Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Tvorba počítačové hry s využitím frameworku Vaadin

Nikita Kaporulin

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Nikita Kaporulin

Systémové inženýrství a informatika
Informatika

Název práce

Tvorba počítačové hry s využitím frameworku Vaadin

Název anglicky

Creating a computer game using the Vaadin framework

Cíle práce

Hlavním cílem práce je návrh a implementace počítačové hry Šachy ve frameworku Vaadin. Hra bude určena pro online hru, tj. hráč proti hráči. Dílčím cílem je popis vývoje webových aplikací na této platformě.

Metodika

Práce sestává ze dvou části, teoretické a praktické.

Teoretická východiska pro zpracování praktické části, představující první část práce, budou popsána na základě syntézy poznatků získaných odborné literatury.

Praktická část práce spočívá v analýze, návrhu a implementaci webové aplikace Šachy. Během analýzy a návrhu bude využito standardních metod softwarového inženýrství, implementace bude provedena s využitím jazyka Java, frameworku Vaadin a souvisejících technologií.

Aplikace bude dále otestována, budou shrnuty poznatky získané během jejího vývoje a navrženy případné další možnosti jejího budoucího rozvoje.

Doporučený rozsah práce

43. st

Klíčová slova

Web application, Chess, Game, Java, Vaadin, Spring, Spring Boot, Maven

---

Doporučené zdroje informací

Soft Computing: Integrating Evolutionary, Neural, and Fuzzy Systems,Andrea Tettamanzi, Marco Tomassini,Springer Science & Business Media, 7. 9. 2001

Předběžný termín obhajoby

2020/21 LS – PEF

Vedoucí práce

Ing. Josef Pavlíček, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 8. 3. 2021

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 8. 3. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 09. 03. 2021

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Tvorba počítačové hry s využitím frameworku Vaadin" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 11.03.2021 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Josefu Pavlíčkovi, Ph.D. za odborné vedení, za pomoc a rady při zpracování této práce.

Tvorba počítačové hry s využitím frameworku Vaadin

Abstrakt

Cílem této bakalářské práce je vytvořit webovou aplikaci Šachy s využitím programovacího jazyka Java, frameworku Vaadin a dalších platformových technologií. Tato hra je určena k hraní online, tj. hráč proti hráči.

V teoretické části jsou představeny technologie a nástroje, které byly použity při vývoji klientské i serverové části aplikace.

V praktické části byl analyzován a plánován vývoj webové aplikace Šachy. Dále následuje ilustrace procesu vytváření, testování a budování aplikace, jsou popsány všechny jejich nejdůležitější části.

Na konci je shrnutí práce, jsou vyvozeny závěry a budou navrženy další kroky ke zlepšení a rozšíření aplikace.

Klíčová slova: Webová aplikace, Šachy, Hra, Java, Vaadin, Spring, Spring Boot, Maven

Creating a computer game using the Vaadin framework

Abstract

The aim of this bachelor thesis is to create a web application "Chess" using the Java programming language, Vaadin framework and other platform technologies. This game is designed to play online, i.e. player against player.

The theoretical part presents the technologies and tools that were used in the development of client and server part of the application.

In the practical part, the development of the web application "Chess" was analyzed and planned. The following is an illustration of the process of creating, testing and building an application, all the most important parts are described.

At the end is a summary of the work, conclusions are drawn and further steps will be suggested to improve and expand the application.

Keywords: Web application, Chess, Game, Java, Vaadin, Spring, Spring Boot, Maven

Obsah

Seznam obrázků

Seznam zdrojových kódů

Seznam použitých zkratek

AJAX (Asynchronous JavaScript and XML)

API (Application Programming Interface)

RIA (Rich Internet Application)

IDE (Integrated development environment)

JVM (Java Virtual Machine)

JPA (Java Persistence API)

GPL (General Public License)

JEE (Java Enterprise Edition)

XML (Extensible Markup Language)

GWT (Google Web Toolkit)

DOM (Document Object Model)

HTTP (Hypertext Transfer Protocol)

JSP (Jakarta Server Pages)

JSF (Jakarta Server Faces)

MVC (Model–view–controller)

Perft (Performance test, move path enumeration)

POM (Project Object Model)

RPC (Remote procedure call)

RMI (Remote method invocation)

CORBA (Common Object Request Broker Architecture)

SOAP (Simple Object Access Protocol)

JMX (Java Management Extensions)

JMS (Java Message Service)

CLI (Command Line Interface)

Perft (Performance test, move path enumeration)

DI (Dependency injection)

ORM (Object-relational mapping)

ODBC (Open Database Connectivity)

JDBC (Java Database Connectivity)

IoC (Inversion of control)

POJO (Plain old Java object)

JDK (Java Development Kit)

JRE (Java Runtime Environment)

URL (Uniform Resource Locator)

SQL (Structured Query Language)

DBMS (Database Management System)

JSON (JavaScript Object Notation)

FE (Front-End)

DB (Database)

1 Úvod

Samotná hra jako projev lidské povahy je neuvěřitelně složitá věc. Poté, co vznikla u primitivních lidí jako prostředek k přežití, hra se rychle stala prostředkem k navázání sociálních vztahů a teprve v historické éře se začala spojovat se zábavou. „Proč říct jen o hře – dokonce i civilizace vzniká jako způsob společného přežití, poté se z ní stává systém sociálních vztahů a o stovky let později mechanismus generování potěšení.“ (15)

Hlavní výhodou her je, že jsou nástrojem pro rozvoj myšlení. Herní aplikace umožní člověku analyzovat, zobecnit, systematizovat a aktivně rozvíjet znalosti, dovednosti a osobní vlastnosti. Počítačové hry prošly během své existence mnoha kvalitativními změnami, pokud jde o expresivitu a vyrobitelnost.

Jedním ze směrů ve vývoji počítačových her je jejich implementace ve formě webové aplikace. Hlavní výhodou tohoto přístupu je multiplatformní software.

Práce představuje všechny technologie a nástroje, které autor použil v procesu vývoje, jsou rovněž ilustrovány všechny kroky potřebné k nasazení a sestavení projektu, na příkladech jsou vysvětleny všechny nejdůležitější části aplikace.

Proces vývoje probíhal v IntelliJ Idea. Toto vývojové prostředí bylo vybráno proto, že je lídrem mezi konkurenty a nabízí mnoho praktických funkcí ihned po vybalení. H2 byla použita jako databáze, která je standardním nástrojem v aplikačním rámci Spring Boot.

2 Cíl práce a metodika

2.1. Cíl práce

Hlavním cílem práce je návrh a implementace počítačové hry Šachy ve frameworku Vaadin. Hra bude určena pro online hru, tj. hráč proti hráči. Dílčím cílem je popis vývoje webových aplikací na této platformě.

2.2. Metodika

Práce sestává ze dvou částí, teoretické a praktické.

Teoretická východiska pro zpracování praktické části, představující první část práce, budou popsána na základě syntézy poznatků získaných odborné literatury.

Praktická část práce spočívá v analýze, návrhu a implementaci webové aplikace Šachy. Během analýzy a návrhu bude využito standardních metod softwarového inženýrství, implementace bude provedena s využitím jazyka Java, frameworku Vaadin a souvisejících technologií.

Aplikace bude dále otestována, budou shrnuty poznatky získané během jejího vývoje a navrženy případné další možnosti jejího budoucího rozvoje.

3. Teoretická východiska

3.1. Java

„**Java** je přesně typizovaný univerzální objektově orientovaný programovací jazyk, vyvinutý společností Sun Microsystems, která byla následně získaná společností Oracle. Vývoj provádí komunita organizovaná prostřednictvím Java Community Process.“ (3)

Aplikace napsané v Javě mají flexibilní bezpečnostní systém a fungují tak, že program je přeložen do bajtkódu, takže může pracovat na jakékoli počítačové architektuře, pro kterou existuje implementace JVM.

Java není jen programovací jazyk a virtuální stroj, ale také platforma (ekosystém), která zahrnuje mnoho technologií: Java SE, Java EE, Java ME, Java Card. Budeme používat **JEE**, protože Vaadin je založen na této technologii. **Java Enterprise Edition** je sada specifikací pro vytváření softwaru na podnikové úrovni.

3.2. Vaadin

Vaadin je volně šířený aplikační rámec pro webové aplikace. Na rozdíl od knihoven Javascript, Vaadin nabízí serverově orientovanou architekturu založenou na JEE, která umožňuje většinu aplikační logiky spouštět na serveru. Technologie AJAX na straně klienta umožňuje komunikovat s uživatelem stejným způsobem jako aplikace SPA, tj. bez opětovného načítání okna.

„Vaadin používá k vykreslení prvků uživatelského rozhraní a interakci se serverem na straně klienta vlastní sadu webových komponentů nebo knihoven JavaScriptu jakožto Vue, React a Angular.“ (17)

Jednou z hlavních funkcí Vaadinu je používání Java jako jediného programovacího jazyka pro vytváření webových aplikací a webového obsahu. Framework používá model řízený událostmi a má knihovnu komponentů uživatelského rozhraní.

„Organizace datového modelu a widgetů umožňuje zobrazit velké množství dat v prohlížeči bez významného zatížení paměti RAM a bez dalších akcí ze strany vývojáře.“ (17)

Vaadin také ověřuje data na straně serveru: řeší to bezpečnostní problémy spojené s možností spoofingu dat nebo kódu Javascript. Pokud se data přicházející z prohlížeče změní nebo poškodí se, server poté, co to určí, nenechá projít dotazy.

„Rozšiřitelnost je zajištěna schopností používat další widgety napsané pro GWT, stejně jako přizpůsobení pomocí CSS.“ (17)

3.2.1. Programování řízené událostmi

Programování řízené událostmi je paradigma programování, ve kterém je provádění programu určováno událostmi - činnostmi uživatelů, zprávami z jiných programů a vláken, událostmi operačního systému.

Programování řízené událostmi lze také definovat jako způsob sestavení počítačového programu, ve kterém je v kódu výslovně zvýrazněna hlavní cyklus aplikace (obvykle v hlavní funkci programu), jehož tělo se skládá ze dvou částí: výběr události a její zpracování.

3.2.2. Spring

“**Spring Framework** je univerzální open source aplikační rámec pro platformu Java. Spring poskytuje řešení mnoha výzev, kterým čelí vývojáři. Spring není zcela spojen s platformou Java Enterprise, a to i přes jeho rozsáhlou integraci, což je důležitý důvod jeho popularity.“ (9)

Spring implementuje vývojový model založený na nejlepších standardech branže a zpřístupňuje jej v mnoha oblastech Javy.

Spring lze považovat za soubor menších aplikačních rámců. Většina z těchto rámců může fungovat nezávisle na sobě, jsou však velmi efektivní, pokud se používají společně. Podívejme se na některé z nich:

- Inversion of Control-containers (IoC): delegovaná (frameworku) konfigurace komponentů aplikace a řízení životního cyklu objektů Java.
- Data Access Framework: pracuje se systémy řízení relačních databází na platformě Java pomocí nástrojů JDBC a ORM.
- Framework MVC: aplikační rámec založený na HTTP a servletech, který poskytuje mnoho možností rozšiřitelnosti a přizpůsobení.
- Aplikační rámec pro autentifikaci a autorizaci: konfigurovatelná sada nástrojů pro procesy ověřování a autorizace, která podporuje mnoho populárních a standardních protokolů, nástrojů a postupů prostřednictvím podřízeného projektu Spring Security.
- Testování: framework, který udržuje třídy pro psaní testů jednotek a integračních testů.

