VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

Fakulta informačních technologií
Faculty of Information Technology

BAKALÁŘSKÁ PRÁCE
BACHELOR‘S THESIS

Brno, 2017 Vojtěch Průša

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

LOGOVACÍ BOT PRO IRC

LOG BOT FOR IRC

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Vojtěch Průša

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Radek Burget, Ph.D.

BRNO 2017

Abstrakt

Pro tvorbu webových aplikací je v dnešní době možné využít celou škálu webových technologií. Starší technologie byly nahrazeny novými a proto je potřeba držet krok s aktuálními trendy ve vývoji aplikací z důvodu bezpečnosti, spolehlivosti a s nízkou spotřebou zdrojů. Cílem této práce je vytvořit funkční aplikaci IRC logovacího bota, který podporuje záznam více kanálů, umí rotovat záznamy a dohledávat v nich užitím klíčových slov a návrh vhodného uživatelského rozhraní. Využité technologie by měly být programovací jazyk Scala, JavaScriptová knihovna React, kontejnerový nástroj Docker, platforma OpenShift. Popsat využití technologií tak, aby bylo možno jednoduše vytvořit jiné aplikace na podobném principu užitím moderních webových technologií.

Abstract

For web application development it is currently possible to use the full range of web technologies. Older technologies have been replaced by new ones and we need to keep up with the current trend in application development for safety, reliability and low resource consumption. The aim of this work is to create a functional IRC logging boot application that supports multichannel recording, rotate records and search them with keywords and design a suitable user interface. The technologies used should be the Scala programming language, the React JavaScript library, the Docker container tool and the OpenShift platform. Describe the use of technologies to make it easy to create other applications on a similar principle using modern web technologies.

Klíčová slova

IRC, logovací bot, Scala, Play framework, ReactJS, WebSocket, Docker, OpenShift

Keywords

IRC, log bot, Scala, Play framework, ReactJS, WebSocket, Docker, OpenShift

Citace

Vojtěch Průša: Logovací bot pro IRC, bakalářská práce, Brno, UIFS VUT v Brně, 2017

Logovací bot pro IRC

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením za VUT Radkem Burgetem a za Red Hat, Inc. Jiřím Kremserem.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vojtěch Průša
15. 5. 2017

Poděkování

Děkuji v první řadě rodině za podporu při studiu. V druhé řádě vedoucímu práce z VUT a zadavateli a kolegům v Red Hat. V neposlední řadě kamarádům za odreagování. V předposlední řadě kolegům v první řadě lavic přednáškové místnosti D105. A v poslední řadě toastovači.

Obsah

1 Úvod

Webové aplikace jsou v dnešní době rozšířeným produktem a pomocníkem ve velké oblasti internetových technologií a internetových služeb. Slouží jako prostředek pro prezentaci jak lidí tak jejich výrobků nebo služeb, kterými se tito lidé zaobírají. Technologie pro tvorbu takových webových aplikací je stále v rozvoji a vyvíjí se ruku v ruce s potřebami spolehlivosti, rychlosti aplikací, jejich bezpečnosti a znovupoužitelnosti. Webové technologie se za poslední dekádu dost změnily, to co bylo dříve dostatečné je dnes zastaralé a je doporučeno se některým technologiím a postupům tvorby webových aplikací vyhnout, ať už z důvodu zabezpečení nebo rychlosti aplikace nebo její tvorby. Pracovní postupy pro tvorbu takových aplikací se také změnily a zdokonalily. Hlavním cílem této práce je pomocí moderních pracovních postupů vytvořit aplikaci, která využívá technologie moderních webových rozhraní a byla by v praxi také využitelná, ať už jako aplikace nebo jako veřejná funkční kostra pro aplikace jiné. Tento dokument popisuje tvorbu takové aplikace s jejím praktickým aspektem IRC klienta a robota zaznamenávajícího události IRC klienta. Zadavatelem této práce je firma Red Hat, Inc.[1] definující technologie, které je v této práci nutné využít (Scala, IRC, ReactJS, WebSocket, Docker, OpenShift).

Následující text se věnuje popisu a prostředkům tvorby takové aplikace, problematik se kterými je možné se při vývoji webových aplikací setkat. Tento text se také věnuje analýze zadání od dodavatele a specifikaci funkčních částí takové aplikace.

První kapitola je úvod. V druhé kapitole je rozepsány základní pojmy: Webová aplikace, HTML, programovací jazyky pro tvorbu webových aplikací, IRC. Třetí kapitola je zaměřena na popis využitých technologií, popisuje jazyk Scala a jeho možnosti, seznámí se softwarem Docker a platformou OpenShift. Čtvrtá kapitola se věnuje implementaci samotné aplikace, řešení reálných problému při vývoji a testování. Pátá kapitola obsahuje závěr a zhodnocení výsledků.

1 Oficiální web: https://www.redhat.com

2 Základní pojmy

V následujících kapitolách jsou rozebrány základní pojmy, vysvětleny jejich souvislosti a prerekvizity pro vytvoření podobné aplikace, kterou se tato práce zaobírá. Webové technologie se neustále inovují a proto jsou v pojmech rozebrány starší verze technologií pro pochopení verzí nových.

2.1 Webová aplikace

Webová aplikace je aplikace, která je poskytnuta uživatelům z webového serveru užitím počítačové sítě. Webové aplikace lze rozdělit podle toho jestli jsou spuštěné ve webovém prohlížeči nebo využívají vlastní prostředí pro spuštění. Oblíbenější jsou aplikace spustitelné přímo ve webovém prohlížeči neboť nepotřebují dodatečný software pro jejich běh. Dnes jsou také používány aplikace typu klient-server, které potřebují nainstalovat na osobní počítač klientský program a na straně serveru spuštěnou příslušnou část aplikace starající se o přijetí komunikace. Aplikace klient-server však mají vyšší nároky na podporu. Webové aplikace oproti aplikacím využívajícím klientský software generují HTML obsah přímo v prohlížeči.

2.2 HTML

Je v dnešní době standardní značkovací jazyk určen pro tvorbu webových stránek a aplikací. Byl poprvé vydán v roce 1993, ale snaha o to mít jazyk pro zobrazení textových dat přišla již dříve pod názvem SGML [1]. Pro každou verzi HTML jsou definovány značky (dále jen tagy) a jejich povolené atributy. Tagy se rozdělují na jednoduché a párové. Jednoduchý tag je například <*br*>, který reprezentuje zalomení řádku. Párový tag obklopující text nebo další tagy určuje jak se s jeho obsahem bude pracovat.

Od prvního vydání se jazyk několikrát a dost změnil. V roce 1995 byla vydána verze HTML 2.0 a v následujících několika letech se vydaly další verze, v některých byly zveřejněny nové tagy a doporučení na tvorbu webových stránek, v dalších verzích byly odebrány zastaralé tagy (např. tag iframe již není doporučeno používat. Webové aplikace, které využívají Flash Player mohou mít bezpečnostní rizika, aj.). V roce 2014 byla zveřejněna verze HTML 5.0, jejíž prvky se zaměřili na nová zobrazovací zařízení. Umožňuje zobrazení na široké škále přenosných zařízení a možnost integrace jako aplikace běžící přímo na operačních systémech. S touto verzí byla také přidána multimediální podpora a nová podpora pro vyhledávací stroje. V předchozí verzi se pro zobrazení multimediálních dat na webové stránce používal software třetí strany (např. Flash Player), který ale nutil uživatele instalovat dodatečný software a v některých případech toto dělalo aplikace nedostatečně zabezpečené a mohlo poskytnout neautorizovaný přístup k zařízení toto využívající. Vyhledávací stroje s HTML 5.0 mohou dohledat obsah stránky dle navigace na stránce a tím zvýšit kvalitu vyhledávání. HTML 5.0 je stále pouze značkovací jazyk a pro animace, manipulaci s multimediálním obsahem je nutné využívat CSS3 styly a JavaScript. HTML 5.0 je konstruován tak, aby ho bylo možné zobrazit i na starších prohlížečích nebo aby na starších prohlížečích alespoň nezobrazoval chyby.

2.3 HTTP

Hypertext Transfer Protokol je protokolem aplikačním vrstvy. Slouží k distribuci dat na internetu a je to základní komunikační prvek WWW [2]. Vývoj tohoto protokolu začal roku 1989 v CERN [2] a jeho první běžně užívaná verze vyšla roku 1997 pod názvem HTTP/1.1. V roce 2015 byl standardizován jeho následovník HTTP/2, tato verze je dnes podporována většinou webových serverů. HTTP funguje na bázi dotazů a odpovědí v prostředí klient-server. Klientem je obvykle webový prohlížeč, ale může se jednat také o konzumenta určité služby webového serveru (např. aplikace přijímající data z REST API). Protokol obvykle běží na portu 80. Rozšířením tohoto o zabezpečení TLS [3] dostaneme HTTPS. HTTPS obvykle běží na portu 443. HTTP dotaz je složen (dle RFC2616 [4]) z hlavičky (nebo hlaviček) a těla zprávy. Pro zrychlení přenosu dat se zpráva pošle po síti po částech, http hlavička pak obvykle dojde k serveru dříve než tělo zprávy. Obsahuje-li pak server s aplikací podporu rozlišení zpráv dle hlavičky (Play toto podporuje), pak je možno v čase před přijetím zbytku zprávy a po přijetí hlavičky, určit z dat hlavičky o jaký typ dotazu se jedná. Toho lze využít při rozlišení zda se jedná o REST API dotaz, pro který je volána databáze, AJAX dotazem, pro který je nutné vykreslit data do zobrazení, nebo pro zobrazení stránky se statickým obsahem, která nepotřebuje využívat vstupně-výstupní operace modelu aplikace. Serverová aplikace pak reaguje rychleji na klientské dotazy chytrým způsobem alokace zdrojů:

- vytvoření databázového spojení při příchodu hlavičky, z které lze rozlišit nutnost daného modelu,
- žádost o manipulaci se souborem (logu),
- načtení zobrazení ze souboru do paměti serveru, která se v něm mají vykreslit, před příchodem dat v těle dotazu.

Protokol HTTP také poskytuje možnost vytvoření sezení. Sezení zajišťuje trvalost spojení v rámci více dotazů a odpovědí mezi klientem a serverovou aplikací. Jednotlivá sezení jsou na straně serveru rozlišena pomocí unikátního identifikátoru předávaného v HTTP hlavičkách (podvržení tohoto identifikátoru je na straně serveru bráněno identifikací také pomocí IP adresy klienta a dalšími technikami). Díky sezení lze pak udržovat a měnit stav v jakém se klientská aplikace nachází a odpadá nutnost autentizovat každý dotaz pomocí uživatelského jména a hesla. Tímto nicméně vzniká bezpečnostní riziko falšování dotazů (angl. cross-site request forgery [5]) využívající bezpečnostních možných děr v prohlížeči. To je řešeno například pomocí CSRF žetonu.

2.4 CSS

Kaskádové styly (angl. Cascading Style Sheets) je jazyk sloužící pro popis způsobu zobrazení dokumentů napsaných ve značkovacích jazycích. Byl vydán roku 1996 a je udržován organizací W3C (World Wide Web Consortium). Je využíván většinou webových stránek k vizualizaci a k lepší čitelnosti jejich obsahu. Cílem je rozdělit oddělit popis struktury a obsahu dokumentu od popisu jeho vzhledu. Soubory kaskádových stylů mají příponu *.css*. Do HTML dokumentu mohou být vloženy přímo jako atribut tagů dokumentu, použití atributu *style*, párovým tagem *<style>...</style>* nebo jako přiložený soubor tagem *<link rel="stylesheet" href="cestaKCssSouboru.css"/>* v hlavičce HTML souboru. Výhody CSS:

- Oddělení popisu vzhledu od popisu struktury a obsahu dokumentu.
- Vzhled je snadno změnitelný.

2 Evropská organizace pro jaderný výzkum, oficiální web: http://home.cern

- Prohlížeči trvá načtení stránky kratší dobu a CSS soubory jsou na straně serveru i v prohlížeči ukládány do paměti *cache [6]*.
- Udržuje konzistenci vzhledu stránek webu uložením CSS jako samostatného souboru přiloženého k jednotlivým stránkám.

Nevýhodou CSS někdy bývá špatná podpora u prohlížečů, obzvláště jejich starších verzí, zapříčiněná změnou standardů s jednotlivými verzemi CSS a jejich nedodržením v prohlížečích. Z tohoto důvodu je obtížné zařídit, aby vzhled stránky byl stejný na všech prohlížečích.

