wysłana zostaje nowa lista graczy. Dzięki temu gracz nie musi opuścić pokoju, żeby zwolnić swoje miejsce dla innego użytkownika.

* + Rozpoczęcie nowej partii lub rundy

Funkcje udostępniają gospodarzowi możliwość rozpoczęcia nowej partii oraz nowej rundy w czasie partii.

Partia również określana jako gra jest to zbiór jednej lub więcej rund gry planszowej wewnątrz pokoju. Każda partia ma własną punktację, która przechowuję rezultaty wszystkich rund, które odbyły w czasie jej trwania.

Runda jest to pojedyncza rozgrywka gry planszowej, która kończy się zwycięstwem jednego z graczy lub remisem.

* Funkcja dostępna wyłącznie dla gracza
  + Wykonanie ruchu

Funkcja pozwalająca na wysłanie propozycji ruchu. Każdy gracz w swojej turze zostanie poproszony o wykonanie ruchu lub serii ruchów według oczekiwań serwisu logiki. W przypadku, gdy ruch będzie nie prawidłowy, gracz zostanie o tym poinformowany i poproszony o wykonanie innego.

* Funkcje dostępne dla wszystkich ról
  + Pobranie stanu gry

Podstawowa funkcja wywołująca uaktualnienie lokalnego stanu oraz z odświeżeniem interfejsu użytkownika. Uruchamiana automatycznie przy każdej zmianie stanu pokoju. Może również zostać wykonana na żądanie.

* + Pobranie listy użytkowników

Funkcja wywołująca uaktualnienie lokalnej listy członków pokoju z podziałem na role. Uruchamiana automatycznie podczas dołączenia lub opuszczenia pokoju przez dowolnego użytkownika.

* + Opuszczenie pokoju gier

Automatyczna funkcja wywołująca zwolnienie ówcześnie zajmowanych ról przez użytkownika.

# Udostępnione API

Interfejs programistyczny aplikacji (ang. Application Programming Interface, API) jest to zbiór zasad opisujący sposób komunikacji pomiędzy wieloma programami komputerowymi. Umożliwia ustandaryzowanie procesu wymiany informacji zapewniając hermetyzację poszczególnych aplikacji. Dobrze zaprojektowane API pozwala na korzystanie w pełni   
z udostępnianej funkcjonalności, nie narażając poszczególne podsystemy na szkodliwe interakcje pomiędzy nimi. Elementy systemu traktują się jak czarne skrzynki.

Kluczowym elementem API jest specyfikacja będąca dokumentem opisującym sposób korzystania z interfejsu oraz jakie funkcjonalności aplikacja musi implementować, by móc wystawić określony interfejs.

Serwis do udostępniania gier planszowych jest systemem składającym się z dwóch serwisów internetowych, które komunikują się za pomocą interfejsu internetowego. Serwer logiki gier jest fragmentem implementującym API, a serwer główny jest aplikacją korzystającą z wystawionego interfejsu.

## Opis

Specyfikacja interfejsu internetowego została opracowana zgodnie z projektem OpenAPI (dawniej Swagger). Specyfikacja OpenAPI opisuje format plików, które pozwalają na projektowanie serwisów internetowych w stylu RESTful. Bogata gama narzędzi obsługujących specyfikację pozwala na automatyzację procesu wdrażania oraz walidacji interfejsu.

Dokument API opisuje wszystkie funkcje niezbędne do prawidłowej komunikacji między serwisami. Składa się z listy ścieżek REST-owych, pod którymi znajdują się odpowiednie funkcjonalności implementowane przez logikę gier. Opracowany dokument zakłada ściśle określony format zasobów internetowych. Wynika to z limitacji specyfikacji OpenAPI, która wymaga podania wszystkich punktów końcowych serwisu, natomiast interfejs pozwala na dowolny format ścieżek. Proponowana forma zasobów umożliwia obsługę wielu gier planszowych przez jeden serwis logiki. Każdy punkt końcowy zaczyna się od nazwy gry, do której się odnosi. Takie rozwiązanie pozwala na pominięcie jednego parametru zapytania HTTP, odpowiedzialnego na identyfikację gry.

### Funkcje interfejsu

W ramach interfejsu zostało opisane osiem funkcji: API, NewGame, NewRound, Move, Update, Status, Open, Close. Dla każdej z nich zdefiniowano przyjmowane parametry oraz format odpowiedzi. Wszystkie funkcjonalności dostępne są metodą HTTP POST. Wyjątek stanowi polecenie API, które jest obsłużone również metodą GET, co pozwala użytkownikom na łatwe podejrzenie zawartości deklaracji. Rysunek 3 przedstawia listę wszystkich funkcji wyświetloną w programie Swagger Editor, będącym oprogramowaniem do edycji specyfikacji OpenAPI.



Rysunek Funkcje interfejsu

Poniższa lista zawiera dokładny opis każdej funkcjonalności interfejsu.

* API

Bezparametrowa funkcja, w której odpowiedzi znajduje się deklaracja gry, zawierająca opis środowiska niezbędnego do prawidłowego obsłużenia gry przez serwis główny. Poprawna deklaracja składa się z następujących pól:

* + Nazwa gry
  + Krótki opis
  + Adres internetowy, pod którym serwis jest dostępny
  + Numer niestandardowego portu protokołu TCP, na którym serwis nasłuchuje
  + Minimalna i maksymalna liczba graczy
  + Opis planszy(plansz) wykorzystywanej przez grę
    - Rodzaje plansz opisane w Rozdziale 3.1.2.
  + Lista ścieżek, pod którymi dostępne są określone funkcje

Informacje zawarte w deklaracji pozwalają na udostępnienie jej przez serwisy pośredniczące lub zapisanie w pamięci serwisu głównego, bez utraty możliwości kontynuacji komunikacji. Przykładowa deklaracja widoczna jest na rysunku 4, wraz ze wszystkimi pozostałymi informacja oferowanymi w ramach edytora specyfikacji.

* + Odpowiedź:
    - Flaga określająca stan zaakceptowania zapytania
    - Deklaracja gry

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek Opis funkcji API w edytorze

* NewGame

Funkcja udostępnia polecenie rozpoczęcia nowej gry wewnątrz wybranego pokoju. Odpowiada za przywrócenie stanu początkowego plansz oraz wyzerowanie punktacji.

* + Ciało zapytania:
    - Nazwa identyfikująca instancję pokoju
    - Liczba aktywnych graczy
  + Odpowiedź:
    - Flaga określająca stan zaakceptowania zapytania
    - Opisowa odpowiedź na zapytanie
* NewRound

Pod tym adresem znajduje się polecenie rozpoczęcia nowej rundy wewnątrz wybranego pokoju. Akcja resetuje wyłącznie plansze.

* + Ciało zapytania:
    - Nazwa identyfikująca instancję pokoju
  + Odpowiedź:
    - Flaga określająca stan zaakceptowania zapytania
    - Opisowa odpowiedź na zapytanie
* Move

Funkcja odpowiadająca za przekazanie propozycji ruchu przez gracza wewnątrz wybranego pokoju. Na parametry narzucona jest wyłącznie walidacja formy, sprawdzenia poprawności ruchu należy do części implementującej logikę gry.

* + Ciało zapytania:
    - Nazwa identyfikująca instancję pokoju
    - Numer użytkownika wykonującego zapytanie
    - Dane proponowanego ruchu

Interfejs obsługuje trzy rodzaje ruchów:

* + - * choice – wybór jednego pola planszy

Przesłana wartość odpowiada nazwie wybranego pola.

* + - * move – wybór pary dwóch pól planszy

Pierwsze pole określa, z którego miejsca zaczyna się ruch. Natomiast drugie jest miejscem docelowym. Nazwy pól są złączone ze sobą znakiem pionowej kreski („|”).

