VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

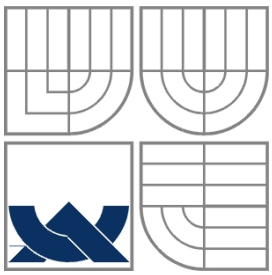
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO VEDENÍ DAŇOVÉ EVIDENCE

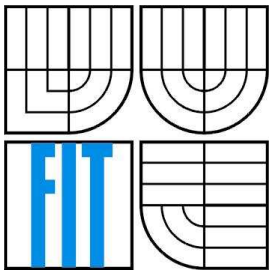
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. JAKUB DOSTAL
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO VEDENÍ DAŇOVÉ EVIDENCE

ACCOUNTING SYSTEM

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAKUB DOSTAL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2007

Licenční smlouva

Licenční smlouva je uložena v archívu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

Cílem práce je navrhnout a realizovat základ webového informačního systému pro podporu vedení daňové evidence s ohledem na českou legislativu. Součástí projektu je obecné seznámení s problematikou daňové evidence, dříve jednoduchého účetnictví a s podmínkami pro korektní vedení příjmů a výdajů důležitého pro výpočet základu daně z příjmu. Výsledkem práce je analýza požadavků, návrh informačního systému s ohledem na možnosti a potřeby uživatele a popis implementace vzhledem k použitým technologiím. Navíc práce obsahuje pojednání o existujících produktech v dané oblasti a jejich porovnání právě z pohledu uživatele, jež se stalo výchozím bodem pro stanovení požadavků na systém.

Klíčová slova

J2EE, JSP, Servlet, Struts, Hibernate, EJB, daňová evidence

Abstract

The aim of the work is to design and implement a basic interface of a web information system for the accounting management that corresponds to the Czech legislation. The project contains a brief introduction to the subject of accounting and the terms of proper evidence of receipts and expenditures, which are significant for basic income-tax computing. The results of the work are the requirements analysis, the system design considering the user options and needs and the description of the system implementation process with reference to used technology. In addition, the work includes a description and comparison of four commercial products from the user point of view, which became an initial point for requirements determination.

Keywords

J2EE, JSP, Servlet, Struts, Hibernate, EJB, tax accounting

Citace

Dostal Jakub: Systém pro vedení daňové evidence. Brno, 2007, diplomová práce, FIT VUT v Brně.

Systém pro vedení daňové evidence

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Radka Burgeta, Ph.D.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jakub Dostal
21. května 2007

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Radkovi Burgetovi za jeho nápady, podněty a odborné vedení při realizaci práce.

Obsah

1 Úvod

Dnešní doba širokého rozmachu možností využití internetu je zdá se vhodnou příležitostí k vytváření informačních systémů různých zaměření a funkčností. Právě proto jsem se přiklonil k úvaze o realizaci informačního systému zaměřeného na pro mě ne zcela známou oblast. Pokusil jsem se propojit výhody internetového prostoru s nezbytnou součástí všech právních subjektů ve sféře podnikání na řekněme základnější úrovni. Jedná se o systém pro vedení daňové evidence, který si klade za cíl v co největší míře zjednodušit a usnadnit zpracování veškerých náležitostí, které všem podnikajícím elementům ukládá zákon a to i s ohledem na aktuální českou legislativu. Je třeba si mimo jiné uvědomit, že oblast finanční sféry se neustále vyvíjí a pozměňuje, a proto je třeba již při návrhu zohlednit i možné budoucí legislativní úpravy. Na tuto problematiku je ve spoustě případů často zapomínáno a pak bývá aplikování daných problémů v již existujícím systému mnohem náročnější. Co ovšem je daňová evidence a co všechno musí daný systém pro její korektní plnění obsahovat, bude tématem následující kapitoly.

Nyní jen stručně, proč vlastně využívat pro danou problematiku možností internetových aplikací. Většina dostupných systémů pro vedení daňové evidence, tedy zjednodušenou formu účetnictví, předpokládá použití pro daný účel samostatných aplikací. Z nich některé využívají dostupné databázové technologie různých licencí či firem a jiné zase sází na vývoj a uplatnění technologií vlastních. Některé aplikace považují distribuovanou správu evidence za nadstandardní funkčnost a další ji, díky své technologii, vůbec neumožňují. Pokud ano, jedná se o složité klient-server aplikace, které svojí robustností převyšují nejen potřeby, ale často i finanční možnosti potenciálního uživatele. Proto čistě serverová internetová aplikace, která ke svému chodu na straně klienta potřebuje pouze standardní internetový prohlížeč, je alternativou jistě výhodnější po všech stránkách jak pro klienta, tak i samotného vývojáře. Je jisté, že daná technologie nabízí více možností, než využije systém pro vedení jednoduchého účetnictví, je však vynikajícím základem pro další možná rozšíření, které jsou nutností pro mnohem náročnější formy vedení finančních záležitostí. Na první pohled by mohla vyvstat otázka bezpečnosti, avšak například nemálo bank využívá danou technologii pro službu internetového bankovnictví svých klientů. Můžeme tedy s klidem potvrdit, že pokud nastane nějaká bezpečnostní chyba, je jisté, že nepochází z použité technologie, nýbrž selhání lidského faktoru s ní pracujícího. O použitých technologiích a jejich možnostech bude pojednáváno v některé z následujících kapitol.

Obsahem semestrálního projektu, který předcházel samotnému zpracování diplomové práce, byla prvotní analýza požadavků na systém podobného zaměření a předběžný plán projektu výstavby konkrétní aplikace, dále pak mimo jiné i analýza již existujících systémů a jejich porovnání s ohledem na možnosti, jež nabízejí. Tato analýza budiž základem pro stanovení požadavků na vlastní systém, jež bude mít snahu rozvinout a využít kladných vlastností testovaných produktů a vyvarovat se

zjevných důležitých nedostatků. Tento souhrn je započítán v navazující analýze požadavků, kde je již sumarizováno, jaké vlastnosti a funkční moduly by měl systém pro daňovou evidenci obsahovat. Je samozřejmě pochopitelné, že v rámci diplomové práce není možné vybudovat plnohodnotně konkurenceschopnou aplikaci co do počtu a rozsahu možných modulů z časo-prostorových důvodů, nicméně musí být a fungovat jako základ pro další rozšíření o moduly, které budou jako celek co možná nejvíce usnadňovat nelehký doplněk podnikání. Tyto přídavné části jsou tedy vedeny v kapitole o možnostech rozšíření.