3.2.2.1. Dependency injection

„**Dependency Injection (DI)** je proces poskytování externí závislosti softwarovému komponentu. Jde o specifickou formu „inverze řízení“ (IoC) při aplikaci na správu závislostí. V plném souladu s principem jednotné služby poskytuje objekt péči o budování potřebných závislostí na externím, speciálně navrženém obecném mechanismu.“ (10)

V praxi se používá k vytvoření a „předání“ potřebných závislostí tam, kde jsou požadovány, a zároveň snižuje množství boilerplate-kódu.

3.2.2.2. Spring Data JPA

„**Spring Data JPA -** součást větší rodiny Spring Data - usnadňuje implementaci softwarových repozitářů založených na JPA. Tento modul se zabývá pokročilou podporou vrstev přístupu k datům založených na JPA. Díky tomu je snadné vytvářet aplikace na základě Spring, využívající technologie přístupu k datům. “ (12)

Spring Data JPA výrazně zjednodušuje manipulaci s daty tím, že udržuje úsilí vývojáře na minimu. V praxi je nutné popsat rozhraní softwarových repozitářů a Spring automaticky zajistí implementaci.

3.2.2.3. Spring Security

„**Spring Security** je výkonný a přizpůsobitelný framework ověřování a kontroly přístupu. Je to de facto standard pro zabezpečení aplikací založených na Spring.

Spring Security je framework, který se zaměřuje na poskytování ověřování a autorizace pro aplikace Java. Stejně jako všechny moduly Spring spočívá skutečná síla Spring Security v tom, jak snadno ji lze rozšířit tak, aby vyhovovala individuálním potřebám.“ (13)

3.2.2.4. Apache Tomcat

„**Tomcat** je otevřený zdrojový servletový kontejner vyvinutý Apache Software Foundation. Implementuje specifikaci servletů, specifikaci JavaServer Pages (JSP) a specifikaci JavaServer Faces (JSF). Je napsán v Javě. “ (14)

Umožňuje spouštět webové aplikace a obsahuje řadu automaticky se konfigurujících programů, používá se jako samostatný webový server, jako server obsahu ve spojení s HTTP serverem Apache a jako kontejner servletů v aplikačních serverech JBoss a GlassFish.

3.2.2.5. Spring Boot

Spring Boot je framework, který pomáhá začít rychle a zahrnuje Spring, Tomcat Server a další užitečné nástroje, jako je rozhraní příkazového řádku (CLI).

"Zde je několik věcí, které Spring Boot dokáže:

- Automatická konfigurace. Spring Boot může automaticky nakonfigurovat aplikaci.
- Počáteční závislosti. Řeknete Spring Boot, jaké funkce potřebujete, a to zajistí, že se do sestavy přidají požadované knihovny.
- Rozhraní příkazového řádku. Tato funkce umožňuje psát aplikace pouze pomocí kódu aplikace, ale nevyžaduje tradiční sestavení projektu.
- Akční člen. Poskytuje vám představu o tom, co se děje uvnitř spuštěné aplikace Spring Boot. “ (6)

3.3. Java Development Kit

„**Java Development Kit (JDK)** je základem vývojového prostředí Java. Obsahuje různé nástroje potřebné k vývoji aplikace Java. Hlavním z nich je Java Runtime Environment (JRE), které umožňuje spouštět aplikace Java a obsahuje kompilátor Java.“ (17)

3.4. Apache Maven

„**Apache Maven** je framework pro automatizaci sestavování projektů na základě popisu jejich struktury v souborech v jazyce POM, což je podmnožina XML. Projekt Maven je publikován komunitou Apache Software Foundation, kde je formálně součástí projektu Jakarta.“ (1)

Maven poskytuje deklarativní sestavení projektu. Soubory popisu projektu obsahují jeho specifikaci, nikoli jednotlivé příkazy provádění. Všechny úlohy zpracování souborů popsané ve specifikaci provádí Maven pomocí pluginů.

3.5. Databáze

„**Databáze** je uspořádaná sbírka strukturovaných informací, které jsou uloženy elektronicky v počítačovém systému. Databáze je obvykle spravována systémem řízení databáze (DBMS). Data společně s DBMS a aplikacemi, které jsou s nimi spojeny, se nazývají databáze. “ (11)

Existuje mnoho různých typů databází: relační databáze, objektově orientované, distribuované, grafové, NoSQL a další.

„Data jsou obvykle formátována jako řádky a sloupce v řadě tabulek, aby bylo zajištěno efektivní zpracování dat a dotazování. Je možné přistupovat, spravovat, upravovat, aktualizovat, kontrolovat a uspořádat data. Většina databází používá k zápisu a dotazování dat Structured Query Language (SQL).“ (11)

3.5.1. Systém řízení báze dat (DBMS)

„DBMS slouží jako rozhraní mezi databází a uživateli nebo programy, které uživatelům poskytuje možnost načítat a aktualizovat informace a spravovat jejich organizaci a optimalizaci.“ (11)

DBMS také umožňuje správu (sledování výkonu, ladění, zálohování a obnovení).

3.5.2. Structured Query Language (SQL)

„**Structured Query Language (SQL)** je programovací jazyk používaný ve většině relačních databází k dotazování, zpracování a definování dat a řízení přístupu.“ (11)

3.5.3. H2

„**H2** je open-source multiplatformní DBMS. Může fungovat jako vložená databáze nebo v režimu serveru a má velikost souboru jar asi 1 MB. “ (7) Navzdory své malé velikosti (něco přes 1 MB) podporuje H2 následující funkce hned po vybalení:

- Dva režimy provozu (klient-server, vestavěný)
- Dva režimy ukládání (souborový systém, paměť)
- Podpora plánů provádění dotazů
- Podpora shlukové analýzy a replikace
- Šifrování dat
- Externí (propojené) tabulky
- Ovladač ODBC
- Fulltextové vyhledávání
- Definice domén
- Multiversion souběžný přístup
- Podpora sekvencí
- Podpora klíčových slov LIMIT a OFFSET v dotazech
- Dočasné tabulky
- Vypočítané sloupce
- Uživatelské agregační funkce
- Uživatelské uložené procedury
- Komprese objektů CLOB / BLOB
- Práce se soubory CSV pro čtení a zápis
- Konzola pro správu prohlížeče
- Spuštění jako služba Windows

3.6. JavaScript

„Vaadin automaticky zpracovává přenos dat mezi serverem a prohlížečem, tedy nemusíte psát žádný JavaScript pro dynamickou aktualizaci obsahu.“ (17)

JavaScript je multiparadigmatický programovací jazyk. Podporuje objektově orientovaný, imperativní a funkční styly. Jedná se o implementaci specifikace ECMAScript.

„JavaScript se běžně používá jako vestavěný jazyk pro programový přístup k aplikačním objektům. Nejčastěji se používá v prohlížečích jako skriptovací jazyk pro přidání interaktivity na webové stránky. “ (4)

Základní architektonické prvky: dynamické typování, slabé typování, automatická správa paměti, funkce jako objekty první třídy.

3.6.1. AJAX

„AJAX je sadou metod a nástrojů pro webový vývoj, které se používají na straně klienta k vytváření asynchronních webových aplikací. Díky Ajaxu webové aplikace mohou odesílat a přijímat data ze serveru asynchronně (na pozadí), aniž by zasahovaly do zobrazení a chování stávající stránky pomocí oddělení komunikační vrstvy od vrstvy prezentace. “ (16)

Ajax umožňuje webovým aplikacím dynamicky měnit obsah, aniž by bylo nutné znovu načíst celou stránku.

3.7. Node.js

„Node.js je běhové prostředí JavaScriptu (velmi podobné JRE v Javě). Vaadin interně používá Node k vytvoření optimalizované aplikace. “ (17)

Node.js je softwarová platforma, založená na motoru V8 (překládá JavaScript do strojového kódu), která transformuje JavaScript z vysoce specializovaného jazyka do jazyka pro všeobecné účely. Node.js přidává schopnost JavaScriptu spolupůsobit se vstupně-výstupními zařízeními prostřednictvím svého API napsaného v C ++, připojovat další externí knihovny napsané v různých jazycích a poskytovat k nim vyvolání z kódu JavaScript. Node.js se používá hlavně na serveru a vykonává funkci webového serveru, ale na Node.js je také možné vyvíjet desktopové aplikace a dokonce i programovat mikrokontroléry. Node.js je založen na programování řízené událostmi a asynchronním programování s neblokujícím vstupem/výstupem.

3.8. Vývojové prostředí (IDE)

„IDE je nástroj, který používáte k psaní a spouštění kódu během vývoje.“ (17)

Vývojové prostředí (IDE) je soubor softwarových nástrojů, který je používán programátory k vývoji softwaru (SW).

Vývojové prostředí zahrnuje:

- textový editor;
- překladač (kompilátor a/nebo interpret);
- nástroje pro automatizaci sestavování;
- debugger

3.8.1. IntelliJ IDEA

IntelliJ IDEA je integrované softwarové vývojové prostředí pro mnoho programovacích jazyků, zejména Java, JavaScript, Python, vyvinuté společností JetBrains. Disponuje širokou škálou nástrojů:

- „Nástroje pro sestavování

- Integrace se systémy správy verzí
- HTTP-klient
- Nástroje profilování
- Debugger
- Dekompilátor
- Pokrytí kódu
- Práce s databázemi a SQL“ (8)

3.9. Systémy správy verzí

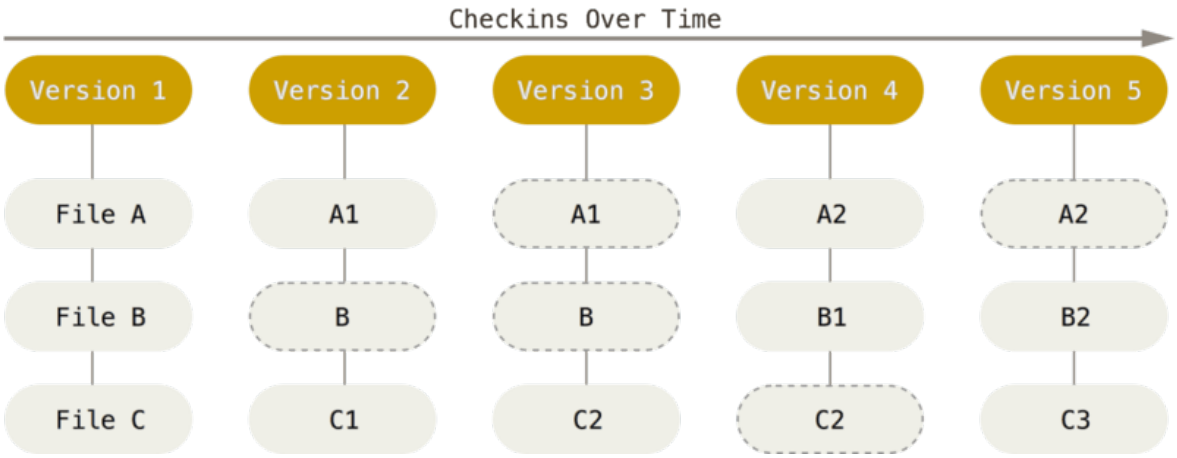
Systémy správy verzí jsou softwary pro usnadnění práce s proměnlivou informací. Umožňuje uložit více verzí stejného souboru, v případě potřeby se vrátit k dřívějším verzím, určit, kdo a kdy provedl změnu, a mnoho dalšího. Takové systémy se nejčastěji používají při vývoji softwarového kódu.