2.5 JavaScript

JavaScript je vysoce dynamický, beztypový a za běhu interpretovaný programovací jazyk a je to v dnešní době nejrozšířenější nástroj pro psaní komplexních front endových aplikací pro webové prohlížeče. Lze ho implementovat i na straně serveru (např. NodeJS), pak se bavíme o serverovém (angl. server-side) JavaScriptu. S jazykem Java má kromě podobnosti v názvu také určité podobnosti v syntaxi, ale v ohledech interpretace, paralelismu, definice a práce s datovými typy se zásadně liší. Vznikl v roce 1995 a za svou existenci zaznamenal několik znatelných změn, vylepšení a standardizací pro použití v širší škále služeb, přibylo množství knihoven a změnily se postupy tvorby aplikací v tomto jazyce, často s ohledem na konkurenci a paralelismus v aplikací. Jeden z posledních trháků ve webové technologii zahrnující JavaScript je knihovna ReactJS. Jedním z hlavních využití ve webové aplikaci je AJAX [7] (asynchronní JavaScript a XML). AJAX je sada technik pro vývoj webových aplikací na straně klienta (webového prohlížeče). Využívá objektu *XMLHttpRequest [8]* k zasílání a přijetí dotazů aplikaci na webovém serveru. Využívá *zpětné volání* (angl. callback [9]), kdy při přijetí asynchronní (vůči zbytku klientské aplikace) odpovědi serveru změní obsah webové stránky. Pro zamezení nekonzistentních stavů, které vznikají asynchronní náturou této technologie, je možno využít více technik. Front endové frameworky se těmto problémům vyhýbají nucením programátorů postupovat při tvorbě aplikace určitým bezpečným způsobem (ReactJS využitím hierarchického uspořádání komponent).

2.6 IRC

Celým názvem Internet Relay Chat (chatování přes internet) je protokol sloužící pro textovou komunikaci v síti internetu fungující na modelu klient-server. Uživatel přistupuje do sítě IRC pomocí klienta, který se připojí na server. Implementací klienta je v dnešní době celá řada (mIRC, Irssi, Xchat, aj.) implementující minimálně základní specifikaci nutnou k komunikaci s IRC serverem, identifikaci uživatele a správu kanálů. IRC síť vznikne spuštěním IRC serveru a jeho případným propojením s jiným IRC serverem. Tato webová technologie byla vymyšlena Jarkko Oikarinen roku 1988 a rozrostla se v následujících letech. Po roce 2003 byl zaznamenán úpadek užívání IRC. V průběhu vývoje se IRC sítě několikrát rozdělily z důvodu neshod v tom, kam by se měl vývoj této technologie ubírat. Od roku 2016 se s novou snahou standardizace vytvořila pracovní skupina IRCv3, s cílem dalších rozšíření o notifikace, lepší správu historie a zlepšení zabezpečení.

2.6.1 Specifikace

IRC protokol využívá protokol TCP a případě TLS. IRC server obvykle využívá jeden port v rozmezí 6990-7000. Klient IRC zasílá příkazy na server a přijímá na ni následně odpovědi. Pro zasílání zpráv mezi více uživateli se využívají kanály. Kanál je cíl na straně IRC serveru, na který lze poslat zprávy. Každý název kanálu má prefix dle typu jeho typu:

- ‚#‘ pro kanály v síti IRC,
- ‚&‘ pro lokální kanály serveru,
- ‚+‘ bezmódový kanál (pro server spuštěný v takovémto módu),
- ‚!‘ pro případ kanálu využívající časové razítko v IRC síti časová razítka nevyužívající.

Administraci na IRC serverech mají na starosti IRC operátoři, kteří mají na serverech zvýšená oprávnění. Tito operátoři zajišťují dobře běžící a čistou IRC síť. Snaha tvůrců protokolu IRC je taková, že operátoři nebudou nutní, protože údržba sítě nebude potřeba a IRC síť bude bezporuchová. Toho však zatím docíleno nebylo.

Zabezpečení může být pro privátní síť pomocí VPN. V konfiguraci serveru lze nastavit globální heslo, hesla pro jednotlivé kanály. Některé servery také podporují SSL/TLS. Jelikož je IRC veřejným komunikačním nástrojem, tak nelze všechny tyto bezpečnostní politiky použít. Některé servery poskytují možnost použití v kanálech značky *+S* povolující pouze SSL spojení.

2.6.2 Roboti

IRC robot (nebo pouze bot) je program, který spustí IRC klienta, jenž se přihlásí na IRC server. IRC robot obvykle zprostředkovává dalším klientům připojeným na server nějakou službu nebo má určitou funkci. Robotů pro IRC je již mnoho a zřizují služby jako:

- odesílání automatických zpráv,
- přidělování práv autentizovaným uživatelům,
- záznam zpráv,
- zabránění převzetí kanálu neoprávněným uživatelem,
- řešení uživatelské podpory,
- spouštění dodatečných akcí nebo programů při daných událostí (např. právu databáze).

2.7 Git

Je systém řízení verzí pro sledování změn v souborovém systému. Je oblíbeným nástrojem pro uchování zdrojových kódů projektů jednotlivců a kolektivů. Git byl vydán roku 2005 a jedná se o software zdarma distribuovaný pod licencí GNUv2 [10]. Pro tuto aplikaci byl vybrán díky své spolehlivosti, dobré podpoře a jednoduché integraci do projektů. Jako hlavním veřejným úložištěm zdrojových souborů je GitHub [11], který jak název napovídá Git využívá a poskytuje integraci s dalšími testovacími nástroji a přívětivé uživatelské rozhraní pro začátečníky i zkušené vývojáře.

2.8 Počítání v Cloudu

Pojem *cloud [12]* byl poprvé použit roku 1977 pro síťového výpočetního vybavení v síti ARPANET [13]. Cloud je model využívání sdílených výpočetní zdrojů, dat a jejich správou. Dává možnost uživatelům a společnostem využití výpočetního výkonu vzdálených strojů pro běh jejich aplikací a ukládání a zpracování dat. Uživatel cloudu se díky tomu nemusí starat o správu počítačové infrastruktury, to za něj dělají správci cloudu. Cloudová aplikace je taková, že ji lze na cloudu spustit za využití sdílených systémových prostředků a umožňuje škálovatelnost. Dalším cílem cloudu je sdílení technologií v cloudových aplikacích. Hlavní sdílenou technologií je pak virtualizace, díky ní mohou běžet aplikace fungující na rozdílných technologiích „*vedle sebe*“ v kontejnerech.

3 Použité technologie

V následující kapitole jsou popsány jednotlivé technologie použité pro návrh a tvorbu této aplikace. Jsou zde zmíněny nástroje a knihovny, mezi kterými jsem vybíral při návrhu tvorby této aplikace a rovněž vysvětleny důvody jejich výběru a popis jejich využití.

3.1 Scala

Scala [14] je univerzální škálovatelný jazyk, to znamená, že je navržen tak, aby rostl s nároky jeho uživatelů, programátorů. Je také klasifikován jako jazyk multiparagmatický a byl navržen tak, aby integroval rysy objektově orientovaného i funkcionálního programování.

Poprvé se objevil v roce 2004 pro platformu Java a na rozdíl od jazyku Java, který v té době byl zaměřen hlavně na objektové programování, přinesl výhody programování funkcionálního, které bylo pro Javu představeno až s verzí Java 8 podporující lambda výrazy. Spustitelnost aplikací ve Scale na JVM dává možnost využití kompilace typu právě-včas (angl. Just-in-time [15]), čímž je docílena rychlejší odezva aplikací na změny v kódu.

3.2 ScalaJS

ScalaJS je kompilátor Scaly do JavaScriptu a přidání o podporu pro využití JavaScriptových knihoven. ScalaJS dává možnost vývojářům pracujících v jazyku Scala psát front endové aplikace, back endové aplikace využívající NodeJS a jiný serverový jazyk se syntaxí JavaScriptu.

ScalaCSS knihovnou ScalaJS určená k tvorbě, užívání, údržby a validity CSS s typovou bezpečností. Typová bezpečnost dává možnost využití skokům k definicím stylů ve vývojovém prostředí pro samostatné CSS zdroje i pro jednořádkové definice, bezpečný přístup ke klíčům a hodnotám Scala Map a jiných kolekcí. Umožňuje tvořit styly podmíněné stavem jiných komponent, tím zjednodušuje využití pseudotříd. Při kompilaci jsou vygenerován jediný CSS dokument obsahující všechny styly v rámci celé aplikace do veřejně přístupného adresáře serveru (obvykle adresář *public/css/*). Tento CSS dokument je možno přidat do dokumentu pomocí tagu *<link />* v hlavičce HTML dokumentu.

3.3 ReactJS

ReactJS je veřejná JavaScriptová knihovna, která je udržovaná částečně společností Facebook a Instagram a dále komunitou samostatných vývojářů. Využívá koncept MVC se snahou, aby knihovna byla rychlá, jednoduchá a snadno rozšiřitelná. Jendou z hlavních přednosti aplikací napsaných s využitím této knihovny je to, že odpadá nutnost opětovného načtení stránky v prohlížeči, protože se všechny komunikace děje na úrovni AJAX dotazů. ReactJS aplikace si vytvoří nejprve virtuální DOM, pomocí kterého mapuje své komponenty, jednotlivé komponenty jsou pak do něj vykresleny s danými daty předanými jako statické vlastnosti, komponenty se mohou měnit pomocí volání asynchronních zpráv, které mohou změnit kontext komponenty, která je vyvolala. Tento koncept toku dat komponentami se nazývá jednosměrný tok dat (angl. one-way_data_flow [16])

Díky této struktuře předávání dat mezi komponentami se dobře pracuje s čekáním na data, o které aplikace požádá AJAX dotazem a následně čeká, než server vyhodnotí a zašle odpověď.

Komponenty jsou psány pomocí JSX [17], proto aby mohly obsahovat a využívat HTML tagy. ScalaJS-React je veřejný projekt [18], který se snaží vývojářům aplikací v ScalaJS dát možnost využití knihovny ReactJS se všemi jejími výhodami.

3.4 Scala vs Java

Scala má silnou typovou kontrolu a návrh tohoto jazyka vychází z výtek k programovacímu jazyku Java [19]. Stejně jako Java je kompilován do Java bajtkódu (angl. bytecode), díky kterému je možno spustit programy napsané ve Scale jakémkoliv virtuálním stroji Javy [20]. Syntaxí připomíná kombinaci Javy a C. Scala má funkcionální podporu podobnou Haskellu nebo Standard ML například tzv. currying, typové rozhraní, vyhodnocování výrazů při výstupních operacích a vzorovou shodu.

Oproti programovacímu jazyku Java má například možnost přetěžovat nejen metody tříd, ale také operátory, možnost volitelných parametry metod, implicitní parametry metod a tříd, předávání parametrů dle jejich jmen, aj.

Je považováno, že Scala je těžší jazyk než Java, ale napomáhá vývojářům psát lepší kód a zvyšuje jeho výkon, myšleno vnucením používání určitých způsobů práce s datovými strukturami a algoritmy. Syntaxe jazyka Scala [21] je méně upovídaná než syntaxe Javy. Při programování ve Scale je dobré dbát ohledy na využití paměti [22], která může být paměťově neohleduplným funkcionálním programování rychle vyčerpána.

3.5 Front a back end

Webové aplikace z pohledu programátora lze rozdělit dle technologických prostředků k její tvorbě. Jedná se o rozdělení prezentační vrstvy aplikace a vrstvy aplikace, která zajišťuje přístup k datům:

1. Front end (uživatelské rozhraní)
    - Definuje a implementuje jak se data zobrazí, strukturu stránky a dává možnost uživateli interakci s obsahem.
    - V prostředí klient-server se jedná o část klienta.
    - Pro webové aplikace se pro jeho tvorbu využívá HTML, CSS, JavaScript a jeho knihovny (pro tuto aplikaci ReactJS, Jquery).
2. Back end (jádro, logické rozhraní)
    - Definuje operace a transformace nad daty, zabezpečení, autentizaci a autorizaci, zálohování, databázové spojení, stará se o vstupně-výstupní operace, určuje jaká data vykreslí front end.
    - V prostředí klient-server se jedná o server.
    - Tato část webové aplikace je psána v Scala, Java nebo také PHP, Python, Perl, Nodejs, aj.