* + - * throw – rzut kością/kośćmi

Ten rodzaj ruchu przyjmuje wyłącznie wartość „throw”, która wyraża chęć wykonania rzutu kośćmi przez gracza.

* + - Numer planszy, na której dokonano ruchu
    - Lista ścianek rzuconych kości do gry (obowiązuje w przypadku ruchu typu throw)
  + Odpowiedź:
    - Flaga określająca stan zaakceptowania zapytania
    - Opisowa odpowiedź na zapytanie
* Update

Polecenie pobrania stanu bieżącej gry przez członka wybranego pokoju. W odpowiedzi znajdują się wszystkie niezbędne informacje do odświeżenia interfejsu użytkownika.

* + Ciało zapytania:
    - Nazwa identyfikująca instancję pokoju
    - Numer użytkownika wykonującego zapytanie
  + Odpowiedź:
    - Flaga określająca stan zaakceptowania zapytania
    - Opisowa odpowiedź na zapytanie
    - Lista stanów pól dla każdej planszy

Wartość stanu jest tożsama z indeksem wyświetlanej w miejscu pola planszy tekstury z listy podanej w deklaracji gry.

* + - Punktacja wewnątrz pokoju dla bieżącej partii
    - Stan pokoju
      * Flaga opisująca czy w pokoju rozgrywana jest runda gry
      * Wartość sygnalizująca, że bieżąca runda zakończyła się zwycięstwem gracza o podanym identyfikatorze
      * Flaga sygnalizująca, że bieżąca runda zakończyła się remisem
    - Rodzaj następnego ruchu oczekiwanego od gracza

Interfejs pozwala wyłącznie na wartości: „choice”, „move”, „throw”.

* + - Identyfikator gracza, który ma wykonać następny ruch
* Status

Funkcja wywoływana wyłącznie na potrzeby serwisu głównego, której odpowiedź służy do określenia aktywności wybranej gry w serwisie logiki.

* + Odpowiedź:
    - Flaga określająca stan zaakceptowania zapytania
    - Opisowa odpowiedź na zapytanie

Możliwe jest przekazanie identyfikatora pokoju, wtedy, podobnie jak polecenie Update, wykonywana jest procedura pobrania stanu gry. W odpowiedzi znajduje się minimalna liczba informacji pozwalającą na określenie bezczynności pokoju.

* + Ciało zapytania:
    - Nazwa identyfikująca instancję pokoju
  + Odpowiedź:
    - Flaga określająca stan zaakceptowania zapytania
    - Opisowa odpowiedź na zapytanie
    - Punktacja wewnątrz pokoju
    - Stan pokoju
      * Flaga opisująca czy w pokoju rozgrywana jest partia gry
      * Wartość sygnalizująca, że bieżąca runda zakończyła się zwycięstwem gracza o podanym identyfikatorze
      * Flaga sygnalizująca, że bieżąca runda zakończyła się remisem
* Open

Funkcja udostępnia polecenie otwarcia nowej instancji pokoju gier. Domyślnie, to polecenie nie powoduje rozpoczęcia nowej partii.

* + Ciało zapytania:
    - Nazwa identyfikująca instancję pokoju
  + Odpowiedź:
    - Flaga określająca stan zaakceptowania zapytania
    - Opisowa odpowiedź na zapytanie
* Close

Analogicznie do funkcji Open wywołanie tej procedury powoduje zamknięcie instancji pokoju gier.

* + Ciało zapytania:
    - Nazwa identyfikująca instancję pokoju
  + Odpowiedź:
    - Flaga określająca stan zaakceptowania zapytania
    - Opisowa odpowiedź na zapytanie

### Dostępne rodzaje plansz

Wewnątrz specyfikacji interfejsu zdefiniowane zostały trzy interaktywne dla użytkownika obszary nazywane planszami. Każdy z nich przedstawia wybraną część wizualną gry planszowej. Rodzaj obszaru identyfikowany jest przez wartość pola type w deklaracji.

Obsługiwane typy plansz:

* „simple”

Obszar przedstawiający standardową prostokątną planszę. Składa się z informacji o liczbie pól w pionie i poziomie, rozmiarze w pikselach pojedynczego pola oraz listy tekstur wykorzystywanych do reprezentacji stanu. Pozycja na planszy jest reprezentowana przez parę znaków, gdzie pierwszy znak oznacza numer kolumny, a drugi wiersza. Wymiar może być indeksowany cyframi lub literami, dodatkowo rodzaje znaków są niezależne od siebie.

* „custom”

Reprezentacja planszy o dowolnym kształcie i liczbie pól. Deklarację rozpoczyna się od podania wymiarów obszaru oraz pojedynczego pola wraz z listą tekstur do wizualizacji stanów. Kolejno precyzuje się rodzaj wypełnienia tła. Do wyboru jest jednolity kolor w formacie zgodnym z kolorami CSS lub grafika, która może być dowolnym adresem URI osiągalnym przez użytkowników. Ostatnim elementem definicji jest lista wszystkich pól planszy, gdzie jedno pole reprezentowane jest przez trójkę wartości: współrzędne x i y lewego górnego wierzchołka pola oraz jego nazwa, którą będzie identyfikowane. Nazwy nie muszą być unikalne, pozwala to na grupowanie pól o tych samych własnościach.

Ze względu na politykę Same-origin wszystkie względne adresy grafik, są obsługiwane przez serwis główny. Dopuszcza się stosowanie adresów absolutnych do serwerów zewnętrznych.

* „random”

Ostatnim dostępnym rodzajem jest obszar reprezentujący pole rzutu kością(kośćmi) do gry. Podczas deklaracji podaje się liczbę kości oraz ile razy mają się przetoczyć po wykonanym rzucie. Kolejno określa się sposób generacji kolejnych ścianek podczas toczenia. Dostępna jest uproszczona symulacja rzeczywistej kości sześciościennej oraz arbitralnej o dowolnej liczbie oczek.

## Przykładowa implementacja

W ramach projektu został opracowany serwis logiki gier wraz z przykładową implementacją interfejsu zgodną ze specyfikacją. W efekcie powstało rozwiązanie pozwalające na obsługę m.in. popularnej gry planszowej w Chińczyka (ang. Man, Don’t Get Angry).

### Deklaracja gry

Pierwszym elementem implementacji było przygotowanie deklaracji gry przesyłanej   
w funkcji API interfejsu. Definicję można podzielić na trzy funkcjonalne fragmenty. Na początku znajdują się informacje podstawowe tj. nazwa gry w formacie akceptowalnym jako adres URL, adres internetowy lub nazwa domenowa serwera, port protokołu TCP na którym nasłuchuje serwer HTTPS, krótki opis dla użytkowników oraz liczba obsługiwanych graczy na pokój.

W drugiej części są opisane wszystkie wykorzystywane plansze na potrzeby interfejsu użytkownika. Gra korzysta z jednej planszy o nieregularnych kształtach. Płótno jest kwadratem o boku długości 736px, wypełnionym grafiką pobraną z serwisu logiki. Pozostałe tekstury pochodzą z repozytorium serwera głównego. Następnie wylistowane są wszystkie pola planszy, gdzie każde z nich jest kwadratem o boku długości 64px. Przykładowe osiem pól   
w formacie JSON:

* {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’},

Miejsca odpowiadające domkowi lub mecie dla danego gracza współdzielą nazwę, dzięki czemu gracz może wybrać dowolne z nich, aby uzyskać ten sam efekt.

Drugim interaktywnym obszarem jest przestrzeń do rzutu pojedynczą kością sześciościenną. Wykorzystywana jest symulacja rzeczywistego toczenia z dziesięcioma obrotami na rzut.

