Masarykova univerzita
Fakulta informatiky


# Od archívu elektronickej konferencie k znalosti databáz

Diplomová práca

**Matej Briškár**

Brno, jar 2014

## Prehlásenie

Prehlasujem, že táto diplomová 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.

Matej Briškár

**Vedúci práce:** Mgr. Marek Grác, Ph.D.

## Poďakovanie

Rád by som tento úsek práce využil na poďakovanie sa pánovi Mgr. Marekovi Grácovi za odborné vedenie, poskytnutú motiváciu a v neposlednom rade za čas, ktorý mi pri tvorbe tejto práce venoval. Moje poďakovanie patrí aj Ing. Lukášovi Vlčekovi za trpezlivosť a odborné vedenie pri nasadzovaní vlastnej inštancie projektu Searchisko.

## Zhrnutie

Práca sa zaoberá tvorbou aplikácie na spracovanie a archivovanie internetových správ. V prvých kapitolách práca opisuje dostupné riešenia a príčiny potreby tvorby novej implementácie. V d'alších kapitolách sú postupne opísané dostupné nástroje s krátkym opisom a argumentáciou výberu jedného z nich. Posledné kapitoly tejto práce sa zaoberajú implementačnými detailami vytvorenej aplikácie s názvom *Mailinglist Online*.

## Kľúčové slová

elektronické konferencie, archív, Java EE, Searchisko, JSF, MongoDB, openshift

# Obsah

# 1 Úvod

Človek je tvor spoločenský a je pre neho typický život v skupinách. Práve existencia takýchto skupín dáva jedincom množstvo výhod. Môžu medzi sebou zdieľať vedomosti, navzájom si pomáhať alebo medzi sebou zdieľať životné zážitky. Vzájomná komunikácia má tak v rámci spoločenského života neodmyslitel'né zastúpenie, a preto je jednou z najväčších motivácií pri vývoji technológií. Postupným rozvojom internetu tak nabrala vzájomná výmena informácií medzi l'ud'mi úplne nové rozmery. Komunikácia naprieč rôznymi štátmi alebo svetadielmi sa stala finančne nenáročná a l'ahko dostupná. Takáto virtuálna komunikácia sa môže deliť na základe odozvy na interaktívnu (s možnosťou okamžitej odozvy) a neinteraktívnu (bez možnosti okamžitej odozvy).

Jednou z najdôležitejších foriem modernej komunikácie je spôsob zasielania emailových správ. V rámci novodobej komunikácie si email postupom času nadobudol formálne postavenie. Patrí medzi najpoužívanejších zástupcov neinteraktívnej komunikácie, dokonca je tento spôsob komunikácie oveľa starší ako samotný internet. Za pravdepodobne prvý emailový systém sa označuje MAILBOX, ktorý bol využívaný na MIT (*Massachusetts Institute of Technology*) od roku 1965 [1]. Postupným zdokonaľovaním od úplných začiatkov nabral dnešnú podobu, s ktorou sa každodenne stretávame.

Jedným z častých využití emailových správ je komunikácia v rámci skupín s dopredu definovanou témou konverzácie. Takéto skupiny sú označované ako elektronické konferencie (angl. *mailing lists*). Z hľadiska využitia sa uvádzajú 2 základné spôsoby využitia konferencií:

**Oznamovacie** Všetkým záujemcom prihláseným na odber sú zasielané novinky, oznamy alebo reklamy.

**Diskusné** Prihlásení odberatelia majú možnosť zasielať emaily, ktoré sú rozposlané všetkým ostatným odberateľom.

Elektronické konferencie sú používané vo veľkej miere v oblasti informačných technológií. S ich využitím je možné sa stretnúť hlavne pri vývoji otvorených (anglicky *open source*) programov. Vďaka týmto skupinám má tak človek možnosť oboznámiť celú skupinu vývojárov o probléme, na ktorý narazil, alebo požiadať o pomoc nadšencov v rámci skupiny venovanej danému produktu.

Cieľom tejto práce je vytvorenie otvorenej aplikácie, ktorá bude spravovať a vhodným spôsobom sprístupňovať emaily zasielané do elektronických konferencií.

Prvá, analytická časť práce sa zaoberá opisom problému, definovaním funkcionalít, ktoré sú očakávane od výslednej aplikácie, ako aj opisom nástrojov vhodných pre využitie s následným argumentovaním výberu.

Druhá, implementačná časť práce opisuje implementačné detaily, spôsob a formu ukladania dát, rozhranie komunikácie medzi komponentmi aplikácie a spôsob nasadenia a pustenia aplikácie.

# 2 Analýza problému

Jednou z najdôležitejších častí problému je uchovávanie a správa emailov zasielaných v rámci udržiavaných elektronických konferencií. Email má okrem samotnej informácie, ktorú si používatelia preposielajú medzi sebou, aj množstvo iných informácií, ktoré je potrebné uchovávať. Celkovo sa tak emailová správa skladá z dvoch základných častí:

**Hlavička** Obsahuje informácie o správe, ako sú napríklad: adresa odosielateľa, adresa príjemcu, predmet správy, kódovanie a pod.

**Telo** Obsah samotnej správy.

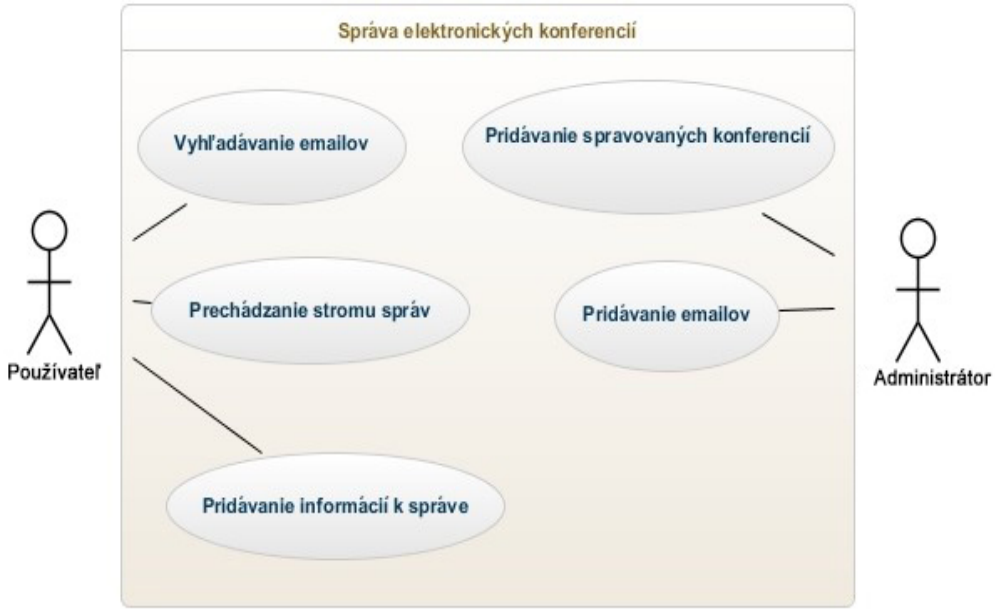
V niektorých z najpoužívanejších konferencií je prísun emailových správ vysoký, dosahujúci okolo 1000 správ za deň. Aplikácia sprístupňujúca takéto správy musí preto brať na vedomie potrebu ukladania a následnej správy veľkého množstva dát. Používateľské rozhranie aplikácie musí umožniť používateľom jednoduchý prístup k rôznorodým odpovediam na otázky týkajúce sa uchovávaných emailov. Používateľ tak bude schopný napríklad:

1. Vyhľadávať emaily na základe hľadaného textu v správe, elektronickej adrese, predmetu správy a pod.
2. Prechádzať strom správ (tzv. *thread*)
3. Pridávať informácie k správe (pridávať jednoslovné návestia určujúce zaradenie správy podľa obsahu a pod.)

Očakávané spôsoby využitia sú znázornené taktiež na diagrame prípadov použitia na obrázku 2.1. Pri tvorbe aplikácie je nutné dbať na bezpečnosť, škálovateľnosť a jednoduchosť rozhrania.

## 2.1 Analýza existujúcich riešení

Pred opisom tvorby aplikácie je vhodné uviesť dôvody tvorby aplikácie na správu emailov zasielaných do elektronických konferencií. Hlavným dôvodom sú súčasné nedostatočné aplikácie, ktorých počet nie je veľký a ktoré problém neriešia úplne alebo sú už do istej miery zastaralé. Podstatnými nedostatkami sú neprívetivé používateľské rozhranie, chýbajúca podpora pridávania atribútov (napr. značiek) alebo neefektivita




Obr. 2.1: Diagram prípadov použitia

skladovania. Táto kapitola zhŕňa najznámejšie využívané aplikácie na uchovávanie elektronických konferencií, konkrétne *gmane* a *pipermail*.

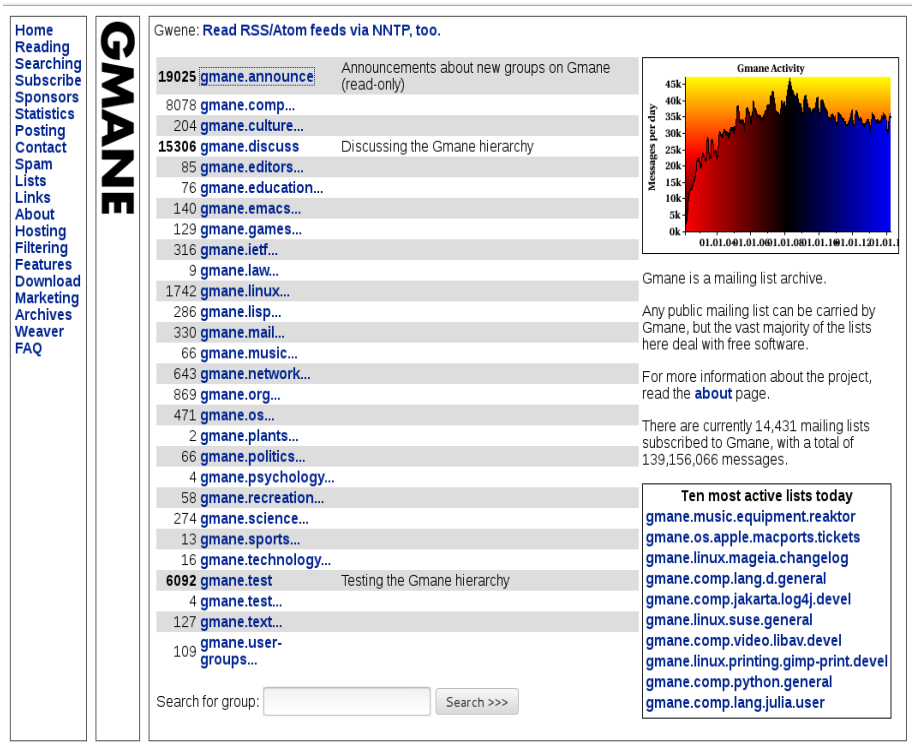
### 2.1.1 Gmane

Gmane je aplikácia, ktorej vývoj začal Lars Magne Ingebrigtsen, nórsky vývojár, v roku 2001 a využíva sa dodnes. Jedinečnosťou implementácie je to, že emaily nie sú mazané pokiaľ o to nie je špecificky požiadané. Tým je možnosť pristupovať k naozaj starým záznamom. Gmane má rozsiahlu množinu funkcií, ktoré ponúka. Medzi najdôležitejšie funkcie patrí [2]:

1. Detekcia spamu využívaním programu *SpamAssassin*, ktorý dokáže spam detekovať na základe hľadania zhody v obsahu.
2. Odstraňovanie vírusov pomocou skenovania.
3. Podpora RSS formátov (rodina formátov používaných pre obsluhu aktualít)

4. Možnosť prezerania emailov v rôznych rozhraniach
5. Podpora prehľadávania emailov
6. Zbieranie štatistík o aktivite

Medzi hlavné nevýhody aplikácie však patrí zastaralejšie používateľské rozhranie bez podpory pridávania značiek a iných atribútov k emailom. Tieto problémy sú riešené vo výslednej aplikácií tvorenej touto diplomovou prácou.




Obr. 2.2: Ukážka aplikácie gmane. Zdroj: http://gmane.org/

### 2.1.2 Pipermail

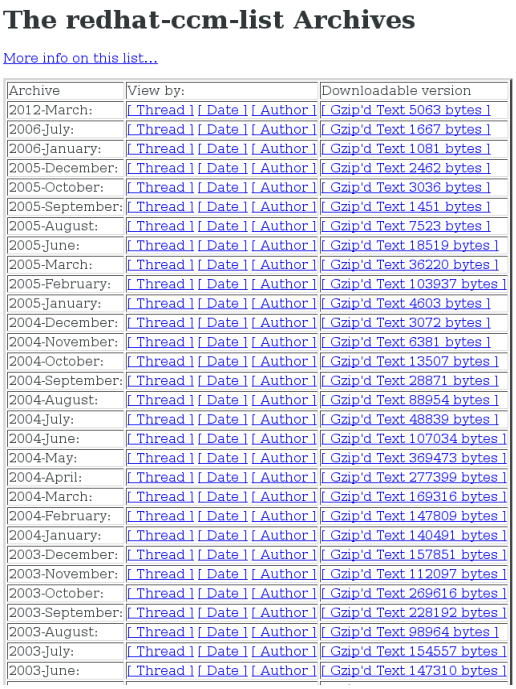
Pipermail je podprogram aplikácie *GNU Mailman*, ktorá patrí medzi najpoužívanejšie programy pre celkové spracovávanie a správu elektronických konferencií, čo sa značne podpísalo na vysokú používanosť

programu pipermail. Samotný Pipermail je modul, ktorý má na starosti správu archívov elektronických konferencií. Pipermail ponúka iba základnú funkcionalitu a možnosť prístupu k archivovaným správam. Ponúka tak [3]:

1. Rozdelenie správ podľa dátumu do skupín, pričom ponúka možnosť stiahnutia si kompresovaných balíčkov jednotlivých konferencií. Prístup je obmedzený len v rámci jednej z uvedených konferencií.