3.6 MVC

Model-view-controller (Model - zobrazení - kontrolér) je ve vývoji softwaru návrhový vzor pro implementaci uživatelských rozhraní. Jeho užitím lze dosáhnout efektivní znovupoužitelnosti kódu. Obvykle je použit pro tvorbu grafických uživatelských rozhraní webových aplikací. Frameworky jej často využívají, protože je možné v nich psát hned po stažení a rozbalení. Je-li v balíčku nastaven přístup k modelu, frameworky často obsahují výchozí komponenty, na které lze pak následně nabalovat pochody dat z modelu do zobrazení.

Model se stará o operace s a nad daty, nejčastěji pracuje přímo s databází nebo jinou vstupně-výstupní složkou aplikací. Zařizuje spojení s databází, vstupně-výstupní operace. Pro vytvoření modelu pro operace s databází je nutné si uvědomit možnosti kontrolérů a REST API.

Kontrolér určuje, které objekty modelu budou vytvořeny při daných HTTP požadavcích a jaké metody těchto objektů jsou volány, toto se děje v metodách akcí (angl. *Action*). Je úzce spjat s směrovačem starajícím se o nastavení cest HTTP požadavků k jednotlivým metodám akcí.

Zobrazení jsou sada znovupoužitelných šablon (ang. template), které řeší to jak se data zobrazí ve webovém prohlížeči. Jednotlivé šablony jsou v Play frameworku soubory s příponou *html.scala* uchované v balíčku serverové aplikace *views*. V kontrolérech aplikace jsou do zobrazení směrována data získaná modelem. Šablonový stroj Play je kompilátor souborů šablon do funkcí jazyka Scala. Je vytvořen s myšlenkou být kompaktní, expresivní a plynulý, tedy napomoci rychlému a plynulému psaní šablon. Také umožňuje v šablonách přistupovat k proměnným a metodám projektu a generovat tak obsah šablony dynamicky. Znovupoužitelnost šablon je zajištěna možností vkládat šablony do sebe. Zkompilované šablony jsou postupně vykreslovány při odpovědi serveru. Hlavní šablona obsahuje základní strukturu stránky, úvodní párové tagy HTML, nastavení metadat, import stylů a scriptů a využití dalších šablon.

Směrovač (Router) je komponenta využívaná ve webových frameworcích k statickému a dynamickému přiřazování URL adres a jejich vzorů k jednotlivým kontrolérům. V Play frameworku lze směrovač nastavit tak, aby četl nastavení cest URL z jediného souboru *conf/routes*.

3.6.1 REST API

Reprezentační převod stavu (angl. representational state transfer, zkráceně REST) je jeden ze způsobů komunikace mezi počítačovými systémy na internetu. K implementaci je obvykle použito rozlišení metody [24] HTTP požadavku GET, POST, PUT a DELETE (další nejsou pro pokrytí CRUD [25] operací nad databází potřeba), které jsou zapsány v HTTP hlavičce. Pro jednotlivé typy HTTP požadavků jsou očekávány odpovědi s výsledkem databázové operace. Metody modelu jsou v kontroléru vybrány pro požadavky jejich typem, tak aby se překrývaly s metodami v DAO [39], tím je ubrána režie kontrolérům. V následujícím listu jsou vypsány typy dotazů, jejich očekávané odpovědi a příslušné ekvivalentní SQL příkazy.

- GET
  - je typ požadavku určený pro získání dat ze serveru obvykle s identifikátorem jako parametr URI adresy,
  - požadavek obvykle neobsahuje ve svém těle data, ale data (obvykle pouze identifikátory) jsou přenášeny jako součást URI adresy,
  - SQL ekvivalent příkazu SELECT,
  - při úspěšném hledání se vrací v těle odpovědi hledaná data a stavový kód v hlavičce odpovědi je 200,
  - při neúspěšném hledání tělo obsahuje příčinu (např. neexistující identifikátor) a stavový kód v hlavičce odpovědi je 404.
- POST
  - je typ požadavku určený pro vložení dat v těle HTTP požadavku do trvalé paměti aplikace,
  - SQL ekvivalent příkazu CREATE,
  - při úspěšném vložení dat je v těle odpovědi vrácen identifikátor nového záznamu a stavový kód v hlavičce odpovědi je 201 (hlavička odpovědi může obsahovat odkaz na REST URI pro vyhledání nově přidaného záznamu),
  - při neúspěšném vložení je v hlavičce odpovědi stavový kód:

        - 404 pro nenalezený záznam,
        - 409 v již existující záznam (konflikt).
- PUT
    - je typ požadavku určený pro uložení a aktualizaci dat v trvalé paměti aplikace z těla HTTP požadavku,
    - SQL ekvivalent příkazu UPDATE, v modelu je z důvodu bezpečnosti blokováno změnit všechny řádky tabulky při jediném příkazu,
    - při úspěšné změně:
        - jsou v těle odpovědi odkazy na změněná data,
        - stavový kód v hlavičce odpovědi je 200.
    - při neúspěšné změně je v hlavičce odpovědi navrácen stavovým kódem důvod:
        - 204 pro žádnou změnu,
        - 404 pro nenalezení záznamů.
- DELETE
    - je typ požadavku určený pro smazání záznamu a strukturou je podobný dotazu typu GET,
    - SQL ekvivalent příkazu DELETE nebo DROP,
    - při úspěšném smazání vrací v hlavičce odpovědi stavový kód 204,
    - při nenalezení záznamů k smazání vrací v hlavičce odpovědi stavový kód 404.

Formáte přenosu dat je často JSON [26].

3.7 JSON

JavaScript Object Notation [27] je formát používající text pro přenos datových objektů obsahující atributy polí a párových hodnot (mapa). Je často použit pro serializaci objektů. Je použit obvykle ve webových aplikací pro přenos asynchronně získaných dat pomocí JavaScriptu, jelikož byl od JavaScriptu odvozen. Jeho oficiální internetový mediální typ je *application/json* a jeho soubory mají příponu *.json.* Rozlišuje několik základních typů hodnot:

- Decimální číslo s případným znaménkem. Může používat E-notaci [28], bez možnosti využití nečíselných hodnot.
- Řetězec znaků (angl. String) skládající se z 0 a více znaků sady Unicode [29].
- Pravdivostní hodnotu *true* nebo *false.*
- Pole seřazených hodnot o 0 a více prvcích podporovaného typu nebo objekt.
- Objekt jako neseřazenou n-tici páru klíč-hodnota, kde klíče jsou atributy objektu (řetězce) nesoucí hodnoty atributů podporovaného JSON typu.
- *null* jako prázdnou hodnotu.

3.8 Play framework

Play [23] je vysoce produktivní webový aplikační framework určen pro jazyky Java a Scala. To značí, že je oběma jazyky možné ho využít a používat i jeho moduly a pluginy, obvykle bez rozdílu na tom jestli je modul napsán v prvním či druhém jazyku. Cíle Play frameworku jsou odlehčenost (angl. lightweight), bezstavovost (angl. Stateless) a přívětivá webová architektura. Jeho funkce se snaží předvídat vývoj webových technologií a to s co možná nejrychlejší odezvou. Je napsán se snahou na nízkou konzumaci prostředků (CPU, paměť, vlákna) pro vysokou rozšiřitelnost aplikací díky

reaktivnímu modelu a tomu, že je založen na Akka Streams. Framework využívá návrhový vzor MVC.

3.9 µpickle

Je odlehčená serializační a deserializační knihovna [31] jazyku Scala. Je možné ji také použít v ScalaJS. Serializace je proces převedení objektu do řetězce znaků nebo hodnot tak, aby neztratil informace o svém stavu. Proces deserializace je pak opakem, převedením řetězce znaků nebo hodnot zpět do formy původního objektu. Knihovna umí serializovat a deserializovat základní datové typy a programovatelné typy *case class* a *case object* a to do a z formátu JSON.

3.10 Autentizace a zabezpečení

Autentizace dává možnost přihlášeným uživatelům využívat určité funkce aplikace, které nejsou dostupné anonymním uživatelům. Play framework nabízí více možností implementace autentizace a autorizace uživatele:

- Kompozicí Akcí, kde je nutné implementovat vlastní přihlašovací zobrazení, registraci a správu uživatelských účtů.
- Silhouette nabízí autentizační metody OAuth1, OAuth2, OpenID, CAS, Credentials, Basic Authentication, Two Factor Authentication nebo implementaci vlastního schématu autentizace.
- Deadbolt poskytující autentizační a autorizační mechanismus k přidělení práv a rolí uživatelům, používající OAuth.
- SecureSocial je modul s podporou autentizace pomocí OAuth, OAuth2, OpenID, jména a hesla nebo definování vlastního schématu.
- Play-pac4j podporuje OAuth, CAS, OpenID a HTTP autentizaci.
- Play20-auth je rozšíření kompozice Akcí o jednoduchou autentizaci uživatele s heslem a HTTP autentizace.

3.10.1 SecureSocial

Je modul pro Play Framework [30] zajišťující podporu autentizaci aplikací pomocí Oauth, Oauth2, OpenID, uživatel s heslem nebo vlastním způsobem. Jedná se o rozšiřitelný modul, lze ho jednoduše rozšířit o vlastní způsob autentizace a zabezpečí aplikaci před CSFR, hashování a přístupu k heslům. Modul obsahuje výchozí operace pro autentizaci uživatele a podporuje změnu hesla a způsobu zobrazení dat (vlastní šablony), modifikaci toho jaká autentizace je využita, uživatelské formuláře, získání veřejných informací o profilu, pomocí kterého je uživatel přihlášen. Tento modul je stále ve vývoji a s jeho poslední verzí, která je využita v aplikaci zatím nebyla aktualizována veškerá dokumentace.

Autentizace je v Play zajištěna pomocí využití objektu *SecuredAction* jako spotřebitele metody dotazu v kontroléru nebo nastavení směrování. Tento objekt také zajišťuje, že je v objektu dotazu v kontroléru je možné přistoupit k autentizovanému uživateli a jeho veřejným datům (uložených v databázi nebo paměti).

3.11 OAuth a OAuth2

OAuth [32] je otevřený standard pro přístup webových aplikací k uživatelským informacím na jiných webových stránkách bez nutnosti poskytnou uživatelské heslo. Tento způsob je využit společnostmi Google, Facebook, Microsoft, Twitter, Dropbox, aj. pro autentizace uživatelů aplikací třetích stran. Specifikuje proces autentizace uživatele pro přístup ke zdrojům dané aplikace pro uživatelský účet tento způsob autentizace podporující. OAuth 1.0 protokol by zveřejněn roku 2010 jako OAuth 2.0 je další verze OAuth, která ale není zpětně kompatibilní s OAuth 1.0 zveřejněna roku 2012 a je využita pro autentizaci webových aplikací, mobilních zařízení, desktopových aplikací a také třeba IoT zařízení s patřičnou softwarovou podporou.

Bezpečnost této služby byla od vydání první verze několikrát zlepšena a byly odstraněny některé bezpečnostní rizika. Aktuálně OAuth 2.0 stále trpí zranitelností na Phishing [33] útoky v případě, že se uživatel pokusí přistoupit pomocí OAuth ke zdroji (např. Sdílenému dokumentu) a tím dovolili potenciálně škodlivému softwaru přistoupit ke svým údajům.

V případě podlehnutí takovémuto útoku a krádeže autentizačních údajů je uživatelem nutné změnit heslo a zabezpečovací žeton aplikace. Při nastavení této autentizace je nutné v nastavení poskytovatele získat klientský identifikátor aplikace (Client ID) a tajné číslo (Secret ID), také je nutné zadat poskytovateli jméno aplikace a URL adresu aplikace odkud je autentizační dotaz očekáván a URL adresu, kam je přesměrována odpověď autentizace, ať úspěšná či ne. Díky těmto informacím je aplikace schopna se identifikovat a autentizovat pro daného poskytovatele této služby. Klientský identifikátor a tajné číslo je vloženo v konfiguraci aplikace. V případě této aplikace je využit Play modul SecureSocial a tyto informace jsou přidány do jeho konfigurace.

3.12 Akka

Akka dle definice v dokumentaci je *„Pružné distribuované zpracování transakcí v reálném čase“* [34]. Akka je soubor nástrojů (angl. toolkit) a snažící se zjednodušit psaní robustních webových aplikací. Robustnost v tomto případě znamená: správnost distribuce, paralelismus, toleranci chyb a rozšiřitelnost. Těchto vlastností je obtížné docílit jednoduchým způsobem. Akka se snaží tento problém řešit množstvím technik (např. metodou „let it crash“ - „nech to rozbít“) pro tvorbu aplikací s sebenápravou.

Akka implementuje několik hybridních způsobu práce s daty. Využívá tzv. Aktéry (angl. Actors), kteří slouží k jednoduché a vysokoúrovňové abstrakci distribuce dat, aplikační konkurence a paralelismu. Aktér je založen na principu odlehčených procesů závislých na událostech a zasílání zpráv. Pro představu těchto aktérů může být několik milionů na 1 GB paměti virtuálního stroje.

Pro splnění tolerance chyb využívá výše zmíněný koncept „let-it-crash“ tak, že využívá více JVM k poskytnutí sebenápravy a nezastavitelnosti aplikace. Průhlednost funkcionality aplikace je zajištěna tím, že jednotlivý aktéři spolu komunikují pouze pomocí průchodu zpráv. Pro sestavení projektu využívající Akka lze použít Maven nebo SBT.

V implementaci této práce byly využito rozšíření Akka-http, které dává aktérům možnost komunikovat pomocí HTTP požadavků jako zpráv. Objekt aktéra je obálkou pro jeho stav a chování. Systém pro správu aktérů (ActorSystem) je v aplikacích dostupný buď jako implicitní objekt v třídě aplikace, kontroléru nebo modelu nebo také jako globální proměnná. Systém aktérů uchovává jednotlivé instance třídy aktéra (class Actor) v hierarchické struktuře a jednotlivé instance lze identifikovat unikátním názvem nebo také cestou v systému aktérů. Každý vytvořený aktér má pak jediného předka, jímž je ten aktér, který ho vytvořil. Pomocí hierarchické struktury předků lze pak docílit bezpečnému rozesílání zpráv mezi jednotlivými aktéry tak, že každý aktér je zodpovědný za příchod zprávy jeho potomkům a v případě chyby vyvolané zprávou zaslanou potomku je pak tento

aktér k nalezení také pomocí toho, který na něj dohlíží. Pro sestavení systému s využitím systému aktérů je pak dobré dbát na praktiky:

1. Aktér by se měl chovat jako samostatný pracovník, který nebude při své práci rušit práci ostatních aktérů a se zprávami by měl nakládat jako s dotazy na které poskytne odpověď.
2. Aktér by neměl přijímat změnitelné objekt jako své zprávy. V případě nedodržení se v návrhu vracíme k standardním problémům řešení sdílených proměnných v paralelismu jazyka Java.
3. Využitím aktérů jinak než pro uchování stavu a chování může způsobit nekonzistenci nezměnitelnosti dat příchozích a odchozích zpráv.

Pro správné využití paralelně pracujících aktérů je nutné správné nastavení blokujících a neblokujících operací a to s ohledem na tyto zásady:

1. Blokující operace provádět uvnitř aktéra proto, aby nedošlo k zablokování jiných aktérů systému během vykonávání procesu paralelně běžícího aktéra.
2. Zapouzdřit blokující operace do instance Scala třídy *Future*, pro docílení navázání blokující operace do systému. Tímto také docílíme vytvořením vlastních správy paralelních vláken pro tuto blokující operaci.
3. Dedikovat samostatné vlákno pro správu blokujících operací nad danými zdroji.

Tyto zásady je nutné z důvodů blokujících operací při návrhu aplikace zohlednit a eliminovat je. Společný zdroj pro všechny aktéry je v aplikaci kolekce typu *Map [String, IrcLogBot]* obsahující mapování jednotlivých uživatelů k jim přiřazené instanci *IrcLogBot* a díky tomu je možné k této proměnné přistupovat také z autentizovaných HTTP dotazů v REST API stejně tak jako při globálních událostech (např. Start aplikace, ukončení aplikace – pro vyčištění dat a bezpečné ukončení běžících vláken). Jako vedlejším efektem výhod systému aktérů je nutnost přistupovat k aktérům z vnějšku (např. jiného systému i na vzdáleném stroji nebo v kontroléru při HTTP dotazu) pomocí odkazů. V této aplikaci se tomuto povedlo vyhnout již v návrhu aplikace.

3.12.1 Akka streams

Je rozšíření aktérů o funkce pro přenosu a manipulaci s datovými proudy. Datové proudy se na internetu běžně vyskytují jako využití komunikace služeb pro přenos multimediálních dat a dat v reálném čase. Toho je využito při WebSocketové komunikaci.

3.12.2 Akka http

Jedná se o rozšíření Akka o podporu HTTP provádějící tyto akce na několika úrovních:

- zajišťuje spojení,
- přijetí, zpracování a odesílání zpráv,
- čekání, odezvy, zpoždění ve zprávách.

Aktéři jsou zde využiti tak, že Akka HTTP nabízí nad jednotlivými částmi spojení programovatelnou kontrolu. Dále pak využívá toků zpráv, kterými lze nastavit chování serveru zprávám daného typu. Tento typ může být dle vlastností zprávy, jejího typu, obsahu http hlavičky, URL adrese. Tok je pak definuje odpověď na zprávu. A může být definován v jednotlivých částech spojení – při příchodu hlavičky, těla, při úspěšném nebo neúspěšném odeslání zprávy.

3.13 WebSocket

Webová technologie WebSocket je plně duplexní komunikační protokol, jedná se o protokol založený na TCP protokolu a byl standardizovaný podle IETF [36] v roce 2011. V dnešní době je protokol podporován ve velké části prohlížečů jako Google Chrome, Microsoft Edge, Internet Explorer, Firefox, Safari a Opera. Jedná se o protokol fungující na principu klient-server a tedy server i klient musí podporovat WebSocketový protokol. Websocketová technologie má podobné navázání spojení (angl. *handshake*) jako HTTP a díky své implementaci může fungovat na stejném portu jako HTTP protokol s tím rozdílem, že po úspěšném přijetí spojení se přepne do obousměrného binárního přenosu.
Důsledkem rozdíl oproti HTTP je, že je WebSocketový protokol plně duplexnímu má server možnost zasílat zprávy klientovy bez nutnosti čekat na dotaz klienta. Play framework podporuje možnost WebSocketové [37] komunikace a přebral ji z Akka HTTP.

3.14 Pircbotx

Je silná a flexibilní IRC knihovna napsaná v jazyce Java [38]. Zveřejněná roku 2013 a od té doby prošla vývojem. Aktuálně podporuje zachycení přes 50 událostí na IRC serveru a jeho kanálech. Podporuje SSL a IPv6.

3.15 Databázová vrstva

Databáze udržuje informace o uživatelích aplikace a jejich zabezpečení. V aplikaci je využita databáze H2 a pomocí Play databázových evolucí je vytvořeno lokální schéma databáze. Do databáze se pak přistupuje pomocí DAO využívajícího Play modul Play-Slick. Play-Slick je Play modul pro knihovnu Slick.

Slick je funkcionální mapování relačních schémat databází na objekty Scaly. Jedná se o moderní knihovnu pro práci s SQL dotazy a přístupem k databázi. Dovoluje pracovat s databázovými objekty podobně jako s kolekcemi implementující rozhraní *Iterable* přirozenými pro jazyk Scala (např. *Collection*, *List*, *Map*, *Set*). Jako knihovna Scaly je Slick typově bezpečný a všechny dotazy na databázi mají statickou kontrolu typu při kompilaci. Typová bezpečnost však nutí programátora definovat datové struktury na dvou místech:

1. ve schématu databáze,
2. znovu v kódu aplikace, pro každou tabulku je nutné vypsat většinu vlastností jejích polí (typ, unikátnost, prázdnost, aj.).

Slick využívá pro přístup k databázi H2 řidič. Jeho nastavení je k nalezení v konfiguraci serverové aplikace *conf/application.conf* a jedná se o položky *slick.dbs.default.{driver, db.driver, db.url}*.

3.16 Docker

Docker [40] je open-source projekt platformy a nástrojů pro automatizaci nasazování aplikací do virtuálních softwarových kontejnerů. Kontejner je běžící instance obrazu. Obraz je báze kontejneru. Obraz je uspořádaná kolekce změn kořenového systému souborů a korespondující parametry využité při jeho běhu. Obraz nemá stav a je neměnný a je využit buďto pro spuštění v podobě kontejneru nebo pro rozšíření a modifikaci v podobě jiného obrazu. Je postaven na Linux jádře a využívá jeho funkce k virtualizaci kontejnerů. Nevyužívá jiný operační systém, tak jako virtuální stroje. Jedná se o multiplatformní program fungující na Windows i Mac OS. V centralizovaném rozbočovači se

nachází veřejné obrazy, které jdou postavit, rozšířit a následně spustit na lokálním serveru s běžící instancí služby Docker.

Virtuální obrazy lze vytvořit pomocí příkazů Docker CLI nebo pomocí Dockerfile, textového dokumentu obsahujícímu Docker příkazy. Dockerfile mohou být zveřejněny na oficiálním rozbočovači (Docker Hub), kde jsou volně k využití. Dalšími nástroji pro virtualizaci pak jsou:

- VirtualBox, pomocí kterého lze vytvořit virtuální stroje a pro účely této aplikace nebyl zvolen z důvodu požadavků na spustitelnosti pomocí OpenShiftu.
- VMWare, pro který jsem nenalezl vhodnou podporu pro OpenShift a SBT nástroje.

Pro spuštění aplikace v kontejneru na stroji, kde běží služba *docker daemon* jsou nutné 2 příkazy:

- *docker build <cesta_k_Dockerfile>*
- *docker run -t -p 9000:9000 <buildID>*
  - buildID je zobrazeno v posledním řádku úspěšného provedení předchozího příkazu,
  - parametre *-t* slouží k povolení TTL a neukončení běhu po spuštění,
  - parametr -p <port_aplikace>:<port_kontejneru> slouží k namapování portů kontejneru na port aplikace.

3.17 OpenShift

Je kontejnerová aplikační platforma firmy Red Hat, Inc., která umožňuje vývojářům rychlou tvorbu, sestavení, nahrání a správu kontejnerových služeb a aplikací v prostředí cloudu[3]. Pro bližší pochopení je nutné si popsat jednotlivé pojmy. Jedná se o kontejnerovou aplikační platformu pro vývoj, sestavení, spouštění a správu služeb běžících v softwarových kontejnerech a aplikací v cloudovém prostředí.
OpenShift projekt je aktuálně rozdělen na 4 části:

- OpenShift Online je veřejná cloudová aplikační platforma pro nasazování a hostování aplikací,
- OpenShift Container Platform je soukromá platforma pro vývoj a hostování on-premise[4] cloudových aplikací,
- OdepnShift Dedicated je služba pro vývoj a hostování veřejných cloudových aplikací pod správou Red Hat,
- OpenShift Origin je veřejný upstream[5] projekt.

3.18 OpenShift Origin

OpenShift Origin je veřejný upstream projekt OpneShiftu. Je pomocí něho možné spustit aplikaci v cloudovém prostředí s určitými parametry. Pro využití této platformy je nutné splnit 2 body:

- Vlastnit účet na serveru platformy OpenShift Origin.
- Vlastnit nainstalované OpenShift CLI.

---

3 Oficiální web OpenShift dostupný na URL: https://www.openshift.com/dedicated/

4 Wikipedie - otevřená encyklopedie: On-premise [online]. Poslední modifikace 2017-04-10 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/On-premises_software>

5 Wikipedie - otevřená encyklopedie: Upstream [online]. Poslední modifikace 2017-01-02 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Upstream_(software_development)>

Rozhraním příkazové řádky OpenShift Origin je možné tvořit aplikace a spravovat projekty s pomocí terminálu. OpenShift má vlastní webové rozhraní, ve kterém lze spravovat aplikace, šablony obrazů a jiné zdroje. Do těchto zdrojů také patří databáze. OpenShift podporuje celou řadu programovacích jazyků pro tvorbu aplikací. Při tvorbě aplikace s využitím nástrojů nabízející OpenShift je vhodné vybrat takové zdroje, které mají v platformě dobrou podporu. Pro příklad je lepší vybrat si jednu z podporovaných databázových vrstev nežli nějakou nepodporovanou z důvodu dynamického nastavení databáze a jednoduchého přizpůsobení databáze na rozdělaném projektu, změnám využití databáze při rozdílu testování a tisíců uživatelů a změnám zátěže na databázi. Seznam podporovaných nástrojů a technologií je k nalezení v oficiální dokumentaci [41].

