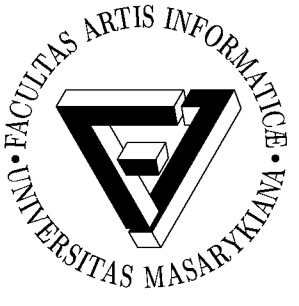
Masarykova univerzita
Fakulta informatiky



Autentizace a autorizace pro systém pro správu požadavků

Diplomová práce

Bc. Vlastislav Ramík

Brno, Jaro 2015

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Bc. Vlastislav Ramík

Vedoucí práce: Mgr. Marek Grác, Ph.D.

Poděkování

Rád bych poděkoval svému vedoucímu práce Mgr. Markovi Grácovi, Ph.D. a zaměstnancům společnosti Red Hat za konzultace a cenné rady při tvorbě této práce.

Shrnutí

Cílem této práce je vývoj autentizace a autorizace v existujícím systému pro správu požadavků eShoe za využití technologie Keycloak. Samotné implementaci, která bude realizována za pomocí technologií Java EE, Wicket a CDI, předchází analýza zabezpečení aplikací na platformě Java.

Klíčová slova

zabezpečení, autentizace, autorizace, Java EE Security, WildFly, Keycloak, open source, systém pro správu pozadavků, issue tracker

Obsah

1 **Úvod** 1
2 **Motivace** 3
3 **Java EE Security** 8
3.1 *Zabezpečení na úrovni kontejneru* 9
3.1.1 Deklarativní zabezpečení 9
3.1.2 Programatické zabezpečení 13
3.2 *Nevýhody a chybějící části specifikace* 14
3.2.1 Správa uživatelů 14
3.2.2 Ochrana hesla 15
3.2.3 Sprava rolí 15
3.2.4 Autorizace 15
4 **Zabezpečení ve WildFly** 16
4.1 *WildFly* 16
4.2 *Uživatelé* 16
4.3 *Aplikace* 17
4.3.1 Security domény 17
4.4 *Technologie používána serverem WildFly* 19
5 **Keycloak** 21
5.1 *Hlavní koncepty v Keycloaku* 21
5.2 *Model Keycloaku* 21
5.2.1 Realm 22
5.2.2 Aplikace 22
5.2.3 Client 22
5.2.4 Uživatelé 23
5.2.5 Mapování rolí 23
5.2.6 Implicitní aplikace 23
5.3 *Hlavní přednosti Keycloak serveru* 25
5.3.1 Log In mechanismus 25
5.3.2 Single Sign On 25
5.3.3 Registrace uživatelů 25
5.3.4 Práce s uživatelským účtem 25
5.3.5 Správa serveru 25
5.3.6 Vícefaktorová autentizace 25
5.3.7 Přihlášení přes sociální sítě 27
5.3.8 Podpora aplikací třetích stran 27

5.3.9 Adaptéry . . . 27
5.3.10 Samostatná služba . . . 27
5.4 *Autorizační proces* . . . 27
5.4.1 Token . . . 28
5.4.2 Skryté heslo . . . 28
6 **Analýza a návrh** . . . 29
6.1 *Současné řešení systému pro správu požadavků* . . . 29
6.2 *Analýza* . . . 30
6.2.1 Use Case diagram . . . 30
6.2.2 Datový model . . . 32
7 **Implementace** . . . 36
7.1 *Servisní vrstva* . . . 36
7.2 *Integrace s Keycloak API* . . . 36
7.3 *Prezentační vrstva* . . . 37
7.4 *Nastavení systémů* . . . 39
7.4.1 Nastavení Keycloak serveru . . . 39
7.4.2 Nastavení systému pro správu požadavků . . . 40
7.5 *Uživatelské rozhraní* . . . 40
7.6 *Použité technologie* . . . 41
7.6.1 Java Persistence API . . . 41
7.6.2 Criteria API . . . 41
7.6.3 Wicket . . . 41
7.6.4 RESTové služby . . . 43
7.6.5 CDI . . . 43
7.6.6 Keycloak . . . 43
7.6.7 Elasticsearch . . . 43
8 **Závěr** . . . 44
Literatura . . . 46

1 Úvod

V současné době je již téměř nezbytností, aby každá větší organizace měla možnost sledovat stavy požadavků, které v rámci jejích procesů vznikají (může se jednat např. o úkoly). Software, který tuto funkci zajišťuje, je obecně nazýván issue tracking system (systém pro správu požadavků). Spravuje jednotlivé úkoly, zaznamenává jejich aktuální a předchozí stavy, přiřazuje je řešitelům a umožňuje jejich kontrolu ve vztahu k zadaným termínům. Kromě toho lze tyto úkoly rozřazovat do kategorií a zajištovat pohodlné vyhledávání dle zadaných kritérií.

Nedílnou součástí každé aplikace by mělo být zabezpečení, aby nedocházelo k neoprávněnému přístupu k informacím. Právě zabezpečením na aplikační úrovni se tato práce zabývá.

Počítačová bezpečnost je všezahrnující termín v oblasti informačních technologií, který se zabývá zabezpečením virtuálního prostředí. To může zahrnovat ochranu a integritu dat, zabezpečení aplikací, hodnocení rizik a zranitelnosti autentizačních a autorizačních protokolů. Počítačová data tvoří v dnešních organizacích nevyčíslitelnou hodnotu. Jejich ochrana je proto pochopitelně enormně důležitá. Bezpečnost může být aplikována na úrovni operačního systému, aplikačního serveru, a také v jednotlivých aplikacích.

První kapitola přibližuje motivaci, která vedla k vytvoření samostatného řešení, i přes existenci jiných. Hlavním důvodem je snaha o moderní přístup, který zpřístupní v rámci open source produkt velkému množství vývojářů a zlepší šanci na postupné zkvalitňování, rozšiřování a vylepšování této aplikace. Pro srovnání rozebírá také jiná řešení zabezpečení u již existujících issue tracking systémů.

V druhé kapitole je představen programovací jazyk, který byl použit pro implementační část práce. Vzhledem k tomu, že celé jádro aplikace, jehož nástavba je předmětem projektu, je psáno v programovacím jazyce Java EE, je pochopitelné, že také toto aplikační zabezpečení je vytvořeno ve stejném jazyce. Jsou popsány jednotlivé koncepty obsažené v Java EE specifikaci, týkající se zabezpečení. Závěr kapitoly jmenuje nedostatky a chybějící části, Java EE specifikace, které by bylo dobré doplnit či vylepšit.

Představení serveru WildFly, pro který je tato konkrétní realizace systému správy požadavků vyvíjena, je obsahem čtvrté kapitoly. Wil-

dFly implementuje specifikaci Java EE 6, navíc obsahuje mechanismy např. pro správu uživatelů, realmů a domén. Je založen na mnoha ověřených open source projektech.

Open source projekt Keycloak poskytuje zabezpečovací funkcionalitu například pro Java aplikace (mobilní, webové, desktopové atd.). Jedná se o technologii, která byla zvolena pro implementaci autentizace a autorizace existujícího systému pro správu požadavků. Podrobnější popis tohoto přístupu je uveden v páté kapitole.

Šestá kapitola rozebírá aktuální stav issue tracking systému, analyzuje potřebná rozšíření systému. K tomu používá diagramy jazyka UML.

Kapitola sedmá popisuje samotnou implementaci projektu. Zahrnuje konfiguraci Keycloak serveru, aby jej bylo možno použít pro zabezpečení, a issue tracking systému. Přibližuje jednotlivé kroky prováděné v datové, servisní a prezentační vrstvě aplikace. Obsahuje ukázku uživatelského rozhraní implementovaného zabezpečení a vyjmenovává použité technologie.

Hlavním cílem je seznámení se se způsoby zabezpečení Java EE aplikací vyvíjených pro server WildFly, a vyvinutí autentizace a autorizace pro systém správy požadavků eShoe. Vedlejším cílem byla snaha o to, aby se samotná implementace, lépe řečeno její část - nacházející se v servisní a datové vrstvě a pracující s oprávněními, dala z aplikace vyjmout, a vytvořit z ní samostatnou knihovnu, použitelnou pro jiné projekty, které by byly zabezpečeny pomocí Keycloaku.

Tato práce je řešena v rámci průmyslové spolupráce fakulty informatiky se společností Red Hat Czech[1]. Během realizace práce došlo k oznámení ukončení dalšího vývoje na projektu PicketLink, což významně ovlivnilo volbu zvolené technologie.

1. `http://www.redhat.com/`

2 Motivace

Existuje několik systémů pro správu požadavků a každý z nich řeší zabezpečení po svém. Přesto zde byla silná motivace k vytvoření nového issue tracking systému, a v návaznosti na to samozřejmě taktéž potřeby jeho zabezpečení.

V rámci diplomové práce Mgr. Moniky Gottvaldové vzniklo jádro systému pro správu požadavků, které je založeno na programovacím jazyce Java EE [6]. Do tohoto existujícího systému bylo třeba vytvořit mechanismy pro autentizaci a autorizaci, které by, stejně jako toto jádro, splňovaly požadavky na moderní open source systém.

Moderním řešením se v těchto souvislostech rozumí použití aktuálních technologií, které ovládá velké množství vývojářů. Tuto skutečnost ještě podporuje fakt, že se jedná o open source projekt, který je dostupný zdarma. Ostatní developeři mohou přispět vlastní částí zdrojového kódu, a projekt obohatit o množství inovací a nových funkcí. Čím větší je počet vývojářů, kteří na kódu pracují, tím vyšší je pravděpodobnost, že výsledná aplikace bude kvalitní. Využití neaktuálních technologií samozřejmě automaticky neznamená nižší kvalitu vytvořeného produktu, omezuje však základnu programátorů, který danou technologii ovládají. Motivace k naučení se starší technologii zpravidla nebývá velká.

Moderní řešení také nabízí podporu pro integraci se sociálními sítěmi, které jsou velmi populární, a tudíž je toto spojení žádané. Pro účely diplomové práce tak byly hledány právě takové technologie, které uvedené propojení umožňují.

Za další princip moderního řešení by mohl být považován fakt, že část systému zabezpečení – správa uživatelů, jejich rolí, včetně vzájemného propojení, apod. - je poskytována jako samostatná služba, která může být provozována externě. Významně to přispívá ke zjednodušení práce vývojářů, vytvářejících zabezpečení libovolné aplikace, a poskytuje jistotu vysoké kvality tohoto zabezpečení.

V současné době existuje několik issue tracking systémů, níže jsou uvedeny některé z nich. Stručný popis těchto systémů se zaměřuje na jejich zabezpečení.

- Bugzilla

  Bugzilla je systém pro správu chybových hlášení zefektivňující jejich odstraňování. Vývojáři věnují otázce zabezpečení velkou pozornost. Bugzilla je naprogramována pomocí jazyka Pearl a používá tzv. taint mode, aby zabránila útokům typu SQL injection, nebo tzv. cross site scripting. Systém je pravidelně udržován a aktualizován prostřednictvím patchů. K prevenci spamování Bugzilla nepřihlášeným uživatelům zamezuje přístup k e-mailovým adresám ostatních uživatelů. Snaží-li se někdo o neoprávněnou autentizaci tím, že se snaží uhádnout heslo, systém účet automaticky po několika pokusech zablokuje. Brání tím použití tzv. brute force k získání přístupu k účtu někoho jiného [9].

- Trac

  Trac je open source systém běžící na webovém rozhraní vytvořený pomocí Pythonu. Trac používá jednoduchý systém zabezpečení k určení, ke kterým zdrojům má uživatel přístup. Nepřihlášení uživatelé, kteří přistupují k systému, jsou označeni jako "anonymní" a existuje možnost pro ně specifikovat oprávnění. Přihlášení uživatelé patří automaticky do skupiny *authenticated*, která dědí oprávnění ze skupiny *anonymous*. Části systému, na které uživatel nemá oprávnění, nejsou zobrazeny v navigaci webové stránky. Uživateli je možné přiřadit dodatečná oprávnění [4].

- JIRA

  Jira je tracking systém vytvořený pro projektový management, řešící sledování chyb, úkolů, problémů a návaznosti teamových aktivit. Poskytuje flexibilní systém zabezpečení, definující kdo má k systému přístup a může zde provádět operace. Je rozděleno na pět druhů. Oprávnění *global* se používá pro přístup do systému, *projektové* oprávnění je aplikováno na jednotlivé projekty a jejich issue. Oprávnění na úrovni *issue* umožňuje zobrazovat jednotlivá issue, také závisí, jaká oprávnění jsou nastavena u příslušného projektu. Dále existují oprávnění týkající se komentářů (*comment*) a záznamů práce (*work-log*) [1].

- Mantis

  Mantis je volně šiřitelný systém pro správu chybových hlášení, běžící na webovém rozhraní, který je vytvořen v PHP. Systém poskytuje možnost nastavit oprávnění rolím. Ty jsou přiřazeny uživatelům. V případě potřeby je možné nastavit nebo změnit oprávnění. To se provádí úpravou konfiguračního souboru. Uživatelé systému mohou hlásit nalezené bezpečnostní hrozby prostřednictvím Mantis systému a vývojáři se jimi zabývají [16].

- BugTracker.NET

  BugTracker.NET je obecně použitelný, volně šiřitelný a open source systém pro správu požadavků a chybových hlášení, vytvořen prostřednictvím ASP.NET, C#. Systém umožňuje přiřazovat oprávnění na základě projektů nebo organizací. Kvůli tomuto modelu je potřeba věnovat nastavení práv velkou pozornost. Systém rozlišuje uživatele na ty, kteří jsou interní a ty, kteří jsou externí. Existují různé úrovně oprávnění. Ty mohou být přiřazeny uživateli, případně může být uživateli přiřazena sada oprávnění přímo [17].

- Redmine

  Redmine je flexibilní systém pro projektový management a sledování chybových hlášení, naprogramovaný v Ruby on Rails. Uživatelé systému mohou hlásit nalezené bezpečnostní hrozby ve formě tiketu. Systém chrání citlivá data tím, že používá kontrolu přístupu založenou na rolích. V systému navíc existují role *non member* a *anonymous*. Role mohou být skládány do skupin [8]. Některé verze Redmine jsou zranitelné tzv. XSS útoky [19].

- FogBugz

  FogBugz je nástroj pro sledování úkolů vyvíjený společností Fog Greek software. Vývojáři Fog Creek udržují zabezpečení na dobré úrovni a aktuální. Uživatelé mohou nahlásit bezpečnostní riziko a jeho řešení má vysokou prioritu. V systému

> platí přísná bezpečnostní pravidla. Nastavení zabezpečení mohou provádět pouze členové týmu FogBugz On Demand a FogBugz Development Lead. Vnitřní předpisy firmy zakazují zaměstnancům přístup k datům zákazníků bez jejich výslovného souhlasu [5].

Každý Issue Tracking systém řeší zabezpečení vlastním způsobem. Některá z nich jsou dostatečná, ale jejich použití mohou bránit různé okolnosti - např. použití starší technologie (programovacího jazyka), případně fakt, že se nejedná o open source, řešení nejsou flexibilní, není možné je nakonfigurovat dle specifických požadavků.

Řešení, vyvíjené v rámci této práce se snaží všechny tyto nedostatky řešit. Zaměřuje se zejména na možnost vlastní libovolné konfigurace.

Vyvinutý systém je od počátku postavený na vysoké míře konfigurovatelnosti, aby každý budoucí uživatel mohl produkt co nejvíce přizpůsobit svým potřebám. Zabezpečení by pak logicky mělo fungovat v podobném duchu. Nesmí klást zvolenému přístupu žádná omezení. Je navrženo tak, aby bylo možné přidávat libovolné množství různých rolí, ke kterým jsou přidávána různá oprávnění k akcím systému.

Jak již bylo v úvodu naznačeno, během realizace této práce bylo oznámeno zastavení dalších developerských prací na projektu PicketLink, který je nástavbou PicketBoxu. Oba tyto projekty byly v rámci fáze výběru technologií, společně s projektem Keycloak, prostudovány z hlediska jejich přístupu k řešení autorizace a autentizace. Výše uvedená skutečnost měla samozřejmě velký vliv při výběru použité technologie, nebyla však jediným kritériem. Hlavním důvodem, proč byl zvolen projekt Keycloak je fakt, že se jedná o vysoce moderní technologii, která splňuje zmíněné požadavky na zabezpečení aplikací. Navíc poskytuje podporu pro propojení se sociálními sítěmi. Keycloak má velký potenciál stát se populárním v rámci vývojářské základny, což pravděpodobně povede k jeho rozšíření mezi velké množství developerů a bude mít významný vliv na jeho výslednou kvalitu. Při výběru hrál roli také fakt, že Keycloak je řešením, které je možné začít používat ihned, bez předchozí implementace do aplikace, je třeba jej pouze nakonfigurovat. Navíc poskytuje uživatelské rozhraní pro zjednodušení konfigurace. Pomocí PicketLinku by mu-

sel být nejdříve vytvořen vlastní zabezpečovací mechanismus v rámci aplikace.

Kromě hlavního cíle, kterým jak již bylo uvedeno je vyvinutí autorizace a autentizace pro systémy správy požadavků, se práce snaží o vytvoření univerzálnějšího řešení pro zabezpečení. A to nejen tohoto konkrétního systému, ale poskytnutí funkcionality pro práci se zabezpečením prostřednictvím Keycloaku pro širší použití.

3 Java EE Security

Kapitola se zabývá hlavními body zabezpečení, které popisuje specifikace Java EE. Jsou zde uvedeny dva přístupy k předání informace o zabezpečení. Prvním z nich je deklarativní způsob, který je prováděn pomocí anotací nebo deployment descriptoru. Druhý přístup popisuje programatický způsob specifikování bezpečnostních mechanismů v aplikacích.

Každá enterprise aplikace, která obsahuje citlivé zdroje informací, a ke které přistupuje mnoho různých uživatelů pomocí nechráněné otevřené sítě jako je Internet, by měla být chráněna.

Ačkoli se kvalita implementace různí, všechny systémy sdílí následující požadavky pro zabezpečení [3]:

- Autentizace

  Způsob, kterým se komunikující strany, například klient a server, navzájem prokáží, že jsou tím, kterým říkají, že jsou, když žádají o přístup k nějakému zdroji.

- Řízení přístupu ke zdrojům (Autorizace)

  Možnost omezit přístup ke zdrojům jen pro určité skupiny uživatelů či aplikací z důvodu zajištění integrity a důvěryhodnosti na základě jejich identity.

- Integrita dat

  Její význam tkví v tom, že prokazuje, zda informace nebyla jakkoli modifikována třetí stranou, entitou různou od té, pro kterou je informace určena. Například odesílatel, který posílá data skrz nechráněnou síť, musí být schopen detekovat, že data byla nějak změněna poté, co byla odeslána.

- Důvěrnost dat

  Zajištění, že informace je dostupná pouze pro uživatele, který je k jejímu přístupu autorizován.

- Nemožnost odmítnutí odpovědnosti

  Vždy je možné prokázat, že uživatel provedl nějakou akci a ten nemůže popřít, že se tak stalo.

- Auditování

  Způsob, jak zachytit neoprávněnou manipulaci s nějakým záznamem s následnou možností přezkoumání efektivity zabezpečení a jejich mechanismů.

Služby Java EE security jsou poskytovány kontejnery a mohou být implementovány deklarativními nebo programatickými technikami. Java EE security poskytuje robustní a jednoduše konfigurovatelný zabezpečovací mechanismus pro autentizaci uživatelů a autorizaci přístupu k aplikačním funkcím na různých úrovních. Služby Java EE security jsou odděleny od zabezpečení operačních systémů.

3.1 Zabezpečení na úrovni kontejneru

Aplikační kontejnery poskytují zabezpečení pro jednotlivé komponenty. Pro zabezpečení specifikované v Java EE Security poskytují kontejnery dva druhy zabezpečení, deklarativní a programatické [3].

3.1.1 Deklarativní zabezpečení

Tzv. deployment descriptor (`web.xml`) je konfigurační soubor Java artefaktů, který je nahráván do běžícího kontejneru. V Java EE tento soubor popisuje, jak by komponenta, modul, nebo aplikace měly být do kontejneru nahrány. Zde se také specifikuje zabezpečení aplikace deklarativním způsobem. Obsažená informace může být změněna bez potřeby úpravy zdrojových kódů systému. Deployment descriptor je za běhu (*runtime*) Java EE serverem načten a server se podle informací obsažených v něm zachová odpovídajícím způsobem. Struktura deployment descriptoru je předepsaná, a pokud zde není hodnota specifikovaná a není daná ani anotacemi, tak se použije výchozí (*default*) hodnota.

Deklarativní způsob zabezpečení je logicky oddělený od aplikace včetně rolí, kontroly přístupu a autentizačních mechanismů. Deployment descriptor je hlavní složkou deklarativního zabezpečení ve webových aplikacích.

Webové aplikace jsou nahrávány do běhového prostředí kontejneru. Vývojáři deklarují zabezpečení použitím anotací nebo deployment descriptoru. Tato informace je předávána do kontejneru, kde

jsou z ní extrahovány role používající se pro autentizaci a nastavuje se zde příslušný bezpečnostní mechanismus. Pokud bezpečnostní požadavky nejsou přímo specifikovány, buď prostřednictvím anotací nebo deployment desriptoru, jsou použity požadavky nezávislé na konkrétní implementaci aplikace, které jsou nastaveny v konkrétním běžícím kontejneru.

Bezpečnostní omezení

Bezpečnostní omezení (*security constraint*) se používá pro zajištění oprávněného přístupu ke zdrojům pomocí URL mapování.

V případě, že je webová aplikace servletová, pro určení informací o zabezpečení je možné použít anotace. Pro specifikování bezpečnostního omezení mohou být použity anotace `@HttpConstraint`, případně `@HttpMethodConstraint`, které se nachází v rámci anotace `@ServletSecurity` [18].

`@ServletSecurity` je anotace specifikována na úrovni třídy, a díky ní servletový kontejner pozná, že má být daná třída zabezpečena. Toto zabezpečení je založeno na HTTP protokolu. Servletový kontejner na základě přijaté informace zajistí, že bude uvedené omezení použito ve stejném smyslu jako `url-pattern` v deployment descriptoru.

Pokud webová aplikace nepoužívá servlety, mohou vývojáři uvést bezpečnostní omezení pomocí deployment descriptor souboru. Autentizační mechanismus nelze specifikovat pomocí anotací. Pokud je potřeba použít jiný autentizační mechanismus než základní (*BASIC*), je deployment desriptor, kde je uvedená autentizační metoda, povinnou součástí aplikace.

Součástí elementu `security-constraint` v rámci deployment desriptoru aplikace mohou být následující podelementy [3]:

- `web-resource-collection`

  Obsahuje seznam URL vzorů (*URL pattern*) a HTTP operací (*HTTP method*), které popisují zdroje, jenž je potřeba zabezpečit. URL vzorem se rozumí část URL, která následuje po protokolu, název hostu (*hostname*) a portu. HTTP operace jsou metody shodující se s URL vzorem, které je možné zabezpečit. Web resource collection element se skládá z následujících podelementů:

- `web-resource-name`

  Web resource name je název zdroje, jeho použití není povinné.

- `url-pattern`

  Url-pattern se používá pro vyjmenování seznamu zdrojů, které je potřeba zabezpečit. Mnoho aplikací používá jak chráněné, tak nechráněné zdroje. Poskytování nezabezpečených zdrojů je určeno tím, že se nenakonfiguruje příslušný url-pattern elementu.

  Specifikováním url-pattern lze určit, který zdroj bude zabezpečený. Například webové stránky internetového obchodu obsahujícího katalog zboží a nákupní košík. Požadavek je, aby byl katalog volně přístupný všem uživatelům obchodu, zatímco nákupní košík jen přihlášeným zákazníkům. Pomocí url-pattern jsou specifikovány URL cesty k webové aplikaci, které jsou zabezpečené. Předpokládejme, že pattern `/cart/*` je zabezpečený, žádné další stránky zabezpečené nejsou a aplikace je nasazena na localhost a portu 8080 a běží na adrese `/app`, pak platí následující.

  * http://localhost:8080/app/index.xhtml není zabezpečeno.
  * Zdroj http://localhost:8080/app/cart/index.xhtml je zabezpečen.

  Jakmile uživatel přistoupí k zabezpečenému zdroji v adresáři `cart/`, je zobrazen formulář se žádostí o přihlášení.

- `http-method` a `http-method-omission`

  Pomocí http-method se dá určit, které metody by měly být zabezpečeny. http-method-omission určuje, které by měly být ze zabezpečení vyjmuty. HTTP metody jsou zabezpečeny, nebo nezabezpečeny následovně:

  * V případě, že nejsou uvedeny žádné HTTP metody, znamená to, že všechny metody jsou zabezpečené.
  * Metody vyjmenované v podelementu http-method jsou zabezpečené, nevyjmenované metody jsou nezabezpečené.

    - HTTP metody zmíněné v podelementu http-method-omission, nejsou zabezpečené.

- `auth-constraint`

  Určuje, zda je požadována autentizace a jmenuje role, které jsou autorizovány k vykonávání zabezpečených požadavků. Autorizační omezení (auth-constraint) obsahuje jeden nebo více podelementů role-name.

  Auth-constraint určuje požadavky pro autentizaci a jména rolí, která jsou autorizována k přístupu k URL pattern a HTTP metodám, které jsou specifikovány v security-constraint. V případě, že v deployment descriptoru není auth-constraint definován, kontejner musí akceptovat všechny požadavky na přístup k systému. Pokud je auth-constraint definován, ale nejsou v jeho rámci definovány žádné role-name, kontejner neakceptuje přístup za žádných okolností. Každá role definována v role-name musí být zároveň definována v elementu security-role v rámci aplikace nebo musí mít hodnotu *, což je rezervované jméno role reprezentující každého uživatele používajícího aplikaci. Jména rolí jsou tzv. case sensitive, to znamená, že záleží na velkosti písmen.

  Příklad použití zmíněných elementů v deployment descriptoru web.xml:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
version="2.5">
...
    <security-constraint>
        <web-resource-collection>
          <web-resource-name>Users</web-resource-name>
            <url-pattern>/login/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>User</role-name>
        </auth-constraint>
```

```
        </security-constraint>

        <security-role>
            <role-name>User</role-name>
        </security-role>
    ...
    </web-app>
```

- `user-data-constraint`

  Určuje, jak jsou zabezpečena data, která jsou transportována mezi klientem a serverem. User data constraint může specifikovat, zda je komunikace zabezpečena na transportní vrstvě spojení, například prostřednictvím HTTPS. Toto může být specifikováno pro všechny HTTP metody nebo URL vzory definované v security-constraint.

3.1.2 Programatické zabezpečení

Zabezpečení tohoto typu je úzce spojeno se samotnou aplikací, přímo v jejím zdrojovém kódu je popsán způsob jak provádět různá bezpečnostní rozhodnutí. Tento způsob je užitečný ve chvíli, kdy deklarativní zabezpečení samo o sobě nestačí. Deklarativní zabezpečení nemusí například stačit, pokud je v aplikaci použito podmíněné přihlášení v nějaké nestandartní situaci.

Podle specifikace se programatické zabezpečení servletové aplikace skládá z následujících metod rozhraní `HttpServletRequest` [18]:

- `authenticate` aplikaci umožňuje iniciovat autentizaci uživatele, který poslal žádost na přístup k určitému zdroji.
- `login` je metoda poskytující aplikaci možnost ověřit uživatelské jméno a heslo uživatele.
- `logout` poskytuje aplikaci mechanismus odhlášení, resetuje identitu uživatele, který je přihlášený.
- `getRemoteUser` vrací jméno uživatele, který vyslal požadavek serveru.

- `isUserInRole` vrací pravdivostní hodnotu, která určuje, zda je daný uživatel zahrnut ve specifikované roli.

- `getUserPrincipal` určuje jméno uživatele, který poslal požadavek a vrací objekt `java.security.Principal` vztahující se k tomuto uživateli. Voláním metody `getName()` na instanci Principal se vrací jméno uživatele.

V případě, že uživatel není autentizován, metoda `getRemoteUser` vrací `null`, metoda `isUserInRole` vrací `false`, a `getUserPrincipal` vrací také `null`.

Jako argument bere `isUserInRole` textový řetězec (`String`), který odkazuje na aplikační roli. Argumentem této metody by měla být textová reprezentace role, která je specifikována v deployment descriptoru v rámci elementu servlet, podelementu security-role-ref a role-name. Každý element security-role-ref by měl obsahovat podelement role-link, jehož hodnota je jméno role v rámci aplikace. Kontejner používá security-role-ref a role-name k porovnání security-role, aby mohl rozhodnout, zda je uživatel ve specifikované roli.

Přidáním metod authenticate, login a logout do servletové specifikace, se deployment descriptor stal nepovinnou součástí webových aplikací. Nicméně stále může být užitečný na specifikování zabezpečení deklarativním způsobem.

3.2 Nevýhody a chybějící části specifikace

Ve specifikaci programovacího jazyka Java EE [3] schází některé důležité aspekty týkající se zabezpečení aplikací. Časté řešení je použití knihoven třetích stran nebo vyvinutí vlastních knihoven. Tato řešení mohou a nemusí být potencionálně zranitelná nebo nezabezpečená.

3.2.1 Správa uživatelů

V současné době neexistuje standardizovaná podpora pro správu uživatelů v Java EE. Aplikace v Java EE platformě nemají žádný přenositelný způsob vytváření, mazání a úpravy uživatelů, také správa skupin chybí.

3.2.2 Ochrana hesla

Zabezpečení hesla je další věcí, která není specifikací Java EE nijak řešena. Aplikace mohou uchovávat hesla na různých místech, jako jsou anotace, deployment desriptor, URL nebo soubory.

3.2.3 Sprava rolí

Aktuálně chybí standardizovaná podpora pro správu rolí v rámci aplikace na plaformě Java EE. Aplikace nemají předepsány způsoby mapování rolí k uživatelům a uživatelským skupinám.

3.2.4 Autorizace

Java EE podporuje autorizaci na základě metod, kontroluje přidělené role autentizovaného uživatele. Chybí standardizovaný způsob jak do autorizačního procesu zahrnout pravidla doménových aplikací.

4 Zabezpečení ve WildFly

V kapitole jsou probrány zabezpečující mechanismy, které se používají v rámci aplikačního serveru WildFly [13].

4.1 WildFly

WildFly je flexibilní a lehce spravovatelné běhové prostředí pro aplikace na platformě Java. Projekt Wildfly, dříve známý jako JBoss AS, je open source projekt, na jehož vývoji se vedle komunity podílí developeři z firmy RedHat. Je napsán v programovacím jazyce Java a implementuje Java Platform – Enterprise Edition (Java EE) specifikaci. Běží na jakékoli platformě, na které běží Java Virtual Machine (JVM). Wildfly je zdarma šířený pod licencí GNU Lesser General Public License (LGPL). Vedle komunitní verze projektu existuje produktová enterprise edition verze pojmenovaná JBoss EAP[1]. Aktuální verze WildFly je 8.2.0.Final, nicméně brzy by se měla vydat následující verze 9.0.0.Final, 9.0.0.CR1 je již k dispozici ke stažení na stránkách projektu [13]. Wildfly 8 je přímým pokračováním projektu JBoss AS[2].

4.2 Uživatelé

Pro správu uživatelů aplikační server WildFly poskytuje zabezpečení založené na realmech. Security realm je množina mapování mezi uživateli a jejich rolemi. Je to mechanismus umožňující přidání autorizace a autenticase do EJB nebo webových aplikací. JBoss WildFly 8 poskytuje dva základní realmy [15]:

- ManagementRealm

  ManagementRealm ukládá autentizační informace pro *Management API*, které poskytuje funkcionalitu pro *Management Command Line Interface (Management CLI)* a konzoli (*Management Console*), která je založena na webových stránkách. ManagementRealm poskytuje autentizační systém pro samotný Wild-

1. `http://www.jboss.org/products/eap/overview/`
2. `http://jbossas.jboss.org/`

Fly server. ManagementRealm je vhodný pro použití pro aplikace, které potřebují přistupovat se stejnými pravidly, které jsou použity pro Management API.

- ApplicationRealm

  V realmu ApplicationRealm jsou uloženy informace o uživatelích, jejich heslech a rolích, které mohou být poskytnuty aplikacím. Existuje možnost vytvořit si vlastní security realm specifický pro potřeby uživatelů aplikačního serveru. Nový uživatel je vytvářený v rámci serveru a jednoho nebo více security realmu. Security realm je zodpovědný za autentizaci a autorizaci uživatelů, kteří mají oprávnění spravovat WildFly 8 server prostřednictvím Management API, Management CLI, nebo webové Management Console. Tyto realmy mohou být použity pro zabezpečení aplikací, které běží v rámci běhového prostředí aplikačního serveru.

4.3 Aplikace

Model Java EE security je deklarativní nebo programatický. Deklarativní přístup izoluje zabezpečení od business logiky aplikace. Ve WildFly serveru deklarativní přístup zabezpečení poskytují tzv. security domains. Programatický způsob zabezpečení se řeší na úrovni jednotlivých aplikací.

4.3.1 Security domény

Domény zabezpečení jsou součást subsystému security v aplikačním serveru WildFly 8 [14]. Konfigurace zabezpečení se nachází na jednom místě a může být jednoduše spravována. Security domény se skládají z konfigurace pro autentizaci, autorizaci, mapování a auditování. Domény implementuji Java Authentication and Authorization Service (JAAS)[3] deklarativní zabezpečení.

3. `http://docs.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html`

Autentizace

Autentizace je proces ověření identity uživatele. V terminologii zabezpečení je uživatel brán jako *principal*. Nejběžnější autentizační mechanismus je kombinace uživatelského jména a hesla. Další mechanismy mohou být například sdílené klíče (*shared keys*), čipové karty, nebo otisky prstů. Výstup úspěšné autentizace je ve smyslu Java Enterprise Edition objekt principal.

WildFly 8 používá systém, který je flexibilní, je možné přidávat pluginy a integrovat jej s již existujícími autentizačními systémy. Každá doména může obsahovat jednu nebo více nakonfigurovaných autentizačních modulů. Každý modul zahrnuje množství konfiguračních možností, aby jej bylo možné nastavit přesně podle potřeb uživatele aplikačního serveru. Moduly se dají jednoduše spravovat prostřednictvím webové management konzole [14].

Autorizace

Autorizace je způsob, jakým server rozhoduje o tom, zda má autentizovaný uživatel právo k přístupu k požadovanému zdroji nebo operaci. Je to mechanismus pro udělení nebo neudělení přístupu na základě identity uživatele.

WildFly 8 používá modulární systém pro konfiguraci autorizace. Každá security doména obsahuje jednu nebo více autorizačních politik, ty obsahují základní modul, který definuje její chování. Pomocí webové management konzole lze nastavit autorizační proces přesně podle požadavků.

Mapování oprávnění

Mapování (*security mapping*) se vztahuje k možnosti přidat, upravit nebo smazat informace uložené v *principal*, roli nebo atributu před předáním této informace aplikaci. Security mapping umožňuje kombinovat autentizaci a autorizaci po jejich provedení, ale ještě před tím než je informace předána aplikaci.

Audit

Audit (*security auditing*) slouží pro spuštění událostí, jako je zápis do logu na základě události, která nastane v security subsystému. Tento

mechanismus je nakonfigurován v rámci security domain, společně s autentizací, autorizací a security mapping.

Příklad nastavení security domény v subsystému v konfiguračním souboru standalone.xml [14]:

```
<subsystem xmlns="urn:jboss:domain:security:1.2">
    <security-domains>
        <security-domain name="other" cache-type="default">
            <authentication>
                <login-module code="Remoting"
                    flag="optional">
                    <module-option name="password-stacking"
                        value="useFirstPass"/>
                </login-module>
                <login-module code="RealmDirect"
                    flag="required">
                    <module-option name="password-stacking"
                        value="useFirstPass"/>
                </login-module>
            </authentication>
        </security-domain>
        ...
    </security-domains>
    ...
</subsystem>
```

4.4 Technologie používána serverem WildFly

Aplikační server WildFly staví na mnoha kvalitních a ověřených open source projektech. Například Hibernate[4], Narayana[5], Infinispan[6], Iron-

4. http://hibernate.org/
5. http://narayana.io/
6. http://infinispan.org/

Jacamar[7], RESTEasy[8], Weld[9], HornetQ[10], JGroups[11], Mojarra[12], Apache CXF[13], Arquillian[14], PicketLink[15] a dalších.

K zabezpečení WildFly serveru je použit projekt PicketLink, ten používá knihovny z projektu PicketBox. PicketBox je open source security framework, který poskytuje podporu pro autentizaci, autorizaci, mapování rolí a bezpečnostní audit java aplikacím [11].

PicketLink je open source projekt postavený na PicketBoxu, který zjednodušuje zabezpečení a poskytuje správu identit (*identity management*) pro aplikace vyvíjené na platformě Java. Tento framework poskytuje širokou škálu mechanismů k zabezpečení aplikací, například autentizaci, autorizaci, API pro práci s oprávněními a flexibilní řešení pro správu identit (IDM). Jedna z hlavních feature, která je široce používána po celém světě, je PicketLink Federation poskytující SAML. Navíc PicketLink obsahuje spoustu nástrojů, pomocných tříd a API, které mohou značně usnadnit práci vývojářům v oblasti zabezpečení, správy uživatelů, jejich rolí, skupin a atributů. PicketLink poskytuje robustní model, který umožňuje kvalitní řízení přístupu ke zdrojům pro aplikace na platformě Java EE [12].

Bylo oznámeno, že se vývojáři z firmy Red Hat, kteří pracují na projektu PicketLink, přesunou na vývoj nového projektu jménem Keycloak. Funkcionalita poskytována PicketLinkem bude sloučena s Keycloakem. Nadále se bude vyvíjet primárně projekt Keycloak, i když projekt PicketLink se rušit nebude, již se na něm bude provádět vývoj pouze ze strany komunity [7].

7. http://www.ironjacamar.org/
8. http://resteasy.jboss.org/
9. http://weld.cdi-spec.org/
10. http://hornetq.jboss.org/
11. http://jgroups.org/
12. https://java.net/projects/mojarra/
13. http://cxf.apache.org/
14. http://arquillian.org/
15. http://picketlink.org/

5 Keycloak

Keycloak je nový open source projekt. Je to autentizační a autorizační server, který se zaměřuje na moderní technologie jako HTML5, nebo mobilní technologie. Má podporu také pro zabezpečení pro desktopové, webové aplikace nebo nástroje příkazové řádky. Keycloak server poskytuje uživatelům možnost přihlašování, odhlašování, registrování a správu jejich uživatelských účtů. Aktuálně poslední vydaná verze je 1.1.0.Final, nicméně nová verze 1.2.0.Final bude vydána brzy, 1.2.0.CR1 je již k dispozici ke stáhnutí ze stránek projektu[1].

5.1 Hlavní koncepty v Keycloaku

Nejdůležitějším konceptem, který Keycloak používá, je *realm*. Realm spravuje a zabezpečuje metadata pro skupiny uživatelů a registrovaných klientů. Uživatel může být vytvořen v rámci konkrétního realmu prostřednictvím administrační konzole. Na úrovni realmu mohou být specifikovány role a oprávnění, které je možné přiřadit konkrétním uživatelům.

Client je služba, která je zabezpečena prostřednictvím realmu. Často se klient používá pro každou aplikaci, která je zabezpečena pomocí Keycloaku. V případě, že uživatel navštíví webovou stránku aplikace, aplikace může přesměrovat uživatele na přihlašovací webový formulář Keyclaok serveru s žádostí o přihlášení. Jakmile je uživatel úspěšně autentizován, má oprávnění k přístupu k jakémukoli klientovi nebo aplikaci, která je spravována příslušným realmem, a to bez nutnosti se znovu přihlašovat. Analogicky to také platí pro odhlášení. Na úrovni klienta mohou být specifikovány role, které je možno přiřadit uživatelům.

5.2 Model Keycloaku

V rámci realmu může existovat jeden nebo více realmů, každý realm obsahuje množství aplikací, rolí a uživatelů.

---

1. `http://keycloak.jboss.org/`

5.2.1 Realm

Realmu poskytuje množství nastavení, které je možno provést. Například povolení nebo zakázání samotného realmu, přihlašování pomocí sociálních sítí, registrační mechanismus, mechanismus pro obnovení zapomenutého uživatelského jména nebo hesla, možnost zapamatování uživatele nebo ověření e-mailové adresy po prvním přihlášení. Lze povolit nebo zakázat tzv. *Direct Grant API*, což umožňuje pracovat s Keycloakem přímo prostřednictvím RESTových dotazů. Nachází se zde konfigurace zabezpečení spojení (HTTPS) nebo možnost nastavit jaké autentizační metody budou vyžadovány, například kombinace jména a hesla, certifikát apod. Je zde konfigurace e-mailového serveru, který se stará o zasílání e-mailových zpráv, například v případě zapomenutého hesla.

Nachází se zde také správa rolí, které jsou dostupné všem aplikacím v rámci realmu.

5.2.2 Aplikace

Na úrovni aplikací se nachází několik nastavení, které je možno provést. Jsou zde specifikovány validní *Redirect URIs*, které se používají v OAuth2. Říkají, na jaké adrese je konkrétní aplikace nasazena a přístupná. Keycloak má podporu pro *Web Origins (CORS)*, jejich správné nastavení dovoluje JavaScriptovým aplikacím z různých domén komunikovat s Keycloakem, toto nastavení lze provádět na úrovni jednotlivých aplikací.

V rámci každé aplikace je možné vytvořit seznam rolí, které jsou unikátní pro každou jednotlivou aplikaci. Aplikace mohou mít definovaný tzv. *scope*, konkrétní aplikace může mít například nastaven scope, který ji umožní používat role ostatních aplikací. Lze nastavit, které specifické role budou určeny k používání.

5.2.3 Client

Klienti jsou služby třetích stran, od aplikací se liší tím, že nejsou asociovány s konkrétním Keycloak serverem. Pro přístup ke zdrojům, které potřebují, musí žádat o povolení přístupu. Například při použití nějaké externí aplikace je po přihlášení uživateli zobrazen požadavek k povolení přístupu k různým zdrojům.

Podobně jako aplikace i klienti mají možnost nastavit *Redirect URIs*, *Web Origins (CORS)* a *Scope*. Na rozdíl od aplikací však nemají role. K dispozici mají pouze role aplikace, kterou používají. Lze nastavit, které specifické role budou možno použít.

5.2.4 Uživatelé

Uživatelé mají svůj profil v rámci realmu. Profil je plně konfigurovatelný, lze například specifikovat, které informace se o uživatelích budou do profilu ukládat (jméno, příjmení, e-mail, ...), nebo jaké validace jsou pro profil potřebné.

V profilu je uloženo, jaké přihlašovací metody (*credentials*) se používají, může to být kombinace uživatelského jména a hesla, nebo vícefaktorové autentizační metody například klientské certifikáty. Každý uživatel může mít v rámci realmu specifikovánu množinu rolí a jejich mapování.

5.2.5 Mapování rolí

V realmu jsou uvedeny globální role. Každá aplikace má vlastní role specifické pro konkrétní aplikaci. Role se dají kombinovat do složených rolí *composite roles*. Role se tím skládají do logických spojení. Každá role, jak na úrovni realmu tak na úrovni aplikace, může být označena jako composite. Označme ji například *Role A*. To k ní umožní přiřazení jiné role (*Role B*) z realmu nebo aplikace. Pokud uživatelovo mapování rolí obsahuje roli A, pak automaticky obsahuje i roli B. Tato asociace je dynamická. To znamená, že při přidání nebo odebrání nové role k nějaké jiné roli způsobí, že všem uživatelům jsou oprávnění automaticky přidána nebo odebrána.

Implicitní role

V rámci realmu lze nastavit implicitní (*default*) role, které jsou automaticky přiřazeny nově vytvořeným uživatelům. Ti se vytváří jak prostřednictvím registrace tak pomocí administrační konzole.

5.2.6 Implicitní aplikace

V každém realmu jsou implicitně přítomny předpřipravené aplikace.

- security-admin-console

  Aplikace se stará o přihlašování do administrační konzole příslušného realmu. Neobsahuje žádné role, ale obsahuje *scopes*.

- account

  Account je aplikace, která obsahuje role *view-profile* a *manage-account*. Pokud jsou tyto role obsaženy v uživatelově mapování rolí, má oprávnění k zobrazení svého profilu nebo ke správě svého účtu, například změna hesla, e-mailu apod.

- realm-management

  Aplikace specifikuje množinu rolí, které slouží ke správě příslušného serveru. Poskytují oprávnění k:

  - *view-realm*: zobrazení realmu
  - *view-users*: zobrazení uživatelů, včetně informací specifických pro jednotlivého uživatele
  - *view-applications*: zobrazení aplikací realmu
  - *view-clients*: zobrazení klientů realmu
  - *view-events*: zobrazení událostí realmu
  - *manage-realm*: úpravě realmu, včetně smazání
  - *manage-users*: vytváření, editaci a mazání uživatelů v rámci realmu
  - *manage-applications*: vytváření, editaci a mazání aplikací v rámci realmu
  - *manage-clients*: vytváření, editaci a mazání klientů v rámci realmu
  - *manage-events*: zakázání / povolení událostí, mazání obsahu logů a spravování tzv. *listeners*.

Role určené k úpravě obsahují příslušné role potřebné k zobrazení. Tyto role mohou být přidány do složených rolí aplikací nebo realmu. Uživatel pak získá příslušná oprávnění ke správě nebo zobrazení realmu.

5.3 Hlavní přednosti Keycloak serveru

5.3.1 Log In mechanismus

Keycloak poskytuje kompletní přihlašovací mechanismus. Jakmile je uživatel autentizován v rámci realmu, je autentizován v samotné aplikaci. Aplikace nemusí poskytovat svůj vlastní přihlašovací formulář. Přihlašovací formulář Keycloaku je konfigurovatelný, je možné vytvořit kompletně vlastní šablonu, která bude zobrazována uživatelům.

5.3.2 Single Sign On

Jednotné přihlášení na úrovni realmu. Po přihlášení do realmu, je uživatel automaticky přihlášený do všech aplikací realmu. Analogicky funguje funkce Single Sign Out při odhlašování.

5.3.3 Registrace uživatelů

Proces registrace nového uživatele. Formuláře jsou nastavitelné a je možné je v případě potřeby vyměnit za vlastní.

5.3.4 Práce s uživatelským účtem

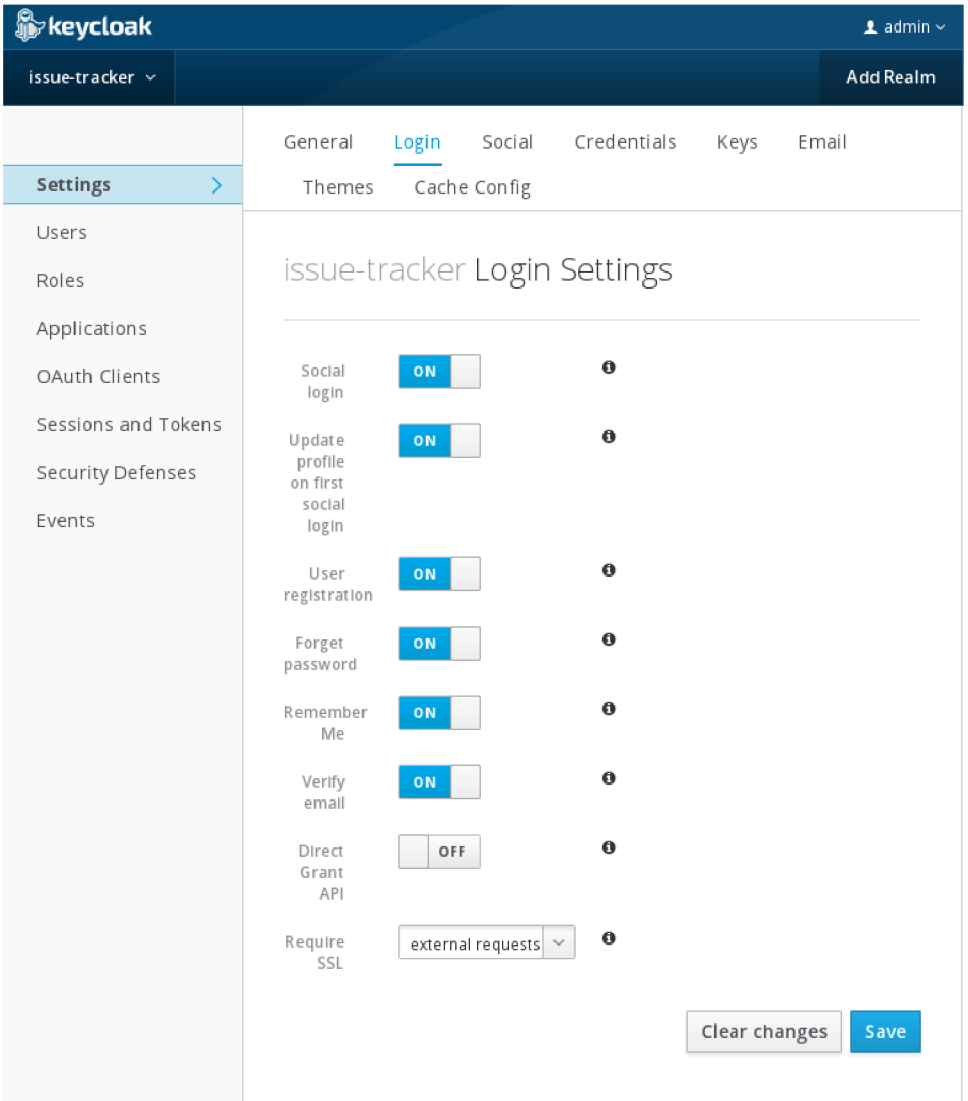
Server podporuje práci v uživatelským účtem uživatele, jako je obnova zapomenutého uživatelského jména nebo hesla, úprava informací v profilu, ověřování e-mailové adresy apod.

5.3.5 Správa serveru

Keycloak obsahuje rozsáhlé možnosti konfigurací a pro jejich nastavení poskytuje webovou administrační konzoli. Další možností jak spravovat server je prostřednictvím REST API. Na obrázku 5.1 je znázorněna ukázka webového rozhraní administrační konzole.

5.3.6 Vícefaktorová autentizace

Uživatelské jméno a heslo nemusí být v některých případech dostatečně silným mechanismem. Proto se používá vícefaktorová autenti-



Obrázek 5.1: Screenshot webové administrační konzole

zace. Keycloak poskytuje podporu pro zabezpečení prostřednictvím Google Authenticator.

5.3.7 Přihlášení přes sociální sítě

Keycloak podporuje přihlášení pomocí Google+[2], Facebook[3], Twitter[4] nebo GitHub[5] účtů. Další jako třeba LinkedIn[6] bude brzy přidán.

5.3.8 Podpora aplikací třetích stran

Zahrnuje podporu pro aplikace třetích stran prostřednictvím protokolu OAuth.

5.3.9 Adaptéry

Adaptéry jsou poskytovány pro snadné použití Keycloaku na různých aplikačních serverech a platformách. Aktuálně existuje podpora pro aplikační servery WildFly a JBoss EAP, nicméně jejich seznam se bude rozšiřovat.

5.3.10 Samostatná služba

Keycloak server je nasazován do aplikačního serveru, stejně jako jiné aplikace. Může běžet na stejném nebo jiném serveru, než aplikace, kterou zabezpečuje, nebo může být nasazen do prostředí Openshift[7]. To je velká výhoda z pohledu výkonu zabezpečené aplikace.

5.4 Autorizační proces

Keycloak používá protokol založený na OAuth2[8]. V momentě kdy uživatel otevře v okně webového prohlížeče zabezpečenou stránku,

2. https://plus.google.com/
3. https://www.facebook.com/
4. https://twitter.com/
5. https://github.com/
6. https://www.linkedin.com/
7. https://www.openshift.com/
8. http://oauth.net/2/

nebo použije přihlašovací odkaz aplikace, a ta jej přesměruje na rozhraní Keycloak serveru. Uživateli je zobrazena stránka obsahující přihlašovací formulář. V případě, že je uživatel již v rámci příslušného realmu pomocí Single Sign On přihlášen, formulář není zobrazen. Po úspěšné autentizaci a následné autorizaci je uživatel přesměrován zpět do aplikace a aplikace obdrží autorizační kód (*authorization code*). To je unikátní textový řetězec, který umožňuje aplikaci přístup k tzv. tokenu. Samotná aplikace nikdy nevidí uživatelovy přihlašovací údaje.

5.4.1 Token

Tokenem se rozumí to, co může aplikace použít k přístupu ke zdrojům. Skládá se z *JSON Web Token (JWT)*[9] a *JSON Web Signature (JWS)*[10] a navíc obsahuje dodatečné informace, například uživatelské jméno, role, čas expirace apod.

Server Keycloak tokeny podepisuje, aby je bylo možné pomocí veřejného klíče realmu lokálně ověřit. To má za následek, že je možné ověřit token bez potřeby komunikovat se serverem přímo. Aplikace posílá společně s požadavkem také token, který potvrzuje identitu uživatele bez nutnosti komunikovat s Keycloakem. Token je přidáván do hlavičky požadavku, který aplikace posílá. Služba zkontroluje token a ověří, že uživatel aplikace má na požadovaný zdroj dostatečné oprávnění.

5.4.2 Skryté heslo

Ani aplikace, ze které se posílají požadavky, ani služba, která požadavky vyřizuje, nikdy nevidí uživatelovy přihlašovací údaje. Veškerá komunikace se provádí prostřednictvím tokenu. Tento přístup je mnohem více bezpečný, navíc dává uživateli možnost změny například hesla bez toho, aby přistupoval k existujícím službám.

9. `http://jwt.io/`
10. `https://kjur.github.io/jsjws/`

6 Analýza a návrh

V úvodu kapitoly je popsán stav, ve kterém se systém pro správu požadavků nachází. Následuje analýza a návrh aplikace, grafické znázornění návrhu části aplikace je vytvořeno pomocí jazyka UML [10]. Jazyk UML je univerzální jazyk pro vizuální modelovaní systému. Práce používá dva diagramy. Prvním diagramem je diagram případu užití (use case diagram) [2], který zobrazuje příklad použití systému. Druhý diagram je diagram entit (ERD), který zobrazuje datový model zabezpečení aplikace [2].

6.1 Současné řešení systému pro správu požadavků

Systém umožňuje vytvářet projekty, které mají několik základních a několik rozšiřujících atributů. Mezi základní atributy patří identifikátor projektu, název, popis a vlastník projektu. Rozšiřující atributy projektu jsou komponenty, verze a pracovní tok (*workflow*). Dále je k projektu možné přidat libovolné množství dalších atributů (*customField*), které vlastník projektu bude potřebovat. Projekt má obvykle několik komponent a verzí, které jsou do systému vkládány společně s novým projektem.

V rámci vytvořeného projektu je možné vytvářet požadavky (*issue*). Issue je hlavní entita celého systému. Její atributy jsou `id`, název (`name`), popis (`description`), shrnutí (`summary`), typ (`issueType`), projekt (`project`), seznam komponent týkajících se konkrétní issue (`components`), seznam verzí, kterých se issue týká (`affectedVersions`), vlastník (`reporter`), priorita (`priority`), čas vytvoření (`createdTime`), čas úpravy (`modifiedTime`), uživatel, kterému je issue přiřazena (`assignee`), seznam uživatelů, kteří issue sledují (`watchers`), seznam dalších issue, které s touto issue souvisí (`relatedIssues`) a seznam komentářů k dané issue (`comments`).

Issue může být různého typu, například chyba (*Bug*), vylepšení (*Enhancement*), žádost o novou funkcionalitu (*Feature Request*), úkol (*Task*) apod. Systém umožňuje vytvářet libovolné typy issue. Každá issue se nachází v nějakém stavu. Například nová (*New*), přiřazená (*Assigned*), vyřešená (*Resolved*), uzavřená (*Closed*), nebude opravená (*Won't fix*) atd. Existující *issue-tracker* systém umožňuje vytvářet libo-

volné stavy a spojovat je do logického toku. Přechod z jednoho stavu do druhého je v systému řešený pomocí tzv. přechodu (`Transition`). Seznam těchto přechodů tvoří celé workflow. Těch může v systému existovat celá řada. Každý projekt je spojen s jedním workflow.

Pro zabezpečení aplikace byly vytvořeny entity oprávnění (`Permission`), role (`Role`), skupina (`Group`) a uživatel (`User`). Uživatelé patří do jedné nebo více skupin. Každá skupina uživatelů má přiřazené role. Role mají oprávnění k provádění různých operací.

6.2 Analýza

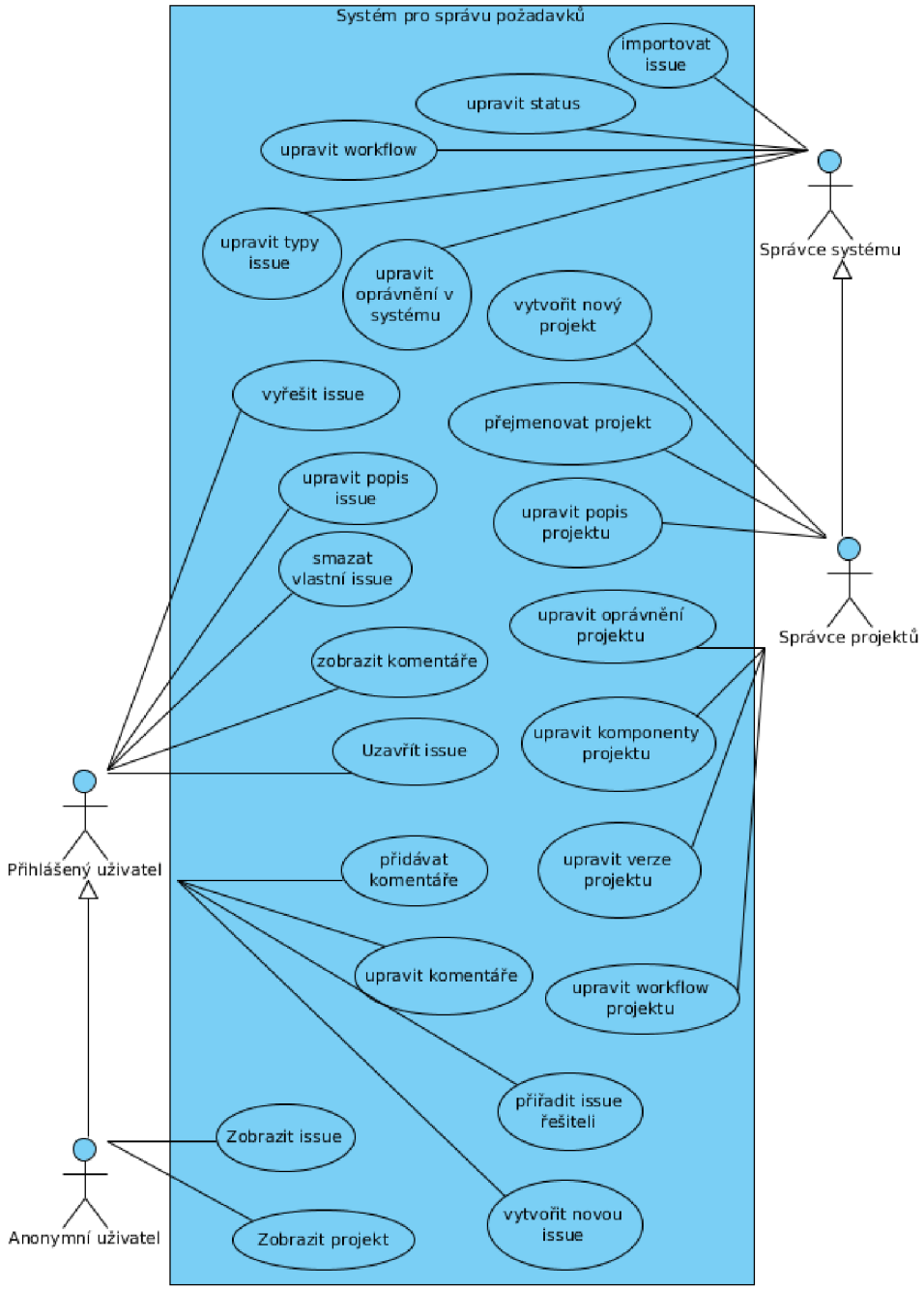
Vzhledem k tomu, že Keycloak poskytuje podporu pro správu uživatelů, jejich rolí a pomocí composite roles je možné vytvořit skupiny, bylo nutné přepracovat model, který byl navrhnut při vytváření jádra projektu. Entity User, Group, Permission a Role nevyhovovaly požadavkům pro vytvoření zabezpečení systému pomocí projektu Keycloak, a proto byly ze systému odstraněny.

6.2.1 Use Case diagram

Během analýzy systému bylo identifikováno, že každý uživatel může v aplikaci provádět různé operace, které mohou být stejné nebo různé u jednotlivých entit systému. Příklad takových akcí ukazuje diagram případů užití na obrázku 6.1. Některé operace souvisí s konkrétním objektem. Například zobrazit projekt, upravit verze projektu, komponenty, popis projektu, vytvářet issue, mazat projekt a další souvisí s konkrétní instancí projektu. Zobrazit issue, upravovat popis issue, měnit stav issue, přidávat nebo měnit komentáře zase souvisí s instancí issue. Ostatní operace jsou vázány na systém samotný, tj. nesouvisí s žádným konkrétním objektem. Takové operace jsou například upravit workflow, status, importovat projekty, upravovat typy issue nebo vytvářet projekty.

Na základě toho byly navrhnuty čtyři typy úrovně oprávnění, které se v systému použijí:

- global
- project



Obrázek 6.1: Diagram případů užití

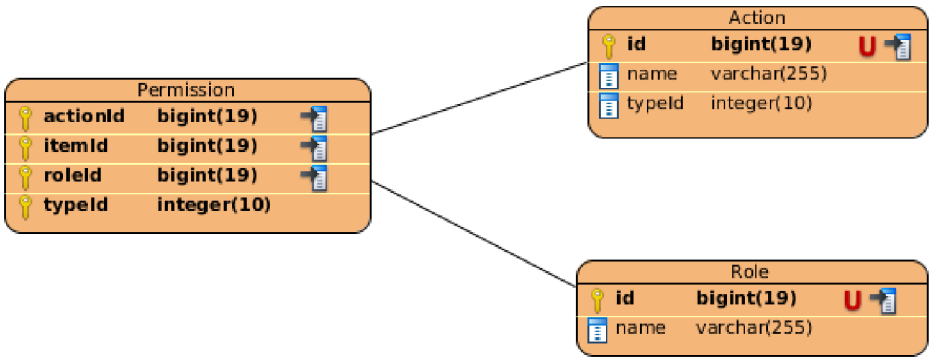
- issue
- comment

Pro jednodušší správu byla navrhnuta hierarchie oprávnění v issue tracker systému. Základní oprávnění jsou úrovně *global*. Pokud akce souvisí s projektem, na úrovni oprávnění *project*, je možné příslušnou akci povolit i pro další roli nebo role. Lze případně povolit akci pro další role na úrovni *issue* nebo *comment*, v případě, že se daná akce vztahuje k issue, resp. ke komentáři.

6.2.2 Datový model

Existují tři entity, které jsou součástí zabezpečení aplikace, navíc se používá výše zmíněný typ oprávnění. Jejich atributy, primární klíče a indexy jsou zobrazeny v diagramu entit 6.2.

- Role
- Action
- Permission



Obrázek 6.2: Diagram entit

V Keycloaku může mít každý uživatel přidělenu jednu nebo více rolí. Aby bylo možné s těmito rolemi pracovat v issue tracker systému, byla přidána entita Role, která má atributy id a název role. Ta

reprezentuje převodník mezi rolemi, které jsou uloženy v Keycloaku a číselnou reprezentací, která se ukládá v systému pro správu požadavků.

Action reprezentuje operace v aplikaci, jejich příklad je uveden v kapitole 6.2.1.

TypeId je výčtový typ, do relační databáze se ukládají číselné hodnoty nula až tři. Číselná reprezentace je mnohem vhodnější než textová. Případná změna jména nevede k úpravě záznamů v celé databázi. Převod je prováděn následujícím způsobem:

