Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Bakalářská práce

Implementace bezpečnostní vrstvy DCIx

Plzeň, 2013 Ladislav Račák

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne ……

……

Ladislav Račák

Poděkování

Rád bych poděkoval zejména panu Janu Brnkovi a Martinu Augustovi, kteří dohlíželi nad celým projektem a měli se mnou trpělivost. Dále paní doc. Dr. Ing. Janě Klečkové, která je garantkou oboru informační systémy. V neposlední řadě také rodině a přátelům, kteří mi dodávali potřebnou podporu ve studiu.

Abstrakt

Cílem této práce je reimplementovat bezpečnostní vrstvu v aplikace DCIx od společnosti Aimtec, a.s. Nejprve je představena aplikace DCIx a původní řešení bezpečnosti. Dále je popsána knihovna Apache Shiro, její struktura a hlavní části. Vysvětlení principu fungování a konfigurace této knihovny. Popis některých základních pojmů z oblasti bezpečnosti týkající se zejména autentifikace a autorizace. Dále jsou porovnávána ostatní dostupná řešení pro reimplementaci bezpečnostní vrstvy. Nakonec je pospáno, jak byla autentifikace a autorizace vyřešena pomocí knihovny Apache Shiro v aplikace DCIx.

Klíčová slova: bezpečnost, Apache Shiro, autorizace, autentifikace, DCIx

Abstract

The objective of this study is to reimplement a security layer in DCIx application created by Aimtec, a. s. First is introduced the DCIx application and the original security solutions. Then it is described how Apache Shiro library works, how are configured its main features and domains. Then follows the description of some basic concepts especially authentication and authorization. Furthermore the Apache Shiro is compared to other available solutions for reimplementation of security layer. At the end it describes how the authentication and authorization were resolved by the Apache Shiro in DCIx application.

Key words: security, Apache Shiro, authorization, authentication, DCIx

Obsah

1. Úvod

Bezpečnost je jedna z klíčových oblastí v každé aplikaci. Zejména interní zabezpečení dat je ve víceuživatelské aplikaci velmi důležité. Nesmí se umožnit, aby uživatel mohl prohlížet nebo dokonce editovat data, na která nemá právo. Firmě využívající aplikaci s takovou chybou by mohly vzniknout velké ztráty. Proto na trhu existuje celá řada nástrojů a knihoven usnadňující řešení zmíněné problematiky. Tato práce je zaměřena především na knihovnu Apache Shiro a její reálné nasazení v aplikaci DCIx od společnosti Aimtec, a. s.

V první kapitole je představena firma Aimtec, a.s. (respektive AIMTEC Outsourcing s.r.o., kam se přesunul vývoj) a její hlavní softwarový produkt pro skladové hospodářství – aplikace DCIx. Popisuje se uživatelské rozhraní a způsob interagování s touto aplikací. Je uvedeno nutné běhové prostředí a komponenty aplikace, stejně jako zažitá konvence pro psaní zdrojových kódů a testování aplikace. V podkapitolách se práce věnuje analýze stávajícího způsobu autentifikace a autorizace uživatelů. Jsou představeny systémové role sloužící zejména k administraci a nakonfigurování aplikace u firmy, která si produkt zakoupila.

Druhá kapitola je věnována popisu bezpečnostní knihovny Apache Shiro, na kterou je tato práce téměř výhradně zaměřena. Ve stručnosti je uvedena historie, důvod vzniku a její tvůrci. Jsou popsány hlavní principy a myšlenky fungování celé knihovny, možnosti konfigurace a přizpůsobení specifickým požadavkům různých aplikací. Hlavním oblastem (autentifikace, autorizace a session management) jsou věnovány jednotlivé podkapitoly.

Následující kapitola stručně porovnává ostatní dostupná řešení pro implementaci bezpečnostní vrstvy. Popisuje JAAS, jakožto součást standardní edice Java od verze 1.4, a knihovnu Spring Security.

Další kapitola již popisuje samotnou implementaci bezpečnostní vrstvy. Nejprve je uvedeno, jak probíhala spolupráce a podmínky firmy Aimtec, a.s. Dále je představena jednoduchá ukázková aplikace využívající knihovnu Apache Shiro. Uživatelská dokumentace k této ukázkové aplikaci je přiložena k tomuto dokumentu ve formě

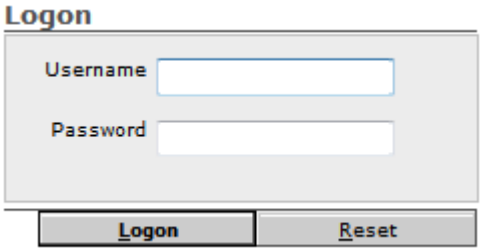
přílohy. Pak již následuje samotná reimplementace dosavadní bezpečnostní vrstvy v aplikaci DCIx. Nejprve je nahrazen dřívější komplikovaný způsob přihlašování novým s využitím metod a principů z knihovny Apache Shiro. Dále je přepracováno ověřování dostupných oprávnění pro uživatele. Kvůli eliminaci kontroly práv na jméno role, musela být nadefinována a roztříděna nová oprávnění. Vznikl nový autorizační servlet filtr pro kontrolu požadavků na server. Nakonec je popsáno, jak se nově kontrolují oprávnění v Java kódu a v JSP souborech.

2. Aplikace DCIx

Aplikace DCIx(1) je jedním z hlavních softwarových produktů firmy Aimtec, a.s. Společnost Aimtec, a.s. působí na trhu od roku 1996, avšak v roce 2008 došlo k vytvoření dceřiné společnosti AIMTEC Outsourcing s.r.o., kam se přesunul vývoj této aplikace. Firma se dále zabývá například poradenstvím v IT odvětví, vývojem vlastních SW nástrojů či systémovou integrací.

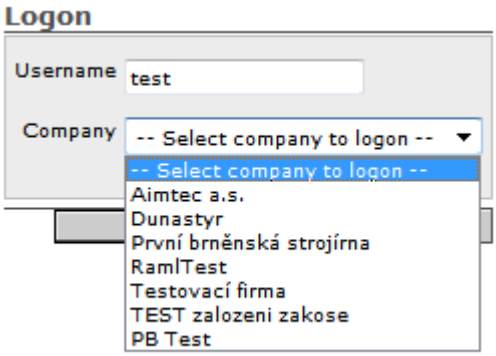
DCIx je informační systém pro výrobní, logistické nebo distribuční společnosti. Obsahuje moduly pro Warehouse Management System (WMS) – systém skladového hospodářství, Just In Time (JIT) – metoda optimalizace skladu založena na principu dodávek právě včas, Manufacturing Execution System (MES) - výrobní informační systémy, apod. Mezi nejvýznamnější klienty patří například: DENSO MANUFACTURING CZECH s.r.o., Pivovary Staropramen a.s., ŠKODA TRANSPORTATION a.s. a další. (2)

Aplikace umožňuje přihlášení pod různými uživatelskými účty s různými rolemi a oprávněním. Přihlášení může probíhat z klasického webového prostředí nebo z terminálu skladového pracovníka. V nedávné době dokonce získal systém i dotykové rozhraní. Přihlášení probíhá standardní kombinací jména a hesla, viz obrázek 1.



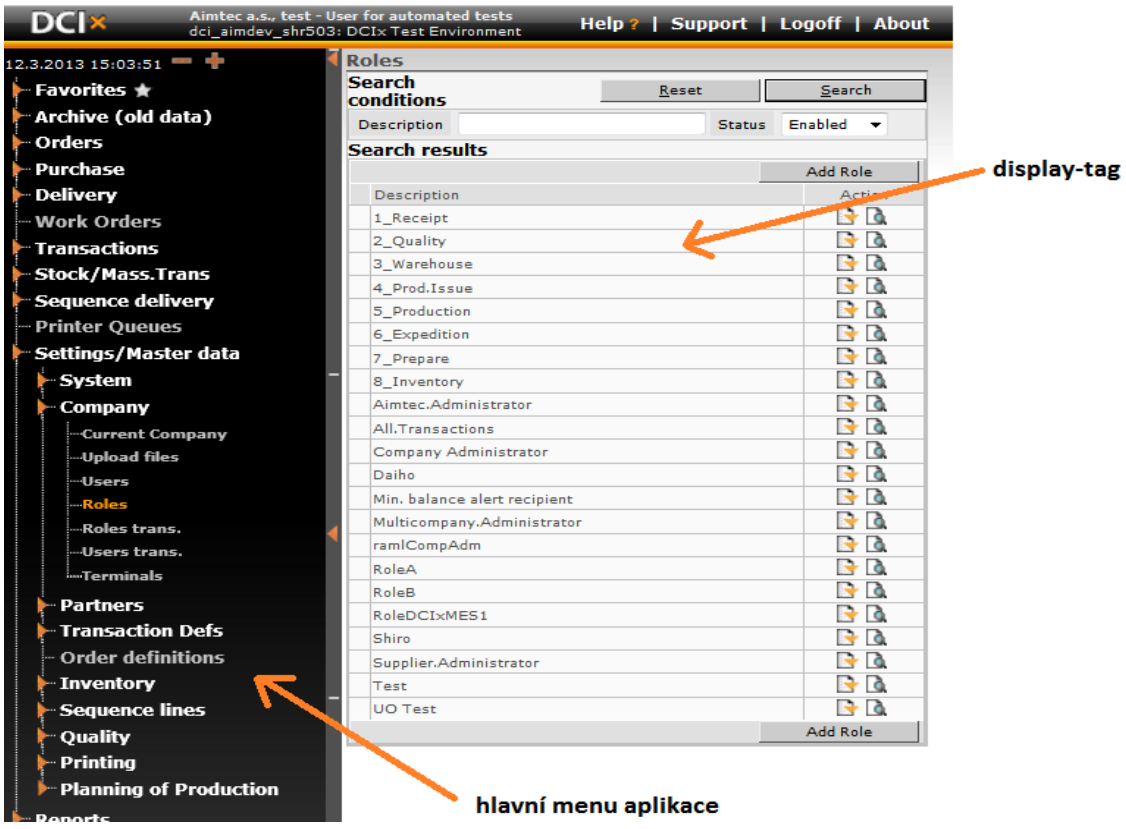
Obrázek 1 Formulář pro přihlášení uživatele do aplikace DCIx

Pokud je uživatel přiřazen ke speciální roli "*multicompany administrator*", zobrazí se ještě před samotnou aplikací rozbalovací menu pro výběr společnosti, ke které bude daný uživatel přihlášen, viz obrázek 2.



