Masarykova univerzita
Fakulta informatiky

Generačný upgrade JavaEE aplikácie SMWeb

Bakalárska práca

Norbert Dopjera

Brno, jeseň 2019

Vyhlásenie

Vyhlasujem, že táto bakalárska práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Norbert Dopjera

Vedúci práce: RNDr. Roman Stoklasa, Ph.D.

Poďakovanie

Chcel by som poďakovať RNDr. Romanovi Stoklasovi, Ph.D. za pomoc, odborné vedenie a cenné rady, ktoré mi ochotne poskytol k tejto práci. Cenním si taktiež našu úspešnú spoluprácu na osobných konzultáciach, ktorá výrazne obohatila moje znalosti z informatiky. Ďalej by som chcel s radosťou poďakovať všetkým mojim blízkym, ktorí mi počas práce poskytli podporu a motiváciu ku jej dokončeniu.

Zhrnutie

Cieľom tejto práce bolo navrhnúť a implementovať generačný upgrade webovej aplikácie SMWeb vyvinutej na platforme Java Enterprise Edition a premigrovať obsiahnutú funkcionalitu s využitím moderných spôsobov tvorenia Java webových aplikácii. Generačný upgrade by mal zjednodušiť správu a zložitosť pôvodnej aplikácie SMWeb, pričom jej funkcionalita musí ostať nezmenená. Práca obsahuje prehľad spôsobov tvorenia moderných Java webových aplikácii, analýzu aplikácie SMWeb a jej súčasti, návrhy rôznych spôsobov jej modernizácie a popis výslednej implementácie modernizovanej aplikácie SMWeb, používajúcej vhodne vybraný spôsob tvorenia Java webových aplikácii.

Kľúčové slová

Java, webové aplikácie, JavaEE, Spring, Spring Boot, SMWeb, Maven, JPA, JSF, JMS, REST, WebSocket, aplikačný server, servletový kontajner

Obsah

1 Úvod

SMWeb je webový informačný systém, ktorý bol navrhnutý pre podporu a organizáciu špeciálnych misií leteckého simulátoru *IL-2 Sturmovik: 1946*. Tento simulátor je zasadený do obdobia 2. svetovej vojny a obsahuje pôvodný letecký simulátor *IL-2 Sturmovik*, jeho rozšírenie *Forgotten Battles* a ďalšie rozšírenia. Server tohto simulátoru je možné ovládať pomocou serverovej konzoly a jeho aplikačné rozhranie umožňuje prístup k aktuálnym informáciám o pilotoch, pripojených hráčoch, pozíciach objektov a pod. To umožňuje systému *SMWeb* obohacovať simulátor o ďalšie nové funkcionality.

Systém *SMWeb* je vyvíjaný od roku 2010 a veľká časť jeho implementácie bola vytvorená v rámci študentských bakalárskych prác a to po jednotlivých častiach. Z toho dôvodu sú postupy v jeho implementácii vzájomne nekonzistentné a navyše spolu s použitými knižnicami („libraries“)[1] môžu byť dnes už zastaralé. Počas jeho vývoja bolo pridaných mnoho nových funkcionalít ako napríklad rozšírenie o projekt dynamickej kampane s názvom *ClodWar*. Keďže tieto funkcionality neboli vhodne modularizované, správa závislosti systému *SMWeb* a jeho prehľadnosť sa výrazne zhoršili.

Táto práca sa zaoberá možnosťami modernizácie systému *SMWeb*. Pre tento účel boli naštudované a opísané aktuálne spôsoby tvorenia Java webových aplikácii. Nevyhnutné bolo taktiež analyzovať architektúru systému *SMWeb*, jeho komponenty a technológie na ktorých je systém vybudovaný. Na základe toho boli vytvorené požiadavky, ktoré musí generačný upgrade systému *SMWeb* spĺňať. Praktická časť práce poskytuje tri návrhy modernizácie, čiastočne spĺňajúce tieto požiadavky, podľa ktorých bol vybraný správny spôsob modernizácie systému *SMWeb*. Ďalej poskytuje jeden testovací projekt, ktorý overuje predpoklady kladené na vybraný spôsob modernizácie a samotnú implementáciu modernizácie systému *SMWeb*. Textová časť práce slúži ako dokumentácia modernizácie systému *SMWeb* pre ďalších vývojárov, ktorý sa budú na projekte podieľať a navyše obsahuje zhrnutie spôsobov tvorenia moderných Java webových aplikácii.

Štruktúra textu práce je nasledovná. V druhej kapitole je čitateľ oboznámený so spôsobmi tvorenia rozsiahlych Java aplikačných softvérov. Táto kapitola rozoberá platformu Java Enterprise Edition, jej implementácie, architektúry a ďalšie možnosti tvorenia webových aplikácii. Tretia kapitola detailne analyzuje webový informačný systém *SMWeb*, jeho architektúru, použité technológie a jeho webové rozhranie. V štvrtej kapitole sú popísane

[1] Knižnice v programovacích jazykoch: `https://en.wikipedia.org/wiki/Library_(computing)`

požiadavky, ktoré musí modernizácia systému SMWeb spĺňať, návrhy modernizácie a odôvodnenie výberu správneho spôsobu modernizácie. Piata kapitola sa venuje podrobne technológiám poskytnutým spôsobom modernizácie, ktorý bol vybraný v štvrtej kapitole. Zameriava sa najmä na opis technológii použitých projektom *SMWeb*. Posledná, šiesta kapitola, sa venuje implementácii modernizácie systému SMWeb. Opisuje jej architektúru, výhody a jej záverečný stav. Záverečná kapitola práce obsahuje zhrnutie výsledkov práce, splnených cieľov a možností pokračovania v modernizovaní systému SMWeb.

2 Java Enterprise Edition

Java Enterprise Edition, skrátene nazývaná JavaEE, je platforma postavená na programovacom jazyku Java[1], ktorá poskytuje sadu špecifikácii pre vyvíjanie rozsiahlych, škálovateľných a distribuovaných systémov. Napriek názvu Java Enterprise Edition nie je táto platforma priamo viazaná na komerčnú sféru a je možné ju využiť aj k tvoreniu bežných aplikačných softvérov. Svoje využitie našla najmä v sfére biznisu, ktorý s narastajúcou popularitou v minulosti vytvoril potrebu pre vývoj podnikových systémov. Dnešná popularita platformy JavaEE je ľahko odvoditeľná zo štatistík stránky GitHub, podľa ktorých je programovací jazyk Java najviac populárnym a to najmä vo vývoji webových aplikácii typu klient-server [1, 2]. Takéto aplikácie sú v jazyku Java najčastejšie vyvíjané práve pomocou platformy JavaEE.

Platforma JavaEE je rozšírením platformy Java Standart Edition (JSE) a pridáva k nej sadu špecifikácii aplikačných rozhraní a ich interakcií [3]. Softvér poskytujúci implementácie týchto špecifikácii je potom možné použiť práve pre vývoj systémov postavených na platforme JavaEE. Vzhľadom k požiadavkám modernej spoločnosti, sú často takéto systémy sieťovo prístupné, poskytujú webové služby a teda musia byť spúšťané na vysokovýkonných výpočtových systémoch, serveroch, ktoré disponujú veľkou priepustnosťou. Platforma JavaEE bola pôvodne známa ako Java 2 Platform, Enterprise Editiion (J2EE), dokým nebola vo verzii 5 premenovaná na JavaEE [4]. V texte práce sa naďalej budú spomínať vo všeobecnosti už len verzie pomenované JavaEE.

Vývoj softvéru na platforme JavaEE je výrazne uľahčený vďaka jej minimalizovaniu potreby vytvárania a udržovania metadát mimo zdrojového kódu. K tomu táto platforma využíva zjednodušený programovací model, kde vývojár vkladá informácie do zdrojového kódu pomocou anotácii [5]. Anotácie sú súčasťou platformy Java Standart Edition a vývojárovi poskytujú možnosť definovať závislosti, zdroje, životné cykly, služby a mnoho ďalších metadát priamo na mieste kde sú použité. Takýmto spôsobom je jednoduché udržiavať informácie potrebné pre behové prostredie platformy JavaEE pevne prepojené a synchronizované. Anotácie taktiež redukujú potrebu vkladania kódu, ktorý sa opakovane využíva vo viacerých aplikáciach a môžu navyše eliminovať celé dodatočné súbory, ktoré slúžia len pre definovanie konfigurácii.

[1] Odkaz na online dokumentáciu jazyka Java verzie 8: `https://docs.oracle.com/javase/8/docs/technotes/guides/language/index.html`

Každá verzia platformy JavaEE určuje niekoľko špecifikácii slúžiacich k rôznym účelom, ako napríklad čítanie a zapisovanie do databázy transakčným spôsobom, generovanie webových stránok, správa distribuovaných dotazov, atď. Online dokumentácia poskytuje podrobnejší popis všetkých špecifikácii [6]. Použité špecifikácie definujú typ aplikácie postavenej na platforme JavaEE, pričom najčastejšie sa jedná o webové aplikácie, ktorým sa venuje aj táto práca. Základom webových aplikácii postavených na platforme JavaEE je špecifikácia servletu.

Java servlety sú triedy programovacieho jazyka Java, ktoré dynamicky spracovávajú a odpovedajú na sieťové požiadavky a to synchrónnym alebo asynchrónnym spôsobom [7]. Tieto požiadavky sú typicky prenášane pomocou HTTP alebo HTTPS[2] protokolu a ich účelom je sprístupnenie komunikácie s webovou aplikáciou. Servlety sú za bežných okolností nasadené do servletových kontajnerov, ktoré majú na starosť ich životný cyklus a sieťové rozhranie webovej aplikácie. Tématike servletových kontajnerov a webových aplikácii sa bude ďalej venovať tretia kapitola. Nasleduje výpis tých komponent špecifikácie JavaEE, ktoré sa často spomínajú v texte tejto práce alebo boli použité v jej implementačnej časti. Ostatné komponenty budú opísané až v jednotlivých sekciách, ktoré sa im priamo venujú alebo je možné o nich zistiť viac z bakalárskej práce Martina Janíka [8], prípadne v online dokumentácii [9].

2.1 Špecifikácie a technológie platformy JavaEE

2.1.1 Java Persistence API

Špecifikácia Java Persistence API, skrátene JPA, je aplikačným rozhraním umožňujúcim prístup a správu dát medzi Java objektami a relačnou databázou. Namiesto komunikovania s databázou pomocou SQL[3] príkazov, JPA umožňuje Java triede/objektu použiť objektovo relačné mapovanie (ORM) pomocou anotácii alebo XML súborov, ktoré definujú to ako bude táto trieda mapovaná do tabuľky danej databázy. Takáto trieda sa nazýva perzistentnou entitou a jej inštancie sú reprezentované jednotlivými riadkami databázovej tabuľky. Perzistentné entity môžu používať dedičnosť a polymorfizmus, pričom JPA garantuje, že objekt bude pri načítaní z relačnej databázy inštanciou tej istej triedy ako v momente keď bol do databázy uložený [10]. Vďaka podpore transakcií JPA umožňuje všetkým operáciám spoje-

[2] Hyper Text Transfer Protocol, viac na adrese: `https://en.wikipedia.org/wiki/HTTPS`
[3] Structured Query Language: `https://en.wikipedia.org/wiki/SQL`

ným s prístupom do databáze spĺňať vlastnosti ACID[4]. Referenčnou implementáciou JPA je projekt EclipseLink a ďalšou známou implementáciou je napríklad aplikačný rámec Hibernate [11].

2.1.2 JavaServer Pages

Zatiaľ čo spomínané servlety používajú nízkoúrovňový prístup k spracovaniu HTTP požiadaviek, špecifikácia JavaServer Pages, ďalej ako JSP, poskytuje omnoho prirodzenejší spôsob vytvorenia textového obsahu odpovedi. JSP špecifikácia dovoľuje do skriptov a kódu značkovacích jazykov, napríklad HTML, priamo vkladať JSP tagy, ktoré vytvoria prepojenie stránky s funkcionalitou na pozadí webovej aplikácie [12]. Toto prepojenie je možné vďaka kombinácii špecifikácie expresného jazyka, EL[5], a knižnice tagov, napríklad JSTL[6]. JSP tak umožňuje vývojárovi vytvárať dynamicky generované webové stránky založené na HTML, XML a podobných technológiách. JSP stránka je v dobe použitia zakompilovaná do servletu a teda s ním zdieľa niektoré charakteristiky, ako napríklad schopnosť odpovedať na prichádzajúce HTTP požiadavky. JSP a servlety typicky spolupracujú a tvoria základ starších webových aplikácii.

2.1.3 JavaServer Faces

JavaServer Faces (JSF), je komponentami orientovaný MVC[7] webový aplikačný rámec, ktorý umožňuje vytváranie užívateľských rozhraní webových aplikácii postavených na platforme JavaEE. JSF poskytuje komponenty pomocou spomínanej knižnice tagov, JSTL, ktoré môžu byť použité v JSP stránke alebo inej prezentačnej technológii JavaEE platformy. Tou môže byť napríklad XML súbor, v tomto prípade anglicky nazývaný Facelet[8], ktorý predstavuje prezentačnú šablónu. JSF poskytuje špeciálny servlet FacesServlet, ktorý spracováva požiadavky, načítava príslušné prezentačné šablóny, vytvára strom komponentov, spracováva udalosti a odosiela užívateľovi vykreslenú odpoveď, typicky HTML stránku [13]. JSF narozdiel od JSP, nevkladá do prezentačnej časti aplikácie celé kusy Java kódu, čím rozdeľuje

[4] Viac o databázovych vlastnostiach na adrese: `https://en.wikipedia.org/wiki/ACID`

[5] EL - „Expression language“ (odkaz na dokumentáciu): `https://docs.oracle.com/javaee/5/tutorial/doc/bnahq.html`

[6] JSTL - „JavaServer Pages Standard Template Library“ (odkaz na dokumentáciu): `https://docs.oracle.com/javaee/5/tutorial/doc/bnakc.html`

[7] Dizajnový vzor „Model-View-Controller“: `https://en.wikipedia.org/wiki/Model-view-controller/`

[8] Odkaz na dokumentáciu technológie Facelet: `https://docs.oracle.com/javaee/6/tutorial/doc/giepx.html`

prezentačné technológie od logiky na pozadí aplikácie. To dovoľuje počas vývoja softvéru asynchrónne pracovať na rôznych vrstvách webovej aplikácie a aj preto je dnes hlavnou prezenčnou technológiou JavaEE platformy špecifikácia JavaServer Faces.