Jednotlivé kontejnery lez pomocí OpenShiftu spustit paralelně v luscích (angl. pods), tyto lusky mají společný souborový systém, IP adresu a zvýšením jejich počtu lze zvolnit zátěž na aplikaci. Několik aplikací pak může běžet paralelně se sdílenou databází a tím rozložit zátěž. OpenShift stále pracuje na zlepšení analýzy a monitorování prostředků a kontejnerů pro zvýšení reakce na změny v běžících aplikacích, například při rychlém nárůstu webových dotazů, změny ve využití a spojení s databází nebo náhle chybě v aplikaci bežící v kontejneru. Pro komunikaci s OpenShift platformou lze také využít REST API, které slouží ke správě uživatelských aplikací, shluku a jeho uživatelů.

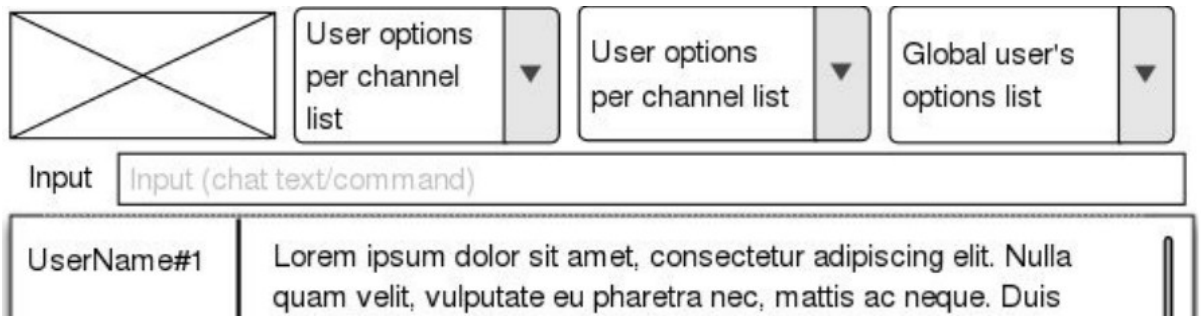
4 Návrh

V následující kapitole je rozebrán návrh aplikace v závislosti na požadavcích na použití a technologie. V návrhu je nutné brát v potaz technologií určených zadáním (Scala, ReactJS, WebSocket, Docker, atd.). Ty jsou výchozím bodem pro návrh architektury aplikace. Je zde přiblížena tvorba kontrolérů, modelu a návrh uživatelského rozhraní. Dále jsou rozebrány problémy, se kterými jsem se setkal v průběhu programování a vysvětleno jejich řešení. Zadavatelem vybraná knihovna ReactJS a technologie WebSocket podnítilo využití k architektury klient-server, protože aplikace psané v ReactJS mají možnost využití vzoru MVC i na straně klienta. ScalaJS jsem zvolil po analýze webových projektů využívajících Play framework a jeho modulů po nalezení knihovny ScalaJS-React. Z pohledu uživatel musí aplikace splnit požadavky na operace:

- přihlášení/odhlášení do webové rozhraní,
- zapnutí/vypnutí IRC bota,
- přihlášení se do IRC kanálů serveru na který je bot připojen,
- možnosti psát a přijímat zprávy v připojených IRC kanálech,
- zobrazení jednotlivých souborů záznamů,
- hledání v záznamech užitím klíčových slov,
- rotace logů (automatická i manuální).

Tyto operace jsou popsány v diagramu případu užití (use case diagram) *Obr. 2*. Jako další krok jsem prostudoval dokumentaci, ukázkové a veřejné aplikace Play, ReactJS, WebSocket[6789]. Požadavkem na UI by mělo být jednoduchost, přehlednost a intuitivnost pro práci. Z těchto důvodů jsem navrhl rozhraní mající jediné menu Obr. 1 v horní části šablony, pomocí kterého lze navigovat na stránky:

- IRC klienta s možností přihlášení do více kanálů, zapnutí/vypnutí IRC bota, psaní a příjmu zpráv do vybraného kanálu, manuální rotaci souborů záznamu.
- Stránku pro správu záznamů, jejich zobrazení s informací data vytvoření souboru a názvu kanálu a hledání v záznamech užitím klíčových slov.



Obr. 1: Návrh menu uživatelského rozhraní

6 Play projekt s React UI. Dostupné na URL: https://github.com/knoldus/playing-reactjs

7 Ukázková aplikace: použití pircbotx nihovny s UI. Dostupné na URL: https://github.com/Jiri-Kremser/ircLogBot

8 Play s WebSocket chatem. Dostupné na URL: https://github.com/jrudolph/akka-http-scala-js-websocket-chat

9 Play zabezpečení REST API. Dostupné na URL: https://github.com/jamesward/play-rest-security

5 Implementace

Následující kapitola se zaobírá implementací návrhu pomocí požadovaných technologií do formy funkční aplikace splňující požadavky zadání. Je zde popsána konfigurace serverové i klientské části, použití návrhového vzoru MVC, implementace WebSocketového spojení využitím aktérů Play, zabezpečení uživatelského rozhraní a implementace IRC bota, konfigurovatelnost výsledné aplikace a tvorba kontejneru pro její spuštění pomocí nástroje Docker. Pro aplikaci běží lokální HTTP server na portu 9000, z důvodu testování a zamezení kolize využití portu 80 pro běžné připojení k WWW. Pro nastavení URL cest k jednotlivým kontrolérům je využito nastavení směrovače v souboru *server/conf/*route. Nastavit směrovače URL je pro front a back end nutné samostatně, protože ReactJS nepodporuje čtení souboru tohoto formátu a využívá k směrování vlastní objekty.

5.1 SBT konfigurace aplikace

Aplikace využívá možnost vícemodulového projektu. Prvním modulem je serverová část aplikace využívající Play framework, pokrývající back end. Druhým modulem je klientská aplikace běžící ve webovém prohlížeči napsaná pomocí ReactJS knihovny JavaScriptu. Zdrojové kódy back endu jsou v adresáři *server* a zdrojové ScalaJS kódy pro front end jsou v adresáři *klient*. Klientský modul je závislý na modulu serverové, což značí, že kompilace zdrojových souborů pro závislý klientský modul proběhne pouze pokud je bez chyb zkompilován modul serveru. Naopak tomu být nemusí, díky čemuž je testování front endu ReactJS možné dělat bez neustálého kompilování back endu. Zkompilované soubory front endu jsou soubory JavaScriptu zveřejněné ve zdrojích aplikace k přiložení do HTML pomocí tagu *<script type="text/javascript" src="cesta_k_souboru.js"/>*. Cesta k vygenerovaným souborům je v konfiguraci v proměnné *jsOutDir* a název vygenerovaného *.js* souboru je definovaný atributem *name* modulu *client*.

V konfiguraci je nutné definovat nejen závislosti, ale také jejich verze, tak aby byly jednotlivé závislosti spolu kompatibilní. Výběr verzí je nutné uskutečnit při návrhu aplikace a dbát přitom ohled na budoucí změny v aplikaci (např. pro nekorektní využití výchozích řidičů databáze při použití Slick závislosti vedoucí k chybě v injekci konfigurací do instancí DAO). Pro spuštění aplikace zapotřebí použít příkaz *sbt run*, který spustí stahování knihoven, zkompiluje aplikaci a následně ji spustí na lokálním webovém serveru.

5.2 Back end

Pro návrh modelu pro back end je potřeba si sepsat zdroje dat se kterými bude aplikace pracovat. V této aplikaci se lze setkat s dvěma druhy modelu. Jeden starající se o databázové operace a druhý, který se stará o operace nad daty získanými IRC robotem nebo nad informacemi o běžící instanci IRC bota. Aplikace využívá REST API, pomocí kterého lze manipulovat s daty modelu (např. rotovat logy pomocí REST dotazu, zobrazit názvy záznamů, zobrazit záznamy, vyhledávat v záznamech).

Jednotlivé části zobrazení jsou rozděleny dle Play modulů a kontrolérů. Autentizace uživatele pomocí Play modulu SecureSocial využívá pro zobrazení svých dat a stránek CSS knihovnu Bootstrap a vlastním zobrazením pro jednotlivé kontroléry. Pro vlastní stránku zobrazení je přidán kontrolér *CustomLoginController*, který zajišťuje nastavení cest k souborům s příslušným zobrazením pro přihlášeného uživatele nebo přesměrování na stránku přihlášení (s výchozím zobrazením).

Aplikace využívá několik kontrolérů, které jsou popsány UML diagramem Obr. 5:

- *Application je* báze pro definici implicitních hodnot.

- *BaseController:*
  - báze pro společné hodnoty REST a jiných kontrolérů,
  - definuje implicitní čekání na *Future*.
- *IrcWebController* obsahuje Akci *chat(botName: String)* definuje chování WebSocketového spojení a navrací instanci WebSocket.
- *RestController* obsahuje Akce pro manipulaci s modelem aplikace, soubory IRC záznamů, názvy těchto souborů a operace aplikace.

Obr. 2: Diagram užití pro konfiguraci perzistentního bota

- *ReactJsController* jeho Akce *react(any: String)* vrací šablonu s přiloženým JavaScriptovým kódem zkompilovanou klientskou částí aplikace napsanou v ScalaJS-React a hodnota *any* reprezentuje jakýkoliv dotaz pro klientskou část aplikace.

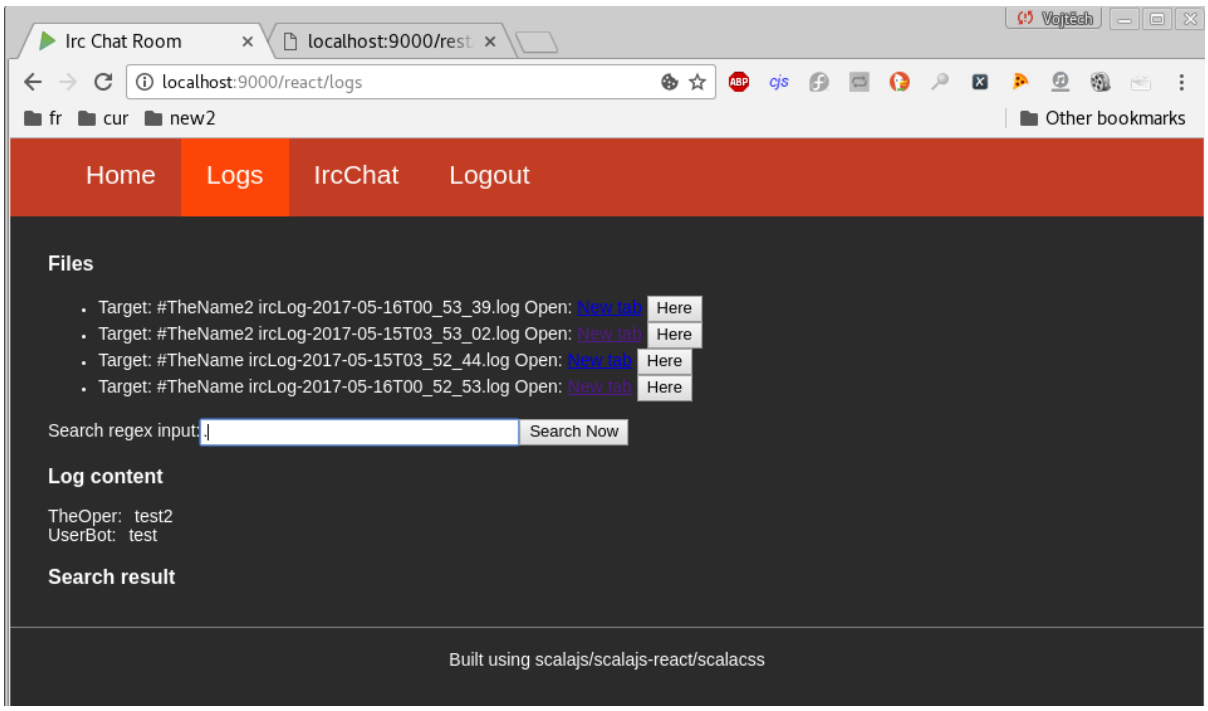
REST API použité v aplikaci využívá JSON formát pro přenos dat. Pro hledání v záznamech jsou využity speciální zprávy ve sdílené části SBT projektu typu *case class*:

- *JsMessageSearchLogsRequest(regex: String, target: String)* obálka pro dotaz hledání, *regex* obsahuje regulární výraz a *target* obsahuje cíle v kterých se vyhledává (výchozí jsou všechny, jiné neimplementováno).
- *JsMessageSearchResults(results: Array[LogSnippet])* obsahující výsledky hledání v záznamech.
- *LogSnippet(line: String, target: String, filename: String, found: String, jsmsg: JsMessage)* obsahuje jeden záznam ve které byla nalezena shoda pro zadaný regulární výraz. Pro vyhledávání záznamů jsem uvážil za dostatečné využití Scala balíčku *scala.util.matchong.Regex*.