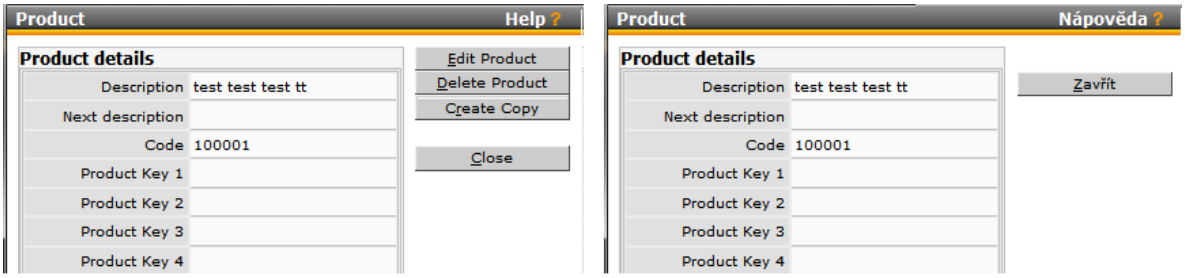
Obrázek 2 Výběr společnosti pro přihlášení

Pokud je ověření jména a hesla úspěšné, zobrazí se úvodní okno aplikace. K základní orientaci slouží levé menu. Po kliknutí na položku v menu se zobrazí v hlavním okně takzvaný "*display-tag*", zobrazující podrobnosti a různé formuláře (většinou pro vyhledávání). Po kliknutí na položku v "*display-tagu*" se otevře nové okno s detaily položky. Prostředí aplikace je znázorněno na obrázku 3. V Menu jsou zobrazeny jen ty položky, na které má daný uživatel právo. Předchozí implementace zabezpečení nekontrolovala, zda skutečně uživatel, který chce stránku zobrazit, má právo danou akci provést. To umožňovalo přistupovat všem přihlášeným uživatelům prakticky na jakoukoliv adresu v aplikaci, pokud ji znali.



Obrázek 3 Prostředí aplikace DCIx

Po kliknutí na položku v "*display-tagu*" se otevře nové okno s detaily a často s funkčními tlačítky jako je editace, smazání, klonování apod. Podmínkou zobrazení jednotlivých tlačítek byla specifická role. Na obrázku 4 lze vpravo vidět detail položky jen s právem prohlížet, vlevo je uživatel přiřazen k roli, která umožňuje editaci objektu.



Obrázek 4 Zobrazení detailu produktu pro různé role.

Aplikace DCIx je vyvíjena na platformě Java EE s použitím Java servlet pages (JSP). Používá databázi Microsoft SQL Server 2008 R2 s objektově relačním mapováním, které v aplikaci provádí Alfa framework. DCIx využívá celou škálu knihoven a frameworků například: Structs jako hlavní framework pro vývoj aplikace; HtmlUnit, Jameleon, JUnit pro testování; xmlParser, Serializer pro snadnější práci se soubory. O organizaci knihoven se částečně stará Maven (software pro řízení projektu a jeho závoslostí), v budoucnu se plánuje jeho plné nasazení. Pro běh serverové části se využívá server Apache Tomcat ve verzi 6, pro klientskou část je doporučován jen Internet Explorer do verze 9, na ostatních prohlížečích není aplikace testována.

Aimtec, a.s. dodržuje standardní konvence pro zdrojové kódy v jazyce Java. Názvy proměnných, tříd a metod musí být maximálně výstižné a stejně jako komentáře musí být psané v anglickém jazyce. Firma řídí vývoj metodou test-driven-development (vývoj řízený testy), proto je kladen velký důraz na otestování veškerých možných stavů i funkčnosti aplikace. Testy můžeme rozdělit do několika skupin:

- Java Unit testy – slouží zejména pro ověření správné funkčnosti jednotlivých metod aplikace (např.: Testuje se, zda metoda pro kódování url vrací správné hodnoty)
- Akceptační testy – kontrolují správné fungování modulů na spuštěné aplikaci. Využívá se nástroj Jameleon, který spouští a vykonává jednotlivé funkce

v aplikaci a porovnává výstup. Simuluje chování uživatele a testuje všechny možné stavy aplikace.

- Databázové Unit testy – ověřují správnou funkčnost procedur a funkcí v databázi. Využívá se framework T-SQL Test Tool (T.S.T.). Tento framework vytvoří speciální databázi obsahující sadu užitečných funkcí na porovnávání hodnot i celých tabulek. Testy probíhají v transakci a tak v případě selhání dojde k odrolování všech změn (rollback).

2.1. Předchozí implementace autentifikace

V předchozí době byla autentifikace řešena jen pomocí nativního Java kódu bez použití jakýchkoliv knihoven. Zároveň byla tato část kódu jedna z nejstarších a velmi nepřehledná. Celý proces přihlášení byl rozprostřen ve více třídách, ve většině případů pokrývajících podobnou funkčnost. Většina volaných metod vracela specifické číslo, které bylo následně porovnáváno v klausuli switch, kde se vykonal zbytek kódu. Po správném zadání uživatelského jména a hesla se uchovala aktuální session pro pozdější možnost identifikace uživatele. Načetlo se menu, podle rolí, ke kterým byl uživatel přiřazen, a provedlo se přesměrování na základní obrazovku aplikace. V případě neúspěchu se zobrazila chybová zpráva s popisem chyby (např: neexistující uživatel apod.). Jedním z požadavků pro reimplementaci bylo, že z pohledu uživatele nesmí dojít k žádným výraznějším změnám.

Aplikace podporuje ověření uživatelských údajů i oproti Microsoft Active Directory. Pro tuto funkcionalitu se již používá starší verze knihovny Apache Shiro.

2.2. Předchozí implementace autorizace

Pro rozhodnutí, zda uživatel má právo danou akci vykonat, se v aplikaci ověřovala zejména jména rolí. Jednotlivá oprávnění specifikující danou akci byla sice nakonfigurována, ale používala se zřídka. Kontrola proběhla jen při přihlášení kvůli vykreslení hlavního menu a při požadavku na otevření detailu položky kvůli vykreslení různých tlačítek. To umožňovalo dostat se na stránky v aplikaci, na které neměl uživatel

právo, přes uložené záložky apod. Toto je velký problém z hlediska bezpečnosti a činí to hlavní podnět celý proces autentifikace a autorizace reimplementovat.

V aplikaci existuje několik speciálních rolí – tzv. "*systémové role*". Ty mají specifickou skupinu oprávnění a slouží zejména k administraci aplikace:

- *Aimtec.Administrator* – hlavní administrátorská role, mající oprávnění na všechno
- *Supplier.Administrator* – role zachovaná z historických důvodů, nyní v aplikaci nepoužívaná
- *Multicompany.Administrator*, *Company Administrator* – sada rolí patřící do skupiny *MULTI_LOCATION_ROLES*, která umožňuje zobrazit informace z jiných společností i jiných lokací[1] jedné společnosti.
- *All.Transactions* – role umožňující prohlížet všechny transakce

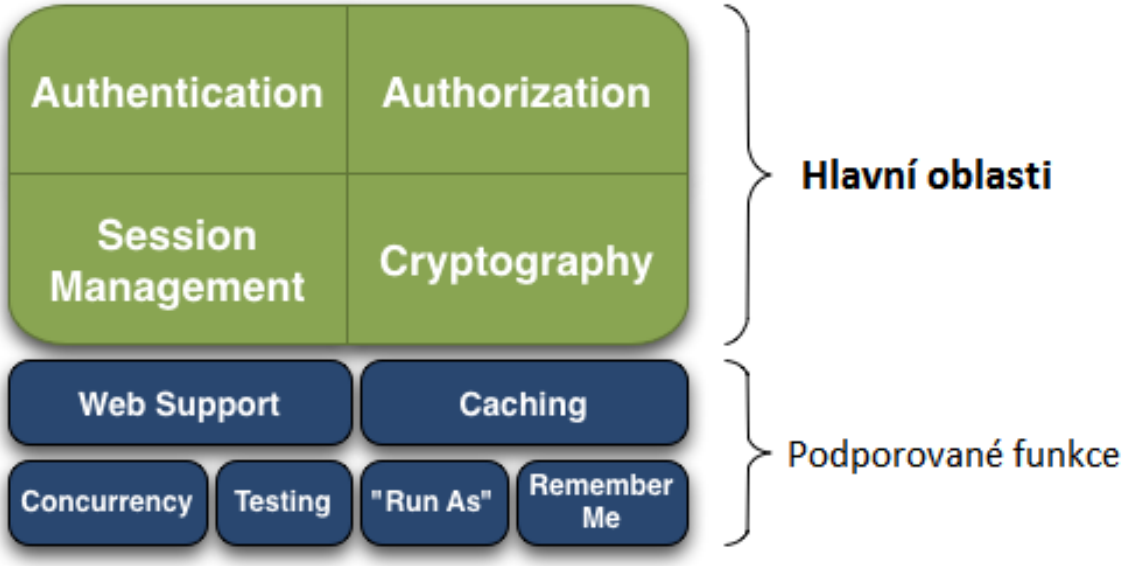
Pro implementaci bezpečnostní vrstvy je nutné, aby kontrola jmen rolí byla všude nahrazena pomocí ověření konkrétních oprávnění. Zároveň se musí zamezit možnosti dostat se na adresu v aplikaci, na kterou uživatel nemá právo.

[1] DCIx umožňuje rozlišovat mezi různými lokacemi stejného podniku. Například: jedna společnost používá DCIx pro skladové hospodářství i pro logistiku, ale role administrátor přiřazena k lokaci společnosti, kde se provádí jen logistika, nemůže zasahovat do lokace skladového hospodářství.

3. Knihovna Apache Shiro

V této kapitole se čerpá zejména ze zdroje (3).

Apache Shiro [šíro] (japonsky hrad) je poměrně robustní a snadno použitelný Java bezpečnostní framework, který dokáže provádět autentifikaci, autorizaci, session management a kryptografii. Podporuje jak základní konzolové aplikace, tak i velké Java EE aplikace. Funkce frameworku jsou znázorněny na obrázku 5. Jednotlivým oblastem bude věnována zvláštní podkapitola.