2.1.4 Java Messaging Service

Rozsiahly distribuovaný systém často v rámci svojej funkcionality vyžaduje schopnosť komunikácie medzi jeho jednotlivými komponentami alebo s celkom iným systémom. Tejto potrebe sa v JavaEE špecifikácii venuje Java Messaging Service (JMS), aplikačné rozhranie, ktoré poskytuje prostriedky pre vytváranie, odosielanie, prijímanie a čítanie správ. Komunikácia tvorená takýmto rozhraním je medzi komunikujúcimi komponentami volne prepojená, zaručená a asynchrónna [14]. Medzi hlavné výhody používania JMS patrí zvýšenie škálovatelnosti systému a jeho rýchlejšia responzivita na prípadné zmeny. Implementáciou JMS aplikačného rozhrania je *JMS provider*[9], ktorý poskytuje dva režimy pre doručovanie JMS správ. V režime fronty je správa od producenta doručená práve jednému konzumentovi, zatiaľ čo v režime zbernice je správa od producenta doručená všetkým konzumentom prihláseným do určitej domény. Poskytovateľ služby JMS môže byť v pamäti („in-memory“), teda priamo nasadený do implementácie JavaEE aplikácie, ktorá bude spravovať jeho životný cyklus, alebo môže predstavovať externú službu na sieti.

2.2 Implementácie platformy JavaEE

Ako už bolo spomenuté, platforma JavaEE je len sadou špecifikácii aplikačných rozhraní a neposkytuje žiadnu použiteľnú implementáciu. To znamená, že aplikácii postavenej na tejto platforme je nutné navyše dodať implementácie špecifikácii, ktoré pre svoj chod používa. K tomu sa najčastejšie využíva softvér tretích strán, nazvaný aplikačný server. Aplikačný server je sada prostriedkov a nástrojov pre spúšťanie webových aplikácii a serverových prostredí, v ktorých budú tieto aplikácie spustené [15]. Taktiež je možné si ho predstaviť ako kontajner, do ktorého je aplikácia nasadená, a ktorý má na starosť priebeh kódu vykonávajúci logiku tejto aplikácie a jej životný cyklus. Vývojárovi takýchto aplikácii aplikačný server poskytuje sadu využiteľných služieb prostredníctvom aplikačného rozhrania, ktoré je definované samotným serverom.

[9] Poskytovateľ služby JMS, `https://docs.oracle.com/javaee/6/tutorial/doc/bncdx.html`

Služby aplikačného serveru sú implicitne alebo explicitne volané v ňom nasadenou aplikáciou podľa potreby a nie je tak vyžadované ich implementovať v samotnej aplikácii. Tú istú implementáciu aplikácie je tak na základe ňou využitých služieb možné nasadiť na rozdielne aplikačné servery od rôznych dodávateľov.

Architektúra platformy JavaEE poskytuje viacúrovňový distribuovaný aplikačný model, ktorý umožňuje rozdeliť implementáciu logiky aplikačného softvéru do skupín, vzhľadom na poskytnutú funkcionalitu. Táto architektúra definuje 4 následne opísané úrovne platformy JavaEE. Viac k jednotlivým úrovniam a ich technológiám je možné sa dozvedieť v online dokumentácii tu [16].

Úroveň klienta („Client Tier“)
: Úroveň klienta pozostáva z aplikačných klientov, ktorí pristupujú k aplikáciam platformy JavaEE, nasadeným na aplikačnom serveri.

Úroveň webu („Web Tier“)
: Úroveň webu pozostáva z komponentov, ktoré spravujú komunikáciu medzi úrovňou klienta a úrovňou biznisovej logiky.

Úroveň biznisovej logiky („Business Tier“)
: Túto úroveň tvoria komponenty biznisovej logiky, ktorá definuje funkčnosť konkrétnej biznisovej domény, napríklad finančné odvetvie.

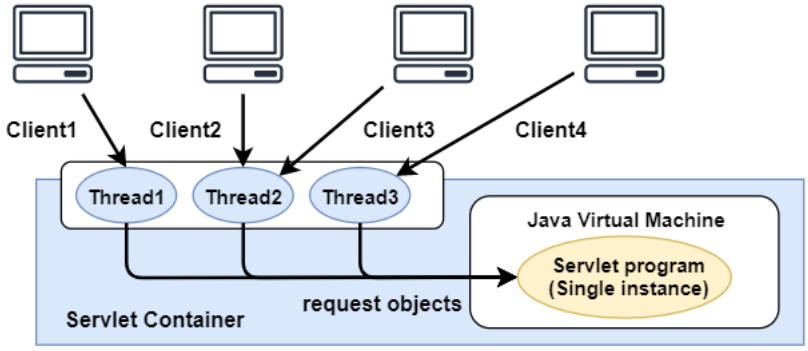
Úroveň informačných systémov („Enterprise Information Systems Tier“)
: Táto úroveň spravuje prístup k dátovým prostriedkom aplikácie postavenej na platforme JavaEE. Tieto prostriedky sú zvyčajne umiestnené na samostatnom výpočetnom systéme a pristupujú k nim komponenty biznisovej logiky.

2.2.1 Servletový kontajner

Nie všetky aplikácie postavené na platforme JavaEE využívajú vo svojej funkcionalite každú súčasť špecifikácie JavaEE, a preto existuje taktiež definícia servletového kontajneru. Servletový kontajner, alebo taktiež webový kontajner[10], je špeciálny typ JavaEE serveru, ktorý ako svoje služby poskytuje implementáciu webovej úrovne platformy JavaEE. Vytváranie jednoduchých a výpočtovo rýchlych Java webových aplikácii je možné vďaka minimalistickej implementácii platformy JavaEE poskytnutej servletovými kontajnermi.

[10] Odkaz na definíciu webového kontajneruhttps://en.wikipedia.org/wiki/Web_container

To ako sú požiadavky zo siete prijímané, parsované a manažované riadi práve implementácia servletového kontajneru, zatiaľ čo úlohou vývojára zostáva implementovať logickú časť aplikácie a to reakcie na jednotlivé požiadavky. Základným prvkom implementácie webového stupňa platformy JavaEE je rozhranie Servlet a typicky z neho dediaca abstraktná trieda HttpServlet[11] deklarujúca metódy, ktoré budú kontajnerom automaticky zavolané pre spracovanie spomínaných požiadaviek. Hlbšie skúmanie servletov a ich špecifických implementácii v tejto práci nie je potrebné, je však dôležité podotknúť, že servletový kontajner často pre spracovanie požiadaviek používa viacero vlákien[12], viď. obrázok 2.1.



Obr. 2.1: Schéma spracovávania požiadavok servletovým kontajnerom.

Tieto vlákna môžu paralelne volať implementované metódy servletu, takže je potrebné správne organizovať ich súbežné volania a synchronizovať prístupy k zdielaným zdrojom. Servletový kontajner ďalej implementuje špecifikácie pre poskytovanie statického obsahu aplikácie, ako napríklad JSF a JSP, ktoré sú často používane na vygenerovanie prezentačnej odpovede pre jednotlivé požiadavky od klienta. Základom JSF a JSP špecifikácii je taktiež servlet, ale v tomto prípade je jeho implementácia poskytnutá priamo servletovým kontajnerom [12, 13].

Servletový kontajner tak implementuje základnú funkcionalitu serverov a to možnosť nasadiť vlastnú aplikáciu do behového prostredia kde bude prístupná pre používateľov, rôzne služby alebo cudzie organizácie. Aplikačné servery na rozdiel od servletových kontajnerov plne implementujú JavaEE špecifikáciu a pridávajú k servletovým kontajnerom navyše ďalšie nástroje vhodné hlavne pri vytváraní komplexných podnikových aplikácii. Tými môžu byť napríklad už spomenuté technológie JPA, JMS a ďalšie špe-

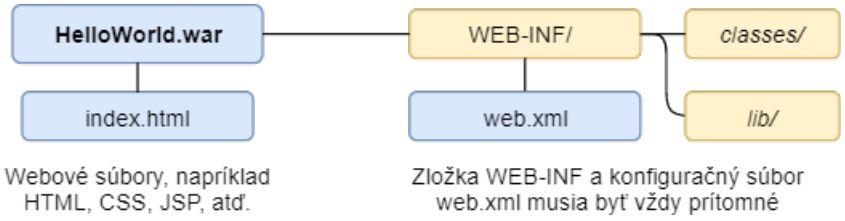
[11] Link dokumnetácie pre triedu HttpServlet a rozhranie Servlet, `https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServlet.html`

[12] Anglicky threads, `https://en.wikipedia.org/wiki/Thread_(computing)`

cifikácie platformy JavaEE. Servletový kontajner síce tieto služby neponúka, avšak nič nebráni aplikácii v ňom spustenej použiť vlastnú implementáciu týchto služieb a to napríklad manuálnym pridaním určitých knižníc. Tento spôsob však vedie ku výraznému zväčšeniu veľkosti samotnej aplikácie nasadenej na server. Java servery sa teda delia na jednoduché servletové kontajnery, taktiež nazývané web servery, a obsiahlejšie aplikačné servery poskytujúce plnú implementáciu platformy JavaEE.

2.2.2 Web Application Resource súbory

Aplikácie vytvorené v jazyku Java sú bežne zabalené do JAR[13] súborov, ktoré využívajú ZIP súborový systém obsahujúci skompilovaný Java kód, jeho zdroje a metadáta o samotnom JAR súbore. Webové aplikácie nasadené na JavaEE serveroch sú zabalené vo formáte špeciálnych Web Application Resource (WAR)[14] súborov, ktoré môžu obsahovať kolekcie viacerých JAR súborov, konfiguračné XML súbory, JSP súbory, statické webové stránky, Java servlety a ďalšie zdroje, ktoré spoločne tvoria webovú aplikáciu. Súborový strom WAR súboru zároveň predstavuje súborový strom webovej aplikácie, preto je zvyčajne súbor *index.html* umiestnený v koreňovom adresári. Bežná štruktúra WAR súboru je zobrazená na obrázku 2.2.



Obr. 2.2: Príklad jednoduchého súboru WAR. Zložka *classes/* obsahuje kompilované Java triedy, napríklad servlety. Zložka *lib/* obsahuje knižnice, teda JAR súbory.

Súbor *web.xml* je používaný najmä servletovým kontajnerom pretože definuje to, do ktorého servletu bude požiadavka od klienta na základe jej URL presmerovaná. Taktiež poskytuje takzvané kontextové premenné, ktoré môžu byť implementáciami servletov použité. WAR súbor je po nasadení na server rozbalený a spustenie aplikácie a jej priebeh má od toho momentu na starosť už len samotný server. Veľkou výhodou WAR súborov je to, že ich podporuje každý JavaEE servletový kontajner a teda Java webová aplikácia môže byť takýmto spôsobom nasadená na ľubovolný JavaEE server [17].

[13] Článok vysvetlujúci JAR súbory, `https://www.geeksforgeeks.org/jar-files-java/`
[14] `https://en.wikipedia.org/wiki/WAR_(file_format)`

Treba si dať však pozor aj na to, že priebeh aplikácie je závislý aj od toho, či je kontajner schopný spustiť Java kód vnútri zložiek *classes/* a *lib/* daného WAR súboru. To, že webová aplikácia bola spustená na jednom servletovom kontajneri ešte nemusí nutne znamenať, že bude spustiteľná aj na inom servletovom kontajneri. Často je totižto v rámci aplikácie potrebné správne nakonfigurovať knižnice v zložke *lib/* tak aby mohla byť webová aplikácia spustená na cieľovom serveri. Zvyšok tejto kapitoly sa bude venovať už len vybraným Java serverom, ktoré boli v rámci implementačnej časti práce použité.

2.2.3 Servletový kontajner Tomcat

Server Apache Tomcat je jeden z najviac populárnych serverov využívaných Java webovými aplikáciami. Tomcat síce neimplementuje úplnú JavaEE špecifikáciu, ale implementuje jej servletovú časť a JSP, a teda sa kvalifikuje ako servletový kontajner [18]. Vďaka jeho jednoduchosti a malej veľkosti (12,8 MB) je možné ho nainštalovať a spustiť behom dvadsiatich sekúnd. V dobe písania implementácie bola použitá verzia 9.0.x[15], ktorá implementuje špecifikácie *Servlet 4.0*, *EL 3.0*, *JSP 2.3* a je kompatibilná s Java verziou 8 a novšími.

Dokumentácia, ktorá je podporovaná rozsiahlou komunitou, je veľmi dobre napísaná a pravidelne obnovovaná. Popularita serveru Tomcat taktiež prispieva ku rýchlejšiemu riešeniu chýb, mnoho náučných zdrojov online a aktívnym diskusným fóram. Server poskytuje jednoduché používateľské rozhranie, ktoré však vo väčšine prípadov nie je potrebné použiť. Existuje aj verzia podporujúca plnú JavaEE špecifikáciu nazvaná TomcatEE. Tá však ani zďaleka nedisponuje tak aktívnou komunitou vývojárov. Konfigurácia serveru Tomcat sa môže začiatočníkom zdať až príliš komplexná, no málokedy je potrebné vôbec pôvodnú konfiguráciu meniť.

2.2.4 Servletový kontajner Jetty

Tak ako Tomcat, webový server Jetty neposkytuje podporu mnohých JavaEE špecifikácii, a taktiež považovaný za servletový kontajner. Napriek tomu, že jeho popularita je oproti serveru Tomcat ďaleko menšia, je dodnes v priemysle široko využívaný. Zatiaľ čo bežné webové servery vo väčšine prípadov poskytujú služby ľuďom, server Jetty našiel svoje uplatnenie v komunikácii najmä medzi strojmi. Najnovšia verzia, 10.x, podporuje servletovú špecifikáciu verzie 4.0 a požaduje minimálnu verziu Java Virtual Machine

[15] Verzie serveru Apache Tomcat: `http://tomcat.apache.org/whichversion.html`

(JVM) 11. Počas písania práce bola táto verzia označená ako nestabilná, a preto sa využila staršia verzia, 9.4[16], ktorá implementuje špecifikácie *Servlet 3.1*, *JSP 2.3* a vyžaduje minimálnu verziu JVM 8.