3.9.1. Git

„Git umožňuje sledovat a spravovat zdrojový kód vaší aplikace. Umožní vám, například, snadno opravit jakékoli chyby související s poruhou.“ (17)

Git je distribuovaný systém správy verzí. Projekt vytvořil Linus Torvalds za účelem řízení vývoje linuxového jádra. První verze byla vydána 7. dubna 2005. Dnes ho podporuje Junio Hamano.

„Přístup Git k ukládání dat je podobný souhrnu snímků miniaturního souborového systému (Obrázek 1). Pokaždé, když uděláte potvrzení, tj. uložíte stav svého projektu do Gitu, systém si zapamatuje, jak každý soubor v danou chvíli vypadá, a uloží odkaz na tento snímek. Aby se zvýšila účinnost, pokud se soubory nezměnily, Git si tyto soubory znovu nepamatuje, ale vytvoří pouze odkaz na předchozí verzi stejného souboru, který již byl uložen. Git prezentuje svá data dejme tomu jako proud snímků.“ (5)



Obrázek 1. Ukládání dat jako snímků projektu v čase (zdroj: 2)

4. Vlastní práce

V teoretické části byly představeny nástroje a technologie, které byly použity při vývoji webové aplikace „Šachy“. Následuje popis procesu analýzy, návrhu, implementace a na závěr shrnutí provedené práce.

4.1. Analýza

Nejprve je nutné odpovědět na otázky:

- Co je to šachy?
- Jakou herní logiku je potřeba implementovat?
- Co je třeba implementovat kromě logiky hry?
- Jaké technologie použít?
- Jaké jsou požadavky na webovou aplikaci „Šachy“?

4.1.1. Co je to šachy?

Šachy je logická desková hra se speciálními figurkami na čtvercové desce, rozdělené na 64 polí pro dva hráče. Hlavním cílem hry je mat soupeřova krále.

4.1.1.1. Pravidla

Šachová partie se hraje mezi dvěma soupeři na šachovnici pohybem šachových figurek. Soupeřem v šachové partii může být jedna osoba, tým několika lidí nebo počítač. Táto práce popisuje proces vývoje hry pouze pro dva lidi (proti sobě).

Hra se odehrává na desce rozdělené do stejných čtvercových polí. Velikost desky je 8×8 polí. Svislé řádky polí (sloupce) jsou označeny latinskými písmeny od a do h zleva doprava, vodorovné řádky (řady) - čísly od 1 do 8 zdola nahoru; každé pole je identifikováno kombinací odpovídajícího písmene a čísla. Pole jsou obarvená tmavými a světlými barvami (a jsou nazývána příslušně černá a bílá), takže svislá a vodorovná

sousedící pole jsou obarvená různými barvami. Deska je umístěna tak, že rohový čtverec vpravo od hráče je bílý (pro bílé je to čtverec h1, pro černé - čtverec a8).

Na začátku hry hráči mají stejnou sadu figurek (Obrázek 2). Figurky jednoho z hráčů se běžně nazývají „bílé“, druhého - „černé“. Bílé figurky jsou barevně světle, černé figurky - tmavě. Samotní hráči se podle barvy svých figurek nazývají „bílí“ a „černí“.

Obrázek 2. Počáteční pozice figurek na obou stranách. (zdroj: autor)

Šachové figurky jsou král (♔, ♚), dáma (♕, ♛), dvě věže (♖, ♜), dva střelci (♗, ♝), dva jezdci (♘, ♞) a osm pěšců (♙, ♟). Bílé zaplňují první a druhou řadu, černé - sedmou a osmou. Pěšáci se nacházejí na druhé, respektive sedmé řadě.

Hra spočívá v tom, že se hráči střídají při tazích. Bílý provádí první tah. S výjimkou braní mimochodem a rošády, popsaných níže, tah spočívá ve skutečnosti, že hráč přesouvá jednu ze svých figurek na jiné pole podle následujících pravidel:

- Figurky (kromě jezdců) se pohybují po přímce, zatímco všechny mezilehlé čtverce mezi počátečním a posledním musí být volné (neměly by mít vlastní nebo cizí figurky). Výjimkou je tah jezdcem.
- Tah na pole obsazené vlastní figurkou je nemožný.
- Při tahu na pole obsazené cizí figurkou, táto je odstraněna z hrací plochy (braní).
- Král se pohybuje na sousední čtverec svisle, vodorovně nebo úhlopříčně.
- Dáma se pohybuje na jakékoli vzdálenosti svisle, vodorovně nebo úhlopříčně.
- Věž se pohybuje na jakékoli vzdálenosti svisle nebo vodorovně.
- Střelec se pohybuje úhlopříčně na libovolnou vzdálenost.
- Jezdec se pohybuje na kterékoli z okolních polí v rozsahu čtverce 5x5, kromě těch, která jsou umístěna vertikálně, horizontálně nebo diagonálně od něj. V tomto případě je barva pole, na kterém se jezdec pohybuje, vždy opačná k barvě pole, na kterém stojí. Jezdec se také od ostatních figurek liší tím, že jiné figurky stojící v jeho cestě nezasahují do jeho pohybu („přeskakuje“ přes ně).
- Pěšec se může pohybovat pouze dopředu (směr „dopředu“ se nazývá směr do osmé řady pro bílé nebo do první řady pro černé): bez braní – na jeden čtverec dopředu svisle a s braním - úhlopříčně na jeden čtverec dopředu-doprava nebo dopředu-doleva. Pokud je pěšec na počátečním čtverci (druhá řada pro bílé a sedmá pro černé), pak se navíc může posunout o dvě pole vpřed, aniž by zajal. Když se pěšec přesune na poslední řadu (pro bílé - na osmou, pro černé - na první), je podle volby hráče nahrazen jakoukoli jinou figurkou stejné barvy, kromě krále (povýšení pěšce). Povýšení pěšce je součástí tahu, kterým je přesunutý na poslední řadu. Pokud tedy například figurka povýšená z pěšce ohrožuje soupeřova krále, je tento král v důsledku tahu pěšce na poslední řadu okamžitě v šachu.

Čtverec je vyvolán pod hrozbou nebo vzat, pokud během jeho tahu může figurka vzít soupeřovu figurku umístěnou na tomto poli (bez ohledu na to, zda se na tomto polí taková figurka nachází). Čtverec je považován za vzatý, i když ve skutečnosti není možné figurku přesunout, protože staví svého krále pod hrozbu.

Existují dva speciální tahy:

- Rošáda — pokud se král a jeden z věží stejné barvy od začátku hry nepohybují, pak může král a tato věž jedním tahem současně změnit polohu. Při rošádě je král posunut o dvě pole směrem k věži, věž je umístěna na pole mezi počáteční a konečnou pozicí krále. Rošáda není možná, pokud se král nebo odpovídající věž již pohybovali. Rošáda je dočasně nemožná, pokud je mezi králem a věží nějaká figurka, stejně i když je pole, na kterém král stojí, nebo pole, které musí překonat, nebo pole, které musí obsadit, napadeno jednou ze soupeřových figurek.
- Braní mimochodem — když pěšec provede dvoučtvercový tah přes vzaté pole, které je pod hrozbou soupeřova pěšce, může být jeho soupeřovým pěšcem zajat odvetným tahem. V tomto případě je pěšec soupeře přesunut na vzaté pole a vzatý pěšec je odstraněn z hrací plochy. Braní mimochodem je možné pouze jako reakce na pohyb pěšce přes vzatý čtverec, při dalších tazích to již není povoleno.

Král, nacházející se na vzatém poli, nazývá se „pod hrozbou“. Tahy, po kterých král, který provedl tah, zůstává nebo stáává se pod hrozbou, jsou zakázány.

Pokud je hráčův král pod kontrolou a hráč nemá žádný tah, aby tuto hrozbu odstranil, jedná se o mat a tedy o porážku. Cílem hry je právě mat soupeřova krále.

Pokud hráč během svého kola nemá možnost provést žádný tah podle pravidel, ale hráčův král není pod hrozbou, tato situace se nazývá pat.

Hra končí výhrou jedné ze stran nebo remízou.

Výhra se zaznamenává v následujících případech:

- Mat. Hráč, který dává mat, vyhrává.
- Jeden z hráčů kapituluje.
- Jeden z hráčů vyčerpal čas.

Remíza se zaznamená v následujících případech:

- Pat.
- Obě strany se shodly s remízou.
- Mat není dosažitelný žádnou posloupností tahů.
- Třikrát opakování stejné polohy (nemusí to být nutně pro tři tahy v řadě), navíc pojem polohy v tomto případě zahrnuje pozici figurek, pořadí tahu a možné tahy (včetně práva na rošádu a braní mimochodem pro každou stranu).
- Pravidlo 50 tahů: během posledních 50 tahů obou stran nedošlo k žádnému braní nebo k žádnému pohybu pěšce.
- Jeden z hráčů vyčerpal čas.

4.1.2. Jakou herní logiku je potřeba implementovat?

- Je nutné implementovat šachová pravidla popsaná výše
- Vytvoření šachové hry pro dva hráče
- Připojování k vytvořené hře
- Zobrazení všech dostupných her

4.1.3. Co je třeba implementovat kromě logiky hry?

Na základě skutečnosti, že se šachová hra hraje mezi dvěma soupeři, je nutné vybrat uživatelskou entitu k identifikaci hráče, proto by webová aplikace „Šachy“ musela mít následující funkce:

- Registrace
- Přihlášení
- Systém oznámení, aby aplikace mohla hlásit chyby uživateli

4.1.4. Jaké technologie použít?

Na základě cíle této práce, jako platforma a jazyk pro vývoj byla vybrána Java verze 11. Hlavním frameworkem je Vaadin verze 14. K sestavení aplikace byl použit Maven a npm (který je dodáván s Node.js, což je také nutné pro vytváření FE). Kromě

toho bylo použito několik užitečných knihoven, například lombok, které umožní používat vzor „Dekoratér“ bez zbytečných potíží. Pro rychlý start byl vybrán framework Spring Boot, který obsahuje hlavní moduly Spring Framework popsané v teoretické části.