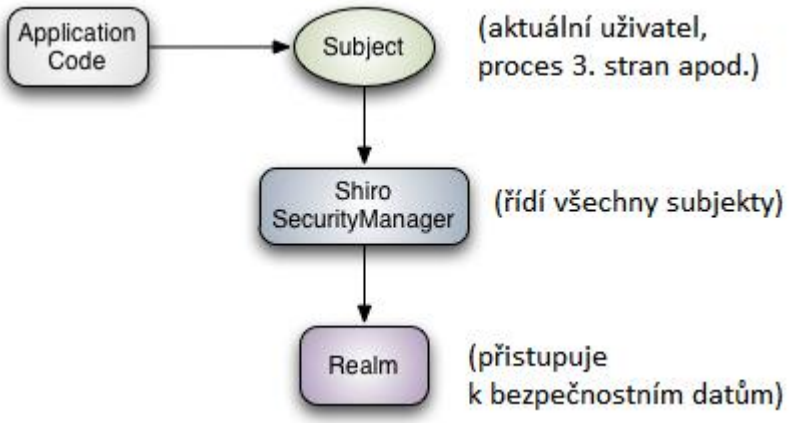


Obrázek 5 Funkce frameworku Apache Shiro

Předchůdce Apache Shiro byl framework JSecurity, který byl založen v roce 2004 pány Les Hazlewoodem a Jeremy Hailem, kteří první řádky kódu frameworku napsali již v roce 2003. Od roku 2004 do 2008 se připojili k projektu ještě Tim Veil, Peter Ledbrook a Allan Ditzel. V roce 2008 byl projekt JSecurity přesunut pod hlavičku Apache Software Foundation do inkubačního programu(4), kde se následně změnilo jméno na Ki a záhy na Shiro, kvůli možnému porušení ochranné známky. Hlavním důvodem pro tvorbu JSecurity (později Apache Shiro) byla celková náročnost a programátorská nepřívětivost ostatních dostupných řešení jako například Java

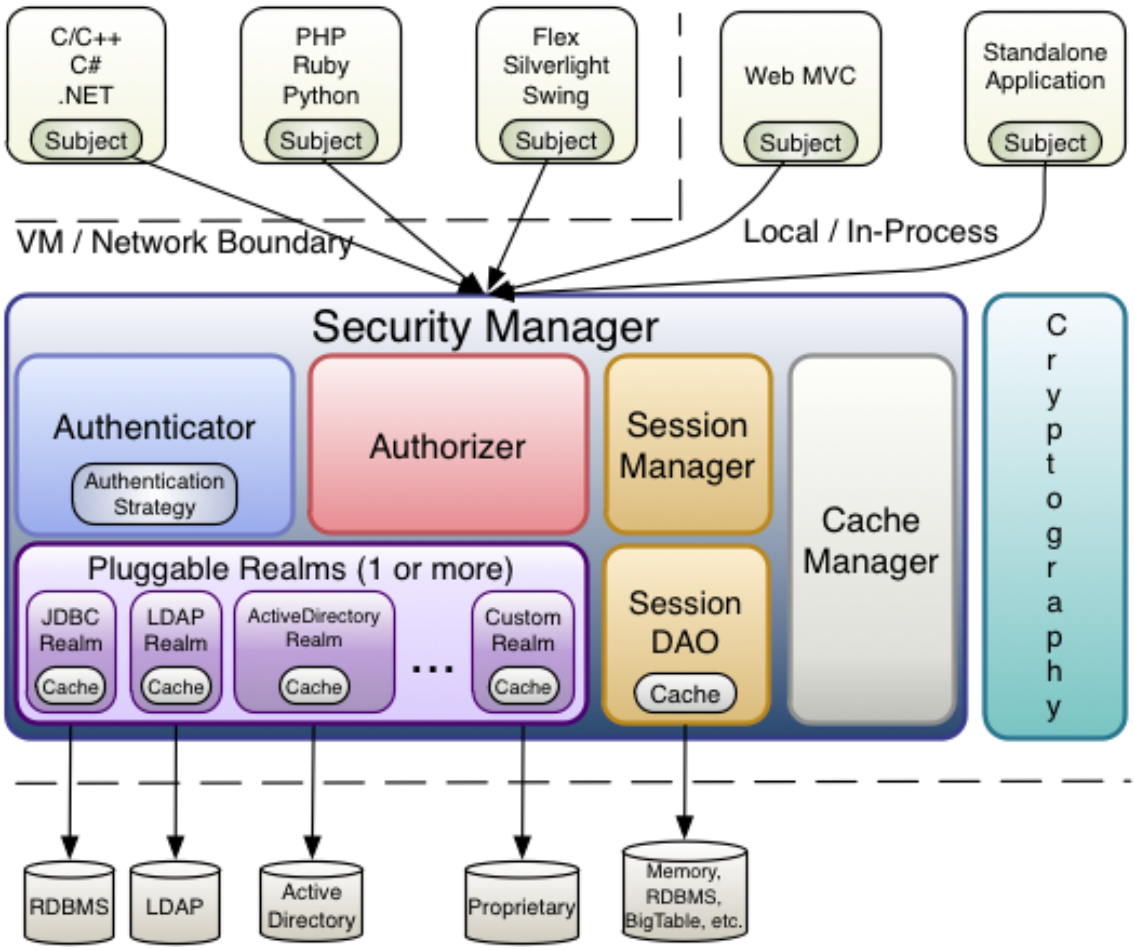
Authentication and Authorization Service (JAAS). Při vývoji frameworku je kladen důraz hlavně na jednoduché použití. (5)

Obrázek 6 znázorňuje základní koncepci frameworku, která spočívá v tom, že dokáže ke každé vykonané akci přiřadit tzv. *Subject* (dále subjekt), to může být jak člověk, tak i nějaký proces. Tento subjekt můžeme autentifikovat – prokázat jeho identitu (např.: člověk se nejčastěji prokazuje uživatelským jménem a heslem) a následně i ověřovat jeho dostupná oprávnění a role. Shiro si tento subjekt stále uchovává v paměti a v jakékoliv části kódu můžeme vykonat výše zmíněné úkony. O správu subjektů se stará *Security Manager*. Pro přístup k datům nutným k autentifikaci/autorizaci slouží *Realm*. Realmy budou detailně popsány dále.



Obrázek 6 Základní koncepce Apache Shiro

Diagram na obrázku 7 ukazuje detailní pohled na architekturu Apache Shiro.



Obrázek 7 Detailní pohled na architekturu Apache Shiro

- *Security Manager* – střed architektury frameworku. Je to zastřešující objekt, který řídí a spojuje komponenty, aby vše fungovalo. Zároveň spravuje všechny subjekty a umožňuje provádět bezpečnostní operace pro každý ze subjektů zvlášť.
- *Authenticator* – komponenta zodpovědná za spouštění a reagování na autentifikační pokusy subjektu (uživatele, procesu apod.). Pokud se subjekt pokusí přihlásit, je volána tato komponenta, která interaguje s dostupnými realmy, které poskytují informace nutné pro ověření identity. Dále poskytuje moduly pro kontrolu hesla, ať už šifrovaných či nikoliv.

    - *Authentication Strategy* – tato komponenta slouží pro popis způsobu interagování s realmy, pokud je nakonfigurován více než jeden realm. Určuje, jestli byl autentifikační pokus úspěšný, či nikoliv. Definuje pravidla, jestli například stačí, aby jen jeden realm prokázal identitu subjektu, ostatní nemusí, nebo jestli musí všechny realmy uspět, nebo pokud stačí, aby uspěl jen první.
- *Authorizer* – komponenta zodpovědná za kontrolu práv subjektu v aplikaci. Jednoznačně určí, jestli subjekt dané právo má, nebo ne. Stejně jako *Authenticator* ví, jak komunikovat s dostupnými realmy, které poskytují potřebná data jako jsou přiřazené role a práva k subjektu.
- *SessionManager* – slouží k vytváření a řízení životních cyklů uživatelské session. Unikátní komponenta na poli bezpečnostních frameworků. Umožňuje nativně pracovat s uživatelskými session v kterémkoli prostředí (nemusí se jednat jen o webovou aplikaci). Ve výchozím stavu se Shiro pokusí využít existující mechanismus na správu session, pokud nějaký existuje (například servlet kontejner), nebo když žádný nenajde, použije svůj session management. Pro určení, kde se bude uchovávat daná session, slouží *SessionDAO*.
    - *SessionDAO* – provádí operace pro persistenci session (CRUD operace) na požádání od *SessionManagera*. Umožňuje zvolit jaký zdroj dat bude použit pro uchování session.
- *CacheManager* – vytváří a spravuje životní cykly instancí keší, které jsou užívané ostatními komponentami Shira. Protože Shiro může přistupovat k několika zdrojům dat pro autorizaci i autentifikace, je nutné tyto data uchovávat v dočasné paměti – keš. Umožňuje připojit jakýkoliv moderní produkt starající se o správu keší. Shiro nabízí již hotovou implementaci *CacheManagera* pro knihovnu Ehcache.
- *Cryptography* – jak už název napovídá, jedná se o komponentu umožňující využívat různé kryptovací a hešovací algoritmy. Obsahuje snadno použitelné a srozumitelné implantace různých šifer, heší a jiných kódovacích algoritmů. Na rozdíl od nativní Java implementace, Shiro poskytuje pro jednotlivé šifry a heše objektově založený mechanismus.

- *Realm* – slouží k přístupu k bezpečnostním informacím. Realmy se chovají jako most nebo spojník mezi Shirem a bezpečnostními daty. Pokud je potřeba provést autorizaci nebo autentifikaci, nejprve se zjistí jaký realm (popřípadě kolik realmů) může obsloužit daný požadavek. K tomuto účelu slouží metoda *boolean supports* s parametrem *AuthenticationToken token*. Tento token je rozhraní, které určuje dvě hlavní metody pro práci s daty sloužící k ověření identity. Nejčastěji se používá implementace *UsernamePasswordToken*, která funguje na principu jména a hesla. Metoda *supports* vrátí buď *true*, pokud realm danou implementaci tokenu podporuje, jinak *false*. Potom již následuje samotný proces autorizace nebo autentifikace, který řídí *Authorizer* nebo *Authenticator*. Pro vlastní implementaci realmu jsou důležité metody *doGetAuthorizationInfo* a *doGetAuthenticationInfo*, kde je potřeba naprogramovat logiku autorizace a autentifikace. Shiro poskytuje několik již připravených realmů, které stačí jen minimálně nakonfigurovat, například: realm pro Microsoft Active Directory nebo realm pro LDAP[2].

Všechny komponenty Apache Shiro jsou založeny na POJO (Plain Old Java Object) a tudíž je lze jednoduše nahradit. Stačí implementovat metody určené jednotlivými rozhraními a napojit třídu do Shira. To můžeme provést buď programově pomocí getrů a setrů, protože Shiro dodržuje koncepci JavaBean, nebo pomocí konfigurace v ini (popřípadě web.xml) souboru. Pro více vláknové aplikace je nutné použít konfiguraci pomocí souboru, pokud bychom pouze změnili komponentu programově, změna by se týkala pouze aktuálního vlákna. Konfigurační soubor je rozdělen do 4 částí:

- [main] – zde se uvádí konfigurace pro SecurityManagera, připojují jednotlivé třídy, mění hodnoty proměnných apod.
- [users] – definuje statickou skupinu uživatelských účtů. Výhodné jen pro aplikace s malým počtem uživatelů.
- [roles] – dovoluje přiřadit práva jednotlivým uživatelům nebo rolím vytvořených v předešlé sekci.
- [urls] – specifická sekce pouze pro webové aplikace. Určuje vzory adres, které vyžadují nějaké oprávnění.

[2] LDAP (Lightweight Directory Access Protocol) – protokol pro čtení a úpravu adresářových struktur na serveru pracující nad TCP/IP.

Na následujícím okomentovaném úryvku lze vidět příklad konfigurace Apache Shiro.

```
[main]
# vytvoření instance třídy Sha256CredentialsMatcher pod jménem sha256Matcher
sha256Matcher = org.apache.shiro.authc.credential.Sha256CredentialsMatcher
# vytvoření instance třídy DatabaseRealm pod jménem myRealm
myRealm = com.company.security.shiro.DatabaseRealm
# vytvoření instance třídy MyFilter
myFilter = com.company.security.filters.MyFilter
# změna timeoutu na 30000ms – zavolá myRealm.setConnectionTimeout(30000);
myRealm.connectionTimeout = 30000
myRealm.username = jsmith
myRealm.password = secret
# použití námi zvolené implementace třídy pro ověření hesla
myRealm.credentialsMatcher = $sha256Matcher
# zavolá securityManager.getSessionManager().setGlobalSessionTimeout(18000);
securityManager.sessionManager.globalSessionTimeout = 18000

[users]
#seznam uživatelů ve formátu: jméno = heslo, přiřazenáRole1, přiřazenáRole2, ...
admin = tajnéHeslo, administrator
uživatel1 = heslo, role0, role1
uživatel2 =heslo12345, roleA,roleB

[roles]
# role administrator má oprávnění na vše
administrator = *
# uživatel1 může tisknout na tiskárně X:
uživatel1 = tiskárna:X:tisk
# roleA může provádět všechny akce s kteroukoli tiskárnou
rolaA = tiskárna

[urls]
# na následující  url může přistupovat anonymní uživatel
/index.html = anon
# na adresy začínající prefixem /user/ smí přistupovat jen přihlášení uživatelé
/user/** = authc
# tato adresa vyžaduje přihlášeného uživatele s rolí administrátor
/admin/** = authc, roles[administrator]
#  všechny požadavky na url začínající prefixem /myFilter/ budou procházet filtrem
# “myFilter“ definovaným v části [main]
/myFilter/** = myFilter
```

3.1. Autentifikace

Autentifikace neboli ověření identity je činnost, při které prověřujeme, že někdo (popřípadě něco) je opravdu ten za koho se vydává. V Apache Shiru se skládá ze 3 základních úkonů:

a) Shromáždit identifikující údaje (nejčastěji přihlašovací jméno) a ověřovací údaje (nejčastěji heslo).
b) Předat nashromážděné údaje do autentifikačního mechanismu.
c) Povolit přístup, opakovat autorizaci, nebo přístup nepovolit.

ad a) Shromáždění údajů – Shiro vyžaduje zkonstruovat autentifikační token, který obsahuje identitu a ověřovací údaje subjektu. Nejčastěji se používá implementace *UsernamePasswordToken*.

ad b) Předání údajů do autentifikačního mechanismu – Nejprve je nutné získat aktuální subjekt, to provedeme voláním statické metody *getSubject()* ve třídě *SecurityUtils*. Pak můžeme provést pokus o přihlášení metodou *login(token)* nad aktuálním subjektem. Kód je zobrazen na Obrázek 8 Ukázka kódu pro autentifikaci.

ad c) Obsluha úspěchu a neúspěchu – Celý proces autentifikace funguje na principu výjimek, pokud v průběhu nastala chyba (například nepodařilo se načíst uživatele s uživatelským jménem v předaném tokenu), je vyvolána příslušná výjimka (například neexistující uživatel). V kódu aplikace je nutné tyto výjimky zachytit a reagovat na ně. Příklad kódu ukazující všechny části procesu autentifikace je zobrazen na obrázku 8.

```
//konstrukce tokenu
UsernamePasswordToken token = new UsernamePasswordToken("jméno", "heslo");
//získání aktuálního subjektu
Subject currentUser = SecurityUtils.getSubject();
//pokus o přihlášní
try{
    currentUser.login(token);
}catch(UnknownAccountException uae){
    //neexistující uživatel
}catch(IncorrectCredentialsException ice){
    //špatné heslo
}catch(AuthenticationException ae){
    //obecná chyba autentifikace, rodič předchozích tříd
}
```

Obrázek 8 Ukázka kódu pro autentifikaci

Shiro nabízí i mechanismus pro zapamatování uživatele. Pro určení, zda tuto funkci využít, slouží další nepovinný parametr při konstrukci tokenu. K ověření, jestli je konkrétní subjekt zapamatován, slouží metoda *subject.isRemebered()*. Je ovšem velmi důležité rozlišovat stav zapamatovaný (*subject.isRemebered())* a ověřený (*subject.is Authenticated()).* Tyto dva stavy se navzájem vylučují.

- **Zapamatovaný** (remembered) je stav, kdy známe identitu uživatele (to znamená, že *subject.getPrincipals()* nevrátí *null* ani prázdnou množinu), ale nemůžeme ji s jistotou prokázat. Tato identita je známá z předešlé session.
- **Ověřený** (authenticated) je stav, kdy aktuální subjekt v session již prošel autentifikačním procesem bez vyhození jakékoli výjimky.

3.2. Autorizace

Autorizace je proces řízení přístupu. Ověřuje se, kdo má právo na co. Existují 3 základní typy autorizace: definovaná na oprávnění, na roli a na uživatele. Framework dokáže ověřovat oprávnění pomocí Java kódu, JDK anotacemi nebo pomocí JSP/GSP taglib. Pro dobrou odezvu a interaktivou je důležitý rychlý chod celého procesu autorizace, proto Shiro nabízí získaná oprávnění a role kešovat.

3.2.1. Role

Role je pojmenovaná entita, která typicky representuje množinu oprávnění a chování v aplikaci. Role jsou většinou přiřazeny k uživatelským účtům. Na základě tohoto spojení lze v aplikaci kontrolovat jednotlivá oprávnění.

Pro ověření, zda má aktuální subjekt přiřazenou specifickou roli, slouží metody:

- *subject.hasRole("specifickáRole")* vracející buď *true* nebo *false*
- *subject.checkRole("specifickáRole"),* která v případě, že daný uživatel nemá přiřazenou roli *"specifickáRole"* vyhodí výjimku.

K ověření role může také sloužit Java anotace *@RequiresRoles("specifickáRole"),* kde se vykoná metoda, jen pokud subjekt má danou roli, jinak vyhodí výjimku. V JSP

souborech lze použít tag *<shiro:hasRole name="specifickáRole "> </shiro:hasRole>* (obsah ohraničený tagem se vykoná jen pro roli *"specifickáRole"*).

3.2.2. Uživatel

Protože s aplikací nemusí nutně komunikovat jen člověk, je v Shiru brán uživatel jako subjekt. Subjekt může být (jak již bylo řečeno) člověk, proces, rutina apod. K jednotlivým subjektům lze přiřadit role nebo konkrétní oprávnění. Pro kontrolu uživatelského jména lze použít metodu *subject.getPrincipals()* vracející aktuální kolekci všech uživatelských jmen přiřazených k aktuálnímu subjektu.

3.2.3. Oprávnění

Oprávnění v Shiru představuji atomické elementy bezpečnostní politiky. Jsou základními výroky, co lze v aplikaci vykonávat (například tisknout na tiskárně X, editace výrobku Y apod.). Oprávnění nedefinují, kdo může akci vykonat, jen popisují samotnou akci. Kdo má dané právo, či nikoli se definuje buď pomocí přiřazených rolí, nebo konkrétnímu uživateli.

Shiro používá tzv. ***WildcardPermission***. Oproti výše zmíněným oprávněním (tisk na tiskárně X a editace výrobku Y) lze WildcardPermission poměrně jednoduše definovat, aby programová interpretace odpovídala požadavkům a zároveň se dala oprávnění číst i člověkem. Základní koncepce spočívá v definování unikátních řetězců, ke kterým můžeme přidávat za dvojtečku další možné operace opět pomocí unikátních řetězců. WildcarPermission pro tisk na tiskárně X by tedy vypadalo *"tiskárna:X:tisk"*. Tato koncepce nám jednoduše umožní definovat další oprávnění třeba pro čtení fronty na tiskárně - *"tiskárna:X:čtení:fronta"*. Pro přiřazení oprávnění můžeme použít zástupný znak * nebo konkrétní řetězec. Pokud uživateli přiřadíme právo *"tiskárna:*:čtení:*"*, může z každé tiskárny číst jakékoli údaje. Pro rozhodnutí, jestli aktuální subjekt má dané oprávnění, se používá implikace, nikoli rovnost. Oprávnění můžeme definovat i s chybějícími částmi, například: oprávnění *"tiskárna"* znamená, že můžeme provádět jakoukoli akci na jakékoli tiskárně. V aplikaci se pak jednotlivá oprávnění kontrolují následovně:

- *subject.isPermited("oprávnění")* – vrátí *true* v případě, že aktuální subjekt implikuje oprávnění, jinak *false*.
- *subject.checkPermission("oprávnění")* – vyhodí výjimku, pokud subjekt neimplikuje oprávnění.
- *@RequiresPermissions("oprávnění")* – metoda s touto anotací je určena jen pro subjekt implikující oprávnění, jinak vyhodí výjimku.
- *<shiro:hasPermission name="oprávnění"></shiro:hasPermission>* - obsah uzavřený do elementu se vykoná, jen pokud subjekt implikuje oprávnění.
- *<shiro:lacksPermission name="oprávnění"></shiro:lacksPermission>* - opačný případ předchozího elementu. Obsah se vykoná, jen pokud subjekt oprávnění neimplikuje.