Server Jetty je menší ako Tomcat a pre svoj beh používa menej operačnej pamäte, čím je kompaktnejší, efektívnejší a viac škálovateľný. Vďaka týmto vlastnostiam je skvelou voľbou v prípade tvorenia aplikácii, ktoré namiesto nasadenia na webový server používajú vlastný vstavaný webový server [18]. Také aplikácie sú najčastejšie službami v mikroservisnej architektúre, ktorej sa táto kapitola venuje neskôr.

2.2.5 Aplikačný server Wildfly

Wildfly, v minulosti známy ako aplikačný server JBoss, je v tejto kapitole prvým spomenutým serverom, ktorý plne implementuje JavaEE špecifikáciu a preto ho je možné považovať za úplný Java aplikačný server [18]. Je to bezplatný aplikačný server s otvoreným zdrojovým kódom („Open Source“), vlastnený firmou RedHat. Počas vyhotovenia implementačnej časti práce bol použitá verzia 17.0[17], ktorá implementuje celú špecifikáciu platformy JavaEE verzie 8.

Keďže firma RedHat je rešpektovaným dodávateľom softvéru, Wildfly poskytuje dobre spracovanú dokumentáciu a garanciu kvality nadchádzajúcich verzií. Spustenie nasadenej webovej aplikácie je možné spustením skriptu `standalone.sh` v priečinku *bin/*, pokiaľ nie je použitý domain mód, ktorý podporuje zhlukovanie viacerých serverov. Server je taktiež možné nakonfigurovať tak, aby použil len webovú úroveň JavaEE špecifikácie, teda aby poskytoval funkcionalitu len servletového kontajneru, čo ho výrazne zrýchli a zároveň skráti dobu potrebnú pre naštartovanie serveru. Základnú konfiguráciu tvorí jediný XML súbor s pár stovkami riadkov a teda je veľmi jednoduché pre nových užívateľov Wildfly server nakonfigurovať.

2.2.6 Aplikačný server GlassFish

Projekt pôvodne slúžiaci ako referenčná implementácia JavaEE špecifikácie, bol vydaný firmou Sun Microsystems. GlassFish je aplikačným serverom s otvoreným zdrojovým kódom („Open Source“), ktorý plne implementuje špecifikácie platformy JavaEE. Ako referenčná implementácia sa GlassFish

[16] Verzie serveru Jetty: `https://www.eclipse.org/jetty/documentation/current/what-jetty-version.html`

[17] Odkaz k verzii Wildfly 17.0: `https://wildfly.org/news/2019/06/10/WildFly17-Final-Released/`

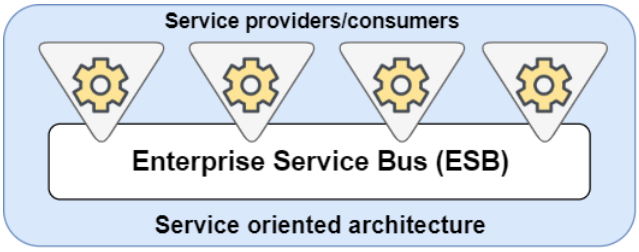
často používa na predstavenie všetkých schopností platformy JavaEE, čo taktiež znamená, že bude vždy ako prvý podporovať jej najnovšie verzie [18]. GlassFish je volne šíriteľným softvérom spadajúcim pod licencie GNU General Public License (GPL)[18] a Eclipse Public License (EPL)[19].

Jeho nevýhodou je nedostatok komerčnej podpory a fakt, že oproti ostatným JavaEE úplným aplikačným serverom je väčší a zložitejší na ovládanie, no napriek tomu poskytuje prívetivejšie užívateľské rozhranie. Zatiaľ čo Wildfly disponuje rozsiahlou dokumentáciou a veľkou komunitou, dokumentácia servera Glassfish je miestami zastaralá a jeho komunita dosahuje len priemernej veľkosti oproti ostatným Java aplikačným serverom. V implementačnej časti práce bola použitá verzia 5.1[20], ktorá úplne implementuje špecifikáciu platformy JavaEE verzie 8.

2.3 Známe architektúry JavaEE aplikácii

2.3.1 Servisne orientovaná architektúra

Servisne orientovaná architektúra (SOA), je štýl softvérovej architektúry, ktorý sa využíva pre vytváranie podnikových aplikácii zložených z jednotlivých volne prepojených komponent. Tieto komponenty rozdeľujú logiku aplikácie do menších celkov, viď. obrázok 2.3, a poskytujú medzi sebou služby pomocou sieťových komunikačných protokolov. Servisne orientovaná architektúra definuje dve základné role, *poskytovateľ* služby a *používateľ* služby. Role spomenutých komponent je možné odvodiť na základe ich funkcionality, pričom je možné aby jedna komponenta predstavovala tieto role na raz.



Obr. 2.3: Servisne orientovaná architektúra je zložená z viacerých volne prepojených komponent komunikujúcich pomocou systému Enterprise Service Bus[22], ktorý môže v platforme JavaEE predstavovať napríklad Java Messaging Service.

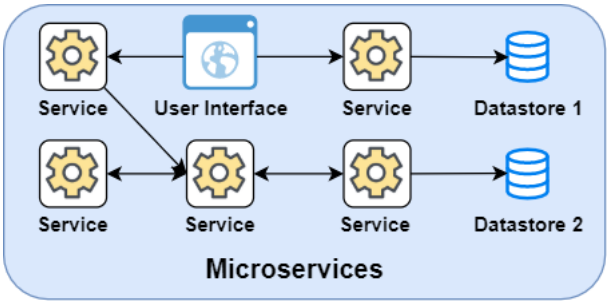
[18] `https://en.wikipedia.org/wiki/GNU_General_Public_License`
[19] `https://en.wikipedia.org/wiki/Eclipse_Public_License`
[20] Glassfish verzie: `https://javaee.github.io/glassfish/download`

Servisne orientovaná architektúra sa skôr ako modularizáciou aplikácie zaoberá jej zostavením integráciou distribuovaných, nasadených a zvlášť udržiavaných softvérových komponent [19].

Táto architektúra bola prvý krát pomenovaná v strede 90. rokov a poskytuje mnoho nasledujúcich výhod pri vývoji softvéru. Jej jednotlivé komponenty je možné vďaka ich voľnému prepojeniu znovu použiť v iných aplikáciach. To prináša aj lepšiu udržatelnosť aplikácie, keďže je omnoho jednoduchšie obnoviť alebo nahradiť jej vzájomne nezávisle časti. Softvér postavený na tejto architektúre ďalej prináša možnosť paralelného a nezávislého vývoja vďaka jeho rozdelenej aplikačnej logike.

2.3.2 Mikroservisy

Mikroservisy sú novým typom servisne orientovaných architektúr, v ktorom sú zložité aplikácie vystavané z malých nezávislých procesov komunikujúcich spolu pomocou jazykovo-agnostických aplikačných rozhraní. Tieto procesy sú navrhnuté tak aby každý poskytoval jedinú špecifickú službu celej aplikácie a sú spolu absolútne nezávisle, čo znamená, že môžu byť vytvorené v inom programovacom jazyku a používať rôzne databázy. Centralizovaný manažment služieb takmer v tejto architektúre neexistuje a namiesto toho mikroservisy komunikujú priamo medzi sebou pomocou jednoduchých HTTP a REST[23] rozhraní.



Obr. 2.4: Schéma mikroservisnej architektúry znázorňujúca komunikáciu, kde šipky predstavujú smer odosielania požiadaviek pre jednotlivé služby.

Tak ako v prípade servisne orientovanej architektúry, jednotlivé služby mikrosevisnej architektúry je možné vyvíjať paralelne s tým rozdielom, že pri vývoji určitej služby nie je potrebná znalosť spoločného komunikačného

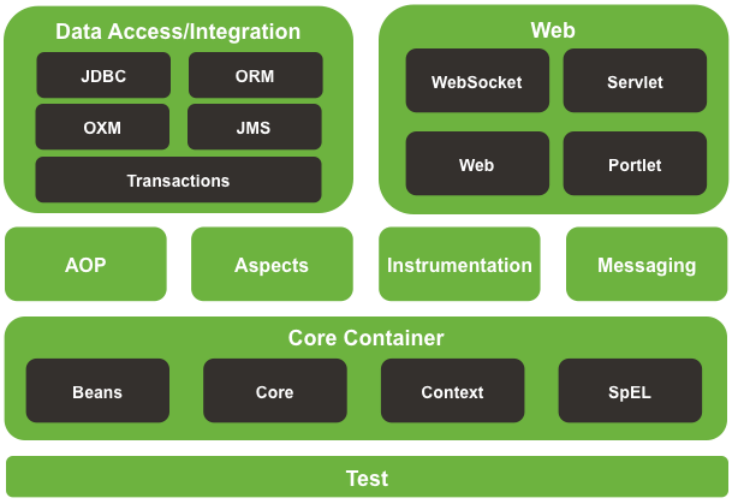
[22] Komunikačný protokol: `https://en.wikipedia.org/wiki/Enterprise_service_bus`
[23] Odkaz ne špecifikáciu webových služieb REST: `https://en.wikipedia.org/wiki/Representational_state_transfer`

mechanizmu. To dovoľuje službám v mikroservisných architektúrach operovať a byť nasadenými absolútne nezávisle od ostatných služieb.

V servisne orientovanej architektúre sa centralizovaný komunikačný systém môže stať jediným bodom zlyhania pre celú aplikáciu. Ak sa jedna zo služieb spomalí alebo zlyhá mohlo by to spôsobiť spomalenie alebo zlyhanie ostatných služieb. Architektúra založená na mikroservisoch disponuje väčšou odolnosťou proti poruchám jednotlivých služieb. Viac o rozdieloch medzi mikroservismi a servisne orientovanou architektúrou je možné sa dozvedieť v článku [20].

2.4 Aplikačný rámec Spring

Spring je aplikačný rámec programovacieho jazyka Java, ktorý poskytuje funkcionalitu a flexibilitu pre vytváranie podnikových aplikácii a rôznych softvérových architektúr podľa potreby. Spring je rozdelený do modulov znázornených na obrázku 2.5, čo umožňuje použitie iba tých častí, ktoré sú pre funkcionalitu aplikácie potrebné. Jeho základnú funkcionalitu je možné použiť pri vývoji akejkoľvek Java aplikácie, ale poskytuje taktiež moduly alebo rozšírenia pre tvorenie webových aplikácii postavených nad platformou JavaEE. Spring síce špecifikácie platformy JavaEE využíva, ale namiesto ich priamej implementácie poskytuje miestami vlastné riešenia. Vďaka tomu nie je vždy nutné aplikácie používajúce Spring nasadiť na JavaEE server.



Obr. 2.5: Prehľad rozdelenia jednotlivých Spring modulov.

Spring poskytuje približne 20 modulov, ktoré sú organizované do väčších celkov znázornených na obrázku 2.5. Bližšie informácie o jednotlivých

celkoch a ich moduloch je možné nájsť priamo v online dokumentácii [21]. Základom funkcionality spomínaných modulov je kontajner inverzného riadenia kontroly („Inversion of Control" [IoC] kontajner)[24], ktorý poskytuje prostriedky pre správu a konfiguráciu Java objektov pomocou reflexie. Kontajner je zodpovedný za manažment životných cyklov špecifických objektov, vytváranie týchto objektov, volanie ich inicializačných metód a konfigurovanie týchto objektov ich vzájomným prepojením.

Objekt vytvorený IoC kontajnerom sa nazýva *bean*[25] a predstavuje týmto kontajnerom manažovaný Java objekt. Kontajner je možné nakonfigurovať pomocou XML súborov alebo použitím Java anotácii v konfiguračných triedach. Metadáta obsiahnuté v týchto zdrojoch určujú definície, ktoré IoC kontajneru poskytujú informácie potrebné pre vytvorenie jednotlivých manažovaných Java objektov. Objekty manažované IoC kontajnerom je možné v prípade potreby poskytnutia závislosti prepájať, čo je proces nazvaný *vkladanie závislosti* (Dependency Injection)[26].

Aplikačný rámec *Spring* je jedným z najviac populárnych Java rozšírení. To za posledné roky viedlo k zvýšeniu jeho komplexity, pridávaním mnohých nových funkcionalít, ktoré výrazne zvýšili čas potrebný pre vytvorenie nového, správne nakonfigurovaného projektu. Z toho dôvodu vznikol projekt Spring Boot.

Spring Boot je aplikačný rámec, ktorého cieľom je uľahčiť vývoj a konfigurácie aplikácii postavených na aplikačnom rámci Spring. Sám o sebe nepridáva do pôvodných Spring modulov žiadnu novú funkcionalitu, ale rozdeľuje ich do ľahšie udržateľných a spravovaných celkov (knižníc jazyka Java). Spring Boot automaticky vytvára konfigurácie aplikačného rámca Spring, ktoré sa pri vývoji mnohých aplikácii neustále opakovali. Taktiež je však možné na základe použitých závislosti, konfiguračných tried a bean objektov tieto konfigurácie prepísať, čo výrazne skracuje čas potrebný pre vývoj aplikácii postavených na aplikačnom rámci Spring.

Webovým aplikáciam poskytuje Spring Boot možnosť využitia vstavaných servletových kontajnerov[27] Tomcat, Jetty alebo Undertow[28], čo eliminuje potrebu nasadenia webovej aplikácie ako WAR súboru na JavaEE server. Vstavaný servletový kontajner je možné využiť najmä pri tvorbe mikro-

[24] Viac k IoC na adrese: `https://en.wikipedia.org/wiki/Inversion_of_control`

[25] Nemýliť si s JavaBean: `https://www.tutorialspoint.com/spring/spring_bean_definition.htm`

[26] Anglicky Dependency Injection: `https://www.jamesshore.com/Blog/Dependency-Injection-Demystified.html`

[27] Využitie vstavaných servletových kontajnerov: `https://www.dynatrace.com/news/blog/the-era-of-servlet-containers-is-over/`

[28] Viac o Undertow na adrese: `http://undertow.io/`

servisnej architektúry, kde je takýmto spôsobom možné jednotlivé webové služby spustiť ako bežné Java aplikácie. To umožňuje ľahšie vytvárať, spúšťať a spravovať takéto služby vo virtualizovaných prostrediach pre správu distribuovaných softvérov ako napríklad v poslednej dobe populárny Docker [22]. Spring Boot hlbšie rozoberá kapitola 5.