```
public enum TypeId {

    global(0),
    project(1),
    issue(2),
    comment(3);

    private final int value;

    private TypeId(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }

    public static TypeId parse(Integer id) {
        TypeId typeId = null;
        for (TypeId item : TypeId.values()) {
            if (item.getValue() == id) {
                typeId = item;
                break;
            }
        }
        return typeId;
    }
}
```

Nejdůležitější částí datové vrstvy zabezpečení aplikace je entita Permission, která má čtyři atributy typeId, itemId, actionId a roleId. Jedna instance pak reprezentuje jeden záznam o oprávnění. Například pokud má role "RedHat Employee" oprávnění editovat jméno projektu s názvem "Projekt A", pak existuje záznam v tabulce Permission.

Role	
id	*name*
10	RedHatEmployee
...	...

Action		
id	*name*	*typeId*
0	Rename Project	1
...	...	...

Project		
id	*name*	...
20	Project A	...
...	...	...

Permission			
typeId	*itemId*	*actionId*	*roleId*
1	20	0	10
...	...	...	...

Pro oprávnění zobrazování jednotlivých objektů byla z důvodu škálovatelnosti zvolena jiná strategie ukládání záznamů v databázi. Ukládány jsou pouze záznamy pro objekty, které jsou neveřejné. Pokud neexistuje žádný záznam například pro akci "View Comment" pro žádnou roli, znamená to, že oprávnění podléhá hierarchii práv vyšší vrstvy. V případě, že pro danou roli má být na úrovni *global*,

project nebo *issue* operace povolena, pak je komentář viditelný a může být zobrazen uživateli na stránce. V případě, že je potřeba komentáři nastavit, že je neveřejný, pak se ignoruje hierarchie oprávnění a použijí se pouze příslušné záznamy. Tento přístup je pak analogicky použit i u dalších objektů v rámci issue tracker systému jako issue a projekt. Díky tomuto přístupu se značně zredukuje počet záznamů v databázi.

7 Implementace

Kapitola na svém začátku popisuje různé součásti zabezpečení systému rozdělené na logické části, V další části kapitoly jsou představeny potřebné konfigurační kroky jak na straně Keycloak serveru, tak na straně systému pro správu požadavků. Kapitola uvádí technologie, které jsou při vývoji modulu zabezpečení použity. Na konci kapitoly je představeno uživatelské rozhraní části issue tracker systému týkající se zabezpečení.

7.1 Servisní vrstva

V rámci servisní vrstvy v balíku `com.issuetracker.service` byly vytvořeny třídy, které mají za úkol, připravit data, která jsou poskytována datovou vrstvou, pro prezentační vrstvu, tak ji zobrazí uživateli. Rozhraní `PermissionService` obsluhuje instance `Permission`. Poskytuje metody pro vkládání nových oprávnění `createPermissions(TypeId typeId, Object item, String actionName, TypeId actionType, String... roleNames)`, aktualizaci záznamů oprávnění `update(TypeId typeId, Long itemId, Long roleId, List<Permission> permissions)` a k poskytnutí oprávnění podle různých kritérií, které jsou potřeba v prezentační vrstvě, existuje sada metod `getPermissionsBy...`. `SecurityService` obsahuje dvě metody. Jedna je určena pro ověření, zda má aktuálně přihlášený uživatel oprávnění provádět nějakou konkrétní operaci: `canUserPerformAction(TypeId typeId, Long itemId, String actionName)`. Druhá metoda vrací seznam povolených operací, které může uživatel na určitém objektu provádět `getPermittedActionsForUserAndItem(TypeId typeId, Long itemId)`. Mimo tyto dvě hlavní servisní třídy jsou vytvořeny další servisní třídy `ActionService` a `RoleService`, které obsluhují příslušné entity `Action` a `Role`.

7.2 Integrace s Keycloak API

Pro práci s Keycloak serverem bylo vytvořeno několik tříd, které jsou umístěny v balíku `com.issuetracker.web.security`. Pro přímou ko-

munikaci se serverem existuje třída `KeycloakService`, která používá technologii založenou na REST API a vrací řetězcovou reprezentaci (`String`) všech dostupných rolí. Pro úspěšnou komunikaci je potřeba přidat do hlavičky požadavku řetězcovou reprezentaci tokenu uživatele. Třída `KeycloakAuthSession` reprezentuje kontext aktuálně přihlášeného uživatele. Pomocí Keycloak API je vrácena instance třídy `org.keycloak.KeycloakSecurityContext`, která obsahuje metody užitečné v systému pro správu požadavků. `KeycloakAuthSession` obsahuje sadu statických metod pro práci s přihlášeným uživatelem: `isSignedIn()`, `getIDToken()`, `getUserRealmRoles()`, `isUserInRhelmRole(String role)`, `getUserAppRoles()`, a `isUserInAppRole(String roleKey)`. V balíku `com.issuetracker.web.security` se nachází properties soubor `roles.properties`, který obsahuje názvy operací v systému a jejich klíče. Ty jsou staticky načítány ve třídě `com.issuetracker.Constants` a použity jako textová reprezentace v kódu. Toto řešení značně zvyšuje udržitelnost kódu. Při případné změně názvu operace lze upravit pouze záznam v jediném souboru. Další záznamy do tohoto souboru mohou být přidány, pro potřeby doplňování nebo rozšiřování funkcionality aplikace.

7.3 Prezentační vrstva

Třída `com.issuetracker.web.IssueTrackerSession` rozšiřuje třídu `org.apache.wicket.protocol.http.WebSession` a přidává k ní funkcionalitu jednoduché cache, kterou lze v případě potřeby vyměnit za jinou, sofistikovanou cache, například Infinispan[1]. Díky tomuto řešení je ukládána informace zda má uživatel oprávnění zobrazit konkrétní stránku a není nutné se opakovaně dotazovat a tím zatěžovat systém. Třída `PageLayout`, kterou rozšiřují všechny stránky v aplikaci, má mimo jiné za úkol rozlišovat, jestli je daný uživatel oprávněn stránku zobrazit. V konstruktoru se používá `IssueTrackerSession`. Při procesu rozhodování o oprávnění zobrazit konkrétní stránku, je použita vlastní anotace `@ViewPageConstraint`, která se nachází v balíku `com.issuetracker.web.quilifiers`.

1. `http://infinispan.org/`

```
@Retention(RUNTIME)
@Target(ElementType.CONSTRUCTOR)
public @interface ViewPageConstraint {
    String allowedAction() default "";
    String allowedRole() default "";
}
```

Tato anotace je použita u konstruktoru stránky, například:

```
public class CreateProject extends PageLayout {
    ...
    @ViewPageConstraint(allowedAction =
        "it.project.create")
    public CreateProject() {
        ...
    }
}
```

Pomocí Java Reflection API je informace z anotace extrahována a použita pro rozhodování, jestli má uživatel oprávnění stránku zobrazit. Pokud anotace přítomna není, rozhoduje se na základě konkrétních oprávnění.

Během implementace zabezpečení systému, bylo potřeba upravit všechny existující stránky v systému. Byly skryty elementy na stránce, které poskytují informace nebo dovolují provádět operace, na něž uživatel nemá oprávnění.

```
public class ProjectDetail extends PageLayout {
    ...
    @Override
    public void onConfigure() {
        super.onConfigure();

        renameForm.setVisible(permittedActions.contains(
            roles.getProperty("it.project.rename")));

        projectVersionForm.setVisible(
            permittedActions.contains(
```

```
            roles.getProperty("it.project.versions")));
        ...
    }
...
}
```

Při odesílání formulářů je přidána kontrola oprávnění. Například:

```
public class ProjectDetail extends PageLayout {
    ...
    Link deleteProject = new Link("delete") {

        @Override
        public void onClick() {
            if (!permittedActions.contains(
                roles.getProperty(it.project.delete))) {
            setResponsePage(AccessDenied.class);
        }
        ...
        }
    };
    ...
}
```

7.4 Nastavení systémů

7.4.1 Nastavení Keycloak serveru

Aby bylo možné použít Keycloak server pro zabezpečení aplikace je potřeba nejprve vytvořit aplikaci v rámci realmu Keycloak serveru. To lze provést pomocí REST API, nebo pomocí webového grafického uživatelského rozhraní tzv. *admin console*. Pro potřeby issue tracker systému byl vytvořen nový realm "issue-tracker". Ten je nakonfigurován podle požadavků zadání a jeho nastavení lze následně upravit. Zejména je povolen tzv. *social login*, který umožňuje přihlášení pomocí sociálních sítí Google+, Facebook, GitHub a Twitter. Dále byla

povolena registrace nových uživatelů, zapamatování uživatele v konkrétním prohlížeči a mechanismus v případě zapomenutí hesla. Pro zasílání e-mailových zpráv je potřeba nastavit SMPT server.

V realmu jsou připraveny dvě základní role. Administrátor issue tracker systému může přidávat libovolné role, které jsou potřebné pro konkrétní instanci systému. Role User je implicitní (*default*) role, která je přiřazena všem novým uživatelům systému. User je *composite role* a obsahuje oprávnění pro zobrazení profilu a správu účtu každého uživatele. Nastavení lze dle potřeby měnit. V rámci realmu je vytvořena aplikace rovněž s názvem *issue-tracker*. Zde je nastaveno URL, na kterém běží systém pro správu požadavků, pro potřeby diplomové práce je nastavena adresa http://localhost:8080/IssueTracker. Při přechodu do produkčního prostředí je možné nastavení změnit. V rámci aplikace existuje role Admin, která slouží pro administrátory issue tracker systému. Má oprávnění k administraci *issue-tracker* realmu. Pro aplikaci *issue-tracker* je vygenerován soubor s názvem keycloak.json, který je potřeba pro konfiguraci systému pro správu požadavků.

7.4.2 Nastavení systému pro správu požadavků

Pro zabezpečení aplikace není nutné provádět rozsáhlé konfigurační změny. Vygenerovaný soubor keycloak.json je vložen do adresáře projektu /src/main/webapp/WEB-INF/ a je upravena <auth-method> v rámci <login-config> ve web.xml z BASIC na KEYCLOAK.