3.3. Session Management

Session je soubor údajů, které patří k danému uživateli po dobu interakce s aplikací. Session se používá zejména u webových aplikací, ale Shiro dokáže vytvořit a používat session ve všech Java aplikacích. K aktuální uživatelské session můžeme přistupovat přes metodu *subject.getSession()*. Vstupem může být parametr *boolean*, pokud bude *true*, metoda vytvoří novou session (když bude potřeba), pokud bude parametr *false*, metoda vrátí *null*, když session neexistuje.

Veškeré session spravuje *SessionManager*. Jako výchozí používá Shiro implementaci *DefaultSessionManager* umožňující jak základní operace, tak pokročilejší jako je validace, cleanup, timeout apod.

Shiro nabízí celou škálu nadstandardních funkcí nad session. Například:

- Vytvoření posluchače nad session, umožňující okamžitě reagovat na události typu start, stop a exspirace.
- Změnit úložiště a použít vlastní DAO[3] pro session. Využít tak například keše pro ukládání a správu session.

[3] DAO (Data Access Object) – objekt poskytující rozhraní pro persistenci určitého objektu. Provádí CRUD (create, read, update, delete) operace.

- Zabránění exspirace session v případě neaktivity. V jakékoliv části kódu je možné zavolat metodu *touch()*, která udrží session aktivní.

Veškeré používání session implementované pomocí Apache Shiro je transparentní se základní Servlet 2.5 specifikací (HttpSession). Proto není nutné nijak zasahovat do existujícího kódu, pokud má být nasazena knihovna Apache Shiro.

4. Ostatní dostupná řešení pro implementaci bezpečnostní vrstvy

4.1. JAAS

V této kapitole je čerpáno ze zdroje (6).

Hlavní alternativou pro knihovnu Apache Shiro zůstává JAAS (Java Authentication and Authorization Service). JAAS je integrován do Java SE od verze 1.4. V předchozí verzi byl dostupný jen jako rozšíření. Skládá se ze dvou částí: část pro autorizaci a část pro autentifikaci. JAAS je založen na modelu Subject-based (založený na subjektu) a používání callbacků. Subjekt, stejně jako v Apache Shiro, je označení pro zdroj požadavku. Může obsahovat kolekci *principal* (prokázané identity subjektu) a *credential* (objekt prokazující identity).

Autentifikace probíhá v následujících krocích:

1. Inicializace třídy *LoginContext*.
2. *LoginContext* načte použitou konfiguraci a moduly pro přihlášení (*LoginModules*).
3. Je zavolána metoda *login()* ve třídě *LoginContext*.
4. Metoda *login()* zavolá načtené *LoginModules*, které se pokusí autentifikovat subjekt.
5. *LoginContext* vrací status, pokud autentifikace byla úspěšná, je vrácen aktuální subjekt.

Pro autorizaci je pak nutné, aby subjekt předchozím procesem prošel úspěšně, musí mít přiřazen kontext pro kontrolu oprávnění a tato oprávnění musí být definována pro třídu *Policy*.

Je vidět, že knihovna Apache Shiro je velmi podobná s tímto řešením. Některé postupy se přímo shodují. Třídy LoginModules a LoginContext korespondují s realmy. Apache Shiro však nabízí jednoduší přístup a konfiguraci. Eliminuje nutnost vytvářet callbacky

a usnadňuje jak autentifikaci, tak autorizaci. JAAS transparentně neřeší ani kešování získaných oprávnění, proto je Apache Shiro vhodnějším řešením vzhledem k požadavkům na aplikaci.

4.2. Spring Security

V této kapitole se čerpá ze zdroje (7).

Spring security (dříve Acegi Security System for Spring) je bezpečnostní framework používaný zejména pro projekty založené na frameworku Spring. Je určený jen pro Java EE aplikace. Podporuje celou škálu autentifikačních mechanismů od základního webového rozhraní, přes OpenID, po LDAP servery. Hlavní koncepce spočívá v konfiguraci manažerů pro autentifikaci – AuthenticationManager a pro autorizaci – AccesDecisionManager. Stejně jako Apache Shiro pro webové aplikace využívá servlet filtr definovaný ve web.xml.

Spring security je určený zejména pro Java EE projekty používající Spring jako hlavní aplikační framework. Aplikace DCIx však používá structs. Opět můžeme pozorovat podobnost tohoto frameworku s Apache Shiro, zejména použitím filtrů, avšak Apache Shiro opět nabízí jednodušší a všestrannou konfiguraci a možnost použití vlastních tříd při autorizaci i autentifikaci.

5. Implementace bezpečnostní vrstvy

Realizace probíhala pod kontrolou vedoucího vývoje firmy Aimtec, a.s - Janem Brnkou. První schůzka byla čistě informační, kde se domluvily detaily ohledně další spolupráce. K projektu byl přidělen firemní supervizor - vývojář Martin Augusta, se kterým probíhala veškerá komunikace ohledně realizace projektu až do začátku roku 2013, kdy projekt převzal Jan Brnka. Vývoj probíhal iteračně většinou po dvou týdnech. Následovala schůzka, kde se probraly uplynulé dva týdny (retrospektiva) a naplánovala práce na další iteraci.

5.1. Zprovoznění projektu

První iterace byla věnována spuštění projektu na soukromém osobním počítači, seznámení se s aplikací DCIx a knihovnou Apache Shiro. Aplikace DCIx pro svůj běh potřebuje server Apache Tomcat ve verzi 6, Microsoft SQL Server 2008 R2 a několik knihoven. Vývoj probíhal ve vývojovém prostředí Eclipse IDE. Speciálně pro tento projekt byl založen privátní git repositář na BitBucket.org, kam se ukládaly provedené změny (commits). Dohlížející programátor kód kontroloval (code review) a aplikoval změny do vývoje interní větve (branch) aplikace.

Soubory a data potřebná pro spuštění a vývoj byla dodána firmou Aimtec, a.s. Pro software, na který neměla firma licenci pro tento účel, byla použita licence získaná přes školní DreamSpark Premium, konkrétně se jednalo o produkt Microsoft SQL Server 2008 R2.

Vývoj se řídil technikou programování řízené testy (Test-driven development). Na ověření správné funkčnosti již z velké části existovaly testy, které musely po ukončení vývoje probíhat správně. Bylo možné je jen přizpůsobit nově vzniklým situacím související s novou bezpečnostní vrstvou.

Součástí seznámení se s knihovnou Apache Shiro bylo vytvoření jednoduché webové aplikace, využívající základní metody ověření a autorizace pomocí zmíněné knihovny.

5.1.1. Základní aplikace využívající Apache Shiro

Hlavním požadavkem pro tuto základní aplikace byla jednoduchost a vyzkoušení různých technik pro autentifikaci a autorizaci. Aplikace načítá uživatele, role a přiřazená oprávnění jen z konfiguračního souboru frameworku Shiro. Obsahuje stránky přístupné jen specifickým rolím nebo uživatelům s daným oprávněním. Pro svůj chod využívá Apache Tomcat ve verzi 6. Je dostupná na přiloženém CD v adresáři *zdrojove_kody/ShiroSampleApp/*. Vygenerovaná JavaDoc se nachází ve složce */doc/ShiroSampleApp*.

Vstupním bodem aplikace je servlet */login*. Tento servlet přesměruje požadavek na JSP soubor *login.jsp*, který obsahuje formulář pro přihlášení uživatele. Po stisku tlačítka *přihlásit* je zavolána příslušná akce v servletu, která využívá metod frameworku Shiro a pokusí se přihlásit uživatele. V případě úspěchu je přesměrován na servlet */account*, který zobrazí jen dostupné odkazy pro aktuálního uživatele. Pokud se uživatel pokusí zobrazit stránku, na kterou nemá právo, Shiro zobrazí http chybu 401 unauthorized.

Uživatelská dokumentace včetně seznamu uživatelů a jejich přiřazených práv je připojená k tomuto dokumentu jako příloha.

5.2. Autentifikace

Tato část aplikace byla jedna z nejstarších, a proto byla nutná její reimplementace. Autentifikace byla řešená pouze pomocí nativního Java kódu. Jako bezpečnostní framework, usnadňující řešení daného problému, byl zvolen Apache Shiro.

Aplikace DCIx podporuje pouze přihlášení pomocí kombinace jména a hesla ve webovém formuláři nebo v osobním terminálu připojeným přes telnet. Následně jsou tyto data porovnávána se záznamy v databázi nebo Active Directory. Pro framework Shiro musí být zkonstruován token obsahující jak identifikační (uživatelské jméno) objekt, tak ověřovací (heslo). Pro kombinaci jména a hesla byla použita základní implementace tohoto tokenu ve formě *UsernamePasswordToken*. Pro aplikace náročnější na bezpečnost lze samozřejmě ověřovat uživatele například pomocí certifikátů, čipové karty nebo jiných objektů. Před samotným ověřováním identity je

zkontrolováno, zda je nějaký uživatel již přihlášen pod aktuální session, pokud ano, zobrazí se (podle nastavení aplikace) varovná zpráva a možnost přehlásit se (tím bude předchozí uživatel automaticky odhlášen). Pokud není žádný uživatel přihlášen, nebo byl v předchozím případě odhlášen, zavolá se metoda z knihovny Apache Shiro *login(UsernamePasswordToken)*. Tato metoda se vždy váže k aktuálnímu subjektu, který celou akci vyvolal (v tomto případě se jedná o uživatele pokoušejícího se přihlásit). V průběhu metody se nejprve ve třídě *ModularRealmAuthenticator* zjistí, jaké realmy se budou používat. Slouží k tomu metoda *supports(Token)*, která by měla být implementována v každém použitém realmu. Pokud vrátí hodnotu *true*, bude použit právě tento realm. Původní návrh implementace se skládal ze dvou použitých realmů: jeden pro ověření oproti údajům v databázi; druhý pro ověření oproti Microsoft Acitve Directory. Nakonec se od tohoto návrhu upustilo kvůli potížím při autorizaci a je použit jen jeden realm kombinující dva předešlé. Vytvořený realm v metodě *supports(Token)* vrací vždy *true*, čímž se zajistí, že bude použit právě tento a žádný jiný ani vestavěný realm z knihovny Apache Shiro. V realmu se kontroluje, jestli daný uživatel existuje, není zablokovaný a firma, ke které patří, je aktivní. Neprovádí se zde kontrola hesla, pouze se načte heslo z databáze, vytvoří se instance typu *SimpleAuthenticationInfo* (která právě toto heslo spolu se jménem uživatele a jménem realmu, který tyto informace získal, obsahuje). Tento nově vzniklý objekt je použit v dalším kroku autentifikace k porovnání hesla. To je z toho důvodu, že Apache Shiro poskytuje v základu několik mechanismů pro ověření hesla. Heslo může být hešované nebo šifrované, v Shiru stačí zavolat příslušnou metodu, která se sama postará o upravení vstupního řetězce do požadovaného (zahešovaného) tvaru. DCIx však používá proceduru v databázi, která pomocí symetrické šifry heslo z databáze převede do původního tvaru, tudíž stačí porovnat dva řetězce, které by se měly shodovat. Pokud autentifikace v nějakém předešlém kroku selže, je vyhozena výjimka s popisem chyby.