2. Prezeranie emailov podľa dátumu, v rámci vlákna odpovedí alebo na základe autorov.

Medzi nevýhody programu pipermail patrí jeho pomerne zlá používateľská prívetivosť bez podpory komplexného prehľadávania emailov. Niektoré problémy sa dajú odstrániť presmerovaním emailov na externú aplikáciu.



**The redhat-ccm-list Archives**

More info on this list...

| Archive | View by: | Downloadable version |
|---|---|---|
| 2012-March: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 5063 bytes ] |
| 2006-July: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 1667 bytes ] |
| 2006-January: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 1081 bytes ] |
| 2005-December: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 2462 bytes ] |
| 2005-October: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 3036 bytes ] |
| 2005-September: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 1451 bytes ] |
| 2005-August: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 7523 bytes ] |
| 2005-June: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 18519 bytes ] |
| 2005-March: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 36220 bytes ] |
| 2005-February: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 103937 bytes ] |
| 2005-January: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 4603 bytes ] |
| 2004-December: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 3072 bytes ] |
| 2004-November: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 6381 bytes ] |
| 2004-October: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 13507 bytes ] |
| 2004-September: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 28871 bytes ] |
| 2004-August: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 88954 bytes ] |
| 2004-July: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 48839 bytes ] |
| 2004-June: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 107034 bytes ] |
| 2004-May: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 369473 bytes ] |
| 2004-April: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 277399 bytes ] |
| 2004-March: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 169316 bytes ] |
| 2004-February: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 147809 bytes ] |
| 2004-January: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 140491 bytes ] |
| 2003-December: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 157851 bytes ] |
| 2003-November: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 112097 bytes ] |
| 2003-October: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 269616 bytes ] |
| 2003-September: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 228192 bytes ] |
| 2003-August: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 98964 bytes ] |
| 2003-July: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 154557 bytes ] |
| 2003-June: | [ Thread ] [ Date ] [ Author ] | [ Gzip'd Text 147310 bytes ] |

Obr. 2.3: Ukážka programu pipermail. Zdroj: http://www.redhat.com/archives/redhat-ccm-list/

# 3 Výber technológie

Pri výbere technológií použitých na implementáciu sa kladol dôraz na mnou už nadobudnuté skúsenosti, projekty patriace do komunity *JBoss* a ich použiteľnosť pri tvorbe veľkých aplikácií. Z týchto dôvodov bol vybraným jazykom programovací jazyk *Java*.

Tvorba internetovej aplikácie v jazyku Java však pozostáva z využitia mnohých knižníc a nástrojov, z ktorých je potrebné vybrať ten najvhodnejší. Mnohokrát sú rozdiely medzi jednotlivými možnosťami minimálne, avšak môžu mať veľký dopad na výslednú aplikáciu. Táto kapitola preto v sekciách a v krátkosti opisuje dostupné nástroje na postupne všetky aspekty vývoja tvorenej aplikácie. Celkovo je tak opísaný výber:

1. Aplikačného servera na nasadenie aplikácie v sekcii 3.1
2. Databázového systému na správu veľkého množstva dát v časti 3.2.
3. Nástroja na vyhľadávanie reťazca v množstve textových dát v sekcii 3.3.
4. Nástroja na tvorbu webového rozhrania v jazyku Java v časti 3.4.

## 3.1 Aplikačné servery

Tvorba mnohých internetových aplikácií v jazyku Java úzko nadväzuje na špecifikácie *Java EE*, ktoré rozdeľujú problémy spojené s tvorbou internetových aplikácií v jazyku Java do jednotlivých častí. Každá špecifikácia je súbor vo formáte pdf, ktorý jednoznačne opisuje a určuje štandardné rozhranie poskytovaných služieb. Tvorba takýchto špecifikácií je spojená s tvorbou testov, ktorými musí prejsť každá implementácia danej špecifikácie, ako aj tvorba referenčnej implementácie, ktorá udáva modelový príklad takejto implementácie. Medzi referenčné implementácie špecifikácie *Java EE* patrí:

| Referenčná implementácia | Špecifikácia |
|---|---|
| Oracle JAXB | Spájanie dát vo formáte *XML* do objektov |
| Oracle JAX-WS | Internetové služby využívajúce formát *XML* |
| JBoss Weld | Spravovanie kontextu a vkladanie inštancií (*Contexts and Dependency Injection*) |
| Hibernate Validator | Validácia objektov typu *Bean* (*Bean Validation*) |
| Oracle Jersey | Tvorba rozhrania typu *REST* (*JAX-RS*) |
| Oracle Mojarra | Nástroj *JavaServer Faces* |
| Oracle GlassFish | Nástroje *Java Servlet 3.1*, *JavaServer Pages 2.2* |
| EclipseLink | Rozhrania perzistencie dát *Java Persistence API 2.1* |
| Oracle Glassfish | Nástroj *Enterprise JavaBeans 3.2* |

Implementácie služieb opísaných pomocou spomínaných špecifikácií sú následne poskytované rôznymi dostupnými riešeniami, ktoré spájajú implementácie jednotlivých špecifikácií do výsledného produktu. Pri splnení množiny špecifikácií je takémuto výslednému produktu udelený certifikát. Pri tvorbe internetových aplikácií je výber z týchto implementácií jedným z kľúčových rozhodnutí, pretože každé riešenie je sprevádzané špecifickými výhodami, resp. nevýhodami. V tejto sekcii sú postupne opísané najznámejšie a najpoužívanejšie riešenia, ktoré implementujú niektoré zo špecifikácií.

### 3.1.1 Apache Tomcat

*Apache Tomcat* je projekt typu *open source*, ktorý je vyvíjaný spoločnosťou *The Apache Software Foundation* [4]. *Tomcat* je *servlet container*, čo v tejto terminológii znamená, že implementuje dve hlavné špecifikácie *Java Servlet* a *JavaServer Pages* zo všetkých špecifikácií uvedených v nadsekcii 3.1 [5]. *Servlet container* nezaberá veľa pamäte, kedže implementuje iba základnú funkcionalitu pre tvorbu aplikácií, čím je vhodnejší na jednoduchšie projekty, ktoré nevyužívajú náročnejšie konštrukcie poskytované rozhraním *Java EE*. *Apache Tomcat* je

na základe prieskumu, ktorý bol robený na používateľoch spoločnosti *Plumbr*, najviac využívaným produktom na nasadenie internetových aplikácií v jazyku Java [6].

V prípade potreby využitia niektorého z rozhraní špecifikovaných v iných častiach *Java EE* je možné k serveru *Tomcat* pripojiť niektorú z konkrétnych implementácií danej špecifikácie, najvhodnejšie implementáciu priamo od spoločnosti *The Apache Software Foundation*. Spojením všetkých implementácií vytvorených spoločnosťou a pripojením ku kontejneru *Apache Tomcat* vznikol projekt *Apache TomEE*, ktorý tak spĺňa väčšinu zo špecifikácií a je nositeľom certifikácie *Java EE 6 Web Profile Compatible Implementation*.

### 3.1.2 GlassFish

*GlassFish* je aplikačný server vyvíjaný spôsobom *open source*, pôvodne tvorený spoločnosťou *Sun Microsystems*, momentálne sponzorovaný spoločnosťou *Oracle Corporation*. Program je licencovaný pomocou dvoch typov licencií, *Common Development and Distribution License* (CDDL) a *GNU General Public License* (GPL). *GlassFish* je referenčná implementácia aplikačného servera, čo znamená, že ide o modelové riešenie špecifikácií uvedených v sekcii 3.1. Je tvorený z podčastí, referenčných implementácií jednotlivých špecifikácií. *GlassFish* je prvá inštancia, ktorá implementuje všetky spomínané špecifikácie [7] pre verziu špecifikácie *Java EE 7*.

### 3.1.3 Wildfly

*Wildfly*, v minulosti známy pod menom *JBoss AS*, je aplikačný server vyvíjaný ako *open source* spoločnosťou *Red Hat* v rámci komunity *JBoss*. Vývoj aplikačného servera začal v roku 1999 vytvorením projektu *JBoss*, ktorý implementoval špecifikáciu EJB, časť zo špecifikácie *J2EE*. Odvtedy projekt narástol do aplikačného servera spĺňajucého všetky časti špecifikácií *Java EE*. *Wildfly* pozostáva z modulov, ktoré spľňajú jednotlivé špecifikácie a z ktorých niektoré sú ich samotnými referenčnými implementáciami. *Wildfly* je licencovaný s využitím lincencie *GNU Lesser General Public License* (LGPL) [8].

Pri tvorbe výslednej aplikácie sú preferované produkty, ktoré sú vyvíjané v rámci komunity JBoss. V budúcnosti je taktiež pravdepodobné

využitie náročnejších nástrojov definovaných v špecifikácii *Java EE*. Preto bol vybraným aplikačným serverom na nasadenie aplikácie server *Wildfly*.

## 3.2 Výber databázy

Kedže hlavným cieľom aplikácie je udržiavanie a sprístupňovanie dát, spôsob ich uloženia má d'alekosiahlé dopady na kvalitu implementácie. Z hľadiska uloženia dát sa ponúkajú 3 možnosti, ktoré sú bližšie opísané v nasledujúcich podsekciách.

### 3.2.1 Databázy SQL

SQL (resp. relačné) databázy sú tradičným a známym spôsobom ukladania veľkého množstva dát založenom na relačnom modele. Dáta sú uložené v navzájom prepojených reláciách (tabuľkách). Pri návrhu tabuliek sa využívajú tzv. normálne formy[1], ktoré sú odporúčaním pre rozdelenie dát medzi viaceré tabuľky [9]. Medzi výhody patrí:

1. nezávislosť na konkrétnom systéme. Zmena databázového systému nemá v bežných prípadoch zásadný dopad na samotnú aplikáciu alebo tvar uložených dát.
2. vysoká podpora transakcií
3. najznámejší a najviac preskúmaný prístup

Nevýhody:

1. neboli navrhnuté na prácu v prostredí viacerých počítačov, s čím je spojený aj netriviálny problém s ich nasadením v takom prostredí (riešený príchodom tzv. newSQL databáz) [10]
2. nutnosť prevodu dát do relačnej formy

Aplikácia ukladá veľké množstvo dát, pričom je pravdepodobné, že budú dáta rozložené na viaceré servery. Popri tom nie je nutná vysoká podpora transakcií, a preto je vhodnejšie využiť alternatívu k bežným relačným databázam.

1. Normálne formy vynucujú rozdelenie dát medzi viaceré tabuľky, čím je zaručená lepšia škálovateľnosť riešenia

### 3.2.2 Databázy NoSQL

Databázy NoSQL patria medzi netradičnejšie metódy riešenia uloženia dát. Ide o široký pojem, ktorý zahŕňa takmer všetky databázy okrem už spomínaných relačných. Záznamy nie sú uložené relačne, ako je tomu v relačných databázach, ale väčšinou len ako dvojice kľúč-hodnota alebo formou záznamov podobných objektom. Väčšina z databáz sa vyhýba potrebe použitia náročných dátovych operácií, ako sú napríklad operácia spojenia (*join*) [11]. Medzi výhody databáz NoSQL patrí:

1. efektivita a rýchlosť prístupu k záznamom (pri správnom využití)
2. nie je nutné ukladať dáta v striktne danej schéme
3. väčšinou ide o otvorené projekty, takže je to považované za finančne dostupnejšie riešenie ukladania dát
4. pomerne jednoduché nasadenie na viaceré servery

Použitie databáz NoSQL má však aj svojé nevýhody, ako napríklad:

1. závislosť na zvolenej implementácii
2. nový spôsob ukladania dát s malou komunitou ľudí
3. slabá až žiadna podpora transakcií

Tieto databázy sa ešte ďalej rozdeľujú do skupín podľa spôsobu uloženia dát a paradigmy, ktorú pri uložení používajú. Existuje niekoľko rozdelení do skupín, avšak väčšinou ide iba o neformálne zaužívanú kategorizáciu týchto databáz. Jedno z najpoužívanejších rozdelení je bližšie opísané v nasledujúcom texte [12].

Key-Value

Radia sa medzi najjednoduchšie z databáz NoSQL. Dáta sú uložené vo forme dvojíc kľúč,hodnota [12]. Veľmi rýchle vyhľadanie hodnoty k danému kľúču, pre zložitejšie hľadania nutnosť ukladania rozšíreného kľúča o ďalšie hodnoty (napr. $meno\_priezvisko\_den$ môže byť kľúč, ktorý má k sebe priradenú hodnotu obsahujúcu odpoveď na otázku prítomnosti danej osoby v daný deň).

Tvorená aplikácia musí ukladať mnoho správ, pričom je kladený veľký dôraz na jednoduchosť a rýchlosť vyhľadávania v správach, preto je tento spôsob ukladania nevhodný.

### Column Databases

Databázy, pre ktoré je príznačné ukladanie na základe stĺpcov. V krátkosti sa dá povedať, že tento typ ukladania je rozšírením princípu key-value s tým, že pod kľúčom nie je uložená iba jedna hodnota ale množina dvojíc (kľúč a hodnota). Je veľmi dôležité dať si záležať na tvorbe schémy ukladania dát, pretože pri nevhodnom uložení sa potrebné dáta nedajú získať. Najznámejším zástupcom je *BigTable* od spoločnosti Google [12].

### Graph Database

Záznamy sú uchovávané pomocou grafových uzlov a hrán. Takéto uloženie je najvhodnejšie v prípade potreby požiadaviek, ktoré sú ľahšie modelovateľné využitím grafov. Najväčšie zrýchlenie nastáva v prípade požiadaviek, ktoré sú cielené na dlhé cestovanie takýmito grafmi. Tento spôsob je vhodný len na špecifické požiadavky. Nie je vhodný na klasické vyhľadávanie záznamov vo všetkých dátach a náročnejšie požiadavky. Preto nie je taktiež vhodný pre tvorenú aplikáciu správy emailov [12].

### Document Store

Dáta sú uložené vo forme záznamov veľmi blízkym objektom z objektového programovania. Najčastejšie sa využíva uloženie vo forme JSON alebo BSON. Najznámejšími predstaviteľmi sú CouchDB alebo MongoDB, no aj napriek tomu, že patria do rovnakej skupiny databáz NoSQL, je medzi nimi veľký rozdiel [12]. MongoDB dokáže iba master-slave replikáciu, využíva BSON, je vhodný na dynamické dáta a je celkovo častejšie využívaný. CouchDB dokáže master-master replikáciu, využíva JSON a je vhodný na dáta, na ktoré sú vhodné preddefinované požiadavky.

Výsledná aplikácia bude ukladať veľké množstvo dát, preto sa očakáva nasadenie na viaceré výpočetné jednotky a replikácia dát v rámci viacerých skladovacích jednotiek. Je kladený veľký dôraz na rýchlosť prístupu a výsledkom väčšiny požiadaviek je množina emailov. Preto je vhodné ukladať dáta ako dokumenty za pomoci MongoDB. Kedže je

ukladanie dát dôležitou súčasťou výslednej aplikácie, je táto databáza opísaná hlbšie v nasledujúcej sekcii 3.2.3.

### 3.2.3 MongoDB

Ako už bolo sčasti spomenuté v sekcii 3.2.2, *MongoDB* je projekt vyvíjaný ako *open source*. Je najpoužívanejším zástupcom databáz zo skupiny *Document Store* a patrí aj celkovo medzi najrozšírenejšie databázy typu *NoSQL* vo svete [13]. Hlavné dôvody širokého uplatnenia sú jednoduchosť, programátorská prívetivosť (dáta sú ukladané v objektoch, jednoduchá inštalácia) a moderný prístup.

S databázou sa pracuje pomocou ovládačov, ktoré značne uľahčujú prístup k databáze v danom programovacom jazyku. Pomocou týchto ovládačov sa dajú vytvoriť indexy, nové tabuľky, ukladať a mazať objekty. Momentálne sú pre databázu *MongoDB* vytvorené ovládače pre správu z 13 rôznych programovacích jazykoch, medzi ktoré patria napríklad Java, Ruby a Python [14].

Každá databáza v *MongoDB* obsahuje kolekcie (ekvivalent tabuľkám v relačných databázach), v ktorých sú ukladané jednotlivé objekty. Kolekcie nešpecifikujú žiadnu schému, ktorej sa musia ukladané objekty držať, preto nenastáva chybový stav v prípade, ak sa jednotlivé uložené objekty v kolekcii navzájom úplne líšia. V prípade potreby udržania istej schémy je nutné takéto obmedzenie uplatňovať v samotnej aplikačnej vrstve pred každým vkladaním objektu do databázy. Takýto model má však svoje výhody, ako napríklad jednoduchá migrácia alebo zmena ukladaných entít, pretože zmenu stačí vykonať iba v aplikačnom kóde.

*MongoDB* ponúka množstvo funkcionalít, ktorými sa líši od ostatných databáz. Tieto body je vhodné brať do úvahy v prípade rozhodovania sa o využití tejto databázy. Medzi najzákladnejšie ponúkané funkcionality patrí [15]:

**Podpora indexovania** Možnosť vytvoriť index nad každým atribútom s možnosťou udať usporiadanie pre objekty s rovnakou hodnotou indexovaného atribútu. Okrem toho je poskytnutá možnosť indexovania polí, tvorby textových alebo priestorových indexov.

**Podpora Map/Reduce funkcií** *Map/reduce* je model výpočtu, ktorý sa používa pri spracovaní veľkého množstva dát pomocou paralelného výpočtu rozdelením problému na menšie podproblémy a ich

následné priradenie medzi viaceré výpočetné jednotky a dátové sklady.

**Podpora replikácie a škálovania** Jednoduchá možnosť nastavenia replikovania dát alebo spravovania dát v rámci viacerých inštancií. Ide o dôležitú súčasť tvorenej aplikácie, preto je dôkladnejšie opísaná v sekciách nižšie.

MongoDB, ako väčšina databáz NoSQL, dosahuje v niektorých oblastiach dobrých výsledkov vďaka tomu, že nepodporuje všetky funkcionality, ktoré sú podporované klasickými relačnými databázami. Kvôli tomu má teda táto databáza aj nevýhody, ktoré je nutné taktiež brať do úvahy pri rozhodovaní o využití takéhoto spôsobu ukladania dát. Medzi najdôležitejšie nevýhody patrí:

**Jednoduchá atomicita** Databáza podporuje len atomické operácie vykonané nad jedným dokumentom uloženým v databázi. V prípade náročnejších operácií (ako je napríklad presun peňazí medzi účtami), je nutné túto funkcionalitu zaručiť v aplikácii. MongoDB tak celkovo nepodporuje 4 základné vlastnosti transakcií (atomicita, konzistencia, izolácia, trvanlivosť) a je ich nutné riešiť v aplikácii.

**Master-slave replikácia** [1] Replikácia v MongoDB je výlučne len typu master-slave bez možnosti využitia replikácie typu master-master. Replikácia je presnejšie opísaná v nasledujúcej sekcii.

**Chýbajúca podpora operácie join** Samotný návrh a spôsob ukladania dát v MongoDB je navrhnutý tak, aby minimalizoval potrebu spájania rôznych záznamov, pretože ide o veľmi náročnú operáciu nad dátami. Operácia join teda nie je v databáze podporovaná.

### 3.2.4 Replikácia a horizontálne škálovanie

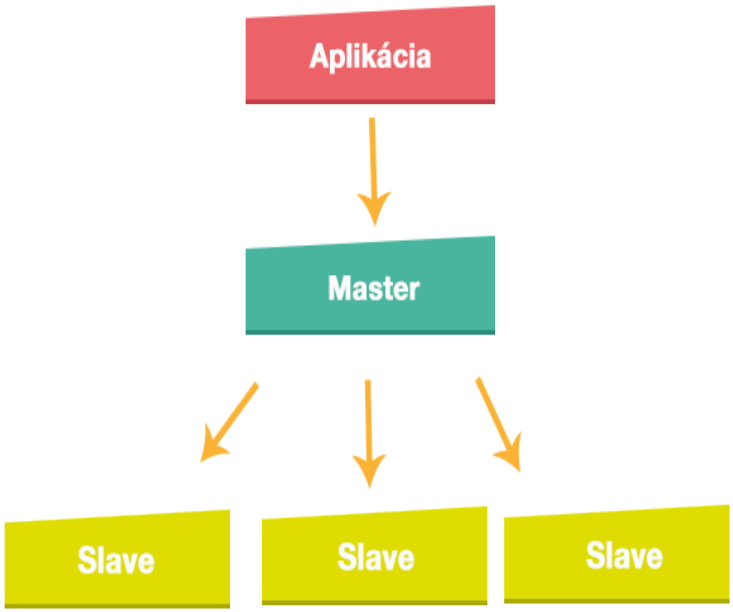
Jednou z hlavných výhod databázy MongoDB je jej schopnosť jednoduchého horizontálneho škálovania a replikácie dát. Túto funkcionalitu je vhodné využiť v prípade veľkého počtu uložených dát alebo zahltenia servera požiadavkami na ukladané dáta.

1. Master-slave replikácia je spôsob uchovávania totožných dát na rôznych serveroch, pričom jeden zo serverov je vždy označený za hlavný a má jediný právo zápisu

### Replikácia

Replikácia značí ukladanie totožných dát vo viacerých inštanciách. Takýmto ukladaním sa predíde strate dát v prípade zlyhania niektorej z inštancií a zaručí sa vyššia dostupnosť informácií, pretože požiadavky môžu byt smerované na viaceré databázové jednotky.

*MongoDB* využíva replikáciu typu *master-slave*, čo znamená, že zápisy do replikovanej množiny sú vždy smerované len na jedného, hlavného (*master*) zastupiteľa tejto množiny. Vďaka tomu nenastávajú problémy spojené s potrebou synchronizovania zápisov medzi členmi tejto množiny. Hlavný zastupiteľ replikovanej množiny po zápise dát do databázy zapisuje zmenu taktiež do logovacieho súboru s názvom *oplog*. Ostatní zastupitelia (*slaves*) sledujú zmeny vykonané v súbore *oplog* a aplikujú ich na záznamy, ktoré spravujú. V prípade zlyhania hlavného zastupiteľa sa spustí voľba nového z ostatných dostupných členov [16].




Obr. 3.1: Replikácia v MongoDB

Horizontálne škálovanie

Horizontálne škálovanie, taktiež označované anglicky ako *sharding*, je rozdelenie a následné ukladanie celej množiny dát naprieč viacerými databázovými inštanciami. Takéto ukladanie má za následok zvýšenie dostupnej kapacity ako aj vyššiu odozvu, pretože všetky požiadavky už nie sú smerované len na jednu inštanciu.

Pri škálovaní zohrávajú úlohu nasledujúce tri rôzne druhy inštancií [17]:

**Smerovač** Vstupný bod do databázy, pri požiadavke získa potrebné informácie o ostatných serveroch od konfiguračných inštancií a presmeruje požiadavky na správnu datovú inštanciu.

**Konfiguračná inštancia** Obsahuje informácie o škálovaní, ako napríklad informácie o dátových inštanciách.

**Dátová inštancia** Anglicky *shard*, je inštancia, ktorá spravuje podmnožinu uchovávaných objektov, väčšinou ide o replikovanú množinu (dáta v rámci dátovej inštancie sú ešte replikované). Pridávanie nových dátových inštancií je jednoduché volanie metódy na smerovači.

Hlavným princípom horizontálneho škálovania je rozdelenie dát medzi dátove inštancie (*shards*) na základe uchovávaného škálovacieho atribútu (*shard key*). Tento atribút musí byť zvolený vhodne, pretože určuje spôsob rozloženia dát naprieč jednotlivými dátovými inštanciami na základe postupného delenia rozsahu hodnôt v atribúte. Každá dátová inštancia má tak pridelený rozsah škálovacieho atribútu, ktorý spravuje. V prípade, že sa hodnoty atribútu nedajú d'alej deliť na menšie rozsahy, dátové inštancie budú nerovnomerne zaťažené bez možnosti jednoduchej nápravy.

### 3.2.5 Uloženie veľkých súborov

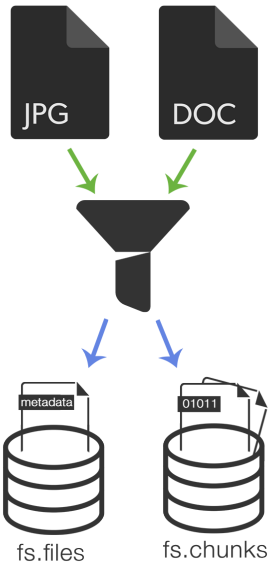
Jeden objekt v databáze *MongoDB* má limit pamäte, ktorú môže zaberať (aktuálne je limit nastavený na 16MB). V prípade ukladania väčších súborov, ako sú napríklad binárne súbory, tak nastáva problém uloženia. Tento problém rieši špecifikácia *GridFS*, ktorá ukladá dáta rozdelením


Obr. 3.2: Horizontálne škálovanie v databáze MongoDB

dátovej informácie do viacerých objektov obsahujúcich samotnú binárnu informáciu (*chunks*). Okrem týchto záznamov databáza uchováva objekt, ktorý obsahuje informácie o uloženom súbore (ako napr. názov a veľkosť súboru). Celkovo tak databáza ukladá súbory do dvoch kolekcií v rámci jednej databázy, ktorých názvy sú štandardne *fs.files* a *fs.chunks*. Tento spôsob delenia súborov do kolekcií je načrtnutý obrázkom 3.3. Názvy kolekcií, ktoré sa používajú na uchovávanie súborov, sa však dajú zmeniť, čím je taktiež možné ukladať súbory v rôznych kolekciách v rámci jednej databázy [18].

## 3.3 Vyhľadávanie v texte

Schopnosť vyhľadávania reťazca v texte je očakávanou súčasťou aplikácie, ktorá uchováva emaily. Databáza *MongoDB* podporuje vyhľadávanie typu *full-text*, čo značí, že dokáže vytvoriť index nad jedným alebo viacerými textovými atribútmi v ukladaných objektoch [19]. V rámci jednej kolekcie objektov je však možné vytvoriť iba jeden takýto index.




Obr. 3.3: Ukladanie binárnych súborov v databáze MongoDB

To je obmedzujúce v prípade potreby definovania rozdielneho spôsobu vyhľadávania.

Výsledná aplikácia však kladie dôraz na rýchlosť, s prípadnou potrebou nadefinovania priorít, preto nie je podpora textového indexovania v databáze *MongoDB* dostačujúca a je nutné na vyhľadávanie v texte využiť iný, externý nástroj. Najznámejšie externé nástroje, ktoré sa špecializujú na vyhľadávanie v texte, sú postupne uvedené v nasledujúcich podsekciách.

### 3.3.1 Solr

*Solr* je otvorená (angl. *open source*) platforma pre indexovanie typu *full-text*, ktorá je vyvíjaná v rámci projektu *Apache Lucene*. Nástroj je napísaný v jazyku Java a spúšťa sa ako samostatný vyhľadávací server využitím niektorého z kontajnerov (angl. *servlet containers*) vytvorených pre internetové aplikácie napísané v jazyku Java. Hlavným cieľom platformy *Solr* je zjednodušenie a rozšírenie knižnice *Lucene Search Library*, ktorú v pozadí využíva. Vyhľadávací nástroj poskytuje rozhranie typu

REST[1], čím umožňuje jednoduchú integráciu s rôznymi programovacími jazykmi. Využitím rozhrania REST je možné súbory nahrávať, meniť a vyberať z indexu. Medzi základné funkcie, ktoré *Solr* poskytuje, patrí [20]:

1. pokročilé vyhľadávanie typu full-text
2. indexovanie takmer bez oneskorenia
3. poskytovanie štatistík využitím JMX[2]
4. škálovateľnosť s využitím viacerých serverov

### 3.3.2 Elasticsearch

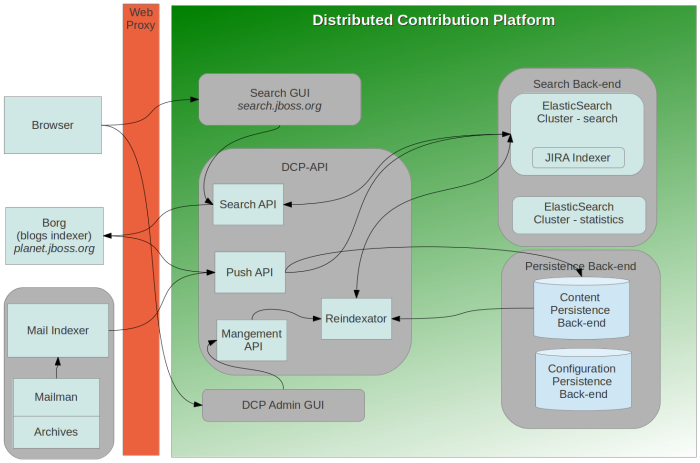
*Elasticsearch* je otvorený (angl. *open source*) nástroj veľmi podobný už spomínanej platforme *Solr*. Ponúka indexovanie typu *full-text*, rozhranie REST, indexovanie v reálnom čase. Elasticsearch je navrhnutý s dôrazom na jednoduchú horizontálnu škálovateľnosť, čo je hlavným ukazateľom v prípade tvorby aplikácie, ktorá má za cieľ uchovávať veľké množstvo dát. To je hlavným dôvodom, prečo bol *Elasticsearch* považovaný za správnu voľbu pri výbere nástroja na indexovanie textu obsiahnutého v skladovaných emailoch, ktorých je veľké množstvo [21]. Výsledná aplikácia využíva vlastnú inštanciu projektu Searchisko, ktorý je vyvíjaný v rámci pobočky spoločnosti Red Hat v Brne, a ktorá interne využíva práve nástroj ElasticSearch. Searchisko je špecifický projekt, je mu preto venovaná osobitná sekcia.

### 3.3.3 Searchisko

Searchisko je otvorený projekt, ktorý vznikol z dôvodu potreby lepšej vyhľadávacej služby pre projekty patriace do komunity *JBoss* bez nutnosti využívania rozdielných, externých systémov. Projekt poskytuje podporu pri indexácii a hľadaní obsahu s následnou možnosťou agregácie výsledkov. Využitím projektu *Searchisko* v aplikácii je vývoj odľahčený od

1. Rozhranie typu REST v jednoduchosti znamená, že je možné s nástrojom komunikovať protokolom HTTP
2. JMX (angl. *Java Management Extensions*) je technológia jazyka Java pre monitorovanie a správu aplikácií, zariadení a sietí

niektorých konfiguračných detailov spojených s konfiguráciou, spustením a správou verzií nástroja *Elasticsearch* [22]. Detaily architektúry projektu sú priblížené obrázkom 3.4.




Obr. 3.4: Architektúra projektu Searchisko prevzatá z oficiálnej dokumentácie projektu. Zdroj: https://github.com/searchisko/searchisko

### 3.3.4 Konfigurácia

Pri nasadení vlastnej inštancie projektu *Searchisko* je nutné konfigurovať niektoré súbory tak, aby vyhovovali dátam, ktoré je potrebné pomocou nástroja indexovať. Na ukladanie súborov do projektu je nutné sa registrovať vyplnením údajov o poskytovateľovi dát (*provider*) a ich uložením alebo poslaním s využitím rozhrania *REST* na server. V rámci údajov o poskytovateľovi dát je nutné uviesť typy dát, ktoré budú na server zasielané, ako aj typ indexu, ktorý bude pre dáta využitý.

**Ukážkový konfiguračný súbor poskytovateľa pre aplikáciu Mailinglist Online:**

```
{
  "name": "mlonline",
  "description" : "Mailinglist online is an application
       for managing the mailing lists",
  "contact_email" : "mbriskar@gmail.com",
```

```
  "super_provider": true,
  "pwd_hash": "heslo",
  "type": {
    "mlonline_email": {
      "description" : "All the emails in the mailing lists.",
      "sys_type": "mailing_list_message",
      "sys_content_content–type" : "text/plain",
      "search_all_excluded" : "false",
      "index": {
        "name": "data_mlonline_email",
        "type": "mlonline_emailpost"
      }
    }
  }
}
```

Nastavenie detailov indexovania prebieha pomocou konfiguračných súborov. Vo verzii 0.9.0, ktorá bola pri tvorbe práce využívaná, sa konfiguračné údaje nachádzajú v priečinkoch *configuration/indexes*, *configuration/mappings* a *data/config*. Tie presne udávajú formu, uloženie a prácu s dátami, ktoré budú zasielané do inštancie projektu *Searchisko*. Týmito súbormi je tak napríklad možné určiť štruktúru uchovávaných dát, spôsob indexovania (napr. využitie špeciálnych nástrojov nazvaných *analyzers*[1] poskytovaných v nástroji *Elasticsearch*) alebo prioritu jednotlivých atribútov, ktorá je využitá pri hľadaní. V prípade úspešného vyplnenia údajov je následne možné pod registrovaným menom a heslom zasielať objekty typu *JSON* do inštancie [23].

V čase konfigurovania projekt ešte nebol v produkčnej fáze, čím jeho využitie v aplikácii napomohlo k otestovaniu dokumentácie projektu. Z rovnakého dôvodu je možné, že uvedené cesty ku konfiguračným súborom sú už zastaralé.

## 3.4 Výber GUI nástroja

Kedže súčasťou práce je aj vytvorenie internetovej aplikácie, je vhodné využiť nástroj (*framework*), ktorý uľahčuje tvorbu grafického rozhrania. V jazyku Java existuje veľké množstvo takýchto nástrojov, avšak je vhodné vyberať z najviac používaných, ktorými sú: SpringMVC, Wicket, GWT, JSF. Tie sú v krátkosti opísané v nasledujúcich podsekciách.

1. *Analyzer* pozostáva z definovania spôsobu spracovania reťazcov spolu s definovaním prípadných výsledných úprav (ako napríklad ukladanie len v malých písmenách)

### 3.4.1 SpringMVC

SpringMVC je nástroj na tvorbu aplikácií orientovaný na požiadavky tvorené protokolom HTTP (angl. *request-based*). Je modulom väčšieho celku nazývaného *Spring framework* a patrí medzi najpoužívanejšie nástroje pre tvorbu internetových aplikácií v programovacom jazyku Java. Skratka MVC v názve označuje architektúru návrhu, ktorá má za cieľ jednoznačné vyznačenie a dostatočné oddelenie troch hlavných častí aplikácií, ktorými sú modely (angl. *models*), pohľady (angl. *views*) a riadiče (angl. *controllers*) [24]

### 3.4.2 Wicket

Je komponentovo riadený, čisto objektovo orientovaný framework, ktorý vznikol v roku 2004 a bol v počiatkoch vyvíjaný 2 ľudmi, ktorých mená sú Jonathan Locke a Miko Matsumura. Od roku 2007 je projekt vyvíjaný organizáciou Apache Software Foundation. Prepojenie java kódu so stránkami *HTML* je dosiahnuté pridaním parametra do tagu v kóde *HTML*. Úprava samotného kódu *HTML* je preto minimálna a grafik nemusí mať hlbšie znalosti o nástroji a dokonca ani o Jave [25].

### 3.4.3 GWT

GWT je množina nástrojov vyvíjaných ako *open source*, ktorá ponúka možnosť tvorby dynamických internetových aplikácií využívajúcich programovací jazyk javascript vo veľkej miere bez potreby ovládať syntax tohto jazyka. GWT pri kompilácii zabezpečuje preklad kódu napísaného v programovacom jazyku Java do optimalizovaného kódu v jazyku javascript. Výhody tvorby aplikácií v GWT sú nasledujúce [26]:

1. tvorba aplikácie je podobná klasickému vývoju v jazyku Java, čo je výhoda v prípade programátorského tímu, ktorý ovláda tento jazyk

2. kompilácia zabezpečuje kompatibilitu naprieč rôznymi verziami javascriptu

3. vykresľovanie stránky prebieha prevažne na strane klienta, server sa tak zbytočne nezaťažuje a vďaka tomu je schopný obslúžiť väčšie množstvo požiadaviek

Nevýhody GWT:

1. obsah je generovaný na strane klienta pomocou využitia javascriptu, preto nastáva problém pri optimalizovaní stránok pre vyhľadávacie stroje (Google, Yahoo Search a pod.). Tento problém sa dá riešiť tvorbou alternatívnych statických stránok pre vyhľadávače, ktoré budú obsahovať potrebné informácie.
2. zvýšená záťaž klienta
3. výsledkom je pomerne veľké množstvo kódu napísaného v jazyku Java

### 3.4.4 JSF

Je jedna zo špecifikácií, ktoré boli opisované v časti 3.1, formalizovaná ako štandard pre tvorbu komponentovo orientovaných grafických rozhraní pre internetové aplikácie [27]. Samotná JSF technológia prešla niekoľkými verziami a väčšími úpravami kvôli kritike v prvotných verziách. V súčasnosti je najnovšia verzia 2.2. Špecifikácia má viacero implementácií, medzi najznámejšie patria *Oracle Mojarra* a *Apache MyFaces*.

Návrh JSF technológie umožňuje svoje rozšírenie ďalšími nástrojmi, ktoré ešte viac uľahčujú tvorbu aplikácií. Medzi takéto nástroje sa radia komponentové knižnice, ktoré uľahčujú prácu a tvorbu, pričom používateľom ponúkajú vlastné komponenty, ktoré je možné upraviť.

Najznámejšie komponentové knižnice pre technológiu JSF sú:

**RichFaces** je projekt vyvíjaný formou *open source*, patrí do komunity *JBoss*. Sústreďuje sa na rozšírenie AJAX podpory v JSF, avšak takisto rozširuje aj iné vlasnosti a komponenty JSF implementácií [28].

**PrimeFaces** je projekt vyvíjaný formou *open source* spoločnosťou, ktorá má názov *PrimeTek*. Rozširuje knižnicu komponent o mnoho ďalších a radí sa medzi najpoužívanejšie nástroje pre tvorbu aplikácií nezáležiac na programovacom jazyku [29].

**ICEFaces** je projekt vyvíjaný formou *open source* spoločnosťou *ICEsoft Technologies Inc.* Do technológie JSF prináša vlastné komponenty [30].

JSF technológie sú pomerne špecifickým nástrojom pre tvorbu aplikácii. Medzi výhody využitia JSF pri tvorbe internetových aplikácií patrí:

1. špecifikácia zaručuje podporu a široké využívanie
2. JSF má mnoho používateľov, je preto jednoduché nájsť potrebné informácie

Nevýhody JSF:

1. je považovaná za zbytočne náročnú technológiu
2. ako technológia orientovaná na server so sebou nesie aj vyššiu záťaž servera (výpočty sa dejú prevažne na serveri)

### 3.4.5 Výber

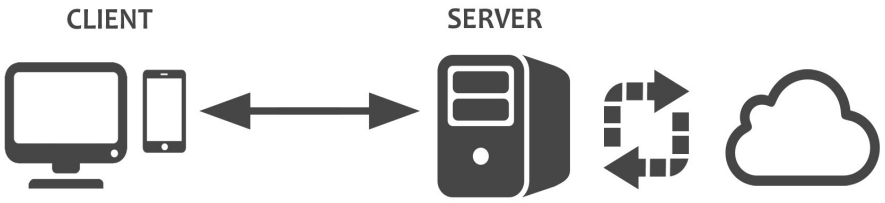
Výber z nástrojov je pomerne ťažká záležitosť, kedže ide o najviac používané spôsoby tvorby internetových aplikácií a každý nástroj má svoje výhody, v ktorých vyniká. Pri tvorbe aplikácie je nutnou podmienkou dostupnosť emailov pomocou internetových vyhľadávačov, a preto použitie GWT je v tomto smere nevýhodou. Na základe nevyužívania iných projektov patriacich do komunity *Spring framework*, a taktiež kvôli veľkej výhode využitia štandardov bol zo zvyšných nástrojov vybraný nástroj JSF s využitím rozširujúcej knižnice Richfaces.

# 4 Mailinglist Online

Po výbere stavebných kameňov, ktorými sú hlavné nástroje na uľahčenie tvorby aplikácií spomenuté v kapitole 3, je ďalším krokom ich vzájomné spojenie a využitie pri implementácii. Táto kapitola opisuje samotnú implementáciu a tvorbu aplikácie. Aplikácia vystupuje pod názvom Mailinglist Online, ktorý v jednoduchosti vyjadruje jej hlavný cieľ. Kvôli lepšej škálovateľnosti bola aplikácia postupne rozdelená do 2 hlavných častí na navzájom komunikujúce komponenty:

**Server** je komponent zastrešujúci ukladanie, exportovanie a správu dát spolu s komunikáciou s inštanciou projektu *Searchisko*.

**Klient** je aplikácia zameraná na zobrazovanie emailov a jednoduchý prístup používateľa k hľadaným informáciám.




Obr. 4.1: Zobrazenie komunikácie jednotlivých komponént

Rozdelením aplikácie na komponenty server a klient je dosiahnutá nezávislosť klienta na konkrétnom spôsobe ukladania emailov. Takéto rozdelenie takisto umožnuje prípadné vytvorenie ďalších klientských aplikácii (napr. mobilnej aplikácie).

Samotný komponent *server* sa ďalej delí na menšie komponenty *import* a *export*. Oddelením časti kódu zodpovedného za vkladanie emailových správ do databázy od zvyšného kódu na serveri je výhodné z hľadiska odľahčenia a jednoduchšej správy najkritickejšej časti, ktorou je práve aplikácia sprístupňujúca správy klientom. Okrem toho oddelenie ponúka aj možnosť vkladania správ z úplne iného servera ako je ten, na ktorom beží komponent *export*.

Pri tvorbe aplikácie je využitý nástroj *Apache Maven*, ktorý uľahčuje správu závislostí, ako aj kompiláciu a spúštanie aplikácie. Projekt je

zdieľaný pomocou internetovej stránky *github*, ktorá je pri zdieľaní projektov typu *open source* bezplatná. Okrem toho stránka ponúka jednoduché rozhranie pre správu verzií, prehľad jednotlivých zmien kódu alebo evidenciu nahlásených problémov a plánovaných vylepšení. Adresa URL projektu je https://github.com/mailinglistonline.

V nasledujúcom texte sú jednotlivé komponenty popísané dôkladnejšie spolu s popisom menších implementačných detailov a opisom technológií, ktoré boli v jednotlivých častiach využité.

## 4.1 Komponent import

Importujúca časť serverového komponentu sa stará o vkladanie emailov a ich ukladanie do databázy. Komponent je schopný prijímať vstup dvoma spôsobmi, ktorými sú:

**súbory formátu mbox** Prečítanie a uloženie emailov uchovávaných vo formáte *MBOX*. Využiteľné pri konfigurovaní novej elektronickej konferencie, čím aplikácia dokáže získať povedomie o všetkých doposiaľ skladovaných emailoch v súboroch. Spracovávanie súborov *MBOX* je pre dosiahnutie lepšieho výkonu paralelizované.

**konzolový vstup** Prísun emailov vo formáte *MIME* pomocou konzolového vstupu. Spôsob je potrebný pre prísun informácií novo prichádzajúcich emailov. Táto možnosť je využiteľná až po vložení súborov typu *MBOX* z archívu.

Pri spracovávaní emailov je využitá knižnica *mstor* pre ukladanie a čítanie internetových správ v textovej podobe. Správy sú následne uložené do databázy *MongoDB*, ktorá bolá opísaná v sekcii 3.2.3. Databáza pritom nemusí byť spustená lokálne, v takom prípade je však nutné upraviť hodnoty v konfiguračnom súbore *database.properties*.

Na prácu s databázou *MongoDB* je využitá oficiálna knižnica pre prístup k datám v jazyku *Java*.

### 4.1.1 Forma uchovávaných dát

Ako už bolo spomenuté, dáta v aplikácii sú ukladané v inštancii databázy *MongoDB*, ktorá uchováva dáta vo formáte *BSON*. Tomu je nutné prispôsobiť formu ich uchovávania. Pre každý nový prichádzajúci

email je v aplikácii vytvorený najmenej jeden nový záznam (v prípade, že má správa v zozname adresátov viaceré elektronické konferencie, je obsah správy uložený viackrát, vždy pod inou elektronickou konferenciou). V prípade rozpoznania zhody správy s už uloženou správou v databáze nie je práve spracovávaná správa uložená. Záznamy sú ukladané v kolekcii *emails*, ktorá je súčasťou databázy s názvom *mailinglists*. Binárne súbory a iné prílohy sú ukladané do špeciálnych kolekcií *fs.files* a *fs.chunks* spôsobom opísaným v sekcii 3.2.5.

Forma ukladania správy bola počas tvorby aplikácie niekoľkokrát menená na základe potreby a využitia dát v klientskej aplikácii. Na ukladanie dát sa využíva spojenie 3 rôznych entít definovaných v balíčku *mailinglistonline.server.export.database.entities*, ktoré sú postupne opísané v nasledujúcich podsekciách.

#### Entita s názvom ContentPart

Využíva sa na uchovávanie informácií o časti správy, ktorá udržiava dátové informácie. Ide tak o uchovávanie tela správy, ktorá môže byť preposlaná v rôznych formách (*HTML*, čistý text a pod.) alebo uchovávanie prílohy pribalenej do správy. Entita uchováva informácie o 3 atribútoch:

**typ** Uchováva informáciu o type dátovej informácie. Príkladmi sú text, binárne dáta, obrázok a pod.

**obsah** Uchovávanie samotnej textovej informácie v type, ktorý je uložený v predchádzajúcom atribúte.

**odkaz** Kedže uchovávaná príloha môže byť príliž veľká a môže tak presiahnuť limit jedného objektu v databáze *MongoDB*, je súbor uložený v špeciálnych kolekciách a v tomto atribúte je uchovávaný len odkaz.

#### Entita s názvom MiniEmail

Postupným vývojom aplikácie sa ukázalo, že pri zobrazovaní jednej správy je vhodné zobraziť istú časť informácie aj o správach, s ktorými je daný email v nejakom vzťahu, čím sa dá správa jednoduchšie zaradiť do kontextu. Taktiež pri prehľadávaní správ nie je potrebné posielať kompletne všetky informácie o danej správe. Z týchto dôvodov bola vytvorená entita *MiniEmail*, ktorá uchováva malú podmnožinu informácií o správe. Entita uchováva nasledujúce informácie:

**ID** Unikátny objekt jednoznačne identifikujúci uloženú správu.

**mailinglist** Elektronická konferencia, do ktorej kontextu je správa uložená.

**messageID** Je identifikátor správy, ktorý je uložený priamo v nej. Mnohokrát je tento reťazec unikátny, ale nie je to tak vždy.

**messageSnipped** Začiatočný reťazec tela správy. Využiteľný v prípade, keď nie je potrebný celý text správy.

**from** Atribút uchovávajúci adresu odosielateľa.

**date** Číslo udávajúce vznik emailovej správy.

**tags** Krátke reťazce presne identifikujúce obsah správy. Takýto reťazec je anglicky označovaný ako *tag*.

Entita s názvom Email

Hlavná entita uchovávajúca uloženú správu, ktorá je vytvorená pre každý nový prichádzajúci email. Pri ukladaní je využitá funkcionalita databázy *MongoDB*, ktorá umožnuje uchovávať objekty v rámci väčších objektov. Táto entita tak vzniká s využitím entít, ktoré boli opísané vyššie, spolu s pridaním dalších informácií o správe. Entita dedí všetky vlastnosti entity *MiniEmail*, takže obsahuje všetky atribúty, ktoré sú v nej obsiahnuté. Okrem toho obsahuje nasledujúce atribúty:

**root** Objekt uchovávajúci entitu typu *MiniEmail*, ktorá udržuje informácie o správe uloženej v samotnom vrchole reťazca odpovedí.

**inReplyTo** Objekt typu *MiniEmail* udržiavajúci informácie o správe, na ktorú je táto správa odpoveďou.

**replies** Množina objektov typu *MiniEmail*, ktoré uchovávajú podmnožinu dát o všetkých odpovediach na túto správu.

**attachments** Pole objektov typu *ContentPart* uchovávajúce informácie o prílohách.

**mainContent** Pole objektov typu *ContentPart* udržiavajúce dáta o tele samotnej správy.

**emailShardKey** Reťazec udávajúci kľúč využiteľný pri horizontálnom škálovaní, ktoré je bližšie opísané v sekcii 3.2.4. Vznikol spojením reťazca *ID* a reťazca *mailinglist*, aby boli jednotlivé správy z rovnakých elektronických konferencií ukladané v čo najmenšom počte databázových inštancií.

Ukážka jednoduchej uloženej správy

Pri opise jednotlivých atribútov entít boli použité názvy premenných dodržiavajúce konvenciu *CamelCase*, ktorá je využitá v programe. V prípade uchovávaných objektov v databáze *MongoDB* je táto konvencia iná, z čoho vyplýva jemne odlišné pomenovanie atribútov. V nasledujúcom príklade sú skutočné hodnoty atribútov zamenené za kratšie alternatívy z dôvodu l'ahšej čitateľnosti.

**Forma správy uloženej v databáze:**

```
{
  "_id" : ObjectId("11...b"),
  "replies" : [ ],
  "message_id" : "<13819@webmail.messagingengine.com>",
  "date" : NumberLong("1381950596000"),
  "from" : "example@example.com",
  "subject" : "Re: Example subject",
  "main_content" : [
      { "type" : "text/plain",
        "text" : "Just a simple message."
      }
  ],
  "message_snippet" : "Just a simple message.",
  "mailinglist" : "linux-cluster@redhat.com",
  "in_reply_to" :
      { "_id" : "53...22a",
        "mailinglist" : "linux-cluster@redhat.com",
        "message_id" : "<13...589@a.com>",
        "subject" : "Example subject",
        "date" : NumberLong("1381949041000"),
        "from" : "example2@example2.com",
        "message_snippet" : "A simple message number 2",
        "tags" : null
    },
   "thread_root" :
      {  "_id" : "53...22a",
         "mailinglist" : "linux-cluster@redhat.com",
         "message_id" : "<13...589@a.com>",
         "subject" : "Example subject",
         "date" : NumberLong("1381949041000"),
         "from" : "example2@example2.com",
         "message_snippet" : "Simple message number 2",
         "tags" : null
      },
    "email_shard_key" : "linux-cluster@redhat.com11...b"
}
```

## 4.2 Komponent export

Komponent využíva podobné nástroje ako komponent import, kvôli čomu je ich spoločná množina závislostí nastavená v rodičovskom priečinku pomocou nástroja *Maven*.

Exportujúca časť serverového komponentu ponúka rozhranie *REST* implementované s využitím projektu *RESTEasy*, ktorý taktiež patrí do komunity *JBoss*. Vďaka rozhraniu je tak vo výsledku možné s využitím protokolu *HTTP* získať informácie o hľadanom emaily alebo podporovaných elektronických konferenciách zaslaním požiadavky na adresu *URL*, ktorá upresňuje hodnoty požadovaných správ. Následne sú opísané jednotlivé metódy detailnejšie. Pre lepšiu čitateľnosť tabuľky neobsahujú opis parametrov v adresách URL.

**Metódy typu GET v rozhraní REST:**

| URL adresa | Odpoveď |
|---|---|
| /emails/all | Množina všetkých správ |
| /emails/email/{id} | Email s reťazcom ID rovným hodnote predanej na miesto id v adrese *URL* |
| /emails?parameters | Emaily, ktoré obsahujú všetky parametre predané v rámci adresy URL. Je možne špecifikovať odosielateľa, elektronickú konferenciu, značky, počet výsledných správ a argumenty udávajúce usporiadanie výsledku |
| /emails/{mailinglist}/roots/all | Všetky emaily v danej elektronickej konferencii dosadenej za mailinglist v adrese, ktoré nie sú odpoveďou na žiadnu inú správu. |
| /emails/{mailinglist}/roots?parameters | Emaily v danej elektronickej konferencii dosadenej za mailinglist v adrese, ktoré sa nachádzajú v intervale čísel špecifikovaných pomocou atribútov v adrese URL. |
| /mailinglists/all | Zoznam všetkých spravovaných elektronických konferencií. |

| | |
|---|---|
| /emails/search/content? parameter | Správy vrátené vyhľadávaním reťazca, ktorý je predaný v parametri. Hľadanie prebieha s využitím projektu *Searchisko*. |

**Metódy typu POST v rozhraní REST:**

| URL adresa | Odpoveď |
|---|---|
| /emails/email/tag?parameters | Pridanie hodnoty parametru *tag* správe s hodnotou *ID* rovnej predanému parametru v adrese *URL*. |

## 4.3 Klientský komponent

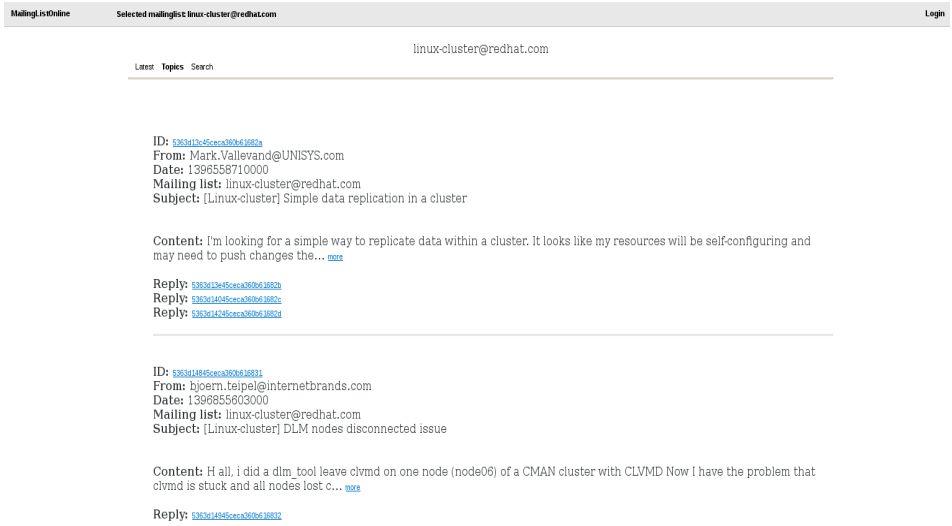
Klientský komponent je úplne oddelený od serverového, komunikácia medzi nimi funguje s využitím rozhrania *REST*. Hlavným cieľom tejto internetovej aplikácie je vytvorenie používateľsky prívetivého rozhrania pre prechádzanie a hľadanie obsahu v archívoch elektronických konferencií.

Aplikácia je tvorená s využitím niekoľkých nástrojov:

1. grafické rozhranie je tvorené s využitím nástroja *JSF*, ktorý je opísaný v sekcii 3.4.4
2. uloženie informácií o registrovaných používateľoch je implementované využitím databázy *MongoDB*
3. komunikácia s využitím rozhrania *REST* je dosiahnutá pomocou nástroja *RESTEasy*

Tvorba rozhrania bola inšpirovaná stránkou *www.reddit.com*, v ktorej sa používateľ stránky stále nachádza v jednej z množstva kategórií, ktorej obsah ho aktuálne zaujíma. V prípade aplikácie ide o zoznam elektronických konferencií, z ktorých používateľ sa nachádza vždy v časti, ktorá opisuje jednu z nich. V časti opisujúcej vybranú elektronickú konferenciu

sú používateľovi ponúkané informácie o najnovších emailoch, zoznam všetkých vlákien alebo vyhľadávanie v danej konferencii. Podpora vyhľadávania nad všetkými konferenciami je implementovaná pomocou špeciálnej konferencie, ktorá nesie názov *ALL*. Používateľovi je taktiež poskytnutá možnosť zobrazenia detailnejších informácií o jednotlivých emailoch. V rámci zobrazenia je možné k danej správe pridávať značky, ktoré môžu byť následne využité pri hľadaní. Pri výbere konferencie




Obr. 4.2: Mailinglist Online, prezeranie emailov

alebo pri prezeraní konkrétneho emailu je informácia o aktuálnej pozícii na stránke uložená v adrese URL. Táto funkcionalita je dosiahnutá pomocou nástroja *PrettyFaces*, ktorý je *open source*[1]. Celkovo sa tak používateľ môže pomocou aktuálnej adresy URL nachádzať v nasledujúcich kontextoch:

1. Počas práce s knižnicou *PrettyFaces* som narazil na závažnú chybu, ktorú som ihneď oznámil a vyžiadala si bezprostrednú opravu s následným vypustením novej verzie. Viac informácií na: http://ocpsoft.org/support/topic/prettyfaces-with-container-based-security

| URL adresa | Kontext |
|---|---|
| /{mailinglist} | Stránka zobrazujúca informácie o danej elektronickej konferencii, ktorej názov sa nachádza v adrese namiesto parametra *mailinglist*. |
| /{mailinglist}/{id} | Stránka zobrazujúca informácie o správe s daným identifikačným reťazcom dosadeným za parameter *id*, ktorá je uložená v konferencii dosadenej na miesto parametra *mailinglist*. Na stránke má používateľ taktiež možnosť pridávať značky k danej správe. |

# 5 Nasadenie a pustenie aplikácie

Pre nasadenie aplikácie je potrebné nasadiť klientskú aj serverovú časť. Pri nasadzovaní serverového komponentu, ktorá má na starosti uchovávanie a sprístupňovanie správ, sa predpokladá nasadenie do serverovne (angl. *cloud*) s viacerými dostupnými výpočetnými a databázovými jednotkami.

## 5.1 Pustenie serverového komponentu import

Komponent import má rôzne metódy pustenia na základe rozdielnych vstupov opísaných v sekcii 4.1:

**súbory formátu mbox** Na pustenie kódu pracujúceho so súbormi typu mbox je nutné zavolať metódu *main* triedy *MboxImporter* s argumentom udávajúcim cestu k súboru alebo priečinku, ktorý obsahuje súbory typu mbox. V prípade zadaného priečinku sa všetky podpriečinky prezerajú automaticky rekurzívne. Na uľahčenie volania je vytvorený skript *import.sh* uložený v koreňovom priečinku, ktorý očakáva na vstupe cestu k súboru alebo priečinku. Komponent vynecháva všetky súbory v priečinkoch, ktoré nemajú príponu *.txt* alebo *.mbox*.

**správa cez konzolový vstup** V prípade potreby importovania správ cez konzolový vstup je nutné zavolať metódu *main* triedy *MessageReceiver*. Metóda má bezparametrickú variantu, ktorá načíta údaje z konfiguračného súboru, a variantu, ktorej sa na vstupe dajú predať konfiguračné informácie v poradí:

1. URL adresa, na ktorej je nasadená inštancia databázy MongoDB
2. názov databázy, do ktorej plánujeme vkladať dáta
3. port databázy
4. názov kolekcie uchovávajúcej dáta
5. prihlasovacie meno do databázy
6. heslo do databázy
7. hodnota *true* alebo *false*, ktorá určuje, či sa správy majú ukladať aj do databázy alebo nie

Takéto priame vkladanie údajov o databáze sa však nepredpokladá a je implementované len z testovacích dôvodov. Pre zmenu štandardných konfiguračných parametrov databázy je potrebné zmeniť údaje v súbore *database.properties*.

Na konfiguráciu umiestnenia inštancie projektu Searchisko slúži súbor *searchisko.properties*, ktorý uchováva informácie o url adrese nasadenej inštancie a prihlasovacích údajov poskytovateľa, ktorého registrácia bola opísaná v sekcii 3.3.3.

Poslednými konfiguračnými údajmi sú údaje o podporovaných elektronických konferenciách v súbore *mailinglists.properties*. Správy, ktorých žiadny z adresátov sa nenachádza v tomto súbore, nie sú uložené do databázy a informácia o nenájdení je presmerovaná na chybový výstup.

Ako bolo už spomenuté, komponent je schopný vkládať dáta aj do vzdialenej inštancie databázy. V prípade posielania dát do platformy *Openshift* je vhodné využiť nástroj s názvom *rhc*, ktorý je voľne stiahnuteľný. Následne príkazom *rhc port-forward*[1] sú všetky zápisy do lokálnej databázy presmerované do databázy v špecifikovanom projekte.

## 5.2 Nasadenie serverového komponentu export

Komponent export je aplikácia kompilovaná do súboru *.war*, preto jej spustením je jej nasadenie na aplikačný server. Správne nasadenie sa dá overiť cez internetový prehliadač. Na hlavnej stránke adresy, na ktorej je komponent nasadený, sa pri správnom nasadení aplikácie ukáže informácia o behu.

Aplikácia využíva rovnaké konfiguračné informácie ako komponent import. Konfiguračné súbory sú bližšie opísané v sekcii vyššie:

**searchisko.properties** Súbor obsahujúci konfiguračné údaje o inštancii Searchisko.

**mailinglists.properties** Súbor obsahujúci všetky akceptované elektronické konferencie.

**database.properties** Súbor obsahujúci informácie o databáze, ktorá ukladá všetky správy aplikácie.

1. Príkaz *rhc port-forward* slúži na presmerovanie adries smerujúcich na adresu *localhost* priamo na inštanciu aplikácie bežiacej v platforme *Openshift* [31].

## 5.3 Klientská aplikácia

Klientská aplikácia je taktiež kompilovaná do súboru *.war*, preto je pre spustenie potrebné jej nasadenie na aplikačný server. Pri správnom nasadení je pomocou prehliadača na adrese aplikácie viditeľná stránka, výzorom podobná obrázku 4.2.

Aplikácia pri zobrazovaní emailov komunikuje so serverom, preto musí byť exportujúca časť servera nasadená. Adresa serverového komponentu export je nastaviteľná pomocou súboru *server.properties*.

Ako bolo spomenuté v sekcii 4.3, klientská časť pre uchovávanie informácií o registrovaných používateľoch využíva databázu *MongoDB*. Parametre pre pripojenie databázy sú konfigurovateľné cez súbor *userDatabase.properties*.

Na zabezpečenie služby aplikácia využíva služby ponúkané aplikačným serverom, ktoré sú špecifikované v konfiguračnom súbore *web.xml*. Na autentifikáciu používateľa je vytvorená špeciálna trieda *MongoDbLoginModule*, ktorá ju implementuje s využitím databázy MongoDB, pretože štandardne táto možnosť implementovaná nie je. Na umožnenie fungovania tohto modulu je potrebné ho v aplikačnom serveri zaregistrovať pridaním do konfiguračného súboru *standalone.xml* časť konfigurácie *XML*:

**Konfigurácia autentifikácie pomocou MongoDB v serveri Wildfly:**

```
<security-domain name="mongo_auth" cache-type="default">
    <authentication>
     <login-module code=
"com.redhat.mailinglistOnline.security.MongoDbLoginModule"
      flag="required">
        <module-option name="hashAlgorithm" value="SHA-256"/>
      </login-module>
    </authentication>
</security-domain>
```

# 6 Problémy a ich riešenia

Pri tvorbe aplikácie som narazil na mnohé problémy, ktoré ovplyvnili tvorenú aplikáciu. Častokrát išlo o chyby v knižniciach, neočakávané události alebo náročnosť implementovania. Problémom, ktoré do značnej miery ovplyvnili vývoj aplikácie, je venovaná táto kapitola.

Chybný preklad objektov do formátu JSON

Preklad objektov do formátu JSON v aplikácii je zabezpečený pomocou knižnice *Jackson* [32], ukladanie dát do databázy je implementované pomocou oficiálneho ovládača pre databázu *MongoDB*, ktorý sa volá *mongo-java-driver* [33]. Knižnica *Jackson* však pri preklade objektov, ktoré sú využívané ovládačom, pracuje iným spôsobom, pri ktorom nie je možné preklad konfigurovať, a tak programátor nemá žiadnu možnosť preklad ovplyvniť[1]. Preto museli byť hodnoty kľúčov vo formáte JSON zhodné s hodnotami kľúčov uloženými v databáze. Okrem toho musí byť hodnota atribútu *ID* pred každým posielaním pomocou rozhrania REST spracovaná metódou, ktorá ju upraví do lepšie čitateľnej podoby. Uložená hodnota v databáze sa však nemení.

Zhoda generovania názvu kľúča záznamu

Projekt *Searchisko* a databáza *MongoDB* majú rovnako nazvaný atribút *ID*, ktorý uchováva unikátnu hodnotu pre každý záznam, pod ktorou je spravovaný. Problém nastáva pri snahe uložiť správu, ktorá má už daný atribút vytvorený, do inštancie projektu *Searchisko*, pretože atribút nespĺňa jeho formát. Riešenie problému je taktiež založené na menení hodnoty atribútu na formát inštancie *Searchisko* v čase posielania správy pomocou rozhrania REST do projektu.

Openshift neponúka škálovanie databázy

Všetky časti aplikácie sú v čase písania práce nasadené v platforme *Openshift*, ktorá však neponúka horizontálne škálovanie databázy *MongoDB* [34], ktoré je bližšie opísané v sekcii 3.2.4. Táto funkcionalita je

1. Chybu som nahlásil najprv na oficiálnom fóre projektu, neskôr bola nahlásena aj na stránke projektu. Viac na https://github.com/FasterXML/jackson-databind/issues/452

vyžadovaná mnohými zákazníkmi platformy, preto je pravdepodobné, že bude v priebehu času implementovaná. V opačnom prípade je možné využiť inú službu typu *PAAS*[1] alebo delegovať správu databázy na službu typu DAAS[2], ktorou je napríklad projekt s názvom *Mongolab* [36].

### Prepojenie databázy s objektami v aplikácii

Pri práci s databázou bol z dostupných nástrojov vybraný oficiálny ovládač, pretože poskytuje možnosť detailnej správy prístupu k databáze. V rámci nástroja však v čase tvorby práce nebolo podporované jednoduché prepojenie databázy s kolekciami objektov. Táto funkcionalita už bola dávnejšie vyžiadaná na stránke podpory projektu pod označením *JAVA-254*[3]. Riešením problému do vydania novej verzie je postupné prepájanie jednotlivých prvkov kolekcií s ovládačom. V prípade veľkých kolekcií je takéto využitie veľmi zdĺhavé. Riešenie problému je uplatnené v triede *DbClient*.

### Vyvíjanie komponentov export a import oddelene

Komponenty *export* a *import* boli od začiatku implementácie vyvíjané oddelene z dôvodu možnosti pustenia na rôznych inštanciách. Komponenty však pracujú s rovnakými entitami, čo malo za následok veľkú časť duplicitného kódu, ktorý viedol k častým opravám a náročnej správe. Z tohto dôvodu bol spoločný kód presunutý do komponenty *export*, vytvorená závislosť časti *import* na časti *export* a obe komponenty spojené do väčšieho, spoločného komponentu *server*.

### Zahltenie tvorbou *session* aplikačným serverom Wildfly

V rámci nasadenia aplikácie do platformy *Openshift* som narazil na problém vysokého počtu tvorby tzv. *session* v škálovanom nasadení servera

1. PAAS (angl. *Platform as a Service*) je jedna z kategórií služieb ponúkaných v rámci servisu, ktorý sa nazýva *cloud computing service*. V tomto prípade poskytovateľ ponúka celú platformu (sieť, databázy, výpočetné stroje a pod.) potrebnú pre nasadenie aplikácie [35].
2. DAAS (angl. *Data as a Service*) je druhou možnou kategóriou služieb v rámci servisu s názvom *cloud computing service*. Ide o prípad, v ktorom poskytovateľ ponúka iba úložný priestor a databázový systém na ukladanie dát [35].
3. Adresa URL s opisom funkcionality je dostupná na adrese https://jira.mongodb.org/browse/JAVA-254

*Wildfly* bez pripojených používateľov. Pri každej tvorbe *session* boli v aplikácií získavané inicializačné dáta, ktoré uchováva komponent *server*. Takýto prístup spôsobil s vysokou tvorbou záznamov typu *session* zahltenie požiadavkami aj bez pripojených používateľov. Obídením problému je inicializačné dáta získavať až v čase ich potreby. Problém som následne nahlásil na oficiálnom diskusinom fóre produktu[1].

1. Adresa URL nahláseného problému je https://community.jboss.org/message/872266#872266

# 7 Vykonnostné štatistiky

Aplikácia musí byť vhodná na nasadenie v prostredí, v ktorom je denne zaslaných viac ako tisíc správ denne. Popritom musí byť schopná reagovať na množstvo požiadaviek používateľov. Preto je jej dostatočná rýchlosť jedným z najdôležitejších kritérií, ktoré sú na aplikáciu kladené. Rýchlosť aplikácie sa dá zmerať na niekoľkých miestach, táto kapitola má preto za cieľ zhrnúť výsledky jednotlivých meraní. V čase merania bolo dostupné internetové pripojenie s hodnotou sťahovania 8009 kB/s a hodnotou posielania dát 3536 kB/s.

## 7.1 Práca s databázou

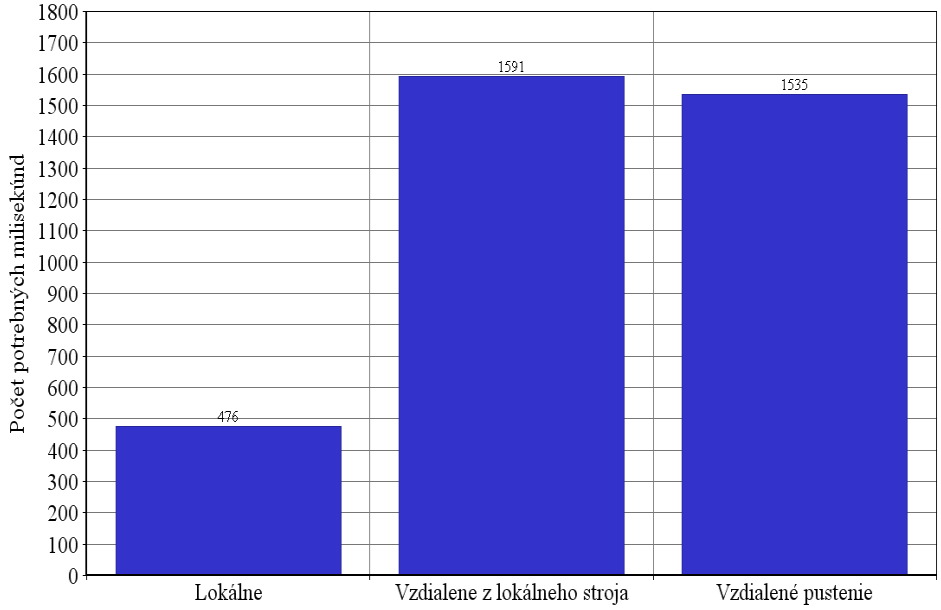
Vkladanie správ do databázy bolo testované z hľadiska práce s databázou pustenou lokálne a databázou pustenou v platforme *Openshift*[1]. So vzdialenou databázou boli vykonané dva druhy testov. Prvý zahŕňal púšťanie komponentu *import* lokálne s pomocou príkazu *rhc port-forward*, ktorý bol opísaný v sekcii 5.1. Druhý pozostával z púšťania komponentu na serveri, na ktorom beží komponent *export*[2]. Celkovo tak bol testovaný čas potrebný na vloženie správ a čas, ktorý je potrebný na inicializovanie spojenia. Testovanie bolo vykonané za pomoci triedy *PerformanceOutputTests*, ktorá vypisuje časy jednotlivých častí na štandardný výstup.

Inicializácia spojenia

Inicializácia spojenia zahrňuje postupné vytvorenie objektov zastupujúcich databázu, kolekciu a zaregistrovanie entít, ktoré budú v rámci inštancie prenášané. Záznamy v testovacej kolekcii boli v rámci inicializácie spojenia taktiež vymazané. Hlavným dôvodom testov je znázorniť dĺžku trvania základných operácií bez samotného prenášania emailov a jej ovplyvnenie v závislosti od prostredia, v ktorom sú testy pustené. Výsledné hodnoty sú založené na čase behu konštruktora testovacej triedy. Čas

1. Príkaz *ping* pustený lokálne s argumentom zodpovedajúcim danej inštancii platformy vrátil v čase testovania hodnotu *120 ms*
2. Meranie spôsobom pustenia modulu import v platforme *Openshift* nie je rovnaké s vkladaním lokálne, pretože v prípade škálovateľného nasadenia je databáza pustená v inom kontexte ako samotný aplikačný server, na ktorom beží aplikácia

potrebný pre inicializovanie objektu pracujúceho lokálne s lokálnou inštanciou databázy je *476 ms*. Inicializácia pri pustení komponentu lokálne presmerovaného na vzdialenú inštanciu trvala *1591ms*, so spustením z platformy *Openshift* trvala inicializácia *1535 ms*. Výsledné hodnoty sú znázornené grafom 7.1. Takáto inicializácia v prípade nasadenia komponentu *export* neprebieha často, preto má táto hodnota zásadný vplyv jedine v prípade vkladania správ.

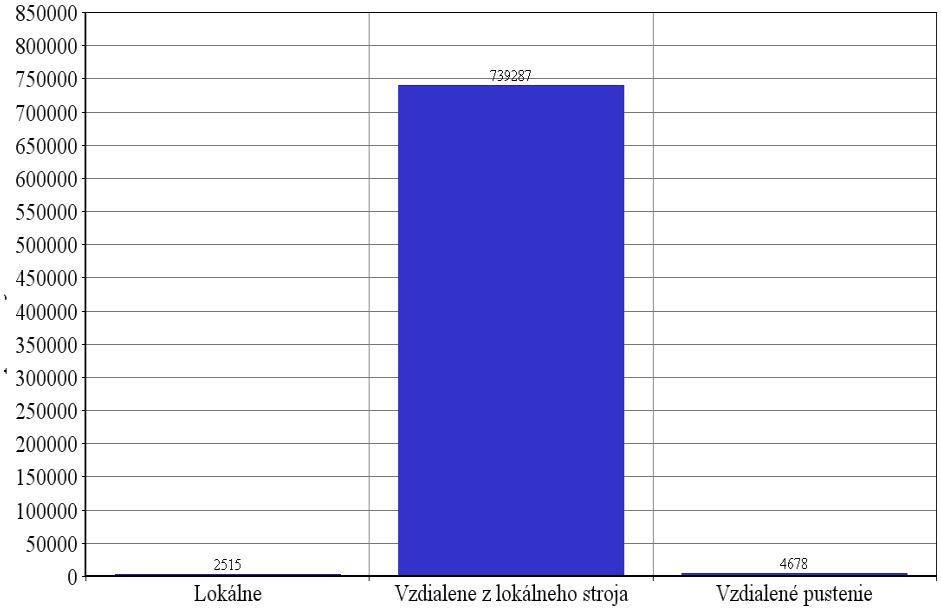



Obr. 7.1: Testovanie času potrebného na inicializáciu testov.

### Vkladanie správ

Testovanie rýchlosti vkladania správ inštancií bolo vykonané s pomocou súboru *test-mails.mbox*, ktorý obsahuje testovacie emaily z konferencie s názvom *linux@lists.linux.sk*. Celkovo súbor obsahuje 13333 riadkov, v ktorých je uložených 62 správ. Súbor zaberá *621.4 kB* na disku. Vykonaný test 10-krát vytvoril objekty zodpovedajúce dátam v súbore a preposlal ich na prislušnú inštanciu, na ktorej následne tieto uložené dáta zmazal. Celkovo tak išlo o vloženie 620 správ s priebežným mazaním tabuliek po skončení práce so súborom. V prípade spustenia testov

lokálne na lokálnej inštancii si táto operácia vyžiadala čas *2515 ms*. Spustenie lokálne s presmerovaním na vzdialenú inštanciu si vyžiadalo čas *739287 ms*, v prípade pustenia z prostredia *Openshift* išlo o čas *4678 ms*. Z merania vyplýva, že v prípade vkladania veľkých súborov je výhodnejšie vkladať správy zo servera, na ktorom je pustená inštancia databázy. Výsledné hodnoty sú znázornené grafom 7.2.



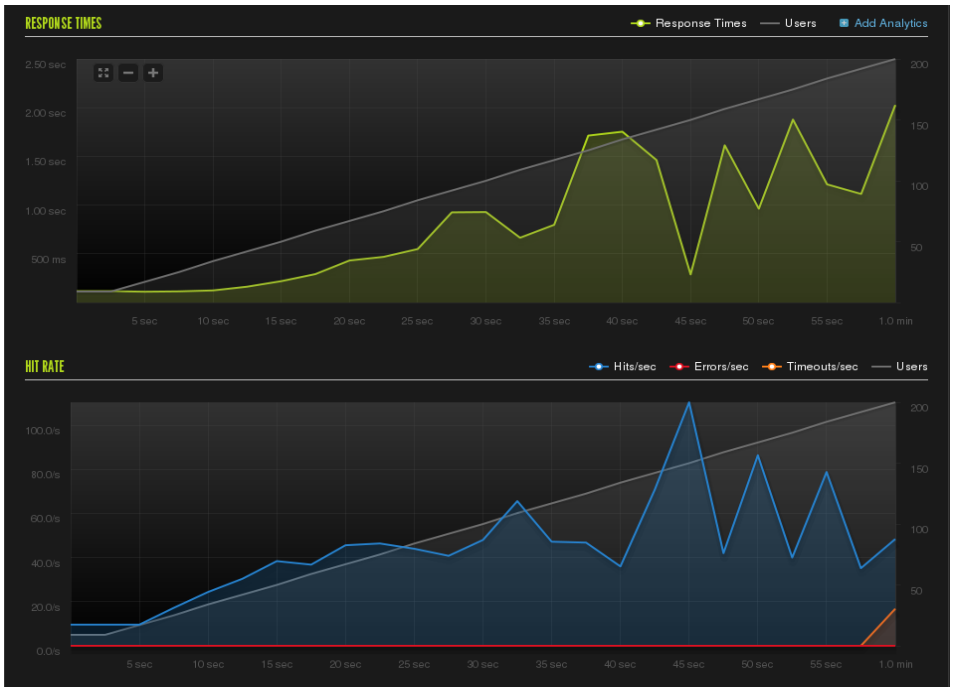

Obr. 7.2: Testovanie času potrebného na vloženie 620 správ.

## 7.2 Počet možných používateľov

Posielaním požiadaviek na aplikáciu je možné odhadnúť záťaž v prípade väčšieho množstva aktívnych používateľov. Takéto testovanie sa nazýva *load testing*. Na testovanie záťaže bola využitá internetová služba *www.blitz.io*, ktorou je možné simulovať veľké množstvo aktívnych používateľov [37].

Záťaž testovaná na úvodnej stránke

V rámci testovania bolo simulovaných 200 aktívnych používateľov, s výslednou priemernou odozvou aplikácie *831 ms*. Testovanie trvalo jednu minútu. Výsledná priemerná hodnota je ovplyvnená vysokým zaťažením servera v konečnej časti testu a taktiež pravdepodobným obsadzovaním novej výpočetnej jednotky v platforme *Openshift*, ktorú vyvolalo celkové zaťaženie. Testovanie prebiehalo na úvodnej stránke aplikácie *client*, čo zahŕňa vygenerovanie stránky na serveri[1] spolu s požiadavkou na komponent *server* na všetky spravované elektronické konferencie. Výsledné hodnoty sú vypísané na obrázku 7.3.