2 Daňová evidence

2.1 Úvodem

Jak již bylo v úvodu naznačeno, v této kapitole zmíním základní a obecné informace o daňové evidenci, co vůbec evidence daní obnáší a co z toho vyplývá pro samotný návrh. Součástí poslední podkapitoly bude i ukázka vedení evidence příjmů a výdajů v tabulkové podobě. Obrázek byl použit ze serveru euroekonom.cz, ze kterého byly ostatně částečně čerpány i doprovodné informace.

Jednotlivé konkrétní části evidence (kniha pohledávek a závazků, karty dlouhodobého majetku atp.) budou podrobněji popsány až v samotném návrhu podle vhodnosti uživatelského rozhraní. Touto částí by bylo zbytečné se zabývat znovu v této kapitole, neboť obecně teorie daňové evidence není hlavním tématem.

2.2 Definice

Pro bližší pochopení dané problematiky je nutné objasnit si základní pojmy. Daňová evidence (DE) obecně slouží ke stanovení základu daně z příjmu. Obsahuje veškeré údaje o příjmech a výdajích, dále pak může evidovat údaje o majetku a závazcích. DE je ve své podstatě velice zjednodušenou a zredukovanou formou účetnictví. Svým způsobem navazuje na jednoduché účetnictví, které bylo v České republice zrušeno v roce 2004. Nyní zůstává pouze účetnictví, kterým se v Zákoně o účetnictví rozumí pouze podvojné účetnictví. Jednoduché účetnictví tedy zaniklo, ale fyzické osoby (FO), které chtějí vést evidenci na podobných principech jako bylo jednoduché účetnictví, mohou vést právě DE, a to podle § 7b Zákona o daních z příjmu.

2.3 Pro koho je určena

DE je určena pro ty subjekty, které nejsou účetní jednotkou (ÚJ) ve smyslu Zákona o účetnictví č. 563/91 Sb. Obecně je tedy určena pro všechny podnikající FO, které nepovedou podvojné účetnictví a své výdaje nebudou prokazovat „paušálně“, jinými slovy procentem z příjmů.

Daňovou evidenci v žádném případě nepovedou ty osoby, které jsou ÚJ. To se týká následujících osob:

- Právnické osoby (PO), které mají sídlo na území České republiky
- FO, které jsou jako podnikatelé zapsání v obchodním rejstříku
- FO, jejichž obrat přesáhl v předchozím kalendářním roce částku 15 mil. Kč

- FO, které vedou podvojné účetnictví dobrovolně

Daňovou evidenci tedy mohou bez problémů vést všichni drobní podnikatelé (FO), kteří nejsou zapsaní v obchodním rejstříku, s ročním obratem pod 15 mil. korun.

2.4 Jak by měla vypadat

O tom, jak má daňová evidence přesně vypadat, nehovoří žádný z paragrafů zákona o daních z příjmu. Potenciálnímu daňovému poplatníkovi dává jen zcela základní informace, o čem daňová evidence příjmů a výdajů vypovídá, jakým způsobem je oceněn majetek a odpovídající závazky. Dále se zákon zmiňuje o tom, že je potřeba provádět pokud možno pravidelnou inventarizaci a archivaci. Záleží tedy na samotném poplatníkovi, jakou formu evidence zvolí. Podle zákona nejsou předepsány knihy závazků, ani knihy a karty majetku, dále pak peněžní deník a pokladní kniha. Celou daňovou evidenci je, podle úrovně zjednodušení, možno vést třeba i v libovolném tabulkovém editoru. Je však třeba si velice dobře uvědomit, že se na základě vedené daňové evidence bude finančnímu úřadu dokazovat, že byly v následném daňovém přiznání uvedeny daňové příjmy a výdaje ve skutečné výši.

Nemálo důležité je taktéž si uvědomit, že DE musí splňovat náležitosti, které vyplývají ze zákona o daních z příjmu. Jinými slovy musí obsahovat informace o příjmech a výdajích, členěné pokud možno způsobem vhodným pro snadné zjištění základu samotné daně. Dále pak informace o majetku a závazcích. Podle serveru euroekonom.cz [1] musí být evidence příjmů a výdajů (peněžní evidence) oddělena od evidence majetku a závazků. Příjmy daňové je taktéž třeba oddělovat od příjmů nedaňových a výdaje na zjištění a udržení příjmů („daňové výdaje“) je třeba oddělovat od výdajů neovlivňujících základ daně („nedaňové výdaje“).

2.5 Cíl

Hlavním cílem vedení daňové evidence je tedy (jak již bylo zmíněno dříve) zjištění základu daně. To je důležité pro vyplnění daňového přiznání a následnému odvedení daní finančnímu úřadu. Pokud je daňová evidence vedena přehledně a se značnou pečlivostí, poskytuje danému podnikateli přehled o stavu a pohybu majetku a dluhů (pohledávky a závazky), nemluvě pochopitelně o samotném výsledku jeho podnikání.

Výhodou určité volnosti oproti například podvojnému účetnictví je to, že si každý může evidenci jakkoli rozšířit, např. o rozdělení jednotlivých příjmů a výdajů podle různých kritérií dle libovůle poplatníka, neboť důležitý je až samotný výsledek.

Obr. 1 (převzato z [1]) znázorňuje přehledné vedení příjmů a výdajů včetně dělení na daňové a nedaňové, o kterém jsem se zmiňoval v předchozí kapitole.

Datum	Doklad	Popis	Příjmy nedaňové	Výdaje nedaňové	Příjmy daňové	Výdaje daňové	Příjmy § 8
1.5.2005	PN1	Vklad na běžný účet	5 000,00				
10.5.2005	VD1	Nákup techniky				8 000,00	
15.5.2005	VD2	Reklamní materiály				4 000,00	
27.5.2005	PD1	Poradenská činnost			15 000,00		
2.6.2005	VD3	Zdravotní pojištění				1 000,00	
2.6.2005	VD4	Sociální pojištění				1 100,00	
5.6.2005	PD2	Instalace software			2 000,00		
21.6.2005	PD3	Tvorba prezentace			5 000,00		
2.7.2005	VD5	Zdravotní pojištění				1 000,00	
2.7.2005	VD6	Sociální pojištění				1 100,00	
13.7.2005	PD4	Instalace software			2 500,00		
31.7.2005	Výpis1 BÚ	Úroky na běžném účtu					50,00
31.7.2005	Výpis1 BÚ	Poplatky za běžný účet				200,00	
1.8.2005	Výpis2 BÚ	Osobní spotřeba (výběz z BÚ)		7 000,00			
Součty			5 000,00	7 000,00	24 500,00	16 400,00	**50,00**
			Základ daně z příjmu			**8 100,00**	

Obr. 1 Evidence příjmů a výdajů

3 Analýza existujících produktů

3.1 Úvodem

Produktů, které se zabývají více či méně podobnou tématikou, je celá řada. Vzhledem k dlouhodobým zkušenostem, které provázejí vývoj jednotlivých aplikací, je zřejmé, že snaha o modelování zcela nové aplikace by bylo pouze opakované „objevování Ameriky“. Zřejmě i proto zadání práce hovoří o prostudování již vytvořených produktů, neboť požadavky a jejich následná analýza bude z větší části ovlivněna právě výsledkem analýzy (porovnání a zhodnocení) existujících produktů. Pro tento účel jsem zvolil ty produkty, u kterých existuje možnost získání zkušební verze bez nutnosti jejich nákupu. Zároveň byla snaha vybrat produkty, jež jsou podle dostupných informací jedny z nejpoužívanějších ve spektru drobných podnikatelů a živnostníků.