Na końcu podane są ścieżki do każdej z funkcji interfejsu zgodne ze specyfikacją OpenAPI. Dla przykładu polecenie API dostępne jest pod adresem /ManDontGetAngry/api.

### Serwer HTTPS

Kolejnym elementem implementacji było opracowanie serwera HTTPS w stylu REST, który udostępnia wymagane funkcjonalności. W tym celu wykorzystany został pakiet Express.js środowiska Node.js. Opracowany serwis logiki przystosowany został do obsługi kilku gier na raz, z tego powodu proponowane rozwiązanie jest parametryzowane nazwą gry. 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. Z tego powodu sprawdzanie wszystkich informacji spoczywa na twórcy gry.

Przykładowa implementacja ścieżki dla funkcji API znajduje się na listingu 1.

1. function processApiRequest(req, res) {
2. if (!description[req.params.GameName]) {
3. res.json({ accepted: false,
4. message: "Game doesn't exist!" });
5. return;
6. }
7. res.json({ accepted: true,
8. api: description[req.params.GameName] });
9. }
10. app.route('/:GameName/api')
11. .get(processApiRequest)
12. .post(processApiRequest);

Listing 1 Implementacja ścieżki dla funkcji API

Na listingu 1 w linii numer 1 znajduje się deklaracja funkcji przetwarzającej zapytanie o deklarację gry. W ciele funkcji sprawdzane jest czy serwer obsługuje żądaną grę. Jeżeli ją obsługuje w odpowiedzi wysłana zostanie deklaracja tej gry w formacie JSON, w przeciwnym przypadku zgłoszony zostanie błąd. W linii numer 11 znajduje się deklaracja ścieżki dla funkcji API. Następne polecenie odpowiada za zdefiniowanie zachowania serwera dla metody HTTP GET poprzez podanie funkcji przetwarzającej zapytania. W linii 13 znajduje się analogiczna implementacja metody POST.

Logikę gry można zaprojektować jako klasę lub pakiet, którego odpowiednie funkcje wywoływane są dla odpowiednich funkcjonalności interfejsu.

Przykład obsłużenia funkcji Update oraz odpowiadającej jej części logiki gry przedstawia listing 2.

1. app.post('/:GameName/Update', (req, res) => {
2. const gameName = req.params.GameName;
3. const game = games[gameName];
4. if (!game) res.json({ accepted: false,
5. message: "Game doesn't exist!”});
6. const rid = req.body.room;
7. const roomName = rid;
8. const room = rooms.get(roomName);
9. if (room) {
10. const player = req.body.player;
11. const data = **game.getUpdate(room, player);**
12. if (Object.keys(data).length) {
13. res.json({
14. accepted: true,
15. message: 'Request successful',
16. ...data,
17. });
18. } else
19. res.json({ accepted: false,
20. message: 'Request unsuccessful' });
21. } else
22. res.json({ accepted: false,
23. message: "Room doesn't exist!”});
24. });
25. function getUpdate(room, playerId) {
26. const board =
27. room.board.concat(room.bases)
28. .concat(room.finishes);
29. return {
30. board: board,
31. score: room.score,
32. state: room.state,
33. nextMove: room.nextMove,
34. nextPlayer: room.player,
35. };
36. }

Listing 2 Implementacja ścieżki dla funkcji Update

Podobnie jak dla implementacji funkcji API na listingu 2 w pierwszej linii znajduje się deklaracja ścieżki. W tym przypadku wykorzystana została skrócona forma zapisu, gdzie ścieżka oraz wyrażenie lambda podane są jako dwa parametry funkcji obsługującej żądania metodą POST. Następnie w liniach 2-9 znajdują się polecenia pobrania wartości podstawowych argumentów zapytania tj. nazwa gry oraz instancji pokoju wraz ze sprawdzeniem ich poprawności. W skład bloku warunku w liniach 10-20 wchodzi wywołanie odpowiedniej metody wraz z pobraniem dodatkowych argumentów z ciała zapytania oraz wysłanie odpowiednich odpowiedzi w zależności od wyniku funkcji getUpdate wyróżnionej w linii 11. Jej definicja zamieszczona została w liniach 25-36. W ciele metody odbywa się przygotowanie planszy na podstawie danych przechowywanych w lokalnej instancji pokoju. Na koniec zwracany jest obiekt, zawierający wartości zgodne ze specyfikacją interfejsu dla funkcjonalności Update. Analogicznie obsłużone zostały pozostałe funkcje interfejsu.

## Testy interfejsu

Opracowany interfejs jest kluczowym elementem aplikacji, ponieważ stanowi jedyne pasmo komunikacyjne pomiędzy serwisem głównym, a serwerem logiki. Ze względu na budowę interfejsu przypadki testowe zostały podzielone na dwie grupy. Pierwszą część stanowią testy poprawności odpowiedzi na żądania HTTP, które przeprowadzane są po stronie serwisu logiki. Druga grupa testów posłużyła do sprawdzenia zdolności obsłużenia wszystkich możliwych odpowiedzi funkcji interfejsu. Testy tej części odbyły się po stronie serwera głównego.

### Testy serwera logiki

Serwer logiki implementuje opracowany interfejs. Do jego zadań należy obsłużenie zapytań do każdej funkcji oferowanej przez interfejs. W aplikacji zastosowana została walidacja OpenAPI, co znacząco ograniczyło liczbę przypadków testowych. Dokument OpenAPI gwarantuje poprawność składniową, jak również oferuje częściową walidację semantyczną odpowiedzi. Każda funkcjonalność serwer została przetestowana pod kątem odporności na nieprawidłowe żądania. Sprawdzone zostały odpowiedzi na zapytania do nieistniejącej gry, żądania z nie pełną treścią oraz prawidłowe zapytania. W tym celu przygotowana została specjalna strona domowa, która przeprowadziła testy za pośrednictwem narzędzia XHR.

### Testy serwera głównego

Serwer główny reprezentuje obiekt korzystający z opracowanego interfejsu. Do jego zadań należy obsłużenie wszystkich możliwych typów odpowiedzi, przy założeniu, że obiekt implementujący odpowiada zgodnie ze specyfikacją interfejsu. Po zastosowaniu walidacji OpenAPI serwis główny może otrzymać jeden z czterech rodzajów odpowiedzi:

* Zaakceptowana

Odpowiedź w formacie JSON posiadająca pole accepted o wartości true. Otrzymanie takiej wiadomości sygnalizuje, że zapytanie było poprawne oraz zostało pomyślnie przetworzone

* Niezaakceptowana

Obiekt JSON, którego wartość pola accepted jest równoznaczna z logicznym fałszem. Jest to sygnał na prawidłowe żądanie, którego przetworzenie zakończyło się niepowodzeniem np. zapytanie o deklarację gry, której serwis nie obsługuje.

* Nieprawidłowa

Odpowiedź w innym formacie niż JSON lub w nieposiadająca pola accepted. Sytuacja mogąca wyłącznie wystąpić, gdy walidacja OpenAPI nie została zastosowana. Oznacza to, że serwis główny nie może zaufać otrzymanej treści i odpowiedź należy odrzucić.

* Odrzucona

Wiadomość otrzymana, gdy nie powiodło się dostarczenie zapytania HTTPS na wskazany adres.

Na potrzeby przeprowadzenia testów do serwisu głównego dodana została podstrona /tests, której otwarcie wywoływało podstawowe testy, których celem było sprawdzenie reakcji serwera na każdy rodzaj odpowiedzi.

