KillerChess

Antyszachy

Testowanie aplikacji zgodnie z wymaganiami

Użyte technologie w projekcie

Uruchomienie projektu

Spis treści

[**Wymagania, testy systemowe**](#_f1t13y7s11ho) **3**

[Historia gry](#_3vr62q2k0utg) 3

[Ranking](#_n7vwrwyd3rdo) 4

[Dźwięki w grze](#_f78mchbq3jmx) 5

[Podświetlane pola](#_2aewpr36rfk7) 6

[Szablony graficzne figur](#_v2771w7brlpb) 7

[Schematy gier](#_2ux790u9j00l) 8

[Pokoje gry](#_3rj5028f3i0c) 9

[Pomocnik użytkownika](#_t7gjo2avuyw9) 10

[Promocja pionka](#_mqaglhcbr387) 11

[Konto użytkownika](#_crbrp0mbldur) 12

[**Użyte technologie**](#_u6w6junf0n3) **13**

[**Testy jednostkowe i integracyjne**](#_1k7dt4c5gl3a) **15**

[**Opis kompilacji i uruchomienia projektu**](#_tg4019vx0zkl) **16**

[Kod źródłowy](#_a3p1d6en0nd) 16

[Kompilacja](#_gci7yg4j1m6) 16

[Uruchomienie](#_xl36qs8aju0z) 17

##### 

##### Wymagania, testy systemowe

###### Historia gry

W czasie rozgrywki gracze mają dostęp do podejrzenia wcześniejszych ruchów bez faktycznego powodowania zmian w rozgrywce. Po kliknięciu w odpowiedni przycisk, figury ustawią się tak, jak w poprzednim ruchu. Po kolejnych kliknięciach gracz będzie się cofał tak aż do początku.

|  |  |
| --- | --- |
| Opis testu | Wynik |
| Po kliknięciu w przycisk historii gry zmienia się układ na planszy i pokazywane jest wcześniejsze ustawienie | POZYTYWNY |
| Po kliknięciu w przycisk historii gry pojawia się drugi przycisk, którym można wrócić do aktualnego ustawienia | POZYTYWNY |
| Po kliknięciu w przycisk wracający do aktualnego ustawienia, pojawia się aktualna plansza | POZYTYWNY |
| Po dojściu do pierwszego ruchu nie można się cofnąć jeszcze bardziej | POZYTYWNY |

###### 

###### Ranking

Będzie on dostępny na głównym ekranie jako zakładka - lista najlepszych graczy z największą liczbą punktów. Podliczanie punktacji wygląda następująco: każda bierka ma swoją liczbę punktów - zwycięski gracz otrzymuje doliczoną liczbę punktów na swoje konto za wytrącone bierki.

\* rozgrywka kończy się gdy jeden z graczy straci wszystkie swoje pionki i tym samym wygrywa

\* gdy dochodzi do pata obu graczom doliczane są punkty

|  |  |
| --- | --- |
| Opis testu | Wynik |
| Po zalogowaniu się dostępna jest zakładka z rankingiem | POZYTYWNY |
| Po kliknięciu w zakładkę z rankingiem wyświetla się lista użytkowników z przypisanymi do nich punktami | POZYTYWNY |
| Nowo zarejestrowany gracz pojawia się na liście rankingowej z liczbą punktów równą 0 | POZYTYWNY |
| Po skończonej rozgrywce liczba punktów rankingowych zwycięskiego gracza wzrasta. | POZYTYWNY |
| Liczba dodanych punktów zależna jest od wytrąconych przez przeciwnika bierek | POZYTYWNY |
| Po skończonej rozgrywce liczba punktów rankingowych przegranego gracza nie zmienia się | POZYTYWNY |

###### 

###### Dźwięki w grze

Podczas: zaznaczania pionka, wykonania ruchu, rozpoczęcia i zakończenia gry gracz będzie słyszał różne dźwięki.

\* gracz może klikać tylko w swoje bierki

\* gracz może zaznaczyć swoją bierkę, tylko w momencie gdy jest jego ruch

\* gra zaczyna się dopiero po dołączeniu drugiego gracza

|  |  |
| --- | --- |
| Opis testu | Wynik |
| Podczas zaznaczania dowolnego pionka słychać unikalny dźwięk | POZYTYWNY |
| Podczas wykonania ruchu słychać unikalny dźwięk | POZYTYWNY |
| Podczas rozpoczęcia gry słychać unikalny dźwięk | POZYTYWNY |
| Podczas zakończenia gry słychać unikalny dźwięk | POZYTYWNY |

###### 

###### Podświetlane pola

Zaznaczona bierka jest podświetlona, jak i bierki przeciwnika, które może w danej turze zbić.

\* gracz może klikać tylko w swoje bierki

\* gracz może zaznaczyć swoją bierkę, tylko w momencie gdy jest jego ruch

\* gdy bierka nie może bić, uruchamiane są inne opcje podświetlania (pomocnik użytkownika)

|  |  |
| --- | --- |
| Opis testu | Wynik |
| Po kliknięciu w dowolną bierkę pole podświetla się. | POZYTYWNY |
| Po zaznaczeniu bierki i kliknięciu w odpowiednio opisany przycisk podświetlane są dodatkowo bierki przeciwnika, które można zbić. | POZYTYWNY |
| Jeżeli dana bierka nie można bić, a są inne, które mogą, to zostają one podświetlone | POZYTYWNY |

###### 

###### Szablony graficzne figur

Możliwość wyboru jednego spośród kilku. Dwóch użytkowników może widzieć różne modele figur, tj. każdy z nich ustawia je w panelu swoich ustawień.

|  |  |
| --- | --- |
| Opis testu | Wynik |
| Zalogowany gracz ma możliwiść wyboru szablony graficznego figur | POZYTYWNY |
| Po wybraniu odpowiedniego szablonu, na planszy pojawiają się poprawne figury zgodne z szablonem | POZYTYWNY |
| Jeśli pierwszy gracz korzysta z innego szablonu figur niż drugi gracz, to oboje widzą różne, wybrane przez siebie szablony | POZYTYWNY |

###### 

###### Schematy gier

Na początku gry niestandardowe ustawienia planszy: brak pionków, brak królowej, przemieszane figury w drugim rzędzie. Minimum trzy takie schematy, maksimum dziewięć. Punkt ten jest opcją, tj. gracz niekoniecznie musi wybrać na początku jakikolwiek schemat niestandardowy.

|  |  |
| --- | --- |
| Opis testu | Wynik |
| Domyślnie zaznaczone jest ustawienie klasyczne | POZYTYWNY |
| Wybranie tylko jednego schematu gry - brak możliwości zaznaczenia kilku | POZYTYWNY |
| Wybranie schematu “Classical arrangement” skutkuje wyświetleniem na planszy schematu tradycyjnego | POZYTYWNY |
| Wybranie schematu “No Pawns” skutkuje wyświetleniem na planszy schematu bez pionków | POZYTYWNY |
| Wybranie schematu “No Queens” skutkuje wyświetleniem na planszy schematu bez królowych | POZYTYWNY |
| Wybranie schematu “Chessmen in first and last row mixed - extreme” skutkuje wyświetleniem na planszy schematu z wymieszanymi figurami z pierwszego i ostatniego rzędu | POZYTYWNY |
| Wybranie schematu “Chessmen positioned vertically, no Pawns - extreme” skutkuje wyświetleniem na planszy schematu ustawionego pionowo, a nie poziomo | POZYTYWNY |
| Specjalnych schematów gry jest nie mniej niż 3 i nie więcej niż 9 | POZYTYWNY |

###### 

###### Pokoje gry

Jeden z graczy tworzy pokój gry. Nazwa się generuje sama, bądź gracz ją ustawia. Po dołączeniu drugiego gracza do gry ta automatycznie startuje.

|  |  |
| --- | --- |
| Opis testu | Wynik |
| Po kliknięciu przycisku tworzącego pokój generuje się on | POZYTYWNY |
| Po stworzeniu pokoju jest on widoczny na liście pokojów | POZYTYWNY |
| Gracz może wygenerować pokój o istniejącej już nazwie | POZYTYWNY |
| Gracz może wygenerować pokój bez nazwy | POZYTYWNY |
| Gracz może wygenerować pokój o dużej liczbie znaków (ponad 200) | POZYTYWNY |
| Po pojedynczym kliknięciu na pokój widoczny na liście wyświetlane są informacje o wybranym pokoju | POZYTYWNY |
| Po podwójnym kliknięciu na pokój widoczny na liście gra zaczyna się automatycznie, a przeciwnikiem jest host | POZYTYWNY |
| Zajęte pokoje (przez gracza *host* i *guest*) nie są wyświetlane na liście pokojów | POZYTYWNY |

###### 

###### Pomocnik użytkownika

Jeżeli zaznaczona przez gracza bierka w danej turze nie może bić, a są inne na planszy, które bić mogą - użytkownikowi podświetlane są te inne bierki, aby wykonał ruch poprawnie. Ruch bez bicia dopuszczony jest dopiero wtedy, gdy żadna z bierek nie może nikogo zbić.

|  |  |
| --- | --- |
| Opis testu | Wynik |
| Po zaznaczeniu bierki, która nie ma możliwości bicia i uruchomieniu pomocnika użytkownika podświetlane są inne bierki gracza, które mają taką możliwość | POZYTYWNY |
| Dodatkowa funkcjonalność. Jeśli zaznaczona bierka nie może bić i żadne inne nie mogą bić - podświetlane są pola, na które może ruszyć się bierka. | POZYTYWNY |
| Jeśli któraś z bierek ma możliwość bicia, to nie można wykonać zwykłego ruchu | POZYTYWNY |
| Jeśli żadna z bierek nie ma możliwość bicia, to można wykonać zwykły ruch | POZYTYWNY |

###### 

###### Promocja pionka

Jeżeli pionek dotrze na drugi koniec planszy - może on być zamieniony w dowolną bierkę.

|  |  |
| --- | --- |
| Opis testu | Wynik |
| Jeśli pionek dotrze na drugi koniec planszy, to po automatycznym pojawieniu się okna użytkownik może wybrać bierkę do zamiany | POZYTYWNY |
| Pionka nie można promować do pionka | POZYTYWNY |
| Jeżeli pionek nie dotrze na drugi koniec planszy nie można go promować | POZYTYWNY |
| Jeżeli bierka nie jest pionkiem, to nigdy nie można jej promować | POZYTYWNY |

###### 

###### Konto użytkownika

Aby zagrać w antyszachy, należy się zarejestrować i zalogować - dzięki temu gracz będzie przywiązany do swojego profilu i będzie mógł kolekcjonować swoje punkty.

|  |  |
| --- | --- |
| Opis testu | Wynik |
| Po kliknięciu w przycisk rejestracji pokazuje się formularz rejestracji | POZYTYWNY |
| Po wprowadzeniu prawidłowych danych (testUser1, Password1!) rejestracja przebiega pomyślnie | POZYTYWNY |
| Po wprowadzeniu drugi raz takich samych danych rejestracja nie przebiega pomyślnie | POZYTYWNY |
| Po wprowadzeniu pustego loginu rejestracja nie przebiega pomyślnie | POZYTYWNY |
| Po błędnym powtórzeniu hasła rejestracja nie przebiega pomyślnie | POZYTYWNY |
| Po wprowadzeniu błędnego hasła (testUser2, password) rejestracja nie przebiega pomyślnie | POZYTYWNY |
| Po wprowadzeniu prawidłowych danych (testUser1, Password1!) logowanie przebiega pomyślnie | POZYTYWNY |
| Po wprowadzeniu błędnego loginu (testUser, Password1!) logowanie nie przebiega pomyślnie | POZYTYWNY |
| Po wprowadzeniu błędnego hasła (testUser1, Password1) logowanie nie przebiega pomyślnie | POZYTYWNY |
| Po prawidłowym zalogowaniu się pokazuje się indywidualny panel użytkownika z możliwością gry, stworzenia pokoju, wyświetlenia rankingu oraz personalizacji profilu. | POZYTYWNY |

###### 

##### Użyte technologie

Cały projekt został wykonany w języku Java 10. Wyjątkiem są pojedyncze zapytania SQL-owe, pliki fxml i inne pliki konfiguracyjne, ale silnie z Javą powiązane.

**PostgreSQL** - obsługa bazy danych. Dzięki temu jesteśmy w stanie przechowywać dane o kontach użytkowników, pokojach gier, samych grach oraz rankingu. Tabele w bazie danych zostały wygenerowane za pomocą frameworka Spring.

**Maven** - narzędzie służące do budowania projektu. Umożliwia, w prosty sposób, pobranie oraz użycie wymaganych zależności. Dzięki temu mając zapisaną konfigurację w jednym lub kilku plikach XML-owych każdy może automatycznie pobrać źródła wymagane do zbudowania projektu. Maven posłużył nam również do umożliwienia połączenia modułu widoku (View) z modułem Core.

**JavaFX** - stworzenie klienckiej części aplikacji z widokiem. To dzięki wykonaniu modelowania plików fxml i ich obsługi w Javie użytkownik jest w stanie zobaczyć naszą aplikację, poruszać się po niej graficznie oraz toczyć rozgrywki.

**Hibernate** - framework umożliwiający mapowanie tabel i kolumn bazodanowych na obiekty. Implementuje metody definiowane przez standard JPA. Użyliśmy go w celu prostszego dostępu i zarządzania bazą danych.

**JUnit, Jupiter** - obsługa testów. Framework wraz z IDE IntelliJ bardzo nas wspomagał przy tworzeniu *unit testów*. Byliśmy w stanie generować je w szybki i usystematyzowany sposób. W potrzebnych miejscach bardzo pomagał korzystać z procesu TDD.

**Jackson** - budowanie obiektów JSONowych. Użyty przy mapowaniu JSONowego łańcucha znaków na obiekt klasy Javy oraz w drugą stronę.

**jBCrypt** - silne haszowanie haseł zapisywanych w bazie danych. Dzięki temu są one zahaszowane oraz posolone. Haszowanie odbywa się w jedną stronę i porównywane są oba hasze do stwierdzenia, czy dane hasła są sobie równe. Dzięki procesowi kryptowania dane użytkowników są lepiej chronione.

**c3p0** - obsługa sterownika JDBC do łączenia się z bazą danych. Dzięki niemu udało się uzyskać komunikację z bazą danych w łatwy i szybki sposób.

**Spring framework** - framework, który kojarzony jest głównie z *Dependency Injection*,tj. wstrzykiwanie zależności. Wykorzystaliśmy go do komunikacji REST-owej klienta z serwerem, wystawienia REST-owych endpointów i kontrolerów, mockowania ich w testach, wstrzykiwania zależności poprzez *Constructor Injection*. Umożliwił nam też użycie wielu komponentów, które są implementowane “w biegu” przez np. Hibernate, JPA czy Springa. Ciężko opisać wszystkie możliwości Springa. Sporą jego część wykorzystaliśmy w projekcie, co zaoszczędziło nam bardzo dużo czasu.

**Spring-Boot-Autoconfigure** - narzędzie odpowiadające za automatyczną konfigurację. Pozwoliło zaoszczędzić nam kodu na zdefiniowaniu wszystkich, niezbędnych do działania aplikacji, zależności (tzw. *Beany*).

**Spring-Boot-Data-JPA** - część Springa odpowiadająca za dostarczenie warstwy komunikacji z bazą danych. Głównym jego elementem, który wykorzystaliśmy było JPA.

**JPA** - standard definiujący sposób, w jaki mapujemy tabele i kolumny bazodanowe na obiekty. Implementacja interfejsu, dostarczanego przez JPA, umożliwia tworzenie zapytań w prosty i wygodny sposób - poprzez nazwę i argumenty deklarowanej metody. “Żywy” przykład z naszego projektu:

getGameStatesByGameOrderByGameStateNumberDesc(Game game);

**Spring-Boot-Web** - narzędzie dostarczyło wszystkie niezbędne zależności, których użyliśmy do utworzenia naszego serwera REST-owego. Odpowiadało również za mapowanie napisanych przez nas metod na odpowiednie adresy URL.

**Spring-Boot-Test** - dostarczył nam narzędzi potrzebnych do testowania.

**Spring MVC Test** - framework będący częścią Springa. Udostępnia API do testowania aplikacji MVC w Junicie lub TestNG. Pozwala generować testy kontrolerów RESTowych w szybki i zautomatyzowany sposób.

##### Testy jednostkowe i integracyjne

1. **Logika szachowa**

Korzystając z JUnita przetestowane zostało zliczanie punktów po skończonej rozgrywce, obsługa planszy z pionkami, promocja pionka oraz posunięcia figur na planszy. Szczególną uwagę skupiliśmy na wykonywaniu ruchów i bić przez poszczególne typy pionków, co stanowi większą część naszych testów.