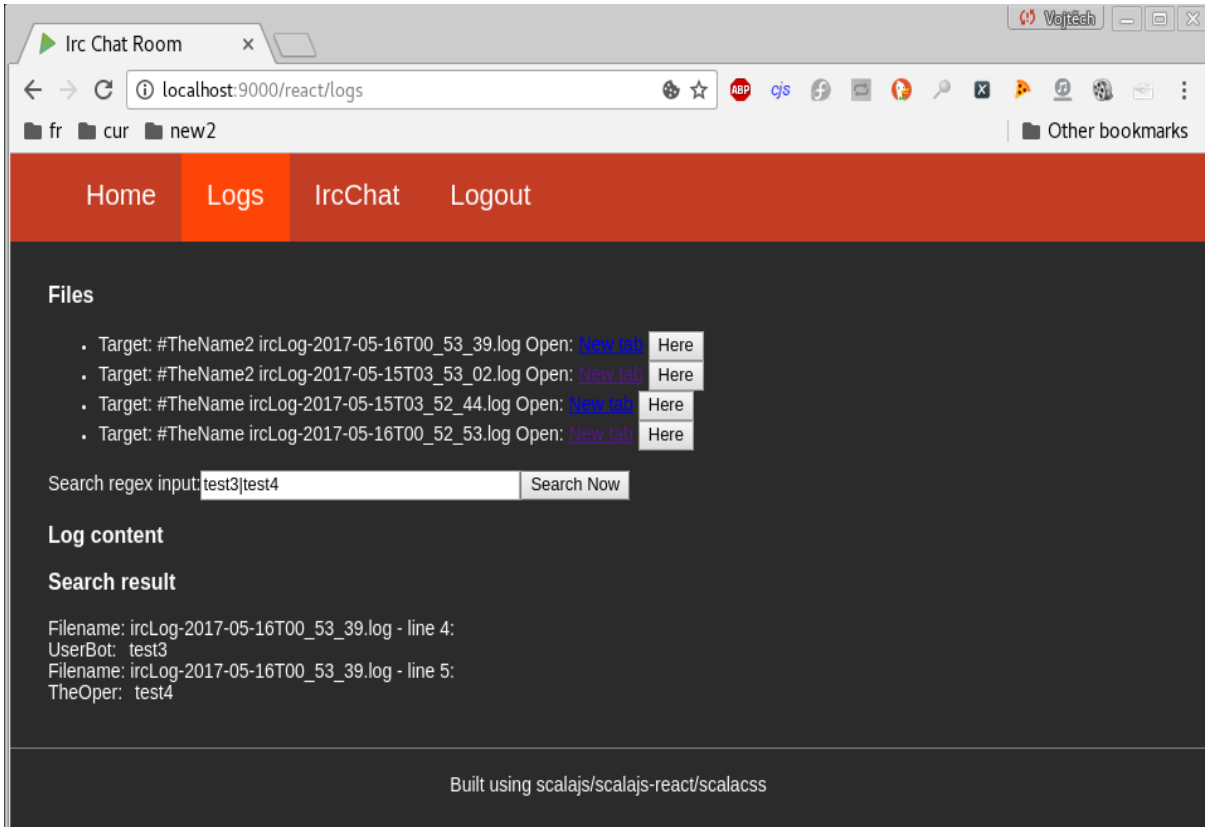
5.3 Front end

Vzor MVC je využit i pro front end. Klientská aplikace běžící v prohlížeči využívá React k zobrazení veškerého obsahu pomocí JavaScriptu a mapování virtuálního DOM. Výchozí URL předpona klientské aplikace je */react*. Front end aplikace se skládá z:

- *ReactApp* objektu aplikace s metodou *main()*, která načte dynamické styly a výchozí data, vytvoří lokální směrovač URL a vykreslí JavaScriptovou šablonu do kořenového tagu * *.
- *AppRouter* objektu směrovače definujícím obsah menu a přiřazením prvkům menu jejich URL, čerpá z výchozích dat načtených ze zobrazení *views/reactJs.scala.html* v serverové části aplikace frameworku Play.
- Jednotlivé stánky aplikace:
  - *Home* obsahuje informace o zdroji.
  - *IrcChat* obsahuje jednoduchou implementaci IRC klienta podporující více kanálů a rotace souboru záznamů (Obr. 6, Obr. 7).
  - Logs zobrazuje odkazy na soubory záznamů, možnost zobrazit čitelný obsah záznamů a možnost v záznamech vyhledávat použitím regulárního výrazu (Obr. 3, Obr. 4).
  - *Logout* přesměruje na adresu */custom/logout* pro odhlášení uživatele.



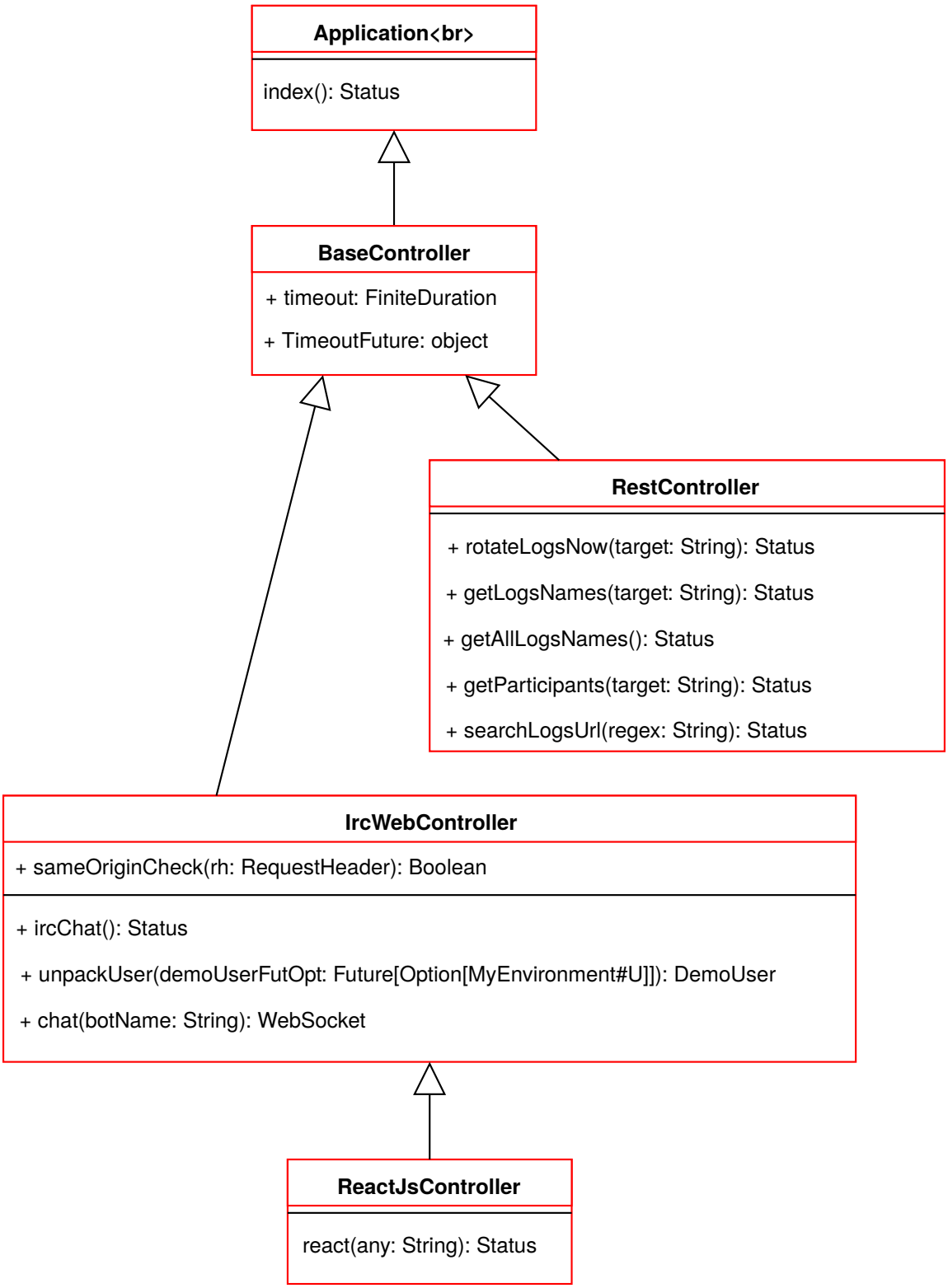
Obr. 3: Ukázka zobrazení souborů záznamů a jejich obsahu



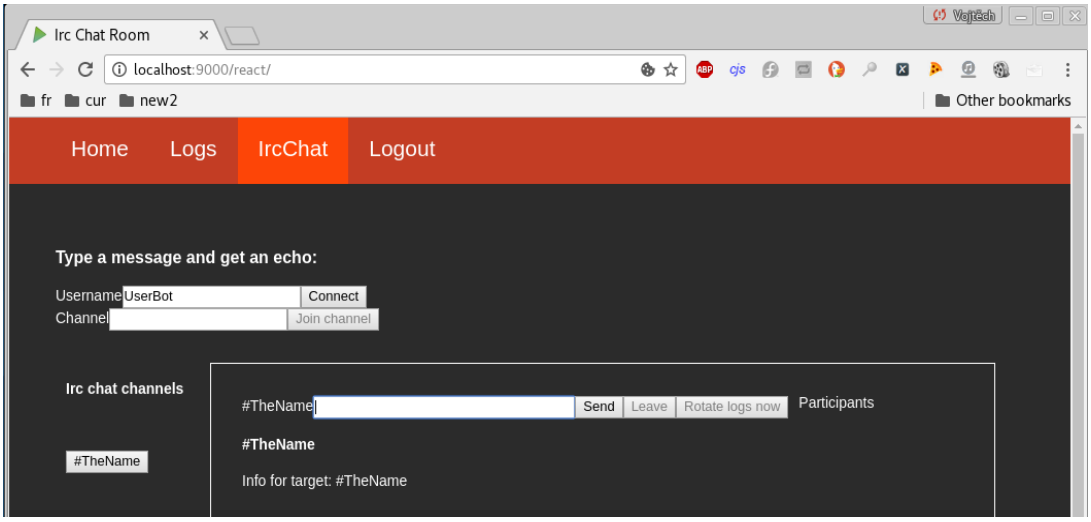
Obr. 4: Ukázka hledání v záznamech užitím regulárního výrazu

5.4 Websocket

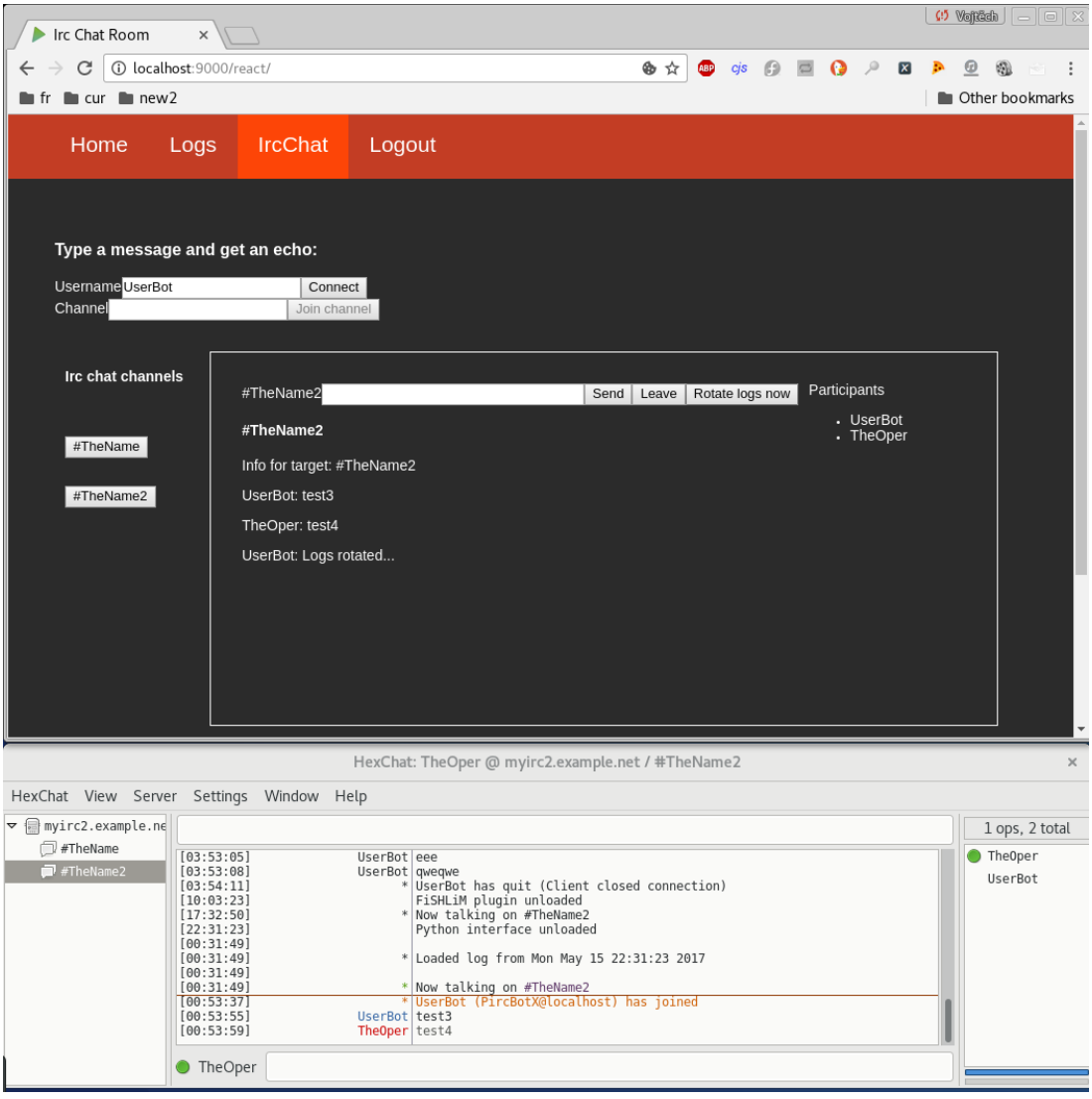
V kontroléru je pak nutné definovat metodu pro vytvoření takového spojení a toku dat v něm. Aktéři se v tomto spojení využívají tak, aby každá část komunikace obsahující stav měla vlastního aktéra, který tento stav bude reprezentovat pro každé spojení. V aplikaci je využit aktér pro udržování spojení a aktér pro klienta WebSocketu. Aktér udržující spojení využívá událostmi přijetí (*case class IrcNewParticipant*), odchod (*case class IrcParticipantLeft*) a příjem zpráv (*case class IrcReceivedMessage*). Aktér pro WebSocket je implementován třídou *WebsocketUser(system:*



Obr. 5: UML diagram dědičnosti kontrolérů



Obr. 6: Výchozí zobrazení webového IRC klienta



Obr. 7: Ukázka komnunikace webového IRC klienta pro více kanálů, rotaci záznamu a HexChat k porovnání

ActorSystem, name: String, channel: String, demoUser:DemoUser, ircBot: IrcLogBot: null) přijímající:

- systém aktérů pro přístup ke konfiguraci,
- jména uživatele (pokud není dáno konfigurací aplikace jinak, pak je využit jako jméno IRC bota),
- výchozím IRC kanálem (prioritně brán z konfigurace, využíváno při testování aktéra),
- odkazem na autentizovaného uživatelem,
- proměnnou, která může obsahovat spuštěnou instanci IRC bota (v případě konfigurace perzistentního bota, který není zastaven při konci WebSocketového spojení).

V metodě *myChatFlow* je definováno zpracování proudu dat užitím aktérů a definice typu zpráv *JsValue*. *JsValue* je datový typ do a z kterého umí serializovat a deserializovat data funkce *write* a *read* knihovny *µpickle*. Komunikace mezi IRC botem a webovým klientem je na bázi zprávy typu dotaz-odpověď nebo pomocí příkazů ze strany serveru. Díky možnosti sdílení datových struktur mezi front a back endem konfigurací vícemodulového SBT projektu je možné využít pro komunikaci zprávy typu *case class* odvozené od *sealed trait JsMessageBase*:

- *JsMessage(sender: String, target: String, msg: String)* obsahující cíl, odesilatele a zprávu.
- *JsMessageOther(sender: String, target: String, msg: String)* stejně jako *JsMessage*, ale pro odlišení druhu zprávy, využito pro záznam událostí IRC, které není nutné zobrazovat uživateli.
- *JsMessageJoinChannelRequest(sender: String, target: String)* dotaz uživatele na připojení do IRC kanálu.
- *JsMessageJoinChannelResponse(sender: String, target: String, participants: Array[TargetParticipant])* odpověď serveru na dotaz *JsMessageJoinChannelRequest* obsahující pole aktuálních účastníků kanálu.
- *JsMessageLeaveChannel(sender: String, target: String)* žádost uživatele o opuštění kanálu.
- *JsMessageLeaveChannelResponse(sender: String, target: String)* odpověď serveru na dotaz *JsMessageLeaveChannel*, při příchodu zprávy je daný kanál IRC botem již opuštěn.
- *JsMessageRotateLogs(sender: String, target: String)* žádost o rotaci logů.
- *JsMessageTargetParticipantsRequest(sender: String, target: String)* žádost o zaslání účastníků kanálu názvu *target.*
- *JsMessageTargetParticipantsResponse(sender: String, target: String, participants: Array[TargetParticipant])* odpověď serveru na dotaz *JsMessageTargetParticipantsRequest* obsahující všechny účastníky kanálu.
- *JsMessageIrcBotReady()* zpráva serveru zaslaná klientovi při úspěšném přihlášení IRC bota k serveru.

5.5 Zabezpečení

Modul SecureSocial je v aplikaci použit k autentizaci uživatele webového rozhraní, REST API a WebSocketovém spojení, které je přístupné pouze autentizovaným uživatelům. Pro testování autentizace uživatele pomocí poskytovatele GitHub jsem vytvořil samostatný účet *echoIrcAut*. Pro autentizaci uživatelů je v něm povoleno OAtuh2 a vygenerované klientský identifikátor a klientský klíč. Přístupový bod OAuth2 autentizace lze změnit konfigurací souboru *server/conf/securesocial.conf* a to atributy *clientId* a *clientSecret* v sekci poskytovatele *github.* Při první návštěvě aplikace je uživatel přesměrován na adresu *hostname:port/custom/login* a je mu

umožněno přihlásit se do aplikace. V konfiguraci je možné nastavit specifická uživatelská jména nebo identifikátory povolené pro přihlášení do aplikace a tím omezit přístup pouze administrátoru aplikace.

5.6 Pircbotx

Knihovna je implementována jako použitím rozšíření třídy *PircBotX* na třídu *IrcLogBot* obsahující odkaz na výchozí instanci naslouchače *IrcListener*. Knihovna sice podporuje více instancí posluchače (třída *IrcListener*) pro jednoho IRC bota, ale pro implementaci aplikace toto není nutné a v této aplikaci je využit pouze jedna instance *IrcListener* pro jednotlivé IRC boty s odkazem v instanci *IrcLogBot*. Metody pro naslouchání třídy *IrcListener* jsou volány asynchronně vůči zbytku aplikace. Proto je nutné zajistit bezpečný přístup ke sdíleným zdrojům aplikace nebo zajistit nedostupnost sdílených zdrojů této třídě. Aktéři Akka jsou stavěni pro paralelní běh a tato jejich vlastnost je využita při přístupu ke sdíleným zdrojům. Každá instanci *IrcListener* obsahuje mapu *Map[String, LogWrapper]*, mapující řetězce názvů kanálů na instanci modelu *LogWrapper* starajícího se o záznamy. Třída *LogWrapper* je odvozena od *LogBase* třídy, která obsahuje základní metody pro správu záznamů. Umožňuje získat názvy všech souborů uživatele a prohledat je pomocí regulárních výrazů. *LogWrapper* obsahuje metody sloužící k ukládání, rotaci, čtení záznamů a dále metody rodičovské třídy.

5.7 Konfigurace aplikace

V aplikace je možnost nakonfigurovat server i klient do několika různých způsobů běhu a řešení určitých situací.

- *app.client.runAsSimpleIRCChat = true*
  - Aplikace udržuje IRC klienta při spuštěném spojení pomocí WebSocketu a při ukončení spojení ukončí klienta (platí pro všechny uživatele).
- *app.client.runIRCBot = true*
  - Spustí jedinou instanci IRC bot při startu serveru (z důvodu kompilace Scaly při dotazu na běžící webový server, při 1. dotazu na webový server).
  - K tomuto klientu se může přihlásit pouze administrátor z jednoho HTTP sezení (pro zamezení problémů sdílených proměnných).
- *app.client.adminPages = [„logs“,“ircchat“]*
  - Seznam výchozích stránek dostupných ve webové aplikaci pro autentizovaného uživatele.
- *app.server.owner.{username, password}*
  - Definuje přihlašovací údaje pro vlastníka spuštěné aplikace, je doporučeno změnit tyto informace.
- *app.web.{interface, port}*
  - Definuje rozhraní na jakém aplikace poběží a její port.
- *app.websocket*
  - Definuje URL adresu pro websocketové spojení (tato vlastnost se nereflektuje ve směrovači Play frameworku a je nutné ji nastavit poté i v ní).
- *app.irc*
  - Obsahuje informace o výchozím IRC botu, povoleným pomocí *app.client.runIRCBot=true.*
  - Definuje:
    - *server* - na jaký se má bot přihlásit,

- *defaultUserName* - výchozí uživatelské jméno pod kterým se bot přihlásí k serveru,
- *defaultChannels* - výchozí kanály, na které se má bot přihlásit při připojení k serveru,
- *defaultLogRotationInterval* - výchozí čekání pro rotaci logů.

5.8 Slick DAO

Objekty datových přístupů (angl. data access object, zkratka DAO) sou v projektu využity pro databázové tabulky uchovávající informace o uživatelích a autentizačních žetonech. Třídy *UserDAO* a *TokenDAO* využívají injekci databázové konfigurace pro zajištění databázového spojení. Tyto třídy také přijímají jako implicitní proměnnou objekt s kontextem aplikace. Toto je nutné pro přístup ke globální konfiguraci a konfiguraci řidiče (angl. driver) databáze. V DAO je nutné definovat závislost na databázovém schématu. Dále metody, které budou sloužit pro manipulaci s databází. Tyto metody z důvodu bezpečnosti paralelních operací budou vždy vracet hodnoty zabalené do Scala struktury *Future*, čímž je zajištěné, že DAO nebude obsahovat blokující operace (nebo by nemělo, lze naprogramovat čekání na výsledek, čímž se ale z DAO stává blokující zdroj).

5.9 Virtualizace

V případě tvorby této aplikace jsem využil veřejný obraz[10] obsahující Linux s nainstalovanou Javou verze 8, SBT, Gitem a dalšími potřebnými závislostmi. Pomocí příkazu *docker build ./docker* v kořeni aplikace je postaven obraz projektu a následným příkazem *docker run <dockerId>* obraz spuštěn jako kontejner, který si stáhne nejnovější verzi aplikace z GitHub[11] úložiště *vprusa/echoIrc* a následně ji spustí. Při sestavení obrazu aplikace je nutné, aby byly otevřeny porty pro vzdálenou komunikaci s kontejnerem. Zde je tomu je port 9000.

5.10 Testování

Testování aplikace proběhlo automatické a manuální. Aplikace obsahuje sadu unit testů pro jednotlivé komponenty back endu ve složce *server/test*. Testovací scénáře zahrnují otestování jednotlivých DAO, autentizaci, URL cest aplikace, aktéry WebSocketu a jejich zpracování zprávy. Testování front endu bylo provedeno manuálně. Aplikace je schopna připojit se do IRC kanálu, zobrazovat konverzaci ve webovém rozhraní využití technologie WebSocket. Aplikace je spustitelná pomocí nástroje pro vývoj a sestavení kontejnerů Docker i OpenShift. Ke spuštění aplikace lze použít platformu OpenShift s využitím webového rozhraní nebo CLI. Pro otestování je použito CLI na lokálním serveru. Pro testování IRC klienta jsem použil IRC server ngIRCd [43] s přiloženou konfigurací a pro sledování tohoto lokálního IRC serveru a jeho kanálů jsem použil klienta HexChat [44].