V aplikaci je zachycováno celkem pět výjimek, které mohou při přihlášení nastat. *IncorrectCredentialsException* značí špatně zadané heslo a k aktuálnímu uživateli je do databáze uložena informace o neúspěšném pokusu o přihlášení, který může vést až k blokaci. *UnknownAccountException* (respektive *DisabledAccountException)* znamená neexistující (respektive zablokovaný) účet. Nově naprogramovaná výjimka *UserCompanyNotActiveException* označuje, že uživatelova firma je zablokovaná.

Poslední *AuthenticationException* je rodičovská třída předchozích výjimek a je zachycena jen když předchozí neodpovídaly aktuálně vyvolané instanci výjimky.

Po úspěšné autentifikaci (metoda *login()* proběhla bez výjimek) se pokračuje v přihlašovacím procesu. Zjistí se, jestli má uživatel povoleno přihlašovat se do více společností, pokud ano, zobrazí se mu list s dostupnými společnostmi a celý proces přihlášení se opakuje. Pokud uživatel toto oprávnění nemá, je přihlášen do své výchozí společnosti. Dále je ověřováno, zda je daný uživatel přihlášen ještě někde jinde pod jinou session, v případě potřeby je z předešlé session odhlášen. Nakonec se uloží informace o přihlášeném uživateli, načtou se role, ke kterým je uživatel přiřazen a načtou se položky menu, na které má oprávnění.

Pro správné fungování celého procesu autentifikace využívající Apache Shiro bylo nutné upravit a nově vytvořit několik tříd a souborů:

- *LogonAction.java* – Servlet volaný při pokusu o přihlášení, obsahuje hlavní logiku celého procesu autentifikace.
- *DciAuthcAndAuthzRealm.java* – Realm přistupující k datům z databáze. Ověřuje, jestli daný uživatel existuje, není zablokován on ani jeho firma.
- *DciCompanyWorkspace.java* – Třída, která dokončí proces přihlášení: načte menu, uloží informace o uživateli do session apod.
- *web.xml* – Definice nového servlet filtru pro framework Apache Shiro a jeho konfigurace (parametr *config*).

Původní návrh ještě zahrnoval několik tříd týkajících se zejména použití více realmů:

- *DciAuthenticationStrategy.java* – Třída implementující metody pro popis autentifikační strategie. Pomocí tří metod: *afterAttempt()*, *beforeAttempt()* a *afterAllAttempts()* se zajistilo, aby autentifikace proběhla nejdříve přes Active Directory, pokud selhala, použil se druhý realm obsluhující klasické ověření oproti databázi.
- *DciCredentialsMatcher.java* – Jednoduchá třída porovnávající hesla, pokud se neshodovala, uložila se informace do databáze o neúspěšném pokusu o přihlášení k danému uživateli. Funkčnost této třídy se přesunula do hlavního servletu pro přihlášení (*LogonAction.java*), kde v případě zachycení výjimky *IncorrectCredentialsException*, se provede výše zmíněná akce.

Pro ověření správné funkčnosti byly vytvořeny automatické jameleon testy, pokrývající všechny možné stavy aplikace při přihlášení. Jedná se zejména o soubor *allLoginScenarios.xml* vytvořený testerem ve společnosti Aimtec, a.s. – Lubošem Kovaříkem. Test se musel mírně upravit, aby fungoval na aktuální větvi aplikace DCIx na soukromém PC. Zejména se neshodovalo volání některých uložených funkcí a procedur v databázi. Na obrázku 9 lze vidět, že všechny testy proběhnou úspěšně.

Test Cases Results					Failure Reasons
allLoginScenarios	Time:	23.578s	Run:	1	-
telnetLogin	Time:	8.08047s	Run:	1	-
telnetLoginCust	Time:	6.143s	Run:	1	-
topMenuOnLogonPage	Time:	4.516s	Run:	1	-
validLogin	Time:	1.633s	Run:	1	-
validLoginCust	Time:	1.285s	Run:	1	-
validLogin_csv	Time:	4.725s	Run:	4	-

Total Execution Time	Total Tests Executed	Total Tests Passed	Total Tests Failed	% Passed
52.317s	10	10	0	100%

Obrázek 9 Všechny testy na přihlášení proběhnou úspěšně.

5.3. Autorizace

V aplikaci DCIx byla autorizace prováděna většinou na jméno role. Cílem bylo nahradit toto ověřování rolí na ověřování pomocí jednotlivých oprávnění přiřazené k dané roli. Kontrola oprávnění byla rozdělena do 5 základních skupin:

- požadované url – kontroluje se jen url oříznuté o parametry a končící "*.do"*(servlet). Výsledný řetězec specifikující oprávnění má tvar url + prefix "*url:*" (např.: "*url:/filterUsers.do*")
- klíč menu – jedinečný identifikátor akce (např.: editace uživatelů), řetězec má tvar samotného klíče (např.: "*user.edit*"). Toto oprávnění slouží zejména pro zobrazování editačních tlačítek na stránce.
- číslo transakce – vytvoří se řetězec ve tvaru id transakce + prefix "*tranactionID:*" (např.: "*tranactionID:11*"). Kontroluje se podle něj, jestli má daný uživatel právo spouštět danou transakci.
- jméno příkazu – řetězec je ve tvaru jméno příkazu + prefix "*orderName:*" (např.: "*orderName:ShowPackages*")

- číslo reportu – vytvoří se řetězec ve tvaru id reportu + prefix "*reportID:*" (např.: "*reportID:11*").

5.3.1. Definice dat

Pro získání dat z databáze ve správném formátu byl vytvořen nový pohled do databáze: *Permissions_View* (definován v souboru *dciowner.Permissions_View.sql*). T-SQL skript získává data z několika tabulek pomocí klauzule *UNION ALL*. Kvůli vnitřní implementaci aplikace DCIx bylo nutné data seřadit a očíslovat. Nově vzniklý sloupec s číslem slouží jako unikátní id. Pro správnou funkčnost bylo potřeba upravit data v databázi tak, aby logicky odpovídala zařazením ve sloupcích. Zejména se jednalo o adresu url, kde některé položky obsahovaly i parametry, přičemž na tyto údaje existuje speciální sloupec. Na obrázku 10 lze vidět, že u položek s primárním klíčem (*Menu_Key*) *asnArchive* a *asnOrders* jsou hodnoty sloupců *Action_Start* a *Action_Start_Params* správné. Naproti tomu u řádky s klíčem *barcodes* obsahuje sloupec *Action_Start* i parametry oddělené znakem '?'. Tyto anomálie v tabulce byly způsobeny postupným přidáváním nových sloupců do tabulky.

Menu_Key	Action_Start	Action_Start_Params
asnArchive	/filterOrdersArchive.do	action=Prepare&sourceObject=ASN
asnOrders	/filterOrdersASN.do	action=Prepare&orderTypeId=ASN&companySup...
barcode.edit		
barcodes	/filterBarCodes.do?action=Prepare	
billOfMaterial		
caches	/clearCache.do	*NULL*

Obrázek 10 Tabulka Menu_Item a špatné rozřídění hodnot v sloupcích

Pro správné roztřídění parametrů z url musel být vytvořen T-SQL skript ořezávající adresu o parametry, které následně vloží do sloupce *Action_Start_Params*. Firma Aimtec vyžaduje ke každému skriptu upravující stávající data v databázi vytvoření testu. K vytváření a spouštění testů je použit nástroj T-SQL Test Tool (zkráceně T.S.T.). Oba skripty jsou okomentovány a přiloženy na CD v adresáři */zdrojove_kody/DCIx/scripts/*.

Všechna oprávnění jsou načítána v realmu (*DciAuthcAndAuthzRealm*) podle rolí, ke kterým je uživatel přiřazen. Pro načtení dat z výše popsaného databázového pohledu

bylo nutné vytvořit 4 nové třídy zajišťující s frameworkem Alfa objektově relační mapování.

- *Permission.java* – Reprezentuje jednu položku z databáze. Obsahuje řetězec s názvem oprávnění a jméno role.
- *PermissionCollection.java* – Kolekce předchozího objektu.
- *PermissionTable.java* – Vytváří a naplňuje objekty *Permission* z dat v databázi. Zároveň definuje, jak bude vypadat klausule *where*.
- *PermissionMapping.java* – Určuje názvy sloupců a databázového objektu, z kterého se budou data číst.

Aby se nemuselo přistupovat při každém požadavku do databáze, jsou oprávnění kešována. Apache Shiro nabízí rovnou k použití modul založený na knihovně Ehcache. Potřebnou konfiguraci je nutné uvést v konfiguraci Shiro – tedy ve *web.xml*, kde se uvede cesta k xml souboru obsahující konfigurační parametry (*ehcache.xml*). Po dohodě byly nastaveny tyto parametry: nevyprchatelná a stále v paměti. Dostupné možnosti konfigurace byly čerpány ze zdroje (8).