2.5 Ďalšie možnosti vývoja Java webových aplikácii

Nad špecifikáciou platformy JavaEE je postavených mnoho ďalších aplikačných rámcov, pričom niektoré poskytujú len jednu špecifickú funkcionalitu, zatiaľ čo iné je možné použiť k tvoreniu celých webových aplikácii. Jedným z takých je webový aplikačný rámec, *Apache Struts*, ktorý tak, ako aplikačný rámec Spring, umožňuje vytváranie komplexných webových aplikácii [23]. Tento framework je rozšíriteľný pretože je postavený na architektúre zásuvných modulov a je možné doňho zakomponovať napríklad moduly aplikačného rámca Spring. Webové aplikácie napísané v programovacom jazyku Java je možné tvoriť aj bez podpory platformy JavaEE. K tomu slúži napríklad aplikačný rámec, *Play*.

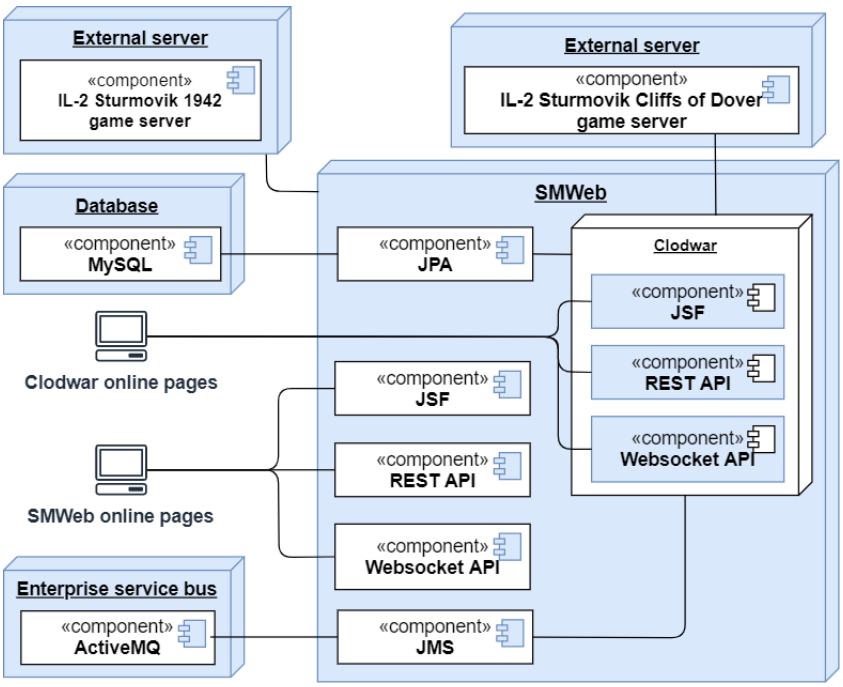
Play je vytvorený v programovacom jazyku Scala[29] a je možné ho použiť v ľubovolnom jazyku, ktorý sa kompiluje do Java bajtového kódu[30]. Scala je funkcionálny a objektovo orientovaný programovací jazyk, a preto tento aplikačný rámec poskytuje aj prvky funkcionálnych paradigiem. Aplikačný rámec *Play* je úplne bez-stavový a postavený na architektúre REST webových služieb. Po podrobnejšom skúmaní webovej aplikácie SMWeb v nasledujúcej kapitole bolo usúdené, že tento aplikačný rámec vôbec nieje vhodný pre jej generačný upgrade, a preto nebude ďalej spomínaný.

[29] Odkaz na oficiálnu stránku jazyka Scala: `https://www.scala-lang.org/`

[30] Množina inštrukcií pre Java Virtual Machine (JVM): `https://en.wikibooks.org/wiki/Java_Programming/Byte_Code`

3 Projekt SMWeb a jeho aktuálny stav

Projekt SMWeb je informačný systém vyvíjaný v jazyku Java na platforme JavaEE. Jeho účelom je rozšírenie leteckých simulátorov IL-2 Sturmovik: *1946*[1] a *Cliffs of Dover*[2] o faktory, ktoré tieto simulátory neposkytujú, ako napríklad rádionavigačné a radarové systémy. Informačný systém SMWeb je vytvorený podľa servisne orientovanej architektúry (vysvetlenej v sekcii 2.3.1), ktorú v prípade systému SMWeb tvorí niekoľko samostatných zásuvných modulov. Každý z týchto modulov sa špecializuje na konkrétnu činnosť a jednotlivé moduly spolu centralizovane komunikujú pomocou implementácie aplikačného rozhrania JMS. Nasledujúci diagram 3.1 znázorňuje hierarchiu systému SMWeb a ním použité komponenty platformy JavaEE.



Obr. 3.1: Hierarchia systému SMWeb a použitých súčasti platformy JavaEE. Tento diagram sa schválne vymyká pravidlám UML diagramov, pretože sa snaží jednoducho zachytiť viaceré súčasti systému SMWeb.

Projekt systému SMWeb je založený v prostredí pre integrovaný vývoj (IDE) Java aplikácii, Netbeans[3]. Implementácie použitých špecifikácii plat-

[1] `https://store.steampowered.com/app/15320/IL2_Sturmovik_1946/`
[2] `https://store.steampowered.com/app/63950/IL2_Sturmovik_Cliffs_of_Dover/`
[3] Oficiálna stránka Netbeans IDE - `https://netbeans.org/`

formy JavaEE mu poskytuje aplikačný server GlassFish, na ktorý je SMWeb nasadený v podobe WAR súboru. V momentálnom stave projektu nie je možné použiť rozdielne IDE alebo iný aplikačný server.

3.1 Databázová vrstva systému SMWeb

Funkcionalita systému SMWeb vyžaduje časté ukladanie dát, ktoré je potrebné uchovávať aj v obdobiach medzi jednotlivými spusteniami systému. Tieto dáta môžu byť napríklad informácie o registrovaných používateľoch, systémových konfiguráciách, moduloch, atď. Pre trvalé ukladanie takýchto informácii používa systém SMWeb relačnú databázu MySQL[4]. Rôzne nástroje pre prácu nad touto databázou, napríklad ORM (definované v sekcii 2.1.1), mu poskytuje JPA aplikačné rozhranie platformy JavaEE a jeho konkrétna implementácia, Hibernate.

MySQL musí pri prvom spustení so systémom SMWeb obsahovať databázu *smweb* a používateľa *smweb*. V Netbeans IDE je potrebné pridať prepojenie *SMWeb2*, ktoré je nie vždy pri prvom spustení úspešne načítané. Taktiež je nutné ručne definovať prepojenia medzi systémom SMWeb a databázou z administratórskeho rozhrania aplikačného serveru GlassFish.

3.2 Webové rozhranie systému SMWeb

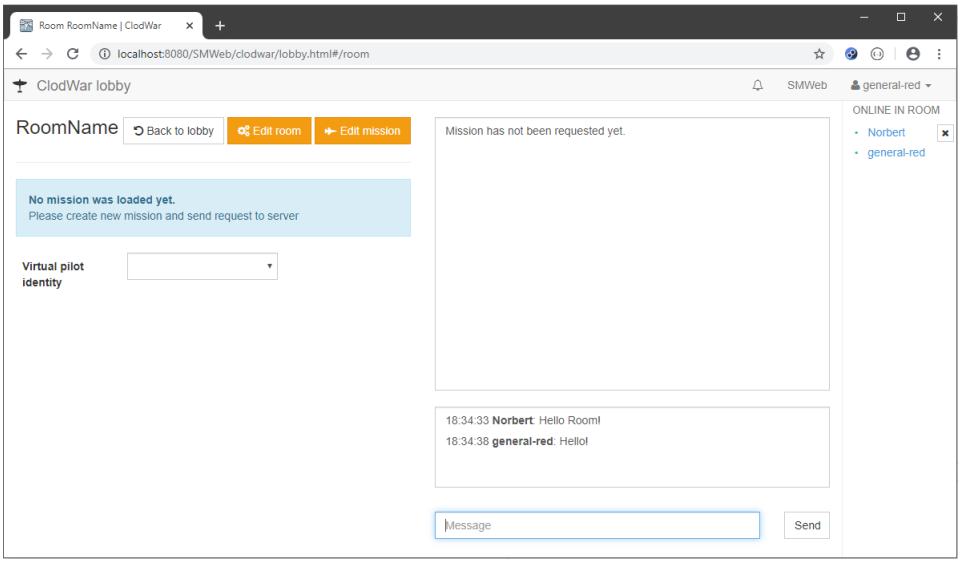
Informačný systém SMWeb poskytuje vo svojej implementácii užívateľské webové rozhranie, ktoré je rozdelené do dvoch častí: *SMWeb* a neskôr pridaný *ClodWar*. Tieto časti sú nezávislé, prezentujú rozdielnu funkcionalitu a poskytujú vlastné registračné a prihlasovacie stránky, čím je možné ich považovať za samostatné webové aplikácie.

Časť *SMWeb* je na základe prístupových práv členená do štyroch sekcií: *verejná sekcia*, *sekcia pilota*, *sekcia generála* a *sekcia administrátora*. K verejnej sekcii má prístup každý a obsahuje napríklad registračný a prihlasovací formulár. Prístupové práva k ďalším sekciám sú určené na základe typu prihláseného používateľa, pričom existujú len typy *pilot*, *generál* alebo *admin*. Podrobnejší opis tejto časti webového rozhrania, jednotlivých sekcií a ich funkcionalít je možné nájsť v práci Mária Kudolániho [24].

Časť *ClodWar*, ako je možné vidieť v diagrame 3.1, poskytuje užívateľské webové rozhranie pre samostatný ClodWar modul, ktorý je implementáciou dynamickej kampane, virtuálnej vojny, pre letecký simulátor *IL-2 Sturmovik: Cliffs of Dover*. Verejná sekcia tejto časti umožňuje napríklad prehliadať

[4] Oficiálna stránka relačnej databázy *MySQL*: `https://www.mysql.com/`

hernú mapu, zobraziť štatistiky pilotov alebo prihlásiť a registrovať používateľa. Po prihlásení sa používateľovi zobrazí prístup k ďalším funkcionalitám ClodWar modulu, ako napríklad aplikácia *ClodWar Lobby*, ktorá organizuje hráčov do miestností (skupín), za účelom plánovania a generovania misií. Táto aplikácia bol vytvorená v bakalárskej práci Jaroslava Rehořku [25] a jej grafické rozhranie je možné vidieť na obrázku 3.2.



Obr. 3.2: Aplikácia *ClodWar Lobby* vytvorená v implementačnej časti bakalárskej práce Jaroslava Rehořku [25].

Webové rozhranie systému SMWeb bolo vytvorené pomocou JSF technológie platformy JavaEE, konkrétne verziou JSF 2. Táto verzia umožňuje v Java kóde používanie anotácii pre zadefinovanie manažovaných Java tried (Managed Beans)[5], pomocou ktorých JSF stránky pristupujú k dátam webovej aplikácie.

3.3 Komunikačná služba Apache ActiveMQ

Apache ActiveMQ[6] zastupuje v hierarchii z diagramu 3.1 poskytovateľa implementácie JMS aplikačného rozhrania špecifikovaného platformou JavaEE (JMS provider, ktorý je definovaný v sekcii 2.1.4). V architektúre systému SMWeb predstavuje Apache ActiveMQ zbernicu, do ktorej moduly systému SMWeb odosielajú novovytvorené správy a taktiež ich z nej odchytávajú.

[5] Java triedy manažované implementáciou JSF - `https://www.tutorialspoint.com/jsf/jsf_managed_beans.htm`

[6] Oficiálna stránka JMS implementácie ActiveMQ - `https://activemq.apache.org/`

SMWeb používa Apache ActiveMQ v režime zbernice, čo umožňuje vysielať JMS správy všetkým klientom prihláseným do danej JMS domény. Jediná JMS doména v systéme SMWeb nesie názov *Modules* a sú do nej prihlásené všetky moduly systému SMWeb, ktoré spolu komunikujú pomocou JMS správ.

3.4 Webová komunikácia so systémom SMWeb

Okrem bežných HTTP a HTTPS požiadaviek tvorených webovými stránkami je možné so systémom SMWeb komunikovať pomocou technológie Websocket a aplikačného rozhrania postaveného na architektúre REST. Hlavným účelom týchto dvoch spôsobov komunikácie je poskytovanie služieb implementáciám webových stránok systému SMWeb, ktorým tieto technológie uľahčujú prácu spojenú s poskytovaním funkcionality systému.

3.4.1 REST webové služby

Webové aplikácie poskytujúce REST (anglicky "REpresentational State Transfer") služby poskytujú v rámci týchto služieb svoje dáta formou informácii o svojich zdrojoch [26]. Zdrojom môže byť akákoľvek informácia, ktorú je webová aplikácia schopná poskytnúť, pričom každý zdroj má svoj unikátny identifikátor (ID). REST služby taktiež umožňujú vykonávať operácie týkajúce sa týchto zdrojov.

Operácie REST webových služieb sú postavené na HTTP požiadavkách typu *GET*, *POST*, *PUT* a *DELETE*. Najčastejšie sú informácie v týchto operáciach prenášané vo formáte zápisu objektov pre JavaScript, nazývaný JSON („JavaScript Object Notation")[7]. Formát JSON umožňuje ľahko analyzovať a jednoducho prenášať dáta po sieti, čo je výhodné najmä pre webové aplikácie vyžadujúce vysokú priepustnosť.

```
1 {"data": {
2   "id": 1,
3   "description": "fruits"
4   "values": ["apple", "orange"]
5 }}
```

Obr. 3.3: Jednoduchý príklad dát prezentovaných formátom JSON.

Webové stránky systému SMWeb používajú REST webové služby napríklad pre synchronizovanie času medzi serverom a prehliadačom klienta

[7] Online dokumentácia k formátu JSON - `https://www.json.org/json-en.html`

alebo pre vyhodenie prihláseného používateľa z miestnosti v aplikácii *ClodWar Lobby*. V systéme SMWeb je v rámci operácii REST služieb vždy použitý len formát JSON.