1. **RESTowe API**

W przypadku RESTa skupiliśmy się na testowaniu metod udostępnianych przez serwisy. Pokryliśmy w ten sposób operacje na bazie danych umożliwiające rejestrację, logowanie, tworzenie rankingu użytkowników oraz dostęp do historii gry. Zdecydowaliśmy się także na przetestowanie logiki zawartej w jednym z kontrolerów. Korzystaliśmy tutaj z klasy MockMVC, która dzięki metodzie perform pozwala wykonywać zapytania zapytania HTTP.

“Żywy” przykład z projektu:

mvc.perform(MockMvcRequestBuilders.post("/register")

.param("username", "jan")

.param("password", "Kowalsky9@"))

.andExpect(status().is(200));

Testy mają charakter integracyjny ponieważ Spring umożliwia nam działanie na “rzeczywistej” bazie danych i po walidacji odpowiedzi przywrócenie stanu sprzed testu (rollback).

W rdzeniu naszej aplikacji, tj. w module z logiką programu, pokrycie kodu testami wynosi **65% dla klas, 56% dla metod i 63% dla linii**. Uważamy, że taka proporcja jest bardzo zadowalająca.

##### Opis kompilacji i uruchomienia projektu

###### Kod źródłowy

Kod źródłowy programu dostępny jest pod adresem: github.com/embchnk/KillerChess

Projekt *KillerChess* składa się z dwóch modułów: **core**oraz **view**. Moduły te uruchamiane są niezależnie od siebie.

Rdzeń (core) jest częścią serwerową gry: uruchamia na serwerze aplikację, obsługuje zapytania REST-owe, zawiera w sobie logikę antyszachów, mapuje JSON-owe plansze na obiekty klas Javy (oraz w drugą stronę), komunikuje się z bazą danych.

Udało nam się dojść do stanu, gdzie moduł view nie łączy się z lokalną wersją modułu core, ale z wersją na chmurze znajdującą się na portalu heroku. Umożliwia to prawdziwą rozgrywkę online.

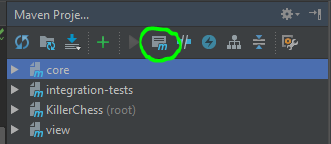
Widok (view) to część kliencka. Można uruchomić kilka widoków. Wiele widoków korzysta cały czas z jednego rdzenia i do niego wysyła zapytania, korzysta z jego logiki itd. W tej aplikacji gracz jest w stanie zalogować się do aplikacji i z niej korzystać (tj. przede wszystkim grać, ale również tworzyć i przeglądać pokoje, sprawdzić ranking graczy, informacje o sobie, wybrać szablon figur, edytować swój profil).

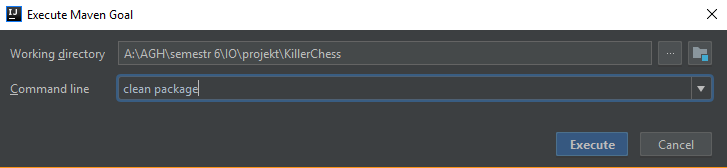
###### Kompilacja

W celu uzyskania plików .jar każdego z dwóch modułów aplikacji, należy z poziomu IntelliJ IDEA otworzyć okno “Maven Projects” (View -> Tool Windows -> Maven Projects).



W oknie “Maven Projects” należy wykonać komendę “clean package”:





###### 

###### Uruchomienie

Aby zagrać w Antyszachy, wystarczy mieć zainstalowaną Javę i uruchomić plik “uber-view-1.0-SNAPSHOT.jar” znajdujący się w folderze view/target. Jest to samodzielny plik .jar uruchamiający kliencką część aplikacji (w dowolnej lokalizacji na komputerze), która, jak wcześniej zostało wspomniane, łączy się z wersją serwera wyhostowaną na portalu heroku.

Plik .jar modułu core nie jest w tym momencie potrzebny do prowadzenia rozgrywki, ponieważ jest to lokalna wersja serwera, wymagająca posiadania lokalnie postgresa, dlatego w celu ułatwienia rozgrywki, zdecydowaliśmy się wyhostować serwer.

Wersja .jar modułu core byłaby niezbędna, gdyby moduł view łączył się właśnie z lokalną wersją modułu core (lokalną instancją serwera).