5.3.2. Kontrola požadavku na stránku

Pro kontrolu oprávnění při požadavku na stránku musel být nadefinován nový autorizační servlet filtr (*SecurityFilter.java*). Ve *web.xml* je namapovaný jen pro url končící "*.do*", tyto url jsou vždy spjaty s nějakým servletem. Mapování servletu je nadefinováno v konfiguračním parametru pro Apache Shiro v části *[url]*. V této části jsou také uvedeny adresy přístupné i nepřihlášeným uživatelům. Toto je zajištěno použitím vestavěného filtru z knihovny Apache Shiro "*anon*". Při vyhodnocování, jaký filtr bude použit, je aplikována metoda „první shoda vítězí", proto je důležité nadefinovat filtry ve správném pořadí:

```
1. /startPage.do = anon          # úvodní obrazovka aplikace
2. /logon.do = anon              # přihlášení uživatele
3. /clearCache.do = anon         # servlet sloužící k vyprázdnění cache
4. /t_logon.do = anon            # přihlášení uživatele přes telnet
5. /*.do = securityFilter        # ostatní url prochází přes securityFilter
```

Ve třídě SecurityFilter je nejprve ověřeno, jestli je aktuální subjekt autentifikovaný, pokud tomu tak není, je přesměrován na přihlašovací stránku (*logon.do*). V druhém případě je ověřeno, jestli je url přístupná všem přihlášeným uživatelům. Tato kontrola závisí na tom, jestli je požadovaná url v databázi, pokud se url v databázi nenachází je přístup povolen. Jedná se zejména o adresy na obecné servlety jako například: kontrola nových soukromých zpráv nebo o adresy na podstránky, například: detail položky produktu. Pokud se url v databázi nachází, je potřeba provést kontrolu specifického oprávnění a je složen řetězec popisující daný požadavek, který lze vždy roztřídit do následujících skupin:

- Http požadavek obsahuje parametr "definitionName" – jedná se o oprávnění na příkaz (order). Řetězec má tvar "*orderName:"+definitionName*
- Http požadavek obsahuje parametr *"transactionDefinitionID"* a zároveň se toto ID nerovná -1. Řetězec je ve tvaru "*transactionID:*" + Id
- Požadavek směřuje na servlet "*reportManager.do*" a parametr "*reportId*" není *null*. Je složen řetězec "*reportID:*" + Id
- Požadavek neodpovídá předchozím pravidlům. Kontroluje se přímo url. Řetězec je ve tvaru "*url:*" + url

Takto vytvořený řetězec je předán jako parametr metodě z knihovny Apache Shiro *subject.checkPermission(String opravneni)*. Pokud uživatel překročil svá práva, metoda vyhodí výjimku, která je v další části aplikace zachycena a zobrazena uživateli.

5.3.3. Kontrola oprávnění v kódu

Jak již bylo zmíněno, v aplikaci DCIx byla kontrola dostupných oprávnění prováděna většinou na jméno role. Každá role má přiřazenou určitou skupinu práv a podle toho může v aplikaci provádět změny. Pokud se kontroluje jen jméno role, musí práva zůstat neměnná a musí existovat dokumentace určující, co daná role může provádět. Pokud se ovšem kontroluje přímo specifické oprávnění, poskytuje to svobodu měnit práva rolí.

V aplikaci reprezentuje oprávnění finální třída *MenuKey*, která obsahuje statické konstanty, které pomocí unikátního řetězce popisují určité oprávnění. Tyto řetězce korespondují se záznamy v databázi v tabulce *Menu*. Vazební tabulka

Menu_Item_For_Role určuje jaké položky tabulky *Menu* jsou přístupné jakým rolím. Tato tabulka je součástí pohledu *Permission_View*, ze kterého se v realmu načítají dostupná oprávnění pro přiřazené role aktuálního uživatele.

Pro eliminaci kontroly na jméno role, musela být vytvořena nová oprávnění (nové položky jak v databázi, tak ve třídě *MenuKey*). Před samotným přidáním nových položek bylo nutné rozšířit sloupec *Menu_Key* tabulky *Menu* a *Menu_Item* z definovaných 20 znaků na 40. To vyžadovalo smazání všech klíčů referencujících sloupec a po změně je opět nadefinovat. Skript je dostupný na přiloženém CD v adresáři */zdrojove_kody/DCIx/scripts* spolu s dalšími T-SQL skripty ovlivňující databázi. Na přidání nové položky menu, tedy i nového oprávnění, existuje v databázi procedura, která se postará o vygenerování ostatních sloupců a vloží záznamy do tabulek *Menu* i *Menu_Item*. Tato procedura však automaticky přidávala nově vzniklá oprávnění i k roli *Aimtec.Administrator*, proto musela být tato vazba zrušena a vytvořena nová, přiřazená k roli, ke které toto oprávnění patří. Na následujícím T-SQL skriptu lze vidět přidání nového oprávnění na přihlášení do více společností. Předchozí implementace ověřovala v kódu jen, jestli má daný uživatel přiřazenou roli *Multicompany.Administrator*. Aby byla logická funkčnost zachována, je nově vytvořené oprávnění přidáno k právě této roli. Celkem bylo vytvořeno osm nových oprávnění, které musely být takto přiřazeny k roli, ke které se vztahovaly.

```
-- vytvoření nové položky menu (nové oprávnění)
-- paramtry: řetězec Menu_Key, url, parametry url, oprávnění (0-oprávnění,
-- 1-položka menu), popis menu, rodičovský Menu_Key, sekvenční číslo,
-- role k přiřazení
EXEC dciowner.addToMenu 'multiCompany.login', null, null, 0, 'Company',
                         'compSettings', 10, 'Multicompany.Administrator'

-- smazání nově vytvořené položky pro roli Aimtec.Administrator
-- (vždy Role_ID=1)
DELETE from dciowner.Menu_Item_For_Role
      where Role_ID=1 and Menu_Item_ID=(select ID from dciowner.Menu
                  where Menu_Key like 'multiCompany.login' )

--přiřazení položky k roli Multicompany.Administrator
INSERT INTO dciowner.Menu_Item_For_Role
      (Role_ID,Menu_Item_ID,Visible,Active)
      SELECT dciowner.Roles.ID as Role_ID,
            (SELECT dciowner.Menu.ID from dciowner.Menu where
                  dciowner.Menu.Menu_Key like 'multiCompany.login') as
                  Menu_Item_ID,1 as Visible,1 as Active
      FROM dciowner.Roles
      WHERE Roles.Description like 'Multicompany.Administrator'
```

Samotná kontrola oprávnění v java kódu již probíhá zavoláním metody z knihovny Apache Shiro - *SecurityUtils.getSubject().isPermitted(MenuKey.OPRAVNENI.value()).* V JSP souborech byl nahrazen většinou přítomný skriplet tagem *<shiro:hasPermission name='<%=MenuKey.OPRAVNENI.value() %>'> </shiro:hasPermission>*. Tato změna kontroly oprávnění postihla více jak 80 souborů. Na obrázku 11 lze vidět vlevo ověření oprávnění před použitím knihovny Apache Shiro a vpravo s použitím knihovny.

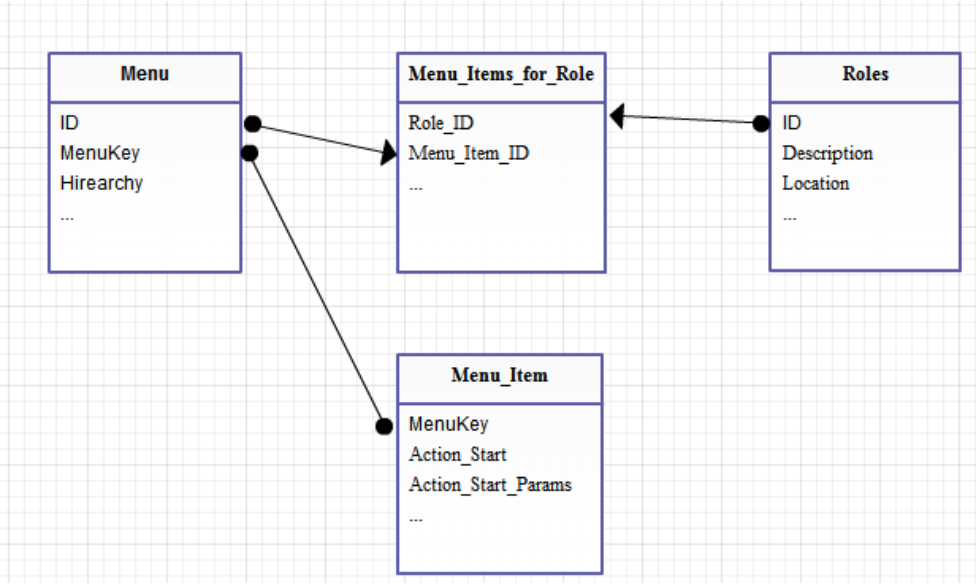
```
<bean:define id="usr" name="wrk" property="currentUser" type="User"/>

<% if (usr.isMenuItemAllowed(MenuKey.EDIT_BARCODE)) { %>
        <skin:action type="button" styleClass="formButton"
                page="/editBarCode.do?action=Edit" paramId="id" param
                paramProperty="id" winWidth="400" winHeight="200" key
        <hr />
<% } %>
<skin:action type="button" styleClass="formButton" key="button.close"
                page="/viewBarCode.do?action=Close" accesskeyPos="0"
```

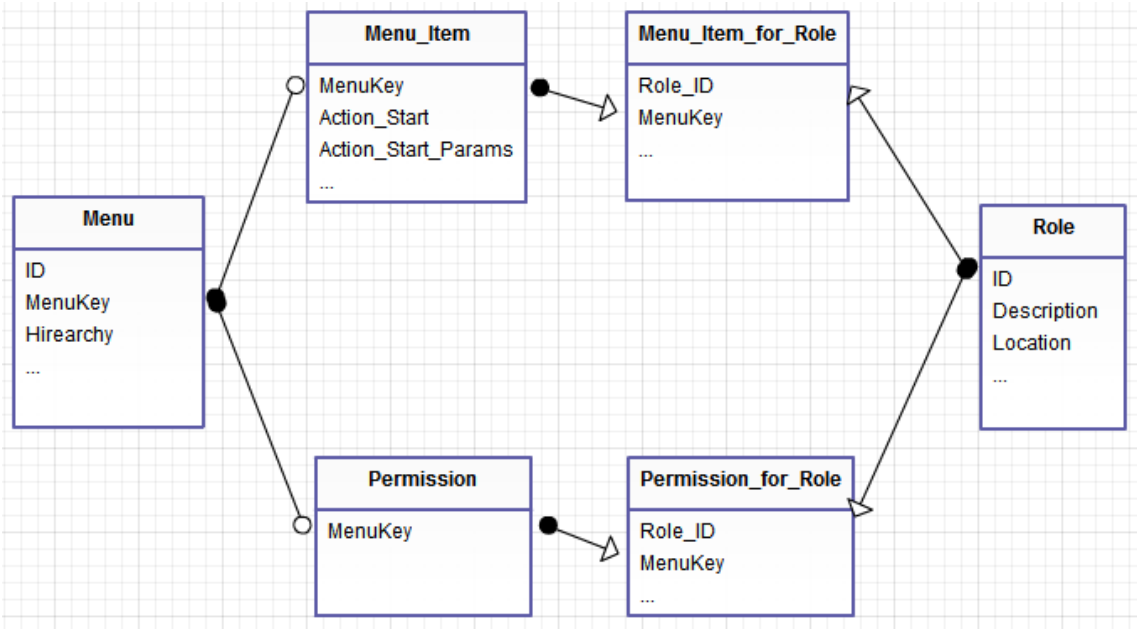
```
<shiro:hasPermission name='<%=MenuKey.EDIT_BARCODE.value() %>'>
        <skin:action type="button" styleClass="formButton"
                page="/editBarCode.do?action=Edit" paramId="id"
                paramProperty="id" winWidth="400" winHeight="200
        <hr />
</shiro:hasPermission>
<skin:action type="button" styleClass="formButton" key="button.c
                page="/viewBarCode.do?action=Close" accesskeyPos
```