U každého ze čtyř testovaných produktů bude zkoušeno vytvoření faktury (v jednoduchém účetnictví též daňový doklad) a její zaúčtování na základě pokladního dokladu. Případně i testování evidence ostatních dokladů. Prakticky všechny produkty mají společné základní vlastnosti, neboť je ukládá zákon a legislativa. Zmíním tedy pouze případné rozdíly v uživatelské části (ovládání, notifikace chyb a jejich oprava, atp.). Použitými technologiemi jednotlivých produktů není nutné se zabývat, na samotný návrh vliv nemají. Pochopitelně veškeré zmíněné klady a zápory jsou čistě subjektivního rázu a slouží pro usnadnění volby požadavků na samotný navrhovaný systém. Všechny aplikace jsou na tak vysoké úrovni, že kontrola uživatele a jeho případných zásahů do evidence je samozřejmostí, není nutné se o ni dále zmiňovat. Množství a druh implementovaných modulů se příliš neliší. Jsou to:

- *Adresář* - není nutné pro samotné účetnictví, nezbytné však pro usnadnění práce
- *Daňová evidence (účetnictví)* – základ aplikace (peněžní deník)
- *Fakturace* – správa faktur přijatých a vydaných
- *Majetek* - správa majetku, hmotný, nehmotný a jejich odpisy
- *Sklady* – správa zásob, příjmových a výdejových dokladů
- *Mzdy a personalistika* – správa zaměstnanců

3.2 Produkty

3.2.1 Stormware Pohoda 2006

Hlavní jednotkou evidence je podle očekávání Peněžní deník. Zápisy v peněžním deníku vytváří Pohoda automaticky při zápisu a opravách dokladů v agendách Banka a Pokladna. Zaúčtování do

peněžního deníku se provádí dle vyplněné předkontace v záhlaví a položkách prvotního dokladu. Přímé zápisy do peněžního deníku se provádějí pouze u dokladů typu Převod a při opravách a rozúčtování jednotlivých řádků deníku.

Klady – zajímavé zpracování filtrace vypsaných položek, pěkně provedená daňová kalkulačka, přehled finančních prostředků, aplikace obecně velice rychlá a na ovládání přívětivá, právě používané agendy jsou přehledně k dispozici, v úvodu výpis ekonomických informací

Zápory – filtrace není příliš intuitivní, poněkud krkolomný výběr např. dodavatele/odběratele do různých dokladů, nepřehledný výpis a editace položek dokladů, zaúčtované faktury v peněžním deníku (uhrazovaný doklad) nejsou přehledně zobrazené, z toho vyplývá horší možnost jejich zpětného dohledání, neukládají se poslední otevřené agendy při ukončení práce s programem

3.2.2 Kastner software Stereo 2006

Záznamy o hotovostních platbách, výpisech z bankovních účtů i ostatní záznamy, které se mají objevit v peněžním deníku, se do peněžního deníku nezapisují přímo, ale pomocí agend prvotních dokladů (Pokladna, Bankovní výpisy, Interní doklady). Do samotného peněžního deníku se tyto doklady zaúčtovávají na základě nastavení údaje *Způsob zaúčtování*, v každé z jednotlivých agend.

V peněžním deníku je pak možno zaúčtované záznamy ze všech agend prohlížet nebo některý záznam rozúčtovat do více záznamů nebo jej změnit. V případě ručního zásahu do zaúčtování v peněžním deníku se nastaví v odpovídajícím dokladu prvotní evidence příznak, že byl záznam zaúčtování ručně změněn (údaj *účtováno* bude obsahovat hodnotu R).

Klady – volby možností jednotlivých atributů dokladů jsou k dispozici intuitivně a přehledně, včetně aplikování daného výběru zpět do dokladu, velkou výhodou je *Mapa programu* (velice intuitivní přehled možných operací přehledně uspořádán v chronologickém sledu dle použití)

Zápory – zbytečné zobrazení duplicitních informací v seznamu dokladů a současně zobrazovaném detailu zvoleného dokladu, v peněžním deníku není přehledně znázorněno, zda-li se jedná o příjem či výdej

3.2.3 ABRA G1

Velice komplexní systém pro vedení účetnictví pro drobné podnikatele a živnostníky. Obsahuje nepřeberné množství různých doplňkových modulů pro vedení evidence všemožných dokladů. Peněžní deník (základ DE) se vykazuje přímo ze zdrojových dokladů za pomoci typů příjmů a výdajů. Systém jako takový je velice robustní, mnohonásobně převyšuje běžné požadavky na podobný systém, což je s jistotou jeho výhoda, ovšem svojí komplexností může odradit zákazníky, kteří vyžadují jednoduchost a přímočarost.

Klady – samotný peněžní deník je veden velice přehledně, neobsahuje nadbytečné informace o dílčích dokladech, modul evidence klasické pošty

Zápory – nemožnost snadného přepínání mezi jednotlivými agendami, díky rozsáhlosti celého systému je samotný jeho běh relativně pomalý.

3.2.4 MRP Daňová evidence

Oproti výše zmíněným je tento systém skutečně velice zjednodušený, ovšem v základní verzi. Je možné jej komponovat z množství doplňků, které mohou společně tvořit komplexní celek, který je možno přizpůsobit právě pro konkrétní potřeby potenciálních zákazníků. Zaúčtování se provádí, tak jako v předchozích případech, na základě přijaté či vydané faktury a přijatého resp. vydaného dokladu o pohybu peněz (pokladna, banka).

Klady – jednoduchost a variabilita systému, celkově přehledná a intuitivní správa potřebných dokladů.