# Opis aplikacji

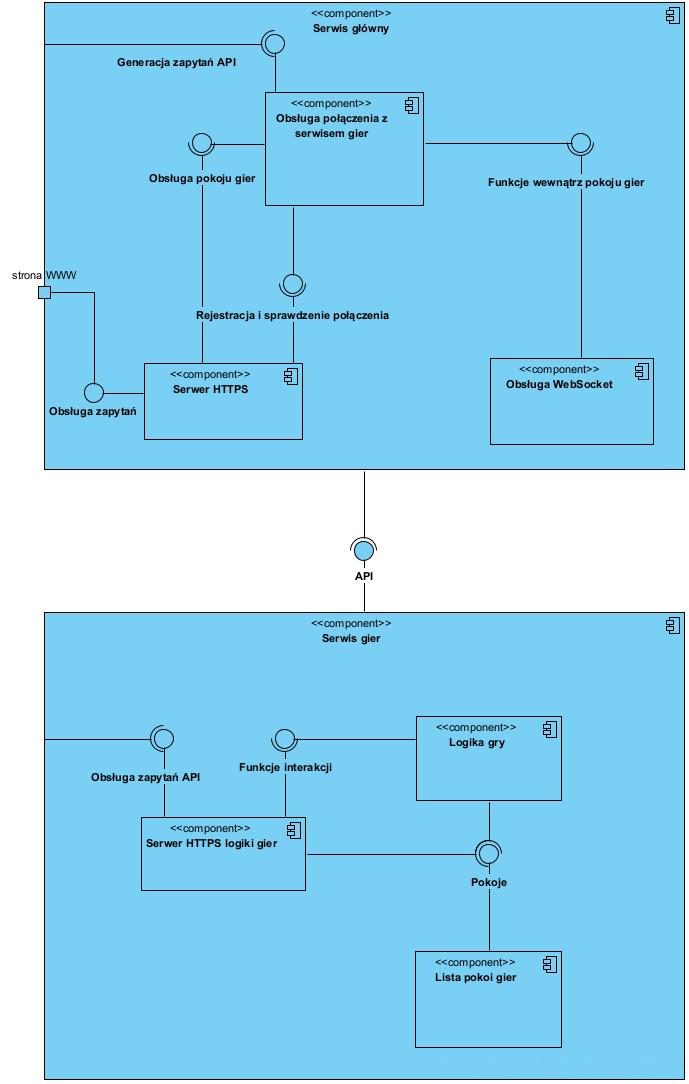
Aplikacja do udostępniania gier planszowych składa się z dwóch współpracujących ze sobą serwisów internetowych. Porozumiewają się ze sobą za pomocą asynchronicznych zapytań poprzez protokół HTTP. Wewnętrzne struktury są uniezależnione od siebie, pozwalając na całkowitą dowolność w implementacji rozwiązania.

Pierwszym elementem aplikacji jest serwis główny, który odpowiada za komunikację z użytkownikami oraz przygotowanie środowiska na potrzeby pokoi gier. Do kluczowych zadań serwera należy również umożliwienie korzystania z funkcjonalności udostępnianych przez serwis logiki.

Serwis logiki stanowi zewnętrzną część aplikacji. Jest fragmentem, którego implementacja nie musi znajdować się w tym samym środowisku co serwis główny. Wprowadzone rozwiązanie technologiczne pozwala na dynamiczne dołączanie serwera logiki w trakcie działania aplikacji. Do zadań tej części systemu należy udostępnianie funkcjonalności opisanej w specyfikacji interfejsu.

## Architektura rozwiązania

W architekturze aplikacji widać naturalny podział na dwa komponenty. Dodatkowo strukturę każdego elementu można podzielić na mniejsze funkcjonalne fragmenty. Zostało to zilustrowane przez diagram komponentów na rysunku 5.



Rysunek Diagram komponentów

Każdy element widoczny na diagramie pełni fundamentalną rolą. Spektrum zadań przynależnych do odpowiednich fragmentów zostało opisane poniżej.

Lista komponentów i ich zadań

* Serwis główny
  + Obsługa połączenia z serwisem gier

Element odpowiedzialny za komunikację z serwisem logiki. Zajmuje się przygotowywaniem zapytań oraz interpretację odpowiedzi zgodnie ze zdefiniowanym interfejsem. Nadzoruje listę udostępnianych gier, jak również wszystkie instancje pokoi.

* + Serwer HTTPS

Komponent reprezentując serwer HTTPS, który udostępnia interfejs użytkownika. Zapewnia dostęp do podstawowej funkcjonalności serwisu tj. dodawanie gier oraz obsługa pokoi.

* + Obsługa WebSocket

W tym fragmencie znajduje się serwer WebSocket do obsługi połączeń stałych z użytkownikami. Głównym jego zadaniem jest udostępnianie funkcjonalności wewnątrz pokoju gier tj. aktualizacja lokalnego stanu gry, wykonywanie ruchów przez graczy oraz rozpoczynanie nowych partii przez gospodarza.

* Serwis logiki
  + Serwer HTTPS

Komponent ten jest implementacją interfejsu wykorzystywanego przez serwis główny. W ramach tego serwera zawarta została walidacja składni zapytań jak również odpowiedzi w oparciu o specyfikację OpenAPI.

* + Logika gry

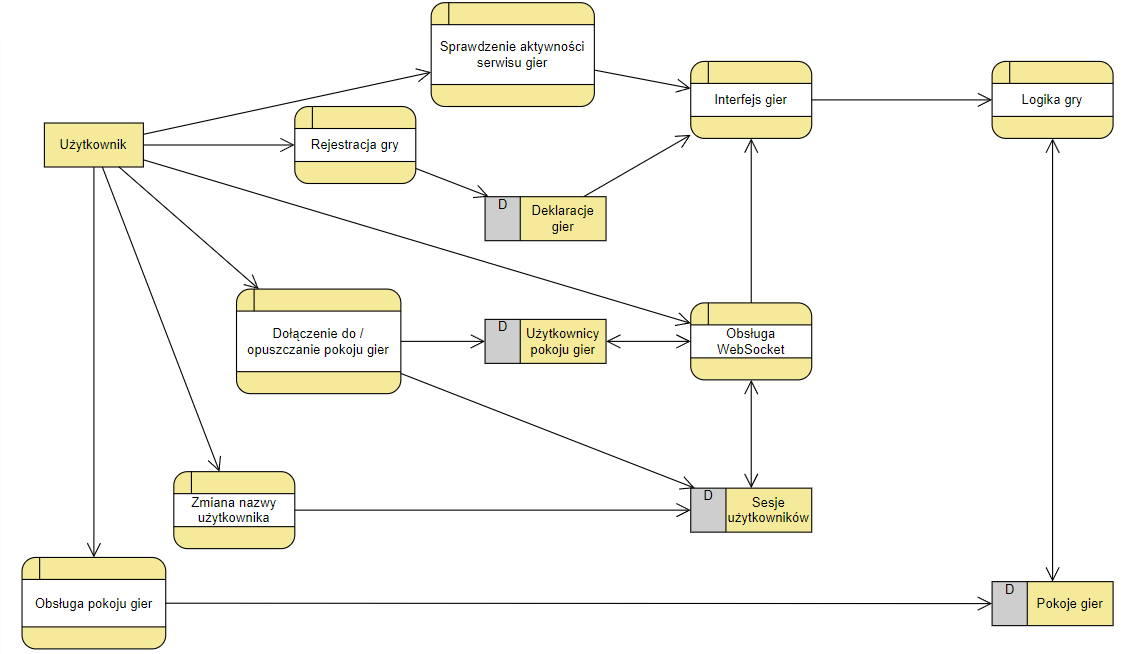
Element reprezentujący maszynę stanów udostępnianej gry planszowej. Odpowiada za sprawdzanie poprawności przychodzących do serwera informacji na poziomie semantyki. W tym miejscu znajduje się definicja reguł opisujących przebieg rozgrywki.