Obrázek 11 Porovnání starého a nového ověření oprávnění v jsp kódu

Původní návrh autorizace zahrnoval přepracování referenčních integrit v tabulkách *Menu* a *Menu_Item*, které mezi sebou měli vazbu 1:1. Měla se vytvořit nová tabulka *Permission*, která by obsahovala jen oprávnění. Původní strukturu tabulek lze vidět na obrázku 12. Naopak obrázek 13 ukazuje novou strukturu a propojení tabulek.



Obrázek 12 Původní struktura a vazby mezi tabulkami



Obrázek 13 Navržená struktura a propojení tabulek

Nová struktura spočívala v tom, že by všechny položky z tabulky *Menu_Item*, které jsou oprávnění (tzn. sloupec *Is_Visible* = 0), byly přesunuty do nově vzniklé tabulky *Permission*. Zrušilo by se propojení 1:1 mezi tabulkami *Menu* a *Menu_Item*, místo toho by se vytvořilo spojení 1 : 0..1 mezi tabulkami *Menu - Permission* a *Menu - Menu_Item*. T-SQL skripty vytvářející nové tabulky a přesouvající položky *Menu_Item* do tabulky *Permission* a *Permission_For_Role* jsou dostupné na přiloženém CD v adresáři */zdrojove_kody/DCIx/scripts/not used/*. Od návrhu bylo pro složitost reimplementace java a JSP kódu upuštěno.

6. Závěr

Aplikace DCIx před reimplementací porušovala některé bezpečnostní zásady. Hlavní problém spočíval v nekontrolování uživatelských oprávnění při zaslání požadavku na server. Pokud uživatel znal cílovou adresu pro editaci položek v aplikaci, nic mu nebránilo akci provést. Tento problém byl odstraněn díky vhodné konfiguraci knihovny Apache Shiro a nově vzniklým servlet filtrem, který kontroluje každý požadavek směřující na servlet. Zároveň bylo z kódu odstraněno ověřování práv na jména rolí a nahrazeno vhodnějším řešením spočívající v kontrole přímo specifického oprávnění. Pro tento účel bylo v databázi založeno několik nových položek menu, které reprezentují oprávnění. Za zmínku stojí například "*transactionID:**" využívající možnosti *WildcardPermission* a umožňuje prohlížení jakékoli transakce. Pomocí vestavěné JSP/GSP tag knihovny je nyní kontrola oprávnění v JSP kódu jednodušší. Většinou složité skriplety, ve kterých se muselo načítat množství dat, jen aby se zjistilo, zda uživatel má právo na danou položku, jsou nyní nahrazeny jednoduchými tagy s použitím JSP expression.

Pro správné fungování autorizace s použitím Apache Shiro bylo nutné reimplementovat i autentifikaci. Kód starající se o celý proces přihlášení byl velice nepřehledný a zastaralý. Nyní je použito moderních programovacích technik a přihlašovací proces je mnohem přehlednější a využívá možnosti Apache Shiro.

Protože funkčnost aplikace musela být zachována, bylo nutné provádět v každé fázi vývoje akceptační automatické testy. Pro autentifikaci bylo stěžejní, aby úspěšně procházel test *allLoginScenarios.xml*, který kontroluje všechny stavy, které mohou při přihlašování nastat. Reimplementace autorizace zasáhla více souborů a bylo nutné, aby všechny zbývající testy procházely bezchybně, maximálně s mírnými úpravami.

Původní návrh reimplementace se v průběhu vývoje značně měnil. Vzhledem ke složitosti a komplikovanosti celé DCIx aplikace bylo poměrně těžké odhadnout práci potřebnou k jednotlivým úkonům. Například první návrh autorizace počítal s načítáním určitých skupin adres spadající pod jedno hromadné oprávnění ze souboru xml. Až při naprogramování funkčního prototypu se zjistilo, že aplikace se tímto načítáním značně

zpomaluje a upustilo se od tohoto návrhu. Další slepou větví byla snaha o rozčlenění tabulky *Menu_Item*, která obsahovala, jak řetězce popisující oprávnění, tak i adresy url sloužící k navádění v menu. Až po vytvoření T-SQL skriptů bylo zjištěno, že vazba jedna ku jedné mezi tabulkou *Menu* a *Menu_Item* je v aplikaci značně používaná.

Knihovna Apache Shiro poskytla vhodné nástroje k reimplementaci celé bezpečnostní vrstvy. Umožnila i pokročilé funkce jako je například kešování nebo konfigurace servletového filtru. Všechny hlavní komponenty jsou založeny na POJO, a proto je snadné v případě potřeby naprogramovat komponentu svou a v konfiguraci ji použít.

7.Zdroje

1. **AIMTEC a.s.** Logistický informační systém DCIx. *Aimtec.* [Online] [Citace: 16. prosinec 2012.] http://www.aimtec.cz/cs/produkty/dcix.html.

2. **CCB spol. s r.o.** Systemonline.cz - DCIx. [Online] [Citace: 12. 3 2013.] http://www.systemonline.cz/prehled-informacnich-systemu/aps-systemy/dcix-1.htm. ISSN 1802-615X.

3. **The Apache Software Foundation.** Apache Shiro. *Apache Shiro.* [Online] [Citace: 16. prosinec 2012.] http://shiro.apache.org/.

4. **Les Hazlewood, Craig Russell.** JSecurityProposal. [Online] [Citace: 28. 3 2013.] http://wiki.apache.org/incubator/JSecurityProposal.

5. **UserGroupsatGoogle.** *Super Simple Application Security with Apache Shiro.* [http://www.youtube.com/watch?v=5ZepGFzYHpE] 2010.

6. **Oracle.** JAAS Reference Guide. *JavaTM Authentication and Authorization Service (JAAS).* [Online] [Citace: 22. 4 2013.] http://docs.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html#Authentication.

7. **Alex, Ben a Taylor, Luke.** Spring Security. [Online] [Citace: 22. 4 2013.] http://static.springsource.org/spring-security/site/features.html.

8. **Terracotta, Inc.** EHCACHE. *Ehcache | Documentation | Configuration.* [Online] Software AG USA, Inc. . [Citace: 20. 4 2013.] http://ehcache.org/documentation/user-guide/configuration.

8. Slovníček pojmů

Apache Tomcat	software umožňující spouštět java servlety a zpracovávat JSP soubory
Autentifikace	ověření identity
Autorizace	proces řízení přístupu
Callback	část spustitelného kódu, předána jako parametr metodě a očekává se zavolání zpět tohoto kódu metodou při určité situaci
CRUD operace	operace pro perzistenci objektu - vytvoření, čtení, úprava a mazání
DCIx	aplikace vyvíjena firmou Aimtec a.s. zejména pro skladové hospodářství
HtmlUnit	knihovna pro práci s html stránkami
Jameleon	nástroj k provádění automatických testů
Java EE	platforma jazyka java pro tvorbu podnikových aplikací
Javadoc	vygenerovaná programová dokumentace přímo ze zdrojových kódů
Jist in Time (JIT)	metoda optimalizace skladu založena na principu dodávek právě včas
JSP	Java Server Pages - technologie umožňující vytvářet dynamické webové stránky
JSP/GSP tag knihovna	knihovna obsahují sadu tagu pro použití v JSP souborech
JUnit	framework pro psaní automatických testů
Keš (cache)	informace uložené přímo v paměti pro rychlý přístup
Manufacturing Execution Systém (MES)	výrobní informační systémy
Realm	komponenta frameworku Apache Shiro sloužící k načtení bezpečnostních dat
Structs	framework pro webové aplikace
T-SQL	Transact-SQL - rozšíření jazyka SQL od společnosti Microsoft
Warehouse Management Systém (WMS)	systém skladového hospodářství
WildcardPermission	oprávnění definováno řetězcem s možným větvením
xmlParser	knihovna pro snadnější práci s xml soubory

9.Přílohy

9.1. Uživatelská dokumentace aplikace ShiroSampleApp

ShiroSampleApp je velice jednoduchá a základní aplikace využívající možnosti Apache Shiro. Ke spuštění vyžaduje Apache Tomcat ve verzi 6.

Pro vstup do aplikace slouží adresa /login, která zobrazí jednoduchý formulář pro zadání jména a hesla. Uživatelská jména a hesla jsou pevně nakonfigurována v aplikaci a nelze je za běhu měnit. Následující tabulka 1 zobrazuje přihlašovací jména, hesla a přiřazené role jednotlivých uživatelů.

Uživatelské jméno	heslo	Přiřazená role
admin	admin	role2
user1	user1	role1
user2	user2	role3

Tabulka 1 Seznam uživatelů, jejich hesel a rolí

Tabulka 2 zobrazuje přiřazená práva k jednotlivým rolím a podle toho dostupné stránky.

Název role	WildcardPermission	dostupné stránky
role1	permission1	/account/permission1.jsp /account/role1.jsp
role2	*	/account/permission1.jsp /account/permission2.jsp /account/role2.jsp
role3	permission2	/account/permission2.jsp

Tabulka 2 Přiřazená práva a dostupné stránky

Pokud uživatel zadá správně jméno a heslo, je přesměrován na adresu /account, která zobrazí jen dostupné odkazy podle přiřazených práv. Pokud zadá heslo špatně, je znovu přesměrován na přihlašovací obrazovku.