VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

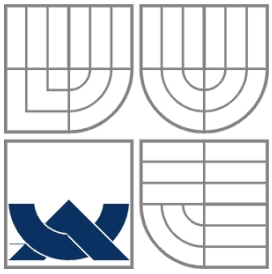
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VIACJAZYČNÝ FAKTURAČNÝ SYSTÉM MALEJ FIRMY

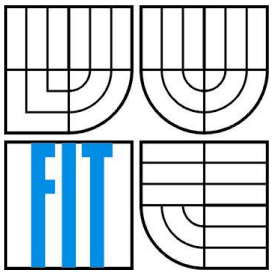
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE IVAN BONDRA
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VÍCEJAZYČNÝ FAKTURAČNÍ SYSTÉM MALÉ FIRMY

MULTILANGUAGE INVOICE SYSTEM OF SMALL COMPANY

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — IVAN BONDRA
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR JAŠA
SUPERVISOR

BRNO 2009

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2008/2009

Zadání bakalářské práce

Řešitel: **Bondra Ivan**
Obor: Informační technologie
Téma: **Vícejazyčný fakturační systém malé firmy**
Kategorie: Databáze

Pokyny:

1. Seznamte se s principy vývoje aplikací na platformě J2EE s využitím technologií pro objektově-relační mapování.
2. Seznamte se s oblastí účetnictví, zejména fakturací, a odpovídající legislativou
3. Po dohodě s vedoucím specifikujte požadavky na systém umožnující správu skladů, objednávek a faktur. Pokuste se proces fakturace co nejvíce automatizovat s možnostmi nastavení běžně se měnících hodnot (DPH, apod.).
4. Prostudujte možnost importů faktur do nějakého komerčního účetního programu.
5. Navržený systém implementujte včetně alespoň jednoho typu importu.
6. Zhodnoťte výsledné řešení a porovnejte jej s požadavky na systém. Navrhněte možné úpravy a rozšření systému.

Literatura:

- Pavel Herout: *Učebnice jazyka Java*, ISBN 80-7232-115-3, Kopp, 2005
- Ivor Horton: *Java 5*, ISBN 80-86330-12-5, Neocortex (WROX), 2005
- Java 2 Platform, Enterprise Edition (J2EE). Dokumentace dostupná na adrese http://java.sun.com/j2ee/.
- Kevin Mukhar, kolektiv autorů: *Beginning Java EE 5. from novice to professional*, ISBN 1-59059-470-3, Apress, 2006
- Mike Keith, Merrick Schincariol: *Pro EJB 3: Java persistence API*, ISBN 1-59059-645-5, Apress, 2006

Při obhajobě semestrální části projektu je požadováno:

- Bez požadavků.

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: **Jaša Petr, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2008
Datum odevzdání: 20. května 2009

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

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Cieľom tejto bakalárskej práce ja navrhnúť a implementovať viacjazyčný fakturačný systém určený pre malé firmy alebo živnostníkov. Systém by mal umožňovať efektívnu evidenciu fakturačných dokladov s nadväznosťou na skladové karty a zoznam partnerských firiem. Použiteľnosť systému by mal podporiť import fakturačných dokladov z iných, dnes bežne používaných, účtovných systémov. Tato práca popisuje jednotlivé technológie použité pri vývoji tohto systému a veľká časť dokumentu sa venuje analýze, návrhu a implementácii celého riešenia.

Klíčová slova

Fakturačný systém, databáza, Java, J2EE, Hibernate, Struts, MVC, JSP, XML, HTML

Abstract

The purpose of this bachelor's thesis is to design and implement multilanguage invoice system of small company or small trader. The system should enable an effective record of billing documents to be bound to the storage card and a list of partner companies. The applicability of the system should support the import invoices from other commonly used accounting systems. This documentation describes the various technologies used in developing this system and a large part of the document is devoted to analysis, design and implementation of the solution.

Keywords

Multilanguage invoice system, database, Java, J2EE, Hibernate, Struts, MVC, JSP, XML, HTML

Citace

Bondra Ivan: Viacjazyčný fakturačný systém malej firmy, bakalárska práca, FIT VUT v Brně, 2009

Viacjazyčný Fakturačný systém malej firmy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Petra Jaši.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Ivan Bondra
18.5.2009

Poděkování

Chcem sa poďakovať Ing. Petrovi Jašovi za dôsledné vedenie a konzultácie, ktoré mi poskytol počas písania bakalárskej práce.

Obsah

Úvod

S rozvojom trhovej ekonomiky, ktorá dnes umožňuje voľné obchodovanie a slobodnú trhovú súťaž, vznikli subjekty, ktorých činnosťou je nákup a predaj služieb alebo tovaru. Tento proces je potrebné efektívne evidovať. S veľkým pokrokom informačných technológií a ich prístupnosťou širokým masám, je táto evidencia prakticky plne v réžii počítačových účtovných programov.

Cieľom tejto bakalárskej prace je navrhnúť jednoduchý fakturačný systém, ktorý by slúžil malej firme alebo živnostníkovi. Celý proces fakturácie by mal byť čo najjednoduchší a mal by nadväzovať na správu skladu a v ňom uložených skladových položiek. Zmena jazyka zobrazeného rozhrania a možnosť importu fakturačných dokladov do jedného z komerčne používaných účtovných programov, by mala rozšíriť funkčnosť programu a podporiť jeho použiteľnosť.

Pre vývoj desktopových aj webových aplikácií existuje veľké množstvo programovacích jazykov, frameworkov, techník a návrhových vzorov, ktoré nám pomáhajú pri vývoji aplikácie. V prvých troch kapitolách je popis technológií, ktorá sú použité pri vývoji bakalárskej práce. Zameriam sa na popis techník prístupu k dátam a predstavím technológiu Hibernate, ktorá slúži na ich perzistenciu. Z pohľadu stavby systému uvádzam návrhový vzor MVC a aplikačný framework Apache Struts. Záver teoretickej časti bude venovaný popisu ostatných technológií, ktoré sú použité pri vývoji uvedenej aplikácie.

Ďalšia časť práce je venovaná návrhu a implantácii fakturačného systému. V štvrtej kapitole je bližšie charakterizovaný návrh systému, zhrnutie hlavných cieľov aplikácie, analýza a zber požiadaviek. Pre dokreslenie celej problematiky uvádzam diagram prípadov použitia a popis jednotlivých akcií. Celá aplikácia je následne rozdelená do troch nezávislých vrstiev podľa návrhového vzoru MVC. Pomocou diagramu perzistentných tried a jeho popisu je podrobne prestavená vrstva Model. Piata kapitola je zameraná na popis implementácie celého fakturačného systému. V už rozdelenej aplikácii vo vrstve Model uvádzam popis konfigurácie a použitie technológie Hibernate. Pri vrstve Controller rovnakým spôsobom popisujem aplikačný framework Apache Struts. Pre úplnosť celého popisu uvádzam vrstva View.

V záverečnej kapitole je zhrnutie celej bakalárskej práce a popis mojich skúsenosti s priebehom jej vytvárania.

1 Java

Výber vhodných programovacích prostriedkov nemá vplyv len na implementáciu projektu, ale vytvára aj predpoklady pre jeho budúce nasadenie a správu. Základom nami vytvorenej aplikácie je technológia Java, ktorá zahŕňa programovací jazyk i platformu. Vďaka jej všestrannosti a výhodným vlastnostiam si vyslúžila veľkú obľubu a široké využitie.

Technológia bola vytvorená spoločnosťou Sun Microsystems a v súčasnosti je voľne dostupná vo verzii 1.6 s označením Mustang, ktorá vyšla koncom roka 2006. Java je vhodná pre vývoj širokej palety aplikácií a je distribuovaná v troch platformách. Platforma Java Micro Edition (Java ME) je určená pre vývoj aplikácií prenosných a vstavaných zariadení. Pre vývoj desktopových aplikácií slúži platforma Java Standard Edition (Java SE). Platforma Java Enterprise Edition (Java EE) je nadstavbou Javy SE a je nasadzovaná pre vývoj distribuovaných aplikácií s viacvrstvovou architektúrou, bežiacich na aplikačných serveroch [1], [2] a [3].

1.1 Základné vlastnosti jazyka

Jednoduchosť

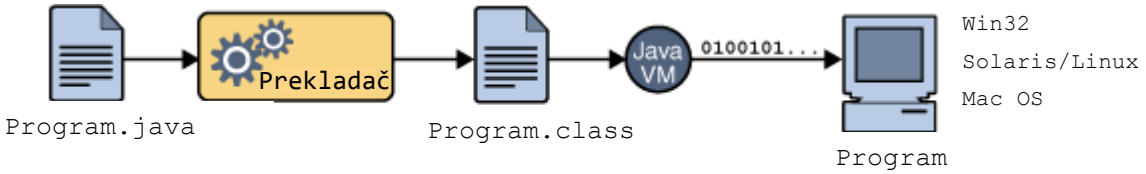
Syntax vychádza z jazyka C a C++, je však jednoduchšia a zrozumiteľnejšia. Java neobsahuje makrá ani direktíva. Nie je povolené preťažovanie operátora a je zavedený jednotný zápis pre prístup k objektom a knižniciam.

Objektovo orientovaný jazyk

Java je plne objektovo orientovaný jazyk. Výnimku tvoria iba primitívne dátové typy.

Platformová nezávislosť

Nezávislosť na operačnom systéme a hardwari počítača zaisťuje spôsob kompilácie. Zdrojové kódy s príponou `.java` sú kompilované do súborov `.class`, ktoré obsahujú bajtový kód. Takto pred pripravený kód je zdrojovým kódom Java Virtual Machine (JVM), ktorý interpretuje bajtový kód na strojový kód daného procesora.



Obrázok 1.1: *Kompilácia a spustenie programu na viacerých platformách (prevzaté z [2])*

Robustnosť

Java neumožňuje niektoré programátorské konštrukcie, ktoré bývali príčinou častých chýb. Použitá je silná typová kontrola a mechanizmus výnimiek ,ktorý slúži k ošetreniu chybových a neočakávaných stavov. Pamäť je spravovaná pomocou Garbage collectoru, ktorý pravidelne uvoľňuje pamäť od objektov, na ktoré už nevedú žiadne odkazy.

1.2 Základné vlastnosti platformy

Platformu je možné popísať ako softwarové alebo hardwarové prostredie, ktorá vytvára základ pre spúšťanie programov. Väčšinu platforiem tvorí kombinácia oboch spomínaných prostredí.