Zápory – v základní verzi chybí zabezpečení a víceuživatelská úroveň přihlášení

4 Analýza problému

4.1 Neformální specifikace požadavků

Pro neformální specifikaci požadavků budeme vycházet ze samotného zadání projektu. To hovoří jednoduše o systému pro vedení daňové evidence. Což je samo o sobě relativně rozsáhlé téma předpokládáme-li, že systém nebude splňovat pouze základ daňové evidence (peněžní deník), ale evidenci i většiny dílčích dokladů, čímž se aplikace velice rozšíří. Jedním z obecných požadavků je s určitostí snaha navrhnout systém pro uživatele co nejpřívětivější. Měl by být komplexní ale současně si musí zachovat přehlednost. Proto by bylo nejvhodnější umožnit nastavení přístupu každému potenciálnímu uživateli do jednotlivých části také s ohledem na bezpečnost. Tomu odpovídá nutnost navrhnout systém modulární. Konkrétní požadavky na celý informační systém by mohly být rozděleny do dvou skupin:

a) požadavky na daňovou evidencí

- Systém tedy musí obsahovat **evidenci příjmů a výdajů**, který se skládá z dokumentů dokazující fyzické příjmy a výdaje v podobě pokladních a bankovních dokladů.
- Pokladní a bankovní doklady jsou vystavovány na základě pohledávek a závazků. Ty jsou reprezentovány přijatými a vydanými fakturami. Jejich zpracování není sice v principu daňové evidence vyžadována, ovšem v případě kontroly vyslané státním institutem zvaným Finanční úřad, budou tyto doklady vyžadovány k předložení. Předpokládá se tedy jejich fyzické uskladnění. Evidence v systému následné vyhledání v mnohém usnadní. **Správa faktur** by měla být v systému obsažena.
- Faktury mohou být vydávány či přijímány dle objednávek. Jenomže **evidence objednávek** by pro relevantní informovanost vyžadovala mimo jiné i **správu skladových zásob**. Tyto moduly můžeme zařadit do seznamu potenciálních rozšíření daného systému.
- **Správu dlouhodobého majetku**, který má při výpočtu základu daně také svoji úlohu, některé systémy neumožňují. Ovšem zjišťování a vedení odpisů majetku je v pravdě velice složité a podléhá legislativním předpisům, proto by bylo vhodné počítat s tímto modulem v dalším rozšíření.
- **Správa bankovních účtů a pokladen** usnadňuje orientaci v evidovaných dokladech a také statistický přehled v jednotlivých agendách.

- Nastavení **předkontace**, tedy seznam druhů možných zaúčtovaných dokladů s ohledem na příjmy/výdaje, daňové/nedaňové dle české legislativy.
- **Personalistika a mzdová evidence** bude pro návrh zohledněna formou faktur přijatých od zaměstnanců.

b) požadavky na systém

- **Adresář kontaktů** pro vedení dodavatelů a odběratelů je sice doplňkovým (provozním) modulem, ale taktéž nezbytnou součástí systémů podobného zaměření. Bylo by vhodné, ba takřka nutné, jej zahrnout do návrhu.
- **Přístupová práva** pro víceuživatelský režim.
- **Reporty, tiskové sestavy** budou realizovány formou tisku zjednodušených a optimalizovaných výpisů z evidence.
- Přehledné **uživatelské rozhraní** v podobě strukturově jednoduchých a intuitivních internetových stránek.

Další moduly budou zmíněny v kapitole možných rozšíření.

4.2 Analýza požadavků

Jelikož by se dalo říct, že doklady vedené v peněžním deníku jsou posledním a nejdůležitějším článkem řetězce či dokonce stromu dokladů, musí být vedení samotných příjmů a výdajů navrženo s největší pečlivostí. V této kapitole se budeme zabývat jednotlivými oblastmi systému a uvedeme i atributy, které musejí být zpracovány. Nebudou zde však uvedeny všechny moduly zmíněné ve specifikaci požadavků, neboť některé přesahují rámec plánované realizace projektu.

Evidence příjmů a výdajů

- Číslo dokladu a druh zaúčtování, případně variabilní symbol
- údaje o firmě od které resp. pro kterou je platba určena
- uhrazovaný doklad (faktura atp., formou dopl. textu)
- okamžik zaúčtování (datum, čas)
- rozepsaná částka (částka + DPH) a částka celková
- definování doplňujícího textu
- kontrola korektnosti zadaných informací
- v závislosti na použitých modulech možnost přidávat a upravovat záznamy

Evidence pohledávek a závazků

- oddělené zpracování přijatých a vydaných faktur
- číslo faktury, variabilní symbol
- informace o dodavateli / odběrateli
- okamžik přijetí / vydání faktury
- druh faktury v závislosti na předkontaci
- částka celkem (rozepsaná)
- číslo souvisejícího dokladu (objednávka, příjemka, výdejka)
- kontrola korektnosti zadaných informací
- v závislosti na použitých modulech možnost přidávat a upravovat záznamy

Evidence dlouhodobého majetku

- kmenové údaje (název, inventární číslo, druh, členění, výrobní číslo, evid. Centrála, datum pořízení, druh původu, poznámka, pořizovací cena, zůstatková cena, atp.)
- údaje pro odpisy (způsob odpisu, odpisová skupina, uplatněný odpis)
- údaje - minulost (zařazení do evidence, atp.)
- celkový přehled cen
- kontrola korektnosti zadaných informací
- v závislosti na použitých modulech možnost přidávat a upravovat záznamy

Předkontace, typ účtování

- druh, název
- směr platby (příjem, výdej)
- typ DPH
- kontrola korektnosti zadaných informací

Evidence pokladních a bankovních dokladů

- typ dokladu (příjem, výdej, odchozí či příchozí platba)
- datum platby, vystavení
- druh účtového dokladu, předkontace
- doplňující informace
- nastavení dodavatele resp. Odběratele
- rozepsaná částka

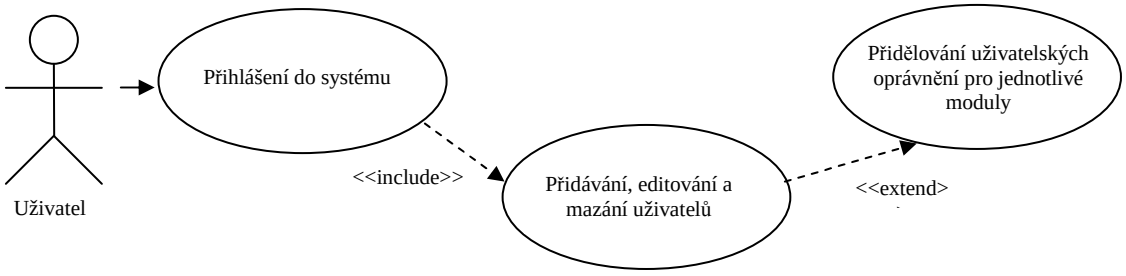
5 Plán projektu

5.1 Úvodem

Pojem samotné iterace zahrnuje několik fází, od specifikace požadavků, přes analýzu, návrh až po implementaci navrženého systému a jeho testování . Výsledkem každé z iterací je tedy funkční systém, který je v konečném důsledku možno nasadit u zákazníka. V kapitole Plán projektu se budeme zabývat specifikací jednotlivých iterací, neboť vstupem každého dalšího cyklu jsou požadavky ovlivněné výsledkem cyklů předchozích. V rámci semestrálního projektu byl realizován předběžný návrh formou diagramu případů použití. Nyní budou iterace přizpůsobeny a rozšířeny pro aktuální podmínky realizace. Deklarované iterace budou tři, z toho první dvě budou realizovány, poslední bude sloužit jako ukázka možného pokračování v rámci rozšíření.

5.2 I. iterace

První cyklus, řekněme prvotní zpracování systému, bude věnováno hlavní kostře aplikace. Kostrou je míněno rozhraní pro funkční komunikaci a běh všech potřebných modulů. Jak již bylo řečeno ve specifikaci požadavků, bude této funkčnosti věnována většina pozornosti, neboť se nám tímto usnadní jakékoli zpracování dalších rozšíření. Kostra systému je tzv. samotné rozhraní pro běh webové aplikace za použitých implementačních prostředků. Jelikož bude systém realizován formou tří-vrstvé architektury (více v kapitole zabývající se návrhem), zaměříme se především na základ logické (aplikační) a databázové vrstvy. Tedy realizace rozhraní pro komunikaci mezi výpočetní a perzistentní vrstvou systému. Jako odraz zpracování prezentační vrstvy bude implementace správy uživatelů, přidělování oprávnění pro jednotlivé moduly a samotné přihlašování do systému.



Obr. 3 Diagram případů použití, I. iterace

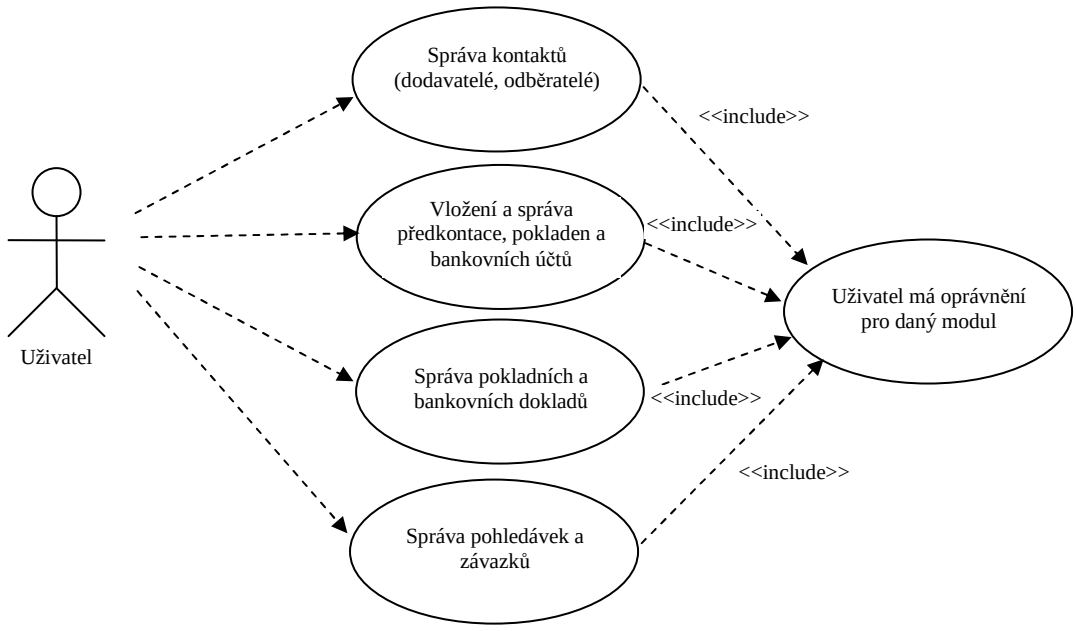
Dále pak připravení GUI pro jednotlivé agendy, především tedy formuláře pro zadávání dat uživatelem a výpisy jednotlivých agend do přehledných tabulek.

Uživatel – pro přihlášení a vytváření dalších uživatelů bude do systému v rámci instalace vložen přednastavený univerzální uživatel, řekněme administrátor, který bude mít oprávnění pro přidělování práv ostatním uživatelům. Ukázku použití znázorňuje Obr. 3

Tento první cyklus se částečně vymyká klasickému pohledu na vývoj systému, neboť jeho výsledkem sice bude funkční část aplikace, nicméně její funkčnost neodpovídá minimálním požadavkům na daňovou evidenci. V podstatě se tedy jedná o přípravu aplikace pro další rozšíření o požadované moduly.

5.3 II. iterace

V této fázi realizace bude systém doplněn o základní moduly dle specifikace požadavků pro daňovou evidenci. Těmi tedy bude *Adresář kontaktů*, *Peněžní deník* realizovaný formou dokladů evidovaných v agendách *Pokladna* a *Banka* a nakonec doplnění agendy pro správu *Pohledávek* a *Závazků* ve formě přijatých a vydaných faktur. Pro názornost poslouží Obr. 4



Obr. 4 Diagram případů použití, II. iterace

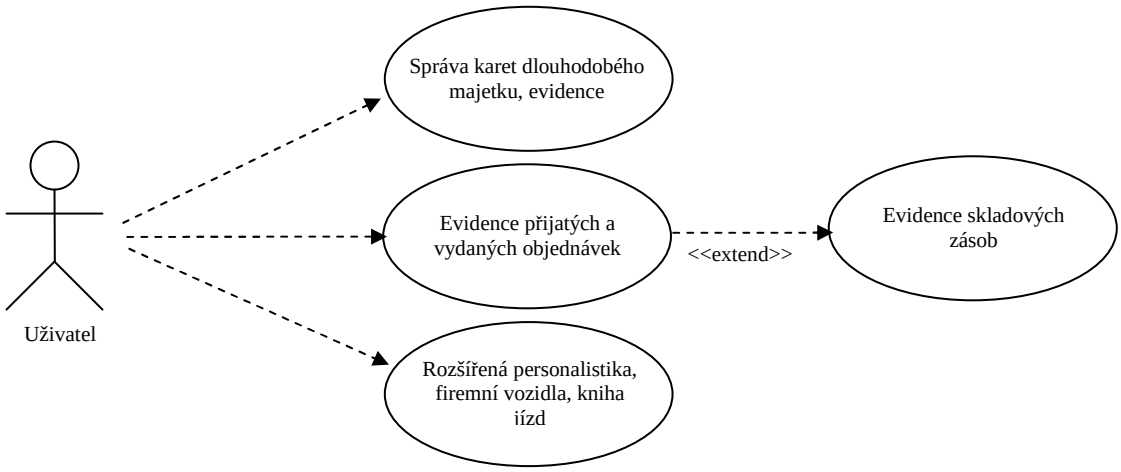
Uživatel – dle jednotlivých oprávnění má uživatel možnost manipulovat s daty uloženými v peněžním deníku. Vkládá a upravuje doklady reprezentující příjmy a výdaje v podobě pokladních či bankovních dokladů. V průběhu celého zúčtovacího období si může nechat zobrazit výpočet předběžné daně z příjmu.

Dále uživatel spravuje nastavení DPH podle typů spojených s aktuální legislativou. Dále může spravovat předkontaci, což je definování různých druhů zaúčtování příjmů a výdajů. Uživatelským nastavením je míněna možnost předdefinovat si některé počáteční hodnoty různých atributů pro zrychlení práce s peněžním deníkem (číselník atp.)

Může provádět určité úpravy související se správou kontaktů na úrovni odběratelů a dodavatelů.

5.4 III. Iterace

Jak již bylo zmíněno v úvodu této kapitoly, budeme se zabývat posledním, tedy třetím cyklem vývoje projektu. Jelikož se jedná o moduly, které jsou zařazeny do fronty možných vylepšení a rozšíření aplikace, nebude nutné příliš rozvádět jejich podobu. Zaměřme se výhradně na samotnou funkčnost.



Obr. 5 Diagram případů použití, III. iterace

Dlouhodobý majetek – evidence dlouhodobého majetku je relativně složitou záležitostí. Součástí vedení evidence je i její detailní rozpis včetně položek daného souboru.

Objednávky – správa objednávek je doplňkovým modulem, realizován bude pouze okrajově. Ve své podstatě bude řešen podobně jako agendy faktur. Tedy pro přehlednost rozdělen do agend přijatých a vydaných objednávek. Tento modul pak může rozšiřovat evidence skladových zásob pro přehledné zpracování a evidenci nabízeného zboží.

Personalistika - zahrnuje formální evidenci zaměstnanců, jejich pracovních poměrů, pojištění a nákladů na mzdy a pod.

Evidence firemních vozidel – správa firemních vozidel a s ní spojená kniha jízd pro evidenci dílčích spotřebních nákladů a detailů příslušného dlouhodobého hmotného majetku

6 Implementační prostředky

6.1 Úvodem

Předtím, než se budeme zabývat samotným návrhem informačního systému, bude vhodné se zmínit o nástrojích pro budoucí vývoj, jelikož jsou stěžejním subjektem udávající směr návrhu.

Kapitola o implementačních prostředcích nechť pojednává o teoretickém základu technologií, které budou použity v rámci realizace systému. Celková implementace v obecném měřítku bude založena na platformě Java J2EE 5.0. Dle zadání má být použito webové architektury Java Servlet a skriptovacího jazyka JSP.

Proč použít právě tuto technologii pro realizaci internetové aplikace? Odpověď je zcela nasnadě. Správná kombinace Servletů a JSP nám umožní vytvářet stránky jednoduchostí PHP a s možnostmi CGI. Přitom aplikace bude přenositelná a bezpečná. Ovšem hlavním důvodem je možnost použití celé webové aplikace jako vrstvy prezentační libovolné jiné aplikace. Je tím myšleno jednoduché propojení systému na platformě např. J2SE, které může bez problémů komunikovat právě se Servlety s využitím tzv. EJB komponent. To například při použití technologie PHP není ani zdaleka možné. Dále budou v této kapitole popsány Servlety a obecně i JSP, Apache Struts, což je framework pro vývoj webových aplikací, technologie hibernate pro transformaci objektů do struktury relační databáze a konečně systém řízení báze dat MySQL.

6.2 Použité technologie

6.2.1 Apache Tomcat

Tomcat je oficiální referenční implementace technologií Java Servlet a Java Server Pages (JSP), obojí vyvíjené pod záštitou společnosti SUN. Ve své stabilní řadě 4.1 Tomcat nabízí podporu pro Java Servlet verze 2.3 a JSP verze 1.2. Řada 5, která se nyní dostává na úroveň stabilní verze, nabízí podporu Java Servlet verze 2.4 a JSP verze 2.0. Mocným nástrojem se však server Tomcat stává až ve spojení s klasickým http serverem. Z důvodu kompatibility a praktických zkušeností byl zvolen webový server Apache 2.0. Hybridní konfigurace serverů Apache a Tomcat se nasazuje tehdy, je-li na aplikaci kladeno veliké zatížení jak na statické, tak i na dynamické dokumenty. Apache 2.0 je velmi rychlý a flexibilní webový server a v kombinaci s kontejnerem Tomcat nabízí stabilní platformu pro webovou Javu. Pokud však aplikace neobsahuje příliš mnoho dotazů na neměnné HTML stránky a

obrázky, je dobré přemýšlet o čistě javovské konfiguraci (tzn. pouze server Tomcat s konektorem Coyote).

6.2.2 Java Servlet a JSP

6.2.2.1 Servlet

Servlety jsou, zjednodušeně řečeno, programy napsané v jazyce Java. Jejich využití je hlavně v generování dynamických webových stránek, provádění operací a akcí na straně serveru a v lepší a hlavně pohotovější interakci se samotným uživatelem. Instance servletu je aplikace běžící na serveru, jež má za úkol zprostředkovávat komunikaci mezi webovým prohlížečem resp. libovolným klientem zpracovávající protokol HTTP a webovým serverem. Aplikace zpravidla obsluhuje požadavky klienta na práci s databází. Servlety provádějí tyto operace:

- čtou data odeslaná uživatelem např. pomocí formuláře nebo Java apletu
- zpracovávají podrobnější informace o požadavku, např. cookies, informace o prohlížeči atp.
- zpracovávají požadavky, komunikují s databází a vytvářejí výsledky
- formátují výsledky, např. do XML či obecně HTML
- odesílají výsledná data v podobě dokumentu zpět uživateli

Co se samotného provozu servletů týče, na serveru stále běží JVM (Java Virtual Machine), který interpretuje kompilované servlety (binární kód instrukcí pro JVM). Každý servlet běží stále, dokonce i po dokončení odezvy, tím je zaručena schopnost uchovat hodnoty a data pro případ, kdy by bylo vhodné je mít dostupné bez opakovaného výpočtu. Servlety mají široké spektrum datových struktur a metod pro práci s http požadavky a odezvami, strukturami cookies, monitorování session a html formuláři.