3.4.2 Aplikačné rozhranie WebSocket

WebSocket je aplikačné rozhranie pre sieťový komunikačný protokol, ktorý poskytuje duplexné komunikačné kanály prostredníctvom jediného TCP/IP spojenia. Vďaka tomu je možné na jedinom HTTP prepojení medzi klientom a webovou aplikáciou otvoriť obojsmernú HTTP komunikáciu. WebSocket eliminuje výpočtovo náročné riešenia asymetrickej HTTP architektúry, ako napríklad neustále sa dotazovanie serveru pre periodické získavanie HTTP odpovedí (HTTP Long polling)[8].

Tak ako v prípade REST webových služieb, dovoľuje aplikačné rozhranie WebSocket zasielanie dát vo formáte JSON, ktorý je použitý ako jediný formát WebSocket správ systému SMWeb. SMWeb používa aplikačné rozhranie WebSocket napríklad k preposielaniu správ medzi používateľmi rovnakej miestnosti v rámci aplikácie *ClodWar Lobby*.

[8] Nástroj pre riešenie problému asymetrickej HTTP architektúry popísaný v tomto článku - `https://www.ably.io/concepts/long-polling`

4 Plánovanie aktualizácie systému SMWeb

Po podrobnom preskúmaní aktuálneho stavu webového informačného systému *SMWeb* bolo v implementačnej časti práci potrebné vhodne naplánovať jeho generačnú aktualizáciu. Preto na základe tohto skúmania vznikla sada požiadaviek, ktoré musí záverečná implementácia dodržať. Tieto požiadavky definujú modernizáciu technológii použitých v systéme *SMWeb* a eliminujú jeho nedostatky.

4.1 Požiadavky na aktualizovaný projekt

Jedným z cieľov tejto práce je modernizácia projektu SMWeb, a preto je nutné definovať vlastnosti, ktoré by mal aktualizovaný projekt spĺňať. Keďže projekt SMWeb bol vyvinutý v programovacom jazyku Java, je potrebné aby jeho aktualizovaná verzia používala posledné stabilné vydanie (LTS) programovacieho jazyka Java, Java 11. Použité komponenty platformy JavaEE by mali spĺňať verzie definované jej najnovším štandardom JavaEE 8[1], najmä *JSF* 2.3. Tieto komponenty je taktiež možné nahradiť inou alternatívou, ktorá ale musí poskytovať rovnakú funkcionalitu a aspoň približne spĺňať špecifickú verziu nahradenej JavaEE 8 komponenty.

Modernizovaný projekt by mal byť nezávislý aspoň od najznámejších Java IDE ako *IntelliJ IDEA*, *Netbeans* a *Eclipse*, čo umožní budúcim vývojárom používať nimi preferované Java IDE.

Po zostavení projektu musí vzniknúť jediný WAR súbor, ktorý bude možné jednoducho nasadiť na nejaký Java aplikačný server alebo servletový kontajner bez dodatočnej konfigurácie. Po nasadení na server musí byť systém SMWeb schopný operovať na ľubovolne nastaviteľnej ceste kontextu (context path)[2]. To znamená, že sa žiadna časť projektu SMWeb nemôže spoliehať na predom zadanú URL, pod ktorou bude systém SMWeb nasadený.

Aktualizovaný projekt by mal používať implementáciu špecifikácie *JPA 2.2*, *Hibernate 5.3* a jej najnovšie postupy pri operovaní na dátovej úrovni aplikácie.

Webové služby REST a aplikačné rozhranie WebSocket by mali používať, dnes už bežne dostupnú, automatickú serializáciu a deserializáciu POJO (Plain Old Java Object) tried, založenú na formáte definovanom systémom *SMWeb*.

---

[1] Plná špecifikácia platformy JavaEE 8: `https://jcp.org/en/jsr/detail?id=366`

[2] Prefix URL adresy pre separovanie kontextu, táto definícia platí všeobecne pre všetky servery: `https://docs.jboss.org/jbossas/guides/webguide/r2/en/html/ch06.html`

Zdrojový kód aktualizovaného projektu musí byť podľa použitia vhodne modularizovaný, čo umožní v ňom ľahšiu orientáciu a jednoduchšiu správu jeho závislosti. Prenesený kód a funkcionalita by mali používať najmodernejšie prístupy a prípadné nájdené chyby musia byť opravené.

4.2 Apache Maven

Apache Maven (ďalej len ako Maven) je nástroj pre správu projektov, založený na koncepte projektovo objektového modelu (POM). Jeho konfigurácia je definovaná XML súbormi s typickým názvom *pom.xml*. Tieto konfiguračné súbory centralizujú informácie na základe ktorých Maven zostavuje, testuje a nasadzuje artefakty[3], ktoré sú prístupné prostredníctvom vzdialeného verzovacieho repozitára. To umožňuje ľahké použitie závislosti v rámci projektu, keďže vývojár definuje len to, ktoré závislosti majú byť použité, zatiaľ čo Maven riadi ich vyhľadanie, stiahnutie a inštaláciu do projektu.

Maven je založený na modulárnej architektúre a vedie k dodržiavaniu štandardov, čo v praxi zvyšuje v projektoch ich konzistenciu a zrozumiteľnosť. Vďaka jeho modulárnej architektúre je možné rozdeľovanie rozsiahlych projektov na menšie ucelené funkcionálne prvky, moduly. Každý modul pritom používa vlastný konfiguračný POM súbor, čo výrazne uľahčuje riadenie konfigurácie celého projektu a umožňuje jednoduchú znovupoužitelnosť modulu na inom mieste. Tento nástroj ďalej poskytuje dedičnosť, ktorá pre každý POM súbor umožňuje špecifikovať rodičovské POM súbory uplatňujúce ich konfiguráciu vo všetkých svojich potomkoch. Vďaka tomu je možné napríklad v modularizovanom projekte vytvoriť pre všetky moduly jeden rodičovský POM súbor, ktorý bude centralizovať verzie všetkých použitých knižníc.

Projekt založený v nástroji *Maven* je možné riadiť v najznámejších Java IDE a zároveň môže byť modularizovaný na základe jeho funkcionality, čím spĺňa požiadavky zo sekcie 4.1. *Maven* bol preto vybratý ako vhodný nástroj pre riadenie aktualizácie systému *SMWeb*. Viac o nástroji Maven, jeho modularizácii, dedičnosti, POM súboroch a ďalších funkcionalitách je možné sa dozvedieť z práce Tatiany Fritzovej [27], ktorou bola táto sekcia inšpirovaná.

4.3 Prototypy modernizovaného projektu SMWeb

Pre ďalšie plánovanie modernizácie *SMWeb* bolo potrebné porovnať jednoduché implementácie konkrétnych spôsobov tvorenia webových aplikácii.

[3] Zabalený projekt dostupný v repozitáry, identifikovaný názvom, skupinou a verziou

K tomu boli vytvorené jednoduché prototypy systému SMWeb, postavené na rôznych spôsoboch tvorenia Java aplikačných softvérov, predstavených v kapitole 2.

Všetky prototypy poskytujú rovnakú funkcionalitu a rovnaké webové rozhranie. Ich účelom je najmä poukázať na rozdiely v implementovaní identickej aplikácie rôznymi spôsobmi tvorenia Java webových aplikácii. Vybrané prototypy prezentujú tieto spôsoby tvorenia Java webových aplikácii: *JavaEE 8*, *Spring Boot 2.1.8* a *Apache MyStruts*.

Úlohou prototypov modernizovaného projektu SMWeb je prezentácia kódu použitého pre ich vytvorenie a na základe toho vybranie správneho spôsobu aktualizácie projektu SMWeb.

Prototypy modernizovaného projektu SMWeb používajú špecifikáciu *JSF 2.3* ako prezentačnú technológiu. V dátovej vrstve prototypov je použitý aplikačný rámec *Hibernate 5.2* v spojení so vstavanou (in-memory) databázou *H2*[4], ktorá uľahčuje prezentáciu prototypu. Do tejto databázy sú pomocou *ORM* ukladané Java objekty reprezentujúce identity zaregistrovaných používateľov, ktoré sa delia na tri typy: *pilot*, *generál* a *admin*.

Prototypy poskytujú jednoduché webové používateľské rozhranie systému SMWeb, ktoré obsahuje *verejnú sekciu*, *sekciu pilota*, *sekciu generála* a *sekciu administrátora*. Do jednotlivých sekcií má prístup len používateľ prihlásený s vhodným typom identity. Tieto sekcie ďalej väčšinou neposkytujú žiadnu funkcionalitu, pretože slúžia len pre prezentáciu zabezpečenia prototypu. Webové rozhranie umožňuje registrovať, prihlasovať a odhlasovať používateľov a v sekcii administrátora je navyše možné registrovať nových generálov. Prototypy *JavaEE 8* a *Spring Boot 2.1.8* demonštrujú vo verejnej sekcii použitie knižnice komponent, PrimeFaces[5], ktorá rozširuje funkcionalitu implementácie JSF.

Každý prototyp je po zostavení zabalený do WAR súboru, ktorý je možné nasadiť na vhodný Java server za použitia ľubovolnej cesty kontextu (context path). Prototypy vždy poskytujú zaregistrovanú identitu typu *admin* s heslom a loginom *admin*, keďže túto identitu nieje možné vytvoriť z webového rozhrania.

4.3.1 Výber správneho spôsobu aktualizácie

Z implementácii jednotlivých prototypov je možné jasne určiť spôsob aktualizovania projektu SMWeb. Prototyp *Apache MyStruts 2*, ďalej len MyStruts,

[4] Prehľad H2 databázy na adrese: `https://www.tutorialspoint.com/h2_database/h2_database_introduction.htm`

[5] Viac o PrimeFaces s implementáciou JSF: `https://www.baeldung.com/jsf-primefaces`

bol zámerne nedokončený, pretože jeho komplexné súčasti vyžadovali až príliš veľa času na ich vyhotovenie. Podporu pre *JSF* je v MyStruts nutné pridať ako zásuvný modul a nebolo ľahké ju správne v projekte nakonfigurovať. Aplikačný rámec *Hibernate* musí vždy pri nasadení prototypu na server navyše zaregistrovať svoju konfiguráciu do servletového kontextu, pretože MyStruts *Hibernate* priamo nepodporuje. Prototyp MyStruts je teda schopný zobraziť len úvodné stránky systému SMWeb a správnu funkcionalitu jeho dátovej vrstvy je možné odôvodniť iba na základe *log*[6] súborov.

Prototypy *JavaEE 8* a *Spring Boot 2.1.8*, ďalej len ako *JavaEE* a *Spring Boot*, sa podarilo úspešne implementovať bez výrazných komplikácii a obidva plne spĺňajú štandard prototypov, definovaný úvodom tejto kapitoly. Hlavným rozdielom týchto prototypov je najmä to, že *Spring Boot* redukuje množstvo kódu potrebného pre vytvorenie rovnakej funkcionality. *Spring Boot* pre vytvorenie jednoduchého JPA kontroléru[7] poskytuje automatické implementovanie jeho operácii (podrobne opísané v sekcii 5.4), zatiaľ čo v prototype *JavaEE* je nutné rovnaké operácie implementovať ručne. Konfiguráciu spojenia aplikačného rámcu *Hibernate* s databázou *H2* definuje prototyp *JavaEE* pomocou dodatočnej konfiguračnej triedy, namiesto použitia konfiguračného XML súboru. Túto triedu *Spring Boot* implementuje automaticky a jednoducho definovanú konfiguráciu spojenia čerpá zo súboru vlastností *application.properties*.

Zabezpečenie je v oboch prototypoch implementované podobným spôsobom s tým rozdielom, že prototyp *JavaEE* musí konfiguráciu zabezpečenia definovať do súboru *web.xml*. *Spring Boot* pre rovnaký účel používa viac prehľadnú konfiguračnú triedu *WebSecurityConfig.java*. Implementáciu JSF je potrebné do prototypu *Spring Boot* pridať manuálne ako závislosť definovanú nástrojom Maven v súbore *pom.xml* ale na rozdiel od prototypu *JavaEE* nie je ďalej nutné definovať konfiguráciu JSF v súbore *web.xml*.

Spôsob tvorenia Java aplikačných softvérov použitý prototypom *Spring Boot* uprednostňuje konvenciu nad konfiguráciu (*Convention Over Configuration*). Tým uľahčuje použitie jeho technológii elimináciou opakujúcich sa konfigurácii, skracuje časové obdobie učenia sa jeho použitia a znižuje počet nutných rozhodnutí pri vývoji softvéru, bez toho aby sa stratila jeho flexibilita [28, 29].

Spring Boot bol na základe týchto vlastností a porovnania s ostatnými prototypmi vybraný ako vhodný spôsob pre aktualizovanie a modernizovanie projektu *SMWeb*.

---

[6] Záznam udalostí serveru `https://www.howtogeek.com/359463/what-is-a-log-file/`

[7] Java trieda poskytujúca prístup k databáze špecifikovanej implementáciou JPA.

5 Spring Boot 2.1.8 a jeho vlastnosti

Aktualizácia projektu *SMWeb* je postavená na aplikačnom rámci *Spring Boot* verzie 2.1.8. Nasledujúce sekcie popisujú jeho vlastnosti a technológie použité v implementácii tejto aktualizácie. Keďže *Spring Boot* od verzie 2.1.0 používa na svojom pozadí aplikačný rámec *Spring 5.1*, sú informácie v celej tejto kapitole čerpané z dokumentácii obidvoch nástrojov, prístupných online tu [30, 31]. Aplikačný rámec *Spring Boot* je podporovaný známymi Java IDE ako napríklad *IntelliJ IDEA* alebo *Eclipse* a je odporúčané *Spring Boot* projektom používať nástroje pre správu závislosti umožňujúce použitie artefaktov z repozitáru *Maven Central*[1]. Sekcie tejto kapitoly pre tento účel predpokladajú práve použitie nástroju *Maven*.