V tomto ohľade je Java odlišná. Jej platformu tvorí iba softwarové prostredie, ktorá pracuje nad rôznymi hardwarovo závislými platformami. Tým sa program oddeľuje od základného hardwaru. Java, ako softwarová platforma, je tvorená týmito komponentmi:

- Java Virtual Machine (JVM)
- Aplikačné programové rozhranie (API)

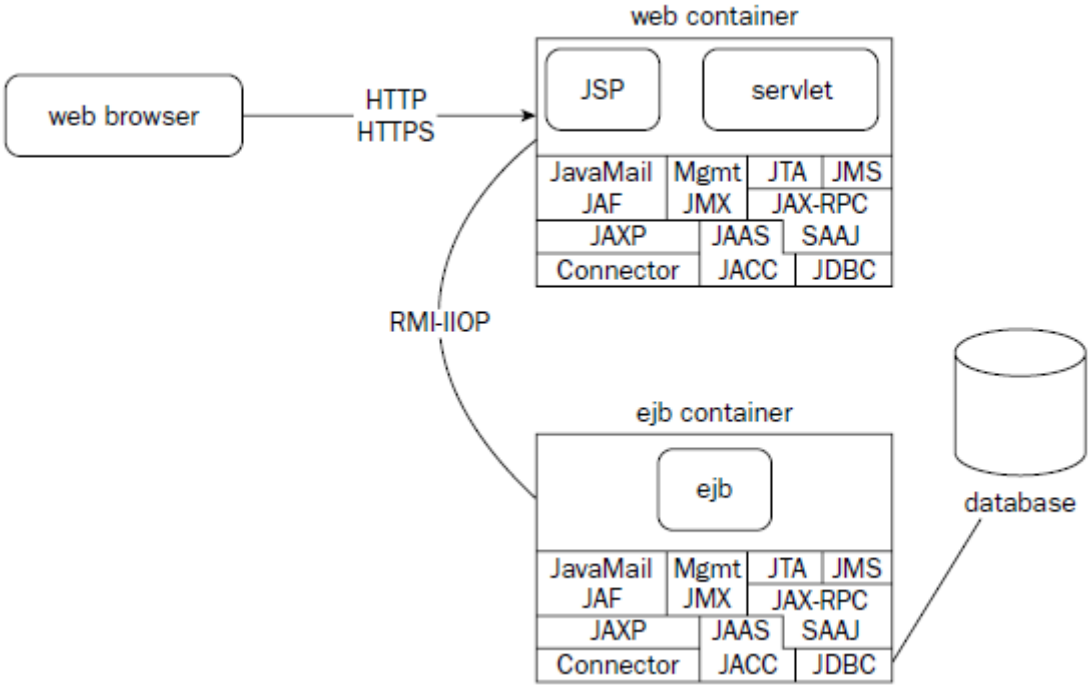
Z pohľadu hardwarovej nezávislosti sa zdá, že Bytcode poskytuje iba výhody. Interpretácia kódu však zapríčiňuje pomalší beh aplikácie. Tento problém sa pokúša riešiť Java HotSpot. Táto technika vyhľadáva úzke miesta z pohľadu výkonu a prekladá často používané časti do natívneho kódu procesora.

API tvorí rozsiahla skupina tried a rozhraní, ktoré sú organizované do balíčkov. Tie obsahujú základné funkcie programovacieho jazyka Java. Od prvej verzie 1.0, ktorá bola vydaná v januári 1996 prešlo API veľkými zmenami a rozšíreniami. Vďaka tomu je dnes platforma Java vhodná pre vývoj prakticky akejkoľvek aplikácie.

1.3 Java EE

Táto podkapitola je venovaná platforme Java Enterprise Edition, ktorá bola použitá pri vývoji uvedenej bakalárskej práce. Java EE rozširuje rozhranie Java SE o aplikačné komponenty a kontajnery, ktoré zjednodušujú vývoj komplexných viacvrstvových podnikových systémov. Pri vývoji sa programátor sústredí hlavne na riešenie business logiky a správu procesov prenecháva na kontajneroch postavených nad komponentmi Java EE. Celá platforma je koncipovaná tak, aby obsiahla širokú škálu použitia. To z Java EE vytvára jednu z najpoužívanejších a najobľúbenejších platforiem pre vývoj robustných serverových systémov.

Na nasledujúcom obrázku je znázornená architektúra Java EE web platformy spolu s vyobrazením jej komponentov.



Obrázok 1.2: *Architektúra J2EE web platformy (prevzaté z [3])*

Použitie väčšiny komponentov je na voľbe a potrebe programátora. Pri vývoji Java EE web aplikácie najčastejšie narazí na nasledujúce komponenty:

- **Servlet:** sú to programové komponenty bežiace na strane servera. Najčastejšie pracujú nad protokolom http, spracúvajú ich požiadavky a vytvárajú odpovede v tvare dynamických generovaných stránok.
- **Java Server Pages (JSP):** je možné si ich predstaviť ako klasické HTML stránky, ktoré obsahujú špeciálne elementy. Obsahom týchto elementov je kód v Jave, ktorý je prekladaný na servlet a následne kompilovaný. Tým sa vytvárajú dynamicky generované stránky. Na základe uvedeného JSP predstavujú akýsi opak servletov.
- **Enterprise Java Beans (EJB):** špecifikácia, ktorá pomocou sady návrhových vzorov pomáha pri vytváraní distribuovaných serverových systémov. Tieto návrhové vzory oddeľujú bussiness logiku od infraštruktúry systému.

Pre vývoj web aplikácií je najčastejšie používaná realizáciu pomocou tzv. tenkého klienta. K aplikácii užívateľ pristupuje pomocou webového prehliadača, ktorý zobrazuje informácie a slúži k prenosu užívateľských vstupov na aplikačný server. Ten vykonáva všetky business operácie. Rovnaký spôsob realizácie bol zvolený aj pri vývoji mojej bakalárskej práce.

2 Perzistencia dát

Pri práci s aplikáciou často potrebujeme pristupovať k trvalo uloženým dátam alebo naopak dáta natrvalo ukladať. Mnou navrhnutá aplikácia k dátam pristupuje a zároveň ich aj uchováva. Ukladanie objektov Javy poznáme pod pojmom perzistencia.

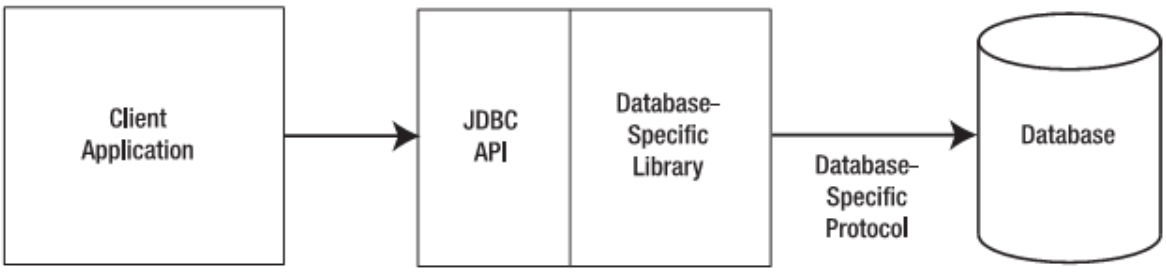
Existuje pomerne mnoho spôsobov perzistencie. Pre ukladanie malého množstva dát s jednoduchou logickou štruktúrou programátori často siahajú k návrhu vlastných formátov. Ak je potrebné zachovať prenositeľnosť súborov, je vhodné použiť formát XML. Pre uloženie zložitých a objemných dát je najvhodnejšia sofistikovaná databáza. Pre vývoj webových aplikácií je tento spôsob prakticky jediný použiteľný. Z pohľadu typu databázy je najčastejšie používaná relačná databáza.

Pri použití objektovo orientovaného jazyka a relačnej databázy nastáva problém známy pod pojmom *impedance mismatch*, ktorý môžeme voľne preložiť ako nezhoda pamäte. Relačné databázy obsahujú uložené procedúry, tabuľky, riadky a kľúče. Na druhej strane, objektovo orientovaná Java obsahuje objekty, komplexné vzťahy a dedičnosť. Tento nesúlad rieši objektovo-relačné mapovanie (ORM). Jeho podstatou je prekladanie objektovo orientovaných dát na relačné dáta. Pre tento účel boli vyvinuté viaceré frameworky a nástroje. Ich použitím odtienime dátový model na úrovni databázy a môžeme sa plne sústrediť na business logiku aplikácie.

Z pohľadu programovacieho jazyka Java základne API pre unifikovaný prístup k databázam poskytuje Java Database Connectivity (JDBC). Pre ORM je možné použiť štandardné aplikačné rozhranie Java Data Objects (JDO) alebo niektorý z frameworkov. Jedným z najpožívanejších je Hibernate. Je jednoduchý, robustný a má veľké množstvo dostupnej dokumentácie. Z komerčných frameworkov pre ORM spomeniem TopLink alebo CocoBase. V nasledujúcich podkapitolách bližšie popišem unifikované JDBC API a ORM framework Hibernate.

2.1 JDBC

JDBC API je tvorené rozhraním a knižnicami pre komunikáciu so systémami na ukladanie dát. Väčšina JDBC aplikácií sa používa pre komunikáciu s relačnými databázami. Pre vytvorenie prístupu k týmto databázam sa využíva JDBC API a ovládač pre databázu. Tým sa docieli jednotný spôsob práce. Ako ovládač je možné použiť JDBC ovládač, ktorý sprostredkúva prácu s inými ODBC alebo ovládač pre konkrétnu databázu. Takéto ovládače sú vyvíjané samotnými výrobcami databáz a sú optimalizované.



Obrázok 2.1: *Spôsob práce s databázou (prevzaté z [4])*

Prístup pomocou JDBC je nenáročný na implementáciu. Nevýhodou je však nutná zmena SQL kódu, pri prechode na inú databázu alebo pri zmene tabuliek a ich záznamov. Tato činnosť je veľmi pracná a neprehľadná. Riešením je vytvorenie ďalšej vrstvy aplikácie, ktorej úlohou bude odtienenie procesu získavania dát.

2.1.1 Typy ovládačov JDBC

- **Most JDBC – ODBC:** Poskytuje prístup cez API ODBC
- **Java a natívny ovládač):** Kód Javy volá ovládač databáze nainštalovaný na klientskej stanici
- **Java a aplikačné programové prostriedky (Middleware):** Java sa spojí so špecializovaným serverom (označuje sa ako Middleware), ktorý sprostredkuje spojenie s databázou alebo viacerými databázami
- **Čistá Java:** Ovládač v Jave sa priamo spojí s databázou (môže byť aj vzdialená). Nie je potrebná žiadna medzivrstva

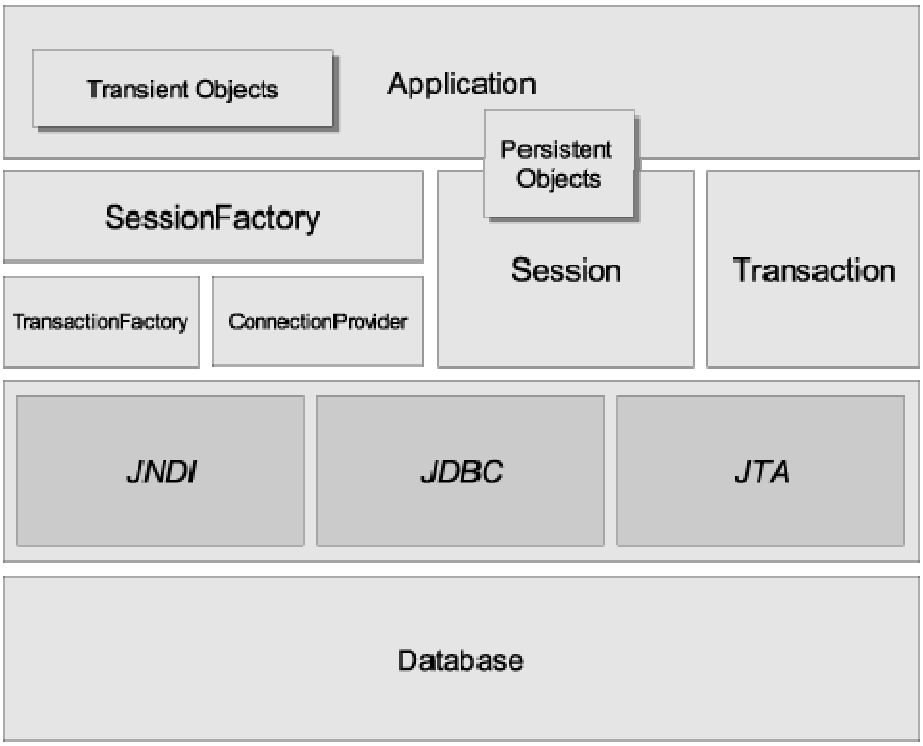
2.2 Hibernate

Hibernate [5], [6], [7] je nástroj pre ORM v prostredí Java. Okrem samotnej techniky perzistencie umožňuje získavanie databázových dát pomocou vlastného plne objektovo orientovaného jazyka nazvaného Hibernate Query Language (HQL). Tento vo veľkej miere uľahčuje a urýchľuje prácu s relačnými dátami a je zároveň syntakticky veľmi podobný jazyku SQL. Podporované je veľké množstvo relačných databáz. Jedná sa napríklad o databázu Oracle 9/10g, MySQL, PostgreSQL, Sybase, SAP DB, Microsoft SQL Server.

Jednoduchosť implementácie perzistencie, podobnosť HQL relačnému SQL jazyku a široká podpora relačných databáz vytvárajú z Hibernate univerzálny nastroj. Hibernate nachádza využitie v klasických desktopových aplikáciách i aplikáciách založených na J2EE.

2.2.1 Architektúra

Architektúru Hibernate je možné začleniť do aplikácie vo viacerých úrovniach. Pri použití tzv. odľahčeného prístupu je API Hibernate využívané len minimálne a aplikácia sama spravuje a vytvára vlastné JDBC spojenia a transakcie. Pre úplné odtienenie aplikácie od JDBC/JTA API je najvhodnejšie použiť tzv. Full-cream architektúru. Správu spojení a transakcií tak preberá Hibernate.



Obrázok 2.2 *Full-cream architektúra Hibernate (prevzaté z [5])*

Najpoužívanejšou časťou architektúry Hibernate je jednotka *Session*. Tá zapuzdruje JDBC spojenie a API pre perzistenciu. Ďalšou z jej mnohých úloh je vytváranie transakcií a udržiavanie cache prvej úrovne. Objekty session sú vytárané v priebehu aplikácie.

Jednotka *SessionFactory* inicializuje samotné Hiberante. Nastavuje pravidlá mapovania a voliteľne môže slúžiť ako cache druhej úrovne.

Jednotka *Transaction* odtieňuje aplikačný kód od implementácie JDBC alebo JTA transakcií na nižšej úrovni. Túto jednotku je odporúčané použiť, lebo pomáha vytvoriť väčšiu prenositeľnosť aplikácie. Samotné transakcie vytvára *TransactionFactory*.

Perzistnent Objects, sú objekty, ktoré sú priamo zviazane s aktuálnou *Session* a reprezentujú konkrétne dáta v databáze. *Transient Object* sú naopak objekty, ktoré nie sú zviazané s aktuálnou *Session,* a tak ich zmena nevyvolá zmenu v databáze.

Ďalšou časťou je *ConnectionProvider*, ktorý slúži na vytvorenie JDBC spojenia a *TransactionFactory*, ktoré slúžia na vytváranie *Transaction* objektov.

2.2.2 Konfigurácia

Pred nasadením vyžaduje Hibernate sadu nastavení, ktoré špecifikujú prostredie a zdroj popisujúci mapovanie tried. Tieto nastavenia je možné vykonať viacerými spôsobmi. Najmenej vhodným a zriedka používaným spôsobom je konfigurácia priamo v kóde aplikácie. Vhodnejším spôsobom je použitie externého konfiguračného súboru. Je možné zvoliť štandardný Java properties súbor, nazvaný `hibernate.properties` alebo XML konfiguračný súbor s názvom `hibernate.cfg.xml`.

Pre vytvorenie väzby medzi triedami aplikácie a tabuľkami relačnej databázy je potrebné definovať konkrétne pravidlá mapovania. Tieto pravidlá je možne uvádzať priamo v kóde tried, a to pomocou Java 5 anotácii alebo vytvorením mapovacích XML súborov, ktoré sú zvyčajne vytvorené pre každú mapovanú triedu. Mapovací XML súbor má spravidla názov v tvare `nazovTriedy.hbm.xml`.

Mapované triedy je odporúčané vytvárať podľa modelu POJO (Plain Old Java Object). Takéto triedy obsahujú bezparametrický konštruktor a k atribútom triedy sa pristupuje pomocou metód get/set. Pre zvýšenie výkonu je vhodné definovať samostatný atribút, ktorý bude použitý ako primárny kľúč a pre konštruktor nadefinovať `public` modifikátor prístupu.

V mnou vytvorenej aplikácii som pre konfiguráciu Hibernate a mapovanie tried použil samostatné XML súbory. Pri zmene databázy je tento spôsob flexibilnejší a ma širšie možnosti nastavenia mapovania. Vytváranie takýchto súborom nám navyše uľahčuje veľké množstvo nástrojov. Popis a ukážky mapovania sú uvedené v kapitole 4., ktorá sa venuje samotnej implementácii mnou vytvorenej aplikácie.

2.2.3 Hibernate query Language

Dotazovací jazyk Hibernate query Language (HQL) je v mnohom podobný jazyku SQL. Hlavným rozdielom je objektová orientácia HQL. Príkazy sa v priebehu činnosti prekladajú do jazyka SQL. Výsledkom dotazovania je objekt alebo kolekcia objektov. HQL je databázovo nezávislý a jeho použitie zvyšuje prenosnosť aplikácie.

Pi dotazovanií je možné použiť aj Vyhľadávacie kritériá (Criteria API). Ich použitie má opodstatnenie všade tam, kde je potrebné vytvoriť dynamický dotaz. V ojedinelých prípadoch alebo pri optimalizácií dotazov je možné použiť SQL jazyk.

3 Ostatné technológie

Okrem uvedených technológií, ktoré uľahčujú a pomáhajú pri vývoji informačných systémov, existuje veľké množstvo iných a inak zameraných produktov. V tejto kapitole sa budeme venovať technológiám, ktoré som použil pri vývoji mojej bakalárskej práce.

3.1 Model View Controller

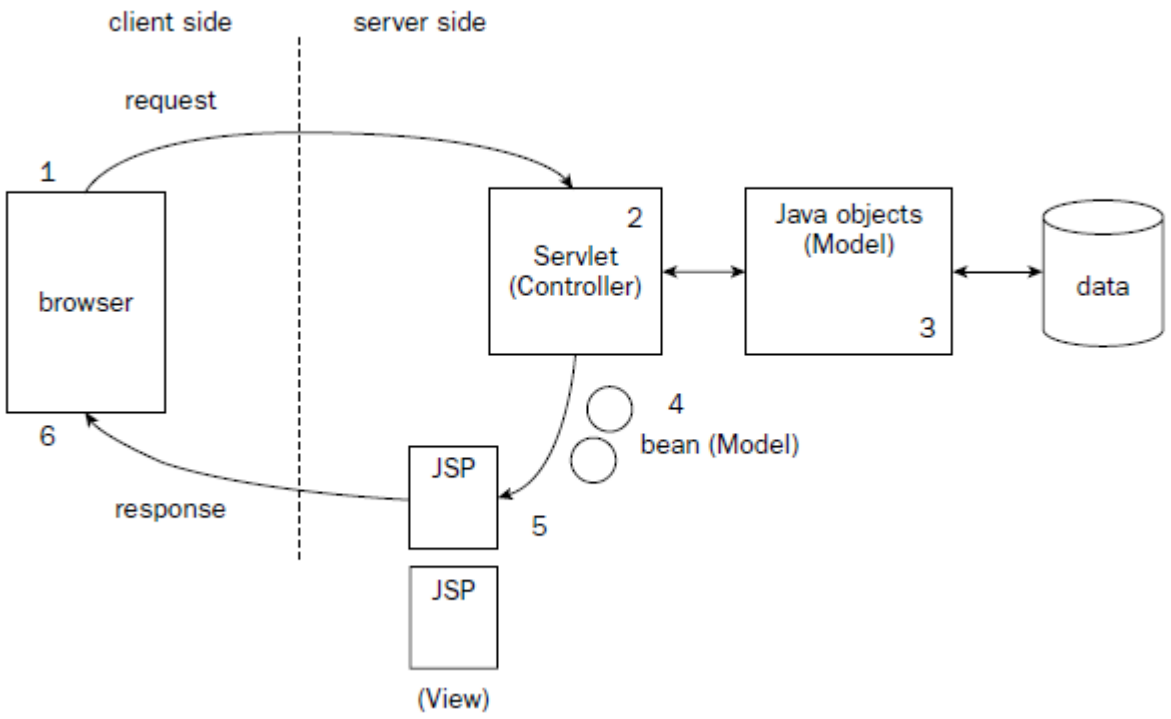
Model View Controller (MVC) je návrhový vzor softwarovej architektúry, ktorý rozdeľuje aplikáciu do troch nezávislých vrstiev. Týmito časťami sú:

Model – reprezentuje objekty z aplikačnej domény. Zjednodušene sú to dáta aplikácie.

View – zabezpečuje vizuálnu reprezentáciu vrstvy Model.

Controller – riadiaca logika aplikácie. Na základe udalosti vytvára Model a ten prezentuje pomocou vrstvy View.

Zmena vykonaná v jednej z vrstiev len minimálne ovplyvní ostatné vrstvy. Tato architektúra, konkrétne jej tzv. Model 2, ktorý je vyobrazený pod týmto textom, je veľmi obľúbená a práve na nej sa zakladá veľké množstvo frameworkov pre návrh Java EE web aplikácií. Z najznámejších je možné spomenúť Spring, Struts, Action Servlet, WebWork a iné. Pre vývoj aplikácie bakalárskej práce som zvolil framework Struts, ktorý je popísaný v nasledujúcej kapitole.



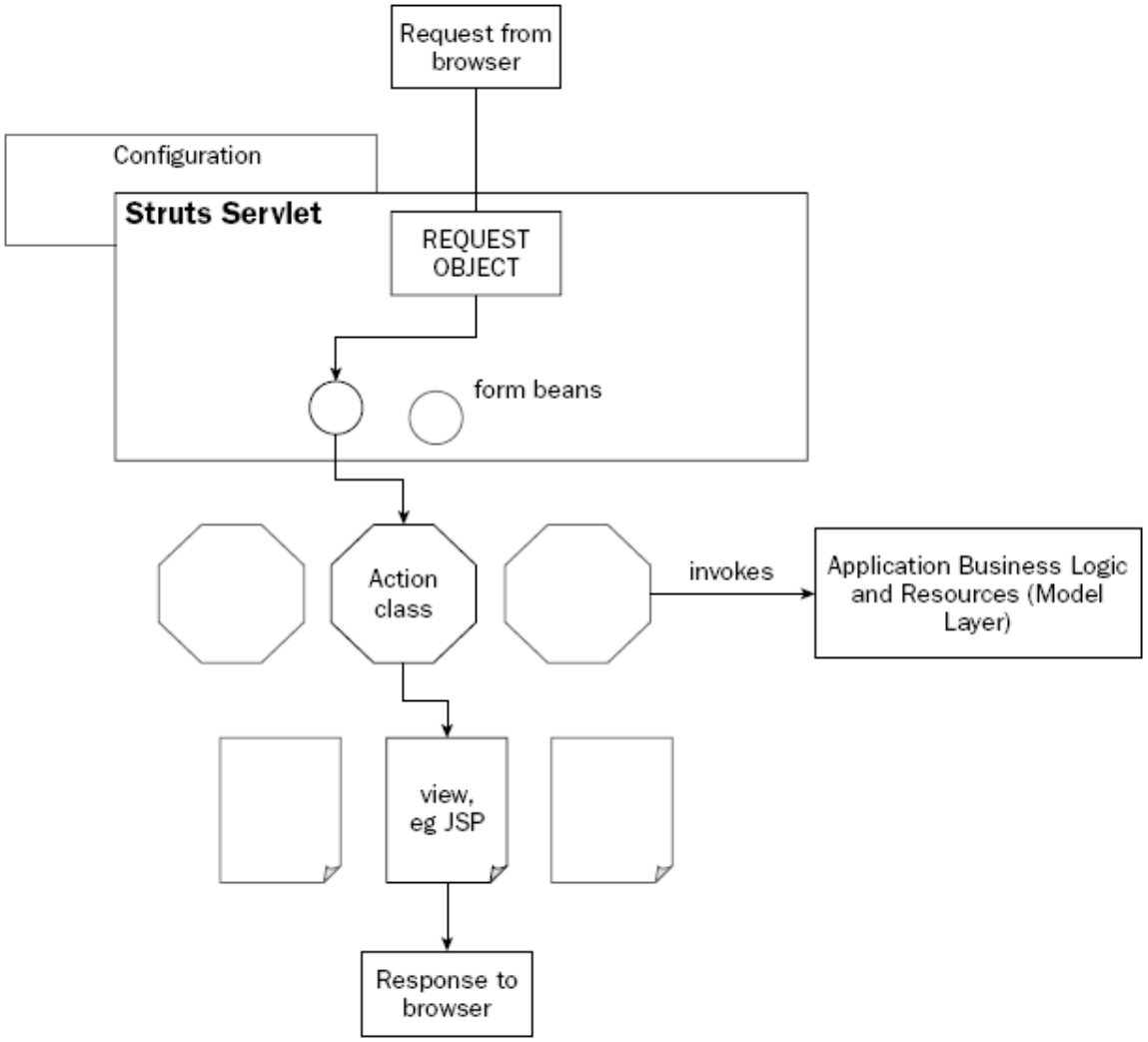
Obrázok 2.3 *MVC Model 2 architektúra (prevzaté z [3])*

3.2 Apache Struts

Struts [3], [9], [10] je aplikačný framework, ktorý nám pomáha pri vytváraní Java EE web aplikácií postavených na MVC Model 2 architektúre. Okrem iného rieši časté problémy pri vývoji ako je internacionalizácia, validácia vstupov, ošetrenie chybových stavov alebo šablónovanie. Poskytuje vlastnú knižnicu užívateľských Struts značiek, ktoré uľahčujú vytváranie JSP stránok. Struts je veľmi obľúbený a používaný aplikačný framework.

3.2.1 Architektúra

Architektúra pozostáva z Struts servletu, ktorý je konfigurovaný pre určité URL, ktoré aplikácia získa pri žiadosti web prehliadača. Tento servlet na základe konfigurácie prenechá riadenie vopred definovaným akciám. Akcie implementované programátorom reprezentujú bussines proces vyvolaný požiadavkami. Výsledkom akcií je vytvorenie vhodného výstupu pre vrstvu View, ktorý spravidla tvorí JSP stránka.



Obrázok 2.4 *Struts architektúra (prevzaté z [3])*

3.2.2 Konfigurácie

Z popisu architektúry vyplýva, že jednou z hlavých častí celého rámca je konfigurácia. Struts servlet je definovaný v `web.xml`. Konfigurácia akcií a ďalších súčastí je sústredená do jedného súboru s názvom `struts-config.xml`. Ten definuje nasledujúce voľby:

- **data sources:** definovanie zdroja dát, len v prípade priameho pripojenia pomocou JDBC
- **form beans:** definovanie názvov formulárov, tento názov je platný na JSP stránkach a v celom konfiguračnom súbore `struts-config.xml`
- **action mappings:** definovanie akcií v závislosti od požiadaviek
- **global forwards:** presmerovanie platné pre celú aplikáciu
- **controller:** nastavenie frameworku Struts, najmenej využívané
- **message resources:** definovanie properties súboru, obsahuje výpisy a chybové hlásenia
- **plug-in:** deklarácia rozširujúcich pluginov Struts; jedná sa napríklad o Tiles a Validator

Pri definovaní jednotlivých častí je potrebne dodržať poradie definícií v poradí, v ktorom boli uvedené vyššie. Ukážka konfigurácie vzorového súboru bude uvedená v kapitole venovanej implementácii aplikácie.

3.3 MySQL

Ako zdroj dát som pre uvedenú aplikáciu zvolil relačnú MySQL databázu. Tá vyniká svojou rýchlosťou, stabilitou, univerzálnosťou a jednoduchou správou. Popri PostrgeSQL je to jedna z najpoužívanejších voľne dostupných databáz v kombinácii s programovacím jazykom Java. V uvedenej aplikácii je použitá verzia MySQL 5.0, aktuálna verzia je 5.1. Zjednodušenie správy databázy mi umožnil balík nástrojov pod názvom MySQL GUI Tools

3.4 Apache Tomcat

Tomcat je voľne dostupný JSP/Servlet kontajner, ktorý používa ich referenčnú implementáciu. To mu zabezpečuje plnú podporu Servletov a JSP stránok. Ich chod je takto zaručený i na Java EE certifikovaných serveroch, a to na všetkých platformách. Apache Tomcat mimo iného obsahuje nástroje pre vývoj a nasadenie webových aplikácií a služieb. Vytvorená aplikácia bola vyvíjaná na Apache Tomcat verzie 6.0, ktorá implementuje špecifikáciu Servletov verzie 2.5 a špecifikáciu JSP verzie 2.1.

3.5 NetBeans IDE

Vývoj celej aplikácie som realizoval vo vývojovom prostredí NetBeans IDE. Je to voľne dostupné vývojové prostredie, ktoré vo veľkej miere uľahčuje a urýchľuje vývoj malých i veľkých projektov. Vyznačuje sa podrobnou kontrolou syntaxe kódu a veľkým množstvom nástrojov, ktorých počet je možné rozšíriť inštaláciou množstva pluginov. Toto prostredie značne uľahčilo použitie nástroja Hibernate a mapovanie tried určených pre perzistenciu. V súčasnosti je Netbeans IDE vhodným vývojovým prostredím, napríklad aj pre implementáciu PHP, Ruby, Ruby on Rails, C/C++ aplikácií. Pre implementáciu mojej aplikácie som použil verziu NetBeans IDE 6.5.

4 Návrh riešenia

V tejto kapitole sa budem venovať formulácii požiadaviek a návrhu riešenia pre mnou vyvinutý Fakturačný systém. Je možné, že táto aplikácia nenájde reálne uplatnenie, napriek tomu je vypracovaná podľa skutočne používaných postupoch pri fakturácii. Popri požiadavkách, ktoré sú kladené na systém z užívateľského hľadiska, bolo potrebne zabezpečiť i správnosť riešenia po legislatívnej stránke. Kvalitný a podrobný návrh je základom pre rýchlu a bezproblémovú implementáciu.

4.1 Hlavné ciele aplikácie

Aplikácia by mala plniť funkciu viacjazyčného fakturačného systému, ktorý by umožňoval efektívnu správu skladu a faktúr. Proces by mal byť v čo najväčšej miere automatizovaný a mal by spĺňať náležité legislatívne pravidlá. Systém by mal umožňovať export vybraných dokladov do jedného z komerčne používaných systémov. Viacjazyčnosť by mala užívateľovi umožniť zvoliť si jazyk fakturačného systému, čo by malo za následok generovanie prehľadov a dokladov v zvolenom jazyku.

4.2 Analýza a zber požiadaviek

Zdrojom informácií mi boli vo veľkej miere komerčné, firmami často používané, účtovné programy. Pri analýze a zbere požiadaviek som čiastočne využil aj vlastne skúsenosti. Osoba, s ktorou som konzultoval a validoval požiadavky, bol aktívny živnostník (otec). V procese fakturácie a náležitej legislatívy je znalý, lebo podstatnú časť fakturácie vykonáva sám.

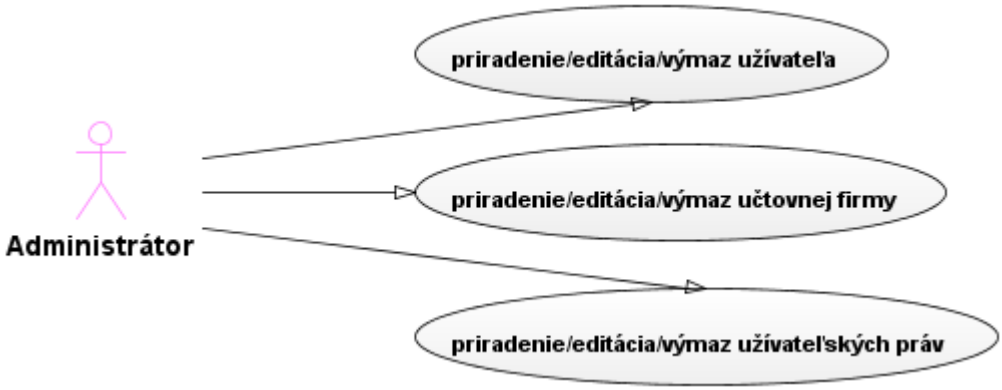
4.2.1 Druhy prístupu k aplikácii

Systém rozlišuje dva typy prístupu k aplikácii:

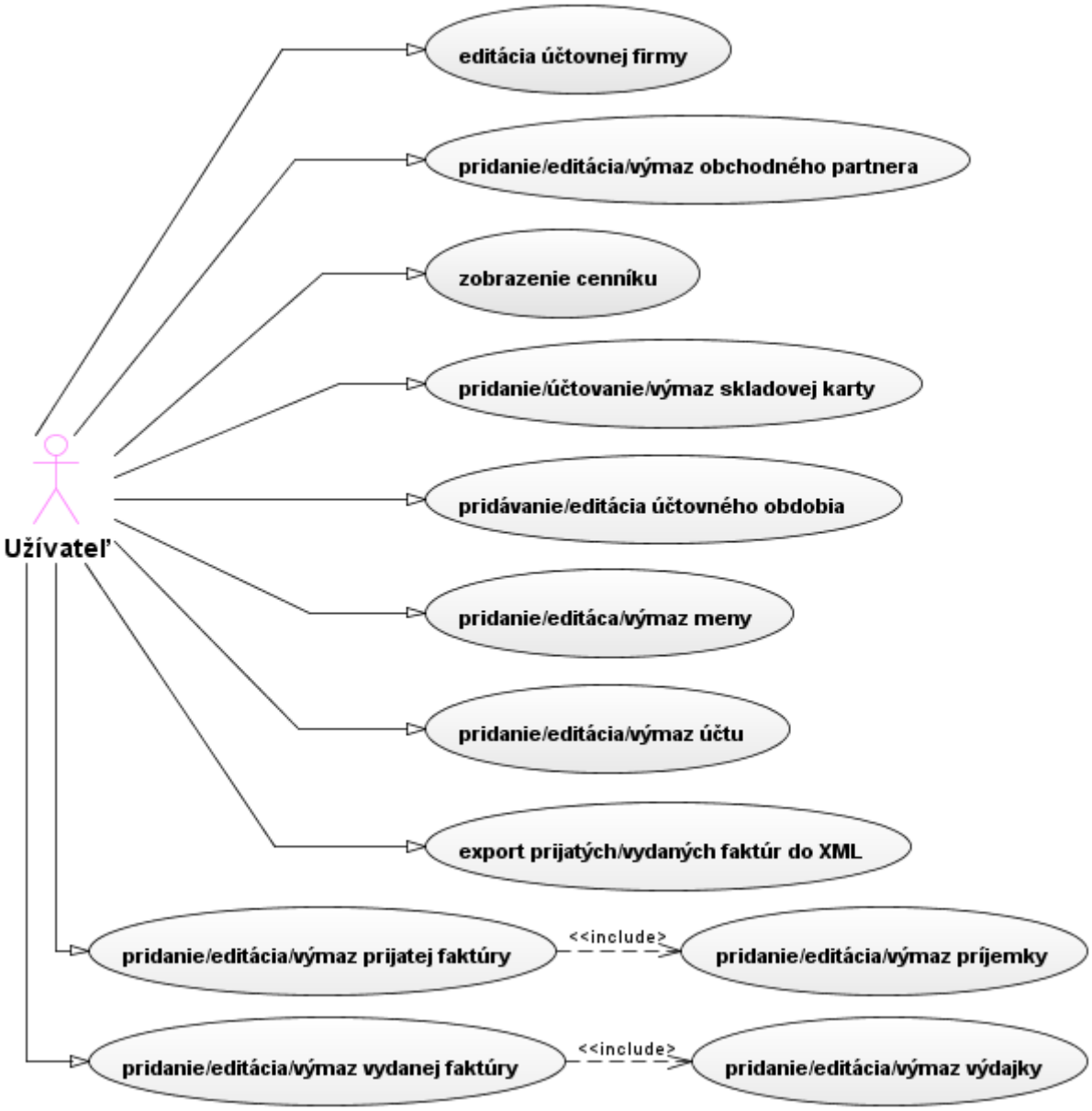
1) **Administrátor** – jeho úlohou je zabezpečovať správu užívateľov a účtovných firiem. K záznamom jednotlivých firiem nemá prístup.
2) **Užívateľ** – predstavuje fakturanta účtovnej firmy, ktorý má plnú správu nad údajmi.

4.2.2 Diagram prípadov použitia

V nasledujúcej kapitole je graficky zobrazený každý z typov prístupu k aplikácii. Pre toto zobrazenie som použil prehľadný diagram prípadov použitia. Pre ich vytvorenie som zvolil UML plugin vývojového prostredia NetBeans.



Obrázok 4.2 *Prípad použitia – Administrátor*



Obrázok 4.1 *Prípad použitia – užívateľ*

4.2.3 Špecifikácia prípadov použitia administrátorskej časti

Administrátorská časť aplikácie, ktorá je prístupná po prihlásení, slúži pre správu celého fakturačného sýtemu. V nasledujúcej podkapitole je uvedený popis jednotlivých prípadov použitia.

4.2.3.1 Pridanie/editácia/výmaz užívateľa

Užívateľ systému predstavuje fakturanta, ktorý vytvára fakturačné a iné doklady účtovnej firmy. Úlohou administrátora je správa týchto užívateľov. Pri ich vytváraní je vykonávaná kontrola jedinečnosti ich zadaných prihlasovacích mien a správneho tvaru zadaného hesla. Výmaz užívateľa nie je podmienený žiadnou záväznosťou na spravovanú firmu.

4.2.3.2 Pridanie/editácia/výmaz účtovnej firmy

Účtovná firma predstavuje subjekt správy fakturačného systému. Administrátor definuje jej základné identifikačné údaje. Ak nastane zmena v týchto údajoch, môže vykonať úpravu. V krajnom prípade môže účtovnú firmu z fakturačného systému úplne odstrániť. Tento úkon však vedie k odstráneniu všetkých nadväzujúcich dokladov a záznamov.

4.2.3.3 Pridanie/editácia/výmaz užívateľských práv

Každý z užívateľov môže byť účtovníkom viacerých účtovných firiem. Práva spravovať tieto firmy definuje administrátor. Okrem prideľovania práv môže administrátor práva odobrať alebo zmeniť.

4.2.4 Špecifikácia prípadov použitia užívateľskej časti

Užívateľská časť aplikácie je hlavnou časťou fakturačného systému a je prístupná len po prihlásení. Prihlásený užívateľ volí účtovnú firmu, pre ktorú bude vykonávať fakturáciu. Po tomto výbere má užívateľ prístupné voľby špecifikované v nasledujúcich podkapitolách.

4.2.4.1 Editácia účtovnej firmy

Umožňuje užívateľovi editovať vybrané atribúty o účtovnej firme. Názov a identifikačné číslo organizácie (IČO) sú nemenné. Všetky editovateľné atribúty sú pri zmene kontrolované na správnosť zadania tvaru. V prípade chyby je zobrazená výstražná stránka, ktorá bližšie popisuje chybu a jej nápravu.

4.2.4.2 Pridanie/editácia/výmaz obchodného partnera

Vychádzame z predpokladu, že každá účtovná firma obchoduje s viacerými partnermi. Týchto obchodných partnerov môžme rozdeliť na dve skupiny:

1) dodávatelia – skupina firiem, ktorá dodáva účtovnej firme tovar a služby
2) odberatelia – skupina firiem, ktorá odoberá od účtovnej firmy tovar a služby.

Užívateľovi je umožnené pridávať, editovať a odstraňovať takýchto obchodných partnerov z fakturačného systému. Jednotlivé atribúty pri vytváraní alebo editácii záznamov podliehajú kontrole zadaných vstupov. Výmaz obchodného partnera je podmienený neprítomnosťou nadväzujúcich dokladov v systéme. Ak sú takéto väzby vytvorené, výmaz nie je možné uskutočniť.

4.2.4.3 Zobrazenie cenníku

Cenník poskytuje užívateľovi kompletný prehľad o tovare a službách poskytovaných účtovnou firmou. Výpis položiek je formou tabuľky, ktorú je možné exportovať. Pre zvýšenie prehľadnosti a za účelom rýchlejšieho vyhľadávanie, je možné cenník zoradiť podľa nasledujúcich atribútov: kód/skratka, typ zásoby, názov, cena.

4.2.4.4 Pridanie/editácia/výmaz skladovej karty

Systém umožňuje užívateľovi spravovať skladové položky. Pridanie skladovej karty je späté s definovaním dodávateľskej firmy. Atribúty pri vytáraní a editácii sú kontrolované a ak ich užívateľ nevyplní korektne, zobrazí sa chybová stránka s popisom a nápovedou pre opravu. Najčastejšou zmenou v záznamoch skladovej karty je zmena stavu, ktorá definuje zásobu položky na sklade.

4.2.4.5 Pridávanie/editácia/ účtovného obdobia

Vystavenie alebo prijatie faktúry sa vzťahuje k určitému účtovnému obdobiu. Užívateľ má možnosť vytvoriť nové účtovné obdobie, prípadne zmeniť jeho platnosť. Zadávané dátumy platnosti účtovného obdobia podliehajú kontrole. Dátumy sa nesmú prekrývať s platnosťou iného účtovného obdobia.

4.2.4.6 Pridávanie/editácia/výmaz meny

Užívateľ má možnosť definovať, v akej mene bude vydaná alebo prijatá faktúra. Základnou menou systému je Euro (€). Výmaz meny je podmienený neprítomnosťou nadväzujúcich dokladov v systéme.

4.2.4.7 Pridávanie/editácia/výmaz účtu

Každá účtovná firma a obchodný partner by mali mať definovaný aspoň jeden bankový účet. Ten je potrebné uviesť na fakturačných dokladoch v prípade, že vyplatenie pohľadávok alebo záväzkov sa uskutoční formou bankového prevodu. Užívateľ pri vytváraní bankového účtu daného subjektu zadáva číslo účtu a vyberá z preddefinovaných bánk, ktoré sú reprezentované ich číselnou skratkou. Výmaz účtu je tak, ako aj v iných záznamoch, podmienený neprítomnosťou nadväzujúcich dokladov.

4.2.4.8 Export prijatej/vydanej faktúry do XML

Užívateľ má možnosť už vytvorený fakturačný doklad exportovať do XML súboru. Jeho formát je vytvorený podľa špecifikácie pre import fakturačných dokladov ekonomického systému Pohoda od spoločnosti Stormware.

4.2.4.9 Pridávanie/editácia/výmaz prijatej faktúry

Hlavnou náplňou prace užívateľa a účelom tejto aplikácie je vytváranie fakturačných dokladov. Pred výdajom tovaru zo skladu a vytvorením faktúry, je potrebne, aby boli skladové položky v systéme už zaevidované a aby boli v sklade prítomné v požadovanom množstve. Tento proces zaručuje zaevidovanie prijatej faktúry. Tá obsahuje tieto položky a vytvára doklad o zaevidovaní príjmu tovaru. Tento doklad má názov príjemka. Užívateľ má možnosť zmeniť už vytvorenú faktúru, prípadne, ak nastane chyba, doklad aj zmazať. Zadané údaje formulára pre vytváranie a editáciu faktúry sú kontrolované a v prípade chyby sa zobrazí chybová stránka.

4.2.4.10 Pridávanie/editácia/výmaz príjemky

Spolu s prijatou faktúrou užívateľ vytvára príjemku tovaru, ktorá zaeviduje príjem tovaru do skladu. V prípade zmeny má užívateľ možnosť tento doklad meniť. Ak dôjde k odstráneniu prijatej faktúry, automaticky dôjde aj k odstráneniu príjemky.

4.2.4.11 Pridávanie/editácia/výmaz vydanej faktúry

Aby mohol užívateľ vydať tovar zo skladu svojmu odberateľovi, je potrebné, aby vytvoril faktúru pre výdaj. Vydaná faktúra obsahuje skladové položky, ktoré vytvárajú doklad zvaný výdajka. Rovnako, ako u prijatej faktúry, má užívateľ možnosť zmeniť už vytvorenú faktúru, prípadne ju zmazať. Jednotlivé atribúty pri vytváraní a editácii faktúry sú kontrolované a, v prípade chyby, je zobrazená chybová stránka.

4.2.4.12 Pridávanie/editácia/výmaz výdajky

Vydaná faktúra automaticky generuje výdajku, ktorá obsahuje zoznam vydaného tovaru zo skladu. Zmenou položiek vo faktúre je možné zmeniť výdajku, výmazom faktúry automaticky dôjde k výmazu výdajky.

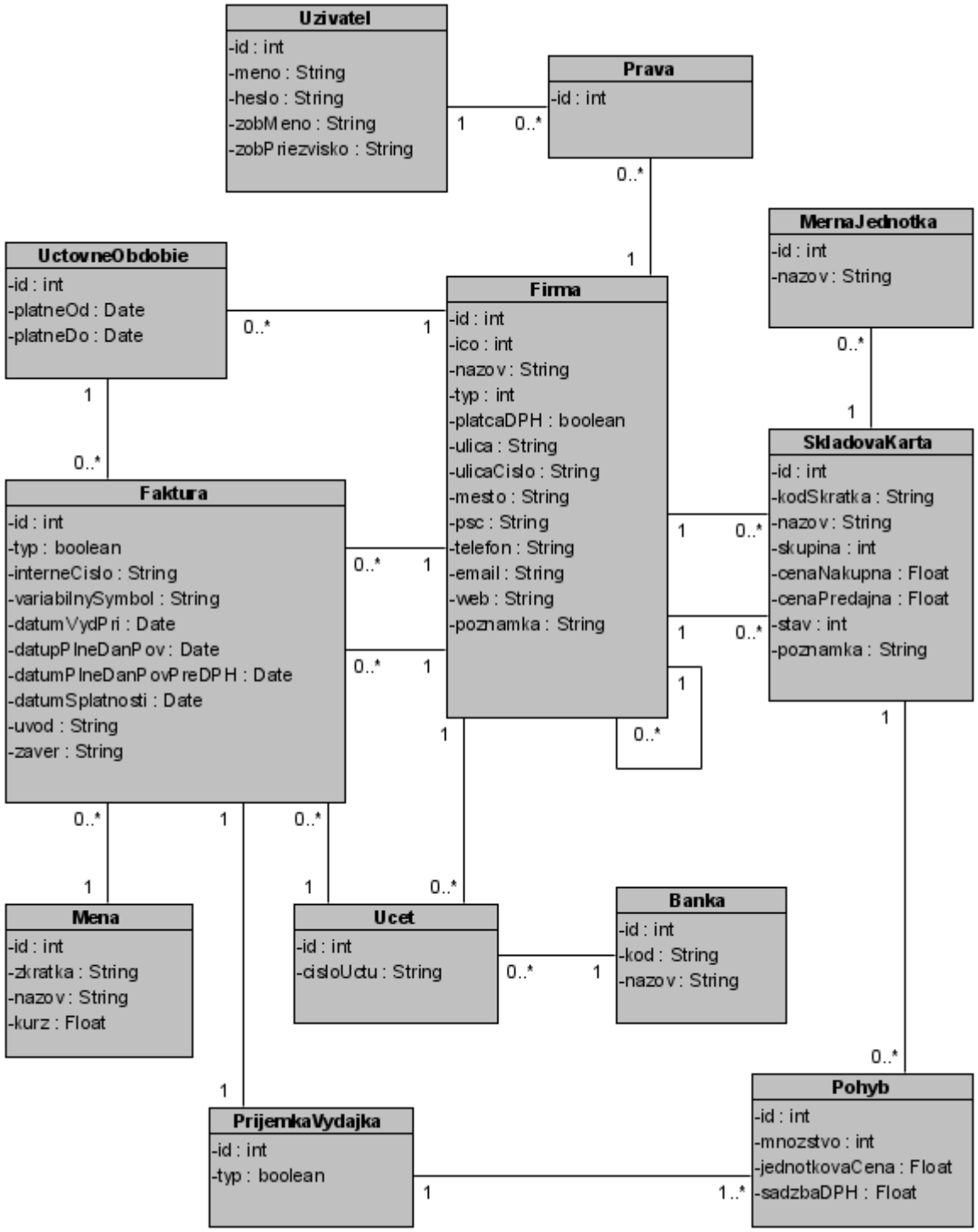
4.3 Dodržiavanie návrhových vzorov

Použitie návrhových vzorov a ich dodržanie nemá význam iba pri samotnej tvorbe aplikácií. Pri ich využití získame ľahko udržiavateľný a škálovateľný systém. Úprava chýb, pridanie nových funkcionalít a prispôsobenie sa novým požiadavkám, sa týmto vo veľkej miere zjednoduší.

Pri použití aplikačného rámca Struts sa vyžaduje dodržiavanie návrhového vzoru MVC. Ako už bolo popísané v predošlých kapitolách, celá architektúra systému sa takto rozdelí do troch vzájomne nezávislých vrstiev. Ich popisu je venovaná nasledujúca kapitola.

4.4 Vrstva Model

Vrstva Model reprezentuje objekty aplikačnej domény. Tieto objekty je potrebné zachovať v tvare záznamov relačnej databázy. V tejto časti graficky a písomne uvádzame jednotlivé perzistentné triedy.



Obrázok 4.4 *Diagram perzistentných tried*

4.4.1 Trieda Firma

Trieda *Firma* uchováva informácie o obchodných partneroch a účtovných firmách fakturačného systému. Trieda obsahuje atribúty *id*, *ico*, *nazov*, *typ*, *platcaDPH*, *ulica*, *ulicaCislo*, *mesto*, *psc*, *telefon*, *email*, *web*, *poznamka*. Atribút *ico* predstavuje identifikačné číslo organizácie a je prideľované štátnymi orgánomi. Toto 8-miestne číslo sa používa na jednoznačnú identifikáciu právnickej osoby alebo podnikateľa. Atribút *Názov* pri právnickej osobe predstavuje obchodné meno alebo pri fyzickej osobe, meno a priezvisko podnikateľa. *Typ* identifikuje firmu ako dodávateľa alebo odberateľa. Atribút *platcaDPH* určuje nutnosť firmy plniť daňovú povinnosť. Jedná sa o platbu dane z pridanej hodnoty. Zmysel zvyšku atribútov je zrejmý a ich zadanie je povinné iba, ak sa jedná o účtovnú firmu.

4.4.2 Trieda Ucet

Táto trieda uchováva informácie o bankových spojeniach jednotlivých firiem. Trieda obsahuje atribúty *id* a *cisloUctu*.

4.4.3 Trieda Banka

Trieda *Banka* uchováva zoznam bankových inštitúcií. Tie sú použité pre úplnú identifikáciu bankových spojení. Atribúty triedy sú *id*, *kod* a *nazov*. *Kod* predstavuje číselnú skratku banky a *nazov* je jej úplné meno, ktoré ale nie je povinne zadávaným atribútom.

4.4.4 Trieda SkladovaKarta

Každá skladová položka vytvára skladovú kartu účtovnej firmy. Jej úlohou je uchovávať záznam položky a evidovať jej cenu a stav na sklade. Trieda *SkladovaKarta* je reprezentovaná atribútmi *id*, *kodSkratka*, *nazov*, *skupina*, *cenaNakupna*, *cenaPredajna*, *stav*, *poznamka*. Atribút *skupina* definuje typ skladovej položky evidovanej v karte. Konkrétne sa môže jednať o službu alebo materiál. *CenaNakupna* je iba informačný atribút a jeho hodnotou sa nemusí fakturant riadiť pri stanovení predajnej ceny. Tento parameter je nepovinný. *CenaPredajna* je atribút povinný a jeho hodnota by mala byt smerodajná pri určení koncovej ceny za skladovú položku. Dôležitým atribútom celej triedy je *Stav*. Ten reprezentuje zásobu skladovej položky vyjadrenú v merných jednotkách. Nepovinne zadávaným atribútom je poznámka.

4.4.5 Trieda MernaJednotka

Reprezentuje merné jednotky, v ktorých je možné vyjadriť množstvo skladovej položky. Táto trieda obsahuje atribúty *id* a *nazov*, ktoré určujú mernú jednotku.

4.4.6 Trieda UctovneObdobie

Účtovné obdobie predstavuje časový interval spravidla v dĺžke jedného roka, vzhľadom ku ktorému je vykonávaný proces fakturácie. Trieda *UctovneObdobie* pozostáva z atribútov *id*, *platneOd* a *PlatneDo*. Posledné dva atribúty určujú časový interval dĺžky obdobia.

4.4.7 Trieda Faktura

Faktúra je doklad, ktorý uvádza účet za dodanie tovaru alebo vykonanie služby odberateľovi. Príslušná trieda obsahuje atribúty *id*, *typ*, *interneCislo*, *variabilnySymbol*, *datumVydPri*, *datumPlneDanPov*, *datumPlneDanPovPreDPH*, *datumSplatnosti*, *uvod* a *zaver*. Atribút *typ* definuje faktúru ako prijatú alebo vydanú. *InterneCislo* uvádza poradie zaevidovanej prijatej alebo vydanej faktúry v danom účtovnom období. *VariabilnySymbol* je výlučne numerické vyjadrenie čísla faktúry a slúži pri identifikácii zadávaných bankových prevodov. Atribút *datumVydPri* definuje dátum prijatia alebo vydania faktúry. *DatumPlneDanPov* a *datumPlneDanPovPreDph* sú povinné atribúty iba pre platcov DPH a určujú dátum plnenia daňovej povinnosti a dátum plnenia daňovej povinnosti pre priznanie DPH. *DatumSplatnosti* má vo väčšine prípadov rovnakú hodnotu ako atribút *datumVydPri*. Tento dátum je dôležitý pre včasné vytvorenie príkazu k úhrade. Atribúty *uvod* a *zaver* slúžia na uvedenie dodatočných informácií na fakturačnom doklade.

4.4.8 Trieda PrijemkaVydajka

Pri prijatí alebo vystavení faktúry sa automaticky generuje príjemka respektíve výdajka. Tieto doklady vytvárajú súpis tovaru a služieb, ktoré boli fakturované. Trieda ktorá reprezentuje tieto doklady ma názov *PrijemnaVydajka* a obsahuje atribúty *id* a *typ*, ktorý určuje, o ktorý z týchto dvoch dokladov sa jedná.

4.4.9 Trieda Pohyb

Táto trieda uchováva informácie o pohybe skladových položiek, ktoré sú uvedené na príjemkách a výdajkách. Atribútmi tried sú *id*, *množstvo*, *jednotkovaCena* a sadzbaDPH, ktorá uvádza výšku dane z pridanej hodnoty.

4.4.10 Trieda Mena

Prijatú a vydanú faktúru je možné evidovať vo fakturantom zvolenej mene. Triedu *Mena* reprezentujú atribúty *id*, *skratka*, *nazov*, *kurz*. Posledný uvedený atribút udáva platný kurz voči 1 €.

4.4.11 Trieda Uzivatel

Táto trieda uchováva informácie o užívateľoch systému. Užívatelia predstavujú fakturantov, ktorí vytvárajú fakturačné a iné doklady účtovnej firmy. Túto triedu tvoria atribúty *id*, *meno*, *heslo*, *zobMeno*, *zobPriezvisko*. Parameter *zobPriezvisko* nie je povinný.

4.4.12 Prava

Užívateľské práva definujú právo užívateľa byť fakturantom jednej alebo viacerých účtovných firiem. Táto trieda pozostáva z jediného atribútu *id*.

5 Implementácia

Poslednou fázou vývoja aplikácie je proces implementácie. V tejto kapitole uvediem implementáciu jednotlivých vrstiev pomocou vybraných fragmentov kódu a ich popisu. Posledná časť kapitoly je zameraná na konfiguráciu a použitie nástroja Hibernate a aplikačného frameworku Apache Struts.

5.1 Vrstva Model

Popis tejto vrstvy je venovaný nástroju Hibernate [3], [7] a [8]. V nasledujúcich podkapitolách uvediem jeho potrebnú konfiguráciu a popíšem vzorovú prácu s perzistentnými objektmi.

5.1.1 Konfigurácia Hibernate

Ako už bolo uvedené v predchádzajúcich kapitolách, Hibernate je možné konfigurovať viacerými spôsobmi. Pri implementácii aplikácie som zvolil konfigurácie pomocou XML súboru, ktorý má názov `hibernate.cfg.xml`. Fragment kódu z tohto súboru:

```
<hibernate-configuration>
    <session-factory>
          <!-- JDBC properties -->
          <property name="dialect">
               org.hibernate.dialect.MySQLInnoDBDialect
          </property>
          <property name="connection.driver_class">
               com.mysql.jdbc.Driver
          </property>
          <property name="connection.url">
               jdbc:mysql://localhost/dbName
          </property>
          <property name="connection.username">user</property>
          <property name="connection.password">password</property>
          <!-- Mapping files -->
          <mapping resource="package/Class.hbm.xml" />
      </session-factory>
</hibernate-configuration>
```

Tieto nastavenia určujú typ používanej databázy, definujú potrebne náležitosti pre jej pripojenie a určujú zdroj mapovania objektov.

Pre vytvorenie JDBC spojenia sú dôležité parametre uvedené pod komentárom `JDBC properties`. Dialekt určuje typ storage enginu databáze. Tie sa od seba líšia kontrolou integrítných obmedzení, kaskádovým mazaním a podobne. Ako príklad pre MySQL uvádzam dialekt InnoDB alebo MyISAM. Druhý uvedený parameter definuje triedu driveru pre zvolenú databázu. Po ňom nasledujú parametre definujúce URL adresu DB serveru, prihlasovacie meno a heslo.

Pod komentárom `Mapping files` je uvedená ukážka popisu umiestnenia mapovacieho XML súboru perzistentnej triedy.

Okrem už uvedených parametrov je možné v tomto konfiguračnom súbore nastaviť veľké množstvo ďalších volieb, ako napríklad connection-pooling alebo cacheovanie. Popis všetkých parametrov je možné vyhľadať v API dokumentácii.

5.1.2 Mapovanie objektov do tabuliek relačnej databázy

Pre mapovanie objektov som zvolil spôsob vytvorenia samostatného mapovacieho XML súboru pre každú z perzistentných tried. V nasledujúcom fragmente kódu je uvedená ukážka mapovania perzistentných tried na tabuľky relačnej databázy.

```
public class Banka{
      private int id;
      private String kod;
      private String nazov;

      // gettery a settery
}
<hibernate-mapping>
    <class catalog="dbName" name="package.Banka" table="banka">
          <id name="id" type="int">
               <column length="4" name="id_banka" />
               <generator class="identity" />
          </id>
          <property name="kod" type="string">
               <column length="4" name="kod" not-null="true"
               unique="true" />
          </property>
          <property name="nazov" type="string">
               <column length="30" name="nazov" />
          </property>
    </class>
</hibernate-mapping>
```

Týmto spôsobom je perzistentná trieda `Banka` mapovaná na tabuľku relačnej databázy. Každý z atribútov triedy má definovaný príslušný stĺpec tabuľky databázy. Jednotlivé parametre a ich hodnoty sú väčšinou zrejmé. Popíšem iba spôsob určenia hodnoty identifikátora. V tejto ukážke bol zvolený spôsob použitia identifikačného stĺpca. Tento spôsob reprezentuje hodnota `identity` parametra `generator class`.

Mapovanie triedy bez vzťahu k iným triedam je veľmi ojedinelé. Takmer každá trieda je asociovaná s jednou alebo viacerými triedami. Poznáme dva typy asociácií. Prístup k asociovanému objektu v obidvoch smeroch zabezpečuje obojsmerná (bidirectional) asociácia. Prístup iba v jednom smere je jednosmerná (undirectional) asociácia. Z pohľadu kardinality poznáme 1:1, 1:N, N:1, N:M asociácie. Najčastejšie používanou je jednosmerná asociácia N:1.

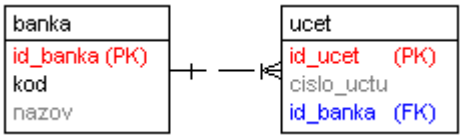
Pre priblíženie tejto problematiky uvediem fragment kódu, ktorý nadväzuje na už uvedenú triedu `Banka` a k nej pridám triedu `Ucet`. Je zrejmé, že každý účet je zriadený v nejakej banke, pričom jedna banka spravuje viacero účtov. Práve tento vzťah predstavuje kardinalitu N:1. Pre vytvorenie asociácie je potrebne do triedy `Ucet` pridať atribút `banka` a pri mapovaní triedy uviesť definíciu mapovania pomocou elementu `<many-to-one>`.

```
public class Ucet {
    private int id;
    private Banka banka;
    private String cisloUctu;

    // gettery a settery
}
```

```
<hibernate-mapping>
     <class catalog="dbName" name="package.Ucet" table="ucet">
          <id name="id" type="int">
               <column name="id_ucet"/>
               <generator class="identity"/>
          </id>
          <many-to-one class="package.Banka" name="banka">
               <column length="4" name="id_banka" not-null="true"/>
          </many-to-one>
          <property name="cisloUctu" type="string">
               <column length="30" name="cislo_uctu"/>
          </property>
     </class>
</hibernate-mapping>
```

Vytvorené mapovacie XML súbory je potrebné uviesť do konfiguračného súboru `hibernate.cfg.xml`. Výsledkom perzistencie tried je relačná databáza zobrazená v nasledujúcom obrázku.



Obrázok 5.1 *Tabuľky relačnej databázy*

5.1.3 Práca s perzistentnými objektmi

Pred začatím práce s perzistentnými objektmi je potrebne vykonať niekoľko úkonov. Najprv je nutné nakonfigurovať a vytvoriť `session factory`, ktorá sa vytvára spravidla iba raz, a to pri prvom otvorení sessiony. Tento úkon vo vytvorenej aplikácii zabezpečuje trieda `SessionProvider`, ktorá okrem iného má za úlohu aj otváranie a uzavretie sessiony. Fragment kódu konfigurácie:

```
private static final SessionFactory sessionFactory;
static {
    try {
          Configuration config = new Configuration();
          config = config.configure("hibernate.cfg.xml");
          sessionFactory = config.buildSessionFactory();
    }
    catch (HibernateException ex) {
          // error
    }
}
```

Pre prácu s perzistentnými objektmi som vytvoril triedu `EntityManager`. Jej úlohou je vykonávať všetky operácie nad databázou. Tato trieda obsahuje metódy pre načítavanie, ukladanie, aktualizáciu a mazanie objektov. Všetky perzistentné triedy používajú inštanciu objektu `EntityManager`. Tým sa zabezpečí jednotný prístup k všetkých metódam a celá práca s objektmi sa tak zjednoduší. Zjednodušenie vytvárania inštancií zabezpečuje `EntityManagerFactory`, ktorá vytvára objekty príslušnej triedy.

Pre načítavanie objektu som použil metódu `get()` a v prípade zložených dopytov je použitý jazyk HQL. Univerzálna metóda pre dopytovanie nad objektmi podľa vybraného parametra je nasledujúca:

```
public List findAllByAttribute(String attribute, String value)
            throws HibernateException {
      Session session = SessionProvider.currentSession();
      String qry = "from " + class.getName() + " as e where e." +
      attribute + "=" + "'" + value + "'";
      Query q = session.createQuery(qry);
      List result = q.list();

      return result;
}
```

Pre ukladanie, aktualizáciu a mazanie objektov boli použité metódy `save()`, `marge()` a `delete()`. Ich podrobný popis je možné vyhľadať v Hibernate API dokumentácii.

5.2 Vrstva Controller

Pri popise tejto vrstvy sa zameriam na aplikačný framework Apache Struts [3], [9], [10], ktorý som použil pri vývoji moje bakalárskej práce.

5.2.1 Konfigurácia Struts

Hlavným komponentom celého frameworku je Struts servlet, ktorý je inštanciou triedy `ActionServlet`. Tento je potrebné definovať v konfiguračnom súbore `web.xml`. Fragment kódu pod týmto textom uvádza nasledujúcu konfiguráciu.

```
<servlet>
      <servlet-name>action</servlet-name>
      <servlet-class>
            org.apache.struts.action.ActionServlet
      </servlet-class>
      <init-param>
            <param-name>config</param-name>
            <param-value>/WEB-INF/struts-config.xml</param-value>
      </init-param>
      <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
      <servlet-name>action</servlet-name>
      <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Uvedené nastavenie zabezpečí, že prijatie URL požiadavky v tvare `*.do` vyvolá reakcia `action` servletu. Ten na základe prijatej adresy určí konkrétnu reakciu systému v zmysle vykonania akcie.

5.2.2 Vytvorenie akcie

Definície jednotlivých akcií sú uvedené v konfiguračnom súbore `struts-config.xml`. Parametre tejto definície sú nasledujúce:

- **path:** relatívna k cesta k akcií v rámci aplikácie
- **type:** názov balíčka a triedy akcie, ktorá vykoná obsluhu
- **name:** názov form beany definovanej pre túto akciu
- **validate:** určenie, či má prebehnúť validácia pre túto akciu
- **input:** akčné mapovanie alebo JSP stránka vstupného formuláru; chyba pri validácii spôsobí návrat na túto stránku
- **scope:** rozsah platnosti akcie (`request, scope`)
- **forward:** cesta k akciám alebo JSP stránkam, ktorým bude odovzdané spracovanie po ukončení potrebnej akcie

Počet použitých parametrov a ich hodnoty sú závislé od konkrétnych akcií. Ukážka vzorovej definície je nasledujúca:

```
<action-mappings>
    <action path="/logon"
           type="struts.LogonAction"
          name="logonForm"
          input="/error.jsp">
          <forward name="success" path="/index.jsp"/>
          <forward name="failure" path="/error.jsp"/>
    </action>
</action-mappings>
```

Ako už bolo popísane v samostatnej kapitole venovanej tomuto frameworku, súbor `struts-config.xml` obsahuje aj konfiguráciu ďalších volieb. Ich bližší popis a konkrétne nastavenie je možné vyhľadať v príslušnej dokumentácii.

Po konfigurácii akcie je potrebné naimplementovať akciu samotnú. Implementovaná trieda, ktorá rozširuje `Action` triedu, musí obsahovať metódu `execute()` pre obsluhu frameworku. Parametre tejto metódy sú:

- **mapping:** reprezentuje `ActionMapping` konfiguráciu použitú pre túto akciu
- **form:** predstavuje `ActionForm`, ktorý môže byť voliteľne použitý touto akciou
- **request:** HTTP požiadavka prijatá aplikáciou

- **response:** reakcia na prijatú požiadavku

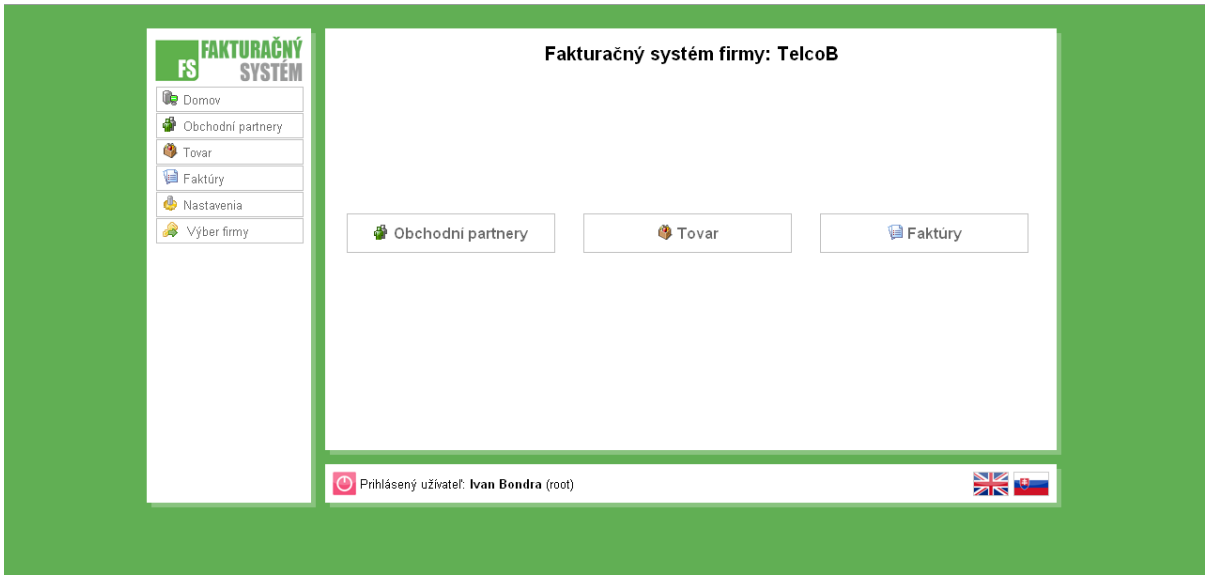
Aj keď parametre metódy sú pevne dané, ich použitie je na voľbe programátora. Ukážka fragmentu kódu takto vytvorenej metódy je nasledujúca:

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response)
    throws Exception
```

5.3 Vrstva View

Túto vrstvu tvoria JaveServer Pages (JSP) obsahujúce elementy z knižníc JavaServer Pages Standard Tag Library (JSTL), Struts Tag Libraty a HTML. Štýl stránok je vytvorený pomocou CSS definícií a vytvorenie nových okien je zabezpečené pomocou JavaScriptu. Pre reprezentáciu tabuľkových dát je zvolená knižnica Dispaly tag. Táto knižnica zjednodušuje výpis dát, ich triedenie podľa zvoleného atribútu a poskytuje možnosť exportovať zobrazenú tabuľku do štyroch rôznych formátov. Pre obmedzenie duplicity kódu stránok je použitá direktíva `include`.

Cieľom návrhu celého užívateľského rozhrania bola prehľadnosť a jednotnosť v grafikách jednotlivých obrazoviek. Užívateľská a administrátorská časť systému obsahujú rozcestník, ktorý zobrazuje odkazy na všetky hlavné časti systému podľa typu prihláseného užívateľa. V ľavej časti rozhrania sa nachádza klasické vertikálne menu a v spodnej časti je umiestnená stavová lišta. Slúži pre zobrazenie údajov o aktuálne prihlásenom užívateľovi a obsahuje tlačidlá pre zmenu jazyka a odhlásenie sa zo systému. Na nasledujúcom obrázku je vyobrazená ukážku tohto rozhrania. Ďalšie ukážky systému je možné nájsť v časti Príloha 1.



Obrázok 5.3 *Ukážka rozhranie (užívateľská časť aplikácie)*

6 Záver

Cieľom tejto bakalárskej práce bolo oboznámiť sa s platformou Java EE s využitím technológie pre objektovo-relačné mapovanie. Nadobudnuté poznatky som uplatnil pri implementácii viacjazyčného fakturačného systému, ktorý spĺňa náležité aktuálne legislatívne nariadenia.

V prvej časti tejto práce som sa venoval použitým technológiám. Zameral som sa na predstavenie platformy Java, nástroja na perzistenciu Hibernate a aplikačného frameworku Apache Strus. Mojím cieľom nebol podrobný popis jednotlivých technológií. Pozornosť som venoval častiam, ktoré som skutočne použil pri implementácii aplikácie. Pre správne pochopenie a použitie zvolených techník som veľa času strávil štúdiom a získavaním potrebných informácií. Je škoda, že tak zaujímavé a vo svete veľmi často používané technológie, sú českými a slovenskými autormi technickej literatúry prehliadané. Zdrojom informácií pre mňa z uvedeného dôvodu bola prevažne zahraničná literatúra a vo veľkej miere aj internet.

Druhá časť práce bola venovaná návrhu a implantácii celého riešenia. Tak ako v iných podobných projektoch, aj v tomto, bolo dôležité správne formulovať jednotlivé požiadavky a vytvoriť podrobnú analýzu celého systému. Zdrojom informácií mi boli už vytvorené komerčné systémy a podnetné názory z praxe (otca, ktorý každodenne prichádza do styku s procesom fakturácie). Celý systém bol vytváraný so zreteľom na platnú legislatívu. Pri opise návrhu riešenia som použil diagramy prípadov využitia a perzistentných tried. Každý z diagramov obsahuje podrobný popis. Implementácia aplikácie vysvetľuje použitie objektovo-relačného mapovania založeného na nástroji Hibernate a vytváranie systému pomocou aplikačného frameworku Apache Struts.

Ako už bolo viackrát uvedené, mnou implementovaný systém obsiahol iba určitú časť procesu fakturácie. V budúcnosti je možné aplikáciu rozšíriť o vedenie pokladne, peňažný denník alebo objednávkový systém, ktorý by napríklad obsahoval prepojenie na internetový obchod.

Stavba aplikácie podľa architektúry návrhového vzoru MVC a použitie už spomenutých technológií sa ukázala ako správna. Zabezpečuje totiž jeho ľahšiu údržbu a prípadné ďalšie zdokonalenie a vývoj aplikácie.

Pred vypracovaním bakalárskej práce som mal iba nejasné rámcové predstavy o vývoji aplikácie na platforme Java EE. Podrobným oboznámením sa s jednotlivými technológiami a ich praktickou aplikáciou som získal cenné vedomosti a skúsenosti, ktoré dokážem zúročiť vo svojom profesionálnom raste.

6.1 Literatúra

[1] Darwin, I.: :*Java kuchárka programátora.* Brno, Computer Press, a.s, 2006.

[2] Sun Microsystems: The *Java Tutorial*. 2009, URL: http://java.sun.com/docs/books/tutorial/

[3] Chopra, V., Li, S., Jones, R., Eaves, J., T.Bell, J.: Beginning JavaServer Pages, Wiley Publishing, Inc., 2005

[4] Mukhar, K., Zelenak, Ch., Weaver, J., Crume, J.: Beginning Java EE 5: From Novice to Professional, Apress, 2006

[5] Hibernate Reference Documentation, 2009, URL: http://docs.jboss.org/hibernate/stable/annotations/reference/en/html/

[6] Šenk, Z.: Technologie pro perzistenci objektů v Javě, diplomová práce, FIT VUT, 2007

[7] Minter, D., Linwood, J.: Beginning Hibernate: From Novice to Professional, Apress, 2006

[8] Bauer, Ch., King, G.: Java Persistence with Hibernate, Manning Publications Co., 2007

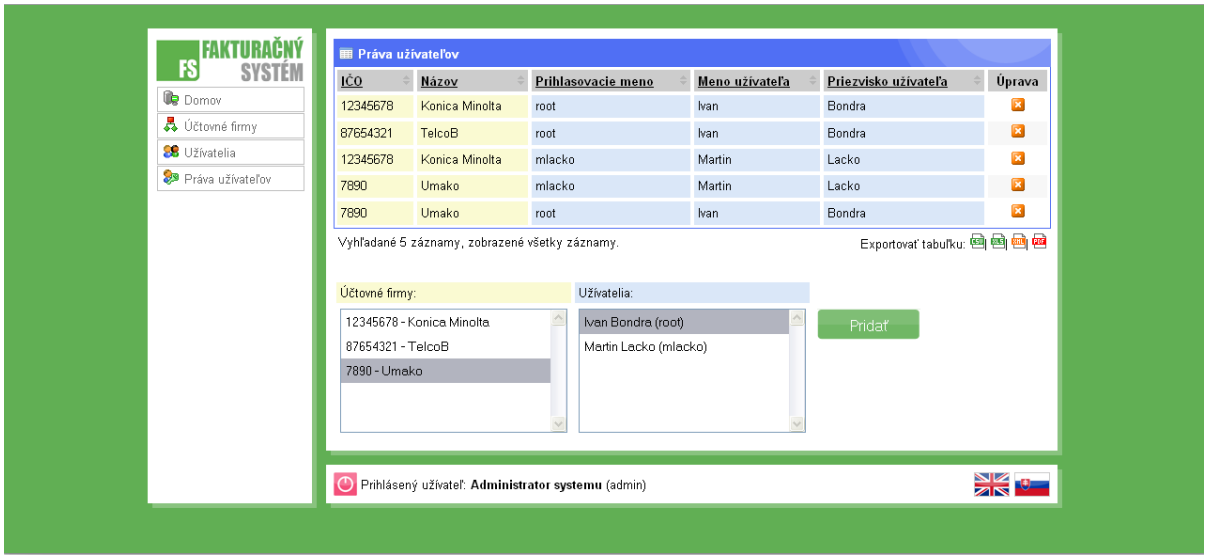
[9] Doray, A.: Beginning Apache Struts: From Novice to Professional, Apress, 2006

[10] Struts Tutorial, 2009, URL: http://www.vaannila.com/struts/struts-tutorial/struts-tutorial.html

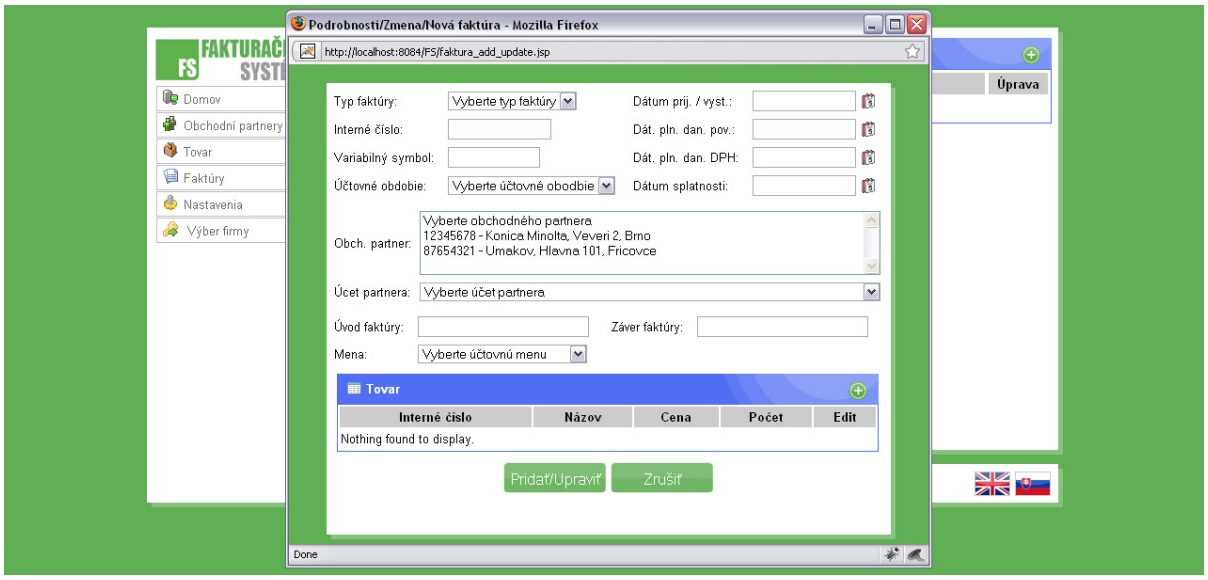
Zoznam príloh

Príloha 1: Ukážky užívateľského rozhrania

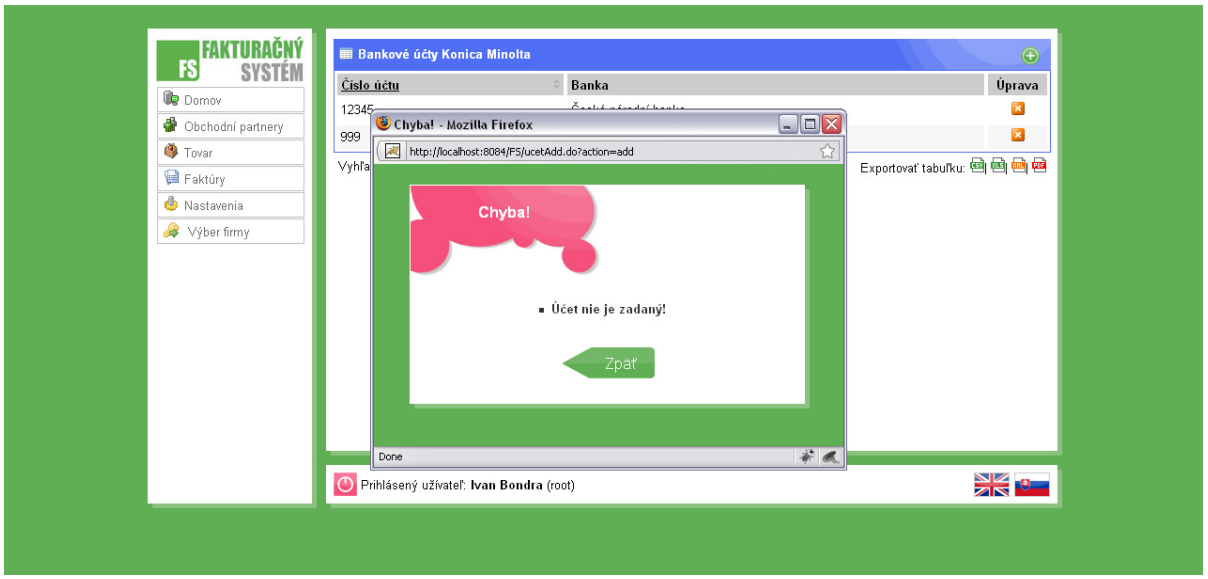
Obrázok P1.1 *Prihlásenie do Fakturačného systému*



Obrázok P1.2 *Definovanie užívateľských práv (administrátorská časť aplikácie)*



Obrázok P1.3 *Pridanie fakturačného dokladu (užívateľská časť aplikácie)*



Obrázok P1.4 *Chybové hlásenie s popisom*