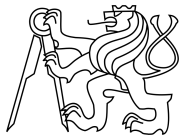
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Autorizační vrstva pro vyhledávač OpenGrok
Student:	Bc. Kryštof Tulinger
Vedoucí:	Mgr. Vladimír Kotal
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Cílem práce je návrh a implementace autorizační vrstvy do vyhledávače OpenGrok (https://github.com/OpenGrok/OpenGrok/). OpenGrok je nástroj pro indexaci zdrojových kódů, jejich zobrazování a vyhledávání v nich skrz webové rozhraní. První částí práce je autorizační vrstva, která musí umožňovat konfiguraci pomocí zásuvných modulů. OpenGrok obsahuje projekty, které se mapují na fyzické adresáře se zdrojovým kódem. Druhou částí práce je navrhnout autorizační vrstvu tak, aby bylo možné provádět autorizaci nad celými skupinami projektů. Třetí částí práce je specifický zásuvný modul pro autorizaci proti internímu LDAP serveru zadavatele.

1) Proveďte analýzu zadavatelských potřeb.
2) Zhodnoťte a porovnejte možná existující řešení.
3) Proveďte návrh vlastního řešení, resp. rozšíření existujícího řešení.
4) Implementujte návrh řešení.
5) Proveďte testy výkonu.
6) Proveďte akceptační testy v produkčním prostředí.
7) Zhodnoťte navržené řešení včetně možných bezpečnostních rizik.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 2. února 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Autorizační vrstva pro vyhledávač OpenGrok

Bc. Kryštof Tulinger

Vedoucí práce: Mgr. Vladimír Kotal

8. května 2017

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 8. května 2017 .....

Odkaz na tuto práci

Tulinger, Kryštof. *Autorizační vrstva pro vyhledávač OpenGrok.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato práce řeší autorizační vrstvu pro vyhledávač OpenGrok. V první části navrhuje, jakým způsobem uspořádat projekty (adresáře se zdrojovým kódem) do skupin pro snazší kontrolu autorizace. Dále pak implementuje autorizační vrstvu založenou na zásuvných modulech, které lze libovolně přidávat do webové aplikace při jejím běhu. Na závěr přichází se zásuvnými moduly pro řešení autorizace OpenGroku oproti vzdálenému LDAP serveru.

Klíčová slova OpenGrok, autorizace, vyhledávač, zdrojový kód

Abstract

This thesis discusses an authorization layer for OpenGrok search engine. In the first part, it deals with how to organize projects (folders with a source code) into groups for a fine-grained authorization control. The next part presents an implementation of an authorization layer based on plugins which can be loaded into the web application during runtime. Lastly, the thesis comes with a pack of plugins to solve the OpenGrok authorization against a remote LDAP server.

Keywords OpenGrok, authorization, search engine, source code

Obsah

Úvod **1**
Autorizační vrstva 1
Struktura 1

1 Popis problému **3**
1.1 Cílová skupina uživatelů 3

2 OpenGrok **5**
2.1 Úvod 5
2.2 Indexer 7
2.3 Webová aplikace 8

3 Abstraktní členění projektů do skupin **13**
3.1 Projekty 13
3.2 Repositáře 14
3.3 Skupiny 15
3.4 Analýza možných řešení 16
3.5 Návrh řešení 17
3.6 Implementace 21
3.7 Testování 27

4 Autorizační framework **29**
4.1 Bezpečnostní model 29
4.2 Autorizační vrstva 30
4.3 Analýza požadavků 30
4.4 Analýza možných řešení 31
4.5 Návrh řešení 33
4.6 Implementace 45
4.7 Testování 55

5 Autorizační plugin **65**
5.1 Úvod 65
5.2 Analýza požadavků 67
5.3 Návrh řešení 69
5.4 Implementace 75
5.5 Testování 78

6 Zhodnocení bezpečnostních rizik **85**
6.1 Timing attack 85
6.2 Chybné autorizační pluginy 85
6.3 Podvržený autorizační plugin 86
6.4 Neúmyslně přesunutý autorizační plugin 86
6.5 Hypertext Transfer Protocol (HTTP) stavové kódy 86
6.6 Detekce útoku hrubou silou 87

Závěr **89**

Literatura **91**

A Seznam použitých zkratek **95**

B Změny v OpenGroku **97**
B.1 Abstraktní členění projektů do skupin 97
B.2 Autorizační framework 98

C Obsah přiloženého CD **101**

Seznam obrázků

2.1 Typická instance OpenGroku 7
2.2 Obsah datového adresáře OpenGroku 8
2.3 Uživatelské rozhraní hlavní stránky OpenGroku 10
2.4 Uživatelské rozhraní při prohlížení souboru v OpenGroku 11

3.1 Obsah zdrojového adresáře OpenGroku 14
3.2 Návrh rozhraní třídy Group 19
3.3 Příklad organizace skupin v paměti 19
3.4 Náhled rozhraní hlavní stránky OpenGroku se skupinami 25
3.5 Náhled rozhraní formuláře pro výběr projektů se skupinami 26

4.1 Příklad využití class loaderů v Tomcatu 34
4.2 Návrh rozhraní pro plugin 37
4.3 Demonstrace cachování v autorizačním frameworku 37
4.4 Architektura Pluggable Authentication Modules (PAM) Frameworku 39
4.5 Návrh rozhraní autorizační entity 41
4.6 Ukázka složení stacku a pluginů 41
4.7 Návrh rozhraní autorizačního frameworku 44
4.8 Testovací autorizační konfigurace 58

5.1 Architektura prostředí zadavatele 66
5.2 Návrh konfigurace autorizace 70
5.3 Návrh abstraktní třídy AbstractLdapProvider 72
5.4 Návrh rozhraní LdapServeru 72
5.5 Návrh třídy AbstractLdapPlugin 73
5.6 Návrh vztahu `LdapFacade` a `LdapServer` 76
5.7 Algoritmus zpracování chyby při požadavku na `LdapServer` 77
5.8 Algoritmus zpracování chyby při požadavku na `LdapFacade` 78

Seznam tabulek

4.1 Sériový test bez autorizace 61
4.2 Sériový test s autorizací 61
4.3 Paralelní test bez autorizace 62
4.4 Paralelní test s autorizací 62
4.5 Sériový test s pozměněnými parametry bez autorizace 63
4.6 Sériový test s pozměněnými parametry s autorizací 63
4.7 Paralelní test s pozměněnými parametry bez autorizace 63
4.8 Paralelní test s pozměněnými parametry s autorizací 63

5.1 Testy odezvy v produkčním prostředí 81
5.2 Testy využití autorizační vrstvy v produkčním prostředí 82
5.3 Krajní případy v produkčním prostředí 83

Úvod

OpenGrok jako internetový vyhledávač umožňuje prohlížet a prohledávat zdrojové kódy. Pro vývojáře softwarového produktu nebo pro programátora hledajícího chybu v produktu je užitečným nástrojem pro zvýšení efektivity práce.

Od řady moderních vývojových prostředí, které umožňují také interaktivně procházet zdrojový kód nebo v něm vyhledávat, se liší dostupností přes HTTP protokol skrz webový prohlížeč. Vývojář díky tomu nemusí mít lokální kopii všech zdrojových kódů, ve kterých by potřeboval vyhledávat, ale stačí mu přistupovat na webovou aplikaci přes prohlížeč.

Také se může stát, že zdrojový kód již dávno není dostupný ke stažení pro vývojáře (například produkt ukončil vývoj a další stažení již není možné) a přístup ke zdrojovému kódu skrz OpenGrok je jedinou možností, jak v kódu vyhledávat a jak ho prohlížet (za předpokladu, že byl kód do OpenGroku přidán, když ho ještě bylo možné stáhnout).

OpenGrok navíc umožňuje uživateli zobrazovat historii změn souborů a adresářů, anotace jednotlivých řádek a rozdíl mezi verzemi souborů pro systémy správy verzí zdrojových kódů (Source code management (SCM)).

Autorizační vrstva

OpenGrok přes webový prohlížeč nabízí všem uživatelům všechny zdrojové kódy, které do něj byly nahrány. To může být v rozporu s vnitřními politikami společností využívající OpenGrok pro svoje produkty a snahou je omezit přístupy k jednotlivým zdrojovým kódům na úrovni OpenGroku pro konkrétní uživatele nebo skupiny uživatelů.

Struktura

Tato práce se zabývá návrhem autorizační vrstvy pro vyhledávač OpenGrok a rozděluje návrh do tří samostatných částí.

1. Návrh a implementace rozčlenění zdrojových kódů v OpenGroku do skupin.
2. Návrh a implementace autorizační vrstvy pro kontrolu oprávnění pro zdrojové kódy nebo skupiny zdrojových kódů.
3. Konkrétní řešení autorizace oproti Lightweight Directory Access Protocol (LDAP) serveru zadavatele.

Kapitola 1

Popis problému

Cílem této práce je navrhnout a implementovat autorizační vrstvu pro vyhledávač OpenGrok. Detailní popis vyhledávače a jeho možnosti budou popsány v následující kapitole, nicméně pro autorizaci je podstatná informace, že OpenGrok umožňuje zobrazovat zdrojové kódy v programovacích jazycích ve webové aplikaci a umožňuje vyhledávání v nich. Nástroj OpenGrok je hojně využíván ve firmě zadavatele na zdrojové kódy různých produktů a pro zadavatele je tedy nezbytná funkcionalita přístupových práv k prohlížení a vyhledávání kódů u jednotlivých produktů či aplikací. Aktuálně OpenGrok žádnou autorizační vrstvu neobsahuje, tedy všichni uživatelé, kteří mají přístup k webové aplikaci vyhledávače, vidí všechny zdrojové kódy, které jsou pod jeho správou.

1.1 Cílová skupina uživatelů

Vyhledávač OpenGrok je navržen zejména pro vývojáře softwarových produktů a pro programátory, kteří opravují chyby v existujících programech. Malou část uživatelů mohou představovat zaměstnanci z části firmy, kteří se starají o komunikaci se zákazníkem ohledně nahlášené chyby, ale i laici, kteří si jen touží prohlédnout zdrojový kód.

Vzhledem k hlavní cílové skupině je navržené rozhraní ve webové aplikaci velmi minimalistické a jednoduché na použití. Často se klade důraz na využití klávesových zkratek před použitím počítačové myši pro výběr další akce.

OpenGrok obsahuje také aplikační část, která slouží k indexaci a správě zdrojových kódů. Pro použití této části je nezbytná základní znalost ovládání počítače pomocí příkazové řádky a všeobecné povědomí uživatele o verzovacích systémech. Z těchto důvodů je tato část oddělená od uživatele webové aplikace, který může libovolně procházet výsledky.

Kapitola 2

OpenGrok

OpenGrok, jak ho popisují sami autoři, je rychlý a užitečný vyhledávač ve zdrojových kódech a nástroj pro spojování souvisejících informací na různých místech (cross reference engine). Jeho použití je cíleno zejména na zdrojové kódy, ale lze prohledávat i obyčejné soubory s textem.

OpenGrok je schopný rozpoznat a zpracovat soubory ve velkém množství programovacích jazyků (např. Java, C, C++, C#, PHP a mnoho dalších) a také vhodně interaguje s verzovacími systémy pro správu zdrojových kódů, jako jsou například Mercurial, Git, Subversion a další. Kromě podpory vyhledávání kódu může uživatel také procházet historii změn souborů, adresářů, vyhledávat v historii podle identifikátoru změny a také zobrazit anotace k jednotlivým řádkám v souboru. To vše může pomoci k pochopení kódu, provedených změn nebo při hledání chyby.

Nástroj OpenGrok je napsán v programovacím jazyce Java [1]. Jeho vývoj probíhá na Githubu[1] ve veřejném repositáři jako otevřený software (open-source software) pod licencí Common Development and Distribution License (CDDL-1.0) (CDDL)[2].

2.1 Úvod

OpenGrok obsahuje dvě oddělené aplikační části:

1. aplikaci generující index[3] (dále jen „Indexer“),
2. webovou aplikaci pro prohlížení a vyhledávání.

Obě části jsou propojené jednotnou konfigurací, která odráží zejména stav aktuálního vygenerovaného indexu.

[1] https://github.com/OpenGrok/OpenGrok

[2] https://opensource.org/licenses/cddl1.php

[3] předzpracované zdrojové soubory umožňující rychlé zobrazování stránek a rychlé vyhledávání

2.1.1 Definice pojmů

Zdrojový adresář Adresář obsahující zdrojové kódy a jiné soubory k indexaci.

Datový adresář Adresář obsahující vygenerovaný index – předzpracovanou verzi souborů ve zdrojovém adresáři pro rychlejší zobrazování a předzpracované záznamy o souborech umožňující rychlé vyhledávání.

Instance OpenGroku Nakonfigurovaná instance se spuštěnou webovou aplikací nad daty ze zdrojového adresáře a datového adresáře.

Správce instance OpenGroku Administrátor s přístupem ke konfiguraci OpenGroku a spouštějící Indexer.

Projekt Projekt označuje přímý podadresář ve zdrojovém adresáři, jméno projektu je shodné s názvem adresáře.

Projekty v OpenGroku slouží ke sloučení více různých zdrojových kódů (např. různých produktů) do jedné instance a jejich použití také nabízí mnohé výhody jako například možnost prohledávání více projektů zároveň.

Hlavní konfigurace Soubor s konfigurací v `/var/opengrok/etc/configuration.xml` viz sekce 2.1.2 a obrázek 2.1.

2.1.2 Konfigurace

Konfigurace je uložena v souboru na souborovém systému ve formátu Extensible markup language (XML) a je obrazem serializovaného objektu z Javy. Její syntaxe není pro nezkušeného uživatele příliš intuitivní, a proto Indexer nabízí možnost, jak nejnutnější nastavení modifikovat.

Kromě přímého souboru nabízí OpenGrok také možnost konfigurovat webovou aplikaci skrze síťové rozhraní, protože webová aplikace poslouchá na konfigurovatelné adrese pro změnu konfigurace. Zaslání nové konfigurace webové aplikaci je vždy poslední krok Indexeru a není tedy nutné restartovat webovou aplikaci, či celý server.

Nejdůležitejšími položkami, které se musejí nakonfigurovat jsou:

- zdrojový adresář,
- datový adresář,
- soubor, kam se vypíše výsledná konfigurace OpenGroku.

Soubor s touto konfigurací budeme v následujícím textu také nazývat hlavní konfigurace.

- `var/opengrok/` ..... Složka obsahující soubory OpenGroku
  - `etc` ..... Složka obsahující konfiguraci
    - `configuration.xml` ..... Hlavní konfigurace viz 2.1.2
  - `log` ..... Informace o běhu Indexeru
  - `data` ..... Datový adresář
    - `index` ..... Složka obsahující index
    - `historycache` ..... SCM historie pro indexované soubory
    - `xref` . Vygenerované Hypertext Markup Language (HTML) stránky zdrojových kódů
  - `src` ..... Složka obsahující zdrojové kódy k indexaci
    - `main.c` ..... Soubor ke zpracování

Obrázek 2.1: Typická instance OpenGroku

2.2 Indexer

Indexer je část OpenGroku, která se spouští z příkazové řádky systému. Hlavní úlohou je vygenerovat nový (nebo inkrementální) index do datového adresáře[4], vygenerovat novou konfiguraci a upozornit webovou aplikaci skrz síťové rozhraní, že došlo ke změně indexu a konfigurace. Typicky vypadá instance OpenGroku na systému (předpokládáme Unixový souborový systém) jako na obrázku 2.1.

Příkladné spuštění Indexeru pro tuto základní instanci by mohlo vypadat takto:

```
$ java -jar opengrok.jar \
    -W /var/opengrok/etc/configuration.xml \
    -s /var/opengrok/src \
    -d /var/opengrok/data
```

Soubor `opengrok.jar` se nachází v distribuovaném balíčku[5] OpenGroku. Protože parametry pro běh z příkazové řádky za řadu let vývoje narostly do velkého počtu, obsahuje OpenGrok jednoduchý skript `OpenGrok`, který zkrátí potřebné argumenty na minimum. V tomto případě by stačilo pouhé:

```
$ ./OpenGrok index
```

Soubor `OpenGrok` se opět nachází v distribuovaném balíčku ve složce `bin`.

2.2.1 Struktura indexu

Struktura datového adresáře vypadá jako na obrázku 2.2.

[4] Adresář s daty indexu podle 2.2

[5] Ke stažení na https://github.com/OpenGrok/OpenGrok/releases

```
data ..... Datový adresář
├── index ..... Složka obsahující index
├── historycache ..... SCM historie pro indexované soubory
└── xref ..... Vygenerované HTML stránky zdrojových kódů
```

Obrázek 2.2: Obsah datového adresáře OpenGroku

2.2.1.1 Adresář index

Pro uchování indexu se v OpenGroku používá knihovna Lucene [2]. Ta nabízí výkonné techniky indexování full-textových souborů a ukládání zajímavých částí z těchto souborů do indexové struktury. Následně umožňuje ve vytvořeném indexu velmi rychle vyhledávat.

2.2.1.2 Adresář historycache

Pro zrychlení odezvy při zobrazování historie pro jednotlivé soubory zpracované OpenGrokem je historie každého souboru předem stažena a uložena do souboru v tomto adresáři. Výsledkem je, že tento adresář mapuje 1:1 historii na soubory obsažené v zdrojovém adresáři[6]. OpenGrok je schopný generovat historii pro daný soubor za běhu webové aplikace, nicméně tento způsob může být pro některé typy repositářů při špatné síťové konektivitě silně zdlouhavý.

Zároveň je každý záznam z historie souboru přidán do celkového indexu s cílem umožnit vyhledávání v jednotlivých změnách.

2.2.1.3 Adresář xref

Každý soubor ve zdrojovém adresáři má v adresáři xref svůj obraz v podobě vygenerované HTML stránky obsahující značky pro zvýraznění syntaxe souboru, odkazů na hledání definicí a symbolů. Toto umožní rychlé procházení zdrojového kódu ve webové aplikaci, aniž by bylo potřeba cokoliv generovat. Zkratka xref pochází z již řečeného cross reference pojmu.

2.3 Webová aplikace

Webová část OpenGroku je také napsána v Javě a využívá převážně technologii Java Server Pages (JSP) pro generování dynamických HTML stránek. Tato technologie překládá soubory obsahující značkovací jazyk na Servlet, jenž je následně zapojen do cyklu zpracování klientova požadavku [3]. Z tohoto důvodu vyžaduje webová aplikace Servlet kontejner, tedy podobu webového serveru, která umí spolupracovat se servlety a přeposílat klientovi dynamicky generovaný obsah.

[6] Adresář se zdrojovými kódy připravenými k indexu podle 2.1

Těchto kontejnerů existuje celá řada, zadavatel využívá Apache Tomcat[7], OpenGrok vyžaduje verzi alespoň 8. Dalším příkladem serveru podporující servlety je GlassFish[8] nebo komerční produkt Oracle WebLogic Server[9].

Webová aplikace při startu vyžaduje sdílenou konfiguraci s Indexerem ze sekce 2.1.2. Zároveň se startem spustí vlákno, které naslouchá pro případnou změnu konfigurace od Indexeru na síťovém Transmission Control Protocol (TCP) portu. Díky tomuto vláknu pak již není nutné při změně indexu či změně konfigurace webovou aplikaci restartovat (nebo restartovat Servlet kontejner), protože se sama umí změně konfigurace přizpůsobit za běhu. Změněnou konfiguraci zasílá po svém běhu Indexer.

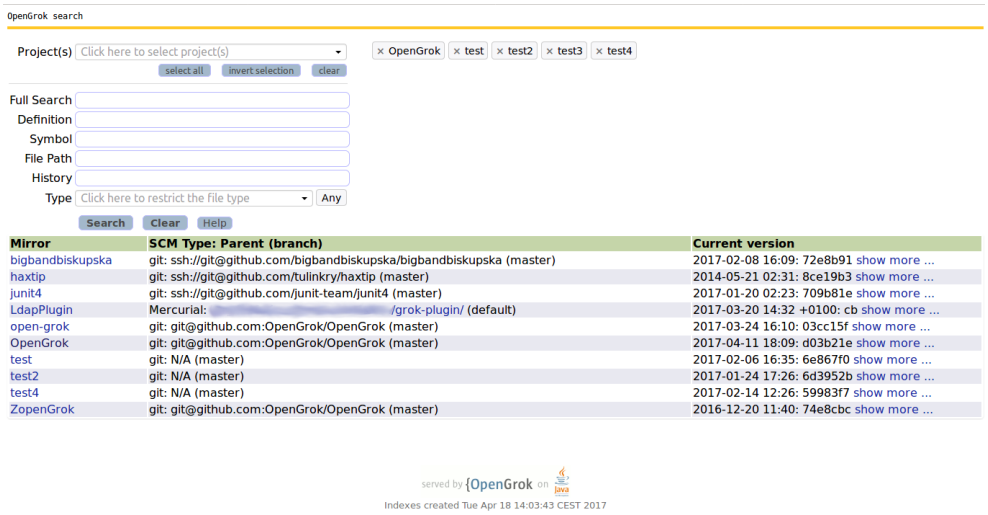
Výše zmíněné informace mají ten důsledek, že přidání nebo odebrání souborů z indexu se může ve webové aplikaci promítnout ihned po doběhnutí Indexeru bez nutnosti restartu. Tento požadavek je nutné brát v úvahu při návrhu abstraktního členění projektů do skupin; o projektech více v sekci 3.1.

2.3.0.1 Základní uživatelské akce

Webová aplikace umožňuje zejména následující uživatelské akce.

- Zobrazení hlavní stránky s přehledem projektů.
- Procházení adresářové struktury indexovaných souborů.
- Zobrazování souboru s vygenerovanými odkazy na definice symbolů, funkcí, ...
- Zobrazování historie úprav souboru mezi jednotlivými změnami.
- Zobrazování historie úprav adresáře.
- Zobrazování anotací v souboru (informace o jednotlivých řádkách – autor poslední změny, datum poslední změny, text změny).
- Porovnávání historie úprav souboru mezi jednotlivými změnami.
- Stáhnutí souboru.
- Vyhledávání s různými parametry:
  - full-text vyhledání,
  - vyhledávání definice symbolu,
  - vyhledávání použití symbolu,
  - vyhledávání pouze v konkrétní cestě na souborovém systému,

[7] http://tomcat.apache.org/
[8] https://glassfish.java.net/
[9] http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html



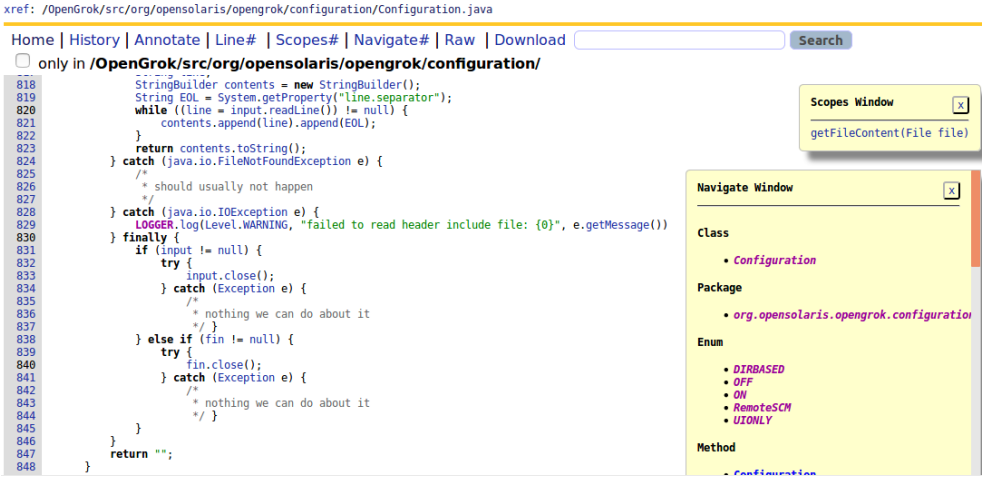
Obrázek 2.3: Uživatelské rozhraní hlavní stránky OpenGroku

- vyhledávání v historii,
- vyhledávání podle typu souboru (C, C#, Java, ...),
- vyhledávání v jednom nebo v několika projektech.