5.1 Správa závislosti a štartéry

Spring Boot štartéry kombinujú skupiny bežných alebo príbuzných závislosti do jednotlivých artefaktov, najčastejšie POM súborov, ktoré definujú všetky závislosti danej skupiny. Funkcionalitu viacerých príbuzných závislosti je tak možné požadovať pomocou konfigurácie závislosti na jediný artefakt. Napríklad pre použitie implementácie celej špecifikácie *JPA* stačí pridať do *Maven* súboru *pom.xml*, konfiguráciu závislosti znázornenú na obrázku 5.1.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
    <version>2.1.8.RELEASE</version>
</dependency>
```

Obr. 5.1: Spring boot štartér poskytujúci *Spring Data*, *Hibernate* a *HikariCP*

Štartéry obsahujú mnoho závislosti, ktoré sú potrebné pre jednoduché a rýchle vytvorenie nového projektu s potrebným prístupom k sade spravovaných tranzitívnych súčasti aplikačného rámca *Spring Boot*. Oficiálne štartéry nasledujú vzor mena *spring-boot-starter-** (* reprezentuje ľubovolné meno konkrétneho štartéru) a sú označené jediným skupinovým identifikátorom *org.springframework.boot* (viď obrázok 5.1), čo umožňuje v nástroji *Maven* vyhľadávanie štartéru jednoducho, podľa mena. Všetky oficiálne štartéry je možné nájsť v online dokumentácii aplikačného rámca *Spring Boot*[2].

[1] Oficiálny repozitár *Maven* artefaktov: `https://maven.apache.org/repository/`

[2] Alebo v *GitHub* repozitári: `https://github.com/spring-projects/spring-boot/tree/master/spring-boot-project/spring-boot-starters`

5.2 Automatická konfigurácia

Aplikačný rámec *Spring* vyžaduje konfiguráciu väčšiny poskytnutých technológii, k čomu je potrebné vytvárať dodatočné XML súbory alebo konfiguračné triedy. *Spring Boot* poskytuje funkcionalitu automatickej konfigurácie, ktorej hlavnou úlohou je redukovanie často opakujúcich sa konfigurácii.

Konfigurácie sú v aplikačnom rámci *Spring Boot* definované *bean*[3] objektami, ktoré sú poskytované konfiguračnými triedami. Konfiguračná trieda je ľubovolná Java trieda označená anotáciou *@Configuration* (viď. obrázok 5.2), naskenovaná pred spustením aplikácie k vytvoreniu jej konfigurácii.

```
@Configuration
public class PriorityConfiguration {
    @Bean(name = "PriorityBean")
    public Integer configurePriority() {
        return 0;
    }
}
```

Obr. 5.2: Jednoduchá konfiguračná trieda, definujúca konfiguráciu vlastného *bean* objektu. Základné *Spring Boot* konfigurácie využívajú nimi špecifikované *bean* objekty.

Pokiaľ *Spring Boot* pri skenovaní pred spustením aplikácie nenájde *bean* objekty definujúce konfiguráciu danej funkcionality, automaticky poskytne ich základnu implementáciu. Takýmto spôsobom vie *Spring Boot* automaticky nakonfigurovať funkcionalitu poskytnutú všetkými štartérmi, čo poskytuje vývojárovi možnosť definovať len tie konfigurácie pri ktorých vyžaduje inú ako ich základnú implementáciu.

Automatická konfigurácia je v *Spring Boot* projekte povolená definovaním *Maven* artefaktu *spring-boot-starter-parent* ako rodiča v konfiguračnom *Maven* súbore *pom.xml*. To mimo spomínané funkcionality prináša automatické riadenie verzií všetkých použitých štartérov a teda nieje ich používateľom nutné definovať označenie verzie z obrázku 5.1.

Automaticky vytvorené konfigurácie používajú vlastnosti definované súborom *application.properties*. V tomto súbore sú vlastnosti uložené ako páry kľúč-hodnota, pričom samotná *Spring Boot* aplikácia môže k požadovanej vlastnosti pristupovať len na základe jej kľúča. Tento spôsob centralizuje hodnoty použité konfiguráciami, čo umožňuje ich jednoduchú správu, prehľadnosť a flexibilitu prípadných zmien v budúcnosti.

[3] Objekty manažované IoC kontajnerom spomenutým v sekcii 2.4

5.3 Inverzia riadenia a vkladanie závislosti

Inverzia riadenia (IoC) je princíp softvérového inžinierstva, ktorý určuje programovací model aplikačného rámca *Spring*. Tento princíp presúva kontrolu riadenia objektov alebo častí kódu z procedurálneho modelu do IoC kontajneru aplikačného rámca. Všetky objekty manažované IoC kontajnerom aplikačného rámca *Spring* môžu pristupovať k ostatným objektom tohto kontajneru pomocou anotácie `@Autowired` alebo definovaním takejto závislosti v konfiguračnom XML súbore. Takýmto spôsobom je možné vkladať závislosti na miesta kde je ich funkcionalita vyžadovaná.

Ako už bolo spomenuté v sekcii 2.4, Java objekty manažované IoC kontajnerom sa nazývajú *bean* objekty a môžu to byť: objekty vrátené metódou označenou anotáciou `@Bean` (viď. obrázok 5.2) alebo celé konfiguračné triedy a komponenty aplikačného rámca *Spring*. Komponenty sú Java triedy, ktoré v IoC kontajneroch poskytujú ostatným *bean* objektom určitý druh funkcionality na základe ich typu, ktorý je určený následne opísanými anotáciami nad Java triedami.

- **@Service** poskytuje služby aplikačnej logiky. Táto komponenta oddeľuje logiku aplikácie od prezentačnej a dátovej vrstvy.
- **@Repository** poskytuje operácie nad databázami a ďalšie funkcionality na úrovni dátovej vrstvy aplikácie.
- **@Controller** umožňuje spracovávať webové požiadavky pomocou metód označených anotáciou `@RequestMapping`.
- **@Component** všeobecne označuje komponentu, ktorá neposkytuje typ funkcionality určený predošlými anotáciami komponent.

Ku všetkým anotáciám definujúcim *bean* objekty je možné pridať anotáciu `@scope`, ktorá svojou hodnotou určuje stratégiu poskytovania daného *bean* objektu. Hodnota *Singleton* určuje IoC kontajneru vytvoriť len jednu inštanciu tohto *bean* objektu, ktorú bude kontajner poskytovať na všetkých miestach označených anotáciou `@Autowired`. Hodnota *Prototype* určuje pre každú anotáciu `@Autowired` vytvoriť novú inštanciu požadovaného *bean* objektu. IoC kontajner v predvolenom nadstavení používa stratégiu *Singleton*.

Inštancie *bean* objektov je možné vkladať priamo do atribútov alebo parametrov metód a konštruktorov, pričom samotné vloženie prebehne až v dobe použitia *bean* objektu. Z toho dôvodu nie je možné pri vkladaní do parametrov konštruktorov použiť cyklické závislosti, kde sa minimálne dva *bean* objekty poskytujú navzájom medzi sebou.

5.4 Repozitáre Java Persistence API

Aplikačný rámec *Spring Boot* poskytuje systém rozhraní repozitárov, ktoré deklarujú operácie nad databázami na základe typu perzistentnej entity (špecifikovanej v sekcii 2.1.1) a typu jej identifikátora. *Spring Boot 2.1.8* umožňuje implementáciu dátovej vrstvy aplikácie podľa špecifikácie JPA 2.2, ktorú implementuje aplikačný rámec *Hibernate 5.3.11*[4] obsiahnutý v štartéri *spring-boot-starter-data-jpa*[5]. Tejto časti aplikačného rámca *Spring Boot* sa venuje *Spring Boot JPA modul*, ďalej len ako *JPA modul*.

Pre splnenie špecifikácie JPA 2.2 je možné použiť v implementáci dátovej vrstvy špecifické rozhranie *JpaRepository*. *JPA modul* podporuje definovanie databázového dotazu manuálne vo forme reťazca alebo jeho odvodením z názvu metódy (viď obrázok 5.3).

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    List<User> findByEmailAndName(String email, String Name);
}
```

.

Obr. 5.3: JPA repozitár nad perzistentnou entitou typu *User* s identifikátorom typu *Long*. *Spring Boot* prekonvertujte meno obsiahnutej metódy na dotaz `"SELECT u FROM USER u WHERE u.email = ?1 and u.name = ?2"` čo umožňuje v implementácii eliminovať prítomnosť „surových“ dotazov

Názvy metód JPA repozitárov sa skladajú z vybranej operácie CRUD[6], ktorú *JPA modul* implementuje automaticky, a voliteľných kľúčových slov. Podporované kľúčové slová v menách metód JPA repozitárov a ich preklad do databázových dotazov je možné nájsť v online dokumentácii tu [32].

Definovanie databázového dotazu formou reťazca je možné označením metódy JPA repozitáru anotáciou `@Query`, ktorej hodnota predpisuje daný dotaz, napríklad `@Query("SELECT u FROM USER u WHERE u.email = ?1")`. Takáto metóda môže niesť ľubovolný názov a tento spôsob vytvárania dotazov je vhodné použiť v prípade kedy by spôsob odvodenia dotazu z názvu metódy nedefinoval požadované kľúčové slovo alebo by zbytočne vytváral neprehľadné meno metódy.

---

[4] Oficiálna dokumentácia verzií 5.3.x: `https://hibernate.org/orm/releases/5.3/`

[5] Obsah štartéru *spring-boot-starter-data-jpa* verzie 2.1.8: `https://repo1.maven.org/maven2/org/springframework/boot/spring-boot-starter-data-jpa/2.1.8.RELEASE/spring-boot-starter-data-jpa-2.1.8.RELEASE.pom`

[6] Základne databázové operácie: `https://stackify.com/what-are-crud-operations/`

V prípade kedy požadovaný databázový dotaz vyžaduje iné ako *JPA modulom* implementované správanie alebo ho nieje možné vytvoriť spomínanými technológiami, je nutné poskytnúť vlastnú implementáciu. Systém rozhraní *Spring Boot* repozitárov umožňuje vytvoriť implementáciu vlastného repozitáru a doplniť o ňu funkcionalitu JPA repozitárov s automaticky implementovanými CRUD operáciami. Možnosť integrácie vlastnej funkcionality do JPA repozitárov bola otestovaná v projekte *smweb-testing-spring-boot-jpa*, pribalenom do prílohy tejto práce, ktorého účelom je testovanie ďalších možností *JPA modulu* ako ORM alebo konfigurácia pripojenia do externej MySQL databázy.

5.4.1 Zabezpečenie na úrovni vlákien

Implementácie *Spring Boot* repozitárov môžu používať *EntityManager* a *EntityManagerFactory* triedy pre operácie s perzistentnými entitami na úrovni perzistentného kontextu[7]. Zatiaľ čo inštancie tried *EntityManagerFactory* sú zabezpečené na úrovni vlákien[8], inštancie tried *EntityManager* nie sú. *Spring Boot* repozitáre sú komponenty, teda *singleton* objekty, ku ktorým môže pristupovať viacero vlákien, a preto by mali požadovať závislosť na inštanciu zabezpečenej triedy *EntityManagerFactory*, ktorá umožňuje vytvárať inštancie triedy *EntityManager*.

Takýmto spôsobom je ale zbytočne v každom repozitáry vytvorená aspoň jedna inštancia triedy *EntityManager*. *Spring Boot* poskytuje anotáciu `@PersistenceContext`, ktorá v spojení s atribútom typu *EntityManager* žiada o transakčný *EntityManager* (nazývaný aj „zdieľaný“), ktorý je zabezpečený na úrovni vlákien a počet jeho inštancií riadi *Spring Boot*. Automaticky implementované metódy JPA repozitárov používajú zdieľaný *EntityManager*.

5.5 Prezentačná technológia JSF

Prezentačná vrstva aplikačného rámca *Spring Boot* nedefinuje použitie konkrétnej prezentačnej technológie, čo dovoľuje vývojárom vytvárať ľubovolné architektúry webových aplikácii. *Spring Boot 2.1.8* umožňuje tvoriť prezentačnú vrstvu aplikácie podľa špecifikácie JSF 2.3, ktorú implementuje štartér *jsf-spring-boot-starter*. Tento štartér nieje oficiálny a je označený skupinovým identifikátorom *org.joinfaces*.

[7] „Persistence Context“ vysvetlený na adrese: `https://www.baeldung.com/jpa-hibernate-persistence-context`

[8] `https://en.wikipedia.org/wiki/Thread_(computing)`

Bean objekty predstavujúce komponenty aplikačného rámca *Spring Boot* popísané v sekcii 5.3, je možné vkladať priamo do JSF stránok, čo nahradzuje funkcionalitu JSF Managed Bean tried (špecifikovaných v sekcii 3.2). Štartér *jsf-spring-boot-starter* umožňuje anotácii `@scope` použiť navyše hodnoty *request*, *session* a *application*, ktoré určujú IoC kontajneru nové spôsoby spravovania *bean* objektov použitých v JSF stránkach.

5.5.1 Knižnica komponent PrimeFaces

Spring Boot 2.1.8 dovoľuje v spojení so štartérom *jsf-spring-boot-starter* použiť knižnicu JSF komponent *PrimeFaces 6.2*. To je možné pridaním závislosti na ďalší neoficiálny štartér *primefaces-spring-boot-starter* zo skupiny *org.joinfaces*. *Spring Boot* automaticky nakonfiguruje verzie týchto závislosti aj napriek tomu, že nie sú oficiálnymi štartérmi.

5.6 Komunikačná služba JMS

Komunikačná služba *JMS* je v aplikačnom rámci *Spring Boot* automaticky poskytnutá pridaním štartéru JMS poskytovateľa (JMS provider) a použitím anotácie `@EnableJms`. Pre využitie možnosti vstavaného (in-memory) JMS poskytovateľa, je nutné k tomuto štartéru pridať aj závislosť na implementáciu JMS poskytovateľa, napríklad *activemq-broker* zo skupiny označenej identifikátorom *org.apache.activemq*.

Spring Boot automaticky konfiguruje JMS tak aby bol použitý mód fronty (definovaný v sekcii 2.1.4). Pre manuálne prijímanie a odosielanie JMS správ poskytuje automatická konfigurácia *bean* objekt triedy *JmsTemplate*. Metódy tejto triedy používajú konvertory správ, ktoré serializujú a deserializujú[9] Java objekty, pričom automatická konfigurácia poskytuje implementáciu konvertoru, ktorý serializuje a deserializuje na základe formátu *JSON*.

Ďalšou možnosťou prijímania JMS správ je anotácia `@JmsListener` označujúca metódu, ktorá bude automatický zavolaná pre spracovanie JMS správy v danej doméne. Táto doména je určená hodnotou atribútu *destination* uvedeného v anotácii `@JmsListener`. Takýto spôsob prijímania JMS správ nieje nutné riadiť ručne a navyše je asynchrónny, čím neblokuje priebeh kódu.

