VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ŘÍZENÍ PŘÍSTUPU VE WEBOVÝCH APLIKACÍCH

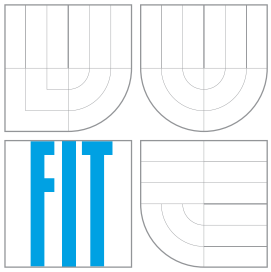
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN PEŠEK

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ŘÍZENÍ PŘÍSTUPU VE WEBOVÝCH APLIKACÍCH

ACCESS CONTROL IN WEB APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN PEŠEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. ZBYNĚK KŘIVKA, Ph.D.
SUPERVISOR

BRNO 2008

Zadání bakalářské práce/3486/2007/xpesek07

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Pešek Martin**
Obor: Informační technologie
Téma: **Řízení přístupu ve webových aplikacích**
Kategorie: Databáze

Pokyny:

1. Seznamte se s problematikou řízení přístupu ve webových aplikacích, komponentním přístupem k programování a využívanými metodami autentizace a autorizace.
2. Navrhněte datové struktury a programové rozhraní pro komponentu pro řízení přístupu (inspirujte se funkcionalitou ACL) v aplikacích s vícevrstvou architekturou. Zvažte integraci komponenty do některého existujícího frameworku (např. PEAR, Zend, atd.).
3. Implementujte tuto komponentu a její funkčnost demonstrujte na internetové aplikaci odpovídající velikosti se zaměřením na demonstraci kladných i záporných vlastností vašeho přístupu.
4. Výsledky vašeho návrhu a implementace v závěru diskutujte. Popište výhody i nevýhody vašeho přístupu.

Literatura:

- Smith, R. E.: Authentication, Addison-Wesley, Boston, 2002, 549 s.
- Gutmans, A., et. al.: Mistrovství v PHP 5 [překlad Bogdan Kiszka], CP Books, Brno, 2005, 655 s.
- Hugh, W. E., Lane, D.: Web database applications with PHP and MySQL, O'Reilly, 2002, 563 s.
- Guerraoui, R.: Object-oriented programming, Springer-Verlag, Berlin, 1999, 528 s.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Křivka Zbyněk, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. **Pan**

Jméno a příjmení: **Martin Pešek**
Id studenta: 79128
Bytem: Křivánky 577/9, 642 00 Brno
Narozen: 06. 09. 1985, Brno
(dále jen "autor")

a

2. **Vysoké učení technické v Brně**

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Řízení přístupu ve webových aplikacích
Vedoucí/školitel VŠKP: Křivka Zbyněk, Ing., Ph.D.
Ústav: Ústav informačních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Abstrakt

Tato práce se zabývá problematikou řízení přístupu ve webových aplikacích a využívanými metodami autentizace a autorizace. Popisuje proces návrhu a implementace komponenty pro řízení přístupu ve webových aplikacích vyvíjených v jazyce PHP 5.

Klíčová slova

webová aplikace, řízení přístupu, autentizace, autorizace, ACL, bezpečnost, MVC, webový aplikační rámec, PHP, MySQL

Abstract

This thesis deals with problems of access control in web applications and used methods of authentication and authorization. It describes the process of the design and the implementation of the component for access control in web applications developped in PHP 5.

Keywords

web application, access control, authentication, authorization, ACL, security, MVC, web application framework, PHP, MySQL

Citace

Martin Pešek: Řízení přístupu ve webových aplikacích, bakalářská práce, Brno, FIT VUT v Brně, 2008

Řízení přístupu ve webových aplikacích

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zbyňka Křivky, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Pešek
13. května 2008

Obsah

Kapitola 1

Úvod

V současné době zažívají webové aplikace obrovský rozmach, velké množství desktopových aplikací se stěhuje do webového prostředí. Hlavním důvodem jejich obliby je zejména „všudypřítomnost“ webového prohlížeče, který je pro přístup k webovým aplikacím určen. Rychlému rozšíření webových aplikací napomáhá také stále lepší podpora a prostředky pro jejich vývoj.

Tím, že je webová aplikace dostupná prakticky odkudkoliv, je ale velmi důležité omezit přístup k ní jen na určité uživatele. Právě tento důležitý aspekt, tedy řízení přístupu uživatelů k webové aplikaci, je hlavní náplní této práce. Kapitola 2 obsahuje stručný úvod do problematiky webových aplikací. Zabývá se zejména používanými architekturami a dostupnými prostředky pro jejich tvorbu.

S řízením přístupu souvisí dva základní pojmy: autentizace a autorizace. Autentizací se rozumí ověřování identity uživatele. Procesem autentizace se zabývá kapitola 3. Popisuje nejprve obecné principy a způsoby provádění autentizace, postupně pak přechází ke konkrétním metodám využívaným ve webových aplikacích.

Autorizace zpravidla následuje po úspěšné autentizaci a ověřuje, zda má uživatel dostatečná oprávnění přístupu k nějakému souboru, či k provedení nějaké akce. Používané způsoby autorizace jsou popsány v kapitole 4.

Kromě teoretického zpracování problematiky řízení přístupu ve webových aplikacích je úkolem této práce vytvořit v jazyce PHP 5 funkční a snadno použitelnou komponentu pro řízení přístupu. Popis jejího návrhu a implementace se nachází v kapitole 5. Funkčnost vytvořené komponenty je otestována a demonstrována na ukázkové aplikaci, sloužící jako knihovna odborných článků. Tvorbou této aplikace se zabývá kapitola 6.

V závěru jsou pak shrnuty a vyhodnoceny výsledky této práce.

Kapitola 2

Webové aplikace

Webová aplikace [1] je aplikace typu klient-server poskytovaná z webového serveru prostřednictvím protokolu HTTP. Uživatelé k ní přistupují zpravidla přes webový prohlížeč, jako je Internet Explorer, Mozilla nebo Opera.

Vysoká popularita webových aplikací je způsobena zejména všudypřítomností webového prohlížeče jako klienta, nazývaného tenkým klientem, jelikož logika aplikace je mu skryta. Klient na základě interakce s uživatelem odesílá požadavky na webový server, který následně odpovídá zasláním webové stránky. Tato stránka je zapsána zpravidla jazykem (X)HTML.

Každá webová aplikace musí dodržovat určitou množinu požadavků [2, 3]. Jedná se jak o požadavky funkční, tedy jaké funkce daná aplikace nabízí, tak zejména o požadavky na kvalitu, výkonnost a dostupnost. Tyto požadavky se typicky velmi rychle mění a jsou závislé na typu a povaze konkrétní aplikace, nicméně mezi základní požadavky, společné všem aplikacím, patří:

- výběr vhodné architektury,
- dostatečné zabezpečení,
- řízení přístupu
  - autentizace uživatele,
  - autorizace uživatele

Problematikou procesu autentizace uživatelů a s ním spojeným zabezpečením se podrobně zabývá kapitola 3, autorizací pak kapitola 4.

2.1 Architektura webových aplikací

Architektura podle [3] popisuje strukturu, která se dekomponuje na komponenty, ty dále na svá rozhraní a vzájemné vztahy; formuje přechod od analázy k návrhu. Vhodná architektura činí systém srozumitelnějším, protože dekompozice systému do menších celků usnadňuje jeho správu, odhalování chyb, umožňuje lépe řídit vývoj aplikace. Volba architektury je proto důležitým faktorem pro tvorbu webové aplikace. Při výběru špatné architektury se mohou objevit problémy se znovupoužitelností vytvořených komponent, rozšiřitelností a udržovatelností aplikace.

Podle [2] se v literatuře (zvlášť týkající se platformy J2EE [4]) objevují termíny „Model 1“ a „Model 2“, které označují dva základní přístupy k architektuře webové aplikace. Je možné je snadno zobecnit na jakoukoliv platformu určenou pro tvorbu webových aplikací.

Model 1

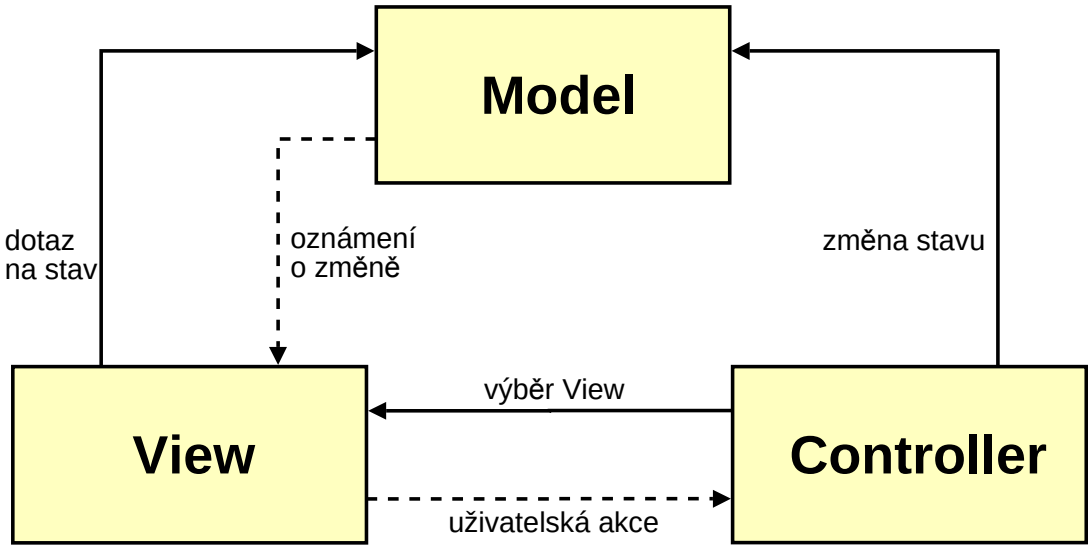
Pro architekturu *Model 1* platí, že prohlížeč přistupuje k jednotlivým stránkám aplikace přímo, a to přes URL, na kterém se skutečně nacházejí. Řízení aplikace založené na tomto modelu je decentralizované. Model 1 neodděluje uživatelské rozhraní od logiky aplikace, odděluje pouze přístup k datovému zdroji. Tato architektura je vhodná pouze pro statické webové prezentace, případně pro velmi jednoduché aplikace.

Model 2

V případě *Modelu 2* se objevuje pojem Controller, který slouží jako prostředník mezi klientem a požadovanými stránkami. Controller centralizuje logiku vyhodnocení požadavků na aplikaci. Na základě přijatého požadavku, otevíraného URL, aktuálního stavu aplikace a případných vstupních parametrů provádí příslušnou akci a rozhoduje, která stránka bude odeslána klientovi jako odpověď. Pro většinu webových aplikací je vhodné použít právě tuto architekturu.

2.1.1 Architektura Model-View-Controller

Architektura *Model-View-Controller* (MVC) [2, 5, 6], neboli model-pohled-kontrolér, je oblíbeným a často používaným vzorem pro tvorbu webových aplikací. V podstatě odpovídá výše popsanému Modelu 2. Rozděluje aplikaci do tří nezávislých logických částí (Model, View, Controller) tak, že změna některé z nich má minimální vliv na ostatní. Vztahy mezi jednotlivými částmi MVC jsou znázorněny na obrázku 2.1.



Obrázek 2.1: Vztahy mezi částmi architektury MVC

Model

Model zapouzdřuje datovou logiku aplikace, je však nejen jejím datovým základem, ale zejména základem funkčním [7]. Je možné si model představit jako množinu různých stavů, do kterých se může aplikace dostat. Jakákoliv událost či akce uživatele představuje přechod do jiného stavu modelu.

Model poskytuje prostředky pro zjištění a změnu aktuálního stavu aplikace a pro přístup k datovému zdroji. Musí být jako celek zapouzdřený a musí nabízet své pevně dané rozhraní. Implementaci modelu je vhodné provádět pomocí objektových tříd.

Pohled

Pohled (*View*) zobrazuje stav modelu, provádí grafický výstup aplikace. U webových aplikací generuje (X)HTML kód, který je odesílán klientovi jako odpověď na jeho požadavky.

Informaci o změně stavu modelu může pohled získat dvojím způsobem. Buď se sám obrací na model, vždy když potřebuje získat aktuální data, nebo si nechává od modelu zasílat oznámení o změnách.

Vzhledem k tomu, že zobrazený výstup, tedy HTML stránka, slouží pro odchytávání událostí od uživatele, předává pohled tyto události kontroléru.

Kontrolér

Kontrolér (*Controller*), neboli řadič či ovladač, se stará o zpracování vstupů a událostí od uživatele, definuje chování celé aplikace. U webových aplikací jsou vstupem zpravidla požadavky GET a POST protokolu HTTP.

Na základě přijatých požadavků kontrolér volá akce modelu, mění jeho stav atd. Podle výsledku volané akce modelu a podle přijatého požadavku vybírá pohled, který má být odeslán zpět uživateli.

Speciálním typem kontroléru je tzv. *Front Controller*. Je to centralizovaný kontrolér, který zpracovává všechny příchozí požadavky.

Výhody MVC architektury

Mezi hlavní výhody MVC podle [2, 5] patří:

- snadná udržovatelnost a rozšiřitelnost aplikace,
- možnost vyvíjet jednotlivé části aplikace samostatně,
- znovupoužitelnost vytvořeného kódu,
- snadné zpřístupnění aplikace pro různé druhy klientů.

Příklady implementací MVC

Poprvé byl tento architektonický vzor použit v programovacím jazyce Smalltalk. Rozšířil se však zejména v oblastni platformy J2EE (Java Enterprise Edition) společnosti Sun Microsystems. Implementaci MVC lze v současné době nalézt také u ASP.NET od společnosti Microsoft.

Pro jazyk PHP existuje mnoho různých frameworků založených na architektuře MVC. Zřejmě nejpoužívanějším je Zend Framework, mezi další patří například CakePHP.

2.2 Platformy pro tvorbu webových aplikací

Pro tvorbu a vývoj webových aplikací existuje několik platforem. Tato sekce obsahuje základní popis tří nejrozšířenějších.

Nejprve je však potřeba objasnit pojem framework. *Webový aplikační framework* [2, 8] je ucelený balík vzájemně provázaných knihoven, které poskytují prostředky pro pokrytí společných požadavků webových aplikací. Výsledná aplikace je pak vyvíjena uvnitř frameworku.

2.2.1 PHP

PHP [9] (původně Personal Home Pages, dnes PHP: Hypertext Preprocessor) je velmi rozšířený skriptovací programovací jazyk určený především pro tvorbu webových aplikací. Nejčastěji je používán v kombinaci s webovým serverem Apache [10] a databázovým serverem MySQL [11].

Hlavní výhodou jazyka PHP je zejména jeho jednoduchost. I přes vylepšenou podporu objektově orientovaného programování ve verzi PHP 5 je ale knihovna základních funkcí PHP procedurální, téměř nevyužívá objekty. Další nevýhodou je absence zabudovaných komponent a vývojářských nástrojů. Nejen z těchto důvodů je použití jazyka PHP jako takového nevhodné pro složitější webové aplikace.

Naštěstí existuje mnoho různých frameworků pro PHP, které v určité míře nedostatky jazyka odstraňují a umožňují tak relativně snadnou tvorbu i rozsáhlejších webových aplikací. Vhodné jsou zvláště frameworky postavené na architektuře MVC.

Repositář PEAR

PEAR [12] (PHP Extension and Application Repository) je systém knihoven a rozšíření pro aplikace psané v jazyce PHP. Nejedná se o framework v pravém slova smyslu, ale spíše o jakýsi repositář nezávislých knihoven. Tvůrce webové aplikace si z nabízených knihoven vybere jen ty, které potřebuje.

Za všechny poskytované komponenty zmíním balík `DB`, který mimo jiné obsahuje stejnojmennou třídu zajišťující jednotný objektově orientovaný přístup ke všem podporovaným databázovým systémům.

Zend Framework

Zend Framework [13] je aplikační framework pro vývoj webových aplikací v jazyce PHP 5.

Tento framework plně využívá architektonického návrhového vzoru MVC. Jedním z jeho hlavních cílů je nabízet knihovny, jenž jsou vzájemně provázané s ostatními komponentami tohoto frameworku.

Zend Framework poskytuje propracovaný systém prostředků pro tvorbu webové aplikace, zároveň je jeho použití relativně jednoduché. Samozřejmostí je objektově orientovaný přístup ke všem dostupným komponentám.

2.2.2 Java Enterprise Edition

Java Enterprise Edition (J2EE) [14] je součást platformy Java. J2EE je určená pro vývoj podnikových aplikací a informačních systémů. Součástí této platformy je mimo jiné specifikace pro vývoj webových aplikací. Snadnost a efektivnost využití jednotlivých technologií,

především v serverové části J2EE, nabízí odstínění od mnoha problémů, které by jinak vývoj aplikace zbytečně zpomalovaly. J2EE umožňuje soustředit se primárně na úlohy samotné aplikace.

Základní technologie platformy J2EE jsou tyto:

- *JSP* (JavaServer Pages) – jsou určeny ke generování HTML stránek odesílaných klientovi jako odpověď na jeho požadavek.
- *JSF* (JavaServer Faces) – slouží pro tvorbu uživatelského rozhraní aplikace na straně serveru.
- *Servlety* – jsou nástrojem pro obsluhu protokolu HTTP na straně serveru.
- *EJB* (Enterprise JavaBeans) – je standardní komponentní architektura, sloužící pro realizaci vlastní aplikační logiky systému.

2.2.3 .NET

.NET [15] je platforma vyvinutá společností Microsoft. Základní komponentou je *.NET Framework*, což je prostředí nutné pro běh aplikací, nabízí všechny potřebné knihovny. .NET není určen zdaleka jen pro vývoj webových aplikací; prostředky pro tvorbu webových aplikací jsou jen jednou z mnoha jeho součástí. Při omezení pouze na serverovou část je možné .NET označit za obdobu J2EE.

Základem koncepce .NET je *Common Language Runtime* (CLR), což je prostředí pro provádění všech .NET aplikací. Platformou .NET není předepsáno, jaký programovací jazyk je nutné pro tvorbu aplikace použít, protože bez ohledu na to v čem je aplikace napsána, je přeložena do společného mezikódu. Nejpoužívanějšími jazyky jsou však C# a Visual Basic .NET.

Pro tvorbu webových aplikací a služeb je k dispozici technologie *ASP.NET* [16], která je nástupcem technologie ASP (Active Server Pages), jedná se však o odlišné technologie. Při srovnání s J2EE je ASP.NET konkurentem JSP.

Kapitola 3

Autentizace

Autentizace [17, 18] je obecně proces ověřování identity protější strany, neboli zda je uživatel či entita opravdu tím, za koho se vydává.

Podle [2, 18] se principielně rozlišuje mezi autentizací uživatelskou a entitní:

- *Uživatelská autentizace* je obvykle prováděna v okamžiku přihlašování uživatele do systému. Po úspěšném přihlášení obdrží uživatel nějaký jedinečný token.
- *Entitní autentizace* probíhá společně s každým dalším požadavkem, který následuje po uživatelské autentizaci. Identita je ověřována na základě tokenu, který uživatel obdržel po úspěšném procesu uživatelské autentizace.

3.1 Základní způsoby autentizace

Proces uživatelské autentizace probíhá na základě určitých údajů, které autentizujícího se uživatele jednoznačně identifikují. Rozdělení způsobů identifikace uživatelů je podle [17, 19] následující:

- *Uživatel něco zná* – například heslo, PIN či šifrovací klíč.
- *Uživatel něco má* – například čipovou kartu.
- *Uživatel se něčím vyznačuje* – biometriky.

3.1.1 Autentizace na základě znalosti

Charakteristickou vlastností je znalost nějaké tajné informace, kterou ovšem ostatní lidé neznají. Většinou se jedná o heslo, PIN, šifrovací klíč a podobně.

Mezi výhody tohoto přístupu patří přenositelnost a snadná implementace. Jedná se o nejrozšířenější způsob autentizace, uživatelé jsou na něj zvyklí.

Nevýhodou je relativně snadná možnost odpozorování hesla. Problém může být také v hledání kompromisu mezi složitostí hesla a jeho zapamatováním. Uživatel by měl zvolit takové heslo, aby nebylo snadné jej uhodnout jinou osobou, avšak zároveň si jej musí bezpečně pamatovat.

3.1.2 Autentizace na základě vlastnictví

Tento přístup je založen na tom, že uživatel vlastní nějakou specifickou věc, označovanou v [17] jako „token“. Často se jedná o různé čipové karty.

Výhodou je, že oproti prvnímu přístupu je těžké takový token zneužít ve smyslu odpozorování nebo uhodnutí.

Nevýhodou je oproti heslu horší přenositelnost, cena a možnost ztráty či krádeže.

3.1.3 Autentizace na základě vlastností

Třetí způsob identifikace uživatele probíhá na základě jeho vlastností, které je možné nějakým způsobem měřit – tzv. *biometriky*. Biometrické systémy jsou pak schopny na základě jedinečnosti těchto vlastností identifikovat přímo člověka.

Takových měřitelných vlastností existuje mnoho, podle [20] je možné je rozdělit do dvou základních skupin – fyziologické a behaviorální. Do těchto skupin patří následující vlastnosti:

- Fyziologické:
  - *Otisk prstu* – jedná se o nejrozšířenější biometriku s mnohaletou historií. Snímání je navíc poměrně rychlé a pohodlné. Existují snímače optické, které využívají laserového světla, kapacitní, které měří kapacitní odpor v ploše dotyku prstu se snímací podložkou, a ultrazvukové.
  - *Geometrie ruky* – tato technika není tolik přesná jako otisky prstů, a proto je využívána pouze k verifikaci, nikoliv identifikaci.
  - *Rozpoznání obličeje* – jedná se o přirozenou metodu identifikace člověka pomocí analýzy snímku jeho obličeje.
  - *Oční duhovka*,
  - *oční sítnice* – hlavní výhodou při použití těchto biometrik je velmi dobrá přesnost, které ale odpovídá vysoká cena.
- Behaviorální:
  - *Ověřování hlasu* – identifikace probíhá analýzou vyslovené, předem určené, věty. Tato biometrika není příliš náročná, může být však ovlivněna aktuálním stavem identifikované osoby.
  - *Dynamika podpisu* – vychází z konvenčního podpisu. Nezkoumá se jeho vizuální podoba, ale styl napsání.
  - *Dynamika stisku kláves*.

3.2 Autentizace ve webových aplikacích

Ve webových aplikacích se téměř výhradně využívá k identifikaci uživatele jeho uživatelské jméno a heslo. U jednotlivých aplikací se pak liší zejména způsob získávání těchto údajů od uživatele a míra zabezpečení přenosu hesla mezi webovým prohlížečem a webovým serverem.

Tato kapitola popisuje a porovnává dva základní přístupy k získávání přihlašovacích údajů od uživatele, zároveň popisuje způsob provádění entitní autentizace u těchto dvou přístupů. Bezpečnostní aspekty autentizace jsou řešeny v sekci 3.3.

3.2.1 HTTP autentizace

Protokol HTTP, který slouží pro přenos dat, většinou HTML stránek, mezi webovým prohlížečem a webovým serverem, umožňuje provádět přihlášení uživatele pomocí uživatelského jména a hesla [21].

Při požadavku klienta na stránku, přístupnou jen autentizovanému uživateli, odešle server odpověď `401 Unauthorized` společně s hlavičkou `WWW-Authenticate`. Na příjem této hlavičky webový prohlížeč reaguje zobrazením dialogového okna pro přihlášení uživatele prostřednictvím jména a hesla.

Po zadání těchto údajů prohlížeč znovu pošle na webový server předchozí požadavek, k němuž ale připojí hlavičku `Authorization` se zadanými přihlašovacími údaji. Způsob reprezentace přihlašovacích údajů v této hlavičce se liší podle použité metody. Protokol HTTP nabízí dvě možné metody: *HTTP Basic* a *HTTP Digest*, jejichž popis se nachází dále v této kapitole.

Server platnost přijatých údajů ověří buď přímo v aplikaci, nebo na úrovni webového serveru [2]. Například u serveru Apache k tomu slouží moduly `mod_auth`, `mod_auth_db` a `mod_auth_dbm`. Prokázal-li uživatel úspěšně svoji identitu, odpovídá server zasláním odezvy `200 OK`, v opačném případě opětovně zasílá `401 Unauthorized`.

Po úspěšné autentizaci uživatele pak prohlížeč s každým dalším požadavkem na server zasílá i hlavičku `Authorization` s reprezentací uživatelova jména a hesla.

Nevýhodou, společnou pro obě metody, je nemožnost jakýmkoliv spolehlivým způsobem provést odhlášení uživatele z aplikace, neboli donutit prohlížeč k zapomenutí zadaných přihlašovacích údajů, aby je již nemohl dále posílat. Je sice možné podstrčit ze strany serveru hlavičku `401 Unauthorized`, nicméně bohužel ne každý prohlížeč na ní zareaguje zapomenutím přihlašovacích údajů.

Basic HTTP autentizace

Jedná se o historicky nejstarší typ autentizace ve webových aplikacích. Při pokusu prohlížeče o přístup k chráněné stránce zasílá server HTTP odpověď, jejíž hlavička obsahuje následující údaje:

```
HTTP/1.0 401 Unauthorized
WWW-Authenticate: Basic realm="Chranena oblast"
```

Obsah pole `realm` zobrazuje prohlížeč v přihlašovacím dialogu jako název oblasti, ke které uživatel přistupuje.

Jakmile uživatel zadá své jméno a heslo, odešle prohlížeč znovu předchozí požadavek doplněný o následující hlavičku:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Jméno a heslo jsou zapsány pomocí formátu Base64, tedy nezašifrovaně. Z tohoto důvodu je Basic HTTP autentizaci možné využívat pouze při přenosu dat přes šifrovaný kanál (SSL/TLS), což v praxi znamená použití protokolu HTTPS.

Digest HTTP autentizace

Digest autentizace [22] byla vyvinuta jako náhrada za Basic autentizaci, při které je heslo přes síť přenášeno v nezabezpečené podobě. Digest autentizaci je tedy vhodné použít tehdy, není-li pro přenos dat využit protokol HTTPS.

Při požadavku klienta o přístup k chráněné stránce server odešle odpověď, která mimo jiné obsahuje například následující údaje (je použit příklad z [22]):

```
HTTP/1.0 401 Unauthorized
WWW-Authenticate: Digest realm="testrealm@host.com",
                         qop="auth",
                         nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                         opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

Po zadání přihlašovacích údajů prohlížeč provede nejprve výpočet odpovědi k odeslání na server, následně odešle předchozí požadavek doplněný o hlavičku `Authorization`. Pro příklad uvedený výše, uživatelské jméno "Mufasa" a heslo "Circle Of Life" by vypadala takto:

```
Authorization: Digest username="Mufasa",
                       realm="testrealm@host.com",
                       nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                       uri="/dir/index.html",
                       qop=auth,
                       nc=00000001,
                       cnonce="0a4f113b",
                       response="6629fae49393a05397450978507c4ef1",
                       opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

Na serveru pak dojde k výpočtu odpovědi obdobným způsobem jako u klienta. Server však dosazuje správné heslo přihlašujícího se uživatele. Výsledek následně porovná s výsledkem přijatým od klienta.

Výpočet odpovědi na klientovi i na serveru probíhá takto:

$$\begin{aligned} HA_1 &= \mathrm{md5}(username : realm : password) \\ HA_2 &= \mathrm{md5}(method : uri) \\ response &= \mathrm{md5}(HA_1 : nonce : nc : cnonce : qop : HA_2) \end{aligned}$$

Proměnná *method* obsahuje typ HTTP požadavku, kterým se k požadované stránce přistupuje (například GET). Ostatní proměnné odpovídají stejnojmenným polím ve výše uvedené hlavičce `Authorization`. Podrobnější informace o tomto typu autentizace lze nalézt v [21, 22].

Podpora v PHP

V jazyce PHP 5 je možné HTTP autentizaci provádět pomocí standardních prostředků samotného jazyka [23]. K odesílání příslušných HTTP hlaviček slouží funkce `header()`. Pro získání zadaných přihlašovacích údajů lze využít prvků superglobálního pole `$_SERVER`.

Pro Basic HTTP autentizaci jsou k dispozici proměnné `$_SERVER['PHP_AUTH_USER']` a `$_SERVER['PHP_AUTH_PW']`, které obsahují uživatelem zadané jméno a heslo.

Podpora Digest HTTP autentizace je v PHP až od verze 5.1.0. Pro tento typ autentizace je nutné použít proměnnou `$_SERVER['PHP_AUTH_DIGEST']`, ve které je uložen obsah hlavičky `Authorization`, zaslané prohlížečem.

3.2.2 Formulářová autentizace

Druhým způsobem autentizace ve webových aplikacích je tzv. *formulářová autentizace* [2]. Tento typ autentizace je v současnosti mnohem častěji používaný než autentizace pomocí prostředků protokolu HTTP.

Základním rysem formulářové autentizace je přihlašování uživatelů pomocí formuláře, který se nachází přímo ve webové stránce. V případě potřeby zašle aplikace klientovi HTML stránku s takovým formulářem. Po odeslání formuláře na server dojde k ověření zadaných přihlašovacích údajů.

Tímto postupem je ovšem zajištěn pouze proces uživatelské autentizace. Na rozdíl od HTTP autentizace, kdy webový prohlížeč automaticky s každým dalším požadavkem posílá serveru přihlašovací údaje, je zapotřebí zajistit i proces tzv. entitní autentizace. Na začátku této kapitoly bylo uvedeno, že po úspěšně provedené uživatelské autentizaci obdrží uživatel nějaký token, pomocí kterého je aplikací na serveru identifikován při dalších požadavcích, a to až do vypršení platnosti přihlášení. Takovým tokenem bývá u tohoto typu autentizace provázání přihlášeného uživatele s aktuální *session*. Klient tak při dalších požadavcích zasílá na server tento session token.

Odhlášení uživatele ze systému je velmi jednoduché, stačí buď zrušit provázání uživatele s danou session, nebo celou session zrušit. Jednoduché je proto i zajištění automatického odhlášení uživatele po určité době nečinnosti.

Při používání formulářové autentizace, stejně jako u každé jiné, by se mělo dodržovat určitých bezpečnostních pravidel. Některé z nich jsou popsány v další sekci této kapitoly. Nyní zmíním jen dvě opravdu základní pravidla [2] spojená s tímto konkrétním typem autentizace. Prvním z nich je použití vstupního pole typu `password` pro zadávání hesla. Text, který je do něj psaný, prohlížeče nezobrazují a není nikdy předvyplňováno předchozími vstupy, jako je tomu u jiných typů vstupních polí. Dalším pravidlem je odesílání formuláře metodou POST. Důvod je takový, že se GET požadavek včetně všech odesílaných parametrů ukládá do historie prohlížeče, případně i na jiných místech během cesty od klienta na server.

Podpora v PHP

Pro samotné přihlášení uživatele není nutná žádná opravdu speciální podpora ze strany jazyka. Naopak tomu je u procesu entitní autentizace. Jazyk PHP standardně nabízí podporu sessions, a to zejména prostřednictvím funkcí `session_*`, kde `*` zastupuje identifikaci příslušné funkce. Pro přístup k proměnným aktuální session je k dispozici superglobální pole `$_SESSION`.

3.3 Bezpečnost při autentizaci

Proces autentizace uživatelů by měl být dostatečně zabezpečen, a to zejména proto, že dochází k manipulaci s přihlašovacími hesly. Ochrana hesel před jejich zneužitím je jedním z nejdůležitějších požadavků na každou aplikaci.

Tato sekce se zabývá používanými způsoby zabezpečení hesla při jeho uložení v databázi a při jeho přenosu mezi webovým prohlížečem a webovým serverem. V závěru je stručně popsán způsob ochrany před možnými útoky na session.

3.3.1 Ukládání hesel

Pokud by byla hesla v databázi uložena ve své čistě textové podobě, mohl by kdokoliv, komu se do databáze podaří proniknout, získat seznam hesel všech uživatelů aplikace. Z tohoto důvodu je zásadním a velmi důležitým pravidlem aplikovat na heslo vybranou hashovací funkci a až výsledek této funkce uložit do databáze [18]. Při ověřování, zda uživatel zadal správné heslo, se z databáze načte tento řetězec a porovnává se z výsledkem hashovací funkce aplikované na zadané heslo.

Hashovací funkce

Hashovací funkce vytvoří z předaného hesla tzv. otisk, neboli hash. Pro jeden řetězec vrací hashovací funkce pokaždé stejný otisk, z otisku ale není možné původní řetězec zpětně získat. Proto jsou takové funkce označovány jako jednocestné.

Jednocestných hashovacích funkcí existuje mnoho. Pro některé běžně používané však lze najít dva různé vstupní řetězce, které vedou na stejný otisk. Z množiny takových funkcí se to týká například populární funkce MD5 či SHA-1. Jelikož možnost nalezení kolizí nepředstavuje pro hashování hesel do databáze žádné riziko, není problém je pro tento účel použít. Z funkcí, u nichž zatím neexistuje žádný způsob pro nalezení kolizí, lze využít některou z rodiny algoritmů SHA-2, například SHA-256.

Solení hesel

Samotné hashovanání hesel však podle [18] rozhodně neřeší všechna bezpečnostní rizika. Dostane-li se útočník do databáze, v níž jsou uložena hashovaná hesla, může se pokusit získat jejich původní podobu systematickým generováním jednoho možného hesla za druhým. U každého vygenerovaného řetězce se spočítá jeho otisk a ten se porovnává s otiskem uloženým v databázi. Jedná se o tzv. útok hrubou silou.

Riziko úspěchu takového útoku lze, alespoň částečně, snížit tzv. *solením hesel*. Při hashování hesla se na vstup hashovací funkce přidává navíc nějaký další řetězec, označovaný jako sůl. Vzhledem k tomu, že sůl je pro každého uživatele jiná, mají i stejná hesla v databázi uložen jiný otisk. Jako sůl je možné zvolit náhodně vygenerovaný řetězec, pro který je ale potřeba v databázové tabulce rezervovat samostatný sloupec. Protože není nutné, aby sůl byla tajná, je daleko výhodnější používat jako sůl například uživatelské jméno.

Podpora v PHP

Pro algoritmy MD5 a SHA-1 jsou v PHP definovány samostatné funkce: `md5()` a `sha1()`. Od verze 5.1.2 je však standardně k dispozici funkce `hash()`, která nabízí univerzální přístup ke všem podporovaným hashovacím funkcím.

3.3.2 Přenos hesla přes síť

Přenos dat mezi webovým prohlížečem a webovým serverem je možné zabezpečit použitím protokolu HTTPS, kdy data jsou přenášena pomocí HTTP, ale jsou šifrována využitím protokolu SSL nebo TLS. Druhou odlišností je číslo portu, na němž probíhá komunikace. Zatímco HTTP používá port 80, u HTTPS je to implicitně 443. Využívá-li aplikace tohoto zabezpečení přenosu dat přes síť, není zapotřebí se zvlášť zaobírat zabezpečením hesla při autentizaci uživatele.

Často je ale zbytečné šifrovaný přenos dat používat, navíc ne každý webový server tuto možnost poskytuje. V těchto případech musíme bezpečný přenos hesla mezi klientem a serverem zajistit sami. Jednou z možností je používat Digest HTTP autentizaci. Při použití formulářové autentizace lze využít metodu challenge/response.

3.3.3 Metoda challenge/response

Označení *challenge/response*, neboli *výzva/odpověď*, používají různé metody z různých oblastí. Zde se jedná o techniku [2, 24], která zajišťuje bezpečný přenos hesla přes síť při používání formulářové autentizace bez šifrovaného přenosu dat.

Při této metodě se využívá skriptování na straně prohlížeče. Ze serveru je společně s přihlašovací stránkou prohlížeči zaslán náhodně vygenerovaný řetězec. Po zadání přihlašovacích údajů uživatelem dojde nejprve k jejich zpracování skriptem v prohlížeči. Ten na zadané heslo aplikuje hashovací funkci a výsledný otisk zřetězí s přijatým náhodně vygenerovaným řetězcem. Na celý takto vzniklý řetězec aplikuje opět hashovací funkci, jejíž výsledek odesílá na server místo zadaného hesla. Jelikož server zná správné heslo uživatele a pamatuje si odeslaný náhodný řetězec, může provést obdobný postup. Výsledek pak porovnává s řetězcem přijatým od klienta.

3.3.4 Ochrana před útoky na sessions

Podle [25] je se správou sessions spojeno několik různých útoků. Jedním z nich je tzv. *Session Hijacking*. Jedná se o útok, při němž útočník získá cizí Session ID (identifikátor session) a použije ho pro sebe. Úniku Session ID na serveru lze zabránit jeho dostatečným zabezpečením, úniku při přenosu dat mezi klientem a serverem je možné zamezit používáním protokolu HTTPS (viz výše).

Druhým typem útoku je tzv. *Session Fixation*. Při tomto útoku útočník uživateli podstrčí nějakou vlastní hodnotu Session ID a po přihlášení uživatele podstrčenou hodnotu použije pro sebe. Ochrana proti tomuto útoku je jednoduchá. Například v PHP stačí při každém požadavku klienta volat funkci `session_regenerate_id()`. Tato funkce změní Session ID, čímž útočník nemůže původní podstrčené ID využít.

Kapitola 4

Autorizace

Autorizace je podle [18, 26] proces, při kterém se ověřuje, zda má uživatel dostatečná oprávnění pro přístup k určitému souboru či pro provedení určité akce. Autorizace zpravidla následuje po úspěšné autentizaci uživatele. V [2] jsou popsány tři používané modely autorizace a řízení přístupu:

- volné řízení přístupu,
- řízení přístupu založené na úrovních citlivosti,
- řízení přístupu založené na rolích.

Volné řízení přístupu

O přístupu k určitému zdroji se rozhoduje na základě identity uživatele, případně jeho členství v uživatelské skupině. Charakteristickým prvkem je, že vlastník zdroje sám rozhoduje o přidělení přístupových práv ostatním uživatelům. Z tohoto důvodu nelze centrálně a jednotně řídit přístup ke všem dostupným zdrojům.

Tato technika řízení přístupu je běžně používána například v unixových souborových systémech.

Řízení přístupu založené na úrovních citlivosti

Každý uživatel je administrátorem zařazen na určitou úroveň důvěryhodnosti. Obdobně mají i zdroje přiděleny různé úrovně citlivosti. Při přístupu uživatele k některému zdroji se úroveň jeho důvěryhodnosti porovnává s úrovní citlivosti tohoto zdroje. O povolení přístupu rozhoduje výsledek porovnání.

Popsaný model se využívá spíše v aplikacích vysoce náročných na zabezpečení.

Řízení přístupu založené na rolích

Přístup k určitému zdroji je založen na rolích, které jsou uživateli v rámci systému přiděleny. O přidělení rolí a definici oprávnění pro jednotlivé role se stará administrátor systému. Ten by měl dodržovat zásadu co nejmenších oprávnění, aby každý uživatel aplikace měl přístup jen k těm částem systému, které opravdu potřebuje využívat.

Model řízení přístupu založený na rolích je ve webových aplikacích často používán.

4.1 ACL - seznam pro řízení přístupu

ACL (*Access Control List*) [27], neboli seznam pro řízení přístupu, je obecně seznam oprávnění pro přístup k nějakému objektu. ACL určuje, kdo nebo co má oprávnění přistupovat k určitému zdroji a jaké operace s ním může provádět. Každý záznam v ACL specifikuje vztah mezi uživatelem a zdrojem, případně operací s ním; určuje tedy úroveň povolení přístupu.

Tradiční seznamy ACL určují oprávnění uživatelům jednotlivě, což může být nevýhodné pro systémy s velkým počtem uživatelů. Druhým možným přístupem je model ACL založený na rolích, kdy jsou jednotlivá oprávnění přidělována rolím a uživatelům jsou přidělovány tyto role. Pro autorizaci uživatelů ve webových aplikacích se používá zejména právě tento druhý model ACL.

Kapitola 5

Návrh a implementace komponenty pro řízení přístupu

Hlavním úkolem této práce bylo vytvořit komponentu pro řízení přístupu ve webových aplikacích vyvíjených v jazyce PHP 5. Obsahem této kapitoly je popis návrhu, implementace a použití této komponenty.

Komponenta pro řízení přístupu se skládá ze tří částí: modulu pro autentizaci, modulu pro autorizaci a abstraktní třídy `AccessControl`, která umožňuje sjednotit použití nabízených prostředků pro autentitizaci i autorizaci.

5.1 Modul pro autentizaci

Jak jsem popsal v kapitole 3.2, ve webových aplikacích se využívají zejména dva principy autentizace uživatelů: formulářová autentizace a autentizace pomocí protokolu HTTP. Vytvořená komponenta oba tyto způsoby autentizace poskytuje prostřednictvím tříd modulu pro autentizaci.

Základní třídou je abstraktní třída `Authentication`, z níž jsou děděním vytvořeny třídy pro jednotlivé typy autentizace. Kompletní diagram tříd modulu pro autentizaci je na obrázku 5.1.

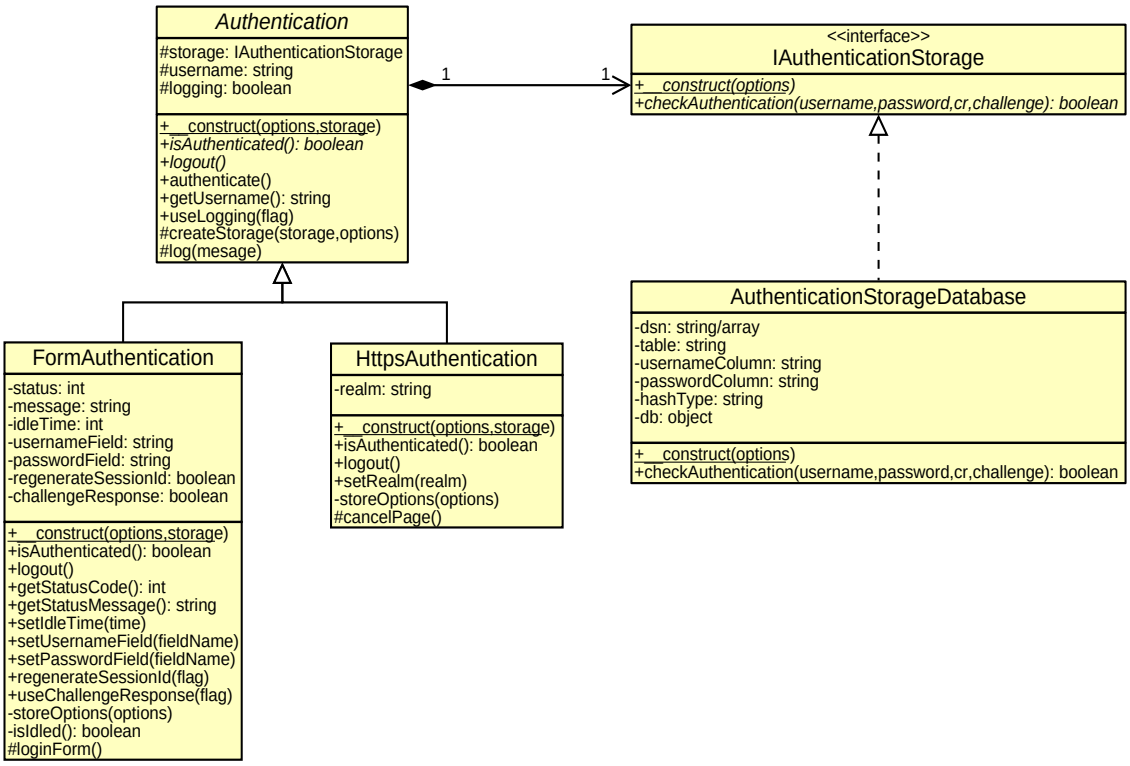
5.1.1 Třída `Authentication`

Abstraktní třída `Authentication` specifikuje povinné rozhraní pro své podtřídy, zároveň implementuje společné vlastnosti obou typů autentizace.

Třídy, oděděné od této třídy, musí povinně implementovat dvě veřejné metody: `isAuthenticated()` a `logout()`. První z nich je určena pro samotnou autentizaci uživatele, vrací boolovskou hodnotu podle výsledku operace. Druhá metoda slouží k odhlášení uživatele.

Pro provedení vlastního procesu autentizace uživatele je možné využít již implementovanou metodu `authenticate()`, která zapouzdřuje volání `isAuthenticated()`. Tato metoda ovšem nemá návratovou hodnotu, ale při neúspěšném ověření identity generuje výjimku `AuthenticationFailedException`.

K ověřování identity uživatele dochází, nezávisle na zvoleném typu autentizace, porovnáním zadaných přihlašovacích údajů se správnými údaji, které musí být někde uloženy. K tomuto účelu se obvykle využívá databáze, nicméně lze využít i jiný druh úložiště dat. Třída `Authentication` při vytváření přístupu k tomuto úložišti předpokládá existenci třídy, která implementuje rozhraní `IAuthenticationStorage`. Tímto přístupem je



Obrázek 5.1: Diagram tříd modulu pro autentizaci

dosaženo nezávislosti na typu úložiště dat a přístupu k němu. Detailněji je způsob přístupu k úložišti dat rozebrán v následující sekci této kapitoly.

Ačkoliv není možné přímo tuto třídu instanciovat, obsahuje vlastní konstruktor, který vytváří objekt třídy pro přístup k úložišti dat a zahajuje session. Tento konstruktor slouží pro volání z konstruktorů podtříd. Dojde-li při provádění jeho těla k chybě, generuje výjimku `AuthenticationErrorException`, kterou pro detekci chyb využívají i podtřídy této třídy.

Logování událostí a chyb

Třída `Authentication` nabízí pro své podtřídy možnost logování událostí vzniklých při procesu autentizace. Implementuje chráněnou metodu `_log()`, která po svém zavolání zapíše předanou zprávu včetně časového razítka do souboru.

Tento způsob logování však nemusí být příliš vhodný. Programátor, který tuto komponentu bude používat, může v některé z podtříd této třídy metodu `_log()` předefinovat a použít tak vlastní způsob.

Implicitně je však logování zakázáno. Povolit jej lze voláním metody `useLogging()`.

5.1.2 Přístup k úložišti dat

Jak jsem již zmínil, pro přístup ke správným přihlašovacím údajům uživatelů aplikace je předpokládána třída implementující rozhraní `IAuthenticationStorage`. Kromě povinného konstruktoru specifikuje metodu pro ověření identity uživatele na základě předaného uživa-

telského jména a hesla. Podrobnější informace k vytváření vlastních tříd, které implementují toto rozhraní, se nacházejí v programové dokumentaci na přiloženém CD.

Tato komponenta pro řízení přístupu obsahuje třídu `AuthenticationStorageDatabase`, která slouží pro ověřování správnosti přihlašovacích údajů proti údajům uloženým v databázi. Pro přístup k databázi jsou využívány prostředky balíku `DB` z repositáře PEAR. Tento balík obsahuje stejnojmennou třídu, která umožňuje jednotný objektově orientovaný přístup ke všem podporovaným databázím (MySQL, PostgreSQL, MS SQL a mnoho dalších). Pro připojení k databázi pomocí třídy `DB` je nutné vytvořit tzv. *DSN* (Data Source Name). Jedná se o řetězec definovaného formátu obsahující údaje potřebné pro přístup k databázi. Formát DSN je popsán v dokumentaci k balíku `DB` na webových stránkách PEAR [12].

Objekt třídy pro přístup k úložišti dat je vytvářen v konstruktoru třídy `Authentication`. Pokud konstruktoru není předána informace o tom, kterou třídu má instanciovat, vybírá implicitně právě třídu `AuthenticationStorageDatabase`.

5.1.3 Formulářová autentizace

Pro provádění autentizace uživatelů prostřednictvím formuláře ve webové stránce je určena třída `FormAuthentication`, která je vytvořena děděním z výše popsané abstratní třídy `Authentication`. Formulářová autentizace (viz kapitola 3.2.2) spočívá v odeslání webové stránky s přihlašovacím formulářem prohlížeči vždy, když přistupuje ke stránce, která je přístupná jen přihlášeným uživatelům a momentálně žádný uživatel přihlášen není. Na začátku každého skriptu, který generuje takovou stránku, je proto nutné provést proces autentizace.

Průběh autentizace

Prvním krokem je ověření, zda je již nějaký uživatel přihlášen – entitní autentizace. Toto ověření probíhá na základě existence session proměnné s uživatelským jménem přihlášeného uživatele. Pokud je některý z uživatelů přihlášen, kontroluje se, jestli je přihlášení platné. Tato kontrola probíhá porovnáním doby uplynulé od posledního požadavku uživatele s nastavenou maximální povolenou dobou nečinnosti (implicitně je nastavena 0, což indikuje platnost až do uzavření prohlížeče). Mohou nastat tyto dvě situace:

- Přihlášení je platné – autentizace proběhla úspěšně.
- Platnost přihlášení vypršela – dojde k automatickému odhlášení uživatele, autentizace je neúspěšná.

Pokud neexistuje příslušná session proměnná, žádný uživatel tedy přihlášen není, mohou opět nastat dvě možné situace:

- K dispozici jsou přihlašovací údaje získané odesláním formuláře – dochází k ověření jejich správnosti proti údajům získaným z použitého úložiště dat.
  - Zadané přihlašovací údaje jsou správné – autentizace proběhla úspěšně.
  - Zadané přihlašovací údaje nejsou správné – autentizace je neúspěšná.
- Přihlašovací údaje nejsou k dispozici, jedná se tedy o první přístup ke stránce – autentizace je neúspěšná.

Jelikož k neúspěšné autentizaci může dojít z několika různých důvodů, nabízí třída `FormAuthentication` možnost získání číselného kódu a zprávy s informací o výsledku autentizace. K tomuto účelu slouží metody `getStatusCode()` a `getStatusMessage()`.

Vždy, když autentizace neproběhla úspěšně, je automaticky volána metoda `loginForm()`, která je určena pro generování HTML stránky s přihlašovacím formulářem. Vzhledem k tomu, že je tělo této metody prázdné, je pro její využití zapotřebí děděním od třídy `FormAuthentication` vytvořit novou třídu a v ní podle potřeby metodu `loginForm()` předefinovat. Alternativně lze generování přihlašovacího formuláře umístit mimo logiku třídy.

Za předpokladu, že `$a` reprezentuje existující objekt třídy `FormAuthentication`, je možné vyvolat proces autentizace uživatele dvěma způsoby, které ovšem provádějí stejnou činnost. První možností je využít metodu vracející výsledek operace:

```
if (!$a->isAuthenticated()) {
    // neúspěšná autentizace
} else {
    // úspěšná autentizace
}
```

Druhá možnost spočívá v použití metody generující výjimku při neúspěšném výsledku autentizace:

```
try {
    $a->authenticate();
} catch (AuthenticationErrorException $e) {
    // neúspěšná autentizace
}
```

Podrobnější popis způsobů a možností použití včetně ukázkových příkladů se nachází v programové dokumentaci na přiloženém CD.

Použití metody challenge/response

Metoda challenge/response (viz kapitola 3.3.3) slouží k zabezpečení přenosu hesla mezi webovým prohlížečem a serverem při použití formulářové autentizace bez šifrovaného přenosu dat.

Třída `FormAuthentication` použití této metody podporuje, je ovšem nutné zajistit následující požadavky:

- Musí být k dispozici Javascriptová implementace použité hashovací funkce.
- Použitá třída pro přístup dat musí tuto metodu podporovat.
- Přihlašovací formulář musí navíc obsahovat dvě skrytá textová pole s názvy challenge a response pro přenos výzvy a odpovědi.
- Musí být povoleno použití této metody přes parametr konstruktoru třídy nebo voláním `useChallengeResponse()`.

5.1.4 HTTPS autentizace

Komponenta pro řízení přístupu umožňuje kromě formulářové autentizace provádět i autentizaci pomocí protokolu HTTP (viz kapitola 3.2.1), kontrétně Basic HTTP autentizaci s přenosem dat přes šifrovaný kanál. Slouží k tomu třída `HttpsAuthentication`, která je oděděna od abstraktní třídy `Authentication`.

Podobně jako u formulářové autentizace je zapotřebí vyvolat proces autentizace na začátku každého skriptu, který generuje stránku přístupnou jen autentizovaným uživatelům.

Průběh autentizace

Po spuštění procesu autentizace je nejprve ověřeno, zda komunikace probíhá pomocí protokolu HTTPS, tedy přes šifrovaný kanál. Není-li tomu tak, dojde k pokusu o přesměrování na stejnou stránku, ale s využitím protokolu HTTPS.

Jako token pro entitní autentizaci je využívána, stejně jako u formulářové autentizace, session proměnná, ačkoliv při HTTP autentizaci není zapotřebí, jelikož prohlížeč po úspěšném přihlášení zasílá s každým dalším požadavkem jméno a heslo uživatele. Tento přístup jsem zvolil zejména proto, aby mohl server rozlišit, zda dochází k autentizaci uživatelské nebo entitní. V opačném případě by totiž při každém požadavku server zbytečně musel ověřovat zasílané přihlašovací údaje proti údajům v databázi, což běh aplikace může zpomalovat.

Dalším krokem je tedy na základě existence session proměnné s uživatelským jménem přihlášeného uživatele ověřeno, zda některý uživatel přihlášen je či není. Pokud ano, autentizace končí úspěchem. V opačném případě dochází k ověření dostupnosti přihlašovacích údajů:

- Přihlašovací údaje jsou k dispozici – dochází k ověření jejich správnosti proti údajům získaným z použitého úložiště dat.
  - Zadané přihlašovací údaje jsou správné – autentizace proběhla úspěšně.
  - Zadané přihlašovací údaje nejsou správné – autentizace je neúspěšná.
- Přihlašovací údaje k dispozici nejsou, jedná se o první přístup ke stránce – autentizace je neúspěšná.

Na rozdíl od formulářové autentizace je výsledek procesu autentizace vracen jen při úspěchu. Při neúspěšné autentizaci totiž dochází vždy k odeslání příslušné HTTP hlavičky webovému prohlížeči a následnému ukončení skriptu na straně serveru. Proto k vyvolání procesu autentizace při použití této třídy stačí zavolat jednu z metod `authenticate()` a `isAuthenticated()` a předpokládat, že kód, který se nachází za voláním některé této metody bude proveden jen pro uživatele, jenž úspěšně prokázal svoji identitu.

Společně s HTTP hlavičkou, která prohlížeči oznamuje požadavek na autentizaci uživatele, je možné odeslat HTML stránku, jejíž obsah prohlížeč zobrazí, pokud uživatel odmítne přihlášení a zavře zobrazený přihlašovací dialog. K tomutu účelu slouží metoda `cancelPage()`, která je volaná automaticky při odesílání požadavku na autentizaci webovému prohlížeči. Tělo této metody generuje pouze krátkou textovou informaci o požadavku na autentizovaný přístup. Pokud by chtěl programátor generovat vlastní obsah této stránky, může vytvořit novou třídu oděděním z třídy `HttpsAuthentication` a v ní metodu `cancelPage()` předefinovat.

Podrobnější popis použití této třídy včetně ukázkového příkladu se nachází v programové dokumentaci na přiloženém CD.

Odhlášení uživatele

HTTP autentizace bohužel neumožňuje žádným zaručeným způsobem donutit prohlížeč, aby přestal posílat zadané přihlašovací údaje. Pro odhlášení uživatele jsem zvolil způsob, kdy je prohlížeči podstrčena HTTP hlavička oznamující požadavek na autentizovaný přístup. Bohužel však existují prohlížeče, které ani po přijetí této hlavičky přihlašovací údaje nezapomenou.

5.2 Modul pro autorizaci

Proces autorizace (viz kapitola 4) zpravidla následuje po úspěšné autentizaci některého uživatele. Při autorizaci se určuje, zda má přihlášený uživatel povolen přístup k požadované stránce, případně k některé její části. V rámci této komponenty pro řízení přístupu je pro provádění autorizace uživatelů k dispozici třída `Acl`, která implementuje autorizaci pomocí seznamu pro řízení přístupu (ACL) založeného na rolích uživatelů.

5.2.1 Třída `Acl`

Třída `Acl` používá způsob řízení přístupu uživatelů založený na rolích. Mimo logiku třídy má každý uživatel aplikace přidělenu jednu či více rolí. ACL pak obsahuje pravidla definovaná pro jednotlivé role. Stejně tak samotná autorizace probíhá formou dotazu, zda je pro danou roli přístup povolen. ACL, implementovaný třídou `Acl`, obsahuje tři seznamy:

- seznam všech definovaných rolí,
- seznam zdrojů (*resources*), k nimž je možné v rámci aplikaci přistupovat,
- seznam pravidel definujících vztahy mezi rolemi a zdroji.

Kromě vlastní třídy `Acl` obsahuje modul pro autorizaci další dvě třídy: `AclRoles` a `AclResources`. Tyto třídy zapouzdřují přístup k seznamům rolí a zdrojů. Diagram tříd modulu pro autorizaci je na obrázku 5.2. Všechny metody, které poskytuje třída `Acl`, generují při vzniku chyby výjimku `AclErrorException`.

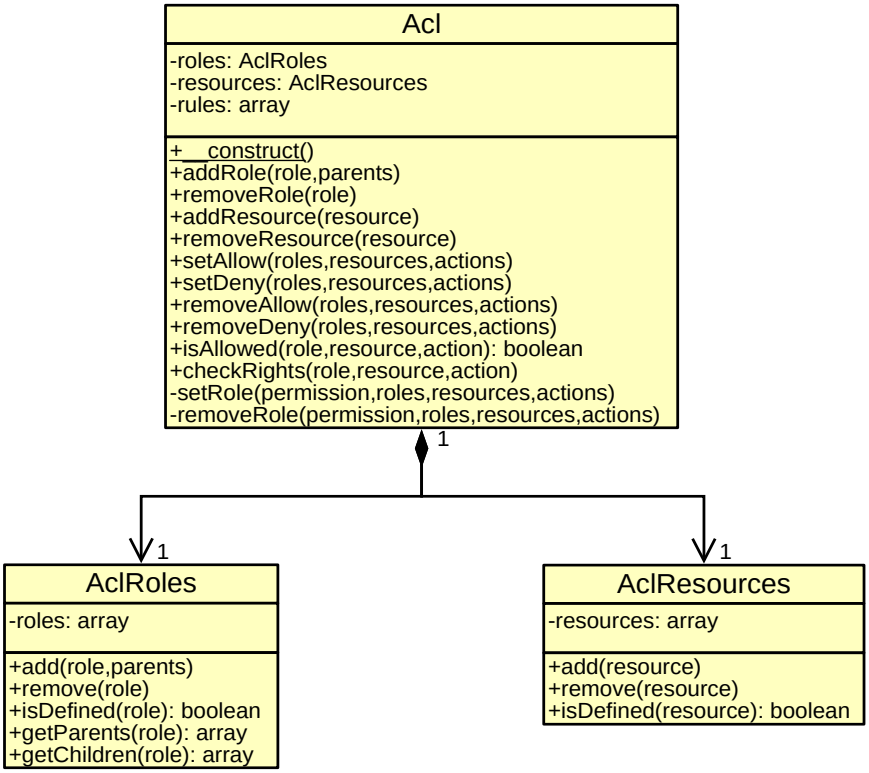
Dědičnost rolí

Často se při definování rolí uživatelů v aplikaci hodí vytvářet hierarchické vztahy mezi rolemi. Přístupová práva definovaná pro výše postavenou roli potom platí i pro její podrole.

Třída `Acl` umožňuje takovou dědičnost rolí používat, a to dokonce dědičnost vícenásobnou. Pro vkládání rolí do ACL slouží metoda `addRole()`, jejímž prvním parametrem je název vkládané role. Přes druhý, nepovinný, parametr je možné předávat pole názvů všech rodičovských rolí. Pro následující příklad platí předpoklad, že `$acl` je existující objekt třídy `Acl`.

```
$acl->addRole('a');
$acl->addRole('b');
$acl->addRole('c', 'a');                // příklad jednoduché dědičnosti
$acl->addRole('d', array('a', 'b'));    // příklad vícenásobné dědičnosti
```

Při vícenásobné dědičnosti rolí však může docházet ke kolizím mezi pravidly. V takovém případě se jako platné bere pravidlo definované pro dříve uvedenou rodičovskou roli.



Obrázek 5.2: Diagram tříd modulu pro autorizaci

Definice pravidel

Každé pravidlo v ACL definuje určité vztahy mezi rolemi a zdroji v systému. Třída `Acl` však navíc umožňuje jemnější dělení každého zdroje na akce. Plný formát pravidla pak tedy definuje vztah mezi konkrétní rolí, zdrojem a akcí.

Pro definici pravidel jsou určeny metody `setAllow()` a `setDeny()`. Způsob jejich použití je naprosto totožný, jediným rozdílem je, že první z nich definuje povolující pravidlo, druhá zakazující. Obě metody přebírají tři parametry v tomto pořadí: role, zdroj, akce. Pravidlo může být definováno zároveň pro více rolí, zdrojů či akcí; potom stačí místo jednoho názvu předat pole názvů. Stejně tak může být kterýkoliv parametr nahrazen hodnotou `null`, případně může chybět. Potom se platnost pravidla přenáší na všechny role, zdroje či akce v daném kontextu. Následuje příklad definic pravidla:

```
// Host má povoleno číst zprávy.
$acl->setAllow('host', 'zprávy', 'číst');

// Uživatel má povoleny všechny akce pro zdroj zprávy.
$acl->setAllow('uživatel', 'zprávy');

// Správce má povoleno vše.
$acl->setAllow('správce');

// Host má zakázáno mazat a editovat uživatele.
$acl->setDeny('host', 'uživatelé', array('mazat', 'editovat'));
```

Inicializace ACL

Před samotným použitím ACL pro autorizaci uživatelů je potřeba objekt třídy `Acl` inicializovat, čili naplnit jednotlivé seznamy.

Způsob naplnění seznamu rolí a pravidel jsem již uvedl výše, zbývá ještě naplnění seznamu zdrojů. K tomu je určena metoda `addResource()`, která přebírá jediný parametr obsahující název zdroje.

Takováto inicializace ACL by byla nutná při každém požadavku klienta na nějakou stránku aplikace. Tento postup by však mohl vést na zbytečné prodloužení reakční doby serveru při každém požadavku uživatele. Proto je zřejmě daleko výhodnější příslušný objekt třídy `Acl` vytvářet a plnit, jen když je to nezbytně nutné. Například provést jeho vytvoření a naplnění po přihlášení uživatele do aplikace, uložit tento objekt do nějaké cache a v případě potřeby ho z ní načítat.

Průběh autorizace

Samotný proces autorizace je možné provést voláním jedné ze dvou metod, které se liší jen způsobem získání výsledku autorizace. První z těchto metod je `isAllowed()`, která vrací boolovskou hodnotu podle výsledku operace. Její použití ukazuje následující příklad:

```
// Má uživatel host povoleno číst zprávy?
if (!$acl->isAllowed('host', 'zprávy', 'číst')) {
    echo "Nemáte právo přístupu k této stránce.";
    exit;
}
```

Druhou možností je využít metodu `checkRights()`, která nemá návratovou hodnotu, ale zapouzdřuje volání `isAllowed()` a při detekci nedostatečného oprávnění generuje výjimku `AclDeniedException`. Následuje příklad použití:

```
try {
    // Má správce povoleno mazat zprávy?
    $acl->checkRights('správce', 'zprávy', 'mazat');

} catch (AclDeniedException $e) {
    echo "Nemáte právo přístupu k této stránce.";
    exit;
}
```

Po zavolání některé z těchto metod pro ověření povolení přístupu dochází k průchodu seznamem ACL. Nejprve se ověřují pravidla definovaná přímo pro danou roli. Není-li nalezeno vhodné pravidlo, dochází na ověření pravidel zděděných od rodičovských rolí. Pokud stále žádné vyhovující pravidlo nalezeno nebylo, ověřují se pravidla nastavená pro všechny role, případně všechny zdroje. Při použití třídy `Acl` platí, že co není povoleno je zakázáno. To znamená, že pokud není nalezeno žádné odpovídající pravidlo, je přístup zamítnut.

Bližší informace k použití prostředků pro autorizaci, které tato komponenta nabízí, včetně ukázkových příkladů se nachází v programové dokumentaci na přiloženém CD.

5.3 Třída `AccessControl`

Abstraktní třída `AccessControl` slouží jako základní třída pro vytvoření třídy, která sjednocuje prostředky pro autentizaci a autorizaci, které nabízí tato komponenta pro řízení přístupu.

Třída poskytuje metody pro provádění samotné autentizace či samotné autorizace uživatele, poskytuje však také metodu `checkAccess()`, která provede autentizaci následovanou autorizací uživatele.

Použití této třídy je založeno na tom, že si drží seznam rolí, které přísluší přihlášenému uživateli. Po úspěšně provedené autentizaci je interně volána metoda `_loadRoles()`, která má za úkol načíst seznam rolí, jenž náleží právě přihlášenému uživateli. Při autorizaci uživatele jsou pak ověřována přístupová práva pro každou ze seznamu rolí, tedy pokud jich je více.

Abstraktní metoda `_loadRoles()` je určena pro implementaci vývojářem, který bude třídu `AccessControl` využívat. Kromě toho samozřejmě musí implementovat minimálně ještě konstruktor, v němž bude provedeno vytvoření a inicializace objektů tříd pro autentizaci a autorizaci.

Podrobnější popis metod této třídy se nachází v programové dokumentaci na přiloženém CD. Kompletní diagram tříd komponenty pro řízení přístupu včetně třídy `AccessControl` je na obrázku 5.3.

Obrázek 5.3: Diagram tříd komponenty pro řízení přístupu

Kapitola 6

Ukázková aplikace

Tato kapitola se zabývá návrhem a implementací webové aplikace, na níž je otestována a demonstrována funkčnost komponenty pro řízení přístupu, která je popsána v kapitole 5.

Jedná se o aplikaci sloužící jako „Knihovna odborných článků“. Hlavním účelem této aplikace je správa databáze článků s možností jejich půjčování. Implementačním jazykem je jazyk PHP 5, pro uložení databáze je používán databázový systém MySQL.

6.1 Specifikace požadavků

Požadavky na tuto ukázkovou aplikaci jsou následující:

- Vkládání nových článků přes uživatelské rozhraní aplikace a importem ze souboru ve formátu XML či CSV.
- Údaje o každém článku jsou následující: autoři, název článku, rok vydání, klíčová slova, poznámka, počet kusů, umístění a forma (elektronicky/papírově).
- Vyhledávání článků podle autorů, názvu a klíčových slov.
- Možnost editace, mazání, půjčování, vracení a rezervace článků. Při rezervaci vzniká fronta na vypůjčení, účastníci ve frontě jsou prostřednictvím e-mailu upozorněni na jejich pořadí.
- Půjčování a vracení článků řeší samotný uživatel.
- Logování změn článků s možností prohlížení historie těchto změn.
- Prohlížení aktuálních výpůjček, rezervovaných článků a historie výpůjček.
- Správa uživatelů – přidávání, editace, mazání.
- Údaje o každém uživateli jsou následující: jméno, příjmení, e-mail, telefon, místnost, uživatelské jméno a heslo.
- Tři různé role uživatelů – *správce* (správa uživatelů), *knihovník* (správa článků) a *uživatel* (prohlížení, půjčování a vracení článků). Každý uživatel může být v libovolných rolích.
- Každý uživatel má možnost editovat vlastní profil.

- Bezpečné přihlašování uživatelů, důraz kladen na bezpečné uložení a přenos hesel.
- Pohodlné a intuitivní uživatelské rozhraní s rychlým vkládáním, půjčováním a vracením článků.

6.2 Návrh a implementace datového modelu

Na obrázku 6.1 se nachází ER diagram, který specifikuje jednotlivé entity aplikace a statické vztahy mezi nimi.

Hlavními entitami jsou „Článek“ a „Uživatel“. Mezi nimi existují vztahy, které realizují možnost půjčování a rezervaci článku uživatelem. Entita „Změna“ společně s jejími vztahy k článku a uživateli umožňuje logování změn, které na určitém článku provedl určitý uživatel. Všechny možné typy těchto změn reprezentuje entita „TypZměny“. Forma uložení článku a jeho umístění je reprezentováno pomocí číselníků.

Vztah typu N:N mezi entitami „Uživatel“ a „Role“ reprezentuje fakt, že každý uživatel může být v rámci aplikace ve více rolích. Unární vztah na entitě „Role“ umožňuje vytvářet dědičnou hierarchii mezi jednotlivými rolemi.

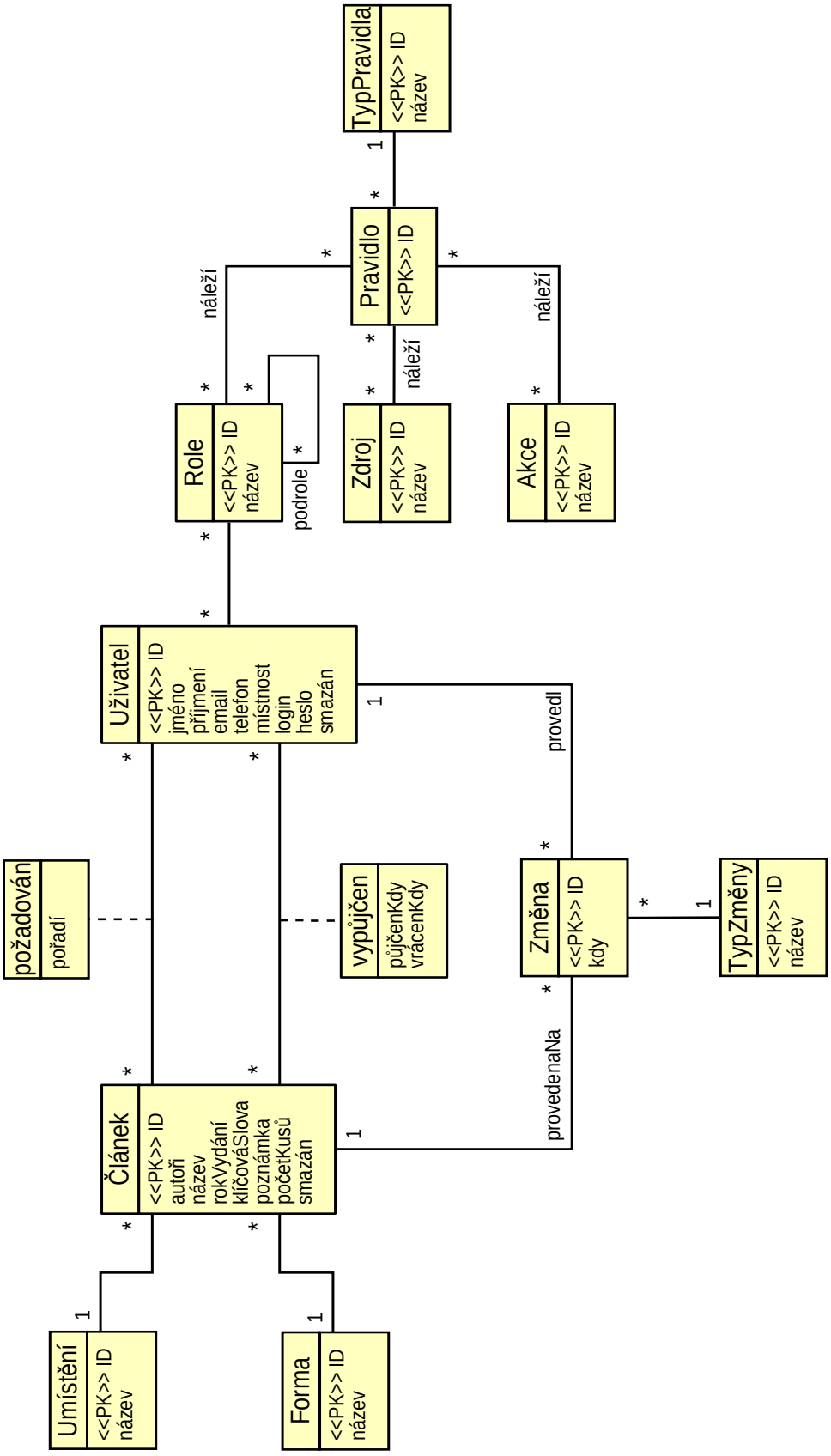
Entity, umístěné v tomto ER diagramu napravo od entity „Uživatel“, slouží k definici přístupových práv jednotlivých rolí ke zdrojům, které jsou v aplikaci dostupné. Tato přístupová práva jsou určena k provádění autorizace auživatelů. Entita „Pravidlo“ reprezentuje jednotlivá pravidla definující přístupová práva. Každé pravidlo je ve vztahu s obecně libovolným počtem rolí, zdrojů a akcí. Úroveň povolení, tedy zda pravidlo přístup povoluje či zakazuje, reprezentuje vztah mezi entitami „Pravidlo“ a „TypPravidla“.

Implementace ER diagramu

Implementace ER diagramu zpravidla spočívá v jeho převodu na tabulky relační databáze, pro tuto aplikaci kontrétně databáze MySQL.

Každé entitě v ER diagramu odpovídá jedna databázová tabulka, jejíž sloupce reprezentují atributy entity. Vztahy typu 1:N jsou implementovány pomocí cizích klíčů. Pouze vztah mezi entitami „Pravidlo“ a „TypPravidla“ jsem implementoval tak, že název typu pravidla je umístěn přímo v tabulce s pravidly, čímž odpadá nutnost samostatné tabulky pro typy pravidla. K tomuto kroku jsem se rozhodl proto, že výčet typů pravidla bude po celou dobu života aplikace neměnný, konkrétně se jedná o dva typy - povolení a zákaz.

Každý vztah typu N:N je v databázi reprezentován spojovací tabulkou, která obsahuje cizí klíče do tabulek pro entity, mezi nimiž je daný vztah definován.



Obrázek 6.1: ER diagram knihovny odborných článků

6.3 Architektura aplikace

Jak jsem popsal v kapitole 2.1, důležitou součástí návrhu webové aplikace je volba vhodné architektury. Zaměřil jsem se zejména na popis architektury MVC (Model-View-Controller), kterou jsem se rozhodl použít i pro tuto aplikaci. Vztahy mezi součástmi této architektury jsem znázornil na obrázku 2.1 na straně 5.

Vstupním bodem aplikace je správce požadavků (*Front Controller*), který přijímá a zpracovává všechny uživatelské akce. Uživatelskou akcí je každý požadavek uživatele na tuto aplikaci prostřednictvím webového prohlížeče. Na základě tohoto požadavku rozhodne, který konkrétní kontrolér zavolat a kterou akci má provést. Po přenesení řízení na konkrétní akci kontroléru dochází k provedení příslušných změn modelu. Následně kontrolér vybírá pohled, který bude odeslán prohlížeči jako odpověď na požadavek uživatele. Vybraný pohled generuje HTML kód odesílané stránky a v případě potřeby se dotazuje modelu na jeho současný stav.

6.3.1 Model

Model v této aplikaci slouží zejména k následujícím účelům:

- zapouzdřuje přístup k datové vrtsvě aplikace,
- poskytuje prostředky pro řízení přístupu uživatelů.

Třídy pro přístup k datům

Vzhledem k povaze aplikace a požadavkům na ní jsem navrhnul tři navzájem nezávislé třídy pro práci s datovou částí aplikace. Jedná se o třídy `User`, `Article` a `Loans`. Jejich diagramy se nacházejí v příloze B. Tyto třídy poskytují metody zejména pro interakci s databází aplikace. Načítají požadovaná data, případně provádějí jejich modifikaci, odstraňování či přidávání. Třídy `User` a `Article` je možné instanciovat, nabízejí však i statické metody. Objekty těchto tříd reprezentují příslušného uživatele, případně článek. Třída `Loans` poskytuje jen statické metody, není tedy určena pro vytváření objektů.

Pro přístup k databázi všechny tyto třídy využívají prostředky poskytované balíkem `DB` z repositáře PEAR.

Řízení přístupu uživatelů

Pro řízení přístupu uživatelů, tedy pro provádění autentizace a autorizace, jsem využil vlastní komponentu pro řízení přístupu (viz kapitola 5).

K autentizace uživatelů používám prostředky pro formulářovou autentizaci s využitím metody challenge/response pro zabezpečený přenos hesla mezi webovým prohlížečem a serverem. Pro formulářovou autentizaci je určena třídy `FormAuthentication`. Z této třídy jsem děděním vytvořil třídu `UserAuthentication`, v níž jsem předefinoval metodu `loginForm()` pro generování přihlašovací stránky. Protože jsou přihlašovací údaje uloženy v databázi, není nutné implementovat žádnou vlastní třídu pro přístup k ní; dostatečně vyhovující je třída poskytovaná komponentou. Kromě zabezpečení metodou challenge/response je nastavena ochrana proti útoku Session Fixation. Hesla jsou v databází uložena v hashované podobě. Pro hashování se používá funkce MD5.

Autorizaci uživatelů zajišťuje třída `Acl`. Seznamy rolí a zdrojů a pravidla definující přístupová práva jsou uložena v databázi. Pro inicializaci ACL je proto nutné všechny tyto

údaje z databáze načíst. K jejich načtení a vložení do ACL však dochází pouze při prvním požadavku uživatele. Vytvořený objekt je pak uložen jako session proměnná, což urychluje reakci serveru na všechny následující požadavky.

Pro řízení přístupu jsem s výhodou využil i abstraktní třídu `AccessControl`. Z ní jsem oddědil novou třídu s názvem `UserAccessControl`. Tato třída umožňuje sjednotit provádění autentizace a autorizace uživatelů. V konstruktoru třídy dochází k vytvoření a inicializaci objektů tříd `Acl` a `UserAuthentication`. Bylo také nutné implementovat metodu pro načtení seznamu rolí, v nichž je přihlášený uživatel. K načtení rolí z databáze však dochází jen po přihlášení uživatele. Seznam rolí je pak až do odhlášení uživatele uložen jako session proměnná.

6.3.2 Pohled

Pro výběr pohledu, neboli View, slouží stejnojmenná třída, která obsahuje řadu metod. Každá metoda třídy `View` odpovídá jednomu konkrétnímu pohledu. Kontrolér při volání těchto metod předává jako parametr odkaz na existující objekt třídy pro řízení přístupu, tedy `UserAccessControl`. V těle každé takové metody dochází k vložení obsahu souboru, který odpovídající pohled, tedy HTML stránku, generuje.

6.3.3 Kontrolér

Jak jsem zmínil výše, vstupním bodem celé aplikace je specializovaný kontrolér pro správu požadavků. Jedná se o skript umístěný v souboru `index.php` v kořenovém adresáři aplikace. Přesměrování všech uživatelských požadavků na tento soubor zajišťuje použití přepisovacích pravidel modulu `mod_rewrite` serveru Apache. Použitá přepisovací pravidla se nacházejí v souboru `.htaccess` v kořenovém adresáři aplikace. Server na základě těchto pravidel přesměruje každou požadovanou URL na soubor `index.php` s nastavenými parametry podle konkrétního požadavku. Správce požadavků provede zpracování a vyhodnocení předaných parametrů. Pokud jsou korektní, vyvolá odpovídající akci vybraného kontroléru. V opačném případě vybírá pohled s chybovou stránkou.

Tato aplikace obsahuje kromě vstupního kontroléru pro správu požadavků celkem čtyři další kontroléry reprezentované těmito třídami: `DefaultController`, `UsersController`, `ArticlesController` a `LoansController`. Každá z těchto tříd obsahuje, kromě konstruktoru, metody, které reprezentují dostupné akce. V těle každé takové metody pak dochází k volání služeb modelu a následnému výběru pohledu. Zároveň se jedná o místo, kde dochází k autentizaci, případně i autorizaci uživatele.

6.4 Požadavky na provoz aplikace

Vytvořená ukázková aplikace je určena pro provoz na webovém serveru Apache, na němž musí jako modul běžet interpet jazyka PHP 5 s rozšířením PEAR. Server musí podporovat použití `.htaccess` souborů a mít nainstalován modul `mod_rewrite`. Pro uložení databáze by měl být použit databázový server MySQL.

Při dodržení těchto základních požadavků by měla aplikace bez problémů fungovat. Všechny zdrojové texty a informace k instalaci aplikace na server se nachází na přiloženém CD.

Kapitola 7

Závěr

Jedním z cílů této práce bylo seznámit se problematikou řízení přístupu ve webových aplikacích, zejména pak s využívanými principy a metodami autentizace a autorizace. Teoretický základ této problematiky jsem rozebral v první části práce. Nejprve jsem se stručně zabýval webovými aplikacemi, zaměřil jsem se zejména na používané architektury a platformy pro jejich vývoj. Další kapitoly jsem již věnoval tématu řízení přístupu, tedy popisu procesů autentizace a autorizace.

Autentizace, neboli ověřování identity uživatele, zaujímá poměrně rozsáhlou oblast problematiky řízení přístupu. Rozebral jsem nejprve obecně platné principy a způsoby autentizace, postupně jsem se ale přesunul ke konkrétním metodám využívaným ve webových aplikacích, kde identifikace uživatelů probíhá zejména na základě uživatelského jména a hesla. Podrobně jsem popsal dva používané typy autentizace, a to formulářovou autentizaci a autentizaci pomocí protokolu HTTP. Věnoval jsem se i způsobům zabezpečení procesu autentizace proti možným útokům.

Kromě autentizace je nedílným procesem při řízení přístupu proces autorizace, při kterém rozhodujeme, zda má již autentizovaný uživatel povolen přístup k určitému zdroji. Rozebral jsem několik používaných způsobů řízení přistupu a autorizace. Popsal jsem zde také techniku používanou pro autorizaci uživatelů, a to ACL.

Hlavním cílem této práce však bylo navrhnout a v jazyce PHP 5 implementovat komponentu pro řízení přístupu uživatelů ve webových aplikacích. Tato komponenta se skládá ze tří hlavních součástí. První součástí je modul pro autentizaci, dále modul pro autorizaci a třetí částí je abstraktní třída `AccessControl`, která umožňuje sjednotit prostředky nabízené oběma moduly. Komponenta poskytuje možnost provádění formulářové autentizace, HTTP autentizace s šifrovaným přenosem dat a autorizace pomocí ACL.

Při návrhu komponenty jsem se zaměřil zejména na jednoduché a pohodlné použití. Přístup je ke všem dostupným prostředkům objektově orientovaný. Pro detekci vzniklých chyb je využíván systém generování výjimek. Pro provádění autentizačních i autorizačních operací jsou vždy k dispozici dva typy metod. První je založena na návratové hodnotě výsledku operace, druhá pak na odchytávání výjimky generované při neúspěchu operace. Vynasnažil jsem se poskytnout prostředky pro zabezpečení při řízení přístupu, zejména pak při procesu autentizace, při němž dochází k manipulaci s hesly uživatelů. Komponenta je dobře použitelná v aplikacích s vícevrstvou architekturou. Teoreticky je možné ji integrovat do některého existujícího aplikačního frameworku. V praxi se toho však zřejmě nevyužije, protože většina frameworků poskytuje vlastní prostředky pro řízení přístupu, většinou provázané s ostatními komponentami.

Oproti některým existujícím systémům či knihovnám pro řízení přístupu lze v mém

řešení nalézt i některé nevýhody. Nedostatečná je podpora logování chyb a událostí. V současné podobě komponenty je možné logovat jen do souboru se staticky nastaveným umístěním a bez možnosti nastavení formátu zapisované zprávy. Oproti třídě `Auth` z repositáře PEAR či komponentě Zend_Auth poskytované Zend Frameworkem, nenabízí mé řešení implementaci tříd pro přístup k úložišti dat jinému, než je databáze. Při autentizaci neposkytuji zabezpečení kontrolou změny IP adresy a prohlížeče klienta, zaměřil jsem se spíše na bezpečný přenos hesla. Způsob autorizace pomocí ACL implementovaný v mém řešení je pro většinu aplikací vyhovující. Nemusel by však být dostačující, pokud by bylo potřeba řídit přístup navíc pomocí nějaké specifické podmínky.

Funkčnost vytvořené komponenty jsem otestoval a demonstroval na ukázkové webové aplikaci. Tato aplikace slouží jako knihovna odborných článků. Je postavena na architektuře MVC.

Literatura

[1] Wikipedia. *Web application — Wikipedia, The Free Encyclopedia*. URL: http://en.wikipedia.org/w/index.php?title=Web_application&oldid=208998519, 2008. [Online; cit. 3.5.2008].

[2] Jan Tichý. *Programová podpora tvorby webových aplikací*. URL: http://www.jantichy.cz/diplomka, 2004. [Online; cit. 3.5.2008].

[3] David Toth. *Architektury webových aplikací*. URL: http://webing.felk.cvut.cz/hs/download/DT-webarch-CZ-art.pdf. [Online; cit. 3.5.2008].

[4] Sun Microsystems. *Web-Tier Application Framework Design*. URL: http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html, 2002. [Online; cit. 3.5.2008].

[5] Cristobal Baray. *The Model-View-Controller Design Pattern*. URL: http://cristobal.baray.com/indiana/projects/mvc.html, 1999. [Online; cit. 3.5.2008].

[6] Wikipedia. *Model-view-controller — Wikipedia, The Free Encyclopedia*. URL: http://en.wikipedia.org/w/index.php?title=Model-view-controller&oldid=209528411, 2008. [Online; cit. 3.5.2008].

[7] Jan Tichý. *Model není pouze databáze*. URL: http://www.phpguru.cz/clanky/model-neni-pouze-databaze, 8.12.2007. [Online; cit. 3.5.2008].

[8] Wikipedia. *Web application framework — Wikipedia, The Free Encyclopedia*. URL: http://en.wikipedia.org/w/index.php?title=Web_application_framework&oldid=209426363, 2008. [Online; cit. 3.5.2008].

[9] WWW stránky. Php: Hypertext preprocessor. URL: http://www.php.net/. [cit. 3.5.2008].

[10] WWW stránky. The apache http server project. URL: http://httpd.apache.org/. [cit. 3.5.2008].

[11] WWW stránky. Mysql :: The world's most popular open source database. URL: http://www.mysql.com/. [cit. 3.5.2008].

[12] WWW stránky. Pear :: The php extension and application repository. URL: http://pear.php.net/. [cit. 3.5.2008].

[13] WWW stránky. Zend framework. URL: http://framework.zend.com/. [cit. 3.5.2008].

[14] WWW stránky. Java ee at a glance. URL: http://java.sun.com/javaee. [cit. 3.5.2008].

[15] WWW stránky. Microsoft .net framework. URL: http://www.microsoft.com/net. [cit. 3.5.2008].

[16] WWW stránky. The official microsoft asp.net site. URL: http://www.asp.net/. [cit. 3.5.2008].

[17] Richard E. Smith. *Authentication: From Passwords to Public Keys*. Addison-Wesley, Boston, first edition, 2002. 549 s. ISBN 0-201-61599-1.

[18] Jan Tichý. *Autentizace*. URL: http://www.phpguru.cz/rubriky/autentizace, 16.9.2007. [Online; cit. 4.5.2008].

[19] Ondřej Bitto. *Průvodce zajímavým světem autentizace, 1. část*. URL: http://www.zive.cz/default.aspx?article=122526, 10.2.2005. [Online; cit. 4.5.2008].

[20] Ondřej Bitto. *Průvodce zajímavým světem autentizace, 2. část*. URL: http://www.zive.cz/default.aspx?section=21&server=1&article=122527, 11.2.2005. [Online; cit. 4.5.2008].

[21] J. Franks aj. *RFC 2617: HTTP Authentication: Basic and Digest Access Authentication*. URL: http://www.ietf.org/rfc/rfc2617.txt, 1999. [Online].

[22] Wikipedia. *Digest access authentication — Wikipedia, The Free Encyclopedia*. URL: http://en.wikipedia.org/w/index.php?title=Digest_access_authentication&oldid=208043814, 2008. [Online; cit. 4.5.2008].

[23] PHP Manual. *HTTP authentication with PHP*. URL: http://www.php.net/manual/en/features.http-auth.php, 2.5.2008. [Online; cit. 4.5.2008].

[24] Jakub Vrána. *Bezpečné přihlašování uživatelů*. URL: http://www.root.cz/clanky/bezpecne-prihlasovani-uzivatelu/, 13.4.2006. [Online; cit. 5.5.2008].

[25] Jakub Vrána. *Zabezpečení session proměnných*. URL: http://php.vrana.cz/zabezpeceni-session-promennych.php, 1.7.2005. [Online; cit. 5.5.2008].

[26] Wikipedia. *Authorization — Wikipedia, The Free Encyclopedia*. URL: http://en.wikipedia.org/w/index.php?title=Authorization&oldid=204408575, 2008. [Online; cit. 5.5.2008].

[27] Wikipedie. *Access control list — Wikipedie: Otevřená encyklopedie*. URL: http://cs.wikipedia.org/w/index.php?title=Access_control_list&oldid=2527409, 2008. [Online; cit. 5.5.2008].

Seznam příloh

Příloha A

Adresářová struktura přiloženého CD

- **/aplikace/**
  - **databaze/** – adresář s SQL skripty pro vytvoření tabulek databáze
  - **zdrojovy kod/** – adresář se zdrojovými texty ukázkové aplikace
  - **instalace.txt** – pokyny pro instalaci aplikace na server
- **/komponenta/**
  - **diagramy/** – adresář s diagramy tříd
  - **dokumentace/** – adresář s programovou dokumentací
  - **zdrojovy kod/** – adresář se zdrojovými texty komponenty
- **/technicka zprava/**
  - **zdrojovy kod/** – adresář se zdrojovými texty technické zprávy
  - **bp-xpesek07.pdf** – technická zpráva

Příloha B

Třídy Modelu ukázkové aplikace