```
<login-config>
    <auth-method>KEYCLOAK</auth-method>
    <realm-name>issue-tracker</realm-name>
</login-config>
```

7.5 Uživatelské rozhraní

Do systému byla potřeba vyvinout novou sadu wicket stránek, které administrátorům systému poskytují nástroje pro práci s oprávněními. Tato funkcionalita se nachází v balíku `com.issuetracker.pages.per-`

`missions`. Obsahuje třídu `AccessDenied`, která slouží pro zobrazení informace uživateli, že nemá pro danou operaci oprávnění. Třída `GlobalPermission` poskytuje možnost nastavit globální oprávnění, analogicky třída `ProjectPermission` projektové oprávnění a třída `IssuePermission` oprávnění související s issue. Obrázek 7.1 ukazuje úpravu globálních oprávnění.

7.6 Použité technologie

7.6.1 Java Persistence API

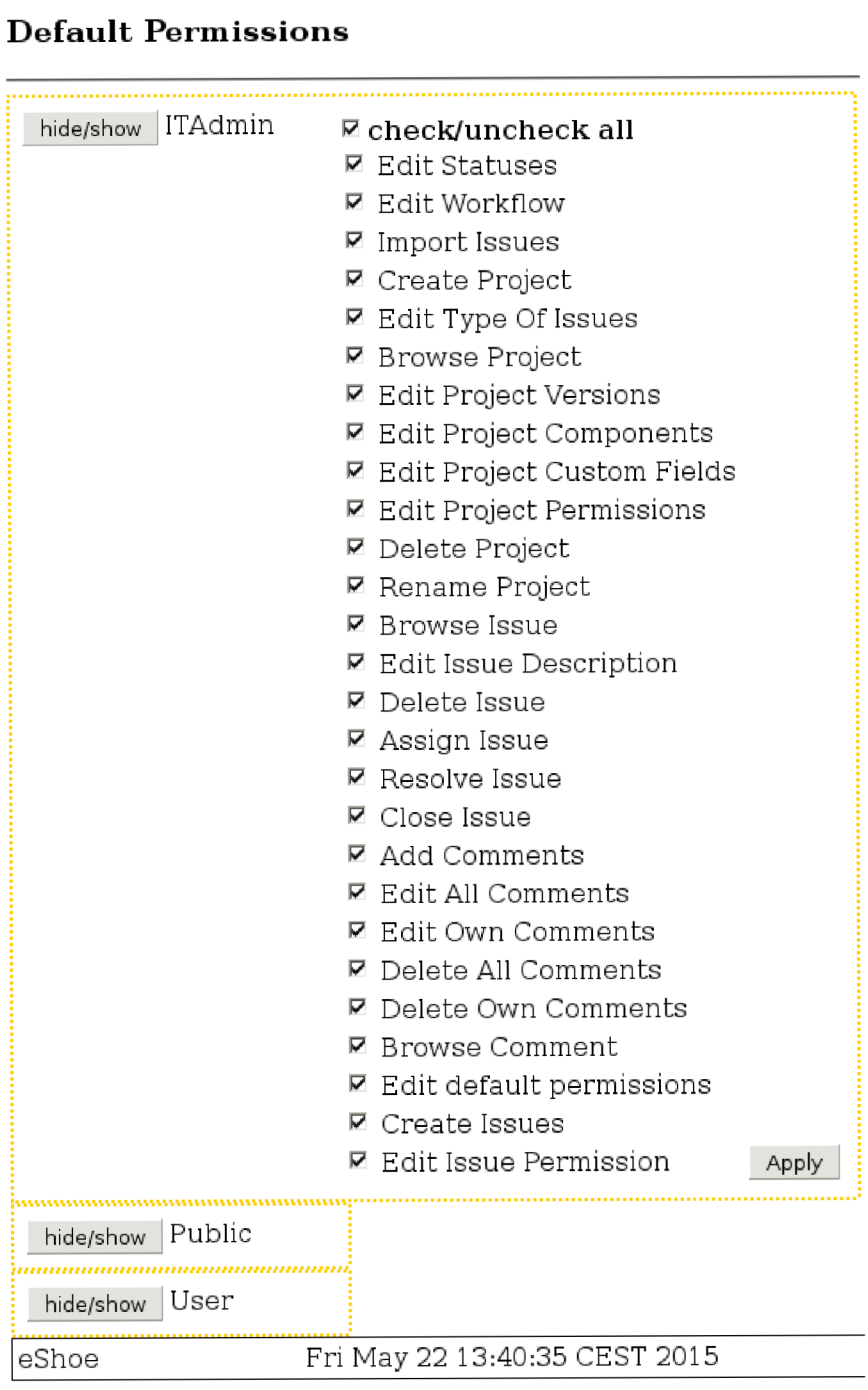
Datová vrstva systému pro správu požadavků používá Java Persistence API, což je Java EE standard pro persistenci dat pomocí objektově relačního mapování (ORM), tj. technika automatického převodu objektů objektově orientovaného jazyka na data relační databáze. Java objekty, které reprezentují data, jsou tzv. POJO objekty (Plain Old Java Object). Ty pro svůj běh nepotřebují kontejner, který by se staral o jejich životní cyklus. Je možné s nimi pracovat stejně jako se standardními Java objekty. Objekty jsou pomocí anotací, které poskytují dodatečné informace o způsobu mapování, ukládány do relační databáze.

7.6.2 Criteria API

Pro vytváření dotazů do relační databáze je použita technologie Criteria API. Criteria API je založena na dotazech, kde jsou definovány objekty. Dotazy jsou vytvářeny pomocí API programovacího jazyka Java, jsou typově bezpečné a přenositelné. Jejich výhoda je, že fungují nezávisle na datovém úložišti. JPA Criteria API nabízí alternativu k SQL nebo JPQL. Výhodou je dynamické definování dotazů, protože závisí na struktuře, která je známá až za běhu aplikace.

7.6.3 Wicket

Prezentační vrstva zobrazuje informace uživateli pomocí webových stránek. Aplikace k tomu používá webový framework Wicket. Wicket je založen na programovacím jazyku Java a používá komponentový model. Webové stránky a komponenty jsou POJO objekty, které podporují zapouzdření, dědění a události. Wicket se používá společně se



Obrázek 7.1: Ukázka uživatelského rozhraní

základním HTML jazykem, který je rozšířený pomocí jmenných prostorů (*namespace*), což je v souladu s XHTML standardem. Zdrojový kód psaný pomocí wicketu je znovupoužitelný. Wicket je vhodný z pohledu zabezpečení, v URL nejsou obsaženy citlivé informace a všechny cesty ke komponentům se vážou ke konkrétní session. Ke sdílení informací mezi session je potřeba provést specifické konfigurační kroky. Wicket je zdarma a je šířený pod licencí Apache License, Version 2.0.

7.6.4 RESTové služby

Representational State Transfer (REST) je architektura, která umožňuje pomocí HTTP volání číst nebo editovat informace nebo služby na serveru. Práce s RESTem poskytuje jednotné rozhraní. Zdrojem (*resource*) jsou označena data i funkcionalita, ke zdroji je přistupováno prostřednictvím URIs (Uniform Resource Identifiers), což jsou například webové odkazy. REST je navrhnut k použití bezstavového protokolu HTTP. Pomocí RESTových služeb je umožněna komunikace mezi systémem pro správu požadavků a Keycloak serverem.

7.6.5 CDI

Context and Dependency Injection (CDI) je standard programovacího jazyka Java EE pro vkládání závislostí (DI). CDI umožňuje, aby byly třídám závislosti poskytovány automaticky bez nutnosti vytvářet jednotlivé objekty manuálně nebo je předávat pomocí parametrů. Vkládání závislostí je typově bezpečné a pro určení, které závislosti mají být vloženy, se používá anotace `@Inject`.

7.6.6 Keycloak

Pro zabezpečení systému je použit projekt Keycloak, viz kapitola 5.

7.6.7 Elasticsearch

Elasticsearch je distribuovaný vyhledávácí a analytický nástroj, který pracuje v reálném čase. V systému pro správu požadavků se používá pro fulltextové vyhledávání.

8 Závěr

Hlavním cílem této práce bylo navrhnout a implementovat mechanismy pro autentizaci a autorizaci do existujícího systému pro správu požadavků eShoe, za pomoci vhodné technologie. Touto se na základě analýzy stal Keycloak. Prozkoumání bezpečnostní rysů existujících issue tracking systémů přispělo k zařazení několika vhodných feature do návrhu zabezpečení aplikace.

Díky použití Keycloaku, který je možně jednoznačně považovat za moderní technologii, bylo dosaženo požadovaných kritérií na zabezpečení. Jedná se zejména o skutečnost, že projekt je vyvíjen jako open source, takže je velká pravděpodobnost, že kvalita kódu bude vysoká, dále Keycloak nabízí možnost propojení se sociálními sítěmi, navíc poskytuje přehledné uživatelské rozhraní pro správu realmu (uživatelů a jejich rolí, aplikací atd.) a v neposlední řadě poskytuje tyto funkce jako externí službu, což přináší řadu výhod. Mezi ně můžeme zařadit například to, že při velkém počtu požadavků na naši aplikaci (systém), Keycloak může být nasazen na jiném serveru a tím pádem nezatěžuje hardware původního serveru, což v konečném důsledku zvyšuje výkon.

Vzniklé řešení je implementováno pomocí jazyka Java EE, jsou kromě projektu Keycloak byly použity technologie JPA, CDI, Wicket, REST, Java Reflection API. Ve všech případech se jedná o standardní, ověřené a široce používané technologie, což přispělo ke kvalitě řešení.

Zdrojové kódy projektu jsou přístupné v systému správy verzí GitHub, dostupné k volnému stažení a používání a výsledné řešení splňuje všechny prvky zadání. Projekt se již od počátku zaměřil na to, aby vytvořená funkcionalita mohla být zapouzdřena, a tedy jednoduše vyjmutelná za účelem vytvoření samostatné knihovny, která poskytuje sadu operací pro zabezpečení aplikací pomocí Keycloaku. Díky této knihovně bude zabezpečení aplikací, které k tomu budou používat Keycloak, mnohem jednodušší.

Projekt má pochopitelně i svá slabá místa a možnosti k dalšímu rozšíření. Za slabé místo by se dalo označit nedokonalé uživatelské rozhraní (GUI), které by zasluhovalo mnohá vylepšení, přesněji řečeno celkové kvalitní zpracování.

Další možností rozšíření je vytvoření samostatné knihovny, za-

pouzdřením funkcionality zabezpečení, která se nachází na servisní vrstvě, a její poskytnutí dalším vývojářům k širšímu použití.

Literatura

[1] ATTLASSIAN. *Configuring Security*, [online], [cit.dne 21.5.2015], https://confluence.atlassian.com/display/JIRA/Configuring+Security

[2] ARLOW, Jim a NEUSTADT, Ila. *UML2 a unifikovaný proces vývoje aplikací*, AddisonWesley, 2007, ISBN 978-80-251-1503-9.

[3] CHINNICI, Roberto a SHANNON, Bill. *JSR316: Java Enterprise Edition 6 Specification*, [online], 2009, [cit. dne 10.5.2015], http://www.jcp.org/en/jsr/detail?id=316

[4] EDGEWALL SOFTWARE. *Trac Permissions*, [online], [cit.dne 20.5.2015], http://trac.edgewall.org/wiki/TracPermissions

[5] FOG GREEK, *FogBugz On Demand Infrastructure*, [online], [cit. dne 23.5.2015], http://www.fogcreek.com/fogbugz/on-demand-infrastructure/

[6] GOTTVALDOVÁ, Monika. *Modern open source Java EE-based process and issue tracker*, Diplomová práce, Brno: Fakulta informatiky Masarykovy univerzity, 2014.

[7] JBOSS.ORG. *PicketLink and Keycloak projects are merging!*, [online], [cit. dne 15.5.2015], http://picketlink.org/news/2015/03/10/PicketLink-and-Keycloak-project-merge/

[8] MCREE, Russ. *Implementing Redmine for Secure Project Management*, [online], [cit.dne 21.5.2015], https://www.sans.org/reading-room/whitepapers/testing/implementing-redmine-secure-project-management-34122

[9] MOZILLA FOUNDATION. *Bugzzila features, security*, [online], [cit. dne 20.5.2015], https://www.bugzilla.org/features/#security

[10] Object Management Group. *OMG Unified Modeling Language, Infrastructure*, [online], 2010, [cit. dne 17.5.2015], http://www.omg.org/spec/UML/2.3/

[11] RED HAT. *JBoss PicketBox*, [online], [cit. dne 20.5.2015], http://picketbox.jboss.org/

[12] RED HAT. *JBoss PicketLink*, [online], [cit. dne 20.5.2015], http://www.picketlink.org/

[13] RED HAT. *JBoss WildFly application server*, [online], [cit. dne 20.5.2015], http://wildfly.org/

[14] RED HAT. *WildFly Documentation, Security subsystem configuration*, [online], [cit. dne. 21.5.2015], https://docs.jboss.org/author/display/WFLY8/Security+subsystem+configuration

[15] RED HAT. *WildFly Documentation, Security realms*, [online], [cit. dne. 21.5.2015], https://docs.jboss.org/author/display/WFLY8/Security+Realms

[16] THE MANTISBT TEAM. *Mantis Bug Tracker Administration Guide*, [online], [cit.dne 20.5.2015], http://www.mantisbt.org/docs/master-1.2.x/en/administration_guide.html

[17] TRAGER, Corey. *BugTracker.NET Documentation, User Permissions*, [online], [cit.dne 21.5.2015], http://ifdefined.com/doc_bug_tracker_permissions.html

[18] WAI CHAN, Shing a MORDANI, Rajiv. *JSR340: Java Servlet 3.1 Specification*, [online], 2013, [cit. dne 11.5.2015], http://www.jcp.org/en/jsr/detail?id=340

[19] *Netsparker identifies a High Risk XSS Web Vulnerability in Popular Redmine*, [online], [cit. dne 22.5.2015], https://www.netsparker.com/XSS-vulnerability-in-Redmine/