[9] Popis mechanizmu serializácie a deserializácie na adrese: `https://www.geeksforgeeks.org/serialization-in-java/`

5.7 Spring Boot Security

Spring Boot 2.1.8 poskytuje v Java prostrediach verzie 8 a vyšších možnosť zabezpečenia webových aplikácii použitím štartéru *spring-boot-starter-security* a anotácie `@EnableWebSecurity`. Tento štartér definuje automaticky nakonfigurované *bean* objekty, ktoré poskytujú funkcionalitu potrebnú v implementáciách autentifikácie a autorizácie webových aplikácii.

Zabezpečenie webovej aplikácie je možné definovať pomocou konfiguračných tried (triedy označené anotáciou `@Configuration`). Konfigurácia zabezpečenia môže využívať vývojárom definované služby (*Spring Boot* komponenty označené anotáciou `@Service`), ktoré svojou implementáciou určujú model zabezpečenia webovej aplikácie.

Pre bežné použitie autentifikácie a autorizácie je možné implementovať rozhranie služby *UserDetailsService*, ktoré definuje metódy poskytujúce informácie o registrovaných používateľoch. Tieto informácie sú následne použité pri validácii používateľa, ktorý sa chce autentifikovať. *Spring Boot* poskytuje rôzne spôsoby zabezpečenia, popísané v online dokumentácii tu [33].

5.8 REST webové služby

REST webové služby je možné v aplikačnom rámci *Spring Boot* vytvárať pomocou komponent označených anotáciou `@RestController`, poskytnutou štartérom *spring-boot-starter-web*. Metódy týchto komponent definujú aplikačné rozhranie REST služieb pomocou anotácie `@RequestMapping`. Parametre týchto metód definujú obsah prichádzajúcej HTTP požiadavky a odpoveď na ňu definuje ich návratová hodnota.

Anotácia `@RequestMapping` umožňuje definovať URL adresu a HTTP metódu, ktoré sú použité pre mapovanie prichádzajúcej HTTP požiadavky do správnej metódy. Komponenty predstavujúce REST služby používajú automaticky implementovaný konvertor správ, ktorý serializuje a deserializuje Java triedy pomocou formátu *JSON* (viď obrázok 5.4).

5.9 Aplikačné rozhranie WebSocket

Webové aplikácie prezentujúce svoj stav v reálnom čase („*Real-time web*“) je možné v aplikačnom rámci *Spring Boot* implementovať pomocou protokolu aplikačného rozhrania *WebSocket* (definovaného v sekcii 3.4.2). Funkcionalitu tohto rozhrania poskytuje štartér *spring-boot-starter-websocket*. Protokol *WebSocket* je nízkoúrovňový sieťový protokol, operujúci priamo nad prúdom bajtov (byte stream), ktorý neposkytuje žiadne informácie o tom ako by mali

```java
@RestController
public class SimpleRestController {

    @RequestMapping(path = "/numberService",
                    method = RequestMethod.GET)
    public Object getNumber() {
        return new Object() {
            final private Integer number = 5;
            public Integer getNumber() { return number; }
        };
    }
}
```

.

Obr. 5.4: Jednoduchá služba REST, ktorá zachytí HTTP *GET* požiadavku na URL */numberService* a pomocou automatickej serializácie anonymnej triedy odošle HTTP odpoveď s obsahom vo formáte *JSON*: `{ number: 5 }`

byť *WebSocket* správy smerované alebo spracované. Z toho dôvodu umožňuje použitie prihlasovacích protokolov (sub-protocols) operujúcich na aplikačnej úrovni, napríklad protokol *STOMP* používaný aplikačným rámcom *Spring Boot*.

STOMP je jednoduchý, textovo orientovaný protokol, ktorý bol pôvodne vytvorený pre poskytnutie prepojenia so sprostredkovateľmi JMS správ použitých v podnikových systémoch. Vďaka tomuto protokolu môžu klienti a sprostredkovatelia správ vytvorený v rôznych jazykoch spolu komunikovať [34]. Štartér *spring-boot-starter-websocket* umožňuje použitie a nakonfigurovanie sprostredkovateľa *WebSocket* správ využívajúcich protokol *STOMP* pomocou anotácie `@EnableWebSocketMessageBroker`.

Konfigurácia umožňuje použitie vstavaného (in-memory) sprostredkovateľa a určuje koncové body, ku ktorým je možné sa pripojiť pomocou *WebSocket* protokolu. Ďalej je možné definovať domény, do ktorých sa klienti registrujú pre prijímanie správ, podobne ako v prípade módu zbernice komunikačnej technológie JMS (definovaného v sekcii 2.1.4). Konfigurácia taktiež umožňuje použitie núdzovej voľby *SockJS*, ktorá použije alternatívne spôsoby komunikácie v prípade nedostupnosti technológie *WebSocket* u klienta.

Spring Boot spravuje *WebSocket* správy podobným spôsobom ako HTTP požiadavky REST webových služieb. V komponentoch označených anotáciou `@Controller` je možné označiť metódy anotáciami `@MessageMapping` a `@SendTo`, ktoré svojimi parametrami a návratovou hodnotou určujú ob-

sah prijímaných a odosielaných *Websocket* správ. Anotácia `@SendTo` definuje doménu do ktorej bude správa odoslaná a anotácia `@MessageMapping` určuje, ktoré správy majú byť do označenej metódy mapované sprostredkovateľom správ. Odosielanie správ konkrétnemu klientovi je možné pomocou automaticky nakonfigurovaného *bean* objektu implementujúceho rozhranie `SimpMessageSendingOperations`[10]. Tak ako v prípade REST webových služieb sú Java objekty týmito komunikačnými metódami prijímané a odosielané pomocou automaticky implementovaného konvertoru správ, používajúceho formát *JSON*.

[10] Odkaz na dokumentáciu rozhrania: `https://docs.spring.io/spring-framework/docs/5.1.x/javadoc-api/`

6 Implementácia aktualizácie systému SMWeb

Hlavným cieľom implementačnej časti tejto práce je vytvoriť aktualizovanú verziu systému *SMWeb*. K tomu účelu vznikol úplne nový projekt s názvom *SMWeb-NG*. Tento projekt je riadený nástrojom *Maven* a bol implementovaný v Java IDE *IntelliJ IDEA*. Jeho implementácia prenáša časť funkcionality pôvodného systému *SMWeb*, pričom používa najmodernejšie technológie popísané v texte tejto práce. Projekt *SMWeb-NG* je súčasťou prílohy tejto práce a nasledujúci text opisuje jeho architektúru.

6.1 Modularizácia projektu SMWeb-NG

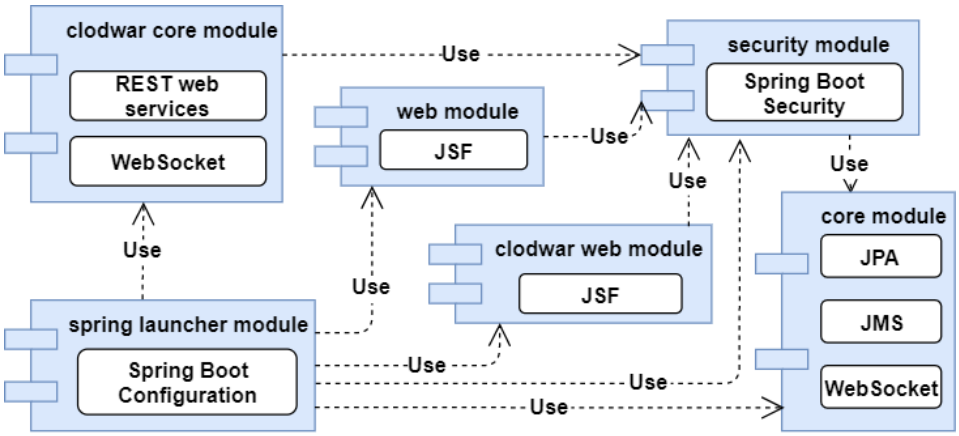
Počas implementácie systému *SMWeb-NG* bola prenášaná funkcionalita, pôvodného systému *SMWeb*, vhodne rozdelená do modulov podporovaných nástrojom *Maven*. Projekt definuje agregačný POM súbor v koreňovom adresári, ktorý manažuje všetky moduly projektu *SMWeb-NG*. Taktiež určuje verziu jazyka Java, verzie všetkých použitých knižníc a definuje *spring-boot-starter-parent* artefakt ako svojho rodiča. Tým sprístupňuje automatickú konfiguráciu aplikačného rámca *Spring Boot* pre všetky manažované moduly, pomocou dedičnosti POM súborov popísanej v sekcii 4.2.

Každý modul definuje vlastný POM súbor, v ktorom špecifikuje svoje závislosti, ktorých verzie sú definované rodičom tohto modulu, spomínaným agregačným POM súborom z koreňového adresáru projektu. Moduly vytvárajú medzi sebou závislosti, pričom nemôžu vytvoriť cyklickú závislosť, pretože by nástroj *Maven* nevedel odôvodniť, ktorý modul zostaviť ako prvý. Schéma závislosti medzi modulmi je znázornená na obrázku 6.1 a nasledujúce sekcie popisujú implementáciu a funkcionalitu jednotlivých modulov.

6.1.1 spring-launcher modul

Tento modul centralizuje konfiguráciu a vlastnosti celého projektu. Definuje triedu `SMWebApplicationLauncher` v balíku `cz.sm.ng.spring.launcher`, ktorá je vstupným bodom systému *SMWeb-NG* a viaže ho s Java serverom, na ktorý bude nasadený. Pomocou anotácii `@ComponentScan`, `@EntityScan` a `@EnableJpaRepositories` skenuje súčasti aplikačného rámca *Spring Boot* obsiahnuté v balíku *cz.sm.ng*, čím registruje funkcionalitu všetkých modulov.

Ďalej poskytuje konfiguračné triedy, ktoré definujú nastavenie zabezpečenia, komunikačnej služby JMS a služby *WebSocket*. Všetky vlastnosti projektu *SMWeb-NG* definuje v súbore *application.properties*, pomocou ktorého



Obr. 6.1: Aktuálna schéma závislosti medzi modulmi systému *SMWeb-NG*. Architektúra závislosti je navrhnutá tak aby *spring-launcher-module* registroval všetky moduly, zatiaľ čo ostatné moduly sa môžu spoliehať aj na tranzitívne závislosti.

je automaticky konfigurované pripojenie do databázy. Tento modul zaobaľuje funkcionalitu ostatných modulov, ktoré sú balené ako JAR súbory, do súboru WAR, so všetkými potrebnými závislosťami. Nasadenie na rôzne Java servery je možné nastavením určitých závislosti tohto modulu ako poskytnutých (pomocou tagu `<scope>`*provided*`</scope>`).

6.1.2 core modul

Tento modul zaobaľuje základnú logiku systému *SMWeb-NG*. Umožňuje riadiť dátovú vrstvu systému *SMWeb-NG* pomocou ORM a JPA repozitárov (popísaných v sekcii 5.4). To odstránilo potrebu prítomnosti pôvodných, zložitých JPA kontrolérov zo systému *SMWeb*. V dátovej vrstve tento modul aktuálne spravuje identity, virtuálnych pilotov, životné cykli, systémové nastavenia a výzbroj lietadla.

SMWeb často používa dizajnový vzor *singleton* (singleton design pattern) ako triedy s privátnym konštruktorom a statickou metódou pre získanie ich jedinej inštancie. Túto funkcionalitu nahradzuje systém *SMWeb-NG* pomocou anotácie `@Service`, ktorá danú triedu sprístupni ako *bean* objekt (*singleton* objekt) na mieste označenom anotáciou `@Autowired`.

Pri prenášaní funkcionality bol kód pôvodného projektu *SMWeb* vhodne modernizovaný, napríklad trieda `SystemConfiguration` refaktoruje opakujúcu sa logiku všetkých jej metód v projekte *SMWeb*. Tento modul ďalej poskytuje službu pre správu vnútorných modulov systému *SMWeb* (opísa-

ných v kapitole 3). Systém *SMWeb-NG* pridáva ku týmto modulom *WebSocket*, ktorý umožňuje vo webovom rozhraní upozorniť klienta na prípadné zmeny stavu modulov.

6.1.3 security modul

Modul zabezpečenia (*security modul*), poskytuje funkcionalitu pre autentizáciu a autorizáciu identít, ktorá je ďalej konfigurovaná modulom *spring-launcher*. Vyhľadáva identity podľa mena v databáze a poskytuje o nich informácie potrebné pre validáciu hesla a ich uloženie do kontextu zabezpečenia (security contet). Po uložení do kontextu zabezpečenia je pri ďalších požiadavkách známe, ktorou identitou sa identifikátor prepojenia (session ID)[1] autentifikoval a autorizoval.

6.1.4 clodwar-core modul

Tento modul zaobaľuje logiku potrebnú pre rozšírenie systému *ClodWar*. Aktuálne poskytuje implementáciu aplikačných rozhraní, pre REST webové služby a technológiu *WebSocket*, použitých aplikáciou *ClodWar Lobby*.

REST služby tejto aplikácie umožňujú spravovať miestnosti a prihlásených používateľov, k čomu využívajú zabezpečenie z modulu *security-modul*. V komunikácii využívajú formát *JSON* pomocou automaticky implementovaných konvertorov správ. Správy v jednotlivých službách zachovávajú rovnaký obsah, aký bol definovaný pôvodným systémom *SMWeb*.

Aplikačné rozhranie *WebSocket* definuje v pôvodnom systéme *SMWeb* tri koncové body. *chat*, *online* a *room*, ktoré umožňujú reprezentovať vo webovom rozhraní reálny stav aplikácie *ClodWar lobby*. Systém *SMWeb-NG*, využíva protokol STOMP, v ktorom sa klienti prihlasujú do určitých domén a teda definuje jediný koncový bod s troma doménami. Tak ako v prípade REST služieb, je v komunikácii, pomocou automaticky implementovaných konvertorov, použitý formát *JSON* a obsah správ je definovaný projektom *SMWeb*.

Oproti pôvodnej implementácii, *Spring Boot* použitím STOMP protokolu v technológii *WebSocket* redukuje veľkosť a zložitosť kódu potrebného pre implementovanie koncových bodov, pôvodného projektu *SMWeb*. Implementácie pôvodných konvertorov správ (REST webových služieb a aplikačného rozhrania WebSocket) systému *SMWeb* poskytujú rovnakú funkcionalitu ako automaticky implementované konvertory správ aplikačným rámcom *Spring Boot*, a preto nebolo potrebné ich funkcionalitu prenášať.