2.3.0.2 Uživatelské rozhraní

V úvodu již bylo naznačeno, že uživatelské rozhraní je sporé, ale poskytuje rychlou navigaci k dosažení uživatelského cíle, často pomocí klávesových zkratek. Pro lepší představu je možné vidět na obrázku 2.3 hlavní stránku OpenGroku, obsahující vyhledávací formulář a seznam projektů, které OpenGrok rozeznal ve zdrojovém adresáři. Také je vidět možnost prohledávat jeden nebo více projektů, na obrázku je konkrétně vybráno 5 projektů k prohledávání „OpenGrok“, „test“, „test2“, „test3“ a „test4“.

Na obrázku 2.4 je vyobrazen náhled konkrétního souboru s podporou zvýrazněné syntaxe a několika pomocných nástrojů pro navigaci ve zdrojovém kódu. Návrh na členění projektů do skupin by měl na hlavní stránce toto rozdělení zobrazovat a umožnit uživateli výběr všech projektů v jedné skupině.



Obrázek 2.4: Uživatelské rozhraní při prohlížení souboru v OpenGroku

Kapitola 3

Abstraktní členění projektů do skupin

První částí této práce je navrhnout mechanismus, kterým by bylo možné uspořádat projekty v OpenGroku do skupin, s nimiž se často lépe manipuluje než se samotnými projekty. Tyto skupiny mohou sloužit k snadnějšímu výběru prohledávaných projektů na hlavní stránce OpenGroku (obrázek 2.3) a budou sloužit k pohodlnější kontrole autorizace v OpenGroku přes jednotlivé skupiny projektů. Nejprve je nutné se seznámit s konceptem projektů v OpenGroku.

3.1 Projekty

OpenGrok podporuje dva operativní módy:

1. mód bez projektů,
2. mód s projekty.

3.1.1 Mód bez projektů

Tento mód interpretuje zdrojový adresář a jeho podadresáře jako pouhé adresáře. Pokud existuje informace o použitém SCM pro zdrojový adresář, je tato přidána do konfigurace a využita při získávání historie a anotací. Tento režim je vhodný pro rychlé spuštění nad zdrojovými kódy jednoho produktu, pro práci s více produkty je nevhodný, protože pro vyhledávání v konkrétním produktu nebo jeho části je třeba využít filtr na cestu v souborovém systému

3.1.2 Mód s projekty

V případě kdy bychom chtěli poskytovat uživatelům více produktů v módu bez projektů, bylo by nutné připravit několik oddělených instancí OpenGroku, pro každý projekt jednu. Každá by obsluhovala HTTP požadavky na jiném

```
src ..... Zdrojový adresář
├── bigbandbiskupska ..... Repositář se SCM typu git
├── haxtip ..... Repositář se SCM typu git
├── junit4 ..... Repositář se SCM typu git
├── LdapPlugin ..... Repositář se SCM typu Mercurial
├── open-grok ..... Repositář se SCM typu git
├── OpenGrok ..... Repositář se SCM typu git
├── test ..... Repositář se SCM typu git
├── test2 ..... Repositář se SCM typu git
├── test3 ..... Obyčejný adresář
├── test4 ..... Repositář se SCM typu git
├── test5 ..... Obyčejný adresář
├── Untitled Folder ..... Obyčejný adresář
└── ZopenGrok ..... Repositář se SCM typu git
```

Obrázek 3.1: Obsah zdrojového adresáře OpenGroku

portu, stejně tak jako by naslouchala konfiguračním změnám na jiném portu. Taková sestava se těžkopádně konfiguruje a složitě udržuje, proto OpenGrok nabízí mód s projekty. Při módu s projekty je každý adresář ve zdrojovém adresáři uvažován jako projekt a každý projekt lze tak spravovat odděleně v jedné instanci OpenGroku. Informace o použitém SCM se může lišit pro každý projekt, to je reflektováno v konfiguraci, kterou vytváří Indexer.

V konfiguraci instance na obrázku 2.3 byly použity projekty a když se podíváme na obsah zdrojového adresáře, tak vypadá jako ve výpisu 3.1. Každý adresář ve zdrojovém adresáři v hloubce 1 je chápán jako projekt a odpovídající informace o typu SCMje přiřazena k tomuto projektu.

3.2 Repositáře

Při objevování projektů v OpenGroku se informace o tom, zda-li projekt je typem nějakého známého SCM nástroje ukládá vedle projektů do pole repositářů. Pokud navíc tento známý SCM nástroj podporuje vnořené repositáře, tak probíhá skenování rekurzivně v daném projektu do určité konfigurovatelné hloubky a všechny nalezené podrepositáře se přidávají do pole repositářů. Tato metodika způsobí, že uživatelé mohou korektně prohledávat historii podrepositářů nějakého projektu a dostávají správná data.

Obecně tedy platí, že v jedné instanci OpenGroku je maximálně tolik projektů, kolik je adresářů ve zdrojovém adresáři a zároveň počet repositářů může být větší (i menší) než je počet projektů.

Na detekci repositářů nemá vliv druh operačního módu vzhledem k projektům, repositáře jsou detekovány i v módu bez projektů.

Uživatel může vizuálně zjistit detekované repositáře na hlavní stránce (obrázek 2.3), kde projekty, jež disponují informací o repositáři, jsou uvedeny v tabulce pod vyhledávacím formulářem, vnořené repositáře se zobrazují vždy u svého nadřazeného repositáře. Oproti tomu výběr prohledávaných projektů ve vyhledávacím formuláři obsahuje všechny projekty, tedy i ty, které informaci o repositáři nemají.

3.2.1 Definice pojmu repositář

Pokud to nebude explicitně uvedeno jinak, budeme pro zjednodušení v následujícím textu v kontextu projektů používat pojem repositář označující projekt s alespoň jednou informací o repositáři v něm obsaženou.

3.2.2 Úprava pojmu projekt

Projekt dle kontextu bude označovat dva různé pojmy. Pokud bude explicitně uvedeno, že v textu se uvažují projekty a repositáře odděleně, tak projekt označuje všechny projekty, které nejsou repositáři. V ostatních případech se projektem rozumí jakýkoliv projekt nebo repositář.

3.2.3 Synchronizace

Synchronizace jednotlivých stažených repositářů oproti jejich vzdálenému obrazu je v režii správce instance OpenGroku a je předmětem automatizovaných úloh spouštěných přes plánovač událostí (cron).

OpenGrok podporuje inkrementální index a do již existujícího indexu zahrnuje pouze nově přijaté změny, které správce instance stáhl.

3.3 Skupiny

3.3.1 Analýza požadavků

3.3.1.1 Funkční požadavky

Z hlediska manipulace se skupinami, jejich zobrazování a autorizace jsou na návrh kladeny následující funkční požadavky:

- možnost zanoření skupin minimálně do třetí úrovně,
- zobrazení na hlavní stránce v hierarchickém stylu,
- skupina musí rozlišovat projekty a repositáře[10] (kvůli zobrazování),
- snadná konfigurace příslušnosti projektu[11] k dané skupině,

[10] viz definice 3.2.1 a 3.2.2

[11] projekt nebo repositář viz definice 3.2.2

- projekt může být ve více skupinách zároveň.

Přestože klademe požadavek zanoření do minimálně třetí úrovně, nejčastějším případem uživatelské konfigurace bude nejspíše sada skupin v nejvyšší úrovni třídící projekty disjunktně do všech skupin.

3.3.1.2 Nefunkční požadavky

- Vyhnutí se další závislosti na knihovně třetí strany, pokud to není nezbytně nutné.

Tento jediný požadavek vyplývá z potřeby nezvýšit počet závislostí OpenGroku na dalších knihovnách a také z potřeby nenavýšit zbytečně velikost distribuovaného balíčku OpenGroku.

3.4 Analýza možných řešení

Abychom vyhověli požadavku zanořování skupin do sebe, bude nutné realizovat hierarchii skupin pomocí stromové struktury, kde kořenem bude vždy skupina první úrovně. Pro větší počet skupin první úrovně vznikne seznam stromových struktur pro každou takovou skupinu.

3.4.1 Swing Tree

Java obsahuje poměrně kvalitní řešení pro práci se stromovými strukturami a to se nachází v knihovně Swing v balíčku `javax.swing.tree.*`[12]. Mezi rozhraní a třídy v tomto balíčku je i třída `TreeNode`, pomocí které je vývojář schopný vytvořit jakoukoliv stromovou strukturu.

Výhody

- Vestavěné rozhraní v Javě.

Nevýhody

- Metody mají příliš obecný název pro účely implementace skupin v OpenGroku (neexistují metody `getProjects()`, `getGroups()`):
  - `children()`,
  - `getRoot()`,
  - `getNumberOfChildren()`,
  - ...,
- Do implementace musíme dodat další podpůrné metody, které naše specializace vyžaduje.

[12] http://docs.oracle.com/javase/8/docs/api/javax/swing/tree/package-summary.html

3.4.2 Implementace třetích stran

Existují generické implementace stromových struktur v Javě a jsou svým rozhraním podobné vestavěné implementaci v knihovně Swing. Jako příklad lze uvést knihovnu GenericTree[13], která je k dispozici pod licencí BSD-3-Clause. Další možnou implementací, jež ovšem neobsahuje žádnou licenci, je kód pro C# a Javu v repositáři yet-another-tree-structure[14].

Nevýhody

- Metody mají příliš obecný název pro účely implementace skupin v OpenGroku (neexistují metody `getProjects()`, `getGroups()`):
  - `getChildren()`,
  - `getRoot()`,
  - `getChildCount()`,
  - ...,
- Do implementace musíme dodat další podpůrné metody, které naše specializace vyžaduje.

3.4.3 Vlastní implementace

Dalším možným řešením je implementovat si stromovou strukturu po svém, bez využití výše zmíněného balíčku. Implementace takové struktury v Javě není příliš složitá.

Výhody

- Jména metod můžeme volit s přihlédnutím na kontext jejich využití.

Nevýhody

- Neimplementuje žádné vestavěné rozhraní Javy.

3.5 Návrh řešení

Z výše popsané rešerše zvolíme vlastní implementaci. Docílíme tím větší čitelnosti v kontextu kódu OpenGroku i pro ostatní vývojáře tohoto nástroje za cenu podobně náročné implementace jako při využití rozhraní v knihovně Swing. Také tím splníme nefunkční požadavek dle sekce 3.3.1.2.

[13] https://github.com/vivin/GenericTree
[14] https://github.com/gt4dev/yet-another-tree-structure

3.5.1 Příslušnost projektu ke skupině

Zvláštní pozornost musíme věnovat návrhu snadné konfigurace příslušnosti projektu[15] ke skupině. Zde je několik možností:

1. výčet projektů,
2. společný prefix pro skupinu projektů,
3. regulární výraz popisující široké spektrum projektů.

Výčet projektů je velmi nežádoucí pro instance s větším množstvím projektů, konfigurace takové sestavy se nepřehledně udržuje. Navíc pro každý nový projekt se musí konfigurace skupin změnit.

Společný prefix může být zajímavým řešením, nicméně skupiny mohou seskupovat i velmi nesourodé množiny projektů, u kterých nemusí být zaručeno, že budou mít společný prefix.

Z předchozích důvodů se jeví jako ideální volba regulární výraz popisující množinu projektů. Regulární výraz navíc pokrývá obě zmíněné možnosti – výčet projektů může být zapsán např. jako `projekt-1|projekt-2|projekt-3|.*-projekty-.*` a společný prefix pro více projektů jako `prefix\_[a-z]+`. Každá skupina bude obsahovat projekty vyhovující danému regulárnímu výrazu skupiny.

3.5.2 Návrh rozhraní skupiny

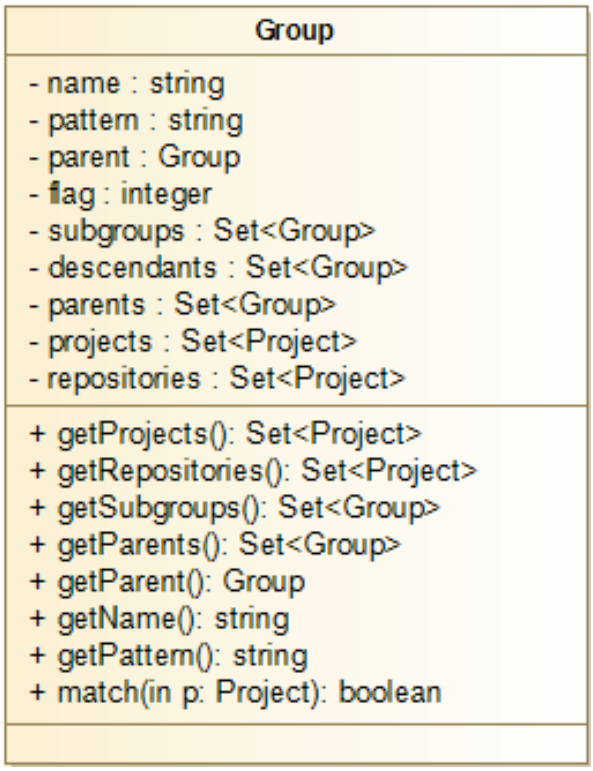
Funkční požadavky na objekt skupiny v sekci 3.3.1.1 jsou splněny následujícím rozhraním třídy `Group` zobrazeném na obrázku 3.2. Tento obrázek obsahuje pouze čtecí metody na různé vlastnosti objektu `Group`, zapisovací metody jsou také pochopitelně součástí implementace, avšak učinily by tento obrázek zbytečně nepřehledným.

3.5.3 Reprezentace v paměti

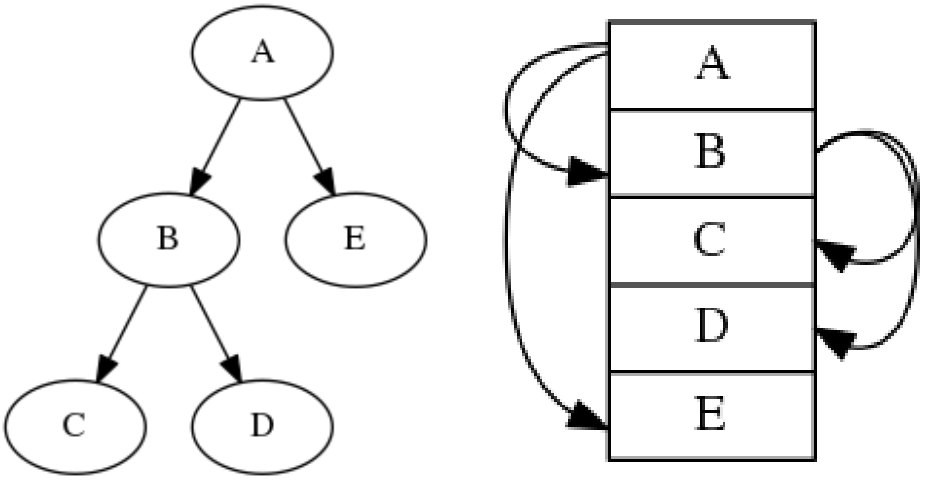
Předpokládejme situaci jako na obrázku 3.3a. Jistě by bylo výhodné tuto strukturu procházet lineární smyčkou pro každý objekt skupiny včetně podskupin. Psaný zdrojový kód by byl více transparentní pro vývojáře a z hlediska výpisů a iterace by byla práce s množinou skupin pohodlnější.

Zavedeme unikátní identifikátor pro skupinu a tím bude její jméno porovnávané bez ohledu na velikost písmen. Následně místo seznamu všech skupin první úrovně zavedeme množinu všech skupin. Tato množina bude seřazená podle identifikátoru skupiny. Toto uspořádání stejných skupin z předchozího odstavce zobrazuje obrázek 3.3b. Relace mezi skupinami (relace rodiče a potomků) zůstávají zachovány i po této transformaci. Zároveň máme možnost

[15] projekt nebo repositář viz definice 3.2.2



Obrázek 3.2: Návrh rozhraní třídy Group



(a) Strom skupin

(b) Množina skupin

Obrázek 3.3: Příklad organizace skupin v paměti

tradičním zápisem smyček v Javě procházet všechny skupiny jako lineární strukturu.

3.5.4 Konfigurace

Konfigurace skupin se nachází v hlavním konfiguračním souboru OpenGroku. To znamená, že zápis skupin do konfigurace je ve formátu XML. V konfiguraci existuje položka `groups`, která je množinou všech skupin v aplikaci organizovaných podle obrázku 3.3b. Příklad zápisu skupiny může být jako ve výpisu 3.1.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <java version="1.8.0_111" class="java.beans.
    XMLDecoder">
3  <object
4 class="org.opensolaris.opengrok.configuration.
    Configuration"
5       id="Configuration0">
6   <void property="groups">
7    <void method="add">
8     <object
9   class="org.opensolaris.opengrok.configuration.Group
      ">
10      <void property="name">
11       <string>skupina 1</string>
12      </void>
13      <void property="pattern">
14       <string>.*-projekty-.*</string>
15      </void>
16     </object>
17    </void>
18   </void>
19  </object>
20 </java>
```

Výpis 3.1: Příklad konfigurace skupiny ve formátu XML

Indexer hlavní konfiguraci OpenGroku **přepisuje** po svém běhu, a proto není vhodná pro perzistentní uložení konfigurace skupin. Pro OpenGrok existuje základní konfigurace, kterou Indexer načítá před svým během a sloučí ji s vygenerovanou konfigurací na konci svého běhu, výsledek uloží na disk a zašle webové aplikaci. V kontextu OpenGroku je tato konfigurace pouze pro čtení „read-only configuration“[16] a je místem trvalých nastavení různých možností OpenGroku, jež nelze nastavit pomocí parametrů Indexeru. Tedy právě konfigurace skupin.

[16]definice pojmu v 3.5.4.1

Příkladné spuštění s konfigurací pro čtení v souboru `/var/opengrok/etc/read-only.xml` by, dle sekce 2.2, mohlo vypadat takto:

```
$ java -jar opengrok.jar \
    -W /var/opengrok/etc/configuration.xml \
    -s /var/opengrok/src \
    -d /var/opengrok/data \
    -R /var/opengrok/etc/read-only.xml
```

nebo pro nástroj `OpenGrok`:

```
$ OPENGROK_READ_XML_CONFIGURATION\
  =/var/opengrok/etc/read-only.xml \
   ./OpenGrok index
```

3.5.4.1 Definice pojmu

Konfigurace pro čtení Označuje soubor načítaný Indexerem při jeho startu, jenž je sloučen s výslednou hlavní konfigurací OpenGroku.

3.5.4.2 Kompatibilita s verzí před změnami

Důležitou poznámkou k tématu konfigurace je to, že pokud konfigurace skupin není uživatelem definovaná (není žádná skupina k dispozici), OpenGrok zobrazuje všechny projekty dle původního návrhu (obrázek 2.3).

3.6 Implementace

Zdrojové kódy obsahující změny pokrývající abstraktní členění projektů do skupin jsou dostupné online ve veřejném repositáři na serveru Github[17] pod verzí 1.1-rc2 a jsou také uloženy na přiloženém médiu k této práci, obsah média je popsán v příloze C.

3.6.1 Transformace

Nejpodstatnější částí implementace je transformace z formátu na obrázku 3.3a do formátu na obrázku 3.3b. Po bezchybném načtení konfigurace ze souboru XML se načtená struktura skupin převede do výhodnějšího formátu. Převod dokumentuje následující útržek zdrojového kódu s komentáři v anglickém jazyce.

```
// the new configuration from the file
Configuration conf = ((Configuration) ret);

// Removes all non root groups.
```

[17] https://github.com/OpenGrok/OpenGrok

```
5 // This ensures that when the configuration is
    reloaded
6 // then the set contains only root groups.
7 // Subgroups are discovered again
8 // as follows below
9 conf.groups.removeIf(new Predicate<Group>() {
10     @Override
11     public boolean test(Group g) {
12         return g.getParent() != null;
13     }
14 });
15
16 // Traversing subgroups and checking for duplicates,
17 // effectively transforms the group tree to a
    structure
18 // supporting an iterator.
19 TreeSet<Group> copy = new TreeSet<Group>();
20 LinkedList<Group> stack = new LinkedList<Group>(conf.
    groups);
21 while (!stack.isEmpty()) {
22     Group group = stack.pollFirst();
23     stack.addAll(group.getSubgroups());
24
25     if (!copy.add(group)) {
26         throw new IOException(
27                 String.format(
28     "Duplicate␣group␣name␣'%s'␣in␣configuration.",
29                         group.getName()));
30     }
31 }
32 conf.setGroups(copy);
```

3.6.2 Klasifikace projektů

Po načtení konfigurace a transformaci, je nutné naplnit skupiny projekty, protože konfigurace projektů i skupin se mohla během běhu Indexeru změnit. Na tomto úryvku je vidět naplnění funkčního požadavku „Projekt může být ve více skupinách zároveň“, viz sekce 3.3.1.1, jelikož je projekt přidán do každé skupiny, která vrací úspěch z metody `match` ověřující regulární výraz skupiny a jméno projektu.

```
1 /**
2  * Classifies projects and puts them in their groups.
3  */
```

```
4  private void populateGroups(Set<Group> groups,
5          List<Project> projects) {
6      if (projects == null || groups == null) {
7          return;
8      }
9      for (Project project : projects) {
10         // filter only groups which
11         // match project's description
12         Set<Group> copy = new TreeSet<>(groups);
13         copy.removeIf(new Predicate<Group>() {
14             @Override
15             public boolean test(Group g) {
16                 return !g.match(project);
17             }
18         });
19
20         // add project to the groups
21         for (Group group : copy) {
22             if (repository_map.get(project) == null)
                   {
23                 // the project is a simple project
24                 group.addProject(project);
25             } else {
26                 // the project has a repository
27                 group.addRepository(project);
28             }
29             project.addGroup(group);
30         }
31     }
32 }
```

3.6.3 Zobrazení na hlavní stránce

Zobrazení na hlavní stránce musí respektovat hierarchii skupin definovaných uživatelem a zároveň se musí držet původního konceptu – na hlavní stránce jsou zobrazeny repositáře[18] a normální projekty jsou zobrazeny pouze ve formulářovém políčku pro výběr projektů.

[18] repositář viz definice 3.2.1

Výpis skupin na hlavní stránce nejlépe dokládá následující úryvek kódu, který implementuje variantu algoritmu Depth First Search (DFS) [4]. V průběhu algoritmu pomocí vlastnosti `flag` nejprve označí skupinu za navštívenou, ale ponechá ji v seznamu skupin pro další průchod a teprve když je skupina spatřena podruhé, je ze seznamu odebrána a HTML značky obklopující skupinu jsou uzavřeny.

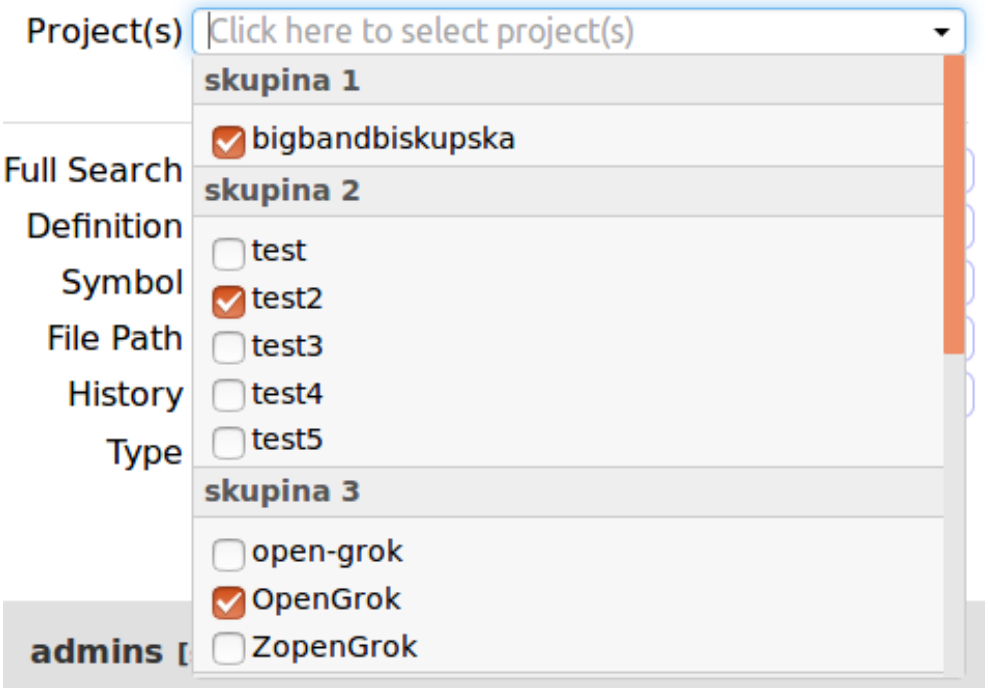
```
1  LinkedList<Group> stack = new LinkedList<>();
2  for ( Group x : groups ) {
3      if (x.getParent() == null) {
4          x.setFlag(0); // not visited yet
5          stack.addLast(x);
6      }
7  }
8
9  while ( ! stack.isEmpty() ) {
10     Group group = stack.element();
11
12     if (group.getFlag() > 0) {
13         // already processed - closing visited group
14         stack.pollFirst();
15         // close the group HTML tags here
16         closeTags();
17         continue;
18     }
19
20     // group's been visited now
21     stack.element().setFlag(1);
22
23     for (Group x : pHelper.getSubgroups(group)) {
24         // not visited yet
25         x.setFlag(0);
26         stack.addFirst(x);
27     }
28
29     // print the group HTML here
30     printGroup(group);
31 }
```

Výsledek algoritmu pro množinu skupin je možné vidět na obrázku 3.4 – skupiny je možné zanořovat do sebe a také je v ukázce patrné, že jeden stejný projekt může být zároveň ve více skupinách. Každá skupina má možnost se sbalit a zakrýt tím i všechny vnořené skupiny, stejně tak jako se umí analogicky rozbalit; to může zlepšit navigaci na hlavní stránce pro instance s mnoha projekty, viditelný příklad zabalené a rozbalené skupiny je na obrázku 3.4.

Obrázek 3.4: Náhled rozhraní hlavní stránky OpenGroku se skupinami

Všechny projekty jsou zobrazeny v formuláři s výběrem projektů. Využito je také rozdělení do skupin, nicméně protože HTML element `<optgroup>` nemůže obsahovat násobně vnořené `<optgroup>` elementy, výběr obsahuje všechny skupiny bez viditelné hierarchie, příkladný rozdíl je patrný na obrázku 3.5 pro skupinu „skupina 2“, která by měla být vnořená ve skupině „skupina 1“, ale není.

Zajímavou zkratkou ve formulářovém políčku je, že jednoduché kliknutí na název skupiny označí automaticky všechny projekty uvnitř dané skupiny. Pokud navíc při kliknutí držíme stisknuté tlačítko `ctrl`, tak se obsah skupiny přidá k již aktuálnímu výběru.



Obrázek 3.5: Náhled rozhraní formuláře pro výběr projektů se skupinami

3.6.4 Zjednoduššení výpisu

Pro jednodušší manipulaci se skupinami, s projekty a s repositáři[19] je pro vývojáře připravena třída `ProjectHelper`, která nabízí zkratky pro získávání uvedených objektů. Tento objekt bude hrát velikou roli při filtrování projektů a skupin během autorizace a měl by být použit vždy, když se projekty nebo skupiny zobrazují uživateli.

3.6.5 Nástroj Groups

Konfiguraci skupin je možné zadávat ve formátu XML do konfigurace pro čtení pro běh Indexeru. Protože XML syntaxe serializovaných Java objektů není pro nezkušeného uživatele příliš přívětivá, nabízí OpenGrok nástroj `Groups`, který umožňuje lehčí manipulaci s konfigurací skupin – přidávání, mazání, výpis. Demonstraci výpisu konfigurace skupin použité pro obrázky 3.4 a 3.5 je možné vidět v následujícím úryvku.

[19] repositář viz definice 3.2.1

```
1 $ tools/Groups list -i /var/opengrok/etc/
    configuration.xml
2 admins ~ "test-project-1|test-project-2|test-project
    -3"
3 skupina 1 ~ "bigbandbiskupska"
4   skupina 2 ~ "test.*"
5 skupina 3 ~ ".*rok"
6 skupina 4 ~ ".*te.*"
```

Skript `Groups` se nachází v distribuovaném balíčku OpenGroku v podadresáři `bin`.

3.7 Testování

Tato část diplomové práce je pokryta jednotkovými testy testující zejména tři aspekty chování:

- schopnost skupinu převádět do XML formátu a zpět,
- schopnost skupiny vyhodit výjimku při nevalidním regulárním výrazu,
- schopnost skupiny klasifikovat projekt dle regulárního výrazu.

Kapitola 4

Autorizační framework

4.1 Bezpečnostní model

Než přistupíme ke druhé části práce, je potřeba zmínit jaký bezpečnostní model se OpenGroku aktuálně používá.

OpenGrok předpokládá, že stroj, na kterém je instance[20] OpenGroku spuštěná, je přístupný pouze administrátorům – správcům instance. Webová aplikace poslouchá na TCP portu pro změny konfigurace a na tomto portu není připraveno žádné zabezpečení, a je tudíž nezbytné, aby se k tomuto portu nedostal nikdo jiný než správce instance. Obdobně pro konfigurační soubory, zdrojový adresář a datový adresář je nutné zařídit, aby nebyly přístupné nikomu jinému než správci instance na úrovni souborového systému.

Webová aplikace OpenGroku nemá žádné zabezpečení proti přístupu od uživatelů. Veškeré zamezení přístupu musí být konfigurováno na jiné úrovni:

- na úrovni servlet kontejneru (Tomcat) nebo
- na úrovni webového serveru (Apache), který přeposílá požadavky na servlet kontejner.

Stručně řečeno OpenGrok spoléhá na to, že autentizaci provádí jiná vrstva. V následujícím textu je nezbytné rozlišovat dva pojmy.

4.1.1 Autentizace

Autentizace je proces ověření identity uživatele [5].

4.1.2 Autorizace

Autorizace je proces ověření přístupových práv uživatele k nějakému zdroji [5].

[20] viz definice 2.1.1

4.2 Autorizační vrstva

Druhou částí této práce je navrhnout v OpenGroku autorizační vrstvu, aby umožnila filtraci projektů nebo skupin v jedné instanci podle kritérií, které si správce instance sám zvolí. V některých instancích OpenGroku může být důležité, že někteří uživatelé mají přístup k některým projektům nebo skupinám a jiní nikoliv. Obecně toto vyplývá z vnitřních bezpečnostních opatření firmy, protože častokrát jsou informace o zdrojovém kódu produktů a jeho prohlížení pod stupněm nejvyšší ochrany.

Autorizační vrstva by se měla zvlášť rozhodovat pro každý projekt a zvlášť pro každou skupinu, protože dle sekce 3.5.4.2 není garantováno, že cílový správce instance používá skupiny.

Nově navržená autorizační vrstva musí být dostatečně obecná, aby umožnila správci instance použít libovolný typ autorizace. Pro příklad může existovat instance, kde jsou jednotlivá oprávnění uložena v souboru na disku a metoda testující klientova oprávnění tedy spočívá ve čtení a interpretaci tohoto souboru. Jiným příkladem může být seznam oprávnění uložený v LDAP strukturách a autorizace probíhá připojením se na LDAP server a sérií dotazů na zjištění jednotlivých oprávnění – situace zadavatele.

4.3 Analýza požadavků

4.3.1 Případy užití

V systému se nachází dvě skupiny osob, před analýzou požadavků by bylo dobré prozkoumat případy užití pro obě skupiny.

4.3.1.1 Uživatelé

Uživatelé přistupují do webové aplikace pomocí prohlížeče, s aktivní autorizační vrstvou by se měli setkat při následujících akcích:

- zobrazení hlavní stránky (viditelné jsou pouze povolené projekty a skupiny),
- zobrazení zdroje uvnitř projektu (uživatel by se neměl dostat do zakázaného projektu),
- vyhledávání v projektech (uživatel by neměl mít možnost vyhledávat v zakázaných projektech),
- používání JavaScript Object Notation (JSON) Application Programming Interface (API) pro vyhledávání v projektech vracející JSON datový formát.

4.3.1.2 Správce instance

Správce oproti tomu má přístup ke konfiguraci a nastavuje různé parametry autorizace, typickými případy užití autorizační vrstvy jsou:

- nastavení autorizační metody,
- nastavení a údržba parametrů autorizační metody,
- kontrola správné činnosti autorizační vrstvy.

Z případů užití můžeme vycházet a zformulovat tak požadavky na autorizační vrstvu.

4.3.2 Funkční požadavky

- Rozhodnutí, jestli má daný HTTP požadavek přístup ke projektu.
- Rozhodnutí, jestli má daný HTTP požadavek přístup ke skupině.
- Umožnit správci instance libovolně si zvolit metodu autorizace.
- Konfigurace by pro jednoduché nastavení měla být adekvátně jednoduchá.

Projekt a skupina jsou jediné dva objekty v OpenGroku, na které půjde uplatnit přístupová oprávnění.

4.3.3 Nefunkční požadavky

- Snížit výpočetní složitost autorizačních operací na minimum.

Přidáním autorizační vrstvy mezi každou operaci s projektem nebo skupinou jistě zvýší dobu odezvy jednotlivých požadavků. Cílem návrhu je vhodnou implementací snížit tuto prodlevu oproti používání stejné instance bez autorizační vrstvy na minimum.

4.4 Analýza možných řešení

4.4.1 Apache Authorization

Webový server Apache obsahuje nabízí sadu modulů určených pro řešení běžných případů autorizace. Nabízené moduly ve verzi 2.4 zahrnují autorizaci oproti Structured Query Language (SQL) databázi, autorizaci oproti LDAP serveru, autorizaci oproti souboru a povolení přístupu pouze přihlášeným uživatelům [6].

Konfigurací serveru Apache můžeme dosáhnout stavu, kdy klientovy požadavky míří na server Apache, ten provede autorizaci a případné povolené požadavky přepošle na servlet kontejner obsluhující instanci OpenGroku. Navíc konfigurace autorizace Apache obsahuje všechny obvyklé zdroje autorizačních oprávnění, takže situace by se mohla zdát vyřešená.

Toto řešení má zásadní nevýhodu. Konfigurace server Apache není schopná podchytit projekty a skupiny v OpenGroku. Pokud bychom předpokládali zadavatelovu situaci, tak touto sestavou pouze můžeme omezit přístup na konkrétní Uniform Resource Locator (URL) webové aplikace OpenGroku pro uživatele vyhovující LDAP filtru. To může být vhodné k prvotnímu odstínění aplikace od nechtěných uživatelů, ale nelze to použít pro rozsáhlou autorizaci pro projekty nebo skupiny.

Dalším možným nastavením serveru Apache jsme schopni spustit na jednom serveru více instancí OpenGroku (například pro každý projekt jednu) a mohli bychom pro každou takovou instanci nastavit jinou metodu autorizace z těch, které jsou v Apache podporované. To se kýženému výsledku přibližuje nejvíce, nicméně v tomto sestavení nelze hledat napříč projekty umístěnými v různých instancích.

Výhody

- OpenGrok pracuje bez jediné změny.

Nevýhody

- Není možné autorizovat uživatele pro projekt nebo skupinu.
- Při více instancí v jednom serveru Apache nelze prohledávat napříč všemi projekty.
- Technika autorizace je závislá na možnostech serveru Apache.

4.4.2 Tomcat Authorization

Servlet kontejnery zpravidla poskytují nadstavbu umožnující autentizaci a autorizaci pomocí propojeného konceptu uživatelů a jejich rolí. Tomcat, uvedený v sekci 2.3, tuto možnost má. Možnosti technik autentizace a autorizace jsou zde podobné jako u serveru Apache, jsou to autentizace a autorizace oproti databázi, oproti LDAP serveru, oproti souboru nebo oproti kolekci držené v paměti za běhu aplikace [7].

Rozdíl mezi serverem Apache je ten, že zde se provádí autentizace i autorizace zároveň. Druhý zásadní rozdíl je, že OpenGrok je spuštěn v tomto servlet kontejneru, a tudíž má přístup k informaci o rolích uživatele.

Veliká nevýhoda tohoto přístupu spočívá v mapování uživatelských rolí na projekty a skupiny OpenGroku. Informace o tomto mapování musí být uložena

ve zdroji autentizace a autorizace a správce musí mít možnost tyto informace upravovat.

Další nevýhoda je, že stejně jako u serveru Apache, nemá Tomcat přístup k informacím o projektech nebo skupinách. Opět lze zabezpečit pouze konkrétní URL webové aplikace, ale nelze vynutit autorizaci pro všechny projekty a skupiny v OpenGroku.

Výhody

- OpenGrok umí získat seznam rolí uživatele.

Nevýhody

- Autentizace a autorizace se provádí v Tomcatu.
- Technika autentizace a autorizace je závislá na možnostech Tomcatu.
- Mapování projektů a skupin na role uživatele.
- Není možné autorizovat uživatele pro projekt nebo skupinu.

4.4.3 Vlastní implementace

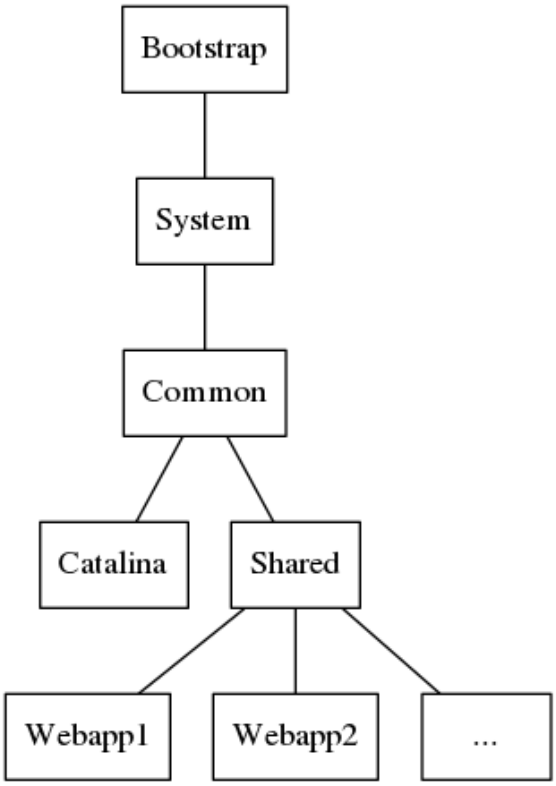
Klíčová nevýhoda většiny možných řešení se projevuje v interní interpretaci projektů a skupin v OpenGroku, která není transparentní pro externí nástroje, a tudíž není možné použít pro autorizaci jinou techniku než vlastní implementaci autorizační vrstvy. Vhodným návrhem můžeme taktéž zvýšit obecnost, než jaká je u serveru Apache nebo kontejneru Tomcat.

4.5 Návrh řešení

Než se ponoříme do návrhu řešení, musíme si vysvětlit několik zásadních vlastností programovacího jazyka Javy týkajících se dynamického načítání tříd za běhu programu.

4.5.1 Dynamické načítání tříd v Javě

O načítání tříd v Java Virtual Machine (JVM) se starají class loadery, což jsou objekty implementující rozhraní `java.lang.ClassLoader` [8]. Práce jednotlivých class loaderů spočívá v interpretaci binárních dat z libovolného zdroje a jejich překlad na objekt třídy `java.lang.Class`. Aplikace mohou sami tvořit vlastní podtřídy `java.lang.ClassLoader` a formovat je do stromu class loaderů. Kořenem tohoto stromu je pak základní class loader „bootstrap class loader“, který nemá žádného předka a stará se o načtení základních tříd Javy při startu programu. Dalším příkladem class loaderu je systémový class loader,



Obrázek 4.1: Příklad využití class loaderů v Tomcatu [8]

který načítá aplikací definované třídy. Kompletní ukázku můžeme naleznout například v kontejneru Tomcat (na obrázku 4.1), jenž sám definuje class loader pro každou aplikaci, kterou v něm spouštíme [9].

Další vlastností načítání tříd je, že v momentě, kdy je třída již jednou načtena a definována, nelze ji načíst a definovat podruhé. Většina class loaderů využívá vhodného cachování, aby se urychlilo načítání tříd a vymazávat položky z jejich interní cache není možné. Toto opatření znamená, že nelze dvakrát načíst stejnou instanci třídy `java.lang.Class` do paměti.

Poslední důležité rozšíření patří identifikaci třídy, protože třída je identifikovatelná nejen plně kvalifikovaným jménem (balíček a název třídy), ale také instancí class loaderu, který se postaral o její načtení [10]. Může se tak stát, že se do aplikace nahrají dvě totožné třídy (stejné plně kvalifikované jméno) pod různými class loadery a nejedná se pak o stejné instance! Ačkoliv to vypadá zrádně, může to mít své využití. Díky této vlastnosti lze obejít opatření v předchozím odstavci a totiž to, že nelze načíst stejnou instanci do paměti dvakrát. Pokud chceme načíst stejnou instanci dvakrát, stačí změnit (nebo zrušit starý a vytvořit nový) class loader pro danou třídu.

Jako shrnutí předešlých poznatků získáváme možnost dynamicky za běhu

aplikace opětovně načítat (re-loadovat) třídy z jejich binární podoby.

4.5.2 Pluginy

Vlastností z předchozí sekce využijeme v našem návrhu autorizační vrstvy a necháme správce instance, aby si sám zvolil autorizační metodu implementací vhodně navrženého rozhraní a následně začlenil svůj modul do OpenGroku. Takovému modulu budeme říkat plugin. Plugin se postará o rozhodnutí o přístupových právech ke skupině nebo projektu, může se například ve své implementaci dotazovat do databáze nebo posílat požadavky na LDAP server.

4.5.3 Framework

Předpokladem bude, že připravený plugin správce ponechá v adresáři, odkud ho OpenGrok bude moci načíst do paměti a začlenit do autorizačního procesu. Je nezbytné připomenout bezpečností model OpenGroku (4.1), který klade zodpovědnost na správce instance, aby zabezpečil přístup k tomuto adresáři pouze administrátorům nebo dalším správcům instance a adresář nebyl přístupný nikomu jinému.

Sofwarové části OpenGroku, která se bude starat o životní cyklus pluginů, o jejich načítání a která bude sloužit pro OpenGrok jako rozhraní pro autorizaci nazveme plugin framework nebo také autorizační framework.

Návrh dynamického načítání tříd sestává z následujících částí:

1. definice API pluginů – Java interface,
2. určit způsob, jakým může uživatel přidat nové pluginy do aplikace,
3. připravit class loader, který načte třídy pluginů.

4.5.4 Návrh rozhraní pluginu

Rozhraní pluginu je klíčová část celé autorizační vrstvy. Určuje jaké operace nám plugin umožní a jak je můžeme používat. Zároveň musí být navrženo tak, aby, pokud možno, již nedocházelo k dalším změnám a správci OpenGroku nemuseli měnit své pluginy podle nového rozhraní.

Při návrhu vyjdeme z funkčních požadavků v sekci 4.3.2. Nutným rozhraním je operace na rozhodnutí o přístupových právech k objektu skupiny a projektu. Dále by bylo vhodné umožnit autorovi pluginu reagovat na životní cyklus pluginu a to zejména na:

- vytváření,
- zánik.

4.5.4.1 Vytváření pluginu

Vytváření nové instance třídy z objektu třídy (`java.lang.Class`) může být ošemetnou záležitostí, protože neznáme předem jaký druh konstruktoru a s jakými parametry má cílová třída. Budeme se proto muset spolehnout na přístupný bezparametrický konstruktor v každé třídě pluginu, kterým bude framework schopný vytvářet nové instance pluginů.

Funkci konstruktoru s předem známými parametry převezme jiná metoda, nazveme ji `load()`. Tato metoda bude volaná na již vytvořené instanci pluginu a plugin si může v této metodě inicializovat své položky a celkově zahájit svůj životní cyklus v paměti aplikace. Nikde není do budoucna garantováno, že tato metoda je volána na právě nově vytvořené instanci.

4.5.4.2 Zánik pluginu

Jelikož plugin může během svého cyklu vytvářet reference na různé zdroje, bylo by vhodné, aby v momentě, kdy má být odstraněn, tyto zdroje uvolnil. Zdrojem může být například již zmíněné databázové spojení. Tuto metodu nazveme `unload()`. Opět není garantováno, že po zavolání této metody bude instance pluginu opravdu zrušena.

4.5.5 Definice rozhraní

Pro rozhraní nám tedy vychází celkem čtyři metody:

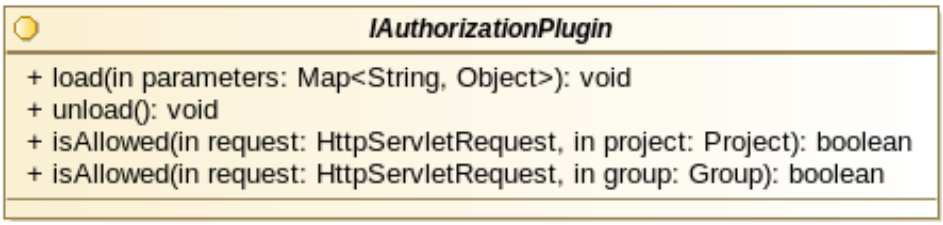
- `load(parameters)` pro inicializaci pluginu,
- `unload()` pro destrukci pluginu,
- `isAllowed(request,project)` pro kontrolu práv k projektu,
- `isAllowed(request,group)` pro kontrolu práv ke skupině.

Podrobný návrh je možný vidět na obrázku 4.2. Při kontrole práv potřebuje plugin znát identitu uživatele webové aplikace. Abychom ponechali návrh dostatečně obecný, předáváme metodám `isAllowed()` vždy ještě objekt aktuálního požadavku. Přes tento požadavek se může autor pluginu dostat k veškerým informacím o identitě uživatele, nezávisle na použité autentizační metodě. K dispozici má hlavičky požadavku a sezení uživatele (session).

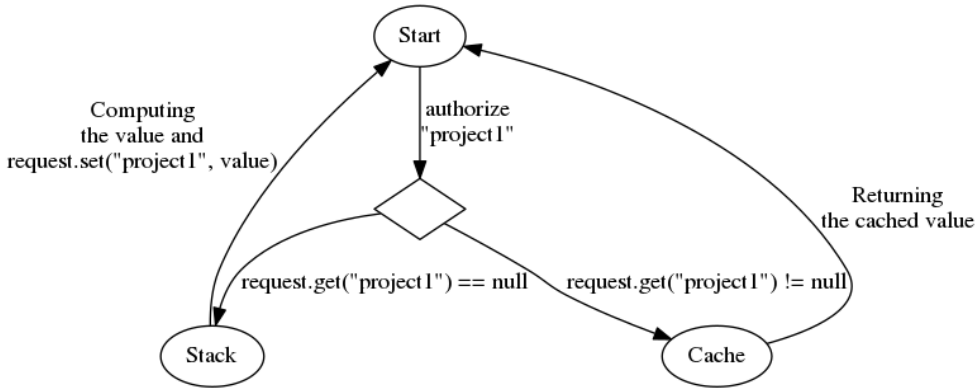
Očekáváme tedy, že správce implementuje nabízené rozhraní a jeho plugin pak může být zahrnut to webové aplikace OpenGroku.

4.5.6 Kontrola práv

Rozhodnutí o povolení nebo zakázání skupiny či projektu je pro daný požadavek definitivní. Pro zvýšení efektivity se framework postará, aby se pro konkrétní skupinu nebo projekt volaly metody pluginu právě jednou, protože



Obrázek 4.2: Návrh rozhraní pro plugin



Obrázek 4.3: Demonstrace cachování v autorizačním frameworku

se předpokládá, že tyto metody mohou být výpočetně drahé. Výsledky prvního volání pro daný projekt nebo skupinu si framework uloží do objektu požadavku a při příštím požadavku na autorizaci použije již vypočtenou hodnotu. Do objektu požadavku lze uložit libovolný atribut, a proto může sloužit jako dočasné úložiště pro již vypočtené hodnoty. Tento postup demonstruje obrázek 4.3.

Aby se však zabránilo nadbytečným restrikcím, tak framework nepředpokládá, že autorizační právo pro konkrétní skupinu se automaticky přenáší na vnořené projekty a skupiny. Příkladem může být, že pokud plugin povolí (plugin rozhodne metodou `allow()` o povolení pro) skupinu A, tak to neznamená, že všechny projekty a podskupiny v ní obsažené jsou také povolené. Plugin se musí automaticky sám postarat o to, že povolí všechny vnořené skupiny a projekty, pokud se to od jeho implementace očekává. Zároveň, pokud se jedná o vnořenou skupinu, je vhodné povolit všechny rodičovské skupiny, aby se tato dostala do výpisu na hlavní stránce.

Toto návrhové rozhodnutí bylo učiněno, aby neomezovalo plugin v rozhodovacích akcích. Když by plugin povolil celou skupinu A (včetně podskupin a projektů), neexistuje již možnost, jak selektivně zakázat jediný vnořený projekt nebo podskupinu. Pokud se toto návrhové rozhodnutí ukáže v praxi jako zbytečné a svazující, bude návrh pozměněn.

4.5.7 Organizace pluginů

Druhá klíčová část autorizační vrstvy je způsob reprezentace a uložení pluginů v OpenGroku. A to, jakým způsobem bude probíhat autorizace přes jednotlivé pluginy. Z funkčních požadavků v sekci 4.3.2 je pro nás důležitý poslední požadavek, který ovšem nespecifikuje, jak pluginy v OpenGroku organizovat. Jeho podstatou je, aby nejčastější případ užití šel snadno nakonfigurovat. Je tedy čistě na nás, jak pluginy ukládat a provádět autorizaci.

4.5.7.1 PAM Framework

Pluggable Authentication Modules (PAM) Framework přináší možnost oddělení autentizačního procesu od funkcionalit konkrétních aplikací skrze transparentní rozšiřitelné API. Jeho hlavní výhodou je, že používá systém zásuvných modulů, kterými lze libovolně rozšířit možnosti autentizace v systému a zároveň cílové aplikace mohou být odstíněny od procesu a způsobu autentizace a neovlivní to jejich chování [11].

Architektura PAM Frameworku je na obrázku 4.4. Jednotlivé aplikace přes jednotné API komunikují s knihovnou PAM Frameworku. Tato knihovna současně poskytuje rozhraní pro autentizační moduly implementující některou ze čtyř možných kategorií. Výsledkem je, že aplikace mohou komunikovat s autentizačními moduly přes jednotné API a implementace modulů je oddělena od aplikací.

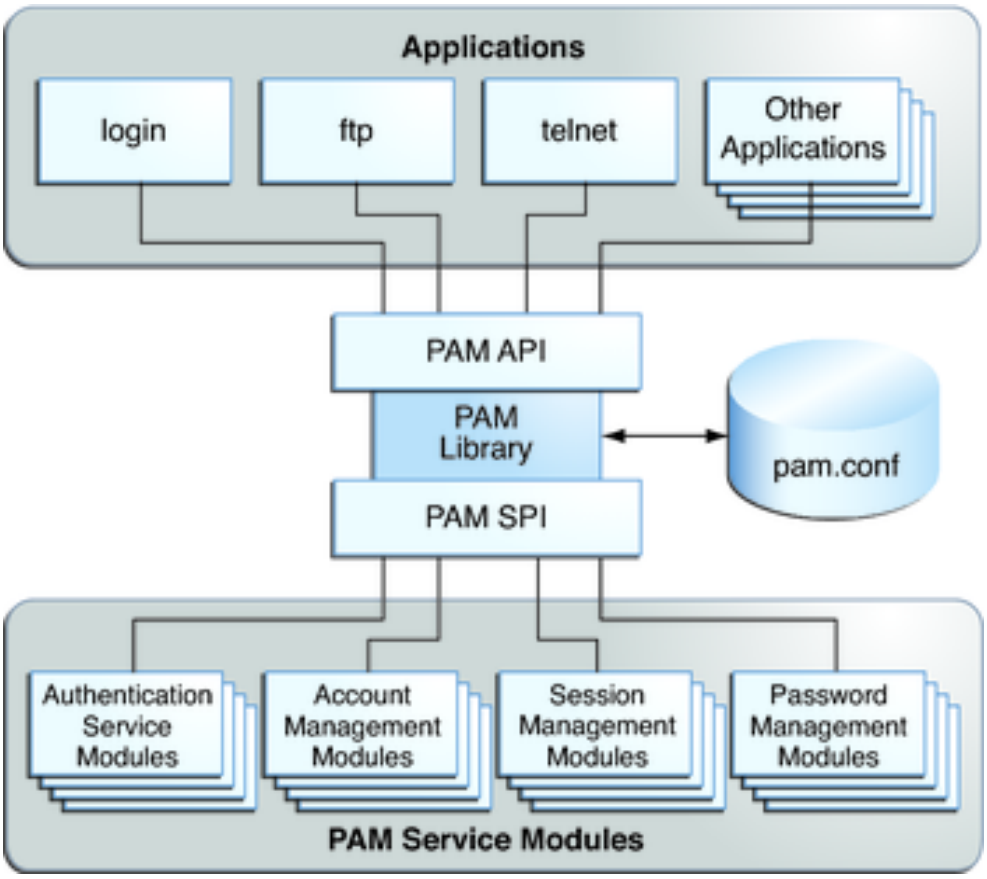
Jádrem frameworku je konfigurační soubor `pam.conf`. Příklad jeho syntaxe je zobrazen ve výpisu 4.1 [12].

```
1  su      auth required        pam_inhouse.so.1
2  su      auth requisite       pam_authtok_get.so.1
3  su      auth required        pam_dhkeys.so.1
4  su      auth required        pam_unix_auth.so.1
5
6  login    auth requisite      pam_authtok_get.so.1
7  login    auth required       pam_dhkeys.so.1
8  login    auth required       pam_unix_auth.so.1
9  login    auth required       pam_dial_auth.so.1
10 login    auth optional       pam_inhouse.so.1
11
12 rlogin   auth sufficient     pam_rhosts_auth.so.1
13 rlogin   auth requisite      pam_authtok_get.so.1
14 rlogin   auth required       pam_dhkeys.so.1
15 rlogin   auth required       pam_unix_auth.so.1
```

Výpis 4.1: Příklad syntaxe a konfigurace pam.conf

Stručně vysvětleno uvedená syntaxe je vždy ve tvaru

```
1 akce   typ   vyznam modul
```



Obrázek 4.4: Architektura PAM Frameworku [11]

a to přesně vystihuje, jak s určitou akcí nakládat. Navíc PAM zavádí pro sekce, kde se shoduje akce a typ akce pojem „stack“ - seznam modulů. Význam klíčových slov „required“, „requisite“ a dalších bude předmětem budoucích odstavců, ale v tuto chvíli je nutné poznamenat, že tato slova indikují, jaký význam se dává návratové hodnotě z daného modulu. Tato návratová hodnota pak ovlivňuje průchod stackem modulů a výsledek autentizace.

V ukázce konfigurace (4.1) pozorujeme tři stacky pro různé programy „su“, „login“ a „rlogin“ a v každé akci je trochu jiný seznam modulů a jejich význam pro výsledek autentizace se také liší.

Podstatnou vlastností je také konfigurovatelné pořadí volání jednotlivých modulů, které jsou vždy volány ve stejném pořadí, v jakém jsou za sebou uvedeny v konfiguračním souboru.

Závěrem lze říci, že tento způsob reprezentace pluginů v OpenGroku by mohl být adekvátně obecný a zároveň přímočarý, proto se uvedeným schématem budeme inspirovat. Přestože PAM Framework je framework zaměřený na autentizaci, můžeme schéma se stacky, konfigurací a procházením stacků

lehce aplikovat na autorizaci a autorizační pluginy. Správce instance mající zkušenost s PAM Frameworkem bude mít navíc výhodu při psaní konfigurace.

4.5.7.2 Autorizační flagy

Dle terminologie PAM Frameworku se klíčová slova „required“, „requisite“ a další nazývají kontrolní flagy „control flags“. V OpenGroku není zapotřebí celé široké spektrum, jaké nabízí PAM Framework, nicméně podmnožinu těch flagů využijeme. Konkrétně použijeme tyto tři:

- REQUISITE,
- REQUIRED,
- SUFFICIENT.

Definici těchto flagů přejmeme přímo z konfigurace PAM Frameworku [12], pouze ji pozměníme do našeho kontextu. Tyto tři flagy zahrneme do návrhu autorizační vrstvy pro OpenGrok.

REQUISITE Pokud daný plugin vrátí úspěch, pokračujeme dál v procházení stackem. V případě, že je vrácen neúspěch, je procházení stackem ukončeno bez vyvolání dalších pluginů ve stacku a návratová hodnota je neúspěch.

REQUIRED Pokud daný plugin vrátí úspěch, pokračujeme dál v procházení stackem. V případě, že je vrácen neúspěch, bude výsledkem celého stacku neúspěch, nicméně pokračujeme dál v procházení stackem s vyvoláním všech zbývajících pluginů.

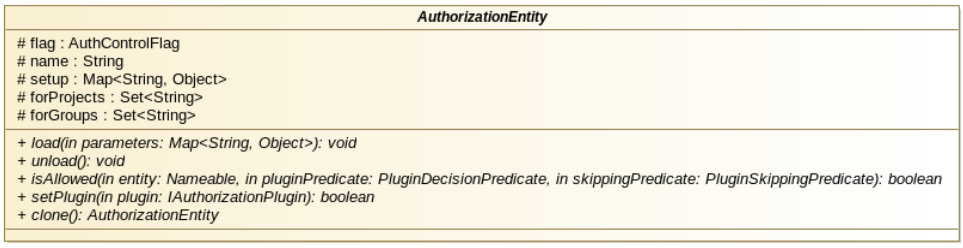
SUFFICIENT Pokud daný plugin vrátí úspěch a žádný jiný REQUIRED plugin nevrátil neúspěch, okamžitě se ukončí procházení stacku s návratovou hodnotou úspěchu bez vyvolání dalších pluginů ve stacku. V případě, že plugin vrátí neúspěch, neovlivní to výsledek stacku.

4.5.7.3 Stack pluginů

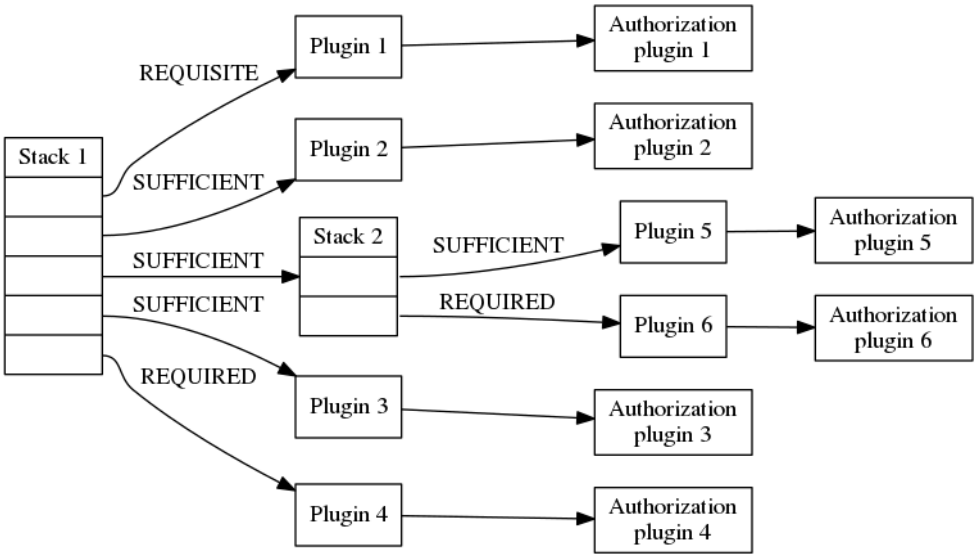
V OpenGroku nadefinujeme dvě nové entity:

1. Stack,
2. Plugin.

Obě entity mají přiřazený autorizační flag dle sekce 4.5.7.2. Od tohoto momentu také rozlišujeme autorizační plugin – třída dodaná správcem a nahrána do paměti za běhu aplikace – a plugin – obálku pro autorizační plugin poskytující autorizační rozhraní delegující klíčové operace na vnořený autorizační plugin. Kompletní ukázku jednoduché konfigurace je možné vidět na



Obrázek 4.5: Návrh rozhraní autorizační entity



Obrázek 4.6: Ukázka složení stacku a pluginů

obrázku 4.6. Obě entity schováme pod abstraktní třídu autorizační entity, jejíž základní rozhraní můžeme pozorovat na obrázku 4.5.

4.5.7.4 Definice pojmů

Autorizační flag Autorizační flag je klíčové slovo označující intepretaci výsledku autorizačního rozhodnutí pro další rozhodování o autorizaci. V OpenGroku je to jedna z možných hodnot definovaných v sekci 4.5.7.2.

Autorizační entita Autorizační entita je rodič (v objektovém návrhu) obou tříd Stack i Plugin. Obsahuje některé společné implementace a zároveň několik abstraktních metod, které stack a plugin implementují různými způsoby. Její rozhraní je na obrázku 4.5.

Stack Stack je nově definovaná entita v OpenGroku, obsahuje další stacky nebo pluginy, při autorizaci se rekurzivně vnořuje do obsažených stacků a vyhodnocuje je jako celek. Návratové hodnoty kombinuje dle významu autorizačních flagů.

Plugin Plugin je nově definovaná entita a obsahuje instanci autorizačního pluginu a tedy činí konkrétní autorizační rozhodnutí pro daný požadavek a projekt nebo skupinu. Výsledek autorizace je interpretován dle autorizačního flagu u daného pluginu.

Autorizační plugin Od této chvíle definujeme pojem autorizační plugin, který označuje plugin dodaný správcem nahraný za běhu webové aplikace do paměti a provádějící konkrétní autorizační rozhodnutí o projektu nebo skupině. Tedy označuje pojem uvedený pod sekcí 4.5.2. V následujícím textu je podstatné rozlišovat plugin (viz předchozí odstavec) a autorizační plugin (viz tento odstavec).

Autorizační rozhodnutí Autorizační rozhodnutí označuje rozhodnutí stacku, pluginu nebo autorizačního pluginu o povolení nebo zamítnutí konkrétního projektu nebo skupiny pro konkrétní HTTP požadavek.

4.5.7.5 Cílení na skupiny a projekty

Implementace PAM Frameworku (viz příklad v sekci 4.5.7.1) umožňuje cílit autorizaci na konkrétní akce. V kontextu OpenGroku by to znamenalo použít daný plugin nebo stack pouze pro konkrétní skupinu nebo projekt. Proto přidáme každému stacku a každému pluginu ještě množinu skupin a množinu projektů, které má tento stack nebo plugin autorizovat, a autorizační test pak proběhne jen pro ty skupiny a projekty obsažené v těchto množinách. V případě, že nakonfigurované množiny jsou obě prázdné, autorizační testy proběhnou pro všechny skupiny a projekty, tuto kombinaci hodnot interpretujeme tak, že stack nebo plugin není cílen na žádnou skupinu nebo projekt.

Obdobně bychom mohli cílení na skupiny provést opačným způsobem, aby se stack nebo plugin zapisoval u konkrétní skupiny v konfiguraci. Tento směr ale není možný, protože jméno pluginu nebo stacku nemusí být nutně unikátní, a tudíž by toto mapování nebylo jednoznačné.

Zvážíme-li, že by se stack nebo plugin zapisoval přímo ke skupině jako objekt (ne jako řetězec popisující jméno jako v předchozím odstavci), tak takové řešení by bylo možné. Nevýhodou je, že by se konfigurace autorizace rozdělila na několik částí a byla by na různých místech v konfiguračním souboru, čímž by se stala velmi nepřehlednou. Z tohoto důvodu zůstaneme u původního návrhu a konfigurace zůstane zvlášť rozdělena na autorizační část a na konfiguraci skupin.

Příklad konfigurace zacíleného pluginu je v následujícím výpisu. V konfiguraci v základním stacku je nakonfigurovaný jeden plugin vyžadující autorizační plugin se třídou `sample.plugin.SamplePlugin` a je cílen na 2 skupiny a 2 projekty.

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.8.0_111" class="java.beans.
   XMLDecoder">
 <object class="org.opensolaris.opengrok.
    configuration.Configuration">
  <void property="pluginStack">
    <void method="add">
     <object class="org.opensolaris.opengrok.
        authorization.AuthorizationPlugin">
      <void property="flag">
       <string>SUFFICIENT</string>
      </void>
      <void property="forGroups">
       <void method="add">
        <string>skupina 1</string>
       </void>
       <void method="add">
        <string>skupina 2</string>
       </void>
      </void>
      <void property="forProjects">
       <void method="add">
        <string>project 1</string>
       </void>
       <void method="add">
        <string>repository 1</string>
       </void>
      </void>
      <void property="name">
       <string>sample.plugin.SamplePlugin</string>
      </void>
     </object>
    </void>
  </object>
</java>
```

Výpis 4.2: Příklad konfigurace autorizace

Obrázek 4.7: Návrh rozhraní autorizačního frameworku

4.5.8 Autorizační framework

Rozhraní pro práci s autorizačními pluginy bude v OpenGroku poskytovat autorizační framework, `AuthorizationFramework.java`. Instance této třídy bude ke správné funkčnosti potřebovat dvě informace:

- adresář obsahující správcem definované autorizační pluginy,
- instanci stacku (může být i prázdný).

Tato instance bude umožňovat opakované nahrání autorizačních pluginů do paměti a základní rozhraní pro testování autorizace pro skupinu a projekt. Návrh můžeme vidět na obrázku 4.7.

Načítání autorizačních pluginů postupuje podle následujících kroků:

1. získá se aktuální konfigurace autorizačního stacku první úrovně (výchozího stacku),
2. framework načte všechny autorizační pluginy v adresáři s pluginy do paměti a
   - přidá novou instanci autorizačního pluginu do všech míst ve stacku, kde je to vyžadováno;
   - pokud takové místo neexistuje, přidá novou instanci autorizačního pluginu na konec stacku s flagem „required“;
3. další body jsou uvedeny v implementaci v sekci 4.6.2.

Výsledkem uvedeného postupu je, že všechny třídy implementující rozhraní autorizačního pluginu se dostanou do autorizačního stacku a jsou:

- seřazeny dle konfigurace správce nebo
- náhodně, při objevování tříd, přidány na konec stacku

Závěrem lze říci, že pokud chce správce využívat aktivně více než jeden autorizační plugin, je vhodné připravit pro autorizační stack konfiguraci. V opačném případě riskuje, že více autorizačních pluginů nemusí vždy být volány ve stejném pořadí.

4.5.9 Kompatibilita s verzí před změnami

Dodané změny nijak nenaruší zpětnou kompatibilitu. Výchozí hodnota pro autorizační stack v autorizačním frameworku je prázdný autorizační stack – tedy stack, který ihned povolí všechny požadavky.

4.6 Implementace

Zdrojové kódy autorizační vrstvy jsou dostupné online ve veřejném repositáři na serveru Github[21] pod verzí 1.1-rc2 a jsou také uloženy na přiloženém médiu k této práci, obsah média je popsán v příloze C. Kód autorizačního frameworku včetně ostatních souvisejících tříd se nachází v balíčku `org.opensolaris.opengrok.authorization`.

4.6.1 Autorizační plugin

Definice rozhraní tohoto pluginu je uvedena v sekci 4.5.5. Při kontrole práv potřebuje plugin znát identitu uživatele webové aplikace. Abychom ponechali návrh dostatečně obecný, předáváme metodám `isAllowed()` vždy ještě objekt aktuálního HTTP požadavku. Tento požadavek je popsán třídou `HttpServletRequest`[22] a přes tuto třídu se může autor autorizačního pluginu dostat k veškerým informacím o identitě uživatele, nezávisle na použité autentizační metodě. K dispozici má hlavičky požadavku, sezení uživatele (session) nebo dostupnou autentizaci ze servlet kontejneru – přes metody `getRemoteUser()` či `getUserPrincipal()`.

4.6.2 Autorizační framework

Z implementace autorizačního frameworku je nejpodstatnější metoda nahrávající všechny objevené autorizační pluginy v jejich adresáři do paměti a uložení těchto autorizačních pluginů do konfigurovaného stacku.

Protože požadavek na nahrání pluginů do paměti může přijít kdykoliv, nezávisle na požadavcích na webový server, je nezbytné, aby výměna starého stacku za nový (mohly se změnit autorizační pluginy, ale i konfigurace stacků a pluginů) proběhla atomicky vzhledem k probíhajícím autorizacím.

Načítání pluginů a jejich inicializace může trvat libovolně dlouhou dobu, zvolíme tedy metodu konstrukce nového stacku, zatímco starý stack je používán pro autorizaci, následně vyměníme reference na starý a nový stack v synchronizovaném bloku a poté proběhne destrukce starého stacku, zatímco je pro autorizaci používán stack nový.

Následující ukázka zdrojového kódu ukazuje metodu `reload()`, která se stará o načtení autorizačních pluginů. Základní struktura této metody je:

[21] https://github.com/OpenGrok/OpenGrok

[22] http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequest.html

1. nahrazení starého class loaderu novým (umožní nahrát stejnou třídu do paměti vícekrát, viz 4.5.1),
2. kopie nového stacku z aktuální konfigurace,
3. nahrání všech autorizačních pluginů z adresáře do paměti, rekurzivně se vyhledávají soubory s příponou `*.jar` a `*.class`,
4. inicializace všech instancí autorizačních pluginů (volání metody `load(parameters)` na jednotlivých autorizačních pluginech),
5. prohození instancí,
6. destrukce starých instancí autorizačních pluginů (volání metody `unload()` na jednotlivých autorizačních pluginech).

Kód metody včetně komentářů v anglickém jazyce odpovídá navržené struktuře.

```
/**
 * Calling this function forces the framework to
 * reload its stack.
 *
 * Plugins are taken from the pluginDirectory.
 *
 * Old instances in the stack are removed and
 * a new stack is constructed.
 * Unload and load event is fired on each plugin.
 *
 * @see IAuthorizationPlugin#load(java.util.Map)
 * @see IAuthorizationPlugin#unload()
 * @see Configuration#getPluginDirectory()
 */
@SuppressWarnings("unchecked")
public void reload() {
    if (pluginDirectory == null || !pluginDirectory.
       isDirectory() || !pluginDirectory.canRead()) {
        LOGGER.log(Level.WARNING, "Plugin directory
           not found or not readable: {0}. "
                + "All requests allowed.",
                   pluginDirectory);
        return;
    }
    if (stack == null) {
        LOGGER.log(Level.WARNING, "Plugin stack not
           found in configuration: null. All requests
           allowed.");
```

```
24          return;
25      }
26
27      LOGGER.log(Level.INFO, "Plugins are being
           reloaded from {0}", pluginDirectory.
           getAbsolutePath());
28
29      // trashing out the old instance
30      // of the loaded enables us
31      // to reaload the stack at runtime
32      loader = (AuthorizationPluginClassLoader)
           AccessController.doPrivileged(new
           PrivilegedAction() {
33          @Override
34          public Object run() {
35              return new AuthorizationPluginClassLoader
                   (pluginDirectory);
36          }
37      });
38
39      // clone a new stack not interfering with the
40      // current stack
41      AuthorizationStack newStack = RuntimeEnvironment.
           getInstance().getPluginStack().clone();
42
43      // increase the current plugin version tracked
44      // by the framework
45      increasePluginVersion();
46
47      // load all other possible plugin classes
48      loadClasses(newStack,
49              IOUtils.listFilesRec(pluginDirectory,
50                                   ".class"),
51              IOUtils.listFiles(pluginDirectory,
52                                ".jar"));
53
54      // fire load events
55      loadAllPlugins(newStack);
56
57      AuthorizationStack oldStack;
58      /**
59       * Replace the stack in a write lock
60       * to avoid inconsistent state between
61       * the stack change and currently executing
```

```
62          * requests performing some
63          * authorization on the same stack.
64          *
65          * @see #performCheck is controlled
66          * with a read lock
67          */
68         lock.writeLock().lock();
69         try {
70             oldStack = stack;
71             stack = newStack;
72         } finally {
73             lock.writeLock().unlock();
74         }
75
76         // clean the old stack
77         removeAll(oldStack);
78         oldStack = null;
79 }
```

Mezi veřejné rozhraní autorizačního frameworku patří metody `isAllowed()` pro skupinu a pro projekt. Framework zajišťuje[23], že pro jednu konkrétní skupinu (resp. projekt) se bude průchod stackem provádět právě jednou a pro další volání pro stejnou skupinu (resp. projekt) se použije návratová hodnota předchozího volání.

4.6.3 Stack

Třída `AuthorizationStack` je implementace podtřídy autorizační entity, jejíž návrh je popsán na obrázku 4.5. Klíčovou metodou je `isAllowed(entity,pluginPredicate,skippingPredicate)` volající další metodu `processStack`, která má na starosti prakticky celý průchod autorizační sekvencí, vyhodnocení případných podstacků a pluginů a, dle významu autorizačních flagů, vrátit návratovou hodnotu pro autorizaci `entity` (projekt nebo skupina) v tomto stacku.

Parametry této metody jsou:

1. entita popsaná rozhraním `Nameable` – entita mající jméno, což je v tomto případě projekt nebo skupina;
2. pluginPredicate – predikát vracející úspěch nebo neúspěch pro danou instanci autorizačního pluginu;
3. skippingPredicate – predikát určující, jestli má být daná instance autorizační entity (stack, plugin) přeskočena nebo naopak zahrnuta do výsledného rozhodnutí.

[23] cachování hodnot vysvětleno v sekci 4.5.6

Průchod stackem může rekurzivně volat `isAllowed()` na případné vnořené stacky a zároveň volá `isAllowed()` na pluginech vracející přímé rozhodnutí o autorizaci. Následně je rozhodnutí zpracováno dle definice klíčových slov autorizačních flagů uvedených v sekci 4.5.7.2. Pokud volání `isAllowed()` způsobí výjimku, rozhodnutí z dané autorizační entity je interpretováno jako neúspěch.

Implementace této důležité metody je v následující ukázce, doplněná stručnými komentáři v anglickém jazyce.

```
1  /**
2   * Test the given entity if it should be allowed
3   * within this stack context.
4   *
5   * @param entity the given entity - this is either
6   * group or project and is passed
7   * just for the logging purposes.
8   * @param pluginPredicate predicate
9   * returning true or false for the given entity
10  * which determines if the authorization for such
11  * entity is successful or failed
12  * for particular request and plugin
13  * @param skippingPredicate predicate
14  * returning true if this authorization
15  * entity should be omitted from
16  * the authorization process
17  * @return true if successful; false otherwise
18  */
19 @Override
20 public boolean isAllowed(Nameable entity,
21         PluginDecisionPredicate pluginPredicate,
22         PluginSkippingPredicate skippingPredicate) {
23     boolean overallDecision = true;
24     LOGGER.log(Level.FINER, "Authorization"
25           + "␣for␣\"{0}\"␣in␣\"{1}\"␣[{2}]",
26             new Object[]{entity.getName(),
27                          this.getName(),
28                          this.getFlag()});
29
30     if (skippingPredicate.shouldSkip(this)) {
31         LOGGER.log(Level.FINER, "AuthEntity"
32                   + "␣\"{0}\"␣[{1}]␣skipping"
33                 + "␣testing␣of␣name␣\"{2}\"",
34         new Object[]{this.getName(),
35                      this.getFlag(),
36                      entity.getName()});
```

```
37      } else {
38          overallDecision = processStack(entity,
39                                         pluginPredicate
                                               ,
40                                         skippingPredicate
                                               );
41      }
42
43      LOGGER.log(Level.FINER, "Authorization"
44                 + "␣for␣\"{0}\"␣in"
45                 + "␣\"{1}\"␣[{2}]␣=>␣{3}",
46                new Object[]{entity.getName(),
47                             this.getName(),
48                             this.getFlag(),
49                overallDecision ? "true" : "false"});
50      return overallDecision;
51  }
52
53  /**
54   * Process the stack.
55   *
56   * @param entity the given entity
57   * @param pluginPredicate predicate
58   * returning true or false for the given
59   * entity which determines
60   * if the authorization for such entity is
61   * successful or failed for
62   * particular request and plugin
63   * @param skippingPredicate predicate
64   *returning true if this authorization
65   * entity should be omitted
66   *from the authorization process
67   * @return true if entity is allowed; false otherwise
68   */
69  protected boolean processStack(Nameable entity,
70          PluginDecisionPredicate pluginPredicate,
71          PluginSkippingPredicate skippingPredicate) {
72      boolean overallDecision = true;
73      for (AuthorizationEntity authEntity : getStack())
            {
74
75          if (skippingPredicate.shouldSkip(authEntity))
                {
76              LOGGER.log(Level.FINEST, "AuthEntity"
```

```
77                + "␣\"{0}\"␣[{1}]␣skipping"
78                + "␣testing␣of␣name␣\"{2}\"",
79                      new Object[]{authEntity.getName()
                           ,
80               authEntity.getFlag(),
81               entity.getName()});
82           continue;
83       }
84       // run the plugin's test method
85       try {
86           LOGGER.log(Level.FINEST, "AuthEntity"
87                + "␣\"{0}\"␣[{1}]␣testing"
88                + "␣a␣name␣\"{2}\"",
89                   new Object[]{authEntity.getName()
                        ,
90                                authEntity.getFlag()
                                   ,
91                                entity.getName()});
92
93           boolean pluginDecision = authEntity
94                           .isAllowed(entity,
95                                      pluginPredicate
                                         ,
96                                      skippingPredicate
                                         );
97
98           LOGGER.log(Level.FINEST, "AuthEntity"
99               + "␣\"{0}\"␣[{1}]␣testing"
100              + "␣a␣name␣\"{2}\"␣=>␣{3}",
101                  new Object[]{authEntity.getName()
                        ,
102                               authEntity.getFlag()
                                   ,
103                               entity.getName(),
104                  pluginDecision ? "true" : "false"
                        });
105
106          if (!pluginDecision
107              && authEntity.isRequired()) {
108              // required sets a failure but still
109              // invokes all other plugins
110              overallDecision = false;
111              continue;
112          } else if (!pluginDecision
```

```
113                     && authEntity.isRequisite()) {
114                 // requisite sets a failure
115                 // and immediately returns the
                       failure
116                 overallDecision = false;
117                 break;
118             } else if (overallDecision
119                     && pluginDecision
120                     && authEntity.isSufficient())
                           {
121                 // sufficient immediately
122                 // returns the success
123                 overallDecision = true;
124                 break;
125             }
126         } catch (Throwable ex) {
127             LOGGER.log(Level.WARNING,
128                     String.format("AuthEntity"
129                   + "␣\"%s\"␣has␣failed"
130                   + "␣the␣testing␣of␣\"%s\""
131                   + "␣with␣an␣exception.",
132                             authEntity.getName(),
133                             entity.getName()),
134                   ex);
135
136             LOGGER.log(Level.FINEST, "AuthEntity"
137                   + "␣\"{0}\"␣[{1}]"
138                   + "␣testing␣a␣name␣\"{2}\"␣=>␣{3}",
139                   new Object[]{authEntity.getName()
                          ,
140                                   authEntity.getFlag()
                                      ,
141                                   entity.getName(),
142                                   "false␣(failed)"});
143
144             // set the return value to false
145             // for this faulty plugin
146             if (!authEntity.isSufficient()) {
147                 overallDecision = false;
148             }
149             // requisite plugin may immediately
150             // return the failure
151             if (authEntity.isRequisite()) {
152                 break;
```

```
153                 }
154             }
155         }
156         return overallDecision;
157 }
```

4.6.4 Plugin

Třída `AuthorizationPlugin` slouží jako obálka pro skutečný autorizační plugin dodaný správcem OpenGroku. Atribut `name` u pluginu popisuje jméno třídy autorizačního pluginu, která má být do tohoto pluginu načtena a inicializována. V případě, že je v konfiguraci uvedeno více autorizačních pluginů se stejnou třídou, je nová instance vytvořena pro každou takovou třídu (instance se tedy nesdílí).

4.6.5 Zapojení autorizace do webové aplikace

Jak již bylo zmíněno dříve v sekci 3.6.4 veškeré operace se skupinami a projekty, které se dotýkají uživatele webové aplikace, by měly být prováděny pomocí rozhraní ve třídě `ProjectHelper`. Tato třída poskytuje určité zkratky jak získat množinu projektů, množinu skupin, množinu projektů pro konkrétní skupinu, ...

Zapojení autorizace do zobrazování je tedy poměrně triviální úprava zmíněné sady metod, které pro každou skupinu nebo projekt volají metody autorizačního frameworku pro autorizační rozhodnutí. Při výpisech se v aplikaci často stává, že se některé z dotčených metod volají vícekrát během jednoho požadavku. Pro co nejefektivnější implementaci jsou filtrované množiny projektů a skupin získávány pouze při prvním volání a další volání stejné metody pro stejný požadavek vrací již dříve vyfiltrovanou množinu. Využívá se přitom objekt požadavku, do kterého lze uložit libovolný atribut a může sloužit jako dočasné úložiště pro již vypočtené hodnoty. Téměř stejný postup demonstruje obrázek 4.3.

Na situace, kdy je URL HTTP požadavku spojená s určitým projektem, je připraven autorizační filtr, popsaný v následující sekci, který pro daný projekt provede autorizaci.

Zbylá místa, která vyžadovala autorizaci, jsou také obohacena o volání autorizačního frameworku pro daný projekt nebo skupinu. Stručný výčet těchto míst je:

- filtrace prohledávaných projektů,
- filtrace projektů ve webové službě vracející formát JSON.

4.6.5.1 Autorizační filtr

Specifikace rozhraní Servletů nabízí také možnost zapojit do životního cyklu HTTP požadavku filtr který je pro daný požadavek spuštěn a může tak ovlivnit jeho zpracování. Typickým použitím může být právě kontrola autentizace nebo autorizace [13]. Pokud tedy URL požadavku je asociována s nějakým projektem (to se dá v OpenGroku snadno zjistit[24] – příklad takové URL je ve výpisu 4.3), tak pro tento projekt se provede autorizační ověření a v případě neúspěchu se uživateli vrací stavový HTTP kód `403 Forbidden`. Příklady, kdy je URL spojená s projektem, mohou být například prohlížení historie souboru v projektu, prohlížení zdrojového kódu v projektu (viz obrázek 2.4), stahování souboru v projektu nebo zobrazování anotací u souboru v projektu.

Příklady URL při přístupu do webové aplikace a hledání informace o projektu v URL jsou v následujícím výpisu.

```
1  /xref/OpenGrok/main.c
2  ^^^^ - vyhrazeny prefix
3       ^^^^^^^^ - oznaceni projektu
4                ^^^^^^^ - soubor v~danem projektu
5  /history/OpenGrok/main.c
6  ^^^^^^ - vyhrazeny prefix
7         ^^^^^^^^ - oznaceni projektu
8                  ^^^^^^ - soubor v~danem projektu
9  /css/default/style.css
10 ^^^ - neni vyhrazeny prefix - zadny projekt
```

Výpis 4.3: Příklady URL a detekce projektu

4.6.6 Konfigurace

Konfigurace je jedním z bodů funkčních požadavků v 4.3.2. Nachází se opět v hlavním konfiguračním souboru ve formátu XML pod položkou `pluginStack`, což je instance autorizačního stacku. Výchozí hodnota je prázdný stack se jménem `default stack`.

Indexer ovšem při svém běhu tuto konfiguraci přepisuje (více o tomto procesu viz 3.5.4), a proto se konfigurace pro autorizační vrstvu umisťuje do konfigurace pro čtení a předává se jako parametr Indexeru. Webová aplikace po přijetí nové konfigurace od Indexeru automaticky zavolá metodu `reload()` na autorizačním frameworku a tím obnoví všechny autorizační pluginy, aby odrážely aktuální stav a konfiguraci.

Příklad konfigurace jednoho pluginu se nachází ve výpisu 4.2.

[24] OpenGrok obsahuje třídu s metodou, která pro konkrétné URL rozhodne, jestli patří nějakému projektu nebo ne

4.7 Testování

4.7.1 Sezení klienta

Sezení klienta (session) je mechanismus, jak identifikovat uživatele během různých požadavků na webovou službu. HTTP nepodporuje hlídání stavu a právě sezení klienta řeší tento nedostatek. Data uživatele, která vývojář do sezení uložil, zůstávají na serveru, a uživatel pouze posílá na server identifikátor sezení „session id“ [14].

V prostředí servlet kontejneru existuje třída `HttpSession`[25], jež se dá získat z objektu požadavku a do objektu této třídy může vývojář uložit libovolná data a později je opět získat.

4.7.2 Jednotkové testy

Většina implementace autorizačního frameworku je pokryta jednotkovými testy, které testují různé scénáře konfigurace autorizačních pluginů:

- různé kombinace klíčových slov a výsledek autorizace,
- různé zanoření do 2. úrovně a výsledek autorizace.

Jednotkové testy také testují správnost načítání tříd ze souborů.

4.7.3 Testy výkonu

Protože přidání autorizační vrstvy zcela jistě navýší dobu odezvy pro koncového uživatele, je na místě ověřit, jakým způsobem se tato odezva zvýší, a jestli je závislá na zátěži systému. Klíčovými místy přitom jsou:

- hlavní stránka zobrazující všechny skupiny a projekty,
- vyhledávání přes podmnožinu projektů,
- zobrazování souboru uvnitř projektu (zdrojový kód, historie, anotace).

4.7.3.1 Testovací instance

Pro testování si připravíme modelovou instanci obsahující 30 projektů – 15 normálních projektů a 15 repositářů. Jména projektů obsahují vždy sudá čísla do 28 včetně, jména repositářů obsahují vždy lichá čísla do 29 včetně. Tyto projekty a repositáře rozdělíme do několika skupiny, využijeme přitom zanoření až do 3. úrovně. Jako zdrojový kód použijeme ve všech případech zdrojové kódy OpenGroku[26] ve verzi 1.0.

Projekty a repositáře rozdělíme následujícím způsobem:

[25] https://tomcat.apache.org/tomcat-8.0-doc/servletapi/javax/servlet/http/HttpSession.html
[26] https://github.com/OpenGrok/OpenGrok

- do dvou skupin podle typu projektu - „projekty“ a „repositáře“ a současně,
- do dvou skupin podle první číslice ve jméně (od 1 do 2 včetně),
  - každou takovou skupinu rozdělíme na dvě podskupiny podle parity čísla (sudá a lichá),
    - a každou takovou skupinu se dvojciferným názvem rozdělíme podle druhé číslice na menší nebo rovno 5 a větší než pět.

Celkem nám vznikne velké množství skupin propojených až do hierarchie 3. úrovně. Zároveň vytvoříme 5 projektů, které nebudou obsaženy v žádné skupině.

Konfiguraci skupin demonstruje následující výpis. Z důvodu zkrácení výpisu bylo ve výpisu nahrazeno:

- „group starting wi“ za „***“ a
- „ository|proje“ za „...“.

```
1 starting with 1 ~ "test-(re...ct)-1.*"
2   odd sub***th 1 ~ "test-(re...ct)-1[13579].*"
3     >5 odd sub***th 1 ~ "test-(re...ct)-1[79].*"
4     <=5 odd sub***th 1 ~ "test-(re...ct)-1[135].*"
5   even sub***th 1 ~ "test-(re...ct)-1[02468].*"
6     >5 even sub***th 1 ~ "test-(re...ct)-1[68].*"
7     <=5 even sub***th 1 ~ "test-(re...ct)-1[024].*"
8 starting with 2 ~ "test-(re...ct)-2.*"
9   odd sub***th 2 ~ "test-(re...ct)-2[13579].*"
10    >5 odd sub***th 2 ~ "test-(re...ct)-2[79].*"
11    <=5 odd sub***th 2 ~ "test-(re...ct)-2[135].*"
12  even sub***th 2 ~ "test-(re...ct)-2[02468].*"
13    >5 even sub***th 2 ~ "test-(re...ct)-2[68].*"
14    <=5 even sub***th 2 ~ "test-(re...ct)-2[024].*"
```

Celkem tedy ve zdrojovém adresáři existuje 35 podadresářů.

4.7.3.2 Testovací uživatelé

Naší instanci vyzkoušíme sérií požadavků od 10 smyšlených uživatelů. Každý takový uživatel provede dohromady 135 (resp. 535) požadavků na webovou aplikaci v následujícím složení:

- 100 (resp. 500) požadavků na zobrazení hlavní stránky,
- 10 požadavků na soubory v konkrétním projektu nebo repositáři,

- 25 požadavků na hledání mezi 10 vybranými projekty a repositáři.

V první variantě testu použijeme sekvenční zasílání požadavků (všichni uživatelé přišli postupně za sebou), ve druhé variantě testu použijeme paralelní zasílání požadavků (všichni uživatelé přišli téměř zároveň). Dále zkusíme porovnat výsledky pro větší množství požadavků (500) na výpis hlavní stránky, jejíž stáhnutí by mělo zabrat nejvíce času.

Každý uživatel používá sezení klienta (session).

4.7.3.3 Testovací autorizační pluginy

Testovací požadavky spustíme nejdříve bez autorizace – s prázdným autorizačním stackem (automaticky povoluje všechny příchozí požadavky). Následně pak sestavíme příkladnou konfiguraci v kombinaci s několika autorizačními pluginy, které zapojíme do procesu autorizace.

Aby byly oba výsledky porovnatelné, zařídíme, aby autorizace vždy dopadla úspěchem – autorizace projde vždy celým stackem, nicméně každý autorizační plugin může vykonávat nějakou práci navíc, aby simuloval opravdovou autorizaci předpokládaným zdržením. Použitou konfiguraci můžeme vidět na obrázku 4.8, konfigurace jednotlivých autorizačních pluginů uvádíme v závorkách za jménem autorizačního pluginu.

UserPlugin UserPlugin využívá instanci požadavku a dle nastavené HTTP hlavičky `X-Username` vytvoří objekt uživatele, který do požadavku vloží jako jeho atribut. Hlavička `X-Username` je nastavena pro každý požadavek a pro každého uživatele.

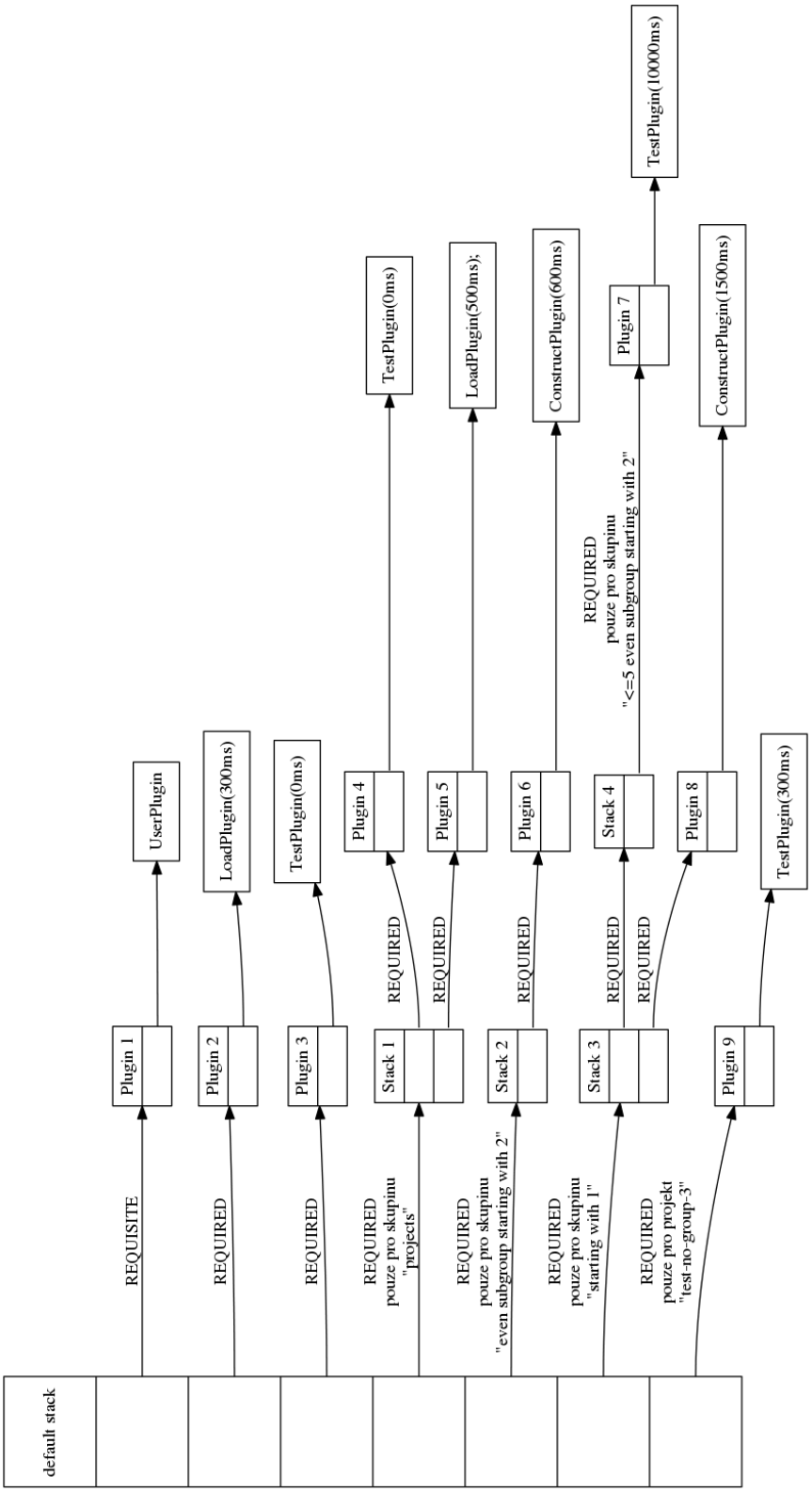
LoadPlugin LoadPlugin symbolizuje plugin, který má delší odezvu při inicializaci (při volání metody `load()`). Tato odezva se dá konfigurovat parametrem pluginu.

TestPlugin TestPlugin obsahuje zpozdění v metodách `isAllowed(...)`. Odezva se dá konfigurovat parametrem pluginu.

ConstructPlugin ContstructPlugin simuluje plugin, který využívá sezení klienta (session) a při prvním použití `isAllowed(...)` toto sezení inicializuje. V inicializaci je opět konfigurovatelné zpozdění.

Použité hodnoty u jednotlivých pluginech v bodech:

- Plugin 2 – `LoadPlugin` – 300ms,
- Plugin 5 – `LoadPlugin` – 500ms,
- Plugin 6 – `ConstructPlugin` – 600ms,



Obrázek 4.8: Testovací autorizační konfigurace

- Plugin 7 – `TestPlugin` – 10000ms (neměl by být nikdy nikdy zavolán, protože se vylučuje podskupina s nadskupinou),
- Plugin 8 – `ConstructPlugin` – 1500ms,
- Plugin 9 – `TestPlugin` – 300ms,
- ostatní autorizační pluginy – 0ms.

Shrnutí zpoždění Očekáváme dohromady $300 + 500 = 800$ms zpoždění při načítání autorizačních pluginů a jejich inicalizaci.

Dále při načítání hlavní stránky je nejnižší možná hodnota 300ms určená posledním autorizačním pluginem.

Maximální hodnota trvání autorizace by měla být 1500ms (tento parametr se v testu nesleduje), nicméně při více požadavcích a používání sezení klienta by tento výkyv neměl příliš ovlivnit celkové zpoždění.

4.7.3.4 Testovací stroj

Pro všechny testy byl použit stejný testovací stroj. Parametry stroje:

- Hardware
  - paměť – 7.5 GiB,
  - procesor – 2.3 GHz, 4 jádra,
  - disk – SSD, 207 GB,
  - operační systém – Ubuntu 16.04 LTS,
- Síť – lokální smyčka,
- Servlet kontejner – Tomat 8,
- Java – verze 8,
- Wget – verze 1.17.1,
- Apache Benchmark (AB) – verze 2.3.

Stroj není izolován od ostatních souběžně běžících aplikací.

Wget Pro přístup na webovou stránku z prostředí příkazové řádky v sekvenčním testu se používá unixový program Wget. Tento nástroj umožňuje ukládat sezení klienta (session) do souboru a opětovně ho načíst, proto je možné pro jednoho uživatele využívat stejné sezení. Nevýhodou tohoto nástroje pro rychlé testování je, že navštěvovanou stránku stáhne a fyzicky ukládá na disk, takže nároky na čas běhu celého testovacího skriptu jsou vyšší [15]. Nástroj byl použit ve verzi 1.17.1.

AB Nástroj pro testování kvality webových serverů umožnující klást na server mnoho požadavků v rychlém časovém sledu navíc s využitím více vláken zároveň, takže požadavky chodí téměř současně [16]. Nemá takové přívětivé rozhraní pro práci se sezením klienta jako má Wget, nicméně i zde se dá sezení nastavit pro každého uživatele. Tento nástroj je použit v paralelních testech pro:

- požadavky na zobrazení hlavní stránky a
- požadavky na hledání.

Použitá verze nástroje je 2.3.

4.7.3.5 Definice pojmů v testech

Typ testu je označen trojicí (např. S100N), kde:

- první písmeno značí paralelnost testu:
  - S – sériový test,
  - P – paralelní test;
- po písmenu následuje číslo:
  - 25 – na hlavní stránku bylo posláno 25 požadavků,
  - 100 – na hlavní stránku bylo posláno 100 požadavků,
  - 500 – na hlavní stránku bylo posláno 500 požadavků;
- a poslední písmeno označuje:
  - N – autorizace nebyla využita v tomto testu,
  - A – autorizace byla využita v tomto testu.

Dále pak tabulka tabulka obsahuje množství zkratek:

- # - počet požadavků v dané kategorii,
- ϕ - průměrná doba zpracování jednoho požadavku v ms,
- HS – zobrazení hlavní stránky,
- X – zobrazení stránky xref (zobrazení souboru v projektu),
- S – vyhledávání (dotaz byl full-text „foo“ – vyskytuje se ve všech projektech, zvoleno bylo 10 projektů – 4 projekty ve skupinách, 4 repositáře ve skupinách a 2 projekty bez skupiny).

4.7.3.6 Sériový test

Sériový test proběhl ve dvou variantách, výsledky obou variant zobrazují tabulky 4.1 a 4.2.

Typ	#	ϕ	# HS	ϕ HS	# X	ϕ X	# S	ϕ S
S100N	1350	11.5378	1000	11.554	100	3.28	250	14.776
S500N	1350	10.5074	1000	10.502	100	3.18	250	13.46
S500N	5350	11.5144	5000	11.4948	100	3.45	250	15.132
S500N	5350	11.463	5000	11.4722	100	3.24	250	14.568

Tabulka 4.1: Sériový test bez autorizace

Typ	#	ϕ	# HS	ϕ HS	# X	ϕ X	# S	ϕ S
S100A	1350	305.5259	1000	333.358	100	3.51	250	315.004
S100A	1350	305.5193	1000	333.412	100	3.34	250	314.82
S500A	5350	310.8376	5000	316.7396	100	3.55	250	315.712
S500A	5350	310.8179	5000	316.7126	100	3.48	250	315.86

Tabulka 4.2: Sériový test s autorizací

Shrnutí Z uvedených tabulek lze sledovat závislost průměrné doby zpracování požadavku, které je v případě zapnuté autorizace naprosto určeno nejpomalejším autorizačním pluginem provádějícím autorizační rozhodnutí. Tím je v tomto případě autorizační plugin (na obrázku 4.8 vnořený v „Plugin 9“) se zpožděním 300ms pro testování projektu „test-no-group-3“, který se v uvedených testech vyskytuje na hlavní stránce a také při vyhledávání.

Zároveň drahá konstrukce u autorizačního pluginu `ConstructPlugin` s odezvou 1500ms se, díky pomalejšímu „Plugin 9“, na průměrných časech téměř neprojevila.

Průměrná odezva na zobrazení náhledu zdrojového kódu (xref) vypadá podobně jako pro verzi bez autorizace, což se dalo předpokládat, protože se testuje pouze jeden konkrétní projekt.

4.7.3.7 Paralelní test

Paralelní test má dvě úrovně paralelizace:

1. každý uživatel je spuštěn paralelně (tedy všech 10 úživatelů téměř najednou) a
2. pro zobrazení hlavní stránky a vyhledávání se používá AB nástroj s maximálním počtem souběžných požadavků 3.

Po startu testovacího skriptu se tak snadno může stát, že ve stejnou chvíli přijde na server až 30 požadavků.

Pro zobrazení souboru v projektu (xref) se využívá nástroj Wget.

Výsledky paralelního testu popisují dvě tabulky 4.3 a 4.4.

Typ	#	ϕ	# HS	ϕ HS	# X	ϕ X	# S	ϕ S
P100N	1350	59.1896	1000	52.212	100	18.59	250	103.34
P100N	1350	54.2489	1000	51.69	100	12.86	250	81.04
P500N	5350	52.8135	5000	52.1298	100	19.32	250	79.884
P500N	5350	49.9746	5000	49.4188	100	15.59	250	74.844

Tabulka 4.3: Paralelní test bez autorizace

Typ	#	ϕ	# HS	ϕ HS	# X	ϕ X	# S	ϕ S
P100A	1350	310.668	1000	339.445	100	11.97	250	315.044
P100A	1350	311.306	1000	338.842	100	9.5	250	321.888
P500A	5350	312.608	5000	318.286	100	10.91	250	319.716
P500A	5350	313.474	5000	319.434	100	8.44	250	316.3

Tabulka 4.4: Paralelní test s autorizací

Shrnutí Z testu můžeme usoudit, že souběžné požadavky server určitě zpomalí, což se více projevilo v testu bez autorizace, kde došlo místy až k pětinásobnému nárůstu odezvy. V testu s autorizací se souběžné požadavky neprojeví díky prodlevě u nejpomalejšího autorizačního pluginu. Znatelný nárůst u testu s autorizací je pouze u zobrazení zdrojového kódu (xref).

Bohužel uvedené testy neprokázaly výrazné zpomalení odezvy webové aplikace kvůli autorizačnímu pluginu s odezvou 300ms, který výrazně ovlivnil všechny výsledky. Pokusme se tedy provést ještě jedno testování s upravenou odezvou u tohoto autorizačního pluginu na 0ms (na obrázku 4.8 vnořený v „Plugin 9“).

4.7.3.8 Sériový test s pozměněnými parametry

Byla změněna hodnota zpozdění na 0ms v autorizačním pluginu, který je vnořený do „Plugin 9“ na obrázku 4.8. Výsledky sériového testu popisují dvě tabulky 4.5 a 4.6.

Typ	#	ϕ	# HS	ϕ HS	# X	ϕ X	# S	ϕ S
S100N	1350	10.0222	1000	10.157	100	2.81	250	12.368
S100N	1350	10.5519	1000	10.656	100	3.02	250	13.148
S500N	5350	10.1045	5000	10.133	100	2.84	250	12.44
S500N	5350	9.7219	5000	9.759	100	2.97	250	11.68

Tabulka 4.5: Sériový test s pozměněnými parametry bez autorizace

Typ	#	ϕ	# HS	ϕ HS	# X	ϕ X	# S	ϕ S
S100A	1350	25.4963	1000	31.099	100	3.11	250	12.04
S100A	1350	26.5578	1000	32.193	100	2.93	250	13.468
S500A	5350	13.8996	5000	14.2012	100	2.87	250	12.28
S500A	5350	13.8794	5000	14.197	100	2.85	250	11.94

Tabulka 4.6: Sériový test s pozměněnými parametry s autorizací

4.7.3.9 Paralelní test s pozměněnými parametry

Výsledky paralelního testu popisují dvě tabulky 4.7 a 4.8.

Typ	#	ϕ	# HS	ϕ HS	# X	ϕ X	# S	ϕ S
P100N	1350	40.6444	1000	39.157	100	11.33	250	58.32
P100N	1350	38.9437	1000	38.183	100	9.16	250	53.9
P500N	5350	38.3077	5000	37.8548	100	10.31	250	58.564
P500N	5350	40.089	5000	39.5712	100	14.41	250	60.716

Tabulka 4.7: Paralelní test s pozměněnými parametry bez autorizace

Typ	#	ϕ	# HS	ϕ HS	# X	ϕ X	# S	ϕ S
P100A	1350	54.9741	1000	58.793	100	8.62	250	58.24
P100A	1350	54.2363	1000	58.035	100	7.7	250	57.656
P500A	5350	42.1568	5000	41.7286	100	9.71	250	63.7
P500A	5350	42.0892	5000	41.843	100	13.18	250	58.576

Tabulka 4.8: Paralelní test s pozměněnými parametry s autorizací

Shrnutí Výsledky těchto upravených testů ukazují, že testy s autorizací obsahující drahou konstrukci sezení klienta v autorizačních pluginech (dohromady činí součet `ConstructPlugin` 2100ms pro první požadavek klienta) se

postupně s přibývajícími požadavky blíží k času bez autorizace (42.0892ms proti 38.3077ms v testu P500A a P500N). Rozdíl v testu P100A a P100N můžeme ověřit jednoduchým výpočtem:

$$\frac{(1350 - 10) * 40.6444 + 10 * 2100}{1350} = 55,898885926$$

A to porovnáváme s číslem 54.9741, které je skoro stejné, takže dostáváme, že zpomalení je pouze důsledek pomalé konstrukce sezení hosta.

Procházení stacku je prakticky velmi rychlá operace, jediné, co autorizaci zpomaluje, je výkon jednotlivých autorizačních pluginů. Na základě testování můžeme vydat sérii doporučení pro jejich implementaci.

4.7.3.10 Závěr

Testování výkonu ukázalo doporučení, kterými by se měl vývojář řídit při navrhování nového autorizačního pluginu.

Inicializace pluginu Inicializace pluginu (metoda `load(...)`) může být výpočetně náročná a nijak to neovlivní probíhající autorizaci, proto je doporučeno provést veškeré akce nezávislé na uživateli a požadavku při inicializaci autorizačního pluginu.

Autorizační rozhodnutí Metody pro autorizační rozhodnutí (`isAllowed(...)`) by měly být nejméně výpočetně náročné, veliký rozdíl byl i pro malé zpoždění 300ms u jednoho autorizačního pluginu v celém stacku, který zpozdí načítání celé stránky.

Konstrukce sezení klienta Pro zrychlení výpočtu v metodách `isAllowed(...)` testy ukázaly, že je výhodné využívat sezení klienta a inicializovat toto sezení užitečnými hodnotami při prvním požadavku klienta. I v případě, že je konstrukce sezení klienta výpočetně drahou operací, z dlouhodobého hlediska je tento přístup průměrně rychlejší než provádět ekvivalentní operaci v každém volání `isAllowed(...)`. Více o sezení klienta je v sekci 4.7.1.

Kapitola 5

Autorizační plugin

Třetí a poslední částí této práce je dodat zadavateli sadu autorizačních pluginů, jež by umožnily autorizaci OpenGroku v zadavatelově produkčním prostředí.

5.1 Úvod

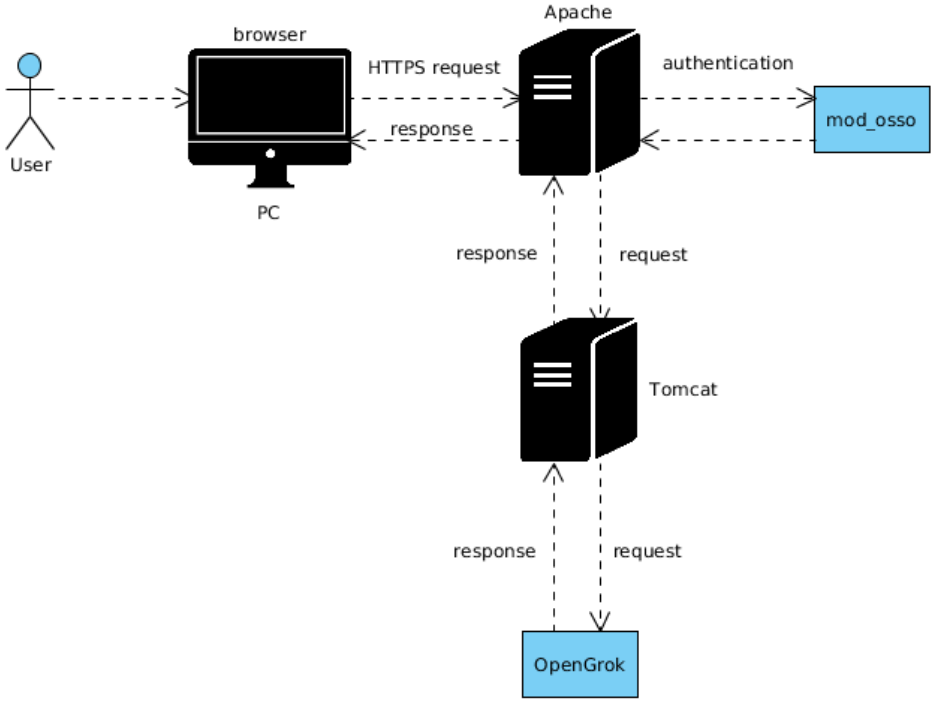
Abychom mohli navrhnout takové autorizační pluginy, musíme nejprve nahlédnout stručně na prostředí zadavatelovy instance a její konfiguraci. Zdrojové kódy jsou u zadavatele klasifikovány jako vysoce důvěrné informace a zadavateli nevyhovuje dosavadní implementace přístupových oprávnění k instanci.

5.1.1 Instance OpenGroku

Instance se nachází v interní síti zadavatele. Obdobně jako i na jiných službách zadavatele, je při přístupu vyžadována autentizace. Ta je pro všechny služby implementována přes koncept Single Sign-On (SSO), to znamená, že pro přístup do aplikace je nejdříve neautentizovaný uživatel přesměrován na externí službu, kde provede autentizaci, a tato služba ho následně vrátí aplikaci jako autentizovaného uživatele. Tento koncept umožňuje oddělit proces autentizace od funkčnosti aplikace a pro uživatele může být pohodlné, že pro přístup k interním službám využívá pouze jeden účet a jedno přihlášení v rámci časového intervalu [17].

U zadavatele je takto nakonfigurován server Apache[27], který slouží jako prvotní štít pro přístup k aplikaci. V Apachi je aktivován `mod_osso`, jenž se postará o komunikaci v rámci SSO. Apache požadavek úspěšně autentikovaného uživatele přepošle, spolu s obdrženými hlavičkami [18], na instanci Tomcatu[28], ve kterém běží instance webové aplikace OpenGroku a ta tento požadavek zpracuje. Tuto architekturu také přehledně znázorňuje obrázek 5.1.

[27] https://httpd.apache.org/docs/2.4/
[28] http://tomcat.apache.org/



Obrázek 5.1: Architektura prostředí zadavatele

V OpenGroku má tedy každý požadavek k dispozici HTTP hlavičky, které jasně identifikují uživatele v interní síti.

Server Apache má ještě nastavený modul `mod_authnz_ldap` a jeho pomocí se aktuálně provádí autorizace pro přístup k celé instanci OpenGroku. Z množiny uživatelů, kteří se autentizují pomocí SSO, se k instanci OpenGroku dostane pouze malá podmnožina nakonfigurovaná uvedeným modulem, přesto tito uživatelé mohou zobrazit všechny zdrojové kódy v instanci OpenGroku. Toto řešení není vyhovující, protože zadavatel požaduje, aby byla kontrola oprávnění prováděna pro specifické projekty (případně skupiny projektů) zvlášť.

5.1.2 Mohutnost instance

V době psaní této práce obsahuje zadavatelova instance přes 370 projektů (většina z nich obsahuje mnoho vnořených repositářů), které by zadavatel rád rozdělil do zhruba 30 disjunktních skupin. Pokročilé rozdělování projektů do skupin a použití vnořených skupin by rád zadavatel použil, až se osvědčí prvotní použití autorizace v jeho systému.

Z počátku by zadavatel měl zájem nasadit autorizaci na několik skupin dle schématu níže.

5.1.3 Kontrola oprávnění

Veškerá oprávnění, jež mají být uvažována pro autorizaci, lze vyčíst z interních LDAP serverů. Použité přístupy se liší pro různé skupiny, základní myšlenkou autorizace nějakého projektu či skupiny u zadavatele je:

1. najít identifikátor uživatele v připraveném souboru (whitelist) - pro autorizaci dostačující;

2. najít konkrétní informaci v LDAP serveru.

5.2 Analýza požadavků

Pro návrh vyhovujícího řešení je třeba prozkoumat požadavky na autorizaci – jaké informace použít k autorizaci, jak je v autorizačních pluginech interpretovat a jak vyhodnotit. Představa zadavatele o prvotní konfiguraci autorizace pro jeho instanci, založená na návrhu autorizační vrstvy, je popsána výpisem 5.1.

```
1  default stack:
2    # decode the user from OSSO headers
3    REQUISITE User
4
5    # required substack for group "Skupina␣1"
6    REQUIRED Skupina 1
7      SUFFICIENT Email Whitelist <mail1.txt>
8      REQUIRED Ldap group "group_name"
9
10   # required substack for group "Skupina␣2"
11   REQUIRED Skupina 2
12     REQUIRED Email Whitelist <mail2.txt>
13
14   # required substack for group "Skupina␣3"
15   REQUIRED Skupina 3
16     REQUIRED Place Whitelist <place.txt>
17
18   # required substack for group "Skupina␣4"
19   REQUIRED Skupina 4
20     SUFFICIENT Email Whitelist <mail3.txt>
21     REQUIRED Ou Whitelist <ou.txt>
```

Výpis 5.1: Představa zadavatele o konfiguraci autorizace

5.2.1 Příchozí požadavek

Příchozí požadavek obsahuje hlavičky dodané `mod_osso` ze serveru Apache a ty plně identifikují přihlášeného uživatele. Tyto hlavičky by bylo vhodné dekódovat a nějakým způsobem uložit pro další zpracování. Příklad hlaviček je naznačen v následujícím výpisu.

```
1 osso-user-dn: cn=some_user,dc=example,dc=com
2 osso-user-guid: 12345678901234567890QWERTZUIO
```

Bohužel dodané hlavičky neobsahují příliš lidsky čitelné formáty identifikace uživatele, takže je není možné použít přímo pro autorizaci (například v souborech), protože soubory s povolenými uživateli (whitelist) zpravidla tvoří lidé.

5.2.2 Emailová adresa

Dalším krokem bude vyčíst z LDAP serveru lidsky čitelný identifikátor uživatele, který se dá dále použít v autorizaci. Pro jednoduchost si můžeme tento identifikátor zvolit jako emailovou adresu daného uživatele a tu lze získat jedním dotazem na LDAP server při využití hlaviček z prvního kroku. Pro větší efektivitu při provádění jednoho dotazu se autorizační plugin rovnou zeptá na více obvyklých atributů, které si společně s identifikátorem uložíme.

5.2.3 Dodatečné informace o uživateli

Uvedený přístup v předchozí sekci bude stačit na pokrytí všech autorizací oproti souboru (whitelist) i na některé přímé atributy uživatele. Bohužel ale již nefunguje na případy, kdy bychom se chtěli ptát LDAP serveru obecnějším způsobem, například na příslušnost uživatele k nějaké skupině v LDAP. Proto bude nutné umožnit dotaz na obecně zadaný LDAP filtr. Zároveň konfigurace zadavatele vyžaduje, aby šlo do filtru dosadit některé již získané informace v průběhu požadavku (třeba emailovou adresu). Jako příklad si můžeme představit jednoduchý filtr:

```
1 (&(objectclass=*)(cn=group_name)(memberId=%email%))
```

Na místo sekvence `%email%` by se vložil již předem zjištěný email z předchozího bodu 5.2.2 a dotaz by se vyhodnotil. Pokud by filtr vrátil prázdnou množinu výsledků, autorizace bude neúspěšná, v opačném případě předpokládáme úspěch.

5.2.4 Funkční požadavky

Z analýzy dostáváme několik funkčních požadavků:

- Autorizační plugin musí umět získat a dekódovat důležité `mod_osso` hlavičky [18] [19].

- Autorizační plugin musí umět autorizovat pomocí lidsky čitelného identifikátoru uživatele oproti dodanému souboru (whitelist).
- Autorizační plugin musí umět autorizovat pomocí LDAP atributu oproti dodanému souboru (whitelist).
- Autorizační plugin musí umět autorizovat pomocí LDAP filtru a neprázdné návratové množiny hodnot.

5.3 Návrh řešení

Nabídka autorizační vrstvy:

1. specifikovat autorizační flagy;
2. zacílit stack (nebo plugin) na konkrétní skupinu, ostatní skupiny jsou nedotčeny;
3. využívat zanoření stacků do sebe;
4. ve stacku může existovat více instancí stejného pluginu s různými parametry.

Uvedených vlastností můžeme využít k návrhu autorizačního stacku, do kterého umístíme sadu nových pluginů. Tyto pluginy navrhneme tak, aby dohromady splňovaly funkční požadavky dle 5.2.4. Komunikace mezi pluginy (například předání již známých atributů uživatele) může probíhat přes sdílenou instanci požadavku.

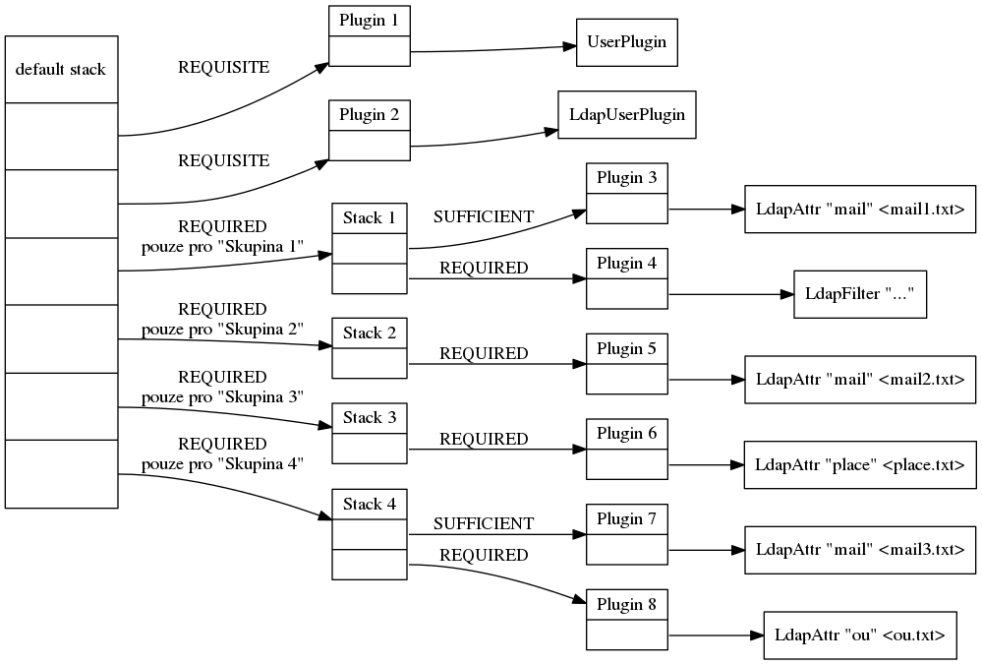
Z funkčních požadavků vyplývá použití kombinace těchto pluginů:

- UserPlugin – dekóduje hlavičky HTTP požadavku;
- LdapUserPlugin – provede dotaz na zjistění nejnutnějších LDAP atributů;
- LdapAttr – provede dotaz (dle potřeby) na dodatečný LDAP atribut a ověřuje jeho přítomnost v souboru (whitelist);
- LdapFilter – provede dotaz dle zadaného filtru a ověřuje mohutnost návratové množiny.

Návrh konfigurace těchto pluginů respektující zadavatelovu představu (ve výpisu 5.1) zobrazuje obrázek 5.2.

5.3.1 Sezení klienta

V kontextu autorizačních pluginů můžeme použít sezení klienta na uložení výsledků z LDAP dotazů, abychom zrychlili autorizaci v následujících požadavcích stejného uživatele. Více o sezení klienta je v sekci 4.7.1.



Obrázek 5.2: Návrh konfigurace autorizace

5.3.2 UserPlugin

Tento plugin bude označený autorizačním flagem „requisite“ a jeho úloha spočívá v načtení HTTP hlaviček z příchozího požadavku. Z hlaviček vyrobí novou instanci třídy `User`, kterou vloží jako atribut do požadavku, kde může být dostupný pro další autorizační pluginy ve stacku.

Pokud hlavičky nejsou přítomné nebo jejich získávání jiným způsobem selže, je návratová hodnota tohoto autorizačního pluginu neúspěch a průchod stackem se ihned ukončuje.

Využitelné hlavičky, dle dokumentace `mod_osso` [18] [19], jsou:

- `Osso-User-Guid` – globální unikátní identifikátor uživatele;
- `Osso-User-Dn` - Distinguished Name (DN) uživatele jako základ hledání v LDAP.

5.3.2.1 Entita User

Entita `User` slouží jako obálka pro získané hlavičky z požadavku (relevantní k `mod_osso`), nejpodstatnější hlavičky jsou uloženy jako přímé atributy této entity, ostatní hlavičky jsou uloženy v mapě atributů.

5.3.3 LDAP Autorizační pluginy

Všechny ostatní autorizační pluginy nějakým způsobem komunikují s LDAP serverem. Java poskytuje rozhraní, jak se připojit na LDAP server (nebo k jiným jmenným nebo adresářovým službám) [20]. Manipulace se spojením probíhá přes objekt kontextu `Context` vytvořený s parametry pro připojení na LDAP server, konkrétně metodou `lookup()`. Krátkou ukázku inicializace spojení v jazyce Java můžeme vidět v následujícím výpisu [21].

```
1 // Set up environment for creating initial context
2 Hashtable env = new Hashtable(11);
3 env.put(Context.INITIAL_CONTEXT_FACTORY,
4     "com.sun.jndi.ldap.LdapCtxFactory");
5 env.put(Context.PROVIDER_URL, "ldap://localhost:389/o
    =JNDITutorial");
6
7 // Specify timeout to be 5 seconds
8 env.put("com.sun.jndi.ldap.connect.timeout", "5000");
9
10 // Create initial context
11 DirContext ctx = new InitialDirContext(env);
12
13 // do something useful with ctx
14 ctx.lookup("ou=NewHires");
```

Výpis 5.2: Příklad používání LDAP spojení v Javě

Dotaz na LDAP spojení může být časově náročný, a proto bychom ho rádi vykonávali jen v nezbytně nutných případech. Pro uložení výsledku LDAP dotazu využijeme sezení klienta (session) a zabráníme tak zbytečným dotazům na LDAP server v momentě, kdy už informace, která nás zajímala, je dostupná v sezení klienta.

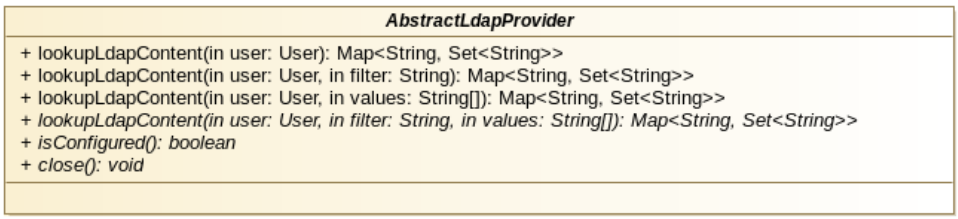
Pochopitelně v případě, kdy sezení klienta ještě neexistuje (klient se poprvé připojil k webové aplikaci), je nutné dotaz provést a výsledek uložit.

5.3.3.1 Cachování výsledků dotazů

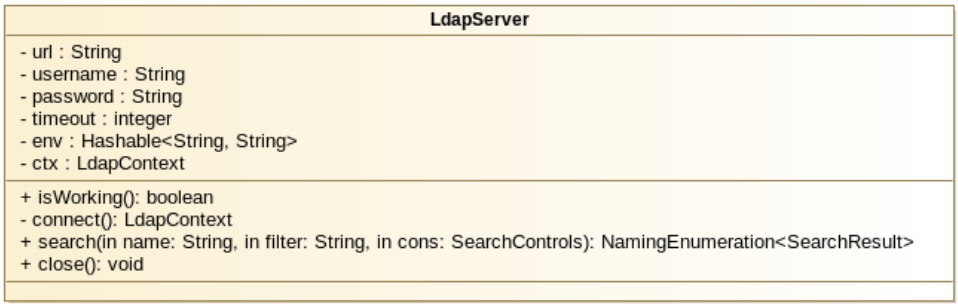
Další variantou zrychlení dotazů na LDAP servery je implementovat cache na úrovni dotazů, tedy přidat vrstvu mezi LDAP autorizační pluginy a LDAP spojení pro všechny uživatele.

Tato strategie je úspěšně využívána na serveru Apache, pokud je konfigurován, aby používal pro autorizaci LDAP spojení (více o server Apache v sekci 4.4.1).

Tato možnost by měla být předmětem budoucí optimalizace autorizační vrstvy.



Obrázek 5.3: Návrh abstraktní třídy AbstractLdapProvider



Obrázek 5.4: Návrh rozhraní LdapServeru

5.3.3.2 Rozhraní pro LDAP operace

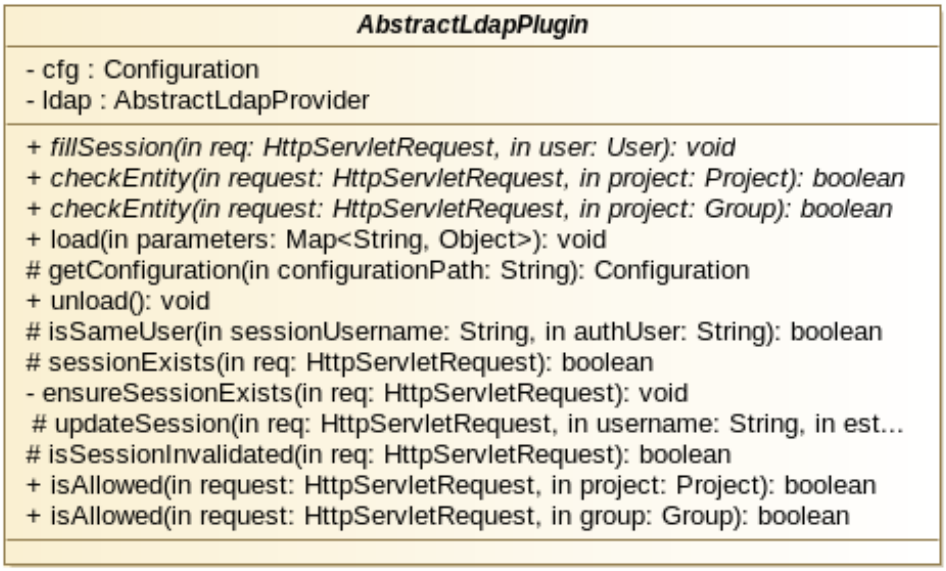
Pro pohodlný vývoj všech LDAP autorizačních pluginů navrhneme rozhraní pro práci s LDAP spojením.

AbstractLdapProvider Obálkou pro operace s LDAP servery bude abstraktní třída `AbstractLdapProvider` poskytující zejména metodu `lookupLdapContent(user, ...)` na získání dat ze spojení. Tato metoda využívá entitu uživatele `User` (popis v sekci 5.3.2.1) vytvořenou v `UserPlugin` již dříve ve stejném požadavku. Návrh této abstraktní třídy můžeme pozorovat na obrázku 5.3.

LdapServer Zadavatel disponuje více LDAP servery, proto bude rozumné reprezentovat server v autorizačním pluginu pomocí vlastní instance. Počet a různé parametry těchto serverů půjdou konfigurovat a zadavatel tak získá možnost, jak se zotavovat z případných výpadků, kdy právě používaný LDAP server přestane dočasně odpovídat.

`LdapServer` bude držet instanci kontextu `Context` pro svoje LDAP spojení; odtud se bude volat metoda `lookup()` zmíněná v sekci 5.3.3.

Návrh rozhraní LdapServeru je na obrázku 5.4.



Obrázek 5.5: Návrh třídy AbstractLdapPlugin

5.3.3.3 AbstractLdapPlugin

Všechny LDAP autorizační pluginy uvažované v sekci 5.3 mají společných několik vlastností:

- Vyžadují konfiguraci LDAP serverů.
- Vyžadují instanci `AbstractLdapProvider` na získávání dat.
- Musí inicializovat sezení klienta, pokud ještě neexistuje.

Bude proto výhodnější navrhnout abstraktní třídu obecného LDAP autorizačního pluginu, který ponechá na podtřídách pouze nezbytnou implementaci. Návrh takové abstraktní třídy je na obrázku 5.5.

Metoda `fillSession()` Tato metoda má dodat do sezení klienta informace, které mohou být později použity k autorizaci, v případě, že sezení hosta ještě neexistuje.

Metoda `checkEntity()` Tato metoda má stejné chování jako `isAllowed()`, tedy rozhodnout o autorizaci pro daný projekt nebo skupinu.

5.3.3.4 LdapUserPlugin

Tento autorizační plugin na základě entity `User`[29] vyplní do sezení klienta (session) novou entitu `LdapUser` obsahující nejpodstatnější informace o klientovi z dotazu na LDAP server.

Entita LdapUser Entita `LdapUser` slouží jako obálka pro získané informace z LDAP spojení. Tyto informace zejména jsou:

- emailová adresa,
- množina organizací klienta, jejichž je členem,
- unixové přihlašovací jméno.

Dále pak může obsahovat libovolné další informace uložené v mapě atributů.

5.3.3.5 LdapAttrPlugin

Tento autorizační plugin získá o uživateli libovolný LDAP atribut a kontroluje jeho přítomnost v dodaném souboru s povolenou množinou hodnot tohoto atributu (whitelist). Zároveň tento atribut uloží do entity `LdapUser`.

Pokud již daný atribut byl v minulosti získán (entita `LdapUser` ho obsahuje), tento plugin neprovede dotaz navíc a použije již dříve zjištěnou hodnotu.

Veškeré hodnoty vrácené LDAP serverem jsou interpretovány jako množina (může existovat více mapování pro stejný atribut), při kontrole přítomnosti atributu v souboru tak stačí, aby soubor obsahoval alespoň jednu hodnotu atributu a autorizace bude úspěšná.

5.3.3.6 LdapFilterPlugin

Tento autorizační plugin položí LDAP serveru dotaz pomocí obecného filtru a výsledkem autorizace bude úspěch, pokud existuje alespoň jeden záznam vyhovující danému filtru.

Navíc je možné ve vstupním filtru nahradit některé sekvence již získanými hodnotami, takové sekvence jsou:

- %mail% - nahradí emailovou adresou z LDAP serveru;
- %uid% - nahradí unixovým přihlašovacím jménem z LDAP serveru;
- %guid% - nahradí hodnotou hlavičky `Osso-User-Guid`;
- %username% - nahradí hodnotou hlavičky `Osso-User-Dn`;

[29] viz 5.3.2.1

- %atribut% - nahradí libovolný atribut dle jména „atribut“, pokud entita `LdapUser` takový atribut již obsahuje a zároveň mohutnost množiny odpovídající tomuto atributu rovná právě 1 (jinými slovy, pokud je nahrazení právě jednoznačné).

5.3.3.7 Shrnutí cachování pro LDAP autorizační pluginy

Spolu s autorizační vrstvou dochází 4 krát ke cachování autorizačních rozhodnutí. Tři cachování probíhají na úrovni požadavku:

1. cachování rozhodnutí pro skupinu nebo projekt v autorizačním frameworku (viz 4.5.6),
2. cachování filtrovaných skupin a projektů v třídě `ProjectHelper` (viz 4.6.5),
3. cachování entity uživatele `User` získaného dekódováním HTTP hlaviček.

Čtvrté cachování je na úrovni sezení klienta:

1. informace vedoucí k časově efektivnímu rozhodnutí o autorizaci (např. příznak úspěch/neúspěch).

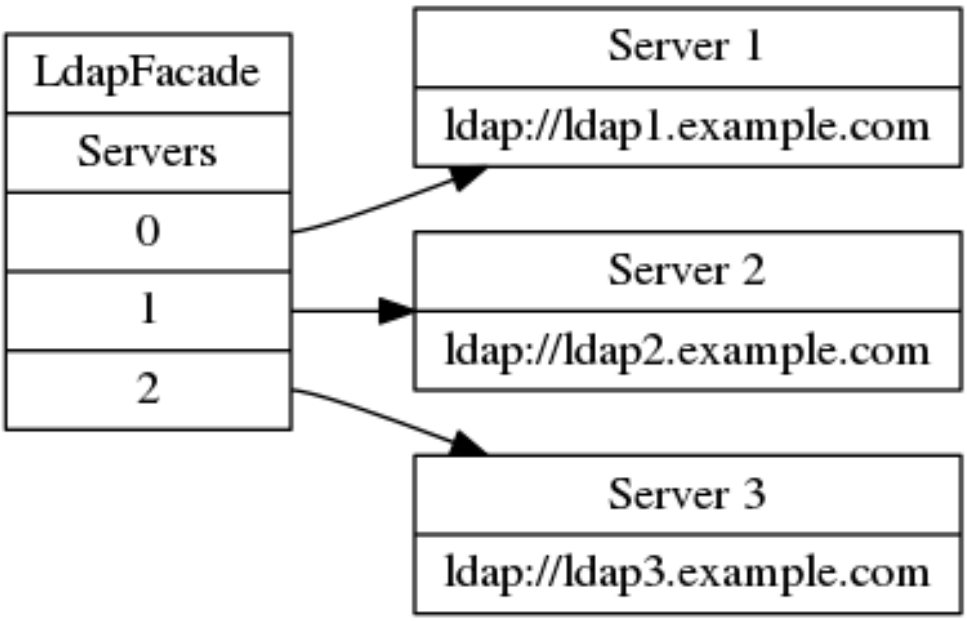
5.4 Implementace

Zdrojové kódy k autorizačním pluginům pro zadavatele jsou umístěny na přiloženém médiu k této práci, obsah média je popsán v příloze C.

5.4.1 Využívání session

Využití sezení klienta je v LDAP autorizačních pluginech naprosto klíčové, protože dotaz na LDAP server je nejpomalejší částí každého autorizačního pluginu. Při práci se sezením klienta je podstatné uchovávat a kontrolovat následující informace:

- informace o tom, jestli bylo sezení klienta správně inicializováno (pokud ne, tak není možné korektně provést autorizaci);
- informace o aktuální generaci sezení (vhodné pro invalidaci sezení v případě, že autorizační framework provedl reload všech autorizačních pluginů);
- informace o uživateli, kterému sezení patří (je potřeba přizpůsobit se situaci, kdy se uživatel změnil – například se odhlásil a přihlásil se někdo jiný);
- další informace specifické pro konkrétní plugin, které mohou usnadnit autorizaci.



Obrázek 5.6: Návrh vztahu `LdapFacade` a `LdapServer`

5.4.2 LDAP spojení

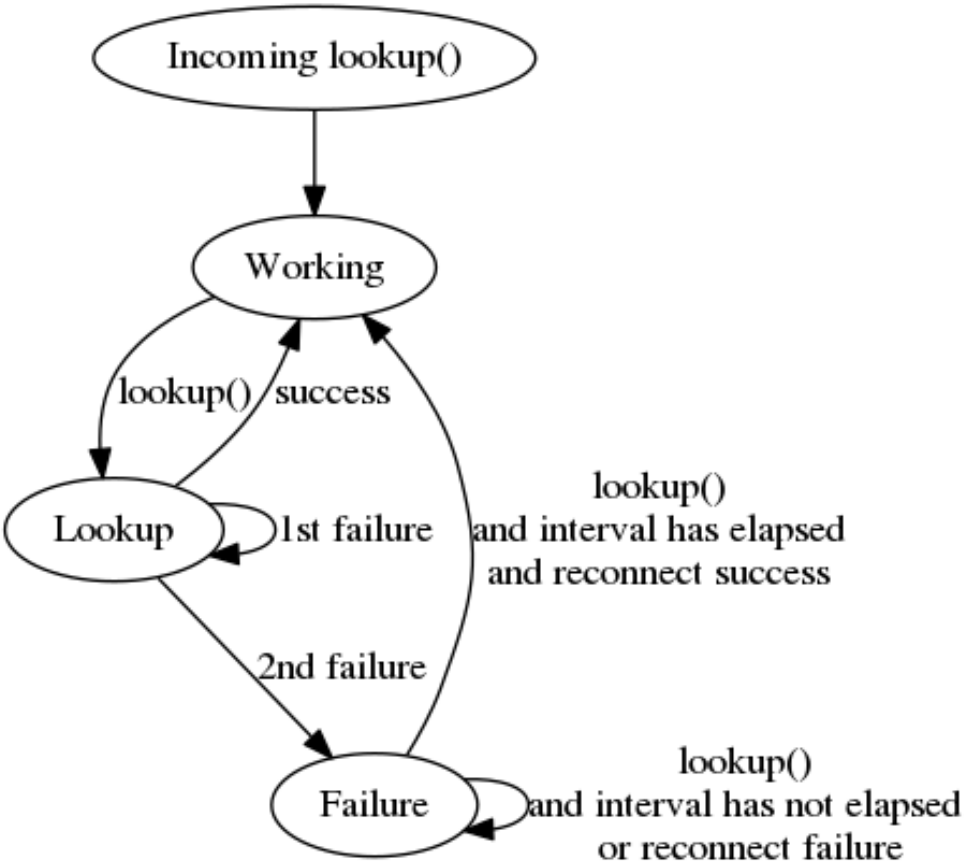
LDAP spojení realizuje v autorizačních pluginech dvojice tříd `LdapFacade` implementující abstraktní třídu `AbstractLdapProvider` a třída server `LdapServer`. Jejich hierarchie je znázorněna na obrázku 5.6.

`LdapFacade` požaduje konfiguraci dostupných LDAP serverů. Tato konfigurace je parametrem všech LDAP autorizačních pluginů a, aby nedocházelo k duplikaci spojení na stejný server, je i sdílená mezi všemi instancemi LDAP autorizačních pluginů. Konfigurace se zapisuje ve formátu XML tak, jak je v OpenGroku zvykem u hlavní konfigurace Indexeru.

`LdapFacade` při inicializaci nalezne první funkční server[30] a ten používá pro následující dotazy.

`LdapServer` implementuje jednoduchý mechanizmus zotavování se z chybového stavu. V případě, že funkční server má problémy s komunikací (vyhodí při hledání výjimku), `LdapServer` zkusí dotaz ještě jednou, ale pokusí se inicializovat připojení na server znovu. Pokud selže inicializace nového připojení, bude `LdapServer` po konfigurovatelný časový interval vždy vracet prázdnou množinu výsledků. Tímto se zabraňuje přehlcení serveru dotazy. Když server úspěšně obnoví spojení, ale ani nové spojení neskončí na stejném dotazu bez chyby, propaguje se chyba do vyšší úrovně k `LdapFacade`. Tento algoritmus s jednotlivými vztahy naznačuje obrázek 5.7.

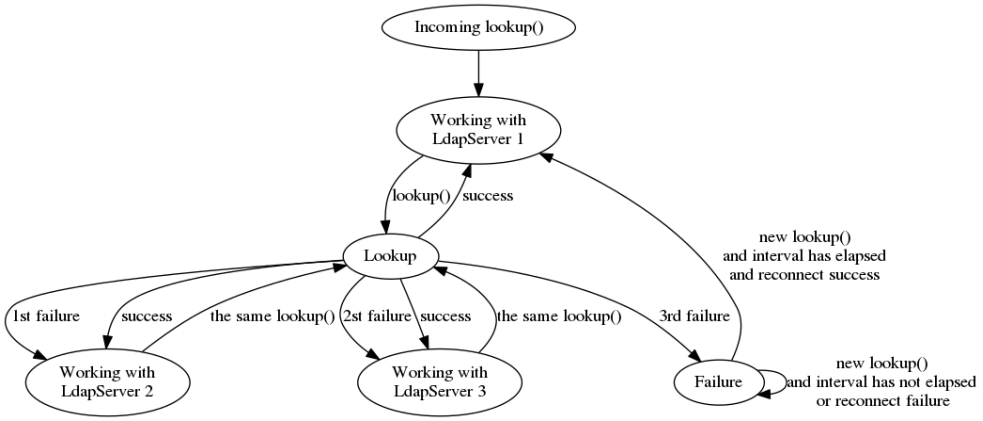
[30] první server, na který se lze bezchybně připojit



Obrázek 5.7: Algoritmus zpracování chyby při požadavku na `LdapServer`

`LdapFacade` se využívá vždy aktuálně funkční `LdapServer`. Pokud se projeví chyba v `LdapServeru`, `LdapFacade` v kruhu zkouší další funkční `LdapServer`, jestli nezpracuje stejný dotaz. Pokud vyzkouší všechny dostupné `LdapServery` a ani jednomu se nepodaří zpracovat dotaz bez chyby, přepne se, podobně jako `LdapServer`, do chybového stavu, kdy po konfigurovatelný časový interval nezpracovává žádné požadavky. Opět s cílem zbytečně nezahlcovat pravděpodobně nefunkční LDAP servery a nezdržovat autorizaci čekáním na neúspěšné pokusy kontaktovat některý ze serverů. Po uplynutí časového intervalu `LdapFacade` zkouší opět nalézt první funkční `LdapServer` a zpracovat nový dotaz.

Tento algoritmus je naznačen sérií interakcí na obrázku 5.8 pro instanci `LdapFacade` se třemi `LdapServery`. Na obrázku předpokládáme, že všechny `LdapServery` jsou na začátku funkční a požadavek zpracovává první `LdapServer`. Při vzniklé chybě se zpracování požadavku přesouvá vždy na další funkční `LdapServer` a případný úspěch ihned vrací výsledky volajícímu. Pokud projdeme všechny `LdapServery` a ani jeden není schopen dokončit po-



Obrázek 5.8: Algoritmus zpracování chyby při požadavku na `LdapFacade`

žadavek, přepne se `LdapFacade` do chybového stavu, jak je popsáno výše.

Všechny detekované chyby při posílání dotazu na LDAP server jsou propagovány skrz `LdapServer` až do `LdapFacade`, která všechny problémy zapisuje do logu.

5.5 Testování

5.5.1 Akceptační testy v produkčním prostředí

5.5.1.1 Abstraktní členění projektů do skupin

V produkčním prostředí zadavatele bylo vytvořeno 37 skupin první úrovně a zadavatel regulárními výrazy rozdělil většinu svých projektů a repositářů do těchto skupin. Skupiny prakticky kopírují vývojové týmy jednotlivých produktů, jejichž kód je uložený v instanci zadavatele. Ve webové aplikaci OpenGroku v produkčním prostředí bylo ověřeno, že skupiny tvoří hierarchický výpis na hlavní stránce OpenGroku.

Funkční požadavky

- možnost zanoření skupin minimálně do třetí úrovně – *splněno (zadavatel tento bod aktuálně nevyužil, testováno pouze na modelové instanci)*;
- zobrazení na hlavní stránce v hierarchickém stylu – *splněno (viz sekce 3.6.3 a obrázek 3.4)*;
- skupina musí rozlišovat projekty a repositáře – *splněno (viz sekce 3.5.2 a 3.6.2)*;
- snadná konfigurace příslušnosti projektu k dané skupině – *splněno (viz sekce 3.5.1)*;

- projekt může být ve více skupinách zároveň – *splněno (zadavatel tento bod aktuálně nevyužil, testováno pouze na modelové instanci; implementace viz sekce 3.6.2).*

Nefunkční požadavky

- Vyhnutí se další závislosti na knihovně třetí strany, pokud to není nezbytně nutné – *splněno (implementováno přímo v OpenGroku).*

5.5.1.2 Autorizační framework

Většina zdrojového kódu autorizační vrstvy je pokryta jednotkovými testy. V produkčním prostředí byla funkcionalita autorizační vrstvy testována empiricky (zkouška přístupových práv různých uživatelů) a testem zátěže.

Funkční požadavky

- Rozhodnutí, jestli má daný HTTP požadavek přístup ke projektu – *splněno (implementováno podle návrhu na obrázku 4.7).*
- Rozhodnutí, jestli má daný HTTP požadavek přístup ke skupině – *splněno (implementováno podle návrhu na obrázku 4.7).*
- Umožnit správci instance libovolně si zvolit metodu autorizace – *splněno (autorizační pluginy vyvíjí správce).*
- Konfigurace by pro jednoduché nastavení měla být adekvátně jednoduchá – *splněno.*
  - *Při použití jednoho autorizačního pluginu není konfigurace vůbec nutná.*
  - *Možnost cílit autorizaci pouze na konkrétní skupiny nebo projekty viz sekce 4.5.7.5.*

Nefunkční požadavky

- Snížit výpočetní složitost autorizačních operací na minimum – *dodrženo.*
  - *Implementace si uchovává autorizační rozhodnutí v instanci požadavku pro každý projekt a skupinu viz sekce 4.5.6.*
  - *Implementace si uchovává vyfiltrované množiny skupin a projektů v instanci požadavku viz sekce 4.6.5.*

5.5.1.3 Autorizační plugin

Funkcionalita autorizačních pluginů v kombinaci s konfigurací autorizace v produkčním prostředí byla testována empiricky na různých uživatelích. Dále pak byly provedeny testy zátěže autorizační vrstvy v produkčním prostředí v ostrém běhu, tyto testy jsou více popsány v následující sekci 5.5.2.

Funkční požadavky

- Autorizační plugin musí umět získat a dekódovat důležité `mod_osso` hlavičky – *splněno viz sekce 5.3.2.*
- Autorizační plugin musí umět autorizovat pomocí lidsky čitelného identifikátoru uživatele oproti dodanému souboru (whitelist) – *splněno viz sekce 5.3.3.4.*
- Autorizační plugin musí umět autorizovat pomocí LDAP atributu oproti dodanému souboru (whitelist) – *splněno viz sekce 5.3.3.5.*
- Autorizační plugin musí umět autorizovat pomocí LDAP filtru a neprázdné návratové množiny hodnot – *splněno viz sekce 5.3.3.6.*

5.5.2 Testy výkonu

Testy výkonu probíhaly za ostrého provozu zadavatelovy instance OpenGroku a smyslem bylo zjistit, jestli nemá instance, po prvotním nasazení autorizační vrstvy, výrazné potíže se zpomalením.

Testy proběhly pod jedním konkrétním uživatelem webové aplikace, který ztratil přidáním autorizační vrstvy přístup k několika skupinám a projektům. Sloupec „U“ v tabulkách s výsledky testů označuje, kolik různých sezení tohoto konkrétního uživatele bylo vytvořeno při běhu daného testu.

5.5.2.1 Testy odezvy

Zkratky použité v tabulce 5.1 jsou vysvětlené v sekci 4.7.3.5.

U	Typ	#	ϕ	# HS	ϕ HS	# X	ϕ X	# S	ϕ S
1	P100A	169	177.40	104	93.93	17	28.82	47	419.51
10	P100A	720	78.89	267	102.97	141	39.14	305	69.21
10	P25A	575	118.85	251	211.65	72	3.76	252	59.30
10	P25A	576	116.24	250	204.25	72	3.16	254	61.67
10	P25A	661	77.11	258	94.84	107	65.90	295	65.66
10	P25A	591	107.21	250	196.31	85	6.09	256	53.77
10	P25A	843	132.64	500	196.70	90	3.32	253	52.05
10	P25A	573	113.04	250	198.02	71	2.42	252	59.91
3	P100A	405	50.86	302	55.26	24	8.79	79	46.79
3	P500A	1613	44.05	1501	44.63	32	26.46	78	36.93

Tabulka 5.1: Testy odezvy v produkčním prostředí

Tyto testy zahrnují i skutečné uživatele, kteří webovou aplikaci využívali v době běhu těchto testů. Průměrnou dobu zpracování požadavku na načtení hlavní stránky, jež se pohybuje okolo 200ms, považujeme za solidní výsledek vzhledem k mohutnosti celé instance.

Bohužel správce zadavatelovy instance nesbíral statistiky provozu před nasazením autorizační vrstvy, takže nemáme tyto výsledky s čím porovnat.

5.5.2.2 Testy autorizační vrstvy

Uvedené testy také mohou doložit i jiné aspekty chování autorizační vrstvy v produkčním prostředí zadavatele.

Počet autorizací během požadavků Tabulka 5.2 ukazuje počty autorizačních rozhodnutí během stejný požadavků jako v předchozím testu. Zároveň také zobrazuje opětovné využití již získaných autorizačních rozhodnutí pro projekt nebo skupinu (dle implementace v 4.5.6). Tabulka využívá tyto zkratky:

- # A – celkový počet volání `isAllowed(...)` na autorizačním frameworku;
- # AH – celkový počet „cache hits“ - použití již vypočítaných hodnot;
- # AM – celkový počet „cache miss“ - volání `isAllowed(...)` na autorizačním stacku;
- # AN – celkový počet negativních autorizačních rozhodnutí;
- # AP – celkový počet pozitivních autorizačních rozhodnutí.

U	Typ	#	# A	# AH	# AM	# AN	# AP
1	P100A	169	276363	217911	58452	821	57631
10	P100A	720	989746	741626	248120	3523	244597
10	P25A	575	652551	467387	185164	2519	182645
10	P25A	576	654161	468635	185526	2526	183000
10	P25A	661	884191	668785	215406	3052	212354
10	P25A	591	665949	478576	187373	2553	184820
10	P25A	843	1041051	763513	277538	3778	273760
10	P25A	573	665771	480602	185169	2517	182652
3	P100A	405	540502	399579	140923	1930	138993
3	P500A	1613	2254539	1672727	581812	7915	573897

Tabulka 5.2: Testy využití autorizační vrstvy v produkčním prostředí

Z tabulky přehledně vyplývá, kolik práce ušetří uchovávání již vypočtených hodnot při autorizaci.

Časově nejhorší případy při provozu Zajímavým ukazatelem také mohou být hodnoty maximální doby zpracování jednoho požadavku. Tyto hodnoty na sobě nejsou nezávislé, záleží jaké maximální hodnoty zaznamenala zadavatelova instance během celého provozu s autorizací. Tyto statistiky zobrazuje tabulka 5.3.

Tabulka využívá tyto zkratky:

- T – maximální doba zpracování jednoho požadavku v ms;
- T HS – maximální doba zpracování jednoho požadavku na hlavní stránku v ms;
- T A – maximální doba zpracování autorizace pro jeden požadavek v ms;
- T AN – maximální doba zpracování autorizace pro jeden požadavek s negativním výsledkem v ms;
- T AP – maximální doba zpracování autorizace pro jeden požadavek s pozitivním výsledkem v ms.

U	Typ	#	T	T HS	T A	T AN	T AP
1	P100A	169	730022	1737	1547	807	1547
10	P100A	720	730022	1737	1547	807	1547
10	P25A	575	730022	4962	2766	807	2766
10	P25A	576	730022	4962	2766	807	2766
10	P25A	661	730022	4962	2766	807	2766
10	P25A	591	730022	4962	2766	807	2766
10	P25A	843	730022	4962	2766	807	2766
10	P25A	573	730022	4962	2766	807	2766
3	P100A	405	730022	4962	2766	807	2766
3	P500A	1613	730022	4962	2766	807	2766

Tabulka 5.3: Krajní případy v produkčním prostředí

Maximální doba zpracování jednoho požadavku je dlouhých 730s, nicméně z tabulky také vyplývá, že to není chyba autorizační vrstvy, jelikož maximální čas strávený autorizací je 2766ms. Při detailním pohledu do statistik zadavatele patří těchto 730s čtení historie souboru uvnitř repositáře, to může být časově náročná operace.

Načtení hlavní stránky OpenGroku u zadavatele trvalo maximálně 5s, to je opět při mohutnosti zadavatelovy instance přívětivý výsledek pro tento krajní případ.

Kapitola 6

Zhodnocení bezpečnostních rizik

6.1 Timing attack

Při návrhu autorizační vrstvy nebylo prioritou, aby byla imunní proti „timing-attack“ časovému útoku [22]. Použití „requisite“ nebo „sufficient“ autorizačního flagu způsobí okamžité ukončení procházení autorizačního stacku a předčasný návrat. Obdobně cílení stacků nebo pluginů na konkrétní skupiny nebo projekty vede k rozdílným časům při vyhodnocování autorizace pro různé skupiny nebo projekty.

Dalším cílem časového útoku může být ten fakt, že do webové aplikace se nahrávají cizí autorizační pluginy a autorizační vrstva neklade žádné předpoklady na délku trvání volání operací `load()` nebo `isAllowed()`.

6.2 Chybné autorizační pluginy

Správná a rychlá funkčnost autorizačních pluginů je předpoklad pro korektní autorizaci v OpenGroku.

6.2.1 Výjimky

Autorizační vrstva zachytává obecný typ `Throwable` při manipulaci s autorizačním pluginem. Chybně napsaný autorizační plugin tedy nepoškodí webovou aplikaci OpenGroku tím, že neočekávaně vyhodí výjimku. Zachytávání všech výjimek se dá využít pokud autorizační plugin potřebuje signalizovat chybu (např. chybějící konfigurace, neznámý stav). OpenGrok všechny tyto chyby z autorizačních pluginů vypisuje do logovacího souboru. Chyby v autorizačních pluginech OpenGrok interpretuje jako negativní autorizační rozhodnutí a postupuje dle autorizačního flagu u daného autorizačního pluginu.

6.2.2 Zaseknutí autorizačního pluginu

OpenGrok nijak nekontroluje čas přidělený provádění autorizačního pluginu, pokud tedy autorizační plugin se ve svých strukturách zasekne (deadlock, dlouhé čekání na nějaký zdroj, pomalá operace na síti) způsobí to zaseknutí vlákna, které obsluhuje daný požadavek. Když se autorizační plugin takto chová pokaždé, může dojít až k odepření služby. Nemělo by se však stát, že by došlo k úniku chráněné informace.

6.3 Podvržený autorizační plugin

Adresář s pluginy musí být adekvátně chráněn přístupovými právy, aby nemohlo dojít k tomu, že neoprávněná osoba přidá do tohoto adresáře libovolný autorizační plugin. Autorizační plugin se může dostat k citlivým informacím v OpenGroku jako jsou nevyfiltrované projekty nebo nevyfiltrované skupiny a může také používat některé třídy z OpenGroku. Bezpečnostní model je také diskutován v 4.1.

Také s tímto souvisí ochrana lokálního spojení na webovou aplikaci, která vždy poslouchá na TCP portu pro novou příchozí konfiguraci. Pokud by se útočníkovi podařilo proniknout do této komunikace a zaslat webové aplikaci svojí konfiguraci, může tak změnit adresář s pluginy a spoustu jiných nastavení.

6.4 Neúmyslně přesunutý autorizační plugin

Pokud správce instance neúmyslně přesune autorizační plugin z adresáře s pluginy, může to znamenat ztrátu autorizace v OpenGroku (žádný plugin, stack je prázdný, autorizace je vždy úspěšná). Pro tento případ se doporučuje vždy uvést konfiguraci stacku v konfiguraci pro čtení, v takovém případě OpenGrok vyžaduje nakonfigurovanou třídu autorizačního pluginu a autorizace je vždy negativní, když třída nebyla nahrána do paměti. Interpretace negativního výsledku pochopitelně záleží na autorizačním flagu.

6.5 HTTP stavové kódy

Autorizační filtr pro OpenGrok při neúspěšné autorizaci vrací stavový HTTP kód 403 (Forbidden). Útočník by tedy mohl pomocí hrubé síly odhadnout existující a neexistující projekty v instanci OpenGroku (404 (Not Found) proti 403 (Forbidden)). Zde se v návrhu dala přednost sémantice problému a vrací se kód indikující existující soubor a nedostatek oprávnění.

6.6 Detekce útoku hrubou silou

OpenGrok obsahuje mechanismus pro měření statistik webové aplikace jako jsou počet požadavků, průměrná doba požadavku, maximální doba požadavku a minimální doba požadavku v různých kategoriích. Tyto statistiky byly použité pro testování autorizační vrstvy a pro testování autorizačních pluginů v produkčním prostředí.

Tyto statistiky mohou pomoci odhalit slabá místa a také umožňují zpětně detekovat útok hrubou silou, který je charakteristický velikým množstvím požadavků v krátkém časovém sledu, a sledovat tak chování útočníka ve webové aplikaci. Okamžitý automatizovaný zásah při podezřelém chování indikující útok hrubou silou v OpenGroku není implementován.

Závěr

Cíl práce bylo navržení autorizační vrstvy a její implementace pro vyhledávač OpenGrok a zároveň implementace autorizace pro instanci zadavatele. Pro splnění bylo nutné rozdělit tuto práci do tří částí.

První částí je možnost rozdělit zdrojové kódy v OpenGroku (projekty) do skupin dle libovolné konfigurace. Důvodem tohoto rozdělení byla samozřejmě možnost jednodušší konfigurace autorizace pro skupiny projektů, nicméně důsledek rozdělení přímo ovlivňuje uživatele ve webové aplikaci. Hlavní stránka nabízející seznam projektů, stejně tak jako formulářová kolonka pro výběr projektu, je strukturovaná podle skupin v OpenGroku poskytující snazší navigaci ve větších instancích OpenGroku s mnoha projekty.

Dalším rozšířením, který na seskupení navazoval, je zobrazování důležitých zpráv u projektů (např. probíhající běh Indexeru, signalizace chybového stavu, ...). Díky implementaci skupin může správce instance posílat tyto zprávy skupinám projektů a ty jsou pak zobrazeny u všech dotčených projektů.

Autorizační vrstva je druhá část práce implementující autorizaci pro jednotlivé projekty nebo skupiny projektů v OpenGroku. Inspirace PAM Frameworkem byla v tuto chvíli klíčová, protože výsledkem je mocný nástroj pro kontrolu práv uživatele spolu s cílením na skupiny a projekty. Nahrávání autorizačních pluginů za běhu aplikace umožňuje správci instance připravit libovolnou metodu autorizace pro svojí instanci.

Závěrečnou částí této práce je implementace skupiny autorizačních pluginů, které řeší autorizaci uživatelů zadavatelovy instance oproti jeho interním LDAP serverům. Autorizační pluginy jsou navrženy obecně a lze je různými způsoby skládat do autorizačních stacků a pluginů, aby se docílilo požadované autorizace v zadavatelově instanci, kde se požadavky na autorizaci se v instanci zadavatele stále mění.

Příležitosti na vylepšení navržené vrstvy se nachází zejména v optimalizaci. Sběrem statistik na serveru zadavatele se dají v dlouhodobém časovém horizontu odhalit určité druhy defektů, které by se jistě dalo vylepšit.

Jedním z těchto problémů může být i návrhové rozhodnutí ze sekce 4.5.6,

neimplementující automatické povolování podskupin a podprojektů přímo v autorizační vrstvě, nýbrž ponechávání této zodpovědnosti na autorizačním pluginu.

Další možnou optimalizací je kontrola celkové doby volání autorizačního pluginu, aby nedocházelo k uváznutí webové aplikace, jak je popsáno v sekci 6.2.2.

Taktéž přidání další cachovací vrstvy mezi LDAP autorizační pluginy a LDAP spojení by způsobilo zrychlení celého procesu autorizace v zadavatelově instanci, jak je diskutováno v sekci 5.3.3.1.

Literatura

[1] OpenGrok: *OpenGrok [online]*. 2017, [cit. 2017-04-19]. Dostupné z: `http://opengrok.github.io/OpenGrok/`

[2] The Apache Software Foundation: *Apache Lucene Core [online]*. 2016, [cit. 2017-04-19]. Dostupné z: `https://lucene.apache.org/core/`

[3] Oracle Corporation: *JavaServer Pages Technology [online]*. 2013, [cit. 2017-04-22]. Dostupné z: `http://www.oracle.com/technetwork/java/javaee/jsp/index.html`

[4] Even, S.: *Graph Algorithms*. Cambridge University Press, druhé vydání, ISBN 978-0-521-73653-4.

[5] Trisul.cz: *Autentizace a autorizace [online]*. 2017, [cit. 2017-04-22]. Dostupné z: `http://www.trisul.cz/bezpecnost-autentizace-autorizace/`

[6] The Apache Software Foundation: *Authentication and Authorization [online]*. 2017, [cit. 2017-04-22]. Dostupné z: `https://httpd.apache.org/docs/2.4/howto/auth.html`

[7] The Apache Software Foundation: *Realm Configuration HOW-TO [online]*. 2017, [cit. 2017-04-22]. Dostupné z: `https://tomcat.apache.org/tomcat-8.0-doc/realm-howto.html`

[8] IBM Corporation: *Java classes and class loading [online]*. 2003, [cit. 2017-04-25]. Dostupné z: `https://www.ibm.com/developerworks/java/library/j-dyn0429/`

[9] The Apache Software Foundation: *Class Loader HOW-TO [online]*. 2017, [cit. 2017-04-25]. Dostupné z: `https://tomcat.apache.org/tomcat-8.0-doc/class-loader-howto.html`

[10] O'Reilly Media, Inc: *Internals of Java Class Loading [online]*. 2005, [cit. 2017-04-25]. Dostupné z: http://www.onjava.com/pub/a/onjava/2005/01/26/classloading.html

[11] Oracle Corporation: *Introduction to the PAM Framework [online]*. 2012, [cit. 2017-04-29]. Dostupné z: https://docs.oracle.com/cd/E26502_01/html/E29016/pam-01.html

[12] Oracle Corporation: *pam.conf [online]*. 2011, [cit. 2017-04-29]. Dostupné z: https://docs.oracle.com/cd/E23823_01/html/816-5174/pam.conf-4.html

[13] The Apache Software Foundation: *Interface Filter [online]*. 2017, [cit. 2017-04-30]. Dostupné z: https://tomcat.apache.org/tomcat-8.0-doc/servletapi/javax/servlet/Filter.html

[14] Oracle Corporation: *Using Sessions and Session Persistence in Web Applications [online]*. 2017, [cit. 2017-05-04]. Dostupné z: https://docs.oracle.com/cd/E13222_01/wls/docs81/webapp/sessions.html

[15] Free Software Foundation, Inc: *GNU Wget [online]*. 2016, [cit. 2017-05-07]. Dostupné z: https://www.gnu.org/software/wget/manual/wget.pdf

[16] The Apache Software Foundation: *Apache HTTP server benchmarking tool [online]*. 2017, [cit. 2017-05-07]. Dostupné z: https://httpd.apache.org/docs/2.4/programs/ab.html

[17] Oracle Corporation: *Oracle Application Server 10g (9.0.4) - Forms Single Sign-On [online]*. 2003, [cit. 2017-05-01]. Dostupné z: http://www.oracle.com/technetwork/developer-tools/forms/documentation/frm10gsso-131511.pdf

[18] Oracle Corporation: *Developing Applications for Single Sign-On [online]*. 2006, [cit. 2017-05-01]. Dostupné z: https://docs.oracle.com/cd/B28196_01/idmanage.1014/b15997/mod_osso.htm

[19] Oracle Corporation: *Configuring Single Sign-On using OracleAS SSO 10g [online]*. 2017, [cit. 2017-05-04]. Dostupné z: https://docs.oracle.com/cd/E23943_01/core.1111/e10043/osso_d_10g.htm

[20] Oracle Corporation: *Java Naming and Directory Interface (JNDI) [online]*. 2013, [cit. 2017-05-04]. Dostupné z: http://www.oracle.com/technetwork/java/jndi/index.html

[21] Oracle Corporation: *Tips for LDAP Users: Connection Creation [online]*. 2004, [cit. 2017-05-04]. Dostupné z: http://docs.oracle.com/javase/jndi/tutorial/ldap/connect/create.html

[22] Rambus: *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems [online]*. 2017, [cit. 2017-05-04]. Dostupné z: `https://www.rambus.com/timing-attacks-on-implementations-of-diffie-hellman-rsa-dss-and-other-systems/`

Příloha **A**

Seznam použitých zkratek

AB Apache Benchmark.

API Application Programming Interface.

CDDL Common Development and Distribution License (CDDL-1.0).

DFS Depth First Search.

DN Distinguished Name.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

JSON JavaScript Object Notation.

JSP Java Server Pages.

JVM Java Virtual Machine.

LDAP Lightweight Directory Access Protocol.

PAM Pluggable Authentication Modules.

SCM Source code management.

SQL Structured Query Language.

SSO Single Sign-On.

TCP Transmission Control Protocol.

URL Uniform Resource Locator.

XML Extensible markup language.

Příloha B

Změny v OpenGroku

V této kapitole následuje seznam akceptovaných „pull-requests“ – požadavků na změnu do veřejného repositáře na Gihubu[31], které implementují návrhy uvedené v této práci.

B.1 Abstraktní členění projektů do skupin

- Implementation of project groupings #1029
  - merged on 27 Jan 2016
- script wrapper for manipulating with groups #1094
  - merged on 2 May 2016
- new fold icon when group of projects is collapsed #1160
  - merged on 16 Aug 2016
- Groups script fix #1209
  - merged on 9 Oct 2016
- fixing autolocation of opengrok.jar #1236
  - merged on 17 Oct 2016
- Messages are group aware #1318
  - merged on 10 Jan 2017
- use click instead of doubleclick for selecting whole group #1394
  - merged on 8 Feb 2017

[31] https://github.com/OpenGrok/OpenGrok

- do not discover the opengrok configuration for reading groups #1409
  - merged on 14 Feb 2017
- Allowing users to generate empty configuration with Groups command. #1410
  - merged on 14 Feb 2017
- hiding empty groups #1417
  - merged on 15 Feb 2017
- making the group collapse threshold a tunable #1420
  - merged on 16 Feb 2017
- Invalid pattern in group always throws an exception #1468
  - merged on 14 Mar 2017
- Configuration checks #1531
  - merged on 21 Apr 2017

B.2 Autorizační framework

- authorizator framework #1083
  - merged on 20 Apr 2016
- Adding a documentation for groupings and authorization #1087
  - merged on 18 Apr 2016
- Changes to authorization framework #1117
  - merged on 22 Jun 2016
- Implementing cache and class resolution #1127
  - merged on 30 Jun 2016
- sample authorization plugin does not implement the interface #1131
  - merged on 7 Jul 2016
- adding more complex plugin example #1154
  - merged on 2 Aug 2016

- Authorization properly unloads plugins #1180
  - merged on 31 Aug 2016
- Adding unauthorized requests into statistics #1313
  - merged on 10 Jan 2017
- Authorization framework enhancements #1457
  - merged on 31 Mar 2017
- Authorization framework configuration in configuration.xml #1465
  - merged on 31 Mar 2017
- using authorization stacks instead of a list #1525
  - merged on 28 Apr 2017
- Easily discover groups in authorization plugins #1527
  - merged on 28 Apr 2017
- Support inheritance in plugin classes #1551
  - merged on 28 Apr 2017
- avoid unnecessary calls to authorization framework #1556
  - merged on 28 Apr 2017
- adding basic stats for authorization decisions #1558
  - merged on 28 Apr 2017
- targeting stacks for particular groups or projects #1560
  - merged on 3 May 2017
- more efficient locking for shared resource with more readers #1565
  - merged on 3 May 2017
- returning copy of the repository list #1566
  - merged on 3 May 2017
- adding the group discovery for forGroups also into the plugin #1577
  - merged on 5 May 2017

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `opengrok-1.1-rc2` ..... zdrojové kódy OpenGroku implementující návrhy v této práci
    - `authorization-plugins` . zdrojové kódy autorizačních pluginů pro zadavatele
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS