České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Richard Cesar

Studijní program: Softwarové technologie a management

Obor: Inteligentní systémy

Název tématu: Webový klient pro databázový systém ExDB

Pokyny pro vypracování:

1. Navrhněte a realizujte webovou aplikaci sloužící jako klient pro databázový systém ExDB.
2. Proveďte rešerši existujících produktů a vytvořte formální analýzu požadavků kladených na tento typ aplikací.
3. Na základě provedené analýzy a požadavků vedoucího práce navrhněte a realizujte prototyp webové aplikace a ověřte jeho funkčnost v testovacím prostředí.
4. Při návrhu řešení se soustřeďte zejména na jeho modularitu, budoucí rozšiřitelnost a dodržování konvencí dohodnutých pro vývoj systému ExDB a jeho komponent.

Očekávanými výstupy práce jsou analytická dokumentace (nejlépe v notaci UML), komentované zdrojové kódy, kompletní instalační balíky vytvořené aplikace a stručná uživatelská a produktová dokumentace.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí bakalářské práce: Ing. Pavel Loupal, Ph.D.

Platnost zadání: do konce zimního semestru 2011/2012

L.S.

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 2. 2. 2011

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra kybernetiky

Bakalářská práce

Webový klient pro databázový systém ExDB

Richard Cesar

Vedoucí práce: Ing. Pavel Loupal, Ph.D.

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Inteligentní systémy

27. května 2011

Poděkování

Tímto bych chtěl poděkovat panu doktoru Pavlu Loupalovi za zasvěcení do projektu ExDB a za pomoc při budování této práce.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 27. 5. 2011 Richard Cesar .....

Abstract

The goal of this work is to implement a full-featured web client for accessing and manipulation of ExDB database servers. Java Enterprise Edition technology will be used for this implementation. Application is ment to be used by regular users and also by system administrators.

Abstrakt

Cílem této práce je implementace plnohodnotného webového klienta pro přístup a manipulaci s databázovými servery ExDB. K implementaci bude použita platforma Java Enterprise Edition. Aplikace by měla být určena jak pro běžné uživatele, tak i pro systémové administrátory.

Obsah

Seznam obrázků

Kapitola 1

Úvod

1.1 Co je ExDB ?

ExDB neboli Experimental Database je experimentální databázový server založený na technologii strukturovaných dat ve formátu XML. ExDB spadá do kategorie takzvaných NXD serverů, což znamená, že celý vnitřní model databází a všechna uložiště dat jsou realizována v XML formátu. Tímto se tento server odlišuje od takzvaných XML-enabled serverů, které řeší za pomoci XML dokumentů pouze importy a exporty dat.

Server ExDB je vyvíjen v jazyku Java a je v něm použita modulární struktura. Díky tomu je možné server snadno rozšiřovat o nové pluginy (moduly) a potažmo funkce. Server je navíc díky jazyku Java platformě nezávislý.

1.2 Motivace

Proč používat XML ? XML neboli eXtensible Markup Language je značkovací jazyk, který byl vyvinut konsorciem W3C [1]. Jde o zjednodušenou verzi jazyka SGML. XML je obecně uznávaný a standardizovaný formát pro přenášení univerzálních dat. XML formát je podporován širokou škálou operačních systémů a programů, má dobrou mezinárodní podporu a je snadno převoditelný do jiných formátů.

Navíc samotný XML dokument neobsahuje žádné formátovací ani vzhledové informace. Obsahuje pouze nutné značky, které vyznačují význam a strukturu jednotlivých částí dokumentu. Díky tomu získavá XML dokument vysokou informační hodnotu. Kromě toho poskytují XML dokumenty i některé pokročilé funkce jako například automatická kontrola struktury, což je zajištěno definičními soubory DTD. Tyto soubory obsahují definice struktury a značek, které mohou nebo musí být použity ve vlastním XML dokumentu.

Další přijemnou funkcí je odkazování se na jednotlivé elementy ve struktuře dokumentu. XML využívá k tomuto účelu některé speciální jazyky jako například XPath, XQuery, XPointer nebo XLink. XML formát standardně využívá kódování Unicode, což umožňuje použití více jazyků v jednom dokumentu.

Díky těmto jmenovaným vlastnostem je XML formát dobrým adeptem pro uložiště strukturovaných dat, což je základním předpokladem pro využití této technologie v prostředí databázových serverů.

Proč používat Javu ? Java je objektově orientovaný programovací jazyk vyvinutý firmou Sun Microsystems. Jedná se o jeden z nejrozšířenějších jazyků na světě. Java má relativně jednoduchou a přehlednou syntaxi. Dá se říci, že je lehce obměněnou obdobou jazyka C++. Přidává však mnohá vylepšení.

Jazyk Java je velmi robustní a ve své základní instalaci obsahuje podporu mnoha technologií. Je určen pro psaní vysoce spolehlivého softwaru. Z tohoto důvodu neumožňuje používání některých potenciálně nebezpečných programovacích technik jako například reference pomocí ukazatelů, uživatelská správa paměti a podobně. Syntaxe jazyka Java striktně vyžaduje typovou kontrolu a ošetření všech výjimek. Větší stabilitě programů také přispívají některé moderní komponenty jako například `GarbageCollector`, který automaticky vyhledává již nepoužívané oblasti paměti a uvolňuje je k dalšímu využití.

Jazyk Java se řadí k takzvaným interpretovaným jazykům, což znamená, že se při kompilaci nevytváří tradiční strojový kód. Místo toho vznikne jakýsi mezikód (byte-kód). Tento formát je již platformě nezávislý a je ho možné spustit na každé platformě, kde běží virtuální stroj Javy. Tento virtuální stroj pak zprostředkuje spojení mezi příslušnou platformou a Java byte-kódem. Z toho však vyplývá jedna zjevná nevýhoda. Programy spouštěné ve virtuálním stroji mají poněkud delší startovací časy. Nikterak to však neubírá na výkonu celého jazyka. Java je také velmi dynamická a umožňuje načítání dalších zdrojů a rozšiřování funkcionality programů a to přímo za chodu dané aplikace.

Z hlediska struktury kódu je navíc Java velmi elegantním a snadno čitelným jazykem.

Kapitola 2

Popis problému, specifikace cíle

V této kapitole se podíváme na obecný popis řešeného problému, specifikujeme si cíle práce a také se podíváme na přehled již existujících řešení.

2.1 Specifikace cíle

Naším úkolem je osvojit si technologie spojené s databázovým prostředím ExDB a následně vytvořit webovou aplikaci na platformě Java EE, která bude schopna sloužit co by klient pro přistup do ExDB databází.

2.2 Požadavky zadavatele na funkcionalitu cílové aplikace

Požadavky na cílovou aplikaci jsou následující :

- aplikace musí být napsána v jazyku Java
- aplikace musí být založena na modelu client-server a musí být schopna připojit se přostřednictvím síťového protokolu TCP/IP k serveru ExDB
- po připojení k serveru musí být aplikace schopna provádět následující úkoly :
  - odeslat autentizaci uživatele
  - vypsat seznam kolekcí a umožnit zvolení jedné z nich
  - vypsat obsah kolekce a umožnit procházení dat
  - umožnit uživateli zápis a změnu dat v kolekci
  - umožnit uživateli import či export dat v nativním XML formátu
  - umožnit uživateli efektivně vyhledávat v uložených datech
  - vypsat informace o serveru a jeho aktuálním stavu
  - umožnit administrátorům upravovat uživatelské účty, oprávnění, případně další administrátorské činnosti podle možností serveru

- aplikace by měla být náležitě odladěna a přizpůsobena pro běh ve webovém Java Servlet kontejneru
- aplikace musí být schopna obsloužit více klientů najednou

2.3 Webová aplikace vs. GUI klient

Proč používat při práci s databázovým serverem ExDB právě webového klienta? Srovnejme si nyní rozdíly mezi klasickým GUI klientem a webovým klientem.

Klasický GUI klient je v podstatě standardní aplikací, která běží v operačním systému klienta. Pro její zavedení do systému je většinou potřeba instalace a k jejímu běhu jsou často nutné určité knihovny, či jiné podpůrné zdroje. Aplikaci může většinou v danou chvíli používat pouze jeden uživatel na konkrétním systému (neuvažujeme nyní terminálové služby). Výhodou tohoto klienta je, že se spojuje přímo s databázovým serverem a nemá tedy žádné další požadavky na serverovou infrastrukturu.

Naproti tomu webový klient dokáže obsloužit mnoho klientů najednou, přičemž zde nejsou kladeny téměř žádné požadavky na softwarové vybavení klienta. Většinou stačí pouze webový prohlížeč, který je běžnou součástí všech známějších operačních systémů. Nevyžaduje žádnou instalaci na počítač klienta a výkonostně nijak nezaměstnává klientský systém. Nevýhodou je, že tato varianta vyžaduje značnou podporu na straně serveru. Webová aplikace potřebuje ke svému běhu určité serverové prostředí, které musí být náležitě nastaveno. Krom toho může aplikace spotřebovat velké množství hardwarových prostředků, zejména při velkém vytížení.

Při srovnání těchto řešení je rovněž nutné brát v úvahu bezpečnost obou aplikací. Klasický GUI klient komunikuje přímo se serverem na úrovni TCP/IP protokolu, což nutně vyžaduje, aby byl databázový server volně přístupný všem potenciálním klientům. To se může jevit jako bezpečnostní riziko zejména při použití na internetu. V tomto ohledu je webová aplikace jednoznačně lepší řešení. Ta totiž pracuje na známém protokolu HTTP, který je na internetu běžně používán. Pro správnou funkčnost aplikace je tedy třeba zpřístupnit pouze tuto službu. Veškerá interní komunikace mezi webovou aplikací a databázovým serverem je díky tomu skryta před vnějším světem a v 'nezabezpečené' síti se pohybují pouze vstupy a výstupy webové aplikace v zapouzdřeném protokolu HTTP. Nabízí se rovněž možnost využití protokolu HTTPS, který je schopen efektivně šifrovat data mezi klientem a webovou aplikací.

2.4 Srovnání existujících řešení

2.4.1 Úvod

V oblasti nativních XML databázových serverů neexistuje na poli webových klientů mnoho hotových řešení. Pokud nějaká existují, pak jsou svázána s konkrétní implementací daného serveru. Servery jsou různě pokročilé a weboví klienti tento fakt pochopitelně reflektují. U většiny případů však tyto aplikace nevyužívají potenciál technologie XML a jejich funkčnost je tedy pouze základní. Důvodem je patrně nedostatečné rozšíření tohoto druhu databázových serverů a dominance SQL technologie v této oblasti.

Z tohoto důvodu se ve srovnání hotových řešení podíváme i na některé SQL klienty, kterých máme na výběr výrazně více. Nabízená řešení jsou většinou dostatečná a víceméně využívají většinu možností jazyka SQL. Některé funkce by nám mohli být inspirací i v kategorii XML serverů.

V samotném projektu ExDB již v tuto chvíli existují dva klienti, kteří v zásadě umožňují základní operace a jsou tedy použitelní pro běžnou práci. Nicméně oba dva klienty je třeba instalovat a oba vyžadují přítomnost virtuálního stroje Java. Toto omezení ovšem u webové aplikace nemáme a stačí nám k ní pouze libovolný webový prohlížeč.

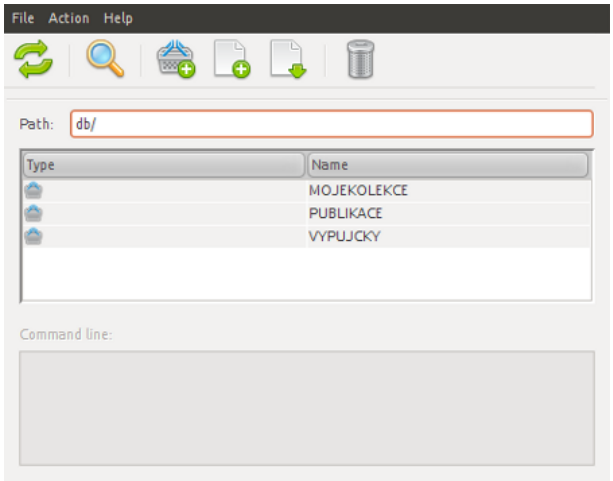
Z těchto důvodů je na místě, aby bylo implementováno řešení webového klienta pro ExDB, neboť tato komponenta může být užitečným nástrojem a příjemným doplňkem k projektu ExDB.

2.4.2 Řešení pro ExDB

Nejprve si představíme hotová řešení, která byla doposud vyvinuta v rámci projektu ExDB. Do této kategorie patří celkem dva klienti a sice ExDB CLI a GUI klient.

ExDB CLI Client Součástí balíku ExDB je konzolový textový klient, který byl vyvinut zejména pro testovací účely přímo vývojáři systému ExDB. Tento klient nemá nijak rozšířené funkce. Implementuje pouze základní příkazy pro práci s daty v rámci serveru. Konzolový klient spíše slouží jako demonstrace využití balíku ExDB-Commons pro potřeby klientské aplikace. Současná verze rovněž neobsahuje nápovědu ani jinou dokumentaci, díky čemuž je klient nepoužitelný pro běžné uživatele.

ExDB GUI Client V době psaní této práce je již na světě další implementace klienta, která vznikla v rámci projektu ExDB. Jedná se o bakalářskou práci Jana Truksy [11], která si kladla za cíl vytvořit grafického GUI klienta pro práci s prostředky ExDB serveru. Práce byla úspěšně obhájena a je nyní distribuována s balíkem ExDB. Klient se dá bez problémů použít pro manipulaci s kolekcemi a XML dokumenty včetně odesílání a zpracovávání XQuery a XPath dotazů. Grafické rozhraní je velmi příjemné a ovládání je intuitivní. Aplikace však neobsahuje žádné administrační funkce a je tedy profilována především na obyčejné uživatele. Pro správný běh programu je zapotřebí přítomnost knihoven ExDB-ClientAPI a ExDB-Commons. Obrázek 2.1 zobrazuje hlavní okno GUI klienta, kde můžeme vidět výpis kořenových kolekcí testovacího ExDB serveru.



Obrázek 2.1: ExDB GUI Client

2.4.3 Řešení pro jiné Native-XML databázové stroje

V této sekci se podíváme na některé Native-XML databázové stroje a jejich vybavenost co se týče klientských aplikací. U databázových serverů není zcela zvykem dodávat jako součást serverového řešení také webového klienta a u XML databází tomu není jinak. Pokud je v serverovém balíku přeci jenom nějaký webový klient zahrnut, bývá často velmi omezený. Databázové distribuce většinou obsahují pouze programátorské API a nanejvýš konzolové nástroje.

BaseX BaseX je NXD databázový server vyvíjený německou univerzitou Konstantz. Využívá efektivní XPath/XQuery procesor a umožuje zpracování rozsáhlých XML dat. Distribuce serveru obsahuje také GUI klienta pro vyhledávání v datech a následnou vizualizaci výsledku. Server je postaven na platformě Java. Nemá však žádnou obdobu webového klienta.

eXist DB eXist DB [3] je open-source XML databázový server vyvíjený komunitou a to v jazyce Java. Podporuje všechny významné standardy jako je XQuery 1.0, XPath 2.0, XSLT 1.0 nebo XSLT 2.0. Dále zde existuje částečná podpora pro XInclude, XPointer nebo XUpdate. Pro programátory je zde dostupné rozhraní XML:DB API.

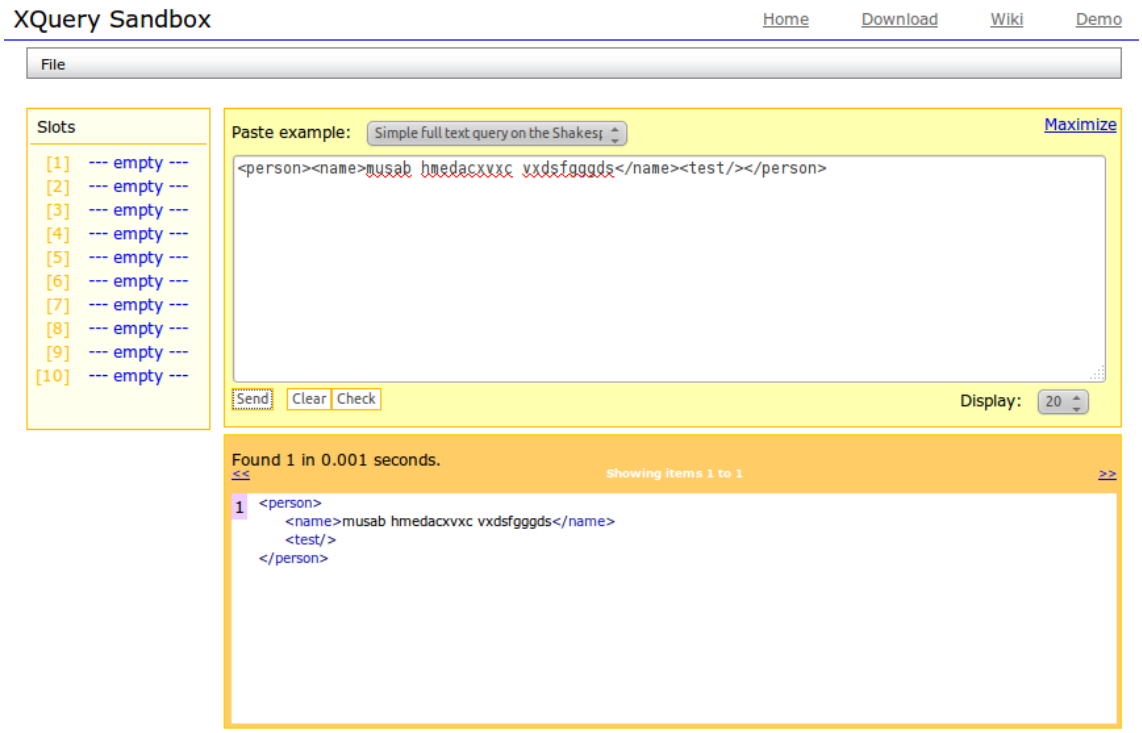
eXist DB může být použit jako samostatný server, nebo může distribuován jako knihovna, či jako modul uvnitř Java EE aplikace.

Distribuce databázového serveru obsahuje rovněž užitečné webové rozhraní pro přístup k serveru. Součástí tohoto rozhraní je mimo jiné také administrační sekce. Tato sekce umožňuje například zobrazení informací o stavu serveru, úpravu uživatelských účtů, procházení kolekcí a v neposlední řadě i zastavení, či restart serveru. Součástí webového rozhraní je i sada webových mini-aplikací.

Jednou z nich je i takzvaný sandbox, který umožňuje uživateli zavolat dotaz v jazyce XQuery a zobrazit jeho výstup z databáze. Celé řešení běží na předkonfigurovaném webovém

serveru Jetty. Mimo toto webové rozhraní obsahuje distribuce také GUI aplikaci (rovněž v Javě), která obsahuje pokročilejší funkce pro procházení a manipulaci s daty.

Obrázek 2.2 zobrazuje zmiňované rozhraní sandbox.

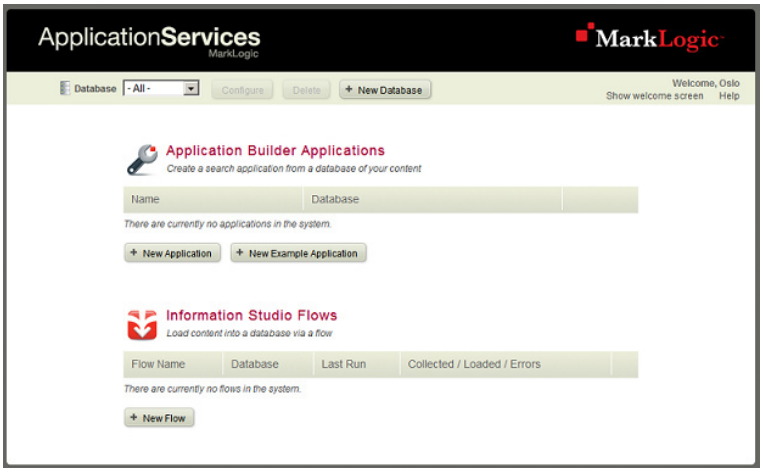


Obrázek 2.2: eXist DB XQuery Sandbox

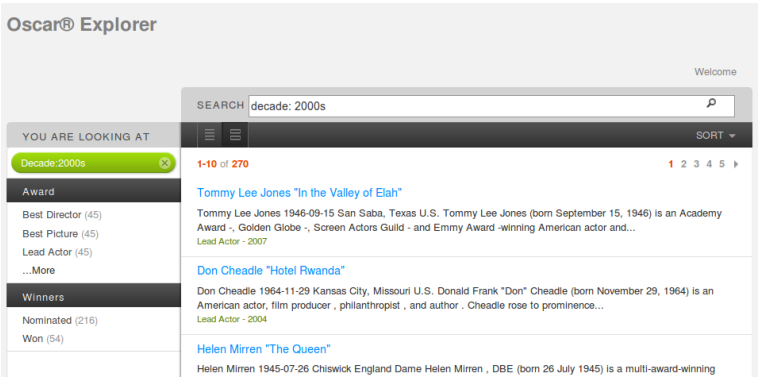
MarkLogic Server MarkLogic Server [2] je produktem stejnojmenné společnosti. Jádrem jeho vyhledávacích schopností je XQuery. Server umožňuje i full-textové vyhledávání v celé šíři XML dat. Součástí balíku služeb je také nástroj MarkLogic Application builder, který je určený k vývoji webových aplikací bez znalosti programovacího jazyka.

Jako příklad využití tohoto nástroje je v distribuci zahrnuta i webová aplikace Oscar Explorer. Aplikace umožňuje vyhledávání podle textových řetězců, nebo za pomoci jednoduchého dotazu. Dotaz se také automaticky upravuje v závislosti na zvolených filtrovacích kriteriích, které je možné zvolit ze zobrazené nabídky. Oscar Explorer rovněž obsahuje funkci 'napovídání' při zadávání hledaného dotazu. Kliknutím na jeden z výsledků vyhledávání se zobrazí konkrétní data. Aplikace však neobsahuje žádné administrační funkce, je pouze příkladem využití nástrojů MarkLogic serveru. Použití Oscar Exploreru je limitováno na konkrétní databázi a na konkrétní typ dat, ale v zásadě je možné vytvořit za pomoci MarkLogic Builderu univerzálnějšího klienta.

Na obrázku 2.3 můžeme vidět zmíněný MarkLogic ApplicationBuilder a na obrázku 2.4 je vidět testovací aplikace MarkLogic Oscar Explorer.



Obrázek 2.3: MarkLogic Application Builder



Obrázek 2.4: MarkLogic Oscar Explorer

2.4.4 Řešení pro databázové stroje SQL

Oracle iSQL*Plus Oracle iSQL*Plus je velice jednoduchý nástroj pro přistup do databází Oracle SQL. Nástroj je v podstatě webovou variantou malého konzolového nástroje SQLPlus, který je standardně dodáván s Oracle SQL Serverem. Tato webová aplikace má velmi jednoduché funkce. Umožňuje přihlášení uživatele k serveru a následné zadávání příkazů jazyka SQL. Po odeslání požadavku pouze zobrazí výsledek. Dále umožňuje načítání SQL skriptů nebo naopak jejich uložení. Nástroj také udržuje historii příkazů. V sekci nastavení umožňuje změnu hesla nebo nastavení některých provozních proměnných.

Nástroj většinou běží v rámci aplikačního serveru Oracle, nazývaný OC4J. Jedná se o implementaci platformy J2EE.

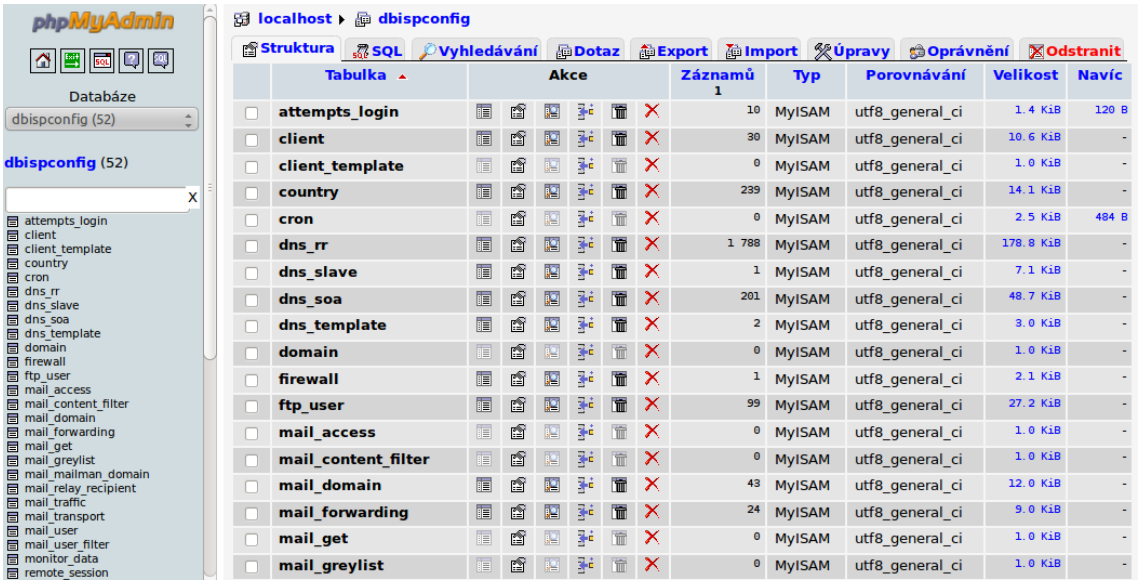
Více informací o používání tohoto nástroje lze nalézt v administrační příručce [4].

phpMyAdmin phpMyAdmin [6] je open-source projekt pro správu databází na serveru MySQL. Na rozdíl od všech předchozích není vyvíjen v Javě. Celý projekt je napsán v jazyce PHP, což je zdánlivě vzdálené téma vzhledem k technologiím, které budou použity v této práci. phpMyAdmin je však velmi komplexním nástrojem pro práci s databázemi a proto je příhodné se o něm zmínit.

Aplikace umožňuje uživateli procházet jednotlivé databáze, provádět malé i hromadné změny v datech. Je zde velmi robustní systém pro vyhledávání dat v tabulkách, či v celých databázích. Aplikace podporuje nejrůznější filtry a exporty, takže je možné velmi efektivně konvertovat data do jiných formátů a ty následně přenášet do jiných aplikací či dokonce jiných platforem. Můžeme zde najít dokonce grafického designera na tvorbu složitých SQL dotazů. U jednotlivých tabulek a databází je možné nastavovat nejrůznější parametry jako například oprávnění nebo použité kódování znaků.

phpMyAdmin rovněž obsahuje rozsáhlé administrační možnosti, které v neposlední řadě zahrnují například správu uživatelů, proměnné prostředí, parametry serveru, backendy uložišť, replikace a synchronizace s jinými servery nebo řízení služeb serveru v rámci operačního systému. Sledování stavu a parametrů serveru je samozřejmostí. phpMyAdmin je poměrně vyspělý i z hlediska zabezpečení. Z hlediska vzhledu podporuje phpMyAdmin témata a je vícejazyčný.

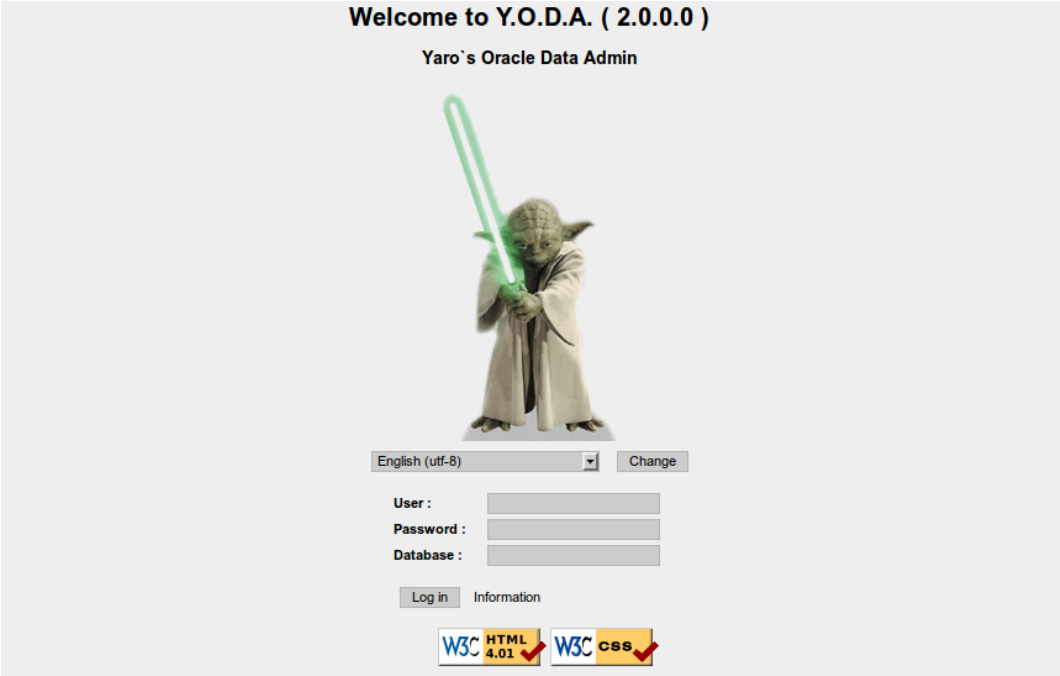
phpMyAdmin můžeme vidět v akci na obrázku 2.5.



Obrázek 2.5: phpMyAdmin

Yaro Oracle Data Admin Yaro Oracle Data Admin je PHP aplikace určená k prohlížení obsahu databází Oracle. Aplikace vznikla jakožto bakalářská [9] a následně diplomová práce na elektrotechnické fakultě ČVUT. Jejím autorem je Josef Jaroš. Mezi hlavní funkce patří prohlížení seznamu tabulek, nahlížení do obsahu, dále řazení, přidávání, mazání a úprava

tabulek. Mimo jiné také obsahuje vyhledávací funkce a možnosti importu/exportu dat. Na obrázku 2.6 vidíme přihlašovací obrazovku.



Obrázek 2.6: Yaro Oracle Data Admin

Kapitola 3

Analýza a návrh řešení

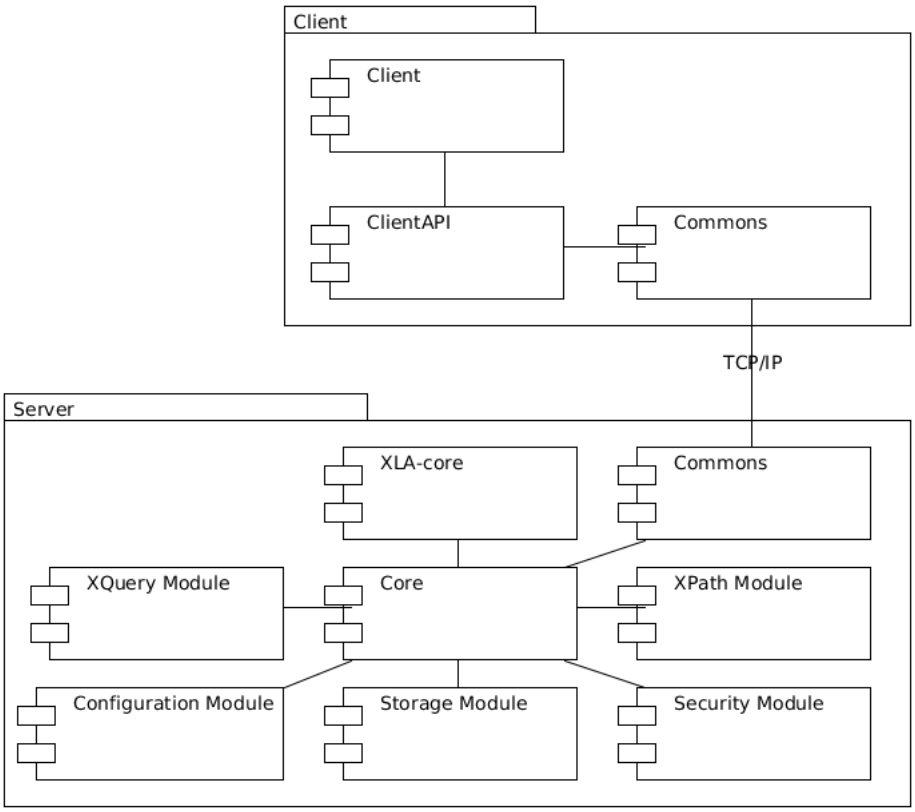
V této kapitole provedeme analýzu požadavků z minulé kapitoly a pokusíme se navhrnout vhodné řešení.

3.1 Architektura ExDB serveru

ExDB server, se kterým má výsledný klient komunikovat, je poměrně komplexní systém. Jeho modulární implementace zajišťuje snadnou rozšiřitelnost a již v době psaní této práce má serveru mnoho funkcí. Pro komunikaci s klienty využívá server svého vlastního protokolu, který je založen na rodině internetových protokolů TCP/IP. Server naslouchá na přednastaveném TCP portu a po přijetí spojení komunikuje s klienty za pomoci textových zpráv, které jsou zapouzdřeny v XML struktuře. Celý systém ExDB se skládá z několika samostatných balíků [8] jež představují jednotlivé komponenty systému. Obrázek 3.1 zobrazuje jednoduchý diagram komponent ExDB. Je zde také zobrazeno spojení s komponentami obecné klientské aplikace.

3.1.1 Balík ExDB-Commons

Tento balík zajišťuje definice všech struktur, které jsou nutné pro komunikaci s ExDB serverem a pochopitelně se používá jak na straně serveru, tak i na straně klienta. Jeho využití je demonstrováno na jednoduchém konzolovém klientovi, který je součástí balíku ExDB. Volba textového protokolu se v tomto případě může jevit jako poněkud nešťastná, neboť jak klient, tak i server jsou implementováni v jazyce Java. Z toho důvodu se jeví jako lepší volba nějaká forma Java serializace, která by byla pro obě strany srozumitelnější. Použití textového protokolu v původní implementaci má však zřejmý důvod. V rámci požadavků implementace serveru ExDB bylo totiž vyžadováno, aby byl server schopen komunikovat i s klienty jiných programovacích jazyků, které Java serializaci pochopitelně nemusí umožňovat. Protokol byl tedy nakonec upraven, aby podporoval komunikaci jak v textové, tak i v serializované podobě. Nová verze ExDB-Commons obsahuje nové dva protokoly, mezi kterými je možno volit. Zároveň byl také upraven způsob komunikace mezi serverem a klientem. Obě strany nyní odesílají před vlastními daty malou hlavičku, která oznamuje protistraně jaký typ přenosu bude následovat. Během testování balíku ExDB-Commons se však ukázalo, že jeho



Obrázek 3.1: Komponenty ExDB systému

využití není příliš vhodné pro implementaci klientské aplikace. Balík obsahuje především definice komunikačního protokolu a sám o sobě neobsahuje konkrétní zapouzdřené funkce a třídy, které by se daly snadno využít. Pokud bychom skutečně chtěli využit balíku ExDB-Commons v jeho současné podobě, museli bychom v klientské aplikaci vytvořit oddělenou vrstvu, která by implementovala komunikační funkce ve spojení s ExDB-Commons. Nakonec bylo zvoleno jiné řešení.

3.1.2 Knihovna ExDB-ClientAPI

V rámci semestrálního projektu, který předcházel této práci, byla vytvořena speciální knihovna, která slouží k zapouzdření komunikačního protokolu a rovněž vytváří ucelené programátorské API pro komunikaci s ExDB serverem. Knihovna nese název ExDB-ClientAPI a právě tato knihovna bude využita v našem webovém klientovi. Ve svém standardním nastavení bude využívat přenosu dat s Java serializací. Zde je však nutno podotknout, že implementace Java serializace je v této knihovně obsažena ve své první verzi. Serializují se zde pouze třídy charakterizující jednotlivé typy zpráv tak, jak jsou definovány v ExDB-Commons. Vlastní data, která jsou obsažena v těle zpráv serializována nejsou. Pokud si tedy například vyžádáme seznam dostupných dokumentů v nějaké konkrétní kolekci, obdržíme

sice serializovanou verzi odpovědi, nicméně vlastní data obsažená v odpovědi (seznam dostupných dokumentů) přijdou v textové podobě. Z tohoto důvodu je nutno mít na straně klienta funkci, která zpracuje textová data a převede je do objektu srozumitelného pro aplikační logiku klientské aplikace. Toto omezení je zde opět z důvodu kompatibility protokolu s ostatními klienty.

3.2 Analýza funkčních požadavků

Podívejme se nyní na funkční analýzu výsledné aplikace. Každý uživatel začíná na přihlašovací stránce, která ho vyzývá k zadání uživatelského jména a hesla pro přístup k prostředkům ExDB. Zde by již uživatel měl mít možnost měnit jazyk aplikace, aby případný zahraniční návštěvník pochopil, co se na něm žádá. Pokud se uživatel dostane do aplikace skrze URL odkaz, který odkazuje na konkrétní komponentu webové aplikace, měl by být přesměrován na přihlašovací stránku. To samozřejmě neplatí za situace, kdy je uživatel již přihlášen a jeho sezení je již aktivní.

Po přihlášení by měl být uživatel přesměrován na informační stránku, která ho uvítá v aplikaci a případně zobrazí některé užitečné informace například o stavu spojení či serveru. Na této stránce by také měla být možnost výběru další akce. Nejlépe nějakou formou hlavní nabídky, která bude pokud možno přístupná po celou dobu používání aplikace. Velmi užitečná by mohla být možnost vstoupit do jakési formy uživatelského panelu, který umožní změnu osobních nastavení uživatele jako například uživatelské heslo nebo některé preference chování webové aplikace.

Další funkce by měly být dosažitelné skrze moduly přístupné přes hlavní nabídku. Ty by měli být ovšem přístupné pouze uživatelům, kteří jsou držitelé patřičných oprávnění. V systému ExDB se dají v zásadě vymezit dvě uživatelské role. Běžný uživatel a administrátor. Běžný uživatel má mít přístup ke svým datům a to včetně manipulace s nimi. Administrátor by měl mít oproti běžnému uživateli přístup k vyšším funkcím systému jako je například správa bezpečnostních nastavení.

Jednotlivé akce přístupné konkrétním rolím uživatelů jsou shrnuty v diagramu 3.2.

Funkční moduly by měli být logicky rozdělené dle funkcí, které poskytují. Vzhledem k požadavkům, jež byly zadány v předchozí kapitole by se daly v aplikaci čekat například moduly jako databázový průzkumník, vyhledávač v datech nebo bezpečnostní manažer.

3.3 Analýza nefunkčních požadavků

Nefunkční požadavky pro výslednou aplikaci jsou dány především jejím webovým charakterem. Všechny grafické výstupy aplikace jsou k uživateli přenášeny protokolem HTTP (případně protokolem HTTPS). Uvnitř protokolu jsou zapouzdřena textová data představující grafický výstup aplikace formátovaný prostřednictvím značkovacího jazyka XHTML, případně pomocí přidružených technologií jako například CSS. Tyto jazyky mají striktně definovaná pravidla syntaxe a nutným požadavkem na aplikaci je tedy dodržování těchto závazných norem. Díky tomu by měla být zajištěna kompatibilita všech známých webových prohlížečů, a tím i všeobecná dostupnost webové aplikace na různých klientských platformách.

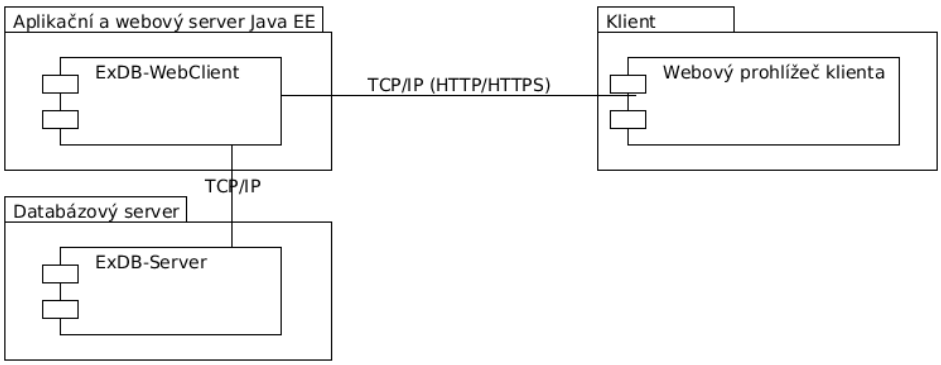
Obrázek 3.2: Use-case diagram uživatelských rolí

Komunikace mezi klientem a serverem je znázorněna v diagramu 3.3.

Dalším požadavkem na výslednou aplikaci je celkem pochopitelně její spolehlivost a ošetření všech předvídatelných chyb, které mohou nastat během užívání aplikace a to jak z hlediska systémových, tak i z hlediska uživatelských chyb.

Bere se rovněž za samozřejmost adekvátní odladění rychlosti aplikace samozřejmě v závislosti na vlastnostech operačního prostředí (rychlost hardwaru, rychlost sítě, použitý operační systém a podobně), ve kterém je systém provozován.

Od aplikace se také očekává snadná přenositelnost mezi platformami. Tento požadavek je naštěstí triviálně splněn díky nezávislosti jazyka Java na konkrétní platformě.



Obrázek 3.3: Schéma komunikace uživatele se serverem ExDB skrze webového klienta

3.4 Návrh modulů uživatelského rozhraní

V analýze funkčních požadavků jsme se dotkli možnosti rozdělení aplikace na dílčí funkční moduly. Nyní se pokusíme navrhnout přibližnou podobu a funkci těchto modulů.

3.4.1 Uživatelský modul

Uživatelský modul se skládá jednak z uvítací stránky a dále z uživatelského panelu. Uvítací stránka obsahuje popis aplikace a také ukazuje informace o verzi aplikace a stavu připojeného serveru. Je dostupná přes první volbu v hlavním menu na levé straně stránky.

Druhou část, tedy uživatelský panel, lze vyvolat kliknutím na stejnojmenný odkaz v pravé horní části obrazovky. Tento odkaz je pochopitelně přístupný až po přihlášení uživatele, neboť do té doby pozbývá smyslu. V uživatelském panelu je možné měnit preference aplikace ohledně vzhledu a také je zde možnost změnit uživatelské heslo do systému ExDB. Vzhled aplikace si může uživatel ve zmiňovaném panelu upravit volbou jednoho ze skinů.

3.4.2 Modul databázového průzkumníka

Dalším modulem je databázový průzkumník, který slouží k procházení datových struktur serveru a také slouží k provádění operací nad daty. Můžeme zde spravovat kolekce, nahrávat a stahovat XML dokumenty. U XML dokumentů se lze odkázat do vyhledávacího modulu. Ovládání databázového průzkumníka je intuitivní a v mnohém se snaží napodobit souborové manažery na které jsou uživatelé zvyklí z běžné práce na počítači. Databázového průzkumníka lze rovněž spustit kliknutím na odkaz hlavního menu na levé straně stránky.

3.4.3 Modul pro vyhledávání

Modul pro vyhledávání je dostupný buďto z hlavního menu nebo z databázového průzkumníka, kde lze rovnou zvolit soubor, nad kterým se má hledání provádět. Zde má uživatel možnost použít dva dotazovací jazyky, které jsou implementovány v serveru ExDB. Jedná se o jazyky XPath a XQuery. Výsledky hledání se zobrazují jako obyčejný text, což je dáno především charakterem komunikačního protokolu, který momentálně vrací za všech okolností pouze textovou odpověď. Možným vylepšením by bylo zobrazovat výsledky hledání například jako interaktivní strom elementů.

3.4.4 Modul bezpečnostních nastavení

Posledním modulem je správa bezpečnostních nastavení. Tento modul je určen pro systémové administrátory. Poskytuje funkce ke správě uživatelských účtů, rolí a oprávnění.

3.5 Použité technologie

Jedním z požadavků zadavatele bylo použití programovacího jazyku Java. V tomto případě bude nutno použít variantu Enterprise Edition, neboť se bude jednat o webovou aplikaci. Pro zjednodušení bude v této práci využito některých již hotových knihoven, které zjednoduší

implementaci běžných operací webové aplikace jako je validace vstupu uživatele, použití HTML šablon a podobně. Knihovny si nyní krátce popíšeme.

3.5.1 SpringSource Spring Framework verze 3.0

Spring je komplexní J2EE framework, který nabízí širokou škálu funkcí pro zjednodušení vývoje malých i rozsáhlých webových aplikací. Je jednoduchý na konfiguraci díky širokému využití Java anotací. Přesto je však dobře rozšiřitelný o další funkcionality. Zajišťuje mimo jiné třívrstvou architekturu založenou na modelu Model-View-Controller. Obsahuje rovněž podporu pro řadu technologií třetích stran jako například Hibernate, Toplink, JDO, iBATIS SQL Maps. Více informací o vlastnostech frameworku najdeme v referenční příručce [10].

3.5.2 Hibernate Validator

Hibernate Validator je knihovna implementující specifikaci JSR303 - Bean validation. Umožňuje validaci dat v modelech na základě pravidel obsažených v definici modelu. Tato knihovna byla zvolena především kvůli dobré podpoře ze strany Spring Frameworku a také díky své snadné použitelnosti.

3.5.3 Apache Tiles 2

Apache Tiles 2 je šablonovací framework určený ke zjednodušení vývoje uživatelského rozhraní webové aplikace. Díky této knihovně je možné nadefinovat jednotlivé fragmenty uživatelského rozhraní, které jsou následně automaticky spojeny na základě zadaných pravidel. Tato technika eliminuje nutnost vytvářet duplicitní kód pro různě části uživatelského rozhraní webové aplikace a tím výrazně zjednodušuje celý vývoj. Bližší informace k používání knihovny Apache Tiles 2 lze nalézt v projektové dokumentaci [7].

3.5.4 Implementační prostředí

K vlastní implementaci bylo využito programovací prostředí Eclipse [5] ve verzi Helios Service Release 1 spolu s modulem pro Java Enterprise Edition. Důvodem pro zvolení tohoto prostředí byla především moje osobní preference, neboť toto prostředí považuji za velmi pokročilé a mám s ním mnoho zkušeností. Eclipse je velmi rozšířené programovací IDE, které díky svým pokročilým možnostem nastavení poskytuje ucelený balík pro vývoj mnoha druhů aplikací. Pomocí pluginů a modulů lze toto prostředí rozšířit o mnoho funkcionalit a také lze touto cestou přidat podporu pro další programovací jazyky. Na druhou stranu je zde dlužno podotknout, že například v porovnání s konkurenčním prostředím NetBeans se může Eclipse jevit jako poněkud zmatečný a složitý na nastavení. Osobně to považuji za věc zvyku.

Kapitola 4

Realizace

V této kapitole se podrobněji podíváme na vlastní realizaci webového klienta.

4.1 Úprava serverové části ExDB

Při zkoumání funkcionalit aktuální verze ExDB serveru se ukázalo, že server postrádá některé funkce, které jsou nutné pro splnění cílů této práce.

Jak bylo popsáno ve schématu 3.1, server je rozdělen na několik funkčních modulů, které spolupracují prostřednictvím společného jádra serveru. Bohužel ne všechny moduly spolupracují tak, jak by měly. Často jim chybí některé důležité funkce nebo jsou tyto funkce poškozeny. Z těchto důvodů bylo nutno zasáhnout do hlavního kódu ExDB projektu s cílem napravit co nejvíce chyb a dostat tak stabilní verzi serveru vhodnou k vývoji a testování klientské aplikace.

Asi největší problémy se objevily u modulu `mod-storage`. U tohoto modulu nabízí ExDB hned dvě implementace, které je možné měnit prostřednictvím konfiguračního souboru. Implicitně je v ExDB nastavena implementace jménem `NativeStorage`. Ta z počátku vypadala použitelně. Korektně se inicializovala, aktivně komunikovala s ostatními moduly a bez problémů u ní fungovaly některé funkce jako vytváření a procházení kolekcí. První problém však nastal ve chvíli, kdy měla být poprvé použita funkce pro vkládání dokumentů. Ta skončila výjimkou a bližší zkoumání tohoto problému ukázalo, že závadu nebude příliš jednoduché odstranit.

Z tohoto důvodu bylo zvoleno alternativní řešení a sice přechod na druhou implementaci modulu `mod-storage` jménem `FileSystemStorage`. Tato implementace zjevně sloužila v ExDB jako vůbec první modul pro ukládání dat, než byl vyvinut `NativeStorage`. `FileSystemStorage` využívá (jak vidno z názvu) schopnosti souborového systému. Jako kolekce vytváří složky na disku serveru a dokumenty jsou do těchto složek ukládány jako samostatné soubory s textem v podobě XML struktury. Bohužel se i zde objevily některé problémy končící výjimkou, nicméně tyto problémy se podařilo eliminovat drobnými zásahy do kódu.

Jistým příslibem řešení problémů s modulem `mod-storage` by mohla být nová implementace s názvem `EnhancedStorage`, která v současnosti vzniká jakožto součást jiné bakalářské práce.

Další zádrhel nastal při zkoušení bezpečnostních funkcí serveru. V rámci předchozích bakalářských případně diplomových prací byl v projektu ExDB vyvinut nový bezpečnostní framework založený na technologii XACML. Tento framework byl nakonec zapouzdřen do oficiálního ExDB modulu jménem `mod-security`. Tento modul sliboval mnoho pokročilých funkcí, se kterými bylo počítáno i při zadávání požadavků a cílů této práce. Při bližším zkoumání se ovšem ukázalo, že jmenovaný modul sice je součástí hlavní vývojové větve ExDB, ale naneštěstí není doimplementována jeho podpora v hlavním serveru.

Tam se využívá dnes již velmi obsolentní třída jménem `AccessControlList`, která historicky sloužila jako první nástřel bezpečnostních funkcí v projektu ExDB. Podle toho také vypadá její stávající zdrojový kód. Obsahuje některé elementární funkce, které zahrnují autentizaci, přidávání a odebírání uživatelů. Jako uložiště uživatelských účtů slouží textový soubor umístěný v kořenové složce serveru. Ten obsahuje na každém řádku dvojici uživatelského jména a hesla. Textový soubor je po startu serveru zpracován třídou `AccessControlList` a jeho obsah je udržován v hashovací tabulce. V případě přidání či odebrání uživatele je obsah hashovací tabulky aktualizován a následně uložen do zmiňovaného souboru. Tato třída každopádně neobsahuje žádné pokročilejší funkce jako například přidělování rolí nebo práv ke kolekcím, či snad dokonce k dokumentům, a je tedy zcela nepoužitelná pro cíle stanovené pro tuto práci.

Jediným řešením tohoto problému se tedy jevilo zprovoznění aktuálního bezpečnostního modulu `mod-security` s implementací XACML. Po dalším zkoumání se však objevily další problémy a od tohoto nápadu bylo upuštěno.

V souvislosti s bezpečnostním modulem serveru byla také objevena závažná chyba, která za jistých okolností umožňovala manipulaci s daty serveru bez řádného přihlášení.

Z těchto důvodů bylo nakonec po konzultaci s vedoucím práce rozhodnuto, že se webový klient bude muset obejít bez pokročilejších bezpečnostních funkcí, ač byly původně přislíbeny v zadání. Webový klient bude tedy podporovat pouze administraci uživatelských kont bez ohledu na jejich úroveň oprávnění.

Kromě řešení problémů s nefunkčními moduly ExDB bylo také třeba některé funkce doplnit. V již zmiňované třídě `AccessControlList` byla doplněna funkce pro výpis systémových uživatelů a rovněž byl přidán související prototyp zprávy do komunikačního protokolu ExDB.

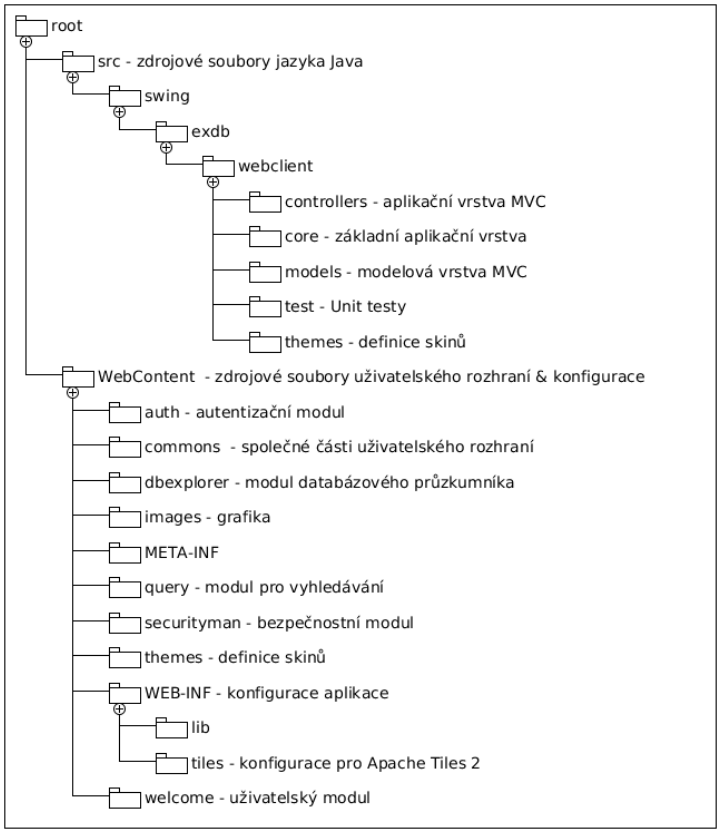
Dále byla přidána podpora pro změnu uživatelského hesla. Ta již byla částečně implementována na straně serveru, ale neexistoval pro ní prototyp zprávy v komunikačním protokolu. Ten byl doplněn a spolu s ním přibyla i další zpráva jménem `NoOperationRequest`, která svou funkcí připomíná známý diagnostický příkaz ping. Jinými slovy se tedy jedná o požadavek klienta na odezvu serveru. Této zprávy je v klientské aplikaci využíváno k udržení spojení se serverem. Mimo to slouží `NoOperationRequest` také ke kontrole připravenosti serveru a k měření jeho odezvy.

Všechny změny v protokolu byly samozřejmě také zohledněny v knihovně ExDB-ClientAPI.

Kromě hlavního kódu serveru byly také upraveny některé části knihovny XLA-core, které souvisely s generováním a načítáním textových XML dat.

4.2 Adresářová struktura webové aplikace

Aplikace využívá poměrně hodně knihoven, které často vyžadují vlastní konfigurační soubory nebo jiné zdroje ze souborového systému. Z tohoto důvodu je vhodné alespoň ve zkratce popsat adresářovou strukturu aplikace a upřesnit tak lokace jednotlivých souborů, které jsou v této práci zmíněny. Některé adresáře jsou běžné i u jiných J2EE aplikací, takže ty nebudou překvapením. Podívejme se nyní na přehled adresářů na obrázku 4.1



Obrázek 4.1: Přehled adresářové struktury webové aplikace

4.3 Architektura webové aplikace

4.3.1 MVC

ExDB-WebClient je standardní webovou aplikací založenou na platformě Java Enterprise Edition. Konkrétně využívá technologie Java servletů, která je ovšem zapouzdřena do Spring Frameworku, který poskytuje všechny nutné služby. Spring Framework jako takový je stavěn na modelu architektury MVC, neboli Model-View-Controller. Tato architektura si klade za cíl rozdělit aplikaci na tři vrstvy a sice datový model, uživatelské rozhraní a aplikační logiku.

Tyto vrstvy spolu komunikují přes předem určené rozhraní a jejich vlastní zdrojový kód je zcela oddělen. Díky tomu je možné provádět dílčí i masivní změny v kódu jedné vrstvy aniž by se musel zároveň přizpůsobovat kód jiné vrstvy.

Controller tedy představuje aplikační logiku, View vrstva představuje zobrazovací technologie (v našem případě je to XHTML kód a vše kolem stavby XHTML struktury) a Model vrstva jsou veškeré datové modely.

V tomto projektu jsou jednotlivé vrstvy rozděleny do dvou zdrojových Java balíků `swing.exdb.webclient.controllers` a `swing.exdb.webclient.models`. Vrstva View je v našem případě skryta ve složce WebContent coby XHTML kód a patří sem také kód knihovny Apache Tiles 2. Zvláštním případem je balík `swing.exdb.webclient.core`, který obsahuje některé systémové třídy (například databázový driver, connection interceptory a výjimky). Tento balík by se dal zařadit rovněž do aplikační logiky, nicméně jeho třídy nejsou kontrolery v pravém slova smyslu.

4.3.2 Komunikační vrstva

Připojení webové aplikace k databázovému stroji má na starost třída `EXDBDatabaseDriver`, která víceméně zapouzdřuje funkcionalitu ExDB-ClientAPI a odděluje tuto knihovnu od zbytku aplikace. V souvislosti s databázovým ovladačem jsou k dispozici i dvě vyjímky a sice `ExDBCommException` a `ExDBConnectionException`. Ty si kladou za cíl zapouzdřit všechny chybové stavy a vyjímky vyhazované knihovnou ClientAPI.

Instance ovladače je součástí třídy `BaseController`, která je bázovou třídou pro každý kontroler, takže je ovladač vždy k dispozici. Mimo to je ovladač také volán ze třídy jménem `ConnectionInterceptor`. Tato třída je automaticky volána webovým frameworkem při každém požadavku klienta a jeho úkolem je kontrola spojení s databázovým strojem.

Kontrola probíhá jednak ověřením stavu vnitřních síťových objektů (například stav otevřeného socketu), ale také zasláním požadavku na odezvu serveru. Zde je mimo jiné měřena odezva serveru. Je rovněž kontrolován stav přihlášení k databázovému stroji. Není-li něco v pořádku, aplikace zobrazí uživateli chybové hlášení a vyzve ho k novému přihlášení do systému.

Problematika interceptorů je blíže popisována později v této kapitole.

4.3.3 Datová vrstva - modely

Datová vrstva je v aplikaci využívána především ke komunikaci aplikační logiky s vrstvou uživatelského rozhraní. Jedná se o modely reprezentované tradiční Java třídou. Obsahují pouze jednoduché datové typy a jediné funkce obsažené v těchto třídách jsou gettery a settery.

Důležitou součástí každého modelu jsou však anotace spojené s validačními pravidly jednotlivých proměnných. Anotace pochopitelně pochází z knihovny Hibernate Validator. Přestože zmíněná knihovna obsahuje mnoho možností validace, v našem webovém klientovi jsou využity pouze základní pravidla jako například test na neprázdnou proměnnou, test na minimální délku proměnné nebo validace dle regulárních výrazů. Využití validace je blíže popsáno v následující sekci.

4.3.4 Vrstva aplikačí logiky - kontrolery

Vrstva aplikační logiky je zde řešena pomocí Spring kontrolerů. Kontrolery jsou obyčejné Java třídy a nejsou založeny na žádné bázové třídě z webového frameworku. Jejich začlenění do Spring frameworku z větší části záleží především na správné konfiguraci pomocí Java anotací. Těch existuje mnoho a proto si zde pro názornost rozebereme jeden z kontrolerů (konkrétně kontroler bezpečnostního modulu) a ukážeme si na něm spolupráci s ostatními vrstvami aplikace a se samotným Spring frameworkem.

Ukažme si nejprve záhlaví třídy `SecurityManController`.

```
@Controller
@RequestMapping("/securityman")
public class SecurityManController extends BaseController {
```

Jsou zde vidět dvě anotace. Význam první anotace je celkem zřejmý. Druhá anotace říká, že tento kontroler se bude volat v případě, že se v URL objeví cesta `/securityman/`. Cesta je pochopitelně relativní k URL aplikace. Tedy v tomto případě je URL například `http://localhost:8080/exdbweb/securityman/`

V každém kontroleru se dále objevují funkce podobné tomuto prototypu.

```
@RequestMapping("/main.do")
public ModelAndView main() {
        ModelAndView mav = new ModelAndView("securityman.main");
        return mav;
}
```

Anotace nad funkcí říká, že tato funkce bude volána, jestliže se v URL objeví `/main.do`. V tomto případě je cesta relativní k URL kontroleru, tedy například
`http://localhost:8080/exdbweb/securityman/main.do`

Povšimněme si, že funkce vrací instanci třídy `ModelAndView`. Tato třída v sobě zapouzdřuje referenci na zobrazení, které se má odeslat klientovi coby uživatelské rozhraní. Jedná se o textovou informaci, která je následně předána vrstvě uživatelského rozhraní. Ta tento řetězec vyhodnotí a odešle klientovi patřičný grafický výstup. V našem případě používáme jako vrstvu uživatelského rozhraní knihovnu Apache Tiles 2. Textový řetězec `securityman.main` z přechozího příkladu je identifikátorem zobrazení dle konfigurace Apache Tiles 2. Zde je nutno brát v potaz, že třída `ModelAndView` není jedinou možnou návratovou hodnotou funkce. Možností je více a jejich konkrétní znění najdeme v referenci frameworku Spring [10]. Můžeme například vrátit pouze hodnotu typu String, jež obsahuje identifikátor zobrazení. Vyvstává otázka proč vlastně používat třídy `ModelAndView` a další jí podobné? Odpověď můžeme nalézt v dalším příkladu.

```
@RequestMapping(value = "/createuser.do", method = RequestMethod.GET)
public ModelAndView showCreateUser() {
        ModelAndView mav = new ModelAndView("securityman.createuserform");
        CreateUserModel user = new CreateUserModel();
        mav.addObject("user", user);
        return mav;
}
```

V této funkci již využíváme možností třídy `ModelAndView`. Konkrétně zde předáváme instanci třídy `CreateUserModel`, kterou následně můžeme využít ve vrstvě uživatelského rozhraní. Povšimněme si také, že zde přibyl parametr anotace `@RequestMapping`, který říká, že tato funkce bude volána pouze v případě, že je daná URL zavolána metodou GET. Naproti tomu by bylo vhodné uvést protipříklad s metodou POST.

```
@RequestMapping(value = "/createuser.do", method = RequestMethod.POST)
public ModelAndView createUser(
  @ModelAttribute("user") @Valid CreateUserModel user,
  BindingResult res) throws DatabaseCommException {
        ModelAndView mav = new ModelAndView("redirect: main.do");
        if(res.hasErrors()) {
                mav.setViewName("securityman.createuserform");
                mav.addObject("user", user);
                return mav;
        }

        if(dbDriver.createUser(user.getUsername(), user.getPassword())) {
          mav.addObject("infoMsgKey", "SecurityMan.CreateUser.Success");
        } else {
          mav.addObject("errorMsgKey", "SecurityMan.CreateUser.Error");
        }
        return mav;
}
```

Tato funkce je typickým příkladem pro zpracovávání údajů z formuláře. Používáme zde metodu POST, neboť není vhodné, aby se parametry formuláře zobrazovali v URL. Povšimněme si zde vstupních argumentů funkce.

Prvním argumentem je instance třídy `CreateUserModel`, která obsahuje data obdržená z formuláře. Před argumentem předcházejí dvě anotace. První anotace `@ModelAttribute` říká, že data tohoto argumentu pocházejí z atributu modelu jménem `user`. Je to v podstatě odkaz na proměnnou, která je využívána ve vrstvě uživatelského rozhraní jako uložiště pro data z formuláře. Dále je tu ještě druhá anotace jménem `@Valid`. Tou oznamujeme Spring frameworku, že bychom rádi dostali tento argument validní dle pravidel, která jsou určena v definici třídy `CreateUserModel`. V tuto chvíli tedy nastupuje na scénu Hibernate Validator.

Druhým argumentem funkce je instance třídy `BindingResult`. Z té se můžeme mimo jiné dozvědět výsledek validace.

Pokračujeme-li dále ve funkci, můžeme si povšimnout poněkud jiného identifikátoru v konstruktoru třídy `ModelAndView`. Na rozdíl od předchozích příkladů zde totiž požadujeme, aby byl uživatel po skončení funkce přesměrován na jinou stránku.

Tohoto postupu využíváme kvůli tomu, aby uživatel nemohl znovu zavolat tuto funkci se stejnými parametry použitím tlačítka pro obnovení stránky ve svém prohlížeči. Jediná situace kdy je přesměrování nežádoucí je případ, kdy uživatel zadal chybné údaje a neprošel tak validací. V takovém případě si přejeme zobrazit formulář znovu. Nastavíme tedy identifikátor zobrazení na stejný formulář, ze kterého byla tato funkce zavolána (viz. první `if` blok ve funkci).

Toto byly příklady funkcí, které jsou spouštěny na základě mapování URL. V kontroleru ovšem existují i jiné funkce, které jsou volány jiným způsobem. Jedním z představitelů takových funkcí vidíme v následujícím příkladu.

```
@ModelAttribute("userlist")
public List<String> populateUserList() throws DatabaseCommException {
        return dbDriver.listUsers();
}
```

Tato funkce je volána Spring frameworkem ve chvíli, kdy si vrstva uživatelského rozhraní vyžádá hodnotu konkrétní proměnné, v tomto případě proměnné `userList`. Návratová hodnota je zde typu `List<String>`, ale je samozřejmě možné použít libovolný datový typ, který je podporován vrstvou uživatelského rozhraní.

A v poslední řadě se ještě podívejme na jeden příklad.

```
@ExceptionHandler(DatabaseCommException.class)
public ModelAndView catchException(DatabaseCommException e) {
        ModelAndView mav = new ModelAndView("dbexplorer.main");
        mav.addObject("errorMsgKey", e.getMessage());
        return mav;

}
```

Příklad ukazuje funkci, která má na starosti odchytávání a zpracovávání výjimek. Předchází jí jednoduchá anotace určující typ výjimky, která se má odchytit.

V naší aplikaci odchytáváme v zásadě pouze dvě výjimky. Jednu z nich můžeme vidět i v příkladu. Zmiňovaná vyjímka `DatabaseCommException` postihuje téměř všechny chyby, které by mohly nastat při komunikaci s databází. Druhá výjimka s podobným názvem `DatabaseConnectionException` je použita pouze při vytváření spojení. Jiné výjimky se při testování neprojevily a proto nejsou žádné další odchytávány.

Závěrem této sekce se ještě jednou vraťme k záhlaví třídy `SecurityManController`. Opomněli jsme zde jeden fakt, že tato třída je potomkem jiné třídy jménem `BaseController`.

Ta má za úkol poskytnout ostatním kontrolerům patřičné zázemí v podobě standardních objektů, které jsou potřeba pro správnou funkčnost všech kontrolerů. Je zde například instance již zmiňované třídy `ExDBDatabaseDriver` nebo také instance třídy `MessageSource`,

která slouží k načítání lokalizovaných textů ze souborových zdrojů. Obě tyto proměnné mají anotaci `@Autowired`. Ta zajišťuje automatické vytvoření zmiňovaných tříd a rovněž zajistí jejich přežití napříč celým sezením připojeného uživatele. Každý kontroler má díky tomu vždy k dispozici aktuální verzi těchto tříd.

`BaseController` mimo jiné obsahuje také tři další funkce, které zajišťují inicializaci některých proměnných pro vrstvu uživatelského rozhraní. Využití těchto funkcí je blíže popsáno v následující sekci.

4.3.5 Vrstva uživatelského rozhraní

Vrstva pro zobrazení uživatelského rozhraní je úzce spjata s knihovnou Apache Tiles 2. Tato knihovna umožňuje postupné skládání výsledné webové stránky z menších fragmentů kódu. Fragmenty jsou postupně skládány jednak na základě pevné konfigurace, ale také na základě dynamických parametrů, které vyplynou z aplikační logiky.

Fragmenty kódu jsou uloženy v šablonách, které jsou rozmístěny ve složce `WebContent`. Víceméně jsou rozděleny do složek podle logických modulů aplikace. Určitou výjimkou je však složka `WebContent/commons`. Tato složka obsahuje některé společné části uživatelského rozhraní jako je hlavička, patička nebo hlavní menu. V této složce můžeme najít i jeden speciální soubor jménem `WebContent/commons/base.jsp`, který obsahuje XHTML hlavičku a určuje základní členění XHTML kódu, do kterého zahrnuje jednotlivé další fragmenty kódu.

Logika skládání fragmentů kódu je určena XML konfigurací, která je umístěna ve složce `WebContent/WEB-INF/tiles/`. Zde jsou umístěny konfigurační soubory pro jednotlivé moduly uživatelského rozhraní. Tyto XML soubory obsahují pojmenované definice, jež určují pořadí skládání fragmentů XHTML kódu.

V rámci konfigurace umožňují Apache Tiles 2 vytvořit určitou dědičnost mezi definicemi a je tedy možné aby jedna definice rozšiřovala jinou. Toho je zde využito u definice `page.base`, která určuje základní členění stránky. To je pak rozšířeno jinými definicemi o konkrétní bloky s dynamickým obsahem. Jména definic jsou použita jako identifikátor konkrétního zobrazení pro aplikační logiku.

Samotné šablony jsou napsány v již zmiňovaném jazyce XHTML konkrétně verze 1.0 Transitional. Kromě tohoto jazyka je v šablonách využiváno technologie JavaServer Pages, konkrétně je zde užíváno tagů z komponenty JSTL, neboli JavaServer Pages Standard Tags Library.

Zajimavým vylepšením aplikace je využití Apache Tiles 2, JSTL tagů a Spring kontrolerů k zobrazování informačních a chybových hlášení.

Pomocí Apache Tiles 2 je do každé stránky vnořen fragment kódu ze zdrojového souboru `WebContent/commons/messages.jsp`. V tomto souboru je umístěna rutina, která zobrazí informační či chybové hlášení a to pouze v případě, že v modelu konkrétního zobrazení nalezne jednu z proměnných `infoMsgKey` nebo `errorMsgKey`. Pokud nalezne jednu z těchto proměnných, použije její obsah jakožto klíč k získání lokalizované zprávy pomocí JSTL tagu `<fmt:message />`. Zpráva se poté graficky zobrazí na konkrétní stránce.

Zmíněné proměnné může do modelu umístit buďto kontroler (přímým přidáním proměnné do modelu například prostřednictvím třídy `ModelAndView`) nebo jí může do modelu umístit libovolná komponenta webové aplikace přidáním adekvátního řetězce do URL zobrazované

stránky. Této možnosti je hojně využíváno při přesměrování mezi funkcemi kontrolerů. Ukládání těchto parametrů z URL do modelu je realizováno ve funkcích třídy `BaseController`.

Parametrů v URL je také využíváno pro jazykovou lokalizaci prostředí a dále pro změnu aktuálního skinu aplikace. Tato funkcionalita je implementována přímo ve Spring frameworku a je blíže popsána v následující sekci.

4.4 Konfigurace webové aplikace

Problematika konfigurace výsledné aplikace by mohla zdánlivě zapadat spíše do instalační příručky. Nicméně instalační příručka popisuje pouze konfiguraci, která je individuální pro každou konkrétní instalaci. Webová aplikace však obsahuje řadu jiných nastavení, která se zadávají již při vývoji a později se nemění. Tato nastavení by tedy běžného uživatele nemusela zajímat, ale pro pochopení některých funkcí je vhodné o této problematice něco vědet.

4.4.1 Konfigurace servletu

Konfigurace servletu je uložena v souboru `WebContent/WEB-INF/web.xml`. V našem případě má však velice jednoduché znění. Zásadní pasáží konfigurace ja následující kus kódu.

```
<servlet>
        <servlet-name>exdbweb</servlet-name>
        <servlet-class>
                org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>exdbweb</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Dle tohoto nastavení se veškeré dění na URL `/exdbweb/*.do` přesměrovává na takzvaný `DispatcherServlet` obsažený ve Spring frameworku, který následně spustí rutinu obsluhy klienta a provádí veškerá vyhodnocení.

4.4.2 Konfigurace Spring frameworku

Konfigurace Spring frameworku je již obsáhlejší. Ta je implicitně uložena v XML souboru `WebContent/WEB-INF/exdbweb-servlet.xml`. V podstatě se jedná o běžný konfigurační soubor J2EE servletu ovšem s několika vylepšeními.

Velice důležitý je již první tag `<mvc:annotation-driven />`. Ten totiž aktivuje využívání Java anotací a výrazně tím snižuje nutnost konfigurace skrze XML soubory. S tímto tagem také souvisí další jménem `<context:component-scan />`, který definuje Java balíky, v nichž se mají hledat třídy s anotacemi. Příklad konfigurace vidíme dále.

```
<mvc:annotation-driven />
<context:component-scan base-package="swing.exdb.webclient"
  scoped-proxy="targetClass" />
```

Následující blok konfigurace ohraničený tagy `<mvc:interceptors>` nastavuje pravidla ohledně volání takzvaných interceptorů. Interceptory jsou speciální třídy, které implementují interface `HandlerInterceptor`, jenž popisuje funkce `preHandle` a `postHandle`. Jakmile jsou tyto třídy zaregistrovány do konfigurace, Spring framework je zařadí do řetězce volání při zpracovávání požadavku klienta. Je tak možné získat kontrolu nad klientským požadavkem ještě předtím, než projde mapováním a dostane se do patřičného kontroleru. V naší konfiguraci se nachází odkaz hned na tři interceptory. První z nich je definován následovně.

```
<bean id="localeChangeInterceptor"
  class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor" >
        <property name="paramName" value="lang" />
</bean>
```

Uváděný `localeChangeInterceptor` slouží k dynamické změně jazyka a tedy i k lokalizaci textů v aplikaci. Využívá k tomu parametr z URL, který je zde definován jako vlastnost `paramName`.

Dalším použitým interceptorem je `themeChangeInterceptor`.

```
<bean id="themeChangeInterceptor"
  class="org.springframework.web.servlet.theme.ThemeChangeInterceptor">
        <property name="paramName" value="theme" />
</bean>
```

`themeChangeInterceptor` dělá v podstatě totéž jako předchozí. Pouze místo lokalizace mění skin.

Poslední interceptor už je náš vlastní a byl popisován již dříve v této kapitole. Jedná se o `ConnectionInterceptor`, který udržuje spojení s ExDB serverem. Je zde pouze malá změna. Tento interceptor je definován pouze pro některé URL adresy. Pokud totiž přistupujeme do autentizačního modulu a chceme se přihlásit k ExDB, je funkce tohoto interceptoru nežádoucí a vznikala by tu směrovací smyčka. Musíme brát v úvahu, že před přihlášením k ExDB je síťový socket uzavřen a přihlášení je neplatné. Kdybychom v takovém případě použili `ConnectionInterceptor` na všechny URL, aplikace by směrovala uživatele donekonečna na přihlašovací stránku.

Příklad konkrétní konfigurace je ukázán v dalším bloku kódu.

```
<mvc:interceptor>
        <mvc:mapping path="/dbexplorer/*" />
        <mvc:mapping path="/securityman/*" />
        <mvc:mapping path="/query/*" />
        <mvc:mapping path="/welcome/*" />
        <bean id="connectionInterceptor"
          class="swing.exdb.webclient.core.ConnectionInterceptor" />
</mvc:interceptor>
```

V tuto chvíli zůstaňme ještě u problematiky jazykových lokalizací textů. V konfiguraci můžeme k tomuto tématu nalézt ještě dvě další sekce. Jednou z nich je definice beanu `messageSource`.

```
<bean id="messageSource"
  class="org.springframework.context.support.
    ReloadableResourceBundleMessageSource">
        <property name="basenames">
                <list>
                        <value>WEB-INF/messages</value>
                </list>
        </property>
        <property name="defaultEncoding" value="UTF-8" />
        <property name="cacheSeconds" value="-1" />
        <property name="fallbackToSystemLocale" value="false" />
</bean>
```

Ta nedělá nic jiného, než že načítá lokalizované texty ze souborů na disku v závislosti na nastaveném jazyce. Tato funkce je běžná i v normálních J2EE aplikacích. Nicméně implementace této funkcionality dodávaná se Spring frameworkem má jednu zásadní výhodu. Umožňuje totiž na rozdíl od běžné Javy EE nativní načítání takzvaných Java Properties souborů v různých kódováních a to včetně UTF-8. V předchozím bloku kódu můžeme vidět patřičný tag nastavující vlastnost `defaultEncoding`.

Druhou definicí související s lokalizaci textů je `localeResolver`.

```
<bean id="localeResolver"
  class="org.springframework.web.servlet.i18n.CookieLocaleResolver">
        <property name="cookieName" value="client_lang" />
        <property name="cookieMaxAge" value="100000" />
</bean>
```

Třída `localeResolver` má na starost ukládání a načítání jazykové preference klienta za pomocí takzvaných Cookies [1], které jsou uloženy v počítači klienta. V konfiguračním souboru najdeme také velice podobné definice pro načítání skinů. Následuje příklad.

```
<bean id="themeResolver"
  class="org.springframework.web.servlet.theme.CookieThemeResolver" >
        <property name="defaultThemeName" value="blues" />
        <property name="cookieName" value="client_theme" />
        <property name="cookieMaxAge" value="100000" />
</bean>
```

[1] Jako Cookie se v protokolu HTTP označuje malé množství dat, které webový server pošle prohlížeči uživatele. Ten pak uloží data do malého souboru v počítači uživatele a při příští návštěvě webové stránky jsou tato data zaslána zpět na webový server. Data většinou slouží k ukládání jednoduchých uživatelských informací, které se vztahují k navštívené webové stránce. [12]

```
<bean id="themeSource"
  class="org.springframework.ui.context.support.ResourceBundleThemeSource">
        <property name="basenamePrefix"
          value="swing/exdb/webclient/themes/theme-" />
</bean>
```

Funkcionalita je prakticky stejná jako u načítání lokalizovaných textů. Rozdíl je pouze v umístění Java Properties souborů. Tyto beany z nějakého důvodu neumožňují umístění definic skinů do složky `WebContent/WEB-INF`, jak je tomu zvykem například u jazykových lokalizací. Soubory musely být umístěny do složky `src` do zvláštního balíku. V předchozím bloku kódu vidíme rovněž nastavení implicitního skinu na `blues`.

Poslední blok konfigurace, který stojí za zmínku, je nastavení takzvaného `viewResolver`. Konstrukcí beany s tímto jménem říkáme Spring frameworku, že pro vrstvu uživatelského rozhraní chceme použít právě tuto třídu určenou parametrem `class`.

Společně s nastavením konfigurace Apache Tiles 2 tvoří tento blok konfigurace kompletní nastavení vrstvy uživatelského rozhraní. Konkrétní nastavení pro naší aplikaci vidíme v dalším bloku kódu.

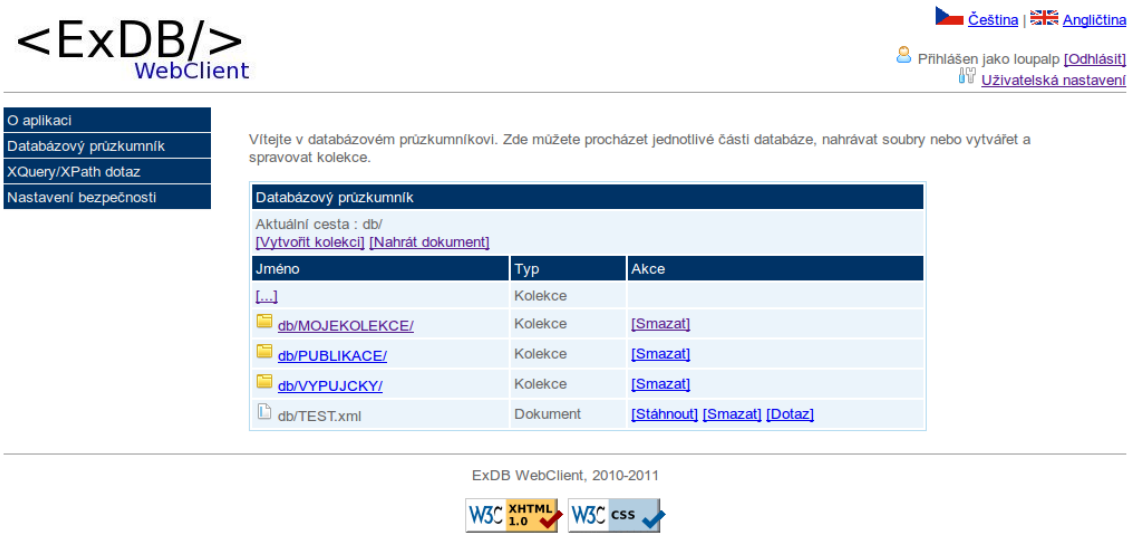
```
<bean id="viewResolver"
  class="org.springframework.web.servlet.view.UrlBasedViewResolver">
        <property name="viewClass"
          value="org.springframework.web.servlet.view.tiles2.TilesView" />
</bean>

<bean id="tilesConfigurer"
  class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
        <property name="definitions">
                <list>
                        <value>/WEB-INF/tiles/*.xml</value>
                </list>
        </property>
</bean>
```

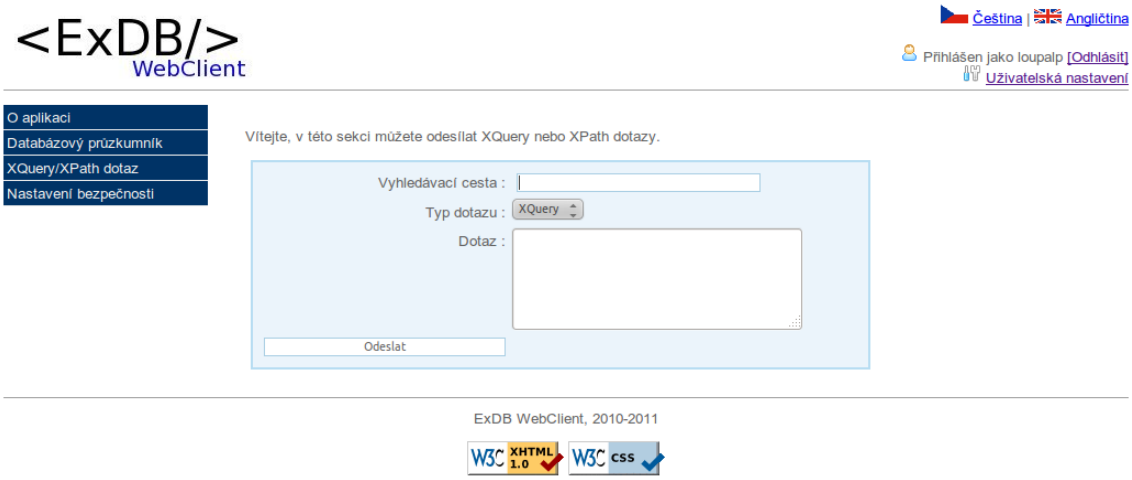
V konfiguraci beany `tilesConfigurer` můžeme vidět nastavení hledací cesty pro definice Apache Tiles 2.

4.4.3 Náhledy aplikace

Obrázek 4.2: Uvítací stránka

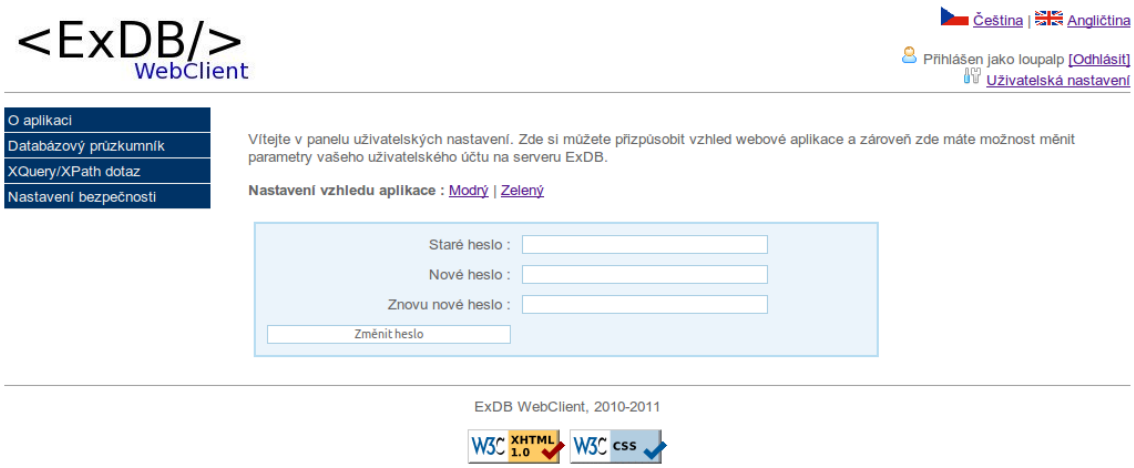


Obrázek 4.3: Databázový průzkumník

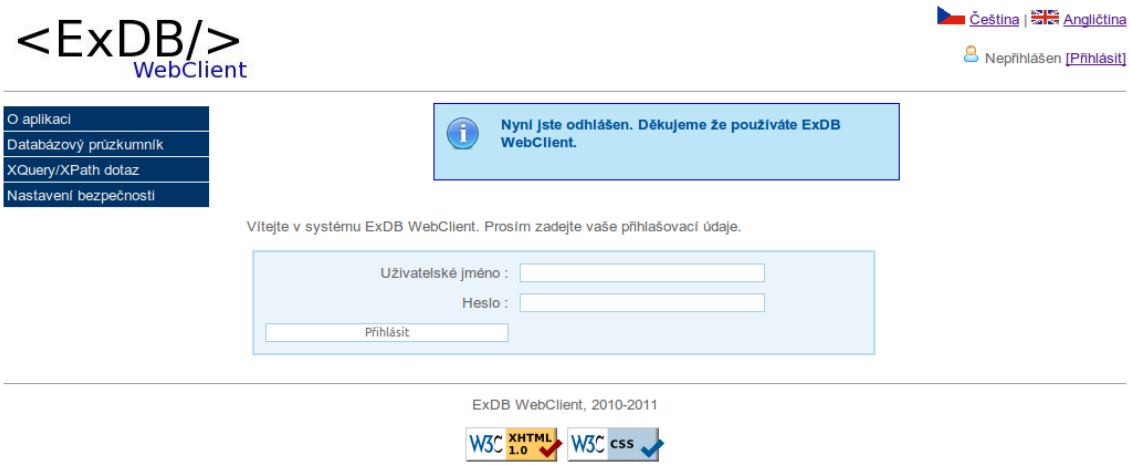


Obrázek 4.4: Pomocník pro vyhledávání

Obrázek 4.5: Nastavení bezpečnosti



Obrázek 4.6: Uživatelský panel



Obrázek 4.7: Přihlašovací stránka

Kapitola 5

Testování

5.1 Testování spolupráce s ExDB serverem

Vyvíjená aplikace byla testována vůči aktuální SVN verzi serveru ExDB. Během implementace webové aplikace se objevila řada problémů na straně serveru, které byly popisovány v sekci 4.1. Na CD, které je součástí této práce, je umístěna zkompilovaná funkční verze, se kterou byl webový klient testován. Výsledky testů dopadly dobře. Klient byl schopen komunikovat se serverem a bylo v něm možné provádět všechny operace, které byly zadány v požadavcích na funkcionalitu aplikace.

5.2 Testování funkčnosti webové aplikace

Samotná webová aplikace byla testována na aplikačním serveru GlassFish verze 3 na operačním systému GNU/Linux s Java Virtual Machine verze 1.6.0. Na straně klienta byla aplikace testována na systémech Windows 7 a GNU/Linux s použitím prohlížečů Mozilla Firefox verze 4 a Internet Explorer verze 8.

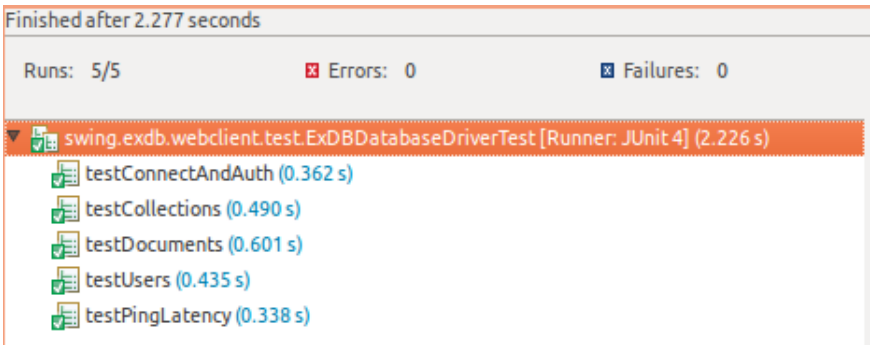
5.3 Validace XHTML a CSS

Uživatelské rozhraní webové aplikace bylo v průběhu vývoje testováno pomocí validátoru konsorcia W3C. Díky tomu je zajištěno, že veškerý XHTML a CSS obsah práce bude v souladu s platnými normami, které jsou pro tento jazyk závazné především kvůli kompatibilitě s různými webovými prohlížeči.

5.4 Unit Testy

Součástí práce jsou rovněž Unit Testy, které testují funkce databázového driveru ExDB. Při konfiguraci ExDB serveru s modulem `mod-storage` ve variantě `FileSystemStorage` vrátily Unit Testy pozitivní výsledek. Nová implementace modulu `mod-storage` s názvem `EnahncedStorage` bohužel zatím neprošla testy a je tedy pro tuto chvíli nepoužitelná. Varianta s implementací `NativeStorage` nebyla testována. Výsledek testů můžeme vidět na

obrázku 5.1. Testy jsou umístěny ve třídě `ExDBDatabaseDriverTest` v balíku `swing.exdb.webclient.test`. K testování bylo využito knihovny JUnit verze 4.



Obrázek 5.1: Unit Testy

Kapitola 6

Závěr

Cílem této práce bylo realizovat implementaci webové aplikace, která by sloužila jako klient pro databázový systém ExDB.

V rámci rešerše hotových řešení jsme se podívali na některé již existující produkty a zhodnotili jsme jejich funkce. Následně jsme si prošli analýzou požadavků a ve shodě se zadáním vedoucího práce jsme navrhli implementaci webové aplikace, která by v závěru měla splňovat všechny funkční požadavky.

Při samotné realizaci se však objevilo množství problémů, z nichž některé nakonec zabránili doslovnému splnění všech vytyčených cílů. Asi největším problémem byly nedostatky v implementaci samotného ExDB serveru, který ve své aktuální verzi postrádá řadu funkcí nebo jsou tyto funkce v nepoužitelném stavu. Některé nedostatky byly triviální a podařilo se je odstranit. Některé však vyžadovali hlubší zásah do zdrojových kódů ExDB, což zcela přesahovalo rozsah této práce.

S ohledem na předešlá fakta byla následně implementována výsledná webová aplikace, která by i přes zmíněné nedostatky měla sloužit jako užitečný nástroj pro práci se servery ExDB a měla by se stát plnohodnotnou součástí kompletního databázového řešení v rámci projektu ExDB.

Při vývoji aplikace byl samozřejmě brán ohled na zaběhnuté programovací praktiky a zvyklosti projektu ExDB. Především byl kladen důraz na modulární strukturu a snadnou rozšiřitelnost aplikace.

I přes všechny obstrukce, které nastaly během tvorby této bakalářské práce lze říci, že byly naplněny cíle dané zadáním a lze proto tuto práci považovat za úspěšnou.

Příloha A

Seznam použitých zkratek

CSS Cascading Style Sheets

DTD Document Type Definition

ExDB Experimental Database

GUI Graphic User Interface

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

J2EE Java Enterprise Edition

JDBC Java Database Connectivity

JSTL JavaServer Pages Standard Tag Library

MVC Model-View-Controller

NXD Native XML Database

PHP Hypertext Preprocessor

SGML Standard Generalized Markup Language

SVN SubVersion

SQL Structured Query Language

TCP/IP Transmission Control Protocol / Internet Protocol

URL Uniform Resource Locator

UTF-8 UCS Transformation Format

XACML Extensible Access Control Markup Language

XHTML Extensible Hypertext Markup Language

XML Extensible Markup Language

XSLT Extensible Stylesheet Language Transformation

Příloha B

Instalační a uživatelská příručka

B.1 Minimální systémové požadavky

B.1.1 Server

- Libovolný operační systém, na kterém běží Java Virtual Machine (testováno na GNU/Linux)
- Java Virtual Machine, verze 1.6 a vyšší
- Libovolný Java Servlet Container podporující Java Servlet verze 2.5 (testováno na aplikačním serveru GlassFish verze 3)
- Připojení k počítačové síti s adekvátní propustností vzhledem k počtu zamýšlených klientů
- Běžící databázový server ExDB dostupný skrze počítačovou síť

B.1.2 Klient

- Libovolný operační systém (testováno na GNU/Linux a Windows 7)
- Libovolný internetový prohlížeč (testováno na Mozilla Firefox verze 4 a Internet Explorer verze 8)
- Připojení k počítačové síti

B.2 Instalace webové aplikace

Samotná webová aplikace je dodávána jako hotový archiv WAR, který je možno přímo spustit ve většině známých Java Servlet kontejnerů. Přímý postup zavedení WAR archivu je nad rámec tohoto dokumentu a závisí na typu použitého softwaru. Aplikace ve svém implicitním nastavení hledá instanci běžícího ExDB serveru na adrese `127.0.0.1` na standardním portu `6124`. Pokud provozujete ExDB server na jiné adrese nebo portu, je třeba upravit nastavení v souboru `exdb-servlet.xml`, který je umístěn ve složce `WebContent/WEB-INF/`.

V tomto souboru nalezněte následující blok řádek a změňte hodnoty proměnných dle libosti :

```
<bean id="dbDriver" class="swing.exdb.webclient.core.ExDBDatabaseDriver">
        <property name="serverAddress"  value="127.0.0.1" />
        <property name="serverPort"  value="6124" />
</bean>
```

Ve zmíněném souboru lze rovněž nalézt některé další parametry nastavení, které však doporučujeme ponechat v původním znění. Pokud byste však přesto chtěli dolaďovat běh webové aplikace, doporučujeme věnovat pozornost následujícím konfiguračním parametrům.

- Parametr `maxCookieAge` ve vlastnostech bean objektů `themeResolver` a `localeResolver` určuje maximální dobu, po kterou jsou v klientském počítači udržovány informace o uživatelských nastaveních webové aplikace.
- Parametr `maxUploadSize` ve vlastnostech bean objektu `multipartResolver` určuje maximální velikost XML dokumentu, který je možno nahrát do aplikace.

B.3 Instalace databázového serveru ExDB

Databázový server ExDB v tuto chvíli nemá žádný typ instalačního programu. Nicméně instalace ExDB není nijak složitá. Na přiloženém CD nosiči lze nalézt složku s názvem `exdb-server`. Tu je potřeba zkopírovat na disk serveru, na kterém má ExDB běžet. Následně spustíte hlavní proces ExDB serveru použitím jednoho ze spouštěcích skriptů, které se nachází v kořenové složce serveru. Uživatelé systému Windows využijí dávkového skriptu `exdb-server.bat` a uživatelé systému GNU/Linux využijí skriptu `exdb-server.sh`.

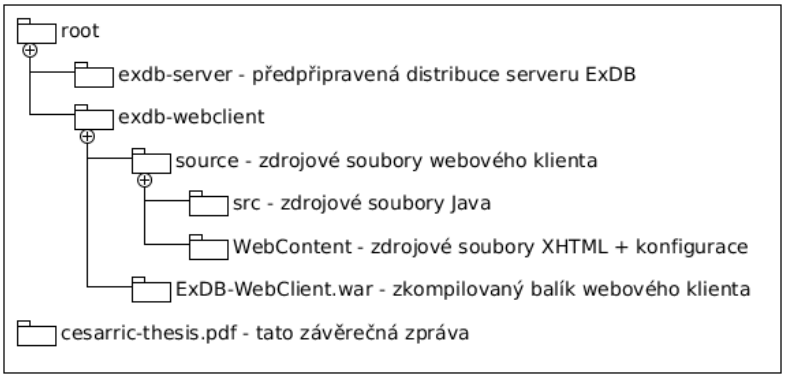
Podmínkou pro úspěšné spuštění je přítomnost Java Virtual Machine. Server musí být také schopen zapisovat do svých vlastních složek, což je otázka nastavení práv. Problematika nastavení práv je nad rámec této práce a záleží na použitém operačním systému.

K serveru ExDB se lze po instalaci přihlásit uživatelským jménem `test` a heslem `test`.

Příloha C

Obsah přiloženého CD

Na nosiči CD, jenž je pevnou součástí této práce, naleznete všechny soubory potřené k otestování zde popisované aplikace, která je produktem této práce. Na CD najdete webového klienta, funkční distribuci ExDB serveru, zdrojové kódy a kopii této závěrečné zprávy. Obrázek C.1 popisuje strukturu CD.



Obrázek C.1: Struktura CD nosiče

Literatura

[1] BRAY, Tim, et al. Extensible Markup Language (XML) [online]. W3C Recommendation 26 November 2008. MIT, ERCIM, Keio : 2008-11-26 [cit. 2011-05-26]. Extensible Markup Language (XML). Dostupné z WWW: <http://www.w3.org/TR/xml/>.

[2] MarkLogic Corporation. MarkLogic - MarkLogic Server [online]. 2011 [cit. 2011-05-26]. MarkLogic Server. Dostupné z WWW: <http://www.marklogic.com/products/marklogic-server.html>.

[3] EXist-db Open Source Native XML Database [online]. 2011, 2011-04-29 [cit. 2011-05-26]. EXist-db Open Source Native XML Database. Dostupné z WWW: <http://exist.sourceforge.net/>.

[4] Oracle Corporation. Oracle9i Administrator [online]. Release 2 (9.2.0.1.0) for UNIX Systems. 2002 [cit. 2011-05-26]. Administering SQL*Plus and iSQL*Plus . Dostupné z WWW: <http://download.oracle.com/docs/html/A97297_01/toc.htm>.

[5] The Eclipse Foundation. The Eclipse Foundation open source community website [online]. 2011. 2011 [cit. 2011-05-26]. Eclipse. Dostupné z WWW: <http://www.eclipse.org/>.

[6] PhpMyAdmin [online]. 2011. 2011 [cit. 2011-05-26]. PhpMyAdmin. Dostupné z WWW: <http://www.phpmyadmin.net>.

[7] The Apache Software Foundation : Tutorial [online]. 2010. 2010 [cit. 2011-05-26]. Apache Tiles Framework. Dostupné z WWW: <http://tiles.apache.org/framework/tutorial/index.html>.

[8] LOUPAL, Pavel. ExDB Homepage [online]. ČVUT v Praze : 2010/11/24, 2010/11/24 [cit. 2011-05-26]. Postup kompilace/sestavení ExDB. Dostupné z WWW: <http://exdb.fit.cvut.cz/doku.php?id=dev:build>.

[9] JAROŠ, Josef. Prohlížení objektů v databázi Oracle s podporou exportu. ČVUT v Praze, 2007. 51 s. Bakalářská práce. České vysoké učení technické v Praze.

[10] JOHNSON, Rod, et al. Spring Java Application Framework : Reference Documentation [online]. 3. [s.l.] : [s.n.], 2009 [cit. 2011-05-26]. Dostupné z WWW: <http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/pdf/spring-framework-reference.pdf>.

[11] TRUKSA, Jan. Klient pro nativní XML databázový systém. ČVUT v Praze, 2010. 39 s. Bakalářská práce. České vysoké učení technické v Praze.

[12] HTTP cookie. In Wikipedia : the free encyclopedia [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2008-11-26, last modified on 2011-04-29 [cit. 2011-05-26]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/HTTP_cookie>.