Servlety mohou mezi sebou sdílet data. To je v praxi používáno například pro sdílení připojení k databázi, předávání informací z požadavku na požadavek (vhodné např. pro sledování session specifické pro každého přihlášeného uživatele). Jsou prakticky přenosné. Díky jejich implementaci v Jave mohou bez větších problémů a úprav kódu fungovat na libovolné platformě či webovém serveru jež servlety podporuje. Jejich podpora může být zaručena přímo, nebo pomocí zásuvných modulů.

6.2.2.2 JSP

Servlety obsahují taktéž metody pro vytváření HTML do výstupního dokumentu. Pokud bychom ovšem ukládali statická data, kterých je v obecném dokumentu většina, stal by se zdrojový kód servletu krajně nepřehledným. Ne jen proto, ale také pro, alespoň částečné oddělení aplikační logiky od prezentační, bylo vyvinuto JSP (Java Server Pages). Ty nám umožňují skládat statické HTML s dynamicky generovaným obsahem. JSP se integruje do dokumentu v podobě specializovaných značek podobných HTML kódu. Ty vytvářejí a definují dynamickou část stránky. Fungují tak, že se při prvním zavolání dokumentu vytvoří servlet, ten se zkompiluje a zpustí. Dále se volá už jen ona zkompilovaná verze. Prvotní kompilace může o něco zpomalit odezvu serveru, což je ovšem zanedbatelné např. oproti PHP, kde se skripty kompilují a interpretují pro každé volání. Dynamická sekce stránek je psána v jazyce Java, jsou tedy programy v ní napsané použitelné jako komponenty programů jiných.

6.2.3 Apache Struts

Struts se snaží být co nejjednodušším webovým rozhraním. Nechává většinu práce a voleb na programátorovi. Pokud je tedy nutné vystavět webovou aplikaci „na zelené louce“, je Struts ideálním řešením. Skládá se minimálně ze tří částí - modelové (model), prezentační (view) a logické (controller). Prezentační vrstvu tvoří JSP dokumenty, které obsahují speciální Struts značky. Další možností jsou případně servlety. Logickou vrstvu tvoří tzv. Action třídy a modely jsou komponenty JavaBeans. Všechny vrstvy dohromady spojí konfigurační soubor struts-config.xml, který dále přidává některé drobnosti navíc, jako je připojení do databáze s JDBC2. V našem případě se naskýtá možnost vytvoření tzv. plug-in komponent pro nástroj Hibernate . Zde je tedy patrná zkratka MVC (Model-View-Controller). Celkově se tak jedná o rámec, který řídí tok aplikace, usnadňuje vícejazyčnou podporu, zpřehledňuje kód JSP dokumentů nebo provádí kontrolu vstupních údajů z webových formulářů.

Klíčovým prvkem frameworku Struts je výše zmíněné MVC (jinak také nazýván Model2), jež řídí kontrolní tok celé aplikace. Majoritní rozdíl mezi Modelem1 a Modelem2 je v tom, že každý z modelů zpracovává požadavky různými komponentami. V případě Modelu1 odpovídá za zpracování požadavků dokument JSP, který má taktéž na starosti zobrazení výstupu na straně klienta. Kdežto Model2 je tvořen třemi komponentami. Za prvotní zpracování požadavků zde odpovídá servlet (Controller). Ten po zpracování požadavků určí, který z JSP dokumentů bude dále volání obsluhovat a případně se zobrazí jako možná odezva. Komponenta Model odpovídá za správu vnitřní struktury. Tu reprezentuje komponenta View.

Nyní jen stručně o architektuře Struts. Úlohu Controlleru zvládá několik různých komponent. Definujme strom tříd *struts* pro zjednodušeni zápisu - *org.apache.struts.action*. Třída

struts.ActionServlet umožňuje směrovat http požadavky k jednotlivým částem frameworku. Mezičlánkem spojujícím požadavek s aplikační logikou je třída struts.Action. V potomkovi je možné převzít a upravit metodu execute, jež je volána komponentou Controller na základě nějaké definované akce. Akce se definují v konfiguračním souboru pro každou aplikaci zvlášť. Dále by bylo možné zmínit už jen třídu struts.RequestProcessor, která je řekněme odpovědná za analýzu a zpracování samotného požadavku. Další množství komponent nemá smysl v rámci této kapitoly popisovat, avšak dá se předpokládat, že se objeví v některé z kapitol popisující samotnou implementaci.

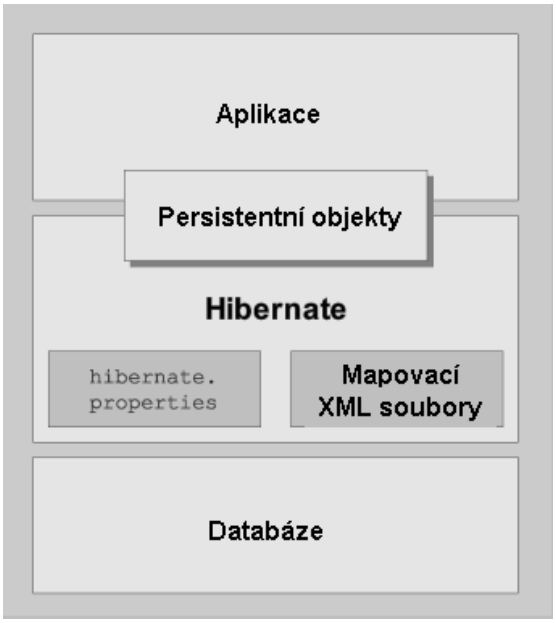
6.2.4 Hibernate

Zajímavý nástroj Hibernate se používá k zajištění perzistence dat. Ve skutečnosti to znamená, že jsou data uchována i po ukončení běhu aplikace. Můžeme zjevit několik vlastností. Jednou z nich je přístup současně několika uživatelů.k datům, tzn. že se jednotliví uživatelé vzájemně neblokují při čtení a manipulaci s daty. Jako další vlastnost by bylo dobré zmínit transakční přístup. Právě ten umožňuje vícero uživatelům pracovat současně nad stejnými daty bez obav z jejich porušení. Stará se o vždy ohroženou konzistenci dat. Poslední z vlastností by mohla být přenositelnost a relativní snadnost použití. Relačním databázím přichází vhod podpora objektově-relačního mapování.Ve své podstatě je to proces převodu (překladu) objektů na tabulkovou reprezentaci a převod vazeb mezi těmito objekty do tabulek dodatečných.Jak lze předpokládat, jedná se o nemálo jednoduchý proces.

