|  |
| --- |
| **Politechnika Wrocławska Wydział Informatyki i Zarządzania Katedra Informatyki** |
| **STUDIA PODYPLOMOWE Technologie Internetowe** |
| Projekt i implementacja systemu społecznościowego do gry w szachy w sieci internetowej |
|  |
| Marcin Kużdowicz |
|  |

|  |
| --- |
| Wrocław 2015 |

Opiekun pracy: Doc. dr inż. Zbigniew Szpunar

Spis treści

[1. Cel pracy i zakres pracy: 4](#_Toc432120233)

[2. Opis wycinka rzeczywistości i słownik pojęć biznesowych. 4](#_Toc432120234)

[2.1. Wstęp 4](#_Toc432120235)

[2.2. Słownik pojęć biznesowych 5](#_Toc432120236)

[2.3. Opis wycinka rzeczywistości 6](#_Toc432120237)

[3. Specyfikacja wymagań funkcjonalnych i niefunkcjonalnych 6](#_Toc432120238)

[3.1 Wymagania funkcjonalne 6](#_Toc432120239)

[3.1.1 Ogólne wymagania systemowe 6](#_Toc432120240)

[3.1.2. Perspektywa administratora 7](#_Toc432120241)

[3.1.3. Perspektywa Użytkownika 7](#_Toc432120242)

[3.2. Wymagania niefunkcjonalne 8](#_Toc432120243)

[3.3. Diagram przypadków użycia 9](#_Toc432120244)

[4. Wybór technologii, ogólna architektura systemu i opracowanie modelu danych oraz bazy danych 10](#_Toc432120245)

[4.1 Wybór technologii 10](#_Toc432120246)

[4.2 Architektura aplikacji. 13](#_Toc432120247)

[4.2.1 Ogólna architektura aplikacji. 13](#_Toc432120248)

[4.2.2 Drzewo plików projektu 13](#_Toc432120249)

[4.2.3 Diagram klas 1](#_Toc432120250)

[4.2.4 Model bazy danych 1](#_Toc432120251)

[4.2.5 Operacje na bazie danych 2](#_Toc432120252)

[5. Realizacja systemu 5](#_Toc432120253)

[5.1 Tworzenie konta użytkownika 5](#_Toc432120254)

[5.2 Autentykacja i uwierzytelnianie 6](#_Toc432120255)

[5.3 Implementacja gry w szachy. 9](#_Toc432120256)

[5.4 Gra w szachy użytkownik versus użytkownik implementacja komunikatora do gry. 13](#_Toc432120257)

[6. Testy 24](#_Toc432120258)

[7. Podsumowanie 24](#_Toc432120259)

[Bibliografia 24](#_Toc432120260)

[Spis ilustracji 24](#_Toc432120261)

[Spis tabel 24](#_Toc432120262)

[Załączniki 24](#_Toc432120263)

# Cel pracy i zakres pracy:

Celem pracy jest zaprojektowanie i implementacja systemu informatycznego służącej do gry w szachy w sieci internetowej. System będzie aplikacją webową z możliwością rejestracji i logowania do systemu. Założeniem jest iż użytkownik aplikacji będzie mógł grać w szachy z komputerem lub po utworzeniu konta i zalogowaniu z innymi użytkownikami w zależności od tego czy inni użytkownicy są w tym momencie zalogowani do systemy i obecni w repozytorium gry online. Aplikacja również ma gromadzić i wyświetlać dane statystyczne na temat wyników pojedynków szachowych użytkowników. System ma prezentować dziesięciu najlepszych graczy, czyli takich którzy wygrali najwięcej pojedynków szachowych, w postaci wykresu słupkowego. Program ma również gromadzić dane na temat odbytych gier poszczególnych użytkowników oraz prezentować aktualnie zalogowanemu użytkownikowi w jego panelu historie gier odbytych przez niego.

# Opis wycinka rzeczywistości i słownik pojęć biznesowych.

## Wstęp

Jednym z najciekawszych funkcji współczesnego Internetu jest możliwość komunikacji w czasie rzeczywistym miedzy dwoma osobami lub grupą osób znajdujących się fizycznie w innych miejscach. Funkcjonalność ta może służyć do wymiany słów między ludźmi w formie tekstowej, dźwiękowej, informacji medialnych w postaci zdjęć czy filmów jak również wymianą zachowań na określone bodźce według określonych reguł, tak jak to ma miejsce w postaci gier społecznościowych online.

Podjętym problemem w niniejszej pracy tej pracy jest stworzenie takiego systemu, który będzie skupiał określoną grupę użytkowników i umożliwiał im prowadzenie pojedynków szachowych między sobą.

Sam temat jest interesujący wyzwaniem programistycznym, gdyż będzie wymagał użycia szerokiej palety technologii, jak i zastosowanie złożonej logiki mającej na celu obsłużenie komunikacji szachowej w czasie rzeczywistym.

# Słownik pojęć biznesowych

|  |  |
| --- | --- |
| Użytkownik | Osoba korzystająca z oprogramowania |
| Administrator | Osoba korzystająca z oprogramowania i mająca uprawnienia do zarządzania innymi użytkownikami |
| Szachy | Rodzaj gry planszowej będący dla dwóch osób |
| Szachownica | Obiekt/ strefa, w kontekście której gracze prowadzą grę |
| Pojedynek szachowy online | Gra w szachy pomiędzy dwoma użytkownikami w czasie rzeczywistym za pomocą oprogramowania |
| Lista najlepszych graczy | Posortowana Lista dziesięciu użytkowników, którzy wygrali największą liczbę pojedynków szachowych |
| Figura szachowa | Obiekt pojedynczego elementu poruszającego się po planszy szachowej. Reprezentowanego w systemie w postaci ikony |
| Kolor figury szachowej | Obiekt dwustanowy albo biały albo czarny. Obiekt biały zawsze rozpoczyna grę |
| Status gry | Aktualna sytuacja w jakiej znajduje się gra, np: ruch dla figur białych, szach, szach mat. |
| Fen | Notacja Forsytha-Edwardsa. Ciąg znaków reprezentujący grę, za pomocą którego można odtworzyć całą jej postać; |
| Ruch szachowy | Ruch szachowy |
| Komunikator do gry online | Część oprogramowania służąca do wysyłania ruchów szachowych użytkowników przez sieć internetową między dwoma użytkownikami. Umożliwiający również zapraszanie do gry, wyświetlanie aktualnie dostępnych użytkowników |
| Oczekiwanie na nową grę | Stan w jakim znajduje Się użytkownik, który w danym momencie z nikim nie gra i jest online |
| Zaproszenie do gry | Stan w jakim znajduje Się użytkownik, który został zaproszony do gry lub sam zaprosił do gry |
| Stan w trakcie gry | Stan w jakim znajduje Się użytkownik, który aktualnie prowadzi grę z innym użytkownikiem |
| Pojedynek szachowy | Dwóch użytkowników grających przeciwko sobie |
| Wykres najlepszych graczy | 10 najlepszych graczy posortowanych według ilości wygranych pojedynków szachowych |
| Wiadomość od gracza do gracza | Wysłany ruch szachowy i aktualny status gry od jednego użytkownika do drugiego po wykonanym ruchu |

1 Słownik pojęć biznesowych

# Opis wycinka rzeczywistości

**Profil działalności**

Użytkownik sieci internetowej, zainteresowany grą w szachy, oraz grami online.

**Wizja systemu**

Dostęp do aplikacji ma się odbywać poprzez sieć internetową. Każdy użytkownik Internetu może odwiedzić stronę z aplikacją, ale ma dostęp tylko do jej niektórych zasobów. Do wszystkich zasobów ma dostęp dopiero po założeniu konta i zalogowaniu. Do zasobów administracyjnych ma dostęp wyłącznie użytkownik z rolą administratora.

System przewiduje dwie role dla zarejestrowanych użytkowników:

1. Administrator
2. Użytkownik

Użytkownikiem można się stać poprzez rejestrację w systemie, za pomocą formularza do rejestracji obecnego w interfejsie użytkownika.

# Specyfikacja wymagań funkcjonalnych i niefunkcjonalnych

## 3.1 Wymagania funkcjonalne

### 3.1.1 Ogólne wymagania systemowe

Na podstawie przeprowadzonych badań sformułowano następujące wymagania funkcjonalne:

* dodawanie/usuwanie użytkownika przez administratora
* rejestracja użytkowników internetu poprzez formularz tworzenia konta użytkownika, weryfikowanych za pomocą linka aktywacyjnego wysyłanego, na wcześniej podany adres email
* możliwość gry w szachy z komputerem
* możliwość gry w szachy z innym użytkownikiem w czasie rzeczywistym
* przeglądanie/danych na temat odbytych wcześniej pojedynków szachowych z innymi użytkownikami
* przeglądanie aktualnego rankingu najlepszych dziesięciu graczy w postaci wykresu słupkowego
* pojedynek szachowy wstępuje w postaci użytkownik kontra użytkownik
* użytkownik może prowadzić maksymalnie jedną grę w tym samym czasie
* użytkownik nie może prowadzić gry z kilkoma użytkownikami jednocześnie
* użytkownik przed rozpoczęciem pojedynku musi zaprosić drugiego użytkownika do gry.
* Zaproszony użytkownik ma 15 sekund na decyzje
* w momencie trwania zaproszenia, dana para użytkowników jest już zablokowana i inni użytkownicy nie mogą do nich wysłać wiadomości.
* użytkownik po rejestracji nie może zmienić swojego login, gdyż login jest identyfikatorem w grze online dla innych użytkowników
* użytkownik może edytować po rejestracji swoje dane takie jak: adres email, imię, nazwisko.
* użytkownik może zmienić hasło
* Wyniki pojedynków z komputerem nie są zapisywane do bazy, pełnią rolę warstwy demonstracyjnej.
* administrator może przeglądać listę użytkowników
* administrator może tworzyć użytkowników
* administrator może usuwać użytkowników

### Perspektywa administratora

* Administrator ma te same uprawnienia co zalogowany użytkownik
* Administrator ma dostęp do tych samych zasobów co zalogowany użytkownik
* Administrator systemu może dodawać nowych użytkowników
* Administrator może usuwać istniejące konta.
* Administrator może tworzyć konta z uprawnieniami administratora
* Administrator może nadawać uprawnienia administratora lub je odbierać.
* Administrator może edytować konta użytkownika

### Perspektywa Użytkownika

Niezalogowany użytkownik ma dostęp do:

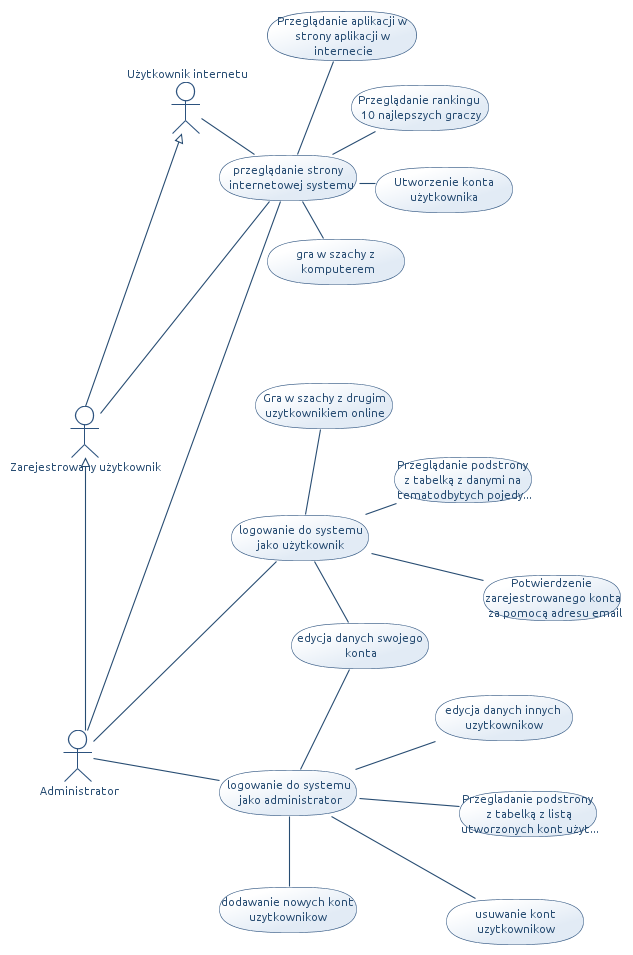
* Strony domowej
* możliwości zagrania w szachy z komputerem, aby zapoznać się z systemem.
* wykresu słupkowego z maksymalnie 10 najlepszymi graczami (przy założeniu że w bazie mamy dziesięciu graczy, którzy odbyli zwycięskie pojedynki w szachowe. Jeżeli jest ich mniej, to wyświetlana jest mniejsza liczba użytkowników)
* formularza logowania
* formularza z możliwością utworzenia konta

Zalogowany użytkownik ma dostęp do:

Tych samych zasobów co użytkownik niezalogowany, do tego:

* Możliwość grania w szachy z innymi użytkownikami online.
* Danych na temat swojego konta i możliwością ich edycji z wyłączeniem zmiany loginu.
* Danych na temat odbytych pojedynków szachowych

## Wymagania niefunkcjonalne

* Program jest aplikacją webową
* Program ma czytelny interfejs dla użytkownika
* Program ma służyć rozrywce
* Program propaguję grę w szachy w sieci
* Program ma rozwijać umiejętności logicznego myślenia u użytkownika
  1. Diagram przypadków użycia****

1 Diagram przypadków użycia

# Wybór technologii, ogólna architektura systemu i opracowanie modelu danych oraz bazy danych

## 4.1 Wybór technologii

* Java – obiektowy język programowania, główna technologia użyta w aplikacji do stworzenia kodu działającego po stronie serwera.

Więcej informacji: <https://www.java.com/pl/>

* Spring MVC – Framework języku java wspierający architekturę Model-View-Controller i zawierający gotowe komponenty, które mogą być wykorzystane do opracowania aplikacji internetowych.

Więcej informacji: <http://spring.io/>

* Spring security - Framework języku java wspierający zagadnienie bezpieczeństwa takie jak uwierzytelnianie i autoryzacja.

Więcej informacji: <http://spring.io/>

* JSP (Java server pages) - to technologia umożliwiająca tworzenie dynamicznych dokumentów WWW z wykorzystaniem języka Java, wplecionego w kod HTML danej strony. Jsp jest formatem pliku w którym można wykorzystać język zarówno htm jak i Java. Składnia jsp zawiera na elementy: html, elementy skryptowe skryplety (kod w języku Java kontrolujący proces generowania dokumentu) oraz elementy składniowe tzw. Expression Language oraz tagi XML wywołujące określone metody.
* JSTL (Java Standard Tag Library) – kolekcja tagów z przypisanymi funkcjonalnościami. Jstl wspiera typowe zadania takie jak iteracja czy konstrukcje warunkowe, wykonywane na widoku.
* Jetty – serwer www oraz kontener serwletów Javy. Więcej informacji: <http://www.eclipse.org/jetty/>
* Maven - narzędzie do zarządzania projektem Javowym. Projekt mavenowy definiuje się poprzez stworzenie i utrzymywanie pliku pom.xml (POM – ang. Project Object Model). **Pom.xml** jest głównym miejscem pracy z projektem i zawiera wszystkie istotne elementy definiujące projekt, jego strukturę, sposób budowania i przede wszystkim zależności. Zależności to m.in. zewnętrzne biblioteki importowane do projektu w postaci plików .jar

Więcej informacji: <http://maven.apache.org/>

* JSON – (Javascript Object Notation) format wymiany danych komputerowych, przypominający swoją strukturą tzw. Hashmapę, organizujący dane w postaci (klucz=wartość). Jest to format transportowy, którym został użyty w aplikacji do wysyłania ruchów szachowych pomiędzy użytkownikami, aktualnego stanu gry, zaproszenia do gry czy zrezygnowania z gry.
* XML (Extensible Markup Language) czyli rozszerzalny język znaczników. W projektach javowoych bardzo często używany do plików konfiguracyjnych, oraz w plikach jsp w postaci tagów z przypisanymi funkcjonalnościami za pomocą zaimportowanych przestrzeni nazw.
* Websocket - jest technologią zapewniającą dwukierunkowy kanał komunikacji (full-duplex) za pośrednictwem jednego połączenia TCP. Websocket jest niezależnym protokółem opartym na protokole TCP. W niniejszej aplikacji ta technologia została użyta do stworzenia komunikatora wysyłającego ruchy szachowe, zaproszenia do gry, rezygnacje z gry, koniec gry jak i listę aktualnie połączonych użytkowników. Dzięki tej technologii gra w szachy zaimplementowana w programie przybrała charakter tzw. *real-time games* (gry czasu rzeczywistego);
* JavaScript - język skryptowy wykonujący się po stronie klienta w przeglądarce, dostarczający programiście możliwości programowania interfejsu użytkownika zdarzeniowego. Interfejs do gry w szachy jak i logika gry w szachy jest obsługiwana zdarzeniowo za pomocą javascript w większości poprzez zdarzenia drag and drop.

Więcej informacji: <https://www.javascript.com/>

* JQuery – biblioteka jasvacritp dostarczająca dodatkowe narzędzia i umożliwiająca zaprogramowanie tych samych funkcjonalności za pomocą mniejszej ilości kodu.

Więcej informacji: <https://jquery.com/>

* Bootstrap 3 – biblioteka css i javascript do tworzenia interfejsów użytkownika stworzona przez programistów Twittera. Główną cechą bootsrapa są dostarczane przez niego gotowe klasy css, gwarantujące dobry wygląd interfejsu i jego responsywność.

Więcej informacji: <https://jquery.com/>

* MongoDB – baza danych typu NoSql.   Charakteryzuje się dużą skalowalnością, wydajnością oraz brakiem ściśle zdefiniowanej struktury obsługiwanych baz danych, co w przypadku rozwijania niniejszego systemu było zaletą. W MongoDb dane składowane są jako dokumenty w postaci JSON. Do aplikacji typu live-game baza danych tego typu stanowi właściwy wybór ze względu na szybkość działania i możliwość dynamicznego dostosowywanie się tabel/kolekcji w bazie pod wpływem zmian jakie są dyktowane przez kod Java, bez konieczności zmiany struktury bazy danych za pomocą dodatkowych zapytań.

*Cechą charakterystyczna jest to że gdy chcemy do tzw. Kolekcji (odpowiednik tabelki w bazach typu sql) dodać dodatkowe pole, to nie musimy najpierw go tworzyć, ono samo się utworzy przy pierwszym zapisie, co jest bardzo wygodne.*

Więcej informacji: <https://www.mongodb.com/>

* Html 5 - jest najnowszą wersją standardu opisującego język HTML. Zalety tej technologii zostały użyte np. przy walidacji formularzy po stronie klienta np. za pomocą atrybutu required czy type=”email” oraz w przekazywaniu danych do javascriptu za pomocą data attributes np. data-username=”admin”.

Więcej informacji: <http://www.w3.org/TR/html5/>

* CSS 3 – kaskadowe arkusze stylów, język odpowiedzialny za wygląd aplikacji.

Więcej informacji: <http://www.css3.info/>

* DataTables – plugin do jquery umożliwający szybkie oprogramowanie tabelki wraz z paginacją zaimplementowaną po stronie klienta w javascript.

Więcej informacji: <https://www.datatables.net/>

* Canvasjs – biblioteka javascript umożliwiająca wyświetlenie responsywnych histogramów, słupków, wykresów prezentujących dane statystyczne w przyjaznym dla użytkownika formacie. W programie użyta do wyświetlenia rankingu 10 najlepszych graczy w postaci wykresu słupkowego.

Więcej informacji: <https://www.datatables.net/>

* Chessboardjs - biblioteka javascript dostarczająca interfejs graficzny do gry w szachy.

Więcej informacji: <http://chessboardjs.com/>

* Chess.js - biblioteka javascript implementująca logikę gry w szachy umożliwiająca integrację z Chessboardjs oraz oprogramowanie gry w szachy zdarzeniowo przez programistę.

Więcej informacji: <https://github.com/jhlywa/chess.js/>

* GIT – System kontroli wersji umożliwiający zarządzanie poszczególnymi wersjami aplikacji i przechowywanie wszystkich wersji na lokalnym i zdalnym repozytorium.

Więcej informacji: <https://git-scm.com/>

## 4.2 Architektura aplikacji.

### 4.2.1 Ogólna architektura aplikacji.

Program jest oparty na wzorcu architektonicznym MVC (model view controller) zaimplementowanej przy pomocy komponentów dostarczanych przez framework Spring MVC.



2 MVC model

### 4.2.2 Drzewo plików projektu

**src.**

├── main

│   ├── java

│   │   └── com

│   │   └── chessApp

│   │   ├── controllers

│   │   │   ├── AdminPanelController**.**java

│   │   │   ├── HomeController**.**java

│   │   │   ├── LoginController**.**java

│   │   │   ├── PlayChessSitesController**.**java

│   │   │   ├── RegistrationConfirmationController**.**java

│   │   │   ├── SignInController**.**java

│   │   │   ├── UserPanelController**.**java

│   │   │   └── UsersAjaxController**.**java

│   │   ├── dao

│   │   │   ├── ChessGamesRepository**.**java

│   │   │   └── UsersRepository**.**java

│   │   ├── daoimpl

│   │   │   ├── ChessGamesRepositoryImpl**.**java

│   │   │   └── UsersRepositoryImpl**.**java

│   │   ├── exceptions

│   │   │   └── UserNotConfirmedException**.**java

│   │   ├── gameProtocol

│   │   │   ├── ChessColor**.**java

│   │   │   ├── ChessMoveStatus**.**java

│   │   │   ├── GameMessageProtocol**.**java

│   │   │   ├── GameMessageType**.**java

│   │   │   └── GameUserCommunicationStatus**.**java

│   │   ├── mailService

│   │   │   ├── MailService**.**java

│   │   │   └── MailSubjectPL**.**java

│   │   ├── model

│   │   │   ├── ChessAppUser**.**java

│   │   │   ├── ChessGame**.**java

│   │   │   ├── ChessMove**.**java

│   │   │   ├── GameMessage**.**java

│   │   │   ├── GameUser**.**java

│   │   │   └── UserAccount**.**java

│   │   ├── props

│   │   │   └── ChessAppProperties**.**java

│   │   ├── security

│   │   │   ├── LocalAuthenticationProvider**.**java

│   │   │   ├── PasswordEncryptor**.**java

│   │   │   └── UserRoles**.**java

│   │   ├── taskService

│   │   │   └── ScheduledTaskService**.**java

│   │   └── websocket

│   │   ├── ChessGamesHandler**.**java

│   │   ├── GameUsersHandler**.**java

│   │   ├── WebSocketServer**.**java

│   │   └── WebSocketSessionHandler**.**java

│   ├── resources

│   │   ├── chessApp**.**properties

│   │   └── log4j**.**xml

│   └── webapp

│   ├── assets

│   │   ├── css

│   │   │   ├── dataTable**.**css

│   │   │   ├── fonts

│   │   │   │   ├── glyphicons**-**halflings**-**regular**.**eot

│   │   │   │   ├── glyphicons**-**halflings**-**regular**.**svg

│   │   │   │   ├── glyphicons**-**halflings**-**regular**.**ttf

│   │   │   │   ├── glyphicons**-**halflings**-**regular**.**woff

│   │   │   │   └── glyphicons**-**halflings**-**regular**.**woff2

│   │   │   ├── images

│   │   │   │   ├── favicon**.**ico

│   │   │   │   ├── sort\_asc\_disabled**.**png

│   │   │   │   ├── sort\_asc**.**png

│   │   │   │   ├── sort\_both**.**png

│   │   │   │   ├── sort\_desc\_disabled**.**png

│   │   │   │   ├── sort\_desc**.**png

│   │   │   │   └── Sorting icons**.**psd

│   │   │   ├── lib

│   │   │   │   ├── bootstrap**.**min**.**css

│   │   │   │   ├── bootstrap**-**theme**.**min**.**css

│   │   │   │   ├── chessboard**-**0.3.0.css

│   │   │   │   └── jquery**.**dataTables**.**min**.**css

│   │   │   └── style**.**css

│   │   ├── icons

│   │   │   └── myicon**.**png

│   │   ├── images

│   │   │   ├── chess2**.**jpg

│   │   │   ├── chess**.**jpg

│   │   │   └── chesspieces

│   │   │   └── wikipedia

│   │   │   ├── bB**.**png

│   │   │   ├── bK**.**png

│   │   │   ├── bN**.**png

│   │   │   ├── bP**.**png

│   │   │   ├── bQ**.**png

│   │   │   ├── bR**.**png

│   │   │   ├── wB**.**png

│   │   │   ├── wK**.**png

│   │   │   ├── wN**.**png

│   │   │   ├── wP**.**png

│   │   │   ├── wQ**.**png

│   │   │   └── wR**.**png

│   │   └── js

│   │   ├── bestPlayersMain**.**js

│   │   ├── chessComputerVsComputer**.**js

│   │   ├── chessUserVsComputer**.**js

│   │   ├── chessUserVsUser**.**js

│   │   ├── lib

│   │   │   ├── bootstrap**.**min**.**js

│   │   │   ├── chessboard**-**0.3.0.js

│   │   │   ├── chess**.**js

│   │   │   ├── jquery**-**2.1.4.min.js

│   │   │   ├── jquery**.**canvasjs**.**min**.**js

│   │   │   └── jquery**.**dataTables**.**min**.**js

│   │   ├── main**.**js

│   │   ├── playChessWithUserMain**.**js

│   │   ├── userGamesHistory**.**js

│   │   ├── websocketClientEndpoint**.**js

│   │   ├── wsClientEndpointFunctions**.**js

│   │   └── wsClientMessageExchangeProtocol**.**js

│   ├── index**.**jsp

│   └── WEB**-**INF

│   ├── application**-**context**.**xml

│   ├── mongo**-**config**.**xml

│   ├── mvc**-**config**.**xml

│   ├── scheduled**-**tasks**-**config**.**xml

│   ├── security**-**config**.**xml

│   ├── views

│   │   ├── adminAccount**.**jsp

│   │   ├── bestPlayers**.**jsp

│   │   ├── confirmRegistrationMessage**.**jsp

│   │   ├── creatAccountMessage**.**jsp

│   │   ├── editUser**.**jsp

│   │   ├── error**.**jsp

│   │   ├── home**.**jsp

│   │   ├── includes

│   │   │   ├── footer**.**jsp

│   │   │   ├── forms

│   │   │   │   ├── addUserForm**.**jsp

│   │   │   │   ├── editUserForm**.**jsp

│   │   │   │   ├── editYourAccountFormADMIN**.**jsp

│   │   │   │   ├── editYourAccountForm**.**jsp

│   │   │   │   ├── logInForm**.**jsp

│   │   │   │   └── signInForm**.**jsp

│   │   │   ├── header**.**jsp

│   │   │   ├── menu**.**jsp

│   │   │   ├── modal\_boxes

│   │   │   │   ├── game\_handshake\_invitaion\_modal**.**jsp

│   │   │   │   ├── game\_handshake\_response\_modal**.**jsp

│   │   │   │   ├── removeUserModal**.**jsp

│   │   │   │   ├── user\_end\_chess\_game\_pos\_modal**.**jsp

│   │   │   │   └── user\_info\_modal**.**jsp

│   │   │   ├── tables

│   │   │   │   ├── userGamesTable**.**jsp

│   │   │   │   └── usersTable**.**jsp

│   │   │   ├── usersSiteWithAddUserFormActive**.**jsp

│   │   │   └── usersSiteWithUsersTableActive**.**jsp

│   │   ├── login**.**jsp

│   │   ├── playChessWithComputer**.**jsp

│   │   ├── playChessWithUser**.**jsp

│   │   ├── signIn**.**jsp

│   │   ├── userGames**.**jsp

│   │   ├── users**.**jsp

│   │   └── yourAccount**.**jsp

│   └── web**.**xml

└── test

├── java

└── resources

└── log4j**.**xml

**Kontroler**

Wszystkie klasy Javowe znajdują się w pakiecie **src/main/java**

Requesty HTTP obsługują klasy w pakiecie **com.chessApp.controllers .**

Te klasy pełnią rolę **kontrolerów**. Są opatrzone anotacji @Controller co informuje frameworw Spring o tym że ta klasa jest kontrolerem.

**Widok** jest realizowany za pomocą technologii: Jsp (java server pages) , Jtsl (JavaServer Pages Standard Tag Library), HTML 5, CSS 3, JavaScript i Jquery oraz Bootstrap . Pliki widoków znajdują się w głównym folderze: **/src/main/webapp/ WEB-INF/** **views**

w folderze **/src/main/webapp/assets** znajdują się pliki css i javascript

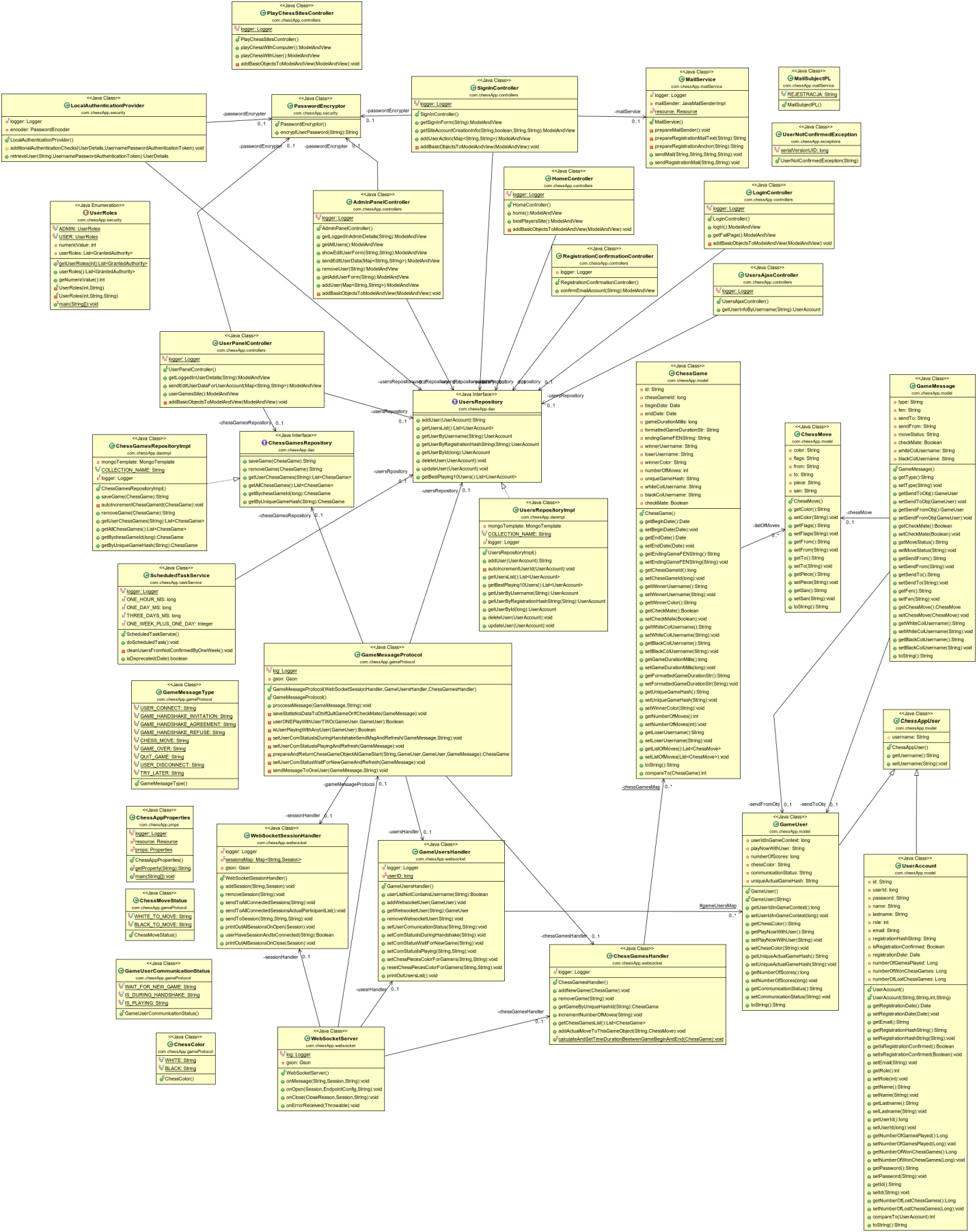
w folderze **assets/js** są pliki źródłowe javascript, w folderze **assets/js/lib** są biblioteki javascript

w folderze **assets/css** są pliki źródłowe css, w folderze **assets/css/lib** są biblioteki css

w folderze **WEB-INF/** **views** są pliki .jsp. W folderze **WEB-INF** znajdują się pliki konfiguracyjne .xml.

**Model** znajduje się w pakiecie **com.chessApp.model.** Klasy UserAccount, ChessApp, Move są obiektową reprezentacją dokumentów przechowywanych w bazie danych

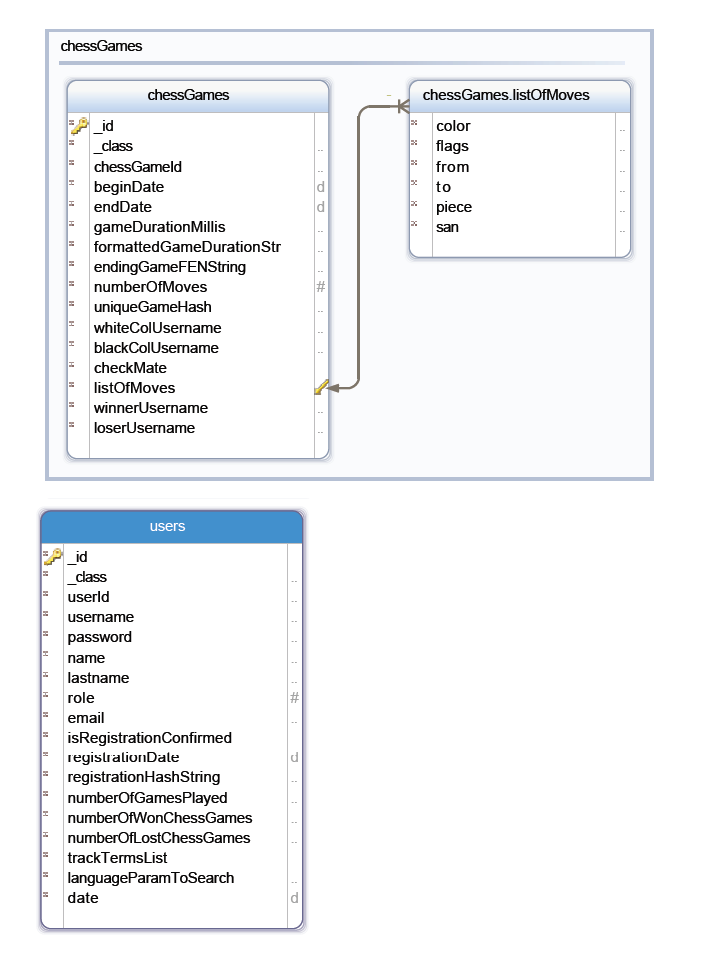
Aplikacja wykorzystuje bazę danych MonogDB, która jest bazą NoSQL. W systemie jest zastosowana koncepcja **ORM** (object relational mapping), za pomocą technologii Spring Data. ORM mapuje obiekty i kolekcje z bazy danych MongoDB do klas Javovych i dzięki zaletom jakie daje MongoDB, ten mechanizm dzieje się również też odwrotnie.

4.2.3 Diagram klas

3 Diagram klas

### 4.2.4 Model bazy danych

Nazwa bazy danych**: chessapp\_db.** Poniżej widać dwie kolekcje chessGames(z zagnieżdżonym obiektem move) i users.

****

4 Model bazy danych

Konfiguracja połaczenia z bazą danych znajduje się w pliku **mongo-config.xml** pod ścieżką **/src/main/webapp/ WEB-INF**

**MongoDb** nie jest bazą relacyjną, nie występuje tu koncepcja kluczy obcych, natomiast jest możliwość przechowywania zagnieżdżonych obiektów i list typów prostych, stringów jak również list obiektów. W MongoDb zamiast pojęcia tabelka stosuje się pojęcie kolekcja bądź dokument.: Kolekcja chessGames przechowuje listę obiektów Move. Obiekt Move reprezentuje ruch szachowy.

Między obiektami ChessGame i Move występuje relacja Has-a. W postaci agregacji: klasa **ChessGames** zawiera listę obiektów Move, która może mieć zero lub wiele elementów.

### 4.2.5 Operacje na bazie danych

Do operacji typu CRUD (*create, read, update, delete*) na zasobach w bazie danych został użyty Framework Spring MVC i jego komponent Spring Data MongoDB służący do interakcji z bazą typu mongodb. W pracy został użyty wzorzec DAO : Data Access Object, czyli koncepcja stworzenia odseparowanej klasy skupiającej się na pojedynczej odpowiedzialności interakcji z bazą danych w postaci działań typu CRUD. Projekt objął zdefiniowanie metod w postaci interfejsów: **UsersRepository** i **ChessGamesRepository** i ich implementacji **UsersRepositoryImpl** i **ChessRepositoryImpl.**

Przykład kodu:

Interfejs:

public interface UsersRepository **{**

public String addUser**(**UserAccount user**);**

public List**<**UserAccount**>** getUsersList**();**

public UserAccount getUserByUsername**(**String username**);**

public UserAccount getUserByRegistrationHashString**(**

String registrationHashString**);**

public UserAccount getUserById**(**long userId**);**

public void deleteUser**(**UserAccount user**);**

public void updateUser**(**UserAccount user**);**

public List**<**UserAccount**>** getBestPlaying10Users**();**

**}**

Implementacja interfejsu:

@Repository

public class UsersRepositoryImpl **implements** UsersRepository **{**

@Autowired

private MongoTemplate mongoTemplate**;**

public static final String COLLECTION\_NAME **=** "users"**;**

private final Logger logger **=** Logger**.**getLogger**(**UsersRepositoryImpl**.**class**);**

public String addUser**(**UserAccount user**)** **{**

logger**.**debug**(**"addUser()"**);**

**if** **(!**mongoTemplate**.**collectionExists**(**UserAccount**.**class**))** **{**

mongoTemplate**.**createCollection**(**UserAccount**.**class**);**

**}**

UserAccount testUser **=** getUserByUsername**(**user**.**getUsername**());**

**if** **(**testUser **==** **null)** **{**

autoIncrementUserId**(**user**);**

mongoTemplate**.**insert**(**user**,** COLLECTION\_NAME**);**

logger**.**info**(**"user added"**);**

**return** "ok"**;**

**}** **else** **{**

logger**.**info**(**"login allready exist"**);**

**return** "fail"**;**

**}**

**}**

private void autoIncrementUserId**(**UserAccount usertoUpdate**)** **{**

logger**.**debug**(**"incrementUserId()"**);**

Query query **=** **new** Query**();**

UserAccount user **=** mongoTemplate**.**findOne**(**query**,** UserAccount**.**class**,**

COLLECTION\_NAME**);**

**if** **(**user **==** **null)** **{**

usertoUpdate**.**setUserId**(**0**);**

**}** **else** **{**

List**<**UserAccount**>** users **=** mongoTemplate**.**findAll**(**UserAccount**.**class**,**

COLLECTION\_NAME**);**

UserAccount lastUserBefore **=** users**.**get**(**users**.**size**()** **-** 1**);**

long lastUserBeforeId **=** lastUserBefore**.**getUserId**();**

logger**.**info**(**"lastUser id before sorting"**);**

logger**.**info**(**lastUserBeforeId **+** ""**);**

Collections**.**sort**(**users**);**

UserAccount lastUser **=** users**.**get**(**users**.**size**()** **-** 1**);**

long lastUserId **=** lastUser**.**getUserId**();**

logger**.**info**(**"lastUser id after sorting"**);**

logger**.**info**(**lastUserId **+** ""**);**

usertoUpdate**.**setUserId**(**lastUserId **+** 1**);**

**}**

**}**

public List**<**UserAccount**>** getUsersList**()** **{**

logger**.**debug**(**"getUsersList()"**);**

Query query **=** **new** Query**();**

query**.**with**(new** Sort**(**Sort**.**Direction**.**DESC**,** "userId"**));**

**return** mongoTemplate**.**find**(**query**,** UserAccount**.**class**,** COLLECTION\_NAME**);**

**}**

public List**<**UserAccount**>** getBestPlaying10Users**()** **{**

logger**.**debug**(**"getBestPlaying10Users()"**);**

Query query **=** **new** Query**();**

query**.**addCriteria**(**Criteria**.**where**(**"numberOfWonChessGames"**).**ne**(null)**

**.**gt**(**0**));**

query**.**with**(new** Sort**(**Sort**.**Direction**.**DESC**,** "numberOfWonChessGames"**));**

query**.**limit**(**10**);**

**return** mongoTemplate**.**find**(**query**,** UserAccount**.**class**,** COLLECTION\_NAME**);**

**}**

public UserAccount getUserByUsername**(**String username**)** **{**

logger**.**debug**(**"getUserByUsername()"**);**

Query query **=** **new** Query**();**

query**.**addCriteria**(**Criteria**.**where**(**"username"**).**is**(**username**));**

UserAccount user **=** mongoTemplate**.**findOne**(**query**,** UserAccount**.**class**,**

COLLECTION\_NAME**);**

**return** user**;**

**}**

public UserAccount getUserByRegistrationHashString**(**

String registrationHashString**)** **{**

logger**.**debug**(**"getUserByRegistrationHashString()"**);**

Query query **=** **new** Query**();**

query**.**addCriteria**(**Criteria**.**where**(**"registrationHashString"**).**is**(**

registrationHashString**));**

UserAccount user **=** mongoTemplate**.**findOne**(**query**,** UserAccount**.**class**,**

COLLECTION\_NAME**);**

**return** user**;**

**}**

public UserAccount getUserById**(**long userId**)** **{**

logger**.**debug**(**"getUserById()"**);**

Query query **=** **new** Query**();**

query**.**addCriteria**(**Criteria**.**where**(**"userId"**).**is**(**userId**));**

UserAccount user **=** mongoTemplate**.**findOne**(**query**,** UserAccount**.**class**,**

COLLECTION\_NAME**);**

**return** user**;**

**}**

public void deleteUser**(**UserAccount user**)** **{**

logger**.**debug**(**"deleteUser()"**);**

mongoTemplate**.**remove**(**user**,** COLLECTION\_NAME**);**

**}**

// UPDATE

public void updateUser**(**UserAccount user**)** **{**

logger**.**debug**(**"updateUser()"**);**

Update updateUserData **=** **new** Update**();**

updateUserData**.**set**(**"name"**,** user**.**getName**());**

updateUserData**.**set**(**"lastname"**,** user**.**getLastname**());**

updateUserData**.**set**(**"role"**,** user**.**getRole**());**

updateUserData**.**set**(**"email"**,** user**.**getEmail**());**

updateUserData**.**set**(**"isRegistrationConfirmed"**,**

user**.**getIsRegistrationConfirmed**());**

updateUserData**.**set**(**"registrationDate"**,** user**.**getRegistrationDate**());**

updateUserData**.**set**(**"registrationHashString"**,**

user**.**getRegistrationHashString**());**

**if** **(**user**.**getNumberOfGamesPlayed**()** **!=** **null)** **{**

updateUserData**.**set**(**"numberOfGamesPlayed"**,**

user**.**getNumberOfGamesPlayed**());**

**}**

**if** **(**user**.**getNumberOfWonChessGames**()** **!=** **null)** **{**

updateUserData**.**set**(**"numberOfWonChessGames"**,**

user**.**getNumberOfWonChessGames**());**

**}**

**if** **(**user**.**getNumberOfLostChessGames**()** **!=** **null)** **{**

updateUserData**.**set**(**"numberOfLostChessGames"**,**

user**.**getNumberOfLostChessGames**());**

**}**

**if** **(**user**.**getPassword**()** **!=** **null**

**&&** **!**user**.**getPassword**().**equalsIgnoreCase**(**""**))** **{**

updateUserData**.**set**(**"password"**,** user**.**getPassword**());**

**}**

Query query **=** **new** Query**();**

query**.**addCriteria**(**Criteria**.**where**(**"username"**).**is**(**user**.**getUsername**()));**

mongoTemplate**.**updateFirst**(**query**,** updateUserData**,** COLLECTION\_NAME**);**