10 Docker obraz s nainstalovanou Java 8 a SBT dostupné na URL: *https://github.com/hseeberger/scala-sbt*

11 GitHub: veřejný zdroj této aplikace dostupné na URL: https://github.com/vprusa/echoIrc

6 Závěr

Zadáním práce je seznámit se s technologií IRC, navrhnout architekturu IRC robota. IRC roboty by měl podporovat více IRC kanálů, mít možnost rotace záznamů a vyhledávání v záznamech použitím klíčových slov. Navrhnout vhodné uživatelské rozhraním takové aplikace a architekturu implementovat užitím zadaných technologií. Dle zadání z Red Hat, Inc. je v aplikaci v jazyku Scala implementována technologie WebSocket s použitím Play frameworku a front endové knihovny ReactJS. Webového rozhraní aplikace je zabezpečené Play modulem SecureSocial. Aplikace zaznamenává události na vybraných kanálech IRC serveru, v záznamech je možno vyhledávat užitím řetězce regulárních výrazů. Časově nejnáročnější bylo zprovoznění kostry projektu, tak aby spolu byly kompatibilní všechny verze závislostí a knihoven. Například front endovou knihovnu ScalaJS-React-Components[12] s hotovými ScalaJS-React komponenty nebylo možné při psaní aplikace použít právě z důvodu nekompatibility verzí této knihovny a knihovny ScalaCSS. Knihovna ScalaJS-React-Components byla při psaní této práce z vetší části již 2 roky netknutá přesto, že knihovna ScalaJS-React je stále ve vývoji. Díky nové verzi knihovny ScalaJS-React, která vyšla 20. dubna 2017, jsem aplikaci změnil použití komponent ReactJS.

V aplikaci je v plánu vylepšit zobrazení dalších uživatelských událostí, možnost uživatelem definovat reakce na tyto události spuštěním uživatelských scriptů nebo podprogramů. Zobrazení statistik z logů (např. počet slov, nejčastější výrazy). Upravení kódu aplikace tak, aby nevyužívala instance tříd nevyužívali *null*, ale raději *Option*, *Some* a *None* objekty jazyka Scala (např. v třídě *WebsocketUser*). Spustitelnost aplikace pomocí nástroje pro virtualizaci Docker byla docílena a otestována. Platforma OpenShift podporuje spuštění Docker kontejnerů. Využití OpenShiftu je pro aplikaci tohoto rozsahu a využití celkem zbytečné, jelikož OpenShift je určen pro aplikace vyžadující daleko větší výpočetní výkon a rychlejší odezvu.

12 Znovupoužitelné komponenty pro ScalaJS-React k nalezení na URL: https://github.com/chandu0101/scalajs-react-components

Literatura

[1] Wikipedie - otevřená encyklopedie: SGML [online]. Poslední modifikace 2017-04-30 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Standard_Generalized_Markup_Language>

[2] Wikipedie - otevřená encyklopedie: World Wide Web [online]. Poslední modifikace 2017-05-17 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/World_Wide_Web>

[3] Wikipedie - otevřená encyklopedie: TLS [online]. Poslední modifikace 2017-05-10 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Transport_Layer_Security>

[4] World Wide Web Consortium: HTTP request [online]. Poslední modifikace 2017-05-14 [cit. 2017-05-15]. Dostupné na URL: <https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html>

[5] Wikipedie - otevřená encyklopedie: CSRF [online]. Poslední modifikace 2017-05-10 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Cross-site_request_forgery>

[6] Wikipedie - otevřená encyklopedie: Cache [online]. Poslední modifikace 2017-04-22 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Cache_(computing)>

[7] Wikipedie - otevřená encyklopedie: AJAX [online]. Poslední modifikace 2017-05-02 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Ajax_%28programming%29>

[8] Wikipedie - otevřená encyklopedie: XMLHttpRequest [online]. Poslední modifikace 2017-03-07 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/XMLHttpRequest>

[9] Wikipedie - otevřená encyklopedie: Callback [online]. Poslední modifikace 2017-05-12 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Callback_%28computer_programming%29>

[10] Wikipedie - otevřená encyklopedie: GNUv2 [online]. Poslední modifikace 2017-05-13 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/GNU_General_Public_License>

[11] Wikipedie - otevřená encyklopedie: GitHub [online]. Poslední modifikace 2017-05-07 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/GitHub>

[12] Wikipedie - otevřená encyklopedie: Cloud computing [online]. Poslední modifikace 2017-05-12 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Cloud_computing>

[13] Wikipedie - otevřená encyklopedie: ARPANET [online]. Poslední modifikace 2017-05-11 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/ARPANET>

[14] Wikipedie - otevřená encyklopedie: Scala [online]. Poslední modifikace 2017-04-30 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Scala_(programming_language)>

[15] Wikipedie - otevřená encyklopedie: Just-in-time compilation [online]. Poslední modifikace 2017-05-11 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Just-in-time_compilation>

[16] Wikipedie - otevřená encyklopedie: React [online]. Poslední modifikace 2017-05-12 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/React_(JavaScript_library)>

[17] Oficiální dokumentace JSX: JSX [online]. [cit. 2017-05-15]. Dostupné na URL: <https://jsx.github.io/>

[18] Oficiální dokumentace: ScalaJS-React [online]. Poslední modifikace 2017-05-15 [cit. 2017-05-15]. Dostupné na URL: <https://github.com/japgolly/scalajs-react>

[19] Blog s veřejnou diskuzí: Why learn Scala [online]. [cit. 2017-05-15]. Dostupné na URL: <https://www.toptal.com/scala/why-should-i-learn-scala/>

[20] Oficiální dokumentace jazyku Scala: Specifikace [online]. [cit. 2017-05-15]. Dostupné na URL: <http://www.scala-lang.org/files/archive/spec/2.12/>

[21] Oficiální dokumentace jazyku Scala: Syntaxe [online]. [cit. 2017-05-15]. Dostupné na URL: <http://www.scala-lang.org/files/archive/spec/2.12/01-lexical-syntax.html>

[22] StackOverflow stránka pro dotazy: Výkonnost jazyku Scala a Java [online]. [cit. 2017-05-15]. Dostupné na URL: <http://stackoverflow.com/questions/5901452/scala-vs-java-performance-and-memory>

[23] Oficiální stránka Play frameworku: Úvod dokumentace [online]. [cit. 2017-05-15]. Dostupné na URL: <https://www.playframework.com/documentation/2.5.x/Home>

[24] Wikipedie - otevřená encyklopedie: HTTP request methods [online]. Poslední modifikace 2017-05-11 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods>

[25] Wikipedie - otevřená encyklopedie: CRUD [online]. Poslední modifikace 2017-04-08 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Create,_read,_update_and_delete>

[26] Wikipedie - otevřená encyklopedie: JSON [online]. Poslední modifikace 2017-05-14 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/JSON>

[27] RFC dokumentace: JSON fromát [online]. Poslední modifikace 2013-03-01 [cit. 2017-05-15]. Dostupné na URL: <https://tools.ietf.org/html/rfc7159>

[28] Wikipedie - otevřená encyklopedie: E notace [online]. Poslední modifikace 2017-04-24 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Scientific_notation#E-notation>

[29] Wikipedie - otevřená encyklopedie: Unicode [online]. Poslední modifikace 2017-05-02 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Unicode>

[30] Oficiální stránka SecureSocial: Play modul [online]. [cit. 2017-05-15]. Dostupné na URL: <http://www.securesocial.ws/>

[31] Oficiální stránka µpickle: serializér a deserializér JSON [online]. [cit. 2017-05-15]. Dostupné na URL: <http://www.lihaoyi.com/upickle-pprint/upickle/>

[32] Wikipedie - otevřená encyklopedie: Protokol OAuth 1.0 [online]. Poslední modifikace 2010-04-01 [cit. 2017-05-15]. Dostupné na URL: <https://tools.ietf.org/html/rfc5849>

[33] Wikipedie - otevřená encyklopedie: Phishing [online]. Poslední modifikace 2017-05-15 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Phishing>

[34] Oficiální dokumentace: Akka [online]. [cit. 2017-05-15]. Dostupné na URL: <http://doc.akka.io/docs/akka/2.5/AkkaScala.pdf>

[35] Oficiální dokumentace: Akka stream [online]. [cit. 2017-05-15]. Dostupné na URL: <http://doc.akka.io/docs/akka/2.4.14/scala/stream/index.html>

[36] Wikipedie - otevřená encyklopedie: IETF [online]. Poslední modifikace 2017-04-16 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force>

[37] Oficiální dokumentace k Akka-http: WebSocket [online]. [cit. 2017-05-15]. Dostupné na URL: <https://www.playframework.com/documentation/2.5.x/ScalaWebSockets>

[38] Oficiální stránka projektu: Pircbotx [online]. [cit. 2017-05-15]. Dostupné na URL: <https://github.com/thelq/pircbotx>

[39] Wikipedie - otevřená encyklopedie: Data access object [online]. Poslední modifikace 2017-05-03 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Data_access_object>

[40] Wikipedie - otevřená encyklopedie: Docker [online]. Poslední modifikace 2017-05-06 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Docker_%28software%29>

[41] Dokumnetace OpenShift: Nová aplikace [online]. [cit. 2017-05-15]. Dostupné na URL: <https://docs.openshift.org/latest/dev_guide/application_lifecycle/new_app.html#dev-guide-new-app>

[42] Wikipedie - otevřená encyklopedie: regulární výraz [online]. Poslední modifikace 2017-05-14 [cit. 2017-05-15]. Dostupné na URL: <https://en.wikipedia.org/wiki/Regular_expression>

[43] Oficiální stránka: ngIRCd [online]. [cit. 2017-05-15]. Dostupné na URL: <https://ngircd.barton.de/index.php.en >

[44] Oficiální stránka: HexChat klient [online]. [cit. 2017-05-15]. Dostupné na URL: <https://hexchat.github.io/>

Seznam příloh

Příloha 1. CD se zdrojovými soubory.

Příloha 2. SQL scripty:
SQL příkaz pro tvorbu tabulky žetonů:

```
create table "token" (
  "uuid" VARCHAR NOT NULL,
  "email" VARCHAR NOT NULL,
  "creationTime" TIMESTAMP NOT NULL,
  "expirationTime" TIMESTAMP NOT NULL,
  "isSignUp" BOOLEAN NOT NULL
);
```

SQL příkaz pro tvorbu tabulky uživatelů:

```
create table "user" (
  "id" BIGINT GENERATED BY DEFAULT AS IDENTITY(START WITH 1) NOT NULL
PRIMARY KEY,
  "userId" VARCHAR NOT NULL,
  "providerId" VARCHAR NOT NULL,
  "firstName" VARCHAR NOT NULL,
  "lastName" VARCHAR NOT NULL,
  "fullName" VARCHAR NOT NULL,
  "email" VARCHAR,
  "avatarUrl" VARCHAR,
  "authMethod" VARCHAR NOT NULL,
  "token" VARCHAR,
  "secret" VARCHAR,
  "accessToken" VARCHAR,
  "tokenType" VARCHAR,
  "expiresIn" INTEGER,
  "refreshToken" VARCHAR
);
```

Příloha 3. Parametry konfigurace kanálů ngIRCd serveru v souboru *ngircd.conf*:

```
        [Channel]
                # Name of the channel
                Name = #TheName

                # Topic for this channel
                Topic = a great topic

                # Initial channel modes
                ;Modes = tnk

                # initial channel password (mode k)
                ;Key = Secret
```

```
        # Key file, syntax for each line: "<user>:<nick>:<key>".
        # Default: none.
        ;KeyFile = /etc/#chan.key

        # maximum users per channel (mode l)
        ;MaxUsers = 23

[Channel]
        # Name of the channel
        Name = #TheName2

        # Topic for this channel
        Topic = a great topic

        # Initial channel modes
        ;Modes = tnk

        # initial channel password (mode k)
        ;Key = Secret

        # Key file, syntax for each line: "<user>:<nick>:<key>".
        # Default: none.
        ;KeyFile = /etc/#chan.key

        # maximum users per channel (mode l)
        ;MaxUsers = 23
```