Vývoj perzistentních objektů s využitím hibernate v jazyce Java splňuje vlastností jako jsou kompozice, asociace, polymorfismus, dědičnost. Dále práce s kolekcemi objektů. Hibernate umožňuje získávání databázových dat na základě vlastního plně objektově orientovaného jazyka nazvaného Hibernate Query Language (HQL), který je syntakticky velmi podobný SQL.

Hibernate, jakožto objektově relační nástroj, je řízen konfiguračními soubory ve formátu XML. V těchto souborech jsou uvedena mapování aplikovaná na jednotlivé tabulky daného databázového schématu na základní třídy. Na základe těchto mapovacích souborů je Hibernate schopen zajistit persistenci objektu. Mapovací soubory lze ale využít i při opačném postupu. Jsou-li nejdříve vytvořeny základní objekty a k nim odpovídající mapovací XML soubory, je možné z těchto souborů vygenerovat odpovídající databázová schémata.

Samotnou architekturu hibernate nejlépe znázorní obr. 6 [2].



Obr. 6 Architektura Hibernate

6.2.5 MySQL

MySQL je databázový systém, vytvořený švédskou firmou MYSQL AB. Je považován za úspěšného průkopníka dvojího licencování – je k dispozici jak pod bezplatnou licencí GPL tak pod komerční placenou licencí. MySQL je multiplatformní databáze.

Komunikace s ní probíhá – jak už název napovídá – pomocí jazyka SQL. Podobně jako u ostatních SQL databází se jedná o dialekt tohoto jazyka s některými rozšířeními. Pro svou snadnou implementovatelnost (lze jej instalovat na Linux, MS Windows, ale i další operační systémy), výkon a především díky tomu, že se jedná o volně šiřitelný software, má vysoký podíl na v současné době používaných databázích.. MySQL byla od počátku optimalizováno především na rychlost, a to i za cenu některých zjednodušení: má jen jednoduché způsoby zálohování, a až donedávna nepodporovalo pohledy, triggery, a uložené procedury. Tyto vlastnosti jsou doplňovány teprve v posledních letech, kdy začaly nejčastějším uživatelům produktu – programátorům webových stránek – již poněkud scházet.

Plnohodnotným databázovým systémem se všemi náležícími funkčnostmi se systém MySQL stal až od verze 5.x. S těmito vylepšeními ovšem poněkud ztratil svoji hlavní výhodu, a tou byla rychlost. Nová verze je proklamována jako plnohodnotná konkurence různým robustním komerčním systémům, avšak názory z praxe se poněkud rozcházejí. Záleží na účelnosti systémů, které MySQL využívají. Ovšem pro naše, v pravdě demonstrační, účely a podle nenáročností na operace, bude tento systém více než postačující.

7 Návrh

7.1 Úvodem

Vzhledem k výše zmíněným použitým prostředkům, především frameworku *Struts* a objektově-relační nadstavby systému řízení báze dat *Hibernate*, se v podstatě sama nabízí možnost přistoupit k tří-vrstvému návrhu informačního systému. Jedním z hlavních úkolů tedy nastává nutnost navrhnout rozhraní pro spolupráci Struts, který v daném systému zastupuje prezentační vrstvu a Hibernate prezentující vrstvu nejnižší, tedy databázovou. Toto spojení v rámci jedné aplikace není dosud obecně obvyklým počinem. Dle předpokladů bude toto rozhraní reprezentováno vrstvou aplikační, jež bude provádět převod mezi perzistentními objekty z databáze do formy vhodné pro prezentaci uživateli.

7.2 Databázová vrstva

Databázová vrstva je zde tedy reprezentována nástrojem pro převod objektů, nazývaných též *Entity* (více v kapitole zabývající se implementací), na relační strukturu databáze. Zabývat se zde návrhem logické struktury databáze tedy smysl postrádá, jelikož ji automaticky generuje systém *hibernate* v závislosti na definovaných entitách a použitých vazbách mezi nimi.

Součástí této kapitoly bude tedy popis a charakteristika některých nejdůležitějších entit, jež modelují reálnou podobu záznamů vedených například v *peněžním deníku* (pro názornost bez uhrazovaných dokladu typu faktura). Jsou rozděleny do balíčků, kterým náleží a které zastupují a zobecňují kupříkladu některou z agend či prezentuje systémový objekt, jako je např. balíček uživatelů s jejich oprávněními. Nejprve následuje výpis jednoduchých objektů s výčtem nejdůležitějších jejich vlastností. Objekty obsahují vlastností mnohem víc, pro zjednodušení výpisu však budou vynechány. Na obrázku *Obr. 7* je diagram modelující propojení těchto tříd.

Adresářový kontakt (AddressBookItem)

Popis: položka adresáře, samostatný kontakt

Atributy:

- jméno/příjmení
- pohlaví
- firma (oficiální název)
- kontaktní údaje pro firmu (rozvedeno do více atributů)
- pozice ve firmě (pozice kontaktu v hierarchii firmy, zaměstnání)
- typ (dodavatel/odběratel)

Účet (Account)

Popis: Objekt reprezentující bankovní účet nebo pokladnu, sjednoceno do společného objektu z důvodu většiny společných atributů

Atributy:

- typ (banka/pokladna)
- název (název banky, pojmenování pokladny)
- zkratka (reprezentační označení)
- číslo účtu (banka)
- popis

Předkontace (AccountAssignment)

Popis: Modeluje uživatelem definované typy zaúčtování pro peněžní deník

Atributy:

- typ (dle legislativy)
- název (uživatelsky přívětivé označení)
- zkratka (reprezentační označení)

Oprávnění (Right)

Popis: Objekt reprezentuje jedu položku z výčtu oprávnění uživatele

Atributy:

- číslo modulu (id modulu kterého se práva týkají)
- povolení číst
- povolení zapisovat (vkládat, upravovat, mazat)

Uživatel (User)

Popis: Objekt uživatel

Atributy:

- jméno/příjmení
- kontaktní informace (rozvedeno do více atributů)
- přihlašovací údaje (login, heslo)
- seznam *oprávnění* (seznam prav pro jednotlivé moduly)