* + Lista pokoi gier

Kontroler magazynu instancji pokoi gier, którego zadaniem jest umożliwienie korzystania z informacji przechowywanych na potrzeby każdej rozgrywki. Monitoruje liczbę pokoi oraz zatwierdza próby otwarcia nowych instancji.

## Model przepływu informacji

Aplikacja posiada nieskomplikowany model przepływu danych z wyłącznie jednym aktorem. Schemat systemu przedstawiony został na rysunku 6.



Rysunek Diagram przepływu informacji

Diagram przepływu danych przedstawia diagram zerowy sytemu od udostępniania gier planszowych, gdzie serwis logiki zostaw włączony do serwisu głównego. Taka prezentacja pozwala na uproszczenie schematu przepływu informacji między funkcjami w całej aplikacji, zwiększając jego czytelność. Poniższa lista zawiera informacje o zadaniach każdego z procesów oraz opis informacji składowanych w magazynach.

* Procesy:
  + Sprawdzanie aktywności serwisu gier

Element odpowiada za przygotowanie zapytania i analizę odpowiedzi serwera logiki, w celu określenia jego dostępności przez serwis główny.

* + Rejestracja gier

Proces przeprowadza rejestrację nowej gry planszowej oraz sprawdza poprawność syntaktyczną deklaracji.

* + Dołączanie do / opuszczanie pokoju gier

Element obsługujący żądanie dołączenia do instancji pokoju gier, poprzez nadanie odpowiedniej roli użytkownikowi, a następnie zapisanie jej w sesji. Wykonuje również automatyczne czyszczenie wartości rekordu sesji po opuszczeniu pokoju.

* + Zmiana nazwy użytkownika

Proces, w którym następuje walidacja semantyczna proponowanej nazwy użytkownika oraz zapisanie jej w magazynie sesji.

* + Obsługa pokoju gier

Element odpowiedzialny za monitorowanie wszystkich instancji pokoju oraz stanu połączeń z ich członkami, zapewnia możliwość dodania nowych, usunięcia istniejących, jak również udostępnia funkcjonalność gospodarzowi.

* + Obsługa WebSocket

Proces interpretacji komunikatów wymienianych między użytkownikiem, a serwerem głównym. Dostępny dla każdego członka pokoju gier.

* + Interfejs gier

Element przetwarza stan rozgrywki do postaci interpretowalnej przez pozostałe części aplikacji. Powoduje wywołanie odpowiedniej funkcji logiki gry w zależności od zapytania

* + Logika gry

Reprezentacja maszyny stanów gry, pozwala na odczyt bieżącego stanu oraz wywołanie przejścia do kolejnego

* Magazyny danych:
  + Deklaracje gier

Opisy gier planszowych zgodne ze specyfikacją API.

* + Użytkownicy pokoju gier

Uchwyty do połączeń stałych wszystkich członków danej instancji pokoju, grupowane według roli użytkownika.

* + Sesje użytkowników

Informacje chwilowe o kliencie aplikacji tj. lista odwiedzonych pokoi wraz z spełnionymi rolami, nazwa, którą użytkownik się identyfikuje.

* + Pokoje gier

Dane o rozgrywce gry przechowywane na potrzeby logiki gry dla danej instancji pokoju. Są to informacje bez określonej struktury. Magazyn ten pełni rolę przestrzeni zmiennych lokalnych dla rozgrywki gry.

Wszystkie dane w aplikacji przechowywane są w sposób nietrwały. Każdy magazyn jest bazą klucz wartość znajdującą się w pamięci operacyjnej.

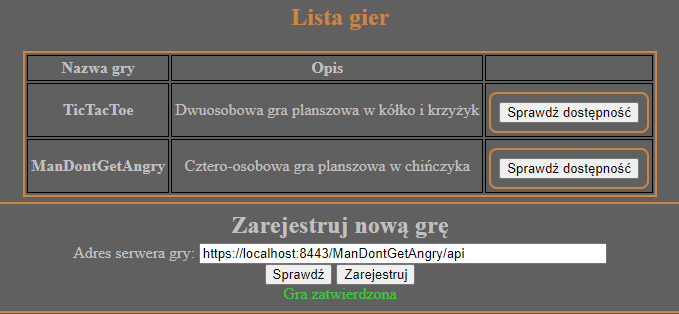
Na diagramie przepływu danych, w przeciwieństwie do diagramu komponentów, nie widać granicy między serwisami. Brak granicy wynika z faktu, że wiele procesów ma swoje odzwierciedlenie zarówno w serwisie głównym jak również po stronie serwera logiki. Ilustruje to element Interfejs gier, gdzie jeden serwis odpowiada na zapytania drugiego korzystając z zaimplementowanego API. W celu lepszego zobrazowania podziału systemu, należałoby przejść do bardziej szczegółowego poziomu diagramu, na którym część procesów zostałaby powielona i prowadziła przez interfejs internetowy.

# Prezentacja

Zaprojektowana aplikacja cechuje się minimalistycznym interfejsem użytkownika, który daje dostęp do wszystkich oferowanych funkcjonalności. Funkcje zostały pogrupowane na oddzielnych stronach według zakresu ich działalności.

## Rejestracja gry

Fragment interfejsu użytkownika widoczny na rysunku 7 udostępnia wszystkie funkcjonalności związane z procesem nawiązywania połączenia z serwisem logiki oraz rejestracji nowej gry planszowej na serwerze głównym.



Rysunek Rejestracja gry

Przycisk Sprawdź w bloku rejestracji pozwala zweryfikować osiągalność podanego adresu przez serwer główny oraz wstępnie walidować poprawność syntaktyczną odpowiedzi. Przycisk Zarejestruj zapisuje deklaracje gry w bazie danych serwisu. Wszystkie uprzednio zapisane gry zostają wylistowane na stronie. Za pomocą guzika Sprawdź dostępność użytkownik może monitorować dostępność gry oferowanej przez serwis logiki.

## Otwarcie pokoju i połączenie użytkownika

Wszystkie funkcje związane z obsługą instancji pokoi gier zostały zawarte na jednej stronie przedstawionej na rysunku 8. Składa się z listy wszystkich utworzonych pokoi oraz bloku do otwierania nowych.