4.1.5. Jaké jsou požadavky na webovou aplikaci „Šachy“?

Zde jsou shrnuty základní požadavky na webovou aplikaci:

- Aplikace musí mít funkci šachové hry
- Aplikace musí mít funkci vytváření hry
- Během hry musí aplikace komunikovat s oběma uživateli a zobrazovat změny na šachovnici
- Aplikace musí mít funkci pro připojení k již vytvořené hře
- Aplikace musí mít funkci zobrazování všech dostupných her
- Aplikace musí mít funkci Registrace
- Aplikace musí mít funkci Přihlášení
- Aplikace musí ověřit údaje zadané uživatelem a reagovat, pokud nejsou správné
- Systém oznámení, aby aplikace mohla hlásit chyby uživateli
- Některá data, jako jsou informace o uživateli, musí být uložena v databázi (heslo uživatele musí být při uložení v DB zašifrováno)
- Přístup k hrám by měli mít pouze registrovaní uživatelé
- Aplikace musí být důkladně otestována

4.1.6. Výsledky analýzy

Na základě analýzy byly formulovány hlavní požadavky na webovou aplikaci „Šachy“, navíc byly vybrány nástroje a technologie nezbytné k dosažení našich cílů.

Java verze 11 byla vybrána, protože se jedná o verzi LTS a plně splňuje požadavky pro daný úkol. Vaadin 14 ve spojení s Spring Boot také velmi usnadní zadání, protože převezme většinu rutinních úkolů, jako je počáteční konfigurace projektu. Jako hlavní nástroj pro implementaci funkcí Registrace a Přihlášení bude použit „vůdce“ Spring Security. V našem případě, H2 bude stačit jako databáze. Veškerá základní logika aplikace je napsána pomocí standardních jazykových nástrojů Java, s výjimkou knihovny lombok,

která pomůže snížit množství boilerplate-kódu, například popisem logiky getter a setter. Aby bylo možné „přehrabávat“ data uvnitř aplikace, je vyžadován DI. Pro tento účel byl zvolen Spring Data JPA, protože je zcela vhodný a je dodáván se Spring frameworkem. Jako nástroj, který implementuje principy ORM, byl vybrán klasický javax.persistence API. Jako vývojové prostředí slouží IntelliJ IDEA.

4.2. Návrh

Dále se pokusíme odpovědět na otázky: „Jak reprezentovat „Šachy“ v abstraktní podobě v programovacím jazyce Java?“. K zvýraznění entit a vztahů mezi nimi bude použít OOP přístup.

Rovněž proběhne návrh webové aplikace „Šachy“, a to specifikace případů užití, struktura databázového modelu, obsah tříd: pole a metody, se stručným popisem toho, k čemu slouží.

4.2.1. Specifikace případů užití

Specifikace případů užití je popis chování systému při interakci s někým (nebo s něčím) z externího prostředí. Systém může reagovat na externí dotazy Aktéra, může sám iniciovat interakci. Jinými slovy, případ užití popisuje „kdo“ a „co“ může udělat s daným systémem, nebo co může systém dělat s „kým“ nebo „čím“.

USE CASE **REGISTRACE**:

Popis: Registrace nového uživatele.

Aktéři: Uživatel, Systém

Podmínky pro spuštění: Uživatel se musí nacházet na stránce pro registraci nového uživatele.

Scénář funkčnosti:

- Systém zobrazí registrační formulář s položkami přezdívka a heslo.
- Uživatel vyplní požadované údaje a odešle.
- Systém provede validaci dat.

- Systém zaregistruje uživatele.
- Systém zobrazí oznámení o úspěšné registraci.
- Systém přesměruje uživatele na stránku přihlášení.

Scénáře pro chyby či odchylky:

- Pokud uživatel data nezadal, systém to nahlásí pomocí oznámení.
- Pokud uživatel zadal data již existujícího uživatele, bude o tom informován systémem.

USE CASE **PŘIHLÁŠENÍ**:

Popis: Přihlášení do aplikace.

Aktéři: Uživatel, Systém

Podmínky pro spuštění: Uživatel se musí nacházet na stránce přihlášení.

Scénář funkčnosti:

- Systém zobrazí uživateli přihlašovací formulář s položkami přezdívka a heslo.
- Uživatel vyplní požadované údaje a odešle.
- Systém provede validaci dat.
- Autorizovaný uživatel bude moct vejít do systému.
- Systém zobrazí oznámení o úspěšném přihlášení.
- Systém přesměruje uživatele na hlavní stránku.

Scénáře pro chyby či odchylky:

- Pokud uživatel data nezadal, systém to nahlásí pomocí oznámení.
- Pokud uživatel zadal nesprávná data, bude o tom informován systémem.

USE CASE **ZAČÁTEK HRY**:

Popis: Vytvoření nové hry nebo připojení se ke stávající. Popsaný scénář iniciuje Uživatel 1.

Aktéři: Uživatel 1, Uživatel 2, Systém

Podmínky pro spuštění: Uživatel 1 a Uživatel 2 se musí nacházet na hlavní stránce.

Scénář funkčnosti:

- Systém zobrazí uživateli seznam dostupných her a tlačítka „Vytvořit novou hru“, „Aktualizovat seznam dostupných her“.

- Uživatel 1 klikne na tlačítko „Vytvořit novou hru“.
- Systém vytvoří hru.
- Systém přesměruje Uživatele 1 na stránku se šachovnicí.
- Uživatel 2 aktualizuje seznam dostupných her.
- Uživatel 2 připojí se ke hře pomocí tlačítka „Připojit se“.
- Systém spustí hru.

Scénáře pro chyby či odchylky:

- Pokud Uživatel 1 po vytvoření hry zavře prohlížeč, přejde na jinou stránku nebo vypne počítač, vytvořená hra se stane nepřístupnou a zmizí ze seznamu dostupných her.
- Pokud se Uživatel pokusí připojit k neexistující hře, systém ho přesměruje na hlavní stránku a informuje ho o problému.

USE CASE **HRA**:

Popis: Proces hry: vyběr figurky, tah, aktualizace šachovnice v souladu s tahem.

Aktéři: Uživatel 1, Uživatel 2, Systém

Podmínky pro spuštění: Uživatel 1 a Uživatel 2 se musí nacházet na stránce hry.

Scénář funkčnosti:

- Uživatel 1 zvolí figurku, kterou bude chodit.
- Systém aktualizuje desku pro Uživatele 1 a zobrazí možné pohyby.
- Uživatel 1 dělá tah.
- Systém aktualizuje desku podle tahu pro Uživatele 1.
- Systém aktualizuje desku podle tahu pro Uživatele 2.
- Systém předá možnost tahu uživateli 2.

Scénáře pro chyby či odchylky:

- Pokud Uživatel po výběru figurky vybere jiný kamen, systém desku obnoví.

USE CASE **ODHLÁŠENÍ ZE HRY**:

Popis: Odhlášení ze hry, pokud uživatel zavřel hru, prohlížeč nebo vypnul počítač.

Aktéři: Uživatel 1, Uživatel 2, Systém

Podmínky pro spuštění: Uživatel 1 a Uživatel 2 se musí nacházet na stránce hry.

Scénář funkčnosti:

- Uživatel 1 opustil hru jakýmkoli způsobem.
- Systém pozastaví hru (dokud někdo znovu nenastoupí na místo Uživatele 1).

Scénáře pro chyby či odchylky:

- Pokud Uživatel 2 opustí hru po uživateli 1, systém ukončí hru a ona již nebude k dispozici.
- Uživatel 1 může, pokud chce, znovu usednout na své místo.

USE CASE **VÝHRA/REMÍZA VE HŘE**:

Popis: Co se stane, když vyhraje jeden z hráčů.

Aktéři: Uživatel 1, Uživatel 2, Systém

Podmínky pro spuštění: Uživatel 1 a Uživatel 2 se musí nacházet na stránce hry.

Scénář funkčnosti:

- Uživatel 1 vyhrává hru.
- Systém informuje hráče o konci hry a výsledcích.
- Systém hru „uzavře“.
- Systém přesměruje oba uživatele na hlavní stránku.

Scénáře pro chyby či odchylky:

- Pokud někdo z hráčů aplikaci v tuto chvíli opustí, bude spuštěn scénář popsaný v odstavci ODHLÁŠENÍ ZE HRY.

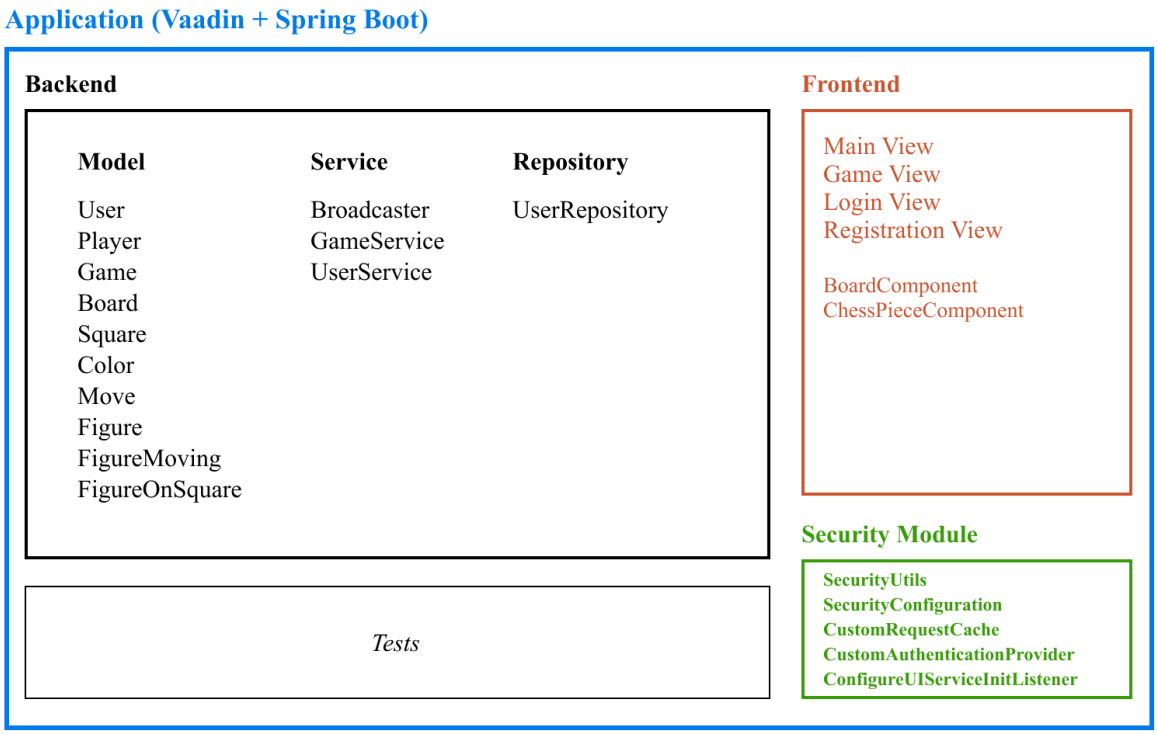
4.2.2. Struktura databázového modelu

Bylo rozhodnuto uložit pouze klíčová data nezbytná pro autorizaci uživatele, jako primární klíč a identifikátor je použito uživatelské jméno (*username*). Jako návrhový nástroj byla původně vybrán servis dbdiagram.io, takže pokud bude potřeba rozšířit funkčnost, je možné použít export modelu, například ve formátu MySQL nebo PostgreSQL.

```
Table User {
    username string [pk]
    passwordHash string [not null]
}
```

1. Kód 1 - Tabulka User (2)

4.2.3. Architektura aplikace



Obrázek 3. Architektura aplikace (zdroj: autor)

Bylo rozhodnuto zvolit monolitický přístup pro tak malou aplikaci, rozdělit logiku serveru (backend) a logiku prezentace (ui) do samostatných balíčků (packages). Modul zabezpečení (Security) byl také odebrán samostatně, protože navzájem působí jak se serverem, tak i s klientem (Obrázek 3). O každé třídě a jejím účelu si podrobněji promluvíme později a na příklady implementace se podíváme v odstavci Implementace.