**}**

**}**

# Realizacja systemu

## 5.1 Tworzenie konta użytkownika

Aplikacja omawiana w niniejszej pracy jest aplikacją webową. Może być dostępna dla każdego użytkownika podłączonego do sieci internetowej. Część zasobów jest dostępna dla niezarejestrowanych użytkowników. Kluczowa funkcjonalność –gra z innymi użytkownikami online jest dostępna po utworzeniu konta. Konto użytkownika może utworzyć administrator w panelu administratora. Jak również konto użytkownika może utworzyć każdy użytkownik sieci internetowej za pomocą formularza do rejestracji. Jednak aby mieć dostęp do jej zasobów musi podać istniejący adres email, na który zostaje wysłany link aktywacyjny. Proces rejestracji od strony serwera realizuje klasa **SignInController**. Użytkownik w formularzu musi podać login, hasło, powtórzyć hasło, adres emial. Hasła są poddawane procesowi hashowania za pomocą klasy **PasswordEncryptor**. Login musi być unikalny dla systemu, za jego pomocą użytkownik będzie identyfikowany. Jeżeli użytkownik wprowadzi do formularza istniejący już w systemie login i kliknie utwórz, to w kontrolerze ta sytuacja zostanie przechwycona i zostanie wyświetlony komunikat walidacyjny w rodzaju: *taki login już istnieje proszę utworzyć inny*. Wysyłanie do użytkowników maila z linkiem aktywacyjnym realizuje klasa **MailService**. Dla każdego użytkownika link aktywacyjny jest unikatowy, jest on generowany za pomocą predefiniowanej klasy javowej **UUID**. Użytkownik ma tydzień czasu na aktywacje linka. Jeżeli konto nie jest aktywowane to jego dane są usuwane z bazy. Tą funkcjonalność realizuje klasa **ScheduledTaskService**, która korzysta z anotacji @Scheduled dsotępnej w Framework Spring MVC, umożliwiającej wykonywanie żądań w określonych odstępach czasowych.

Użytkownicy są przechowywani w bazie danych w kolekcji sers. Reprezentacją obiektową konta zarejestrowanego użytkownika zapisanego do bazy jest klasa **UserAccount**.

## 5.2 Autentykacja i uwierzytelnianie

Logowanie do serwisu jest możliwe jeżeli użytkownik ma potwierdzone konto. Bezpieczeństwo logowania zapewnia Framework Spring security, który jest częścią Spring MVC. Implementacja bezpieczeństwa przy użyciu tego Framework polega na zdefiniowaniu ról użytkowników

i zdefiniowaniu jakie ścieżki url są dostępne dla danego użytkownika oraz za pomocą tagów xml można zdefiniować które zasoby mają być wyświetlane dla użytkownika o danej roli.

Przykłady spring security w xml:

**Dostępność całej strony z pozycji url:**

Przykład z pliku **security-config.xml:**

<http auto-config=**"true"** disable-url-rewriting=**"false"**>

<headers disabled=**"true"** />

<csrf disabled=**"true"** />

<intercept-url pattern=**"/admin/\*\*"** access=**"hasRole('ROLE\_ADMIN')"** />

<intercept-url pattern=**"/user/\*\*"** access=**"hasRole('ROLE\_USER')"** />

<form-login login-page=**"/"** authentication-failure-url=**"/fail/"**

default-target-url=**"/"** />

<logout logout-success-url=**"/"** />

<form-login login-page=**"/login"** />

<logout logout-url=**"/j\_spring\_security\_logout"** />

<session-management>

<concurrency-control max-sessions=**"1"**

session-registry-alias=**"sessionRegistry"** />

</session-management>

</http>

<beans:bean id=**"userModel"** class=**"com.chessApp.model.UserAccount"**></beans:bean>

<!-- import resources/ beans from servlet config file -->

<beans:import resource=**"./mvc-config.xml"** />

<beans:import resource=**"./application-context.xml"** />

<beans:bean id=**"authenticationProvider"**

class=**"com.chessApp.security.LocalAuthenticationProvider"**></beans:bean>

<beans:bean id=**"usersRepositoryImpl"** class=**"com.chessApp.daoimpl.UsersRepositoryImpl"**></beans:bean>

<beans:bean id=**"passwordEncoder"**

class=**"org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"** />

<authentication-manager alias=**"authenticationManager"**>

<authentication-provider ref=**"authenticationProvider"** />

</authentication-manager>

<interceptor> to tag definujący jaki użytkownik może mieć dostęp do danego adresu url.

Atrybut patter reprezentuje url, atrybut Access przyjmuje funkcje hasRole(), która zwraca boolena czy osoba odwiedzająca dany url ma taką role przyznaną. Tag <form-login> przyjmuje atrybut login-page, który definiuje link akcji logowania. Link ten musi być przywołany w formularzu logowania w atrybucie action Tagu <form>. Tag <authentication-manager> odwołuje się do menadżera autentykacji, który oddeleguje akcje logowania do providera klasy **LocalAuthenticationProvider**, która jest zaimplementowana w javie i zajmuje się procesem autentykacji.

Klasa LocalAuthenticationProvider:

@Component

public class LocalAuthenticationProvider **extends**

AbstractUserDetailsAuthenticationProvider **{**

private final Logger logger **=** Logger

**.**getLogger**(**LocalAuthenticationProvider**.**class**);**

@Autowired

private UsersRepository usersRepository**;**

@Autowired

private PasswordEncoder encoder**;**

private PasswordEncryptor passwordEncrypter **=** **new** PasswordEncryptor**();**

@Override

protected void additionalAuthenticationChecks**(**UserDetails userDetails**,**

UsernamePasswordAuthenticationToken authentication**)**

**throws** AuthenticationException **{**

**}**

@Override

public UserDetails retrieveUser**(**String username**,**

UsernamePasswordAuthenticationToken authentication**)**

**throws** AuthenticationException **{**

String password **=** **(**String**)** authentication**.**getCredentials**();**

String hashPassword **=** **null;**

**try** **{**

hashPassword **=** passwordEncrypter**.**encryptUserPassword**(**password**);**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

**}**

**if** **(!**StringUtils**.**hasText**(**password**))** **{**

logger**.**warn**(**username **+** ": no password provided"**);**

**throw** **new** BadCredentialsException**(**"Please enter password"**);**

**}**

**if** **(!**StringUtils**.**hasText**(**username**))** **{**

logger**.**warn**(**username **+** ": no login provided"**);**

**throw** **new** BadCredentialsException**(**"Please enter login"**);**

**}**

UserAccount user **=** usersRepository**.**getUserByUsername**(**username**);**

**if** **(**user **==** **null)** **{**

logger**.**warn**(**username **+** ": user not found"**);**

**throw** **new** UsernameNotFoundException**(**"Invalid Login"**);**

**}**

**if** **(**user**.**getIsRegistrationConfirmed**()** **==** **false)** **{**

logger**.**warn**(**username **+** ": not confirmed"**);**

**throw** **new** UserNotConfirmedException**(**username **+** ": not confirmed"**);**

**}**

**if** **(**username**.**equals**(**user**.**getUsername**())**

**&&** hashPassword**.**equals**(**user**.**getPassword**())**

**&&** hashPassword **!=** **null)** **{**

final List**<**GrantedAuthority**>** auths **=** UserRoles**.**getUserRoles**(**user

**.**getRole**());**

**return** **new** User**(**user**.**getUsername**(),** user**.**getPassword**(),** **true,** // enabled

**true,** // account not expired

**true,** // credentials not expired

**true,** // account not locked

auths**);**

**}** **else** **{**

**throw** **new** UsernameNotFoundException**(**"Invalid Login and password"**);**

**}**

**}**

**}**

**Dostępność zasobów na stronie:**

Przykład z pliku **menu.jsp:**

<%@ page import="com.chessApp.props.ChessAppProperties"%>

<%

String contextURL = ChessAppProperties

.getProperty("app.contextpath");

%>

<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<%@ taglib uri="http://www.springframework.org/security/tags"

prefix="security"%>

<ul id=**"navButtons"** class=**"nav nav-tabs navView"**>

<li><a href=**"<%=**contextURL**%>/"**>**home**</a></li>

<li><a href=**"<%=**contextURL**%>/play-chess-with-computer"**>**play-with-computer**</a></li>

<li><a href=**"<%=**contextURL**%>/play-chess-with-user"**>**play-with-others**</a></li>

<security:authorize access="hasRole('ROLE\_ADMIN')">

<li><a href=**"<%=**contextURL**%>/admin/users"**>**all-users**</a></li>

</security:authorize>

<security:authorize access="hasRole('ROLE\_ADMIN')" var="isAdmin" />

<c:choose>

<c:when test="${isAdmin}">

<li><a href=**"<%=**contextURL**%>/admin/your-account"**>**your-profile**</a></li>

</c:when>

<c:otherwise>

<security:authorize access="hasRole('ROLE\_USER')">

<li><a href=**"<%=**contextURL**%>/user/your-account"**>**your-profile**</a></li>

</security:authorize>

</c:otherwise>

</c:choose>

<security:authorize access="hasRole('ROLE\_USER')">

<li><a href=**"<%=**contextURL**%>/user/your-chessgames"**>**your-games-history**</a></li>

</security:authorize>

<li><a href=**"<%=**contextURL**%>/home/best-players"**>**best-players**</a></li>

<li class=**"pull-right logOutBtn text-capitalize"**><a

href=**"<%=**contextURL**%>/logout"**>**log out**</a></li>

<li class=**"pull-right logInBtn text-capitalize"**><a

href=**"<%=**contextURL**%>/login"**>**log in**</a></li>

</ul>

<div class=**"welcome-user-line"**>

<security:authorize access="hasAnyRole('ROLE\_ADMIN, ROLE\_USER')">

<h4 class=**"text-right logged-user-name"**>

**Welcome:** <span class=**"text-primary"**><b>**${currentUserName}**</b></span>

</h4>

</security:authorize>

</div>

Tag <security:authorize> okresla jakie zasoby w pliku jsp mogą być zaprezentowane dla określonego uzytkownika. Plik jsp to pod spodem Java która buduje plik html według określonych instrukcji i widoków zdefiniowanych w pliku jsp za pomocą tagów xml. W Spring MVC są to najczęściej używane tagi xml z biblioteki jstl jak i spring security. Na początki zacytowanego kodu widać instrukcje <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<%@ taglib uri=<http://www.springframework.org/security/tags> prefix="security"%>

Które importują określone przestrzenie nazw dla tagów xml. Tag <security:authorize> przyjmuje atrybut acces, który przyjmuje funkcje hasRole() zwracającą wartość boole’owską w zależności od tego jakie uprawnienia ma użytkownik odiwedzajacy strone.

## 5.3 Implementacja gry w szachy.

Interfejs do gry w szachy oraz logika do gry jest napisana po stronie klienta w języku javascript. Jest to realizowane przy pomocy biblioteki javascript: **chessboard.js** (interfejs)

Link: <http://chessboardjs.com/>

**chess.js** (silnik gry)

link: <https://github.com/jhlywa/chess.js/blob/master/README.md>

Pliki źródłowe z tymi bibliotekami są obecne w pakiecie w katalogu:

**/src/main/webapp/assets/js/lib/**

W katalogu **/src/main/webapp/assets/js/** umieszczone są pliki źródłowe z implementacji zdarzeniowego api udostępnionego przez powyższe biblioteki.

Są to pliki:

**chessComputerVsComputer.js –** implementacja funkcjonalności komputer versus komputer, widocznej na stronie domowej (home) jako warstwa prezentacyjna aplikacji.

**chessUserVsComputer.js –** implementacja funkcjonalności użytkownik versus komputer, widocznej na podstronie (play-with-computer) jako warstwa demonstracyjna aplikacji.

**chessUserVsUser.js –** implementacja funkcjonalności użytkownik versus komputer, widocznej na podstronie (play-with-users) jako warstwa prezentująca główną funkcjonalność aplikacji.

**Kod z pliku chessUserVsUser.js:**

***var*** startFENPosition **=** "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"**;**

// start new game

$**(**'#startPosBtn'**).**click**(*function*()** **{**

startNewGame**();**

**});**

***function*** startNewGame**()** **{**

board**.**position**(**startFENPosition**);**

game **=** ***new*** Chess**(**startFENPosition**);**

updateStatus**();**

**}**

// chess script ---------------------------------

***var*** board**,** game **=** ***new*** Chess**(),** statusEl **=** $**(**'#status'**),** fenEl **=** $**(**'#fen'**),** pgnEl **=** $**(**'#pgn'**);**

***var*** onDragStart **=** ***function*(**source**,** piece**,** position**,** orientation**)** **{**

console**.**log**(**"onDragStart()"**);**

console**.**log**(**'SENDED\_CHESS\_MOVE\_STATUS: ' **+** SENDED\_CHESS\_MOVE\_STATUS**);**

// if move does not belong to you

// pieces are blocked

// inviting user at start have "white piece"

// at start status is "White to move"

***if*** **(**statusEl**.**text**().**trim**()** **!=** SENDED\_CHESS\_MOVE\_STATUS**)** **{**

***return*** ***false*;**

**}**

// allow only one move

***if*** **(**CHESS\_MOVE\_COUNTER **>** 0**)** **{**

***return*** ***false*;**

**}**

// do not pick up pieces if the game is over

// only pick up pieces for the side to move

***if*** **(**game**.**game\_over**()** **===** ***true***

**||** **(**game**.**turn**()** **===** 'w' **&&** piece**.**search**(**/^b/**)** **!==** **-**1**)**

**||** **(**game**.**turn**()** **===** 'b' **&&** piece**.**search**(**/^w/**)** **!==** **-**1**))** **{**

***return*** ***false*;**

**}**

**};**

***var*** onDrop **=** ***function*(**source**,** target**)** **{**

console**.**log**(**"onDrop()"**);**

// see if the move is legal

***var*** move **=** game**.**move**({**

***from*** **:** source**,**

to **:** target**,**

promotion **:** 'q'

// simplicity

**});**

// illegal move

***if*** **(**move **===** null**)**

***return*** 'snapback'**;**

CHESS\_MOVE\_COUNTER**++;**

CURRENT\_CHESS\_MOVE **=** move**;**

updateStatus**();**

**};**

***var*** onSnapEnd **=** ***function*()** **{**

board**.**position**(**game**.**fen**());**

**};**

***var*** updateStatus **=** ***function*()** **{**

***var*** status **=** ''**;**

***var*** moveColor **=** 'White'**;**

***if*** **(**game**.**turn**()** **===** 'b'**)** **{**

moveColor **=** 'Black'**;**

**}**

// checkmate?

***if*** **(**game**.**in\_checkmate**()** **===** ***true*)** **{**

status **=** 'Game over, ' **+** moveColor **+** ' is in checkmate.'**;**

***var*** winnerColor **=** ""**;**

***var*** looserColor **=** ""**;**

***if*** **(**moveColor **==** 'White'**)** **{**

winnerColor **=** 'black'**;**

winnerUsername **=** BLACK\_COLOR\_USERNAME**;**

loserUsername **=** WHITE\_COLOR\_USERNAME**;**

**}** ***else*** **{**

winnerColor **=** 'white'**;**

winnerUsername **=** WHITE\_COLOR\_USERNAME**;**

loserUsername **=** BLACK\_COLOR\_USERNAME**;**

**}**

// only winner send message, to prevent duplicates

// browser client that create game.in\_checkmate() is winner

// by default

***if*** **(**WEBSOCKET\_CLIENT\_NAME **==** winnerUsername**)** **{**

***var*** fenString **=** fenFromYourMove**.**value**;**

webSocket**.**send**(**JSON**.**stringify**({**

type **:** "game-over"**,**

fen **:** fenString**,**

winnerColor **:** winnerColor**,**

winnerUsername **:** winnerUsername**,**

checkMate **:** ***true*,**

loserUsername **:** loserUsername**,**

sendFrom **:** WEBSOCKET\_CLIENT\_NAME**,**

sendTo **:** $**(**'#quit-game-btn'**).**data**(**"gamePartner"**)**

**}));**

**}**

$**(**'#move-for'**).**html**(**"<h1 class=\"text-success\">YOU WIN !</h1>"**);**

alert**(**"check mate, you Win with " **+** OPPONENT\_USERNAME **+** " !"**);**

$**(**'#startPosBtn'**).**show**();**

$**(**'#game-status'**).**data**(**'isPlaying'**,** ***false*);**

$**(**'#game-status'**).**html**(**''**);**

$**(**'#send-move-btn'**).**data**(**"opponentName"**,** ''**);**

$**(**'#quit-game-btn'**).**data**(**"gamePartner"**,** ''**);**

$**(**'#play-with-opponent-interface-actions'**).**attr**(**"hidden"**,** ***true*);**

OPPONENT\_USERNAME **=** ""**;**

clearParticipantsListView**();**

**}**

// draw?

else if (game.in\_draw() === true) {

status = 'Game over, drawn position';

}

// game still on

else {

status = moveColor + ' to move';

// check?

if (game.in\_check() === true) {

status += ', ' + moveColor + ' is in check';

}

}

statusEl.html(status);

fenEl.html(game.fen());

$('#fenFromYourMove').val(game.fen());

pgnEl.html(game.pgn());

};

var cfg = {

pieceTheme : '/resources/images/chesspieces/wikipedia/{piece}.png',

draggable : true,

position : 'start',

onDragStart : onDragStart,

onDrop : onDrop,

onSnapEnd : onSnapEnd

};

board = ChessBoard('board', cfg);

updateStatus();

W powyższym przykładzie widać zdarzeniową obsługę gry w szachy przy użyciu zdarzeń drag and drop. Podczas zdarzenia onDragStart sprawdzamy czy możemy wykonać ruch, czy ruch należy do na s, czy do przeciwnika. Przy zdarzeniu onDrop, jest tworzony obiekt move i aktualizowany status gry poprzez funkcję updateStatus(). Zdarzenie onSnapEnd buduje planszę na nowo po wykonanym ruchu za pomocą funkcji position() wykonywanej na obiekcie game:

**board.position(game.fen());**

Z kolei funkcja fen() to aktualny stan gry według notacji fen w postaci stringa.

UpdateStatus wywolany w zdarzeniu onDrop aktualizuje obiekty przechowywujace aktualny status gry w postaci ciągu znaków fen i jeżeli to szach mat to wysyłana jest o tym wiadomość do serwera w postaci obiektu GameMessage za pomocą Websocket:

***var*** fenString **=** fenFromYourMove**.**value**;**

webSocket**.**send**(**JSON**.**stringify**({**

type **:** "game-over"**,**

fen **:** fenString**,**

winnerColor **:** winnerColor**,**

winnerUsername **:** winnerUsername**,**

checkMate **:** ***true*,**

loserUsername **:** loserUsername**,**

sendFrom **:** WEBSOCKET\_CLIENT\_NAME**,**

sendTo **:** $**(**'#quit-game-btn'**).**data**(**"gamePartner"**)**

**}));**

Na podstawie danych z tej wiadomości i danych przechowywanych na serwerze dokonuje się zapis gry do bazy danych;

## 5.4 Gra w szachy użytkownik versus użytkownik implementacja komunikatora do gry.

Funkcjonalność gry online była głównym tematem zainteresowań niniejszego projektu i jednym z najtrudniejszych zadań w procesie implementacji oprogramowania.

Możliwość gry online wymagała zaprogramowania komunikatora działającego w czasie rzeczywistym, bez konieczności przeładowywania, czy odświeżania strony po zmianie statusu gry.

Do tego została użyta technologia technologii Websocket. Klasy implementujące komunikator znajdują się w pakiectach: **com.chessApp.websocket** i **com.chessApp.** **gameProtocol**.Implementacja komunikatora opiera się na obsłudze zdarzeń po stronie serwera w postaci **Server endpoint (zaprogramowanego w Javie) i** po stronie klienta w postaci Client endpoint (**zaprogramowanego w JavaScript**), oraz przechowywaniu wybranych danych w pamięci serwera w kontekście połączenia wybranego użytkownika do Server endpoint.

Server endpoint jest zaimplementowany w pliku **WebSocketServer.java** w pakiecie **com.chessApp.websocket.** Client ednpoint w pliku **websocketClientEndpoint.js** obecnym

w katalogu pod ścieżką **/src/main/webapp/assets/js/**

Fragmenty kodu:

**WebSocketServer.java:**

@Service

@ServerEndpoint**(**"/chessapp-live-game/{sender}"**)**

public class WebSocketServer **{**

private final static Logger log **=** Logger**.**getLogger**(**WebSocketServer**.**class**);**

private final WebSocketSessionHandler sessionHandler **=** **new** WebSocketSessionHandler**();**

private final GameUsersHandler usesrHandler **=** **new** GameUsersHandler**();**

private final ChessGamesHandler chessGamesHandler = new ChessGamesHandler();

private GameMessageProtocol gameMessageProtocol = new GameMessageProtocol(

sessionHandler, usesrHandler, chessGamesHandler);

private Gson gson = new Gson();

@OnMessage

public void onMessage(String msg, Session wsSession,

@PathParam("sender") String sender) throws IOException {

log.info("wiadomość odebrana przez server: ");

GameMessage message = gson.fromJson(msg, GameMessage.class);

gameMessageProtocol.proccessMessage(message, msg);

}

@OnOpen

public void onOpen(Session wsSession, EndpointConfig config,

@PathParam("sender") String sender) {

log.info("connection started, websocket session id: "

+ wsSession.getId() + " " + sender + " open connection ");

if (usesrHandler.userListNotContainsUsername(sender)) {

GameUser gameUser = new GameUser(sender);

gameUser.setCommunicationStatus(GameUserCommunicationStatus.WAIT\_FOR\_NEW\_GAME);

synchronized (this) {

wsSession.getUserProperties().put("sessionOwner",

gameUser.getUsername());

sessionHandler.addSession(gameUser.getUsername(), wsSession);

usesrHandler.addWebsocketUser(gameUser);

}

sessionHandler.sendToAllConnectedSessions(gameUser.getUsername());

}

}

@OnClose

public void onClose(CloseReason closeReason, Session wsSession,

@PathParam("sender") String sender) {

log.info("connection closed. Reason: " + closeReason.getReasonPhrase());

log.info(sender);

synchronized (this) {

GameUser cloesingConnectionUser = usesrHandler

.getWebsocketUser(sender);

if (cloesingConnectionUser.getPlayNowWithUser() != null

&& cloesingConnectionUser.getPlayNowWithUser() != "") {

GameUser cloesingConnectionUserGamePartner = usesrHandler

.getWebsocketUser(cloesingConnectionUser

.getPlayNowWithUser());

cloesingConnectionUserGamePartner.setPlayNowWithUser(null);

cloesingConnectionUserGamePartner

.setCommunicationStatus(GameUserCommunicationStatus.WAIT\_FOR\_NEW\_GAME);

GameMessage disconnectMsg = new GameMessage();

disconnectMsg.setType(GameMessageType.USER\_DISCONNECT);

sessionHandler.sendToSession(

cloesingConnectionUserGamePartner.getUsername(),

sender, gson.toJson(disconnectMsg));

}

usesrHandler.removeWebsocketUser(sender);

sessionHandler.removeSession(sender);

}

sessionHandler.sendToAllConnectedSessionsActualParticipantList();

}

@OnError

public void onErrorReceived(Throwable t) {

log.debug("there was an error with connection");

log.debug(t);

}

}

WebsocketServer jest klasą obsługująca zdarzenia: **onOpen**, **onMessage**, **onClose** i **onError**. Na podstawie tych zdarzeń wywoływanych przez klienta aplikacji wywoływane są określone działania na serwerze.

Serwer w swojej pamięci przechowuje listę użytkowników podłączonych do repozytorium gry online. Reprezentacją połączenia przez webSocket jest sesja, każdy użytkownik ma swoją sesję, jest ona innym obiektem niż sesja logowania, która przechowuje kontekst użytkownika dla bezstanowego protokołu http, te sesje są w kontekście połączenia poprzez websocket.

Aby poszczególnych użytkowników dało się identyfikować w implementacji posłużono się kolekcjami dostępnymi w języku Java. Podstawową strukturą użytą do tego celu jest mapa. Została stworzona mapa: **webSocketSessions<String , Session>**, w której kluczem jest nazwa użytkownika, wartością sesja polaczeniem do którego można wysłać wiadomość przy zdarzeniu **onMessage**. Szczególy na temat użytkowników są przechowywanie w **mapie gameUsers<String, GameUser>.** Ta kolekcja przechowuje użytkowników gry w postaci obiektów, zawierających informacje potrzebne do tego aby prowadzić grę z innymi użytkownikami, kluczem do wartości tej kolekcji są także nazwy użytkowników. Przy tym rozwiązaniu zarówno sesja danego użytkownika jak i jego reprezentacja w postaci obiektu maja ten sam klucz dzięki temu pozostają w relacji *one to one*. Konto użytkownika zarejestrowanego w aplikacji reprezentuje obiekt UserAccount. W kontekście gry ten sam użytkownik jest reprezentowany obiektem GameUser reprezentującym jego kontekst w grze szachowej, który z kolei jest powiązany z obiektem Session reprezentującym jego połączenie z warstwą websocket. Wszystkie te obiekty są powiązane jednym kluczem, czyli nazwą użytkownika.

Serwer także przechowuje w swojej pamięci operacyjnej informacje na temat aktualnie toczących się gier w kolekcji **Map<String, ChessGame> chessGamesMap** Kluczem jest tutaj pseudolosowo generowany unikatowy *hash*, wartością obiekt gry. Użytkownicy uczestniczący w pojedynku przechowują w swoim obiekcie GameUser hash aktualnie toczonej przez siebie gry, za pomocą właściwości uniqueGameHash tworzą w ten sposób unikatową parę, zamkniętą w kontekście prowadzonej przez siebie gry.

Akcje związane z obsługą tych kolekcji zostały przeniesione do osobnych klas: **GameUsersHandler**, **ChessGamesHandler**, **WebSocketSessionsHandler**. Zastosowanie tych kolekcji w postaci map oraz obiektów wykonujących na nich operacje dało możliwość wysyłania wiadomości do wszystkich użytkowników w formie *broadcastu*, co jest zastosowane w momencie gdy ktoś nowy dołącza się do repozytorium gry:

public synchronized void sendToAllConnectedSessionsActualParticipantList**()** **{**

String jsonUsersList **=** gson**.**toJson**(**GameUsersHandler**.**gameUsersMap

**.**values**());**

**for** **(**String username **:** sessionsMap**.**keySet**())** **{**

Session userSession **=** sessionsMap**.**get**(**username**);**

**try** **{**

userSession**.**getBasicRemote**().**sendText**(**jsonUsersList**);**

**}** **catch** **(**IOException e**)** **{**

logger**.**info**(**e**);**

**}**

**}**

**}**

Jak i możliwość wysyłania wiadomości w komunikacji *jeden do jeden*, kiedy jest prowadzona gra w parach.

public void sendToSession**(**String toUsernameName**,** String fromUsername**,**

String message**)** **{**

logger**.**info**(**"sendToSession()"**);**

Session userSession **=** sessionsMap**.**get**(**toUsernameName**);**

**if** **(**userSession **!=** **null)** **{**

**try** **{**

userSession**.**getBasicRemote**().**sendText**(**message**);**

**}** **catch** **(**IOException e**)** **{**

logger**.**debug**(**e**);**

**}**

**}**

**}**

Kliencki endpoint obsługuje te same zdarzenia co Serwer endpoint. Jest on obecny w katalogu **/src/main/webapp/assets/js/** w pliku **websocketClientEndpoint.js**

Kod z pliku **websocketClientEndpoint.js**:

***var*** TIMEOUT\_FOR\_HANDSHAKE **=** 15**;**

***var*** CLICK\_REFUSED\_FLAG **=** ***false*;**

***var*** CLICK\_AGREEMENT\_FLAG **=** ***false*;**

// ------CONNECT TO WEBSOCKET FUNCTION, WEBSOCKET EVENTS----------------------

***function*** connectToWebSocket**()** **{**

console**.**log**(**'connectToWebSocket()'**);**

// ----init websocket -------------------------------------

***var*** endpointUrl **=** "ws://" **+** document**.**location**.**host **+** "/chessapp-live-game/"

**+** WEBSOCKET\_CLIENT\_NAME**;**

webSocket **=** ***new*** WebSocket**(**endpointUrl**);**

// websocketClient events -----------------------------

webSocket**.**onopen **=** ***function*(**event**)** **{**

console**.**log**(**"Server connected \n"**);**

console**.**log**(**event**.**data**);**

$**(**'#disconnect'**).**attr**(**"disabled"**,** ***false*);**

webSocket**.**send**(**JSON**.**stringify**({**

type **:** "welcome-msg"**,**

sendFrom **:** WEBSOCKET\_CLIENT\_NAME

**}));**

$('#connection-status').html(

"<div class=\"alert alert-success connection-status-msg\">"

+ "<h2>You are connected!</h2></div>");

var disconnectBtn = $('#disconnect');

var connectBtn = $('#connectToWebSocket');

if (disconnectBtn.attr("disabled", true)) {

disconnectBtn.removeAttr("class");

disconnectBtn.attr("disabled", false);

disconnectBtn.attr("class", "btn btn-danger pull-right");

}

connectBtn.removeAttr("class");

connectBtn.attr("disabled", true);

connectBtn.attr("class", "btn btn-default");

};

// -----------------------------------

webSocket.onmessage = function(event) {

console.log("onmessage: ");

clientMsgProtocol.proccessMessage(event);

};

// -----------------------------------

webSocket.onclose = function(event) {

$('#connection-status').html(

"<div class=\"alert alert-warning connection-status-msg\">"

+ "<h2>You are disconnected!</h2></div>");

var disconnectBtn = $('#disconnect');

var connectBtn = $('#connectToWebSocket');

if (connectBtn.attr("disabled", true)) {

connectBtn.removeAttr("class");

connectBtn.removeAttr("disabled");

connectBtn.attr("disabled", false);

connectBtn.attr("class", "btn btn-primary");

}

disconnectBtn.removeAttr("class");

disconnectBtn.attr("disabled", true);

disconnectBtn.attr("class", "btn btn-default pull-right");

$('#game-status').html('');

$('#participants div ul').html('');

$('#disconnect').attr("disabled", true);

$('#play-with-opponent-interface').attr("hidden", true);

$('#startPosBtn').show();

OPPONENT\_USERNAME = "";

console.log(event);

};

// -----------------------------------

webSocket.onerror = function(event) {

webSocket.send("error: client disconnected");

console.log("Server disconnected \n");

console.log(event);

webSocket.close();

$('#startPosBtn').show();

$('#disconnect').attr("disabled", true);

$('#play-with-opponent-interface').attr("hidden", true);

OPPONENT\_USERNAME = "";

};

};

// close websocket when page reload --------------------------------------------

window.onbeforeunload = function() {

webSocket.onclose = function() {

webSocket.send("client disconnected");

console.log("Server disconnected \n");

}; // disable onclose handler first

webSocket.close();

$('#connectToWebSocket').attr("disabled", false);

OPPONENT\_USERNAME = "";

window.location.reload(false);

};

Jak widać powyżej są tu obsługiwane te same zdarzenia co na serwerze: **onOpen**, **onMessage**, **onClose** i **onError**.

Stan gry i ruchy poszczególnych użytkowników są przekazywane między nimi za pomocą notacji *fen*. **FEN Jest to tak zwana notacja Forsytha-Edwardsa.** Polega ona na zapisaniu położenia wszystkich figur I do kogo należy następny ruch w postaci ciągu znaków. Przykład notacji fen:

**„r1bqkbnr/pppp1ppp/2n5/1B2p3/4P3/5N2/PPPP1PPP/RNBQK2R”.**

Jedną z większych trudności przy implementacji komunikatora do gry było zapewnienie parom użytkowników zamkniętej konwersacji, tak aby ich gra była bezpieczna od wtargnięcia w nią innych użytkowników. Zostało to zaimplementowane za pomocą protokołu komunikacji przy zdarzeniu **onmessage**, który to w zależności od rodzaju wiadomości na podstawie instrukcji warunkowych wykonuje określone czynności. Wiadomość wysyłaną przez użytkowników na serwer reprezentuje Obiekt **GameMessage**. Obiekt ten ma właściwość **private String type**, dzięki której możliwe było zidentyfikowanie jakiego rodzaju wiadomość jest wysyłana aktualnie na serwer i co przy określonym typie wiadomości zrobić. Oprócz tego obiekt wiadomość przechowuje informacje o tym kto ją wysyła i do kogo oraz w zależności od potrzeby dodatkowe informacje na temat toczonego pojedynku szachowego.