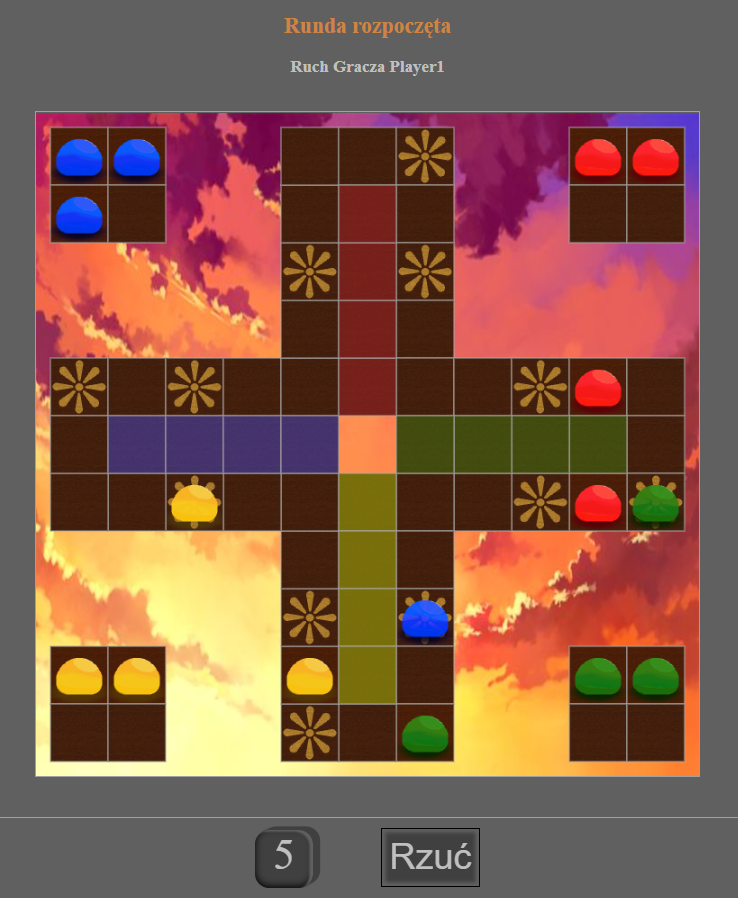


Rysunek Otwarcie nowego pokoju gier

W celu utworzenia nowej instancji pokoju należy podać unikalną nazwę, którą będzie on identyfikowany oraz wybrać z listy, dla której gry jest tworzony. Po wciśnięci przycisku otwórz, na serwer główny zostanie wysłana prośba otwarcia nowego pokoju. Proces ten wymaga zatwierdzenia operacji przez serwis logiki, z tego powodu zaleca się zweryfikować dostępność gry. Każdy użytkownik może dołączyć do dowolnego pokoju jako gracz lub obserwator korzystając odpowiednio z przycisków Dołącz oraz Obserwuj. W dowolnej chwili użytkownik ma prawo zażądać zamknięcia instancji pokoju, służy do tego guzik Zamknij.

## Gra planszowa w Chińczyka

Na podstawie popularnej gry planszowej w Chińczyka zademonstrowany został interfejs udostępniany członkom pokoju. Wygląd planszy budowany jest w oparciu o deklarację gry, a stan rozgrywki jest uzupełniany indywidulanie w zależności od pełnionej przez użytkownika roli.

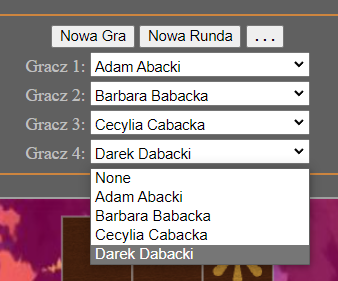


Rysunek Plansza do gry w Chińczyka

Rysunek 9 przedstawia widok z perspektywy gracza numer 2. Każdy członek pokoju jest informowany o stanie rozgrywki oraz kto wykonuje następny ruch. Dodatkowo gracz, do którego przypisana jest bieżąca tura, dostaje informację jakiego rodzaju ruchu oczekuje serwis logiki. Przycisk Rzuć wykonuje rzut kością natomiast ruch pionka odbywa się poprzez kliknięcie odpowiednich pól planszy.

## Panel Gospodarza

Wybrany użytkownik otrzymuje rolę gospodarza, która umożliwia korzystanie   
z dodatkowych funkcjonalności wewnątrz pokoju. Wygląd panelu gospodarza ilustruje rysunek 10.

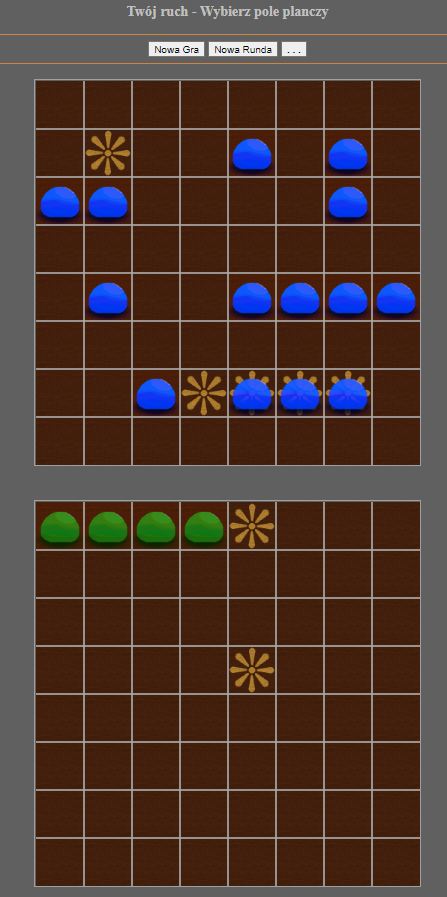


Rysunek Panel gospodarza

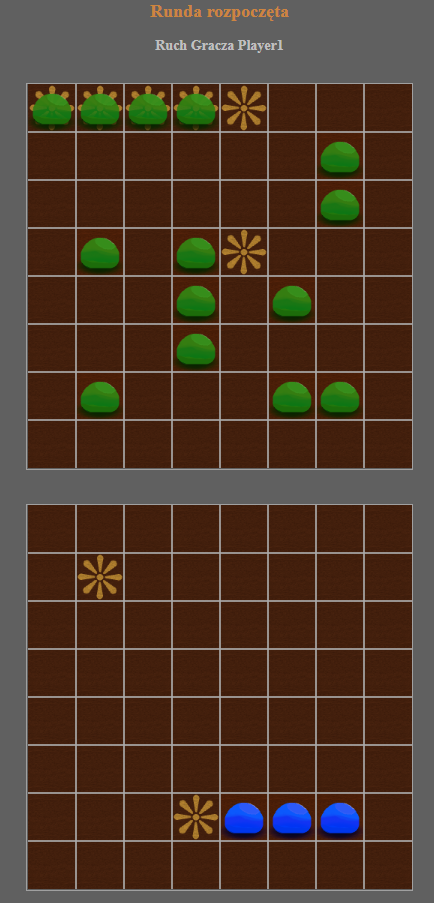
Gospodarz ma możliwość rozpoczęcia partii gry oraz rundy za pomocą odpowiednio przycisku Nowa Gra i Nowa Runda. Dodatkowo przed rozpoczęciem kolejnej partii użytkownik może wybrać członków, którzy będą brać udział w rozgrywce jako gracze. Serwer główny automatycznie przypisuje nowe role pozwalając na płynne przełączanie członków pokoju.

## Statki

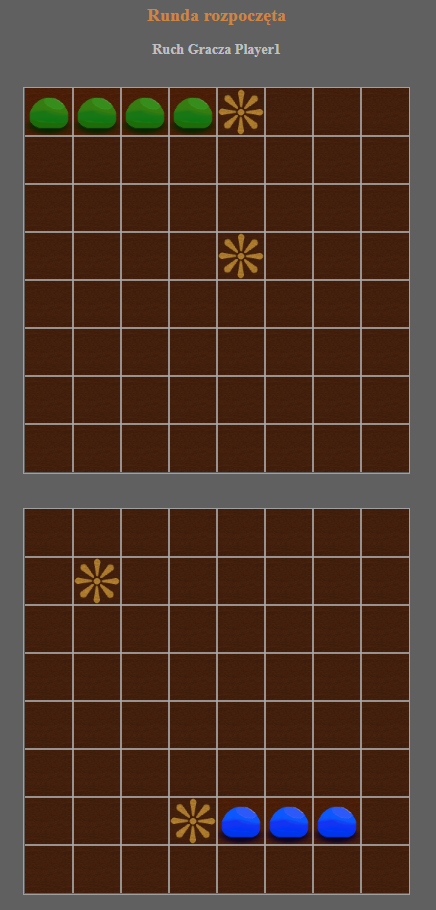
Gra planszowa w Statki stanowi przykład gry, w której każdy członek rozgrywki ma dostęp do różnej ilości informacji. Funkcjonalność ta jest łatwa do zaimplementowania, dzięki identyfikatorze użytkownika przesyłanemu w poleceniu Update. Rysunki 11, 12 oraz 13 przedstawiają hermetyzację stanu rozgrywki w zależności od roli członka pokoju. Każdy gracz widzi rozłożenie swoich okrętów na górnej planszy, natomiast dolna służy do wybierania pozycji do ostrzału przeciwnika. Obserwator ma dostęp wyłącznie do ostrzelanych pozycji.



Rysunek Plansze gracza pierwszego



Rysunek Plansze gracza drugiego



Rysunek Plansze obserwatora

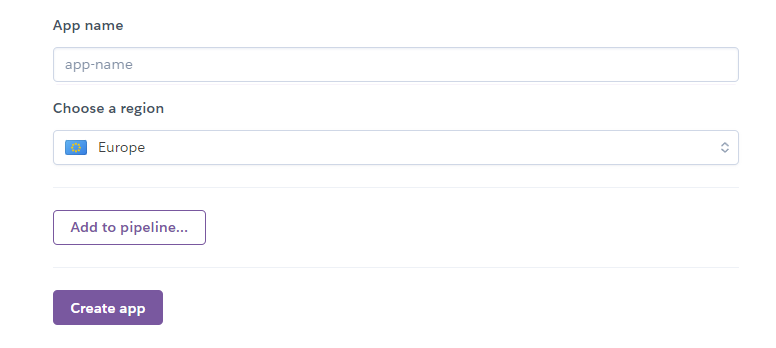
# Wdrożenie

Serwisy główny oraz logiki gier zostały opracowane na środowisku uruchomieniowym Node.js. Jest to popularne narzędzie, dla którego można znaleźć liczne rozwiązania pozwalające na wdrożenie aplikacji w sieci Internet lub Intranet. Każda z metod narzuca pewne ograniczenia. W ramach pracy przygotowany został opis wdrożenia systemu na platformie Heroku, jak również sposób konteneryzacji w oparciu o oprogramowanie Docker.

## Rozwiązanie w chmurze

Wdrożenie na platformie chmurowej Heroku oferującej swoim użytkownikom model PaaS dla kilku popularnych języków programistycznych np. Java, JavaScript lub Python. Daje dostęp do integracji z serwisem GitHub, umożliwiając automatyczną integrację z najnowszą wersją kodu znajdującą się w repozytorium.

W celu wdrożenia serwisów należało najpierw utworzyć pustą aplikację. Odbyło się to poprzez uzupełnienie formularza widocznego na rysunku 14. Po podaniu nazwy dla nowego serwera oraz regionu, w którym ma się on znajdować, wystarczyło kliknąć przycisk Create app.



Rysunek Heroku – utworzenie aplikacji

Każda nowopowstała aplikacja posiada wewnętrzne repozytorium Heroku Git, z którego można korzystać normalnie jak z odpowiednika znajdującego się na platformie GitHub. Środowisko oferuje również możliwość połączenia już istniejącego repozytorium kodu z aplikacją.

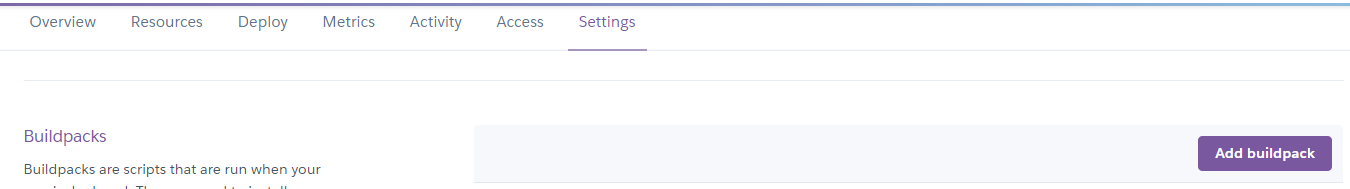
Wdrożona aplikacji korzystała z kodu źródłowego znajdującego się w repozytorium na platformie GitHub. W celu dodania repozytorium do aplikacji należało podać jego adres wraz z gałęzią, z której Heroku będzie pobierało automatycznie kod do uruchomienia serwisu. Efekt prawidłowego połączenia z repozytorium przedstawia rysunek 15.

Obraz zawierający tekst

Opis wygenerowany automatycznie

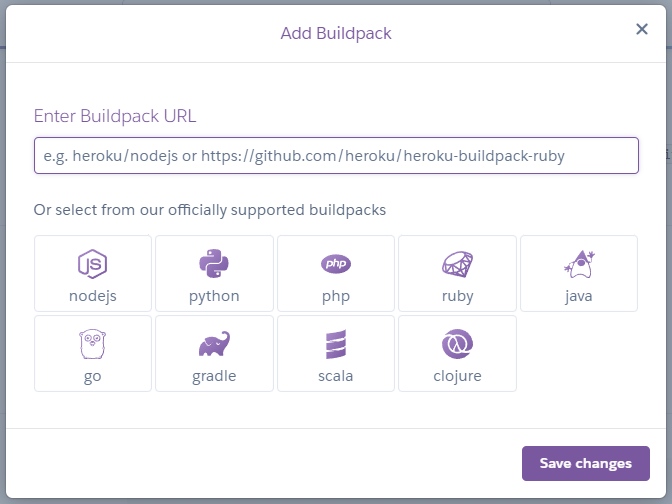
Rysunek Heroku – połączenie z repozytorium

Kolejnym ważnym etapem wdrożenia było wybranie buildpack’a, który definiuje proces uruchomienia aplikacji. Opcja ta znajduję się w zakładce Settings pod przyciskiem Add buildpack, widocznej na rysunku 16 .



Rysunek Heroku – dodanie buildpack'a

Rysunek 17 ilustruje dziewięć oficjalnych buildpack’ów wspieranych przez platformę Heroku. Wśród których znajduje się również ten zastosowany przy implementacji serwisu. Po wybraniu nodejs jako procesu uruchomieniowego konfiguracja aplikacji była prawie ukończona.



Rysunek Heroku – wybór buildpack'a

W celu finalizacji wdrożenia serwisu, należało zmodyfikować jeden fragment kodu, który odpowiada za rozpoczęcie nasłuchiwania na żądania HTTP przez obiekt klasy Express. Zmiana ta jest charakterystyczna dla platformy Heroku, która narzuca numer portu TCP. Fragment kodu na listingu 3 pokazuje jak poprawnie odczytać wartość przydzielonego portu.

1. const port = process.env.PORT;
2. var server = app.listen(port, () => {

customLog('App listening');

1. });

Listing 3 Heroku – odczyt portu

Zauważyć można również, że serwer powstał w efekcie wykonania metody listen klasy Express. Funkcja ta zwraca obiekt klasy http.Server. Platforma Heroku automatycznie przygotowała bramę Reverse-Proxy, która obsługuje komunikację protokołem HTTPS, natomiast do aplikacji trafiają przetworzone żądania w formie zapytań HTTP.

Ważną obserwacją jest różnica w numerze portu TCP, na który wysyłane jest żądanie z tym które je otrzymuje. Wszystkie zapytanie do aplikacji kierowane są na standardowy port HTTPS 443. Natomiast serwer aplikacji otrzymuje żądania na port przydzielony przez platformę. Rozbieżność z numerami portów wymagała również modyfikacji wszystkich deklaracji gier. W każdej deklaracji pole port musiało mieć wartość 443. Jest wartość domyślna dla tego pola, więc dopuszczalne byłoby pominięcie tego pola.

## Rozwiązanie w kontenerach

Alternatywną do rozwiązania chmurowego było zbudowanie własnych kontenerów. W tym celu wykorzystane zostały usługi oferowane przez oprogramowanie Docker, będące narzędziem pozwalającym na generację izolowanych środowisk uruchomieniowych dla aplikacji rozproszonych. Każdy kontener współdzieli jądro systemu, co znacznie zmniejsza ich rozmiar. Przygotowanie kontenerów składało się z dwóch prostych kroków. Najpierw należało przygotować przepis, według którego ma powstały. Odbyło się to przez utworzenie w folderze głównym kody źródłowego aplikacji pliku o nazwie Dockerfile bez rozszerzeń oraz umieszczenie kodu z listingu 5 wewnątrz pliku.

FROM node:16

WORKDIR /app

COPY package.json /app

RUN npm install

COPY . /app

CMD ["npm", "start"]

Listing 4 Treść Dockerfile

Następnie wystarczyło uruchomić polecenie

„docker build nazwa-kontenera .”

wewnątrz katalogów zawierających odpowiednio serwis logiki oraz główny, aby wywołać proces konstrukcji kontenera zgodnie podanym przepisem. Każda linia pliku Dockerfile odpowiadała jednemu etapowi tworzenia kontenera. Najpierw został przygotowany kontener podstawowy, w tym przypadku był to kontener zawierający środowisko Node.js w wersji 16. Następnie został przygotowany katalog roboczy o nazwie app, do którego Docker skopiował plik packege.json. Kolejno wywołane zostało polecenie npm install, które zainstalowało wszystkie niezbędne pakiety dla środowiska Node.js. Później został skopiowany do katalogu roboczego kod źródłowy oraz polecenie

„npm start”

zostało ustawione jako polecenia wywoływane przy starcie instancji kontenera.

Po zbudowaniu kontenerów następnym zadaniem było uruchomienie po jednej instancji kontenera z serwerem głównym oraz z serwisem logiki. W celu umożliwienia komunikacji między kontenerami, oba musiały się znaleźć w jednej sieci lokalnej Docker. Utworzona została dedykowana sieć za pomocą polecenia

„docker network create my-net”.

Następnie każdy kontener został uruchomiony wewnątrz sieci my-net. Posłużyły do tego polecenia

„docker run -it -p 443:443 --network my-net --name glowny server”,

dla serwera głównego oraz

„docker run -it -p 8443:443 --network my-net --name logika games”,

dla serwisu logiki.

Dla obydwóch kontenerów należało ustawić przekierowanie portów, aby umożliwić użytkownikowi dostęp do nich z poziomu przeglądarki. Uruchomienie kontenerów w trybie interaktywnym pozwoliło na łatwy dostęp do adresów IPv4 im przypisanych. Alternatywnie można byłoby uzyskać tę samą informację korzystając z polecenia

„docker network inspect my-net”.

Adres ten był potrzebny do przeprowadzenia rejestracji gry, ponieważ serwis główny może komunikować się z serwerem logiki wyłącznie za pomocą adresów wewnątrz sieci my-net.

Implementacja w oparciu o kontenery wymagała również obsłużenia certyfikatów SSL dla bezpiecznych połączeń protokołu HTTPS. Na potrzeby aplikacji utworzony został certyfikat typu Self-Signed dla domeny localhost, dodatkowo serwer główny został skonfigurowany tak, aby nie sprawdzał poprawności certyfikatów w czasie komunikacji z serwisem logiki. Jest to rozwiązanie wyłącznie dla sieci lokalnych.

# Podsumowanie oraz wnioski

Celem pracy było opracowanie aplikacji do udostępniania gier planszowych, składającej się z dwóch komunikujących się ze sobą serwisów HTTPS. W ramach pracy opracowany został prosty interfejs pozwalający na ustandaryzowanie przepływu informacji między interfejsem graficznym użytkownika, a maszyną stanów reprezentującą zbiór reguł gry planszowej. Zaproponowane rozwiązanie umożliwiło utworzenie czterech różnych gier oraz obsłużenie ich za pomocą jednego serwisu głównego. Dodatkowo zachowana została pełna hermetyzacja danych, jak również separacja implementacji pomiędzy dwiema częściami aplikacji. Oferowane przez interfejs funkcjonalności narzuciły nie liczne obostrzenia na formę rozwiązania.

Opracowana aplikacja cechuje się uproszczonym modelem, składającym się wyłącznie z dwóch serwerów HTTPS. Aplikacja zaprojektowana w sposób, który umożliwia na dołączanie większej liczby serwisów logiki w ramach jednego serwera główne. W przypadku korzystania z publicznie dostępnych serwerów logiki, należałoby przenieść moduł walidacji OpenAPI z serwisu logiki do nowego serwera pośredniczącego. Kierując wszystkie zapytania przez powstałą bramę, można byłoby zagwarantować poprawność odpowiedzi od nieznanych serwerów logiki ze specyfikacją interfejsu. Takie rozwiązanie wyeliminowałoby wymóg implementacji walidacji po stronie logiki.

W czasie wdrożenia aplikacji w formie kontenerów pojawił się problem z nawiązywaniem bezpiecznego połączenia między serwerami HTTPS. Został on rozwiązany poprzez wyłączenie sprawdzania poprawności certyfikatów SSL. W przypadku wdrożenia w sieci Internet potrzebny byłby certyfikat wystawiony przez zewnętrzy ośrodek certyfikacyjny. W celu zapobiegnięcia potrzeby ciągłego rozsyłania nowych certyfikatów po przedawnieniu poprzednich możliwe jest zastosowanie serwera pośredniczącego Reverse-proxy, przez który przesyłane byłyby wszystkie żądania HTTPS, ponieważ większość narzędzi Reverse-proxy oferuje automatyczne odnawianie certyfikatów SSL. Do serwisów aplikacji trafiałyby zapytania nieszyfrowane, wymagałoby to restrukturyzacji budowy połączeń między serwisami.

Funkcjonalność przedstawionego interfejsu implementuje wyłącznie minimalną liczbę procedur potrzebnych do obsługi prostych gier planszowych. Pomimo bieżących limitacji jest to rozwiązanie z możliwością rozbudowy o dodatkowe funkcje pozwalające na wdrożenie bardziej wyszukanego interfejsu graficznego. Do aplikacji warto byłoby dodać m.in. planszę reprezentującą zbiór kart do gry gracza, obszar punktację w czasie pojedynczej rundy, jak również możliwość wyznaczenia przedziału czasu, w którym gracze mogą wykonywać ruchy równocześnie.

# Bibliografia

[https://xhr.spec.whatwg.org](https://xhr.spec.whatwg.org/) – Opis standardu interfejsu XHR [Dostęp 28.11.2022]

[https://websockets.spec.whatwg.org](https://websockets.spec.whatwg.org/) – Opis standard protokołu WebSocket [Dostęp 28.11.2022]

[https://nodejs.org](https://nodejs.org/) – Środowisko uruchomieniowe Node.js [Dostęp 28.11.2022]

<https://expressjs.com> – Pakiet Express.js dla środowiska Node.js [Dostęp 28.11.2022]

<https://www.npmjs.com/package/ws> – Pakiet WebSocket dla środowiska Node.js [Dostęp 28.11.2022]

[https://www.openapis.org](https://www.openapis.org/) – Opis specyfikacji OpenAPI [Dostęp 28.11.2022]

[https://swagger.io](https://swagger.io/) – Oprogramowanie Swagger Editor [Dostęp 28.11.2022]

[https://www.visual-paradigm.com](https://www.visual-paradigm.com/) – Narzędzie użyte do utworzenia diagramów UML [Dostęp 28.11.2022]

[https://www.heroku.com](https://www.heroku.com/) – Platforma chmurowa Heroku [Dostęp 28.11.2022]

<https://www.docker.com> – Narzędzie do konteneryzacji Docker [28.11.2022]