4.2.4. Třídy a metody

Backend - Model

Bylo rozhodnuto reprezentovat herní desku (Board) ve formě matice 8x8 a současně byla pro usnadnění testování použita notace FEN.

- **enum Color** — má tři možné stavy: *none, white a black* a jednu jedinou metodu *flipColor*, která bere barvu jako argument a vrátí jí opačnou barvu.
- **class Square** —má dvě vlastnosti: *x* a *y* typu int. Při vytváření přijímá objekt třídy ve svém konstruktoru buď hodnoty *x* a *y*, nebo název čtvrce typu String z notace FEN, například „h8“. Třída má také metodu *yieldSquares*, která vrací ArrayList <Square>, metodu *onBoard*, která kontroluje, zda je čtverec v povolených hodnotách (na desce), a metodu *getName*, která vrací název čtverce.
- **class User** — je Entity naší databáze. Má dvě vlastnosti: *username* String, *passwordHash String* a jednu metodu *setPasswordHash*, ve které je heslo „hašováno“ a „soleno“, tj. šifrováno pomocí package Security.
- **class Player** — je podobný třídě User, až na to, že zde je místo hesla užitečná pro nás vlastnost *color*.
- **enum Figure** — výčet obsahuje název všech možných šachových figurek nebo absenci figurky (*none*). K dispozici je také vlastnost *figure* typu char a několik pomocných metod: *getColor, getFigureType (přijímá char, vrátí typ Figure), yieldPromotions (slouží k funkci nahrazení figury, když pěšec dosáhl poslední řády).*
- **class FigureMoving** — pomocná třída pro provedení tahu. Má několik konstruktorů, vlastnosti (*Figure figure, Square from, Square to, Figure promotion*) a mnoho pomocných metod (*int deltaX, int deltaY, int absDeltaX, int absDeltaY, int signX, int signY, Figure placedFigure, toString*).
- **class FigureOnSquare** — má pouze dvě vlastnosti *Figure figure, Square square* a jeden konstruktor, který přijímá argumenty stejných typů jako vlastnosti.
- **class Move** — zodpovědný za funkci tahu. Má pouze dvě vlastnosti *FigureMoving fm, Board board*, konstruktor, který přijímá *Board board*, a mnoho pomocných metod: *canMove, canMoveFrom, canMoveTo, canFigureMove, canKingMove* a tak dále pro všechny typy figur.
- **class Board** — je šachovnice. Má vlastnosti *fen, moveColor, moveNumber, figures (matice typu Figure[][]), canCastleA1, canCastleH1, canCastleA8, canCastleH8, enpassant (čtverec pro braní mimochodem), drawNumber (počítadlo pro pravidlo 50 tahů*), konstruktor, který přijímá metody *fen* a další: *init, initFigures,*

initMoveColor, initCastleFlags, initEnpassant, initDrawNumber, initMoveNumber, yieldFigures, generateFen, fenMoveColor, fenCastleFlags, fenEnpassant, fenDrawNumber, fenMoveNumber, fenFigures, getFigureAt, setFigureAt, move, moveCastleRook, updateCastleFlags, dropEnpassant, updateEnpassant, isCheck, isCheckAfter, canEatKing, findEnemyKing.

- **class Game** — má vlastnosti *fen, board, move, player1, player2, currentPlayer, isCheck, isCheckmate, isStarted, isStalemate*, několik konstruktorů a mnoho metod: *setCheckFlags, isValidMove, start, move, flipCurrentPlayer, getMoveColor, getFigureAt, findAllMoves, getAllMoves, nextMoves (pomocní metoda pro testování*).

Backend - Service

- **UserService** — servis funguje ve spojení s UserRepository a uloží uživatele do databáze.
- **GameService** — je Singleton, objekt této třídy obsahuje seznam her typu ConcurrentHashMap <String, Game>, je zodpovědný za vytvoření hry a připojení ke stávajícím hrám.
- **Broadcaster —** speciální servis, který implementuje technologii přenosu dat Server-Push, Vaadin používá WebSockets + XHR jako standardní protokol pro přenos. (17) Má vlastnosti *Map<String, Consumer<String>> listeners a Executor executor* a několik metod: *register (pro registraci „posluchače“) a broadcast (pro odesílání dat „posluchačům“).*

Backend - Repository

- **UserRepository** — je rozhraní zděděné z JpaRepository <User, Long>, ve kterém je definována metoda *findUserByUsername.*

Security

- **class SecurityConfiguration** —standardní konfigurace Spring Security je zodpovědná za autorizaci uživatelů pomocí třídy CustomAuthenticationProvider.
- **class CustomRequestCache** — odpovědný za ukládání do mezipaměti, aby systém mohl později uživatele přesměrovat tam, kam původně chtěli jít, ale neměli na to právo.

- **class CustomAuthenticationProvider** — odpovídá za vyhledání uživatele v databázi a na základě výsledků jej autorizuje či nikoli.
- **class ConfigureUIServiceInitListener** — pomocná třída odpovědná za přesměrování.
- **class SecurityUtils** — pomocná třída obsahující několik užitečných metod: *isUserLoggedIn, encoder, isFrameworkInternalRequest.*

UI - View

Každé View představuje stránku. Každá stránka obsahuje prvky, které jsou zároveň POJO i DOM (volitelně) a určitě má vlastní cestu (Route).

- **RegistrationView** — registrační stránka, obsahuje registrační formulář a odkaz na přihlašovací stránku - */login*.
- **LoginView** — přihlašovací stránka aplikace, obsahuje přihlašovací formulář a odkaz na registrační stránku - */registration*.
- **MainView** — hlavní stránka obsahuje seznam všech dostupných her, tlačítko pro vytvoření hry a tlačítko pro aktualizaci seznamu.
- **GameView** — stránka hry, která obsahuje šachovnici (BoardComponent).

UI - View - Components

- **ChessPieceComponent** — představuje figurku.
- **BoardComponent** — představuje šachovnici, je zde registrován „posluchač“, hrací deska je nakreslena a aktualizována.

4.3. Implementace

Poté, co jsme analyzovali požadavky a navrhli aplikaci, musíme ji implementovat. Tento odstavec popisuje taky proces spouštění aplikace. Nebudeme popisovat vše podrobně, ale budeme se zabývat pouze částmi, které jsou pro systém důležité. Proces práce s git bude vynechán, protože není tématem projektu. Stručně řečeno, git byl použít většinou lokálně, protože na projektu pracoval pouze autor, repozitář obsahuje větve

master a sandbox. Někdy byly vytvořeny samostatné větve pro velké funkce nebo experimenty.

4.3.1. Inicializace projektu

Než začít vyvíjet Java-aplikaci, je potřeba nainstalovat vývojové nástroje a nastavit prostředí.

- Java Development Kit (JDK): Java SE 11
- Nástroj pro sestávení/správu projektů (Maven)
- Systém správy verzí (Git)
- Nástroj pro sestávení FE / Front-end build tool (Node.js)
- IDE IntelliJ IDEA

Pokusíme se to zvládnout několika kroky (*macOS*):

- Spustíme terminál a provedeme příkaz ***brew install node maven git***
- Stáhneme a nainstalujeme IntelliJ IDEA (dále IDE "načte" verzi Java SDK, kterou jsme vybrali)

Dalším krokem je vytvoření projektu. Zbavíme se manuálního přidání všech závislostí a budeme používat servis **start.vaadin.com.** Stáhneme a rozbalíme archiv (již obsahuje téměř všechny potřebné závislosti), otevřeme projekt v IntelliJ IDEA (vybíráme verzi Java SE 11, File > Project Structure > Project SDK), IDE stáhne a nainstaluje všechny potřebné závislosti včetně JDK.

```
/**
 * Vstupní bod Spring Boot aplikace.
 */
@SpringBootApplication(exclude = ErrorMvcAutoConfiguration.class)
public class Application extends SpringBootServletInitializer {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

2. Kód 2 - vstupní bod aplikace (odsud je možné spustit aplikaci z IDE)

Není nutné konfigurovat databázi H2. Spring Boot to pro nás již udělal automaticky. Po spuštění ve vývojovém režimu bude aplikace k dispozici na adrese

http://localhost:8080 a konzole databáze na adrese http://localhost:8080/h2-console. V následujících odstavcích práce nebudou uvedeny všechny části aplikace, ale pouze klíčové a nejzajímavější ke zvážení, to platí i pro fragmenty kódu níže (pro ilustraci byly vybrány pouze některé z metod).

4.3.2. Backend - Model

Package *model* obsahuje základní logiku webové aplikace „Šachy“. Modul je napsán v čisté Javě, jedinou knihovnou je zde lombok. Dekorátory *@Getter* a *@Setter* z teto knihovny jsou zodpovědné za automatické generování pomocných metod.

```
public class Game {
    // Pomocná rekurzivní metoda pro testování
    // https://www.chessprogramming.org/Perft_Results
    static int nextMoves(int step, Game game) {
        // Argument step je zodpovědný za hloubku
        if (step == 0) return 1;
        // Počet všech možných tahů v dalším tahu
        int count = 0;
        ArrayList<String> list = game.getAllMoves();
        for (String moves : list)
            count += Game.nextMoves(step - 1, game.move(moves));

        return count;
    }
}
```

3. Kód 3 - třída Game

Pref funkce (*nextMoves*) — pomocná rekurzivní metoda pro testování, prochází stromem povolených tahů a spočítá všechny konečné (povolené) tahy, které lze porovnat s dříve známými hodnotami a použít k vyhledání chyb. Mat a pat se nepočítají, pravidlo padesáti tahů je ignorováno.

```
@Getter
@Setter
@Entity
@Table(name = "User")
public class User {

    @Id
    private String username;
    @NotEmpty
    private String passwordHash;

    // Hashování a solení hesla pomocí SecurityUtils
    public void setPasswordHash(String password) {
```

```
        passwordHash = SecurityUtils.encoder().encode(password);
    }
}
```

4. Kód 4 - třída User

- *@Entity* — jsou neměnná data uložená v databázi.
- *@Table(name = "User")* — určuje tabulku pro anotované *@Entity*.
- *@Id* — určuje primární klíč *@Entity*.
- @NotEmpty — kontroluje, zda vlastnost není null nebo prázdná; lze použít na hodnoty String, Collection, Map nebo Array.

4.3.3. Backend - Service

Package *service* je vrstva mezi *model* a *view* webové aplikace „Šachy“. Dále zvážíme servis Broadcaster, který je zodpovědný za provoz technologie Server-push, a GameService, který je zodpovědný za vytvoření hry a připojení k existující.

```
public class Broadcaster {
    static Executor executor = Executors.newSingleThreadExecutor();

    static Map<String, Consumer<String>> listeners = new HashMap<>();

    public static synchronized Registration register(
            String username, Consumer<String> listener) {
        listeners.put(username, listener);

        return () -> {
            synchronized (Broadcaster.class) {
                listeners.remove(username);
            }
        };
    }

    public static synchronized void broadcast(String username, String
message) {
        Consumer<String> l = listeners.get(username);
        executor.execute(() -> l.accept(message));
    }
}
```

5. Kód 5 - servis Broadcaster

„Broadcaster je implementován pomocí vzoru *Singleton* a slouží k odesílání zpráv uživatelům. Registruje příjemce a bezpečně k nim přenáší zprávy. Aby se zabránily deadlocks, doporučuje se odesílat zprávy prostřednictvím fronty zpráv v samostatném vlákně. Použití Java ExecutorService s jediným vláknem je obvykle nejjednodušší a nejbezpečnější způsob. Metody ve třídě jsou definovány tak, aby byly synchronizovány, aby se zabránila možnost souběhu. “ (17)

Username se používá jako identifikátor uživatele, stejně jako v celém projektu. Později, když vezmeme v úvahu *View*, bude popsáno, jak zaregistrovat posluchače a jak odesílat signály o změnách v manuálním režimu.

```
@Service
@Scope("singleton")
public class GameService {
    // …

    public String createGame() {
        // …
    }

    public void setGame(String id, Game game) {
        // …
    }

    public Game getGameBySessionId(String id) {
        // …
    }

    public Map<String, Game> getAvailableGames() {
        // …
    }
}
```

6. Kód 6 - servis Game

- *@Service* — znamená, že třída obsahuje obchodní logiku.
- *@Scope("singleton")* — kontejner vytvoří jediný exemplář tohoto bean-objektu. Všechny dotazy na objekt vrátí stejný objekt, který je uložen do mezipaměti.

4.3.4. Backend - Repository

Aplikace má pouze jeden repozitář. Dekorátor *@Repository* - znamená, že třída bude fungovat jako repozitář databáze. Když definujeme rozhraní *JpaRepository*, snížujeme množství boilerplate-kódu (Spring automaticky generuje kód, který potřebujeme).

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    User findUserByUsername(String username);
}
```

7. Kód 7 – UserRepository

4.3.5. Zabezpečení aplikace

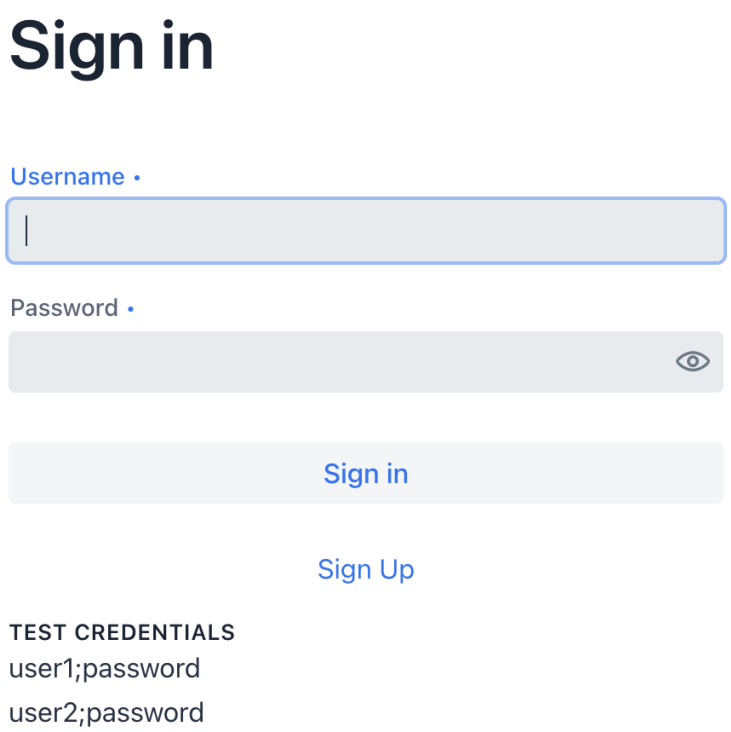
Package Security odpovídá za:

- Přístup ke statickým zdrojům

- Autorizace uživatele
- Ukládání dotazů do mezipaměti
- Tvorba *inMemoryAuthentication* testovacích přihlašovacích údajů

4.3.6. Zobrazení dat

Pro zobrazení dat (*View*) (Obrázek 4) byly vybrány standardní nástroje Vaadin, všechny uzly DOM jsou prezentovány v jazyce Java, to znamená, že bylo rozhodnuto nepoužívat template processor. V generovaném projektu jsou styly umístěny podél cesty *./frontend/styles/shared-styles.css*, bylo rozhodnuto je nerozdělit na moduly, protože aplikace je má málo.



Obrázek 4. Přihlašovací stránka (zdroj: autor)

Rozebereme několik *View* a komponent.

```
@Route(value = RegistrationView.ROUTE)
@CssImport("./styles/shared-styles.css")
```

```
public class RegistrationView extends FormLayout {
    public static final String ROUTE = "registration";

    @Autowired
    public RegistrationView(UserService userService) {
        this.userService = userService;
    }
}
```

8. Kód 8 - Registration View

- @Route — asociuje View s konkrétním fragmentem adresy URL;
- @CssImport — importuje styly do global scope, je také možné určit téma(*scope*);
- @Autowired — potřebné k získání servisu pomocí Spring DI, umožňuje vkládání závislostí.

```
@Push
@Route(value = GameView.ROUTE)
@CssImport("./styles/shared-styles.css")
public class GameView extends Div implements HasUrlParameter<String> {
    @Override
    public void setParameter(BeforeEvent event, @WildcardParameter String
parameter) {

        Location location = event.getLocation();
        QueryParameters queryParameters = location
                .getQueryParameters();

        Map<String, List<String>> parametersMap =
                queryParameters.getParameters();

        sessionId = parametersMap.get("sessionId").get(0);
        game = gameService.getGameBySessionId(sessionId);
    }
}
```

9. Kód 9 - Game View

S dekorátorem **@Push** zapneme funkci Server-push. Pro správnou funkci musíme také v souboru *application.properties* zadat *vaadin.async-supported=true*.

Server Push umožňuje HTTP-serveru odesílat prostředky klientovi dříve, než je klient požaduje. Tento přístup umožňuje poskytnout oběma hráčům aktuální informace o průběhu hry.

Metoda *setParameter* bude volána před navigací (před událostí *BeforeEnter*). Metoda získává *sessionId* vytvořené hry (na hlavní stránce, kterou nezvažujeme) z URL, vytváří a zobrazuje komponent *Borad* na základě dat.

```
public class BoardComponent extends Div {
    // …
    private void broadcast() {
        // Pošleme nový FEN jinému hráči a řekneme mu, že jsme provedli tah
        Broadcaster.broadcast(username, game.getFen());
    }

    @Override
    // Metoda životního cyklu komponentu Vaadin
    protected void onAttach(AttachEvent attachEvent) {
        UI ui = attachEvent.getUI();
        String username = // ...

        broadcasterRegistration = Broadcaster.register(username, newFen -> {
            ui.access(() -> {
                // Na základě údajů přijatých od jiného hráče
                // aktualizujeme stav hry včetně šachovnice
                game = new Game(new Board(newFen), game);
                gameService.setGame(sessionId, game);
                // Aktualizace šachovnici
                updateBoard();
            });
        });
    }

    @Override
    // Metoda životního cyklu komponentu Vaadin
    protected void onDetach(DetachEvent detachEvent) {
        // ...
        // Pozastavíme hru
        // ...
        broadcasterRegistration.remove();
        broadcasterRegistration = null;
    }
}
```

10. Kód 10 - Board Component

V konstruktoru třídy *BoardComponent* (Obrázek 5) inicializujeme a nakonfigurujeme všechny potřebné grafické prvky, poté aktualizujeme stav hry a inicializujeme připojené hráče. Třída obsahuje mnoho metod, které jsou zodpovědné za

výběr figurek, zvýraznění ctverců dostupných pro tah, aktualizaci desky atd., ale budeme uvažovat pouze o těch nejzajímavějších.

Obrázek 5. Komponent Board. (zdroj: autor)

„Metodu onAttach volá platforma Vaadin, když se komponent "připojí" k uživatelskému rozhraní (ui). Jeho obdoba, metoda onDetach, je volána těsně předtím, než se komponent „odpojí" od uživatelského rozhraní (ui)." (17)

V metodě *onAttach* je zaregistrován „posluchač" v servisu Broadcaster.

4.4. Testování

Zde se podíváme na kód několika testů, v jednom z předchozích odstavců jsme uvažovali o Pref-funkci. Během procesu vývoje byly v zásadě použity pouze unit-testy a manuální testování.

```
public class FigureMovingTests {
    // ...

    @Test
    public void deltaX() {
        Assert.assertEquals("to.x - from.x must be 0", 0, deltaX);
```

```
    }

    @Test
    public void signX() {
        Assert.assertEquals("deltaX == 0 must return 0", 0, signX);
        Assert.assertEquals("deltaX > 1 must return 1", 1, signX + 1);
        Assert.assertEquals("deltaX < -1 must return -1", -1, signX - 1);
    }
}
```

11. Kód 11 - Figure Moving Tests

Zde byly testovány některé pomocné metody třídy FigureMovingTests, například vyhledávání rozdílu mezi čtverci šachovnice, metoda je potřebná k výpočtu správnosti tahu.

```
public class GameTests {
    @Test
    public void nextMoves() {
        // Initial Position
        Assert.assertEquals(1, Game.nextMoves(0, new
            Game("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"
        )));
}
```

12. Kód 12 - Game Test

Uzly se počítají až na konci po posledním tahu. „Vyšší“ koncové uzly (například mat nebo pat) se tedy nepočítají, místo toho se bere v úvahu počet cest určité hloubky.

4.5. Sestavení a příprava k produkčnímu režimu

Pro zapnutí ***production-mode*** je nutné přidat vlastnost *vaadin.productionMode* s hodnotou true v pom.xml. Dále provadíme v terminálu příkaz ***mvn vaadin:build-frontend***, po kterém bude vytvořeno sestavení FE. Poté musíme shromáždit naši aplikaci do souboru typu .jar pomocí příkazu ***mvn clean package -Pproduction***. Archiv s aplikací se nachází podél cesty *./target/chess-1.0-SNAPSHOT.jar.*

Pro spuštění aplikaci musíme spustit příkaz *java -jar chess-1.0-SNAPSHOT.jar.* Spuštění programu můžeme také vylepšit a ukládat logy pomocí skriptu:

```
#!/bin/bash
nohup java -jar chess-1.0-SNAPSHOT.jar > chess.log 2>&1 &
echo $! > save_pid.txt
```

13. Kód 13 – Skript spuštění aplikaci v production-mode

Pro spuštění aplikaci zbývá spustit vygenerovaný .jar soubor na serveru, a ujistit se, že port 8080 není ničím blokován.

5. Výsledky a diskuze

Výsledkem práce je webová aplikace „Šachy“ vyvinutá s ohledem na stanovené požadavky. V tomto procesu se autor seznámil s vývojem počítačových her, prohloubil své znalosti o technologii Server-push a užíval si stabilního ekosystému Java s mnoha pohodlnými nástroji.

5.1. Možná vylepšení

- Funkce obnovení přístupu k účtu
- Možnost zadat e-mail během registrace s následným potvrzením registrace
- Stránka profilu s možností měnit data
- Automatická aktualizace seznamu dostupných her na hlavní stránce
- Funkce historie her
- Funkce historie tahů v šachové hře
- Zobrazení ztracených figur v šachové hře
- Funkce času v šachové hře
- Statistika hráčů
- Funkce přátel
- Funkce ratingu
- Funkce vybírání hráčů podle ratingu
- Funkce hry s počítačem
- A téměř neomezené možnosti pro vylepšení algoritmů šachových pravidel

6. Závěr

V průběhu práce bylo dosaženo hlavního cíle práce, webová aplikace „Šachy“ byla analyzována, navržena a vyvinuta pomocí frameworkuVaadin.

V teoretické části byla představena *teoretická východiska*, konkrétně byla popsána platforma Java a všechny související technologie, které byly použity při tvorbě. Spring Boot byl použit pro pohodlnou konfiguraci projektu, Spring moduly pomohly při konfiguraci bezpečnostních pravidel, autorizace, registrace a konfigurace DI. Aplikační rámec Vaadin poskytoval všechny potřebné nástroje ihned po vybalení z krabice: směrování, přístup OOP k psaní View, standardní knihovnu komponentů, zpracování událostí a pohodlné metody životního cyklu pro klienta.

V praktické části práce byla provedena analýza nadcházející práce, byly stanoveny požadavky na webovou aplikaci, zvoleny technologie a přístupy. Dalším krokem byla fáze návrhu, byla popsána specifikace případů užití, struktura databázového modelu, architektura aplikace a základní moduly. Dále byl ilustrován vývojový proces: konfigurace prostředí a projektu, implementace šachových pravidel, tvorba servisů a repozitářů, konfigurace autorizace a bezpečnosti. Na konci je uvedeno testování (které probíhalo během celého procesu vývoje), je popsán proces sestavení a příprava na produkční režim.

Silnou stránkou řešení je to, že je napsáno pomocí výkonných nástrojů z ekosystému Java, přístupu OOP, pokryto testy a v případě potřeby jej lze snadno rozšířit. Slabá stránka řešení spočívá v tom, že ve skutečnosti je to jen obrobek pro plnohodnotnou šachovou aplikaci, jako je chess.com.

Ve výsledku práce se autor seznámil se zajímavými technologiemi a nástroji, prohloubil teoretické a praktické znalosti a vytvořil fungující webovou aplikaci „Šachy“ dostupnou na ulitka.xyz:8080 a zdrojový kód projektu se nachází zde: https://github.com/studioo/czu-project-chess. Byly také hodnoceny nástroje (*Spring, Vaadin*) a výhody, které přinášejí.

7. Seznam použitých zdrojů

1. *Apache Maven* [online]. USA: 2020 [cit. 2021-03-05]. Dostupné z: <https://maven.apache.org/>.
2. Česká zemědělská univerzita v Praze, Praha – Suchdol. *Chess App* [online]. Dostupné z: <https://dbdiagram.io/d/6032bb1cfcdcb6230b20d195>.
3. EVANS, Benjamin J, FLANAGAN, David. *Java in a Nutshell*. 7. vyd., USA: O'Reilly Media, Inc, 2018. ISBN 978-1-492-03725-5.
4. FLANAGAN, David. *JavaScript. The Definitive Guide*. *Make your webpages interactive!*. Překlad do ruštiny: SYSONIUK, A. G. . Moskwa: „Williams“, 2015, s. 320. ISBN 978-5-8459-1948-9.
5. *Getting Started - What is Git?* [online]. USA: Git--fast-version-control, 2021 [cit. 2021-03-05]. Dostupné z: <https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F>.
6. GLOVER, Andrew. *Spring Boot in Action*. 20 Baldwin Rd, Shelter Island, NY 11964: Manning Publications Co., 2016. ISBN 9781617292545.
7. *H2* [online]. USA: 2020 [cit. 2021-03-05]. Dostupné z: <https://baptiste-wicht.com/posts/2010/08/presentation-usage-h2-database-engine.html>.
8. *IntelliJ IDEA* [online]. ČR: JetBrains, 2021 [cit. 2021-03-05]. Dostupné z: <https://www.jetbrains.com/idea>.
9. JOHNSON, Rod. *Expert One-on-One J2EE Design and Development*. 1. vyd., USA: WROX, 2002. ISBN 978-0-764-54385-2.
10. MARTIN, Robert C.. *Clean Code: A Handbook of Agile Software Craftsmanship.* USA: Pearson Education, 2008, s. 157. ISBN 978-0-13-608325-2.
11. *Oracle. What is a database?* [online]. USA: 2020 [cit. 2021-03-05]. Dostupné z: <https://www.oracle.com/ru/database/what-is-database>.
12. Spring Data JPA. *Spring* [online]. USA: 2020 [cit. 2021-03-05]. Dostupné z: <https://spring.io/projects/spring-data-jpa>.

13. Spring Security. *Spring* [online]. USA: 2020 [cit. 2021-03-05]. Dostupné z: <https://spring.io/projects/spring-security>.
14. *Tomcat* [online]. USA: 2020 [cit. 2021-03-05]. Dostupné z: <http://tomcat.apache.org>.
15. TORVALDS, Linus, DIAMOND, David. *Just for Fun: The Story of an Accidental Revolutionary*. 1. vyd., USA: Harper Business, 2002. ISBN 0-06-662072-4.
16. ULLMAN, Chris. *Beginning Ajax*. USA: WROX, March 2007. Archived from the original on 5 July 2008. Retrieved 24 June 2008. ISBN 978-0-470-10675-4.
17. VAADIN TEAM. *Building Modern Web Apps with Spring Boot and Vaadin* [online]. Finland: Vaadin Ltd, 2019 [cit. 2021-03-05]. PDF format, 10,9MB. Dostupné z: <https://vaadin.com/learn/tutorials/modern-web-apps-with-spring-boot-and-vaadin>.

8. Přílohy

Seznam příloh:

- Příloha github repozitář: https://github.com/studioo/czu-project-chess
- Příloha A: neúplný soubor pom.xml
- Příloha B: částečný kód třídy Board
- Příloha C: částečný kód třídy Game
- Příloha D: částečný kód třídy Move
- Příloha E: enum Color
- Příloha F: částečný kód třídy FigureMoving
- Příloha G: servis Game
- Příloha H: Security konfigurace
- Příloha I: Registration View
- Příloha J: Game View
- Příloha K: Board Component
- Příloha L: Figure Moving Tests
- Příloha M: Game Test

Příloha A: neúplný soubor pom.xml

```
<groupId>com.czu.chess</groupId>
<packaging>jar</packaging>
<vaadin.version>14.2.1</vaadin.version>
 <parent>
        <groupId>org.springframework.boot</groupId>
    </parent>
<dependencies>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
        </dependency>
        <dependency>
            <groupId>com.vaadin</groupId>
            <artifactId>com.vaadin</artifactId>
        </dependency>
        <dependency>
            <groupId>com.vaadin</groupId>
            <artifactId>vaadin-spring-boot-starter</artifactId>
        </dependency>
</dependencies>
```

Příloha B: částečný kód třídy Board

```
@Getter
@Setter
public class Board {
    public String fen;            // FEN-notation
    public Color moveColor;       // Aktuální barva tahu
    public int moveNumber;        // Počítadlo tahů
    private Figure[][] figures;   // Šachovnicová matice 8x8
    public Boolean canCastleA1;   // Indikátor možnosti rošády
    public Boolean canCastleH1;
    public Boolean canCastleA8;
    public Boolean canCastleH8;
    public Square enpassant;      // Čtverec pro braní mimochodem
    public int drawNumber;        // Počítadlo pro pravidlo 50 tahů

    public Board(String fen) {
        this.fen = fen;
        this.figures = new Figure[8][8];
        init();
    }

    private void init() {
        // "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"
        //  0                                          1 2    3 4 5
        String[] parts = this.fen.split(" ");
```

```
        if (parts.length != 6) return;
        initFigures(parts[0]);
        initMoveColor(parts[1]);
        initCastleFlags(parts[2]);
        initEnpassant(parts[3]);
        initDrawNumber(parts[4]);
        initMoveNumber(parts[5]);
    }

    // …

    public Board move(FigureMoving fm) {
        // Vytvoříme objekt Board
        Board next = new Board(this.fen);
        // Provedeme tah na nové šachovnici
        next.setFigureAt(fm.from, Figure.none);
        next.setFigureAt(fm.to, fm.placedFigure());
        // Aktualizujeme všechny potřebné indikátory
        next.dropEnpassant(fm);
        next.updateEnpassant(fm);
        next.moveCastleRook(fm);
        next.updateCastleFlags(fm);
        if (this.moveColor == Color.black) next.moveNumber++;
        next.moveColor = moveColor.flipColor(moveColor);
        // Aktualizujeme FEN a vrátíme aktualizovanou desku
        next.generateFen();
        return next;
    }
}
```

Příloha C: částečný kód třídy Game

```
@Getter
@Setter
public class Game {
    public String fen;
    public Board board;
    public Move move;
    public Player player1;
    public Player player2;
    public Player currentPlayer;
    public boolean isCheck;            // Indikátor šachu
    public boolean isCheckmate;        // Indikátor matu
    public boolean isStarted = false;
    public boolean isStalemate;        // Indikátor remízy

    public boolean isValidMove(String move) {
        FigureMoving fm = new FigureMoving(move);
        if (!this.move.canMove(fm))
```

```
            return false;
        if (this.board.isCheckAfter(fm))
            return false;
        return true;
    }

    public Game move(String move) {
        if (!isValidMove(move))
            return this;

        // Jinak provedeme tah
        FigureMoving fm = new FigureMoving(move);

        Board nextBoard = board.move(fm);
        Game nextChess = new Game(nextBoard, this);
        nextChess.flipCurrentPlayer();

        return nextChess;
    }

    // …

    // Pomocná rekurzivní metoda pro testování
    // https://www.chessprogramming.org/Perft_Results
    static int nextMoves(int step, Game game) {
        // Argument step je zodpovědný za hloubku
        if (step == 0) return 1;
        // Počet všech možných tahů v dalším tahu
        int count = 0;

        ArrayList<String> list = game.getAllMoves();
        for (String moves : list)
            count += Game.nextMoves(step - 1, game.move(moves));

        return count;
    }
}
```

Příloha D: částečný kód třídy Move

```
public class Move {
    FigureMoving fm;
    Board board;

    public Move(Board board) {
        this.board = board;
    }

    public boolean canMove(FigureMoving fm) {
        this.fm = fm;
        return canMoveFrom() && canMoveTo() && canFigureMove();
    }

    boolean canMoveFrom() {
        return fm.from.onBoard() &&
```

```
                Figure.getColor(fm.figure).name() == board.moveColor.name()
                && board.getFigureAt(fm.from).name() == fm.figure.name();
    }

    boolean canMoveTo() {
        Figure figure = board.getFigureAt(fm.to);

        return fm.to.onBoard() &&
                ((fm.from.x != fm.to.x) || (fm.from.y != fm.to.y)) &&
                (Figure.getColor(figure).name() != board.moveColor.name());
    }

    boolean canFigureMove() {

        switch (fm.figure) {
            case whiteKing:
            case blackKing:
                return canKingMove() || canKingCastle();

            case whiteQueen:
            case blackQueen:
                return canStraightMove();
            // …
            default:
                return false;
        }
    }

    private boolean canKingMove() {
        // Kontrolujeme, zda se král pohybuje
        // pouze o jedno pole v libovolném směru
        if (fm.absDeltaX() <= 1 && fm.absDeltaY() <= 1)
            return true;
        return false;
    }

}
```

Příloha E: enum Color

```
public enum Color {
    none,
    white,
    black;

    Color flipColor(Color color)
    {
        if (color.name() == Color.black.name()) return Color.white;
        if (color.name() == Color.white.name()) return Color.black;
        return Color.none;
    }
}
```

Příloha F: částečný kód třídy FigureMoving

```
@Getter
@Setter
public class FigureMoving {
    Figure figure;
    Square from;
    Square to;
    // Když pěšec dosáhne poslední řady
    Figure promotion = Figure.none;
    // ...
    // Metody delta vracejí rozdíl
    // potřebný k výpočtu možnosti tahu
    public int deltaX() {
        return to.x - from.x;
    }

    public int deltaY() {
        return to.y - from.y;
    }
    // Existují také metody, které
    // vrátí absolutní hodnotu deltaX/deltaX
    // ...
}
```

Příloha G: servis Game

```
@Service
@Scope("singleton")
public class GameService {
    private Map<String, Game> games = new ConcurrentHashMap<>();

    public String createGame() {
        Game game = new Game();
        String id = UUID.randomUUID().toString();

        String username =
               SecurityContextHolder
                  .getContext()
                  .getAuthentication()
                  .getName();

        game.setPlayer1(new Player(username, Color.white));
        games.put(id, game);

        // Vrácení identifikátoru hry
        return id;
    }

    // …

    public void setGame(String id, Game game) {
        games.put(id, game);
    }
```

```
    public Game getGameBySessionId(String id) {
        return games.get(id);
    }

    public Map<String, Game> getAvailableGames() {
        return games.entrySet()
                .stream()
                .filter(map -> !map.getValue().isStarted())
                .collect(Collectors
                     .toMap(map -> map.getKey(), map -> map.getValue()));
    }
}
```

Příloha H: Security konfigurace

```
@EnableWebSecurity
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
    @Autowired
    CustomAuthenticationProvider customAuthenticationProvider;

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception
    {
        return super.authenticationManagerBean();
    }



    @Bean
    // Instalujeme RequestCache
    public CustomRequestCache requestCache() {
        return new CustomRequestCache();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
        // Instalujeme AuthenticationProvider
        // za účelem kontroly uživatele na základě údajů z DB
        auth.authenticationProvider(customAuthenticationProvider);
        // Registrujeme testovací uživatele
        auth.inMemoryAuthentication()
                .withUser("user1")
                .password("{noop}password")
                .roles("USER")
    }
```

```
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // Spring CSRF se zde nepoužívá,
        // aby byla možnost HTML pro přihlašovací stránku
        http.csrf().disable()
                // Registrujeme CustomRequestCache,
                // přesměruje uživatele na požadovanou stránku

                .requestCache().requestCache(new CustomRequestCache())
                // Omezujeme přístup k aplikaci
                .and().authorizeRequests()
                // Povolujeme všechny interní dotazy
                .requestMatchers(SecurityUtils::isFrameworkInternalRequest)
                    .permitAll()
                // Povolujeme registraci anonymním uživatelům
                .antMatchers("/" + RegistrationView.ROUTE).permitAll()
                // Povolit všechny dotazy od autorizovaných uživatelů
                .anyRequest().authenticated()
                // Konfigurace přihlašovací stránky
                .and().formLogin()
                    .loginPage("/" + LoginView.ROUTE).permitAll()
                 // Konfigurace odhlášení
                .and().logout().logoutSuccessUrl("/" + LoginView.ROUTE);
    }

    @Override
    public void configure(WebSecurity web) {
        web.ignoring().antMatchers(
                // (development mode) H2 debugging console
                "/h2-console/**",
                // (development mode) static resources
                "/frontend/**",
                // (production mode) static resources
                "/frontend-es5/**", "/frontend-es6/**");
        // ...
    }
}
```

Příloha I: Registration View

```
@Route(value = RegistrationView.ROUTE)
@CssImport("./styles/shared-styles.css")
public class RegistrationView extends FormLayout {
    public static final String ROUTE = "registration";
    private UserService userService;
    private final Binder<User> binder = new Binder<>();

    private TextField usrField = new TextField("Username");
    private PasswordField pwdField = new PasswordField("Password");
```

```
    private Notification notification = new Notification("Registration Error",
4000);

    @Autowired
    public RegistrationView(UserService userService) {
        this.userService = userService;

        addClassName("form");
        usrField.setAutofocus(true);
        usrField.setRequired(true);
        pwdField.setRequired(true);
        binder.forField(usrField).asRequired("User may not be empty")
                .bind(User::getUsername, User::setUsername);
        binder.forField(pwdField).asRequired("Password may not be empty")
                .bind(User::getPasswordHash, User::setPasswordHash);

        RouterLink signInLink = new RouterLink("Sign in", LoginView.class);
        signInLink.addClassName("form__link");

        usrField.addKeyPressListener(Key.ENTER, this::register);
        pwdField.addKeyPressListener(Key.ENTER, this::register);

        add(new H1("Registration"), usrField, pwdField, createButton(),
signInLink);
    }

    private Button createButton() {
        Button button = new Button("Sign Up", this::register);
        button.addClassName("form__submit");
        return button;
    }

    private void register(ComponentEvent event) {
       // Ověření bylo zde vynecháno, aby se zmenšila velikost příkladu
       // ...
        User user = new User();
        if (binder.writeBeanIfValid(user)) {
            userService.saveUser(user);
            UI.getCurrent().navigate(LoginView.class);
        } else {
            notification.open();
        }
    }
}
```

Příloha J: Game View

```
@Push
@Route(value = GameView.ROUTE)
@CssImport("./styles/shared-styles.css")
public class GameView extends Div implements HasUrlParameter<String> {
    public static final String ROUTE = "game";

    private Game game;
```

```
    private String sessionId;
    private GameService gameService;
    private BoardComponent boardComponent;

    @Autowired
    public GameView(GameService gameService) {
        addClassName("container");
        this.gameService = gameService;
    }

    @Override
    public void setParameter(BeforeEvent event, @WildcardParameter String
parameter) {

        Location location = event.getLocation();
        QueryParameters queryParameters = location
                .getQueryParameters();

        Map<String, List<String>> parametersMap =
                queryParameters.getParameters();

        sessionId = parametersMap.get("sessionId").get(0);
        game = gameService.getGameBySessionId(sessionId);

        boardComponent = new BoardComponent(game, sessionId, gameService);
        add(new H1("Game"));
        add(boardComponent);
    }
}
```

Příloha K: Board Component

```
public class BoardComponent extends Div {
    private GameService gameService;
    private Registration broadcasterRegistration;
    String username =
        SecurityContextHolder
            .getContext()
            .getAuthentication()
            .getName();

    // …

    private void broadcast() {
        // ...
        // Vybíráme odesílatele
        // …

        // Pošleme nový FEN jinému hráči a řekneme mu, že jsme provedli tah
        Broadcaster.broadcast(username, game.getFen());
    }

    @Override
```

```
    // Metoda životního cyklu komponentu Vaadin
    protected void onAttach(AttachEvent attachEvent) {
        UI ui = attachEvent.getUI();
        String username = // ...

        broadcasterRegistration = Broadcaster.register(username, newFen -> {
            ui.access(() -> {
                // Na základě údajů přijatých od jiného hráče
                // aktualizujeme stav hry včetně šachovnice
                game = new Game(new Board(newFen), game);
                gameService.setGame(sessionId, game);
                // Aktualizace šachovnici
                updateBoard();
            });
        });
    }

    @Override
    // Metoda životního cyklu komponentu Vaadin
    protected void onDetach(DetachEvent detachEvent) {
        // ...
        // Pozastavíme hru
        // ...
        broadcasterRegistration.remove();
        broadcasterRegistration = null;
    }
}
```

Příloha L: Figure Moving Tests

```
public class FigureMovingTests {
    // ...
    // Vytvoření testovacích objektů je kvůli stručnosti vynecháno
    // ...

    @Test
    public void deltaX() {
        Assert.assertEquals("to.x - from.x must be 0", 0, deltaX);
    }

    @Test
    public void deltaY() {
        Assert.assertEquals("to.y - from.y must be 7", 7, deltaY);
    }

    @Test
    public void signX() {
        Assert.assertEquals("deltaX == 0 must return 0", 0, signX);
        Assert.assertEquals("deltaX > 1 must return 1", 1, signX + 1);
        Assert.assertEquals("deltaX < -1 must return -1", -1, signX - 1);
    }

    @Test
    public void signY() {
        Assert.assertEquals(1, signY);
        Assert.assertEquals(0, signY - 1);
        Assert.assertEquals(-1, signY - 2);
    }

}
```

Příloha M: Game Test

```
public class GameTests {

    /*
     * https://www.chessprogramming.org/Perft_Results
     * */
    @Test
    public void nextMoves() {
        // Initial Position
        Assert.assertEquals(1, Game.nextMoves(0, new
Game("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"
        )));
        Assert.assertEquals(20, Game.nextMoves(1, new
Game("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"
        )));
        Assert.assertEquals(400, Game.nextMoves(2, new
Game("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"
        )));
        Assert.assertEquals(8902, Game.nextMoves(3, new
Game("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1")));

        // Position 3
        Assert.assertEquals(14, Game.nextMoves(1, new
Game("8/2p5/3p4/KP5r/1R3p1k/8/4P1P1/8 w KQkq - 0 1")));
        Assert.assertEquals(191, Game.nextMoves(2, new
Game("8/2p5/3p4/KP5r/1R3p1k/8/4P1P1/8 w KQkq - 0 1")));
        Assert.assertEquals(2812, Game.nextMoves(3, new
Game("8/2p5/3p4/KP5r/1R3p1k/8/4P1P1/8 w KQkq - 0 1")));

        // Position 6
        Assert.assertEquals(1, Game.nextMoves(0, new
Game("r4rk1/1pp1qppp/p1np1n2/2b1p1B1/2B1P1b1/P1NP1N2/1PP1QPPP/R4RK1 w - - 0
10")));
        Assert.assertEquals(46, Game.nextMoves(1, new
Game("r4rk1/1pp1qppp/p1np1n2/2b1p1B1/2B1P1b1/P1NP1N2/1PP1QPPP/R4RK1 w - - 0
10")));
        Assert.assertEquals(2079, Game.nextMoves(2, new
Game("r4rk1/1pp1qppp/p1np1n2/2b1p1B1/2B1P1b1/P1NP1N2/1PP1QPPP/R4RK1 w - - 0
10")));
    }
}
```