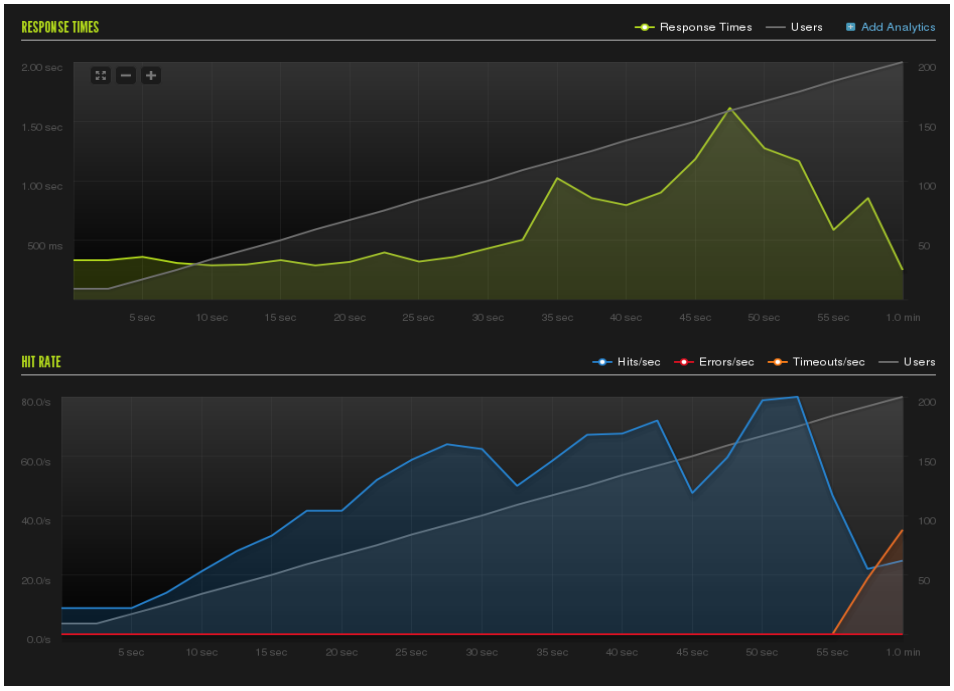

Obr. 7.3: Testovanie výkonu stránky. Zdroj: https://www.blitz.io

Záťaž testovaná na databáze

Databáza je jedna z najdôležitejších častí aplikácie, pretože rýchlosť odpovedí má dopad na rýchlosť celého systému a jej prípadné zahltenie

1. JSF je serverovo orientovaný nástroj, čo znamená, že generovanie stránky z veľkej časti prebieha na serveri

znefunkční všetky 3 komponenty. Testovanie tejto časti bolo vytvorené formou požiadaviek s využitím rozhrania REST. Požiadavky simulovali postupný nárast 200 používateľov v priebehu 1 minúty pristupujúcich na adresu http://server-mlonline.rhcloud.com/rest/emails?mailinglist=linux-cluster@redhat.com&count=10&descending=date. Adresa odpovedá na požiadavku zoznamom desiatich najnovších emailov v elektronickej konferencii *linux-cluster*. Táto požiadavka je pomerne častá z komponentu *client*. Komponent server bol v čase testov spravovaný 5 výpočetnými jednotkami a 1 inštanciou spravujúcou databázu[1]. V špecifikovanej konferencii bolo v čase púštania testov spravovaných 1325 správ. Z výsledku testov záťaže vyplýva, že databáza je schopná uniesť 130 súčasných používateľov, pri ktorých sa spoločne odhaduje 80 klikov za sekundu. Výsledný graf je zobrazený obrázkom 7.4.




Obr. 7.4: Testovanie výkonu databázy. Zdroj: https://www.blitz.io

1. V čase písania práce nebolo na platforme *Openshift* možné škálovanie databázy *MongoDB*

# 8 Záver

Práca sa na začiatku zaoberá analýzou a opisom problému uchovávania veľkého počtu emailov v elektronických konferenciách. V rámci nasledujúcich sekcií sú uvedené rôzne nástroje využiteľné pri riešení istej podčasti problému s opisom ich prípadných silných a slabých stránok. V každej zo sekcii je argumentovaný výber jedného nástroja z nich. Následne práca opisuje implementačné detaily, formát uchovávaných dát a spôsob nasadenia aplikácie.

Cieľom práce bolo vytvoriť aplikáciu schopnú spravovať elektronické konferencie. Hlavnými požiadavkami boli robustnosť návrhu, možnosť pridávania značiek a ich zohľadnenie pri vyhľadávaní v správach. Výsledná aplikácia, ktorá nesie názov *Mailinglist Online*, tieto požiadavky splňuje a obsahuje naviac možnosť prezerania konferencií pomocou preddefinovaných požiadaviek na zobrazenie posledných pridaných správ a koreňových správ jednotlivých vlákien v rámci elektronickej konferencie. Vývoj aplikácie je spravovaný a zdieľaný na stránke https://github.com/MailinglistOnline. Očakáva sa, že sa na aplikácii bude naďalej pracovať a bude pridaná potrebná funkcionalita, ktorá zaručí väčšiu prívetivosť používania. Plánované vylepšenia a opravy sú opísane v rámci stránky vývoja aplikácie v sekcii *Issues* jednotlivých komponentov.

Počas tvorby aplikácie som narazil na niektoré chyby v knižniciach, ktoré som nahlásil a z ktorých niektoré boli už v čase písania práce opravené. Ich zoznam je uvedený medzi prílohami práce. Tvorba aplikácie bola prezentovaná interne vo firme *Red Hat*, ako aj na medzinárodnej konferencii *DevConf* v Brne.

Jednotlivé časti aplikácie sú nasadené pomocou platformy *Openshift* na serveroch *Wildfly 8* a *JBoss EAP 6*. Pri behu aplikácií je využité škálovanie výpočetných jednotiek, ich počet sa teda automaticky navyšuje v prípade potreby. Komponenty sú nasadené na nasledujúcich adresách:

**server** http://server-mlonline.rhcloud.com/

**client** http://client-mlonline.rhcloud.com/

**searchisko** http://searchisko-mlonline.rhcloud.com/

# Literatúra


[1] *Ian Peter: The history of email* [online]. c2004 [cit. 2014-21-4]. Dostupný z WWW: <http://www.nethistory.info/History%20of%20the%20Internet/email.html>.

[2] *Lars Magne Ingebrigtsen : Gmane* [online]. c2004 [cit. 2014-20-5]. Dostupný z WWW: <http://gmane.org/>.

[3] *Free Software Foundation, Inc.: Mailman, the GNU Mailing List Manager* [online]. c2014 [cit. 2014-20-5]. Dostupný z WWW: <https://www.gnu.org/software/mailman/>.

[4] *The Apache Software Foundation: Apache Tomcat* [online]. c2014 [cit. 2014-21-4]. Dostupný z WWW: <http://tomcat.apache.org/>.

[5] *Oracle: Java Servlet 3.1 Specification* [PDF,online]. c2013 [cit. 2014-21-4]. Dostupný z WWW: <https://jcp.org/en/jsr/detail?id=340>.

[6] *Plumbr: Most popular application servers* [online]. c2013 [cit. 2014-21-4]. Dostupný z WWW: <https://plumbr.eu/blog/most-popular-application-servers>.

[7] *Oracle: GlassFish* [online]. c2013 [cit. 2014-21-4]. Dostupný z WWW: <https://glassfish.java.net/>.

[8] *RedHat: WildFly* [online]. c2013 [cit. 2012-22-4]. Dostupný z WWW: <http://www.wildfly.org/>.

[9] *C.J. Date: Database Design and Relational Theory*. 1. vyd. USA : O'Reilly Media, 2012. 278 s. ISBN 978-1-449-32801-6.

[10] *Soumendra Mohanty,Madhu Jagadeesh,Harsha Srivatsa: Big Data Imperatives: Enterprise 'Big Data' Warehouse, 'BI' Implementations and Analytics (The Expert's Voice)* . 1. vyd. New York : Apress, 2013. 320 s. ISBN 978-1-4302-4873-6.

[11] *Shashank Tiwari: Professional NoSQL* . 1. vyd. Indianapolis : Wiley, 2011. 384 s. ISBN 978-0-470-94224-6.

[12] *Pethuru Raj: Cloud Enterprise Architecture*. 1. vyd. USA : CRC Press, 2012. 528 s. ISBN 978-1-4665-0232-1.

[13] *MongoDB, Inc.: http://www.mongodb.com/leading-nosql-database* [online]. c2014 [cit. 2014-10-2]. Dostupný z WWW: <http://docs.mongodb.org/ecosystem/drivers/>.

[14] *MongoDB, Inc.: MongoDB Drivers and Client Libraries* [online]. c2014 [cit. 2014-10-2]. Dostupný z WWW: <http://docs.mongodb.org/ecosystem/drivers/>.

[15] *MongoDB, Inc.: Agile and Scalable* [online]. c2014 [cit. 2014-10-2]. Dostupný z WWW: <https://www.mongodb.org/>.

[16] *MongoDB, Inc.: Replication* [online]. c2014 [cit. 2014-10-2]. Dostupný z WWW: <http://docs.mongodb.org/manual/replication/>.

[17] *MongoDB, Inc.: Sharding* [online]. c2014 [cit. 2014-10-2]. Dostupný z WWW: <http://docs.mongodb.org/manual/sharding/>.

[18] *MongoDB, Inc.: GridFS* [online]. c2014 [cit. 2014-10-2]. Dostupný z WWW: <http://docs.mongodb.org/manual/core/gridfs/>.

[19] *MongoDB, Inc.: Text Indexes* [online]. c2014 [cit. 2014-10-2]. Dostupný z WWW: <http://docs.mongodb.org/manual/core/index-text/>.

[20] *The Apache Software Foundation: Apache Solr* [online]. c2012 [cit. 2014-15-4]. Dostupný z WWW: <https://lucene.apache.org/solr/>.

[21] *Elasticsearch: Elasticsearch* [online]. c2014 [cit. 2014-15-4]. Dostupný z WWW: <http://www.elasticsearch.org/>.

[22] *Red Hat: Searchisko* [online]. [cit. 2014-15-4]. Dostupný z WWW: <https://github.com/searchisko/searchisko>.

[23] *Red Hat: Spring Framework Reference Documentation* [online]. [cit. 2014-15-4]. Dostupný z WWW: <https://github.com/searchisko/searchisko/tree/master/configuration>.

[24] *Pivotal Software: Searchisko* [online]. c2014 [cit. 2014-15-4]. Dostupný z WWW: <http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html>.

[25] *The Apache Software Foundation: Apache Wicket* [online]. c2012 [cit. 2014-15-4]. Dostupný z WWW: <http://wicket.apache.org/>.

[26] *The GWT Team: GWT* [online]. [cit. 2014-15-4]. Dostupný z WWW: <http://www.gwtproject.org/>.

[27] *Oracle: JavaServer Faces Technology* [online]. [cit. 2014-15-4]. c2014 Dostupný z WWW: <http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>.

[28] *Red Hat: RichFaces* [online]. [cit. 2014-15-4]. Dostupný z WWW: <http://www.jboss.org/richfaces>.

[29] *PrimeTek: PrimeFaces* [online]. c2014 [cit. 2014-15-4]. Dostupný z WWW: <http://primefaces.org/>.

[30] *ICEsoft Technologies Inc.: ICEfaces Overview* [online]. c2014 [cit. 2014-15-4]. Dostupný z WWW: <http://www.icesoft.org/java/projects/ICEfaces/overview.jsf>.

[31] *Grant Shipley: Getting Started with Port Forwarding on OpenShift* [online]. c2012 [cit. 2014-21-4]. Dostupný z WWW: <https://www.openshift.com/blogs/getting-started-with-port-forwarding-on-openshift>.

[32] *FasterXML, LLC: Jackson JSON Processor Wiki* [online]. c2012 [cit. 2014-5-5]. Dostupný z WWW: <https://github.com/FasterXML/jackson>.

[33] *MongoDB, Inc.: Java Language Center* [online]. c2014 [cit. 2014-5-5]. Dostupný z WWW: <http://docs.mongodb.org/ecosystem/drivers/java/>.

[34] *Red Hat, Inc.: Openshift* [online]. c2014 [cit. 2014-1-5]. Dostupný z WWW: <https://www.openshift.com/>.

[35] *Chaowei Yang, Qunying Huang: Spatial Cloud Computing: A Practical Approach*. 1. vyd. Florida : CRC Press, 2014. 357 s. ISBN 978-1-4665-9316-9.

[36] *ObjectLabs Corporation: mongolab* [online]. c2014 [cit. 2014-3-5]. Dostupný z WWW: <https://mongolab.com/welcome/>.

[37] *Spirent: Blitz - Load and Performance Testing from the Cloud* [online]. c2014 [cit. 2014-5-5]. Dostupný z WWW: <https://www.blitz.io>.

# Prílohy

## Zoznam nahlásených problémov

Počas tvorby aplikácie som sa v rámci využívania rozličných technológií niekoľkokrát stretol s nečakaným alebo chybným správaním. Takéto správanie som poväčšinou nahlásil príslušným spôsobom. Zoznam problémov, ktoré som pri tvorbe nahlásil alebo bol súčasťou riešenia, je nasledovný:

| Projekt | Problém | Adresa URL |
|---|---|---|
| Arquillian | Nesprávne generovanie adresy URL nasadenej aplikácie | https://issues.jboss.org/browse/ARQ-1770 |
| Wildfly | Pri škálovaní aplikačný server vytvára priveľké množstvo objektov *session* | https://community.jboss.org/message/872266 |
| PrettyFaces | Knižnica pri nasadení spolu s využívaním autentifikácie pomocou *container-based security* túto autentifikáciu úplne obchádzala | http://ocpsoft.org/support/topic/prettyfaces-with-container-based-security/ |
| MongoDB | Podpora prepojenia zoznamov objektov s ovládačom | https://jira.mongodb.org/browse/JAVA-254 |
| Jackson | Nepoužívanie značiek pri preklade objektov z formátu JSON | https://github.com/FasterXML/jackson-databind/issues/452 |

### Priložený kompresovaný súbor

Priložený kompresovaný súbor pozostáva z priečinkov obsahujúcich zdrojový kód jednotlivých komponent aplikácie:

**server** zdrojový kód obsahujúci komponenty *export* a *import*

**client** zdrojový kód komponentu *client*

**searchisko** zdrojový kód inštancie *searchisko* s upravenými konfiguračnými súbormi