Fragment kodu klasy GmaeMessage:

public class GameMessage **{**

private String type**;**

private String fen**;**

private String sendTo**;**

private String sendFrom**;**

private String moveStatus**;**

private GameUser sendToObj**;**

private GameUser sendFromObj**;**

private Boolean checkMate**;**

private ChessMove chessMove**;**

private String whiteColUsername**;**

private String blackColUsername**;**

public GameMessage**()** **{**

**}**

Typy wiadomości zostały zdefiniowane w klasie **GameMessageType** przechowującej pola statyczne w postaci stringów reprezentujące określone typy wiadomości.

public class GameMessageType **{**

public final static String USER\_CONNECT **=** "welcome-msg"**;**

public final static String GAME\_HANDSHAKE\_INVITATION **=** "game-handshake-invitation"**;**

public final static String GAME\_HANDSHAKE\_AGREEMENT **=** "game-handshake-agreement"**;**

public final static String GAME\_HANDSHAKE\_REFUSE **=** "game-handshake-refuse"**;**

public final static String CHESS\_MOVE **=** "chess-move"**;**

public final static String GAME\_OVER **=** "game-over"**;**

public final static String QUIT\_GAME **=** "quit-game"**;**

public final static String USER\_DISCONNECT **=** "goodbye-msg"**;**

public final static String TRY\_LATER **=** "try-later"**;**

**}**

Gdy dany użytkownik chce zagrać z drugim użytkownikiem, to musi wysłać mu zaproszenie GAME\_HANDSHAKE\_INVITATION, drugi Mozę się zgodzić wysyłając: GAME\_HANDSHAKE\_AGREEMENT lub nie, wysyłając: GAME\_HANDSHAKE\_REFUSE. Jeżeli jest zgoda drugiego użytkownika to obiekty tych użytkowników przyjmują status IS-PLAYING. Typy statusów użytkowników są przechowywane w klasie **GameUserCommunicationStatus** również w postaci pól statycznych:

public class GameUserCommunicationStatus **{**

public final static String WAIT\_FOR\_NEW\_GAME **=** "wait-for-new-game"**;**

public final static String IS\_DURING\_HANDSHAKE **=** "during-handshake"**;**

public final static String IS\_PLAYING **=** "is-playing"**;**

**}**

Gdy użytkownik ma status WAIT\_FOR\_NEW\_GAME to każdy może się z nim skontaktować i on może z każdym się kontaktować. Gdy ma status IS\_DURING\_HANDSHAKE, może skontaktować sie tylko z użytkownikiem, którego zaprosił do gry bądź został przez niego zaproszony. Gdy ma status IS\_PLAYING, może wysyłać wiadomości tylko do użytkownika, z którym gra i odbierać wiadomości tylko od użytkownika, z którym gra, co jest bardzo ważne i gwarantuje bezpieczeństwo gry .

Wracając do typów wiadomości, zostaną omówione pozostałe typy. Ruch szachowy to wiadomość typu CHESS\_MOVE, koniec gry to GAME\_OVER lub QUIT\_GAME, wtedy automatycznie użytkownicy zmieniają status na WAIT\_FOR\_NEW\_GAME. Status USER\_CONNECT reprezentuje stan : gdy nowy użytkownik przyłącza sie do gry, wtedy do wszystkich użytkowników wysyłana jest aktualna lista użytkowników. Podobnie dzieje się przy USER\_DISCONNECT: użytkownik zakończył połączenie, do wszystkich wysyłana jest aktualna lista użytkowników. TRY\_LATER gdy użytkownik, z którym chcemy się skontaktować ma status IS\_DURING\_HANDSHAKE albo IS\_PLAYING, przy czym gdy użytkownik jest w jednym ze stanów IS\_DURING\_HANDSHAKE lub IS\_PLAYING to dla pozostałych użytkowników poza swoją para do gry są zablokowane akcje w interfejsie użytkownika umożliwiające kontakt z tym z daną jednostką.

**Opis przebiegu gry z perspektywy komunikacji między dwoma użytkownikami:**

Użytkownik po zalogowaniu do serwisu może połączyć się do repozytorium gry online poprzez podstronie play-with-users. Nowo polaczony użytkownik ma status WAIT\_FOR\_NEW\_GAME. Uzytkownik1 zaprasza Użytkownika2, po tym zdarzeniu u obu użytkowników zmienia się status na IS\_DURING\_HANDSHAKE. Jezeli Użytkownika2 się zgadza rozpoczynaja gre I oboje zmieniają status na IS\_PLAYING. Dopuki, któryś z użytkownikóœ nie zrezygnuje, lub nie nastąpi zdarzenie szach mat, toczy się gra, I oboje użytkowników dalej mają status IS\_PLAYING. Jeżeli jeden z użytkowników wygrywa lub rezygnuje, gra się kończy I oboje użytkowników zmieniaja status na WAIT\_FOR\_NEW\_GAME. Jeżeli Użytkownika2 odmówił po zaproszeniu oboje wracają do statusu początkowego WAIT\_FOR\_NEW\_GAME Iani proces gry ani status IS\_PLAYING nie zaistniał.

Pseudokod stworzony podczas projektowania:

Start**;**

GameUser user1 **=** **new** GameUser**(**WAIT\_FOR\_NEW\_GAME**);**

GameUser user2 **=** **new** GameUser**(**WAIT\_FOR\_NEW\_GAME**);**

user1**.**connect**();**

user2**.**connect**();**

user1**.**invite**(**user2**);**

user1**.**gameStatus **=** IS\_DURING\_HANDSHAKE**;**

user2**.**gameStatus **=** IS\_DURING\_HANDSHAKE**;**

**if** **(**user2**.**agreeToPlayWith**(**user1**))** **{**

user1**.**gameStatus **=** IS\_PLAYING**;**

user2**.**gameStatus **=** IS\_PLAYING**;**

**}** **else** **{**

user1**.**gameStatus **=** WAIT\_FOR\_NEW\_GAME**;**

user2**.**gameStatus **=** WAIT\_FOR\_NEW\_GAME**;**

**}**

**while(**user1**.**gameStatus **=** IS\_PLAYING **&&** user2**.**gameStatus **=** IS\_PLAYING**)** **{**

**if(**user1**.**gameOver**()** **||** user2**.**gameOVER**()** **||** checkMate**())** **{**

user1**.**gameStatus **=** WAIT\_FOR\_NEW\_GAME**;**

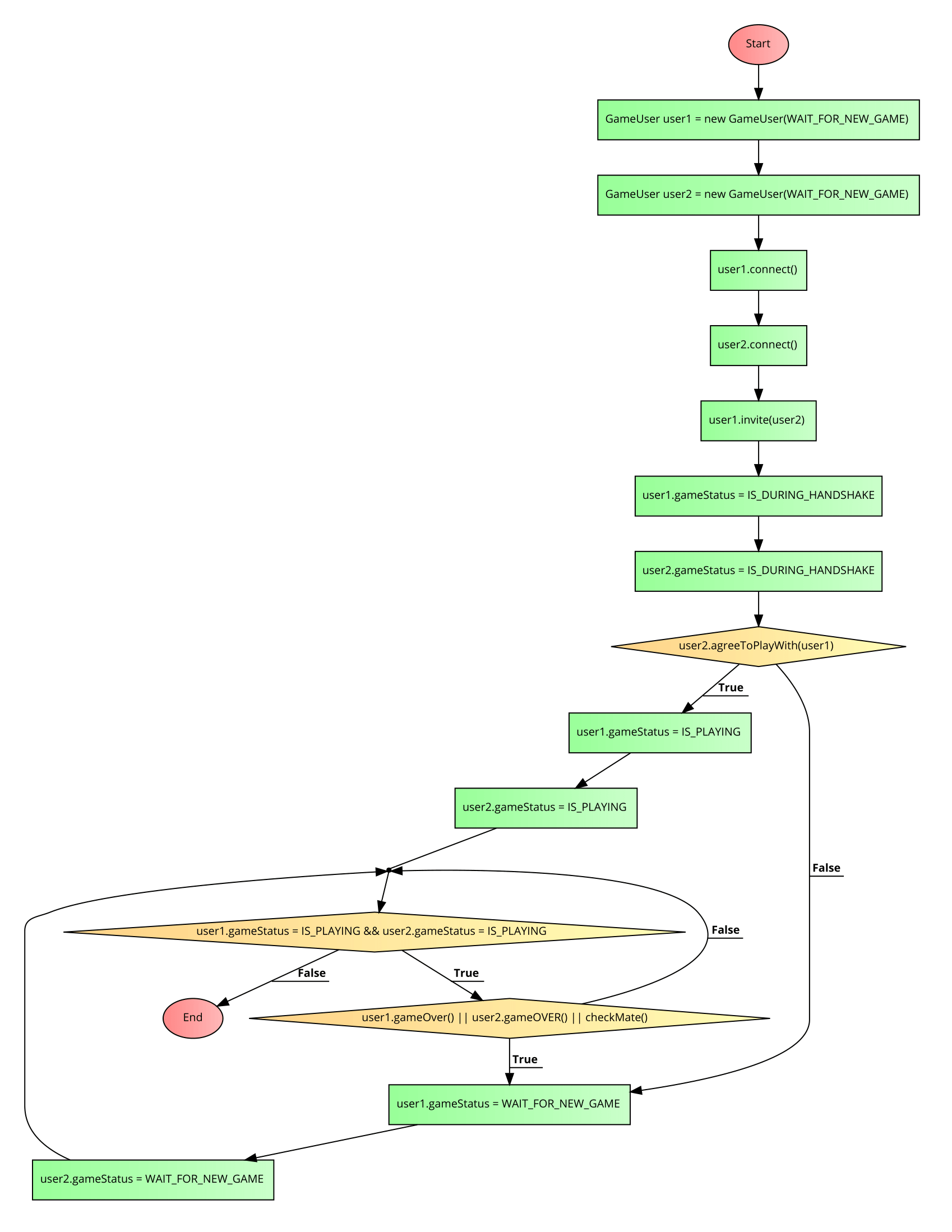
user2**.**gameStatus **=** WAIT\_FOR\_NEW\_GAME**;**

**}**

**}**

End**;**

Schemat blokowy komunikacji graczy stworzony przy użyciu narzędzia: <http://code2flow.com/>



5 Schemat blokowy komunikacja graczy

# Testy

# Podsumowanie

# Bibliografia

1. Java core
2. Thinking In Java
3. Po prostu javascript
4. Spring
5. Algorytmy cormen

# Spis ilustracji

# Spis tabel

# Załączniki