[1] „Session Identifier“: `https://en.wikipedia.org/wiki/Session_ID`

6.1.5 web a clodwar-web moduly

Webové moduly *web* a *clodwar-web* zaobaľujú JSF stránky, komponenty predstavujúce manažované Java triedy (Managed Beans špecifikované v sekcii 3.2) a ostatné zdroje použité webovým rozhraním systému *SMWeb-NG*.

Systém *SMWeb-NG* konfiguruje JSF stránky tak aby bolo možné použiť systém s ľubovolnou cestou kontextu (context path). Tieto moduly sú po zostavení balené do JAR súborov a *Spring Boot* v prípade JAR súboru predpokladá uloženie statických webových súborov pod adresárom `src/main/resources/META-INF/resources`.

Webové stránky aplikácie *ClodWar Lobby* sú spravované samostatným projektom *SM-clodwar-lobby*, vytvoreným v bakalárskej práci Jaroslava Řehořku [25]. Tento projekt musel byť prispôsobený použitiu protokolu STOMP, viď obrázok 6.2. Statické webové súbory tohto projektu sú v minifikovanej podobe použité modulom *smweb-ng-clodwar-web*.

6.2 Záverečný stav projektu SMWeb-NG

Nový projekt *SMWeb-NG* poskytuje webové rozhranie systému *SMWeb*, v ktorom umožňuje nasledovné: autentifikáciu, autorizáciu a registráciu pôvodných typov identít, v admin sekcii správu modulov a nadstavovanie systémových vlastností, v časti *ClodWar* použitie aplikácie *ClodWar Lobby* (popísanej v sekcii 3.2). Všetky ostatné stránky webového rozhrania sú buď prezentované bez použitých dát alebo prezentujú hlavnú stránku danej sekcie. Webové rozhranie je zabezpečené podľa sekcií, v časti *ClodWar* smie do aplikácie *ClodWar lobby* pristupovať len prihlásený používateľ a pri neoprávnenom prístupe je prehliadač presmerovaný na stránky prihlásenia.

Projekt *SMWeb-NG* je možné otvoriť v ľubovolnom Java IDE podporujúcom nástroj *Maven*. Po zostavení projektu vznikne jediný WAR súbor, ktorý obsahuje všetky potrebné závislosti a je možné ho bez dodatočnej konfigurácie nasadiť na aplikačný server *Wildfly 17.0.0*, servletový kontajner *Tomcat 9.0.10* a servletový kontajner *Jetty 9.4.22*. Vďaka podpore vstavaných servletových kontajnerov v aplikačnom rámci *Spring Boot*, umožňuje Java IDE *IntelliJ IDEA* po zostavení projektu jeho automatické spustenie so vstavaným servletovým kontajnerom. To umožňuje výrazne rýchlejšie spustenie projektu (*SMWeb-NG* sa týmto spôsobom spustí do desiatich sekúnd) ako jeho nasadenie v podobe výsledného WAR súboru na lokálny Java server. *SMWeb-NG* je možné použiť s ľubovolnou cestou kontextu (context path) bez potreby jej manuálnej konfigurácie v projekte.

Všetky konfigurácie systému *SMWeb-NG* sú centralizované na jednom mieste a nedefinuje jediný konfiguračný XML súbor. Jeho funkcionalita je závislá na externej databáze *MySQL*, minimálne verzie 5.6 (prvá, stále podporovaná verzia, ktorá používa úložný systém *InnoDB*[2]). Poskytovateľ JMS služby, *Apache ActiveMQ*, je do systému *SMWeb-NG* vstavaný (in-memory), čo robí databázu *MySQL* aktuálne jeho jedinou externou závislosťou.

```
/**
 * Pôvodná funkcia systému SMWeb pre pripojenie klienta
 * k WebSocket koncovému bodu: room.
 */
function start(roomID) {
    if (_.isUndefined(roomID)) {
        roomID = '';
    }

    service.currentRoomID = roomID;
    ws = $websocket(WEBSOCKET_ENDPOINT + 'room/' + service.currentRoomID);
    angular.copy({}, rooms);
    ws.onMessage(proceedIncomingMessage);
}

/**
 * Funkcia nového systému SMWeb pre pripojenie klienta
 * k WebSocket koncovému bodu a jeho prihlásenie do domény room
 */
function start(roomID) {
    if (_.isUndefined(roomID)) {
        roomID = '';
    }

    service.currentRoomID = roomID;
    angular.copy({}, rooms);
    var socket = new SockJS(WEBSOCKET_ENDPOINT);
    ws = Stomp.over(socket);
    ws.connect({}, function(frame) {
        console.log('Connected: ' + frame);
        subHandler = ws.subscribe('/user/clodwar/ws/room/' + roomID,
                                  proceedIncomingMessage);
    });
}
```

Obr. 6.2: Porovnanie pôvodnej funkcie projektu *SM-clodwar-lobby* pre pripojenie k WebSocket koncovému bodu *room* s jej upravenou podobou používajúcou v službe *WebSocket* protokol STOMP a domény `/user/clodwar/ws/room`. Rovnakým spôsobom boli upravené aj funkcie pre pripojenie ku koncovým bodom *online* a *chat*.

[2] *InnoDB*: `https://dev.mysql.com/doc/refman/5.6/en/innodb-introduction.html`

7 Záver

Výsledkom tejto práce je implementácia modernizovaného projektu *SMWeb-NG*, ktorý je postavený na aplikačnom rámci *Spring Boot* a je riadený nástrojom *Maven*. Tento projekt poskytuje modularizovanú architektúru s jednoduchým spôsobom riadenia závislosti. Projekt používa najmodernejšie technológie, a teda eliminoval potrebu udržovania značnej časti kódu pôvodného projektu *SMWeb*. Všetky jeho súčasti je jednoduché nakonfigurovať a spĺňa všetky požiadavky špecifikované v sekcii 4.1, čím uzatvára splnenie zadania tejto práce.

V praktickej časti práce sa nepodarilo preniesť celú pôvodnú funkcionalitu systému *SMWeb*, a preto je ďalej možné na túto prácu naviazať prenesením zvyšnej funkcionality systému *SMWeb*. Modernizovaný systém *SMWeb-NG* používa všetky technológie použité systémom *SMWeb*, čo umožňuje budúcim vývojárom sa zaoberať skôr prenosom funkcionality ako študovaním modernizácie použitých technológii.

Bibliografia

1. GITHUT.INFO. *TOP ACTIVE LANGUAGES: Java* [online] [cit. 2019-11-25]. Dostupné z: https://githut.info/.
2. CHAN, Rosalie. The 10 most popular programming languages, according to the 'Facebook for programmers': Java [online] [cit. 2019-11-25]. Dostupné z: https://www.businessinsider.de/the-10-most-popular-programming-languages-according-to-github-2018-10?op=1.
3. *Differences between Java EE and Java SE: Java EE* [online]. USA: Oracle Corporation [cit. 2019-11-25]. Dostupné z: https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html.
4. *Java Naming: Name Changes* [online]. USA: Oracle Corporation [cit. 2019-11-25]. Dostupné z: https://www.oracle.com/technetwork/java/javase/overview/javanaming-2227065.html.
5. *Annotations* [online]. USA: Oracle Corporation [cit. 2019-11-26]. Dostupné z: https://docs.oracle.com/javase/7/docs/technotes/guides/language/annotations.html.
6. *Java EE 5 Technologies* [online]. USA: Oracle Corporation [cit. 2019-11-26]. Dostupné z: https://www.oracle.com/technetwork/java/javaee/tech/javaee5-jsp-135162.html.
7. *Java Servlet Technology: What Is a Servlet?* [online]. USA: Oracle Corporation [cit. 2019-11-26]. Dostupné z: https://docs.oracle.com/javaee/6/tutorial/doc/bnafd.html.
8. JANÍK, Martin. *Online Shop Web Tier in Java EE* [online]. 2007 [cit. 2019-12-04]. Dostupné z: https://is.muni.cz/th/mdtym/. Bakalářská práce. Masarykova univerzita, Fakulta informatiky, Brno. Vedúci práce Petr ADÁMEK.
9. *The Java EE 6 Tutorial* [online]. USA: Oracle Corporation [cit. 2019-11-29]. Dostupné z: https://docs.oracle.com/javaee/6/tutorial/doc/bnaaw.html.
10. *The Java EE 6 Tutorial: Entity Inheritance* [online]. USA: Oracle Corporation [cit. 2019-12-02]. Dostupné z: https://docs.oracle.com/javaee/6/tutorial/doc/bnbqn.html.

11. JANSSEN, Thorben. What's the difference between JPA, Hibernate and EclipseLink: EclipseLink and Hibernate [online] [cit. 2019-11-29]. Dostupné z: https://thoughts-on-java.org/difference-jpa-hibernate-eclipselink/.

12. *The Java EE 5 Tutorial: JavaServer Pages Technology* [online]. USA: Oracle Corporation [cit. 2019-12-02]. Dostupné z: https://docs.oracle.com/javaee/5/tutorial/doc/bnagx.html.

13. *The Java EE 6 Tutorial: JavaServer Faces Technology* [online]. USA: Oracle Corporation [cit. 2019-11-29]. Dostupné z: https://docs.oracle.com/javaee/6/tutorial/doc/bnaph.html.

14. *Overview of the JMS API* [online]. USA: Oracle Corporation, 2013 [cit. 2019-11-24]. Dostupné z: https://docs.oracle.com/javaee/6/tutorial/doc/bncdr.html.

15. JANSSEN, Thorben. What's the difference between JPA, Hibernate and EclipseLink: EclipseLink and Hibernate [online] [cit. 2019-11-29]. Dostupné z: https://www.itpro.co.uk/strategy/29643/what-is-an-application-server.

16. *Overview of Enterprise Applications: Tiered Applications* [online]. USA: Oracle Corporation [cit. 2019-12-02]. Dostupné z: https://docs.oracle.com/javaee/6/firstcup/doc/gcrky.html.

17. *Learning Java, 4th Edition by Daniel Leuck, Patrick Niemeyer: Reloading web apps* [online]. USA: Safari Books Online [cit. 2019-12-02]. Dostupné z: https://www.oreilly.com/library/view/learning-java-4th/9781449372477/ch15s03.html.

18. ZUKANOV, Vasiliy. Top Java Application Servers: Tomcat vs. Jetty vs. GlassFish vs. WildFly: Tomcat vs. Jetty vs. GlassFish vs. WildFly [online] [cit. 2019-12-04]. Dostupné z: https://stackify.com/tomcat-vs-jetty-vs-glassfish-vs-wildfly/.

19. *Chapter 1: Service Oriented Architecture (SOA)* [online]. USA: Microsoft [cit. 2019-12-02]. Dostupné z: https://web.archive.org/web/20160206132542/https://msdn.microsoft.com/en-us/library/bb833022.aspx.

20. *Microservices vs. SOA — Is There Any Difference at All?: So, Where's The Difference?* [online]. USA: medium.com [cit. 2019-12-02]. Dostupné z: https://medium.com/@kikchee/microservices-vs-soa-is-there-any-difference-at-all-2a1e3b66e1be.

21. *2. Introduction to Spring Framework: Part I. Overview of Spring Framework* [online]. USA: docs.spring.io [cit. 2019-12-03]. Dostupné z: `https://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/html/overview.html`.

22. *Spring Boot Docker: A Basic Dockerfile* [online]. USA: Pivotal Software [cit. 2019-12-03]. Dostupné z: `https://spring.io/guides/topicals/spring-boot-docker/`.

23. *From a birds eye: The Apache Struts web framework* [online]. USA: Apache Software foundation [cit. 2019-12-03]. Dostupné z: `https://struts.apache.org/birdseye.html`.

24. KUDOLÁNI, Mário. *SMWeb - návrh a implementace komunikačního rozhraní kompatibilního s dvěma leteckými simulátory* [online]. 2015 [cit. 2019-12-04]. Dostupné z: `https://is.muni.cz/th/jwqr6/`. Bakalářská práce. Masarykova univerzita, Fakulta informatiky, Brno. Vedúci práce Roman STOKLASA.

25. ŘEHOŘKA, Jaroslav. *SMWeb - doplněk pro organizaci hráčů virtuální války v leteckém simulátoru* [online]. 2015 [cit. 2019-12-04]. Dostupné z: `https://is.muni.cz/th/z2efq/`. Bakalářská práce. Masarykova univerzita, Fakulta informatiky, Brno. Vedúci práce Roman STOKLASA.

26. *CHAPTER 5: Representational State Transfer (REST)* [online]. USA: ics.uci.edu [cit. 2019-12-06]. Dostupné z: `https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm`.

27. FRITZOVÁ, Tatiana. *Maven plugin na generovanie kódu pre UAS Forms* [online]. 2019 [cit. 2019-12-08]. Dostupné z: `https://is.muni.cz/th/kofv8/`. Bachelor's thesis. Masaryk University, Faculty of Informatics, Brno. Vedúci práce Tomáš PITNER.

28. PARASCHIV, EUGEN. How Spring Boot Can Level Up your Spring Application: The Spring Ecosystem [online] [cit. 2019-12-09]. Dostupné z: `https://stackify.com/spring-boot-level-up/`.

29. SÁEZ, Francisco. Convention Over Configuration [online] [cit. 2019-09-12]. Dostupné z: `https://facilethings.com/blog/en/convention-over-configuration`.

30. *Spring Boot Reference Guide: 2.1.8.RELEASE* [online]. USA: Pivotal Software [cit. 2019-12-09]. Dostupné z: `https://docs.spring.io/spring-boot/docs/2.1.8.RELEASE/reference/htmlsingle/`.

31. *Spring Framework Documentation: Version 5.1.12.RELEASE* [online]. USA: Pivotal Software [cit. 2019-12-09]. Dostupné z: https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/.

32. *5.3.2. Query Creation: Table 3. Supported keywords inside method names* [online]. USA: Pivotal Software [cit. 2019-12-09]. Dostupné z: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.

33. *30. Security* [online]. USA: Pivotal Software [cit. 2019-12-09]. Dostupné z: https://docs.spring.io/spring-boot/docs/2.1.8.RELEASE/reference/htmlsingle/#boot-features-security.

34. *4.4. STOMP: 4.4.1. Overview* [online]. USA: Pivotal Software [cit. 2019-12-09]. Dostupné z: https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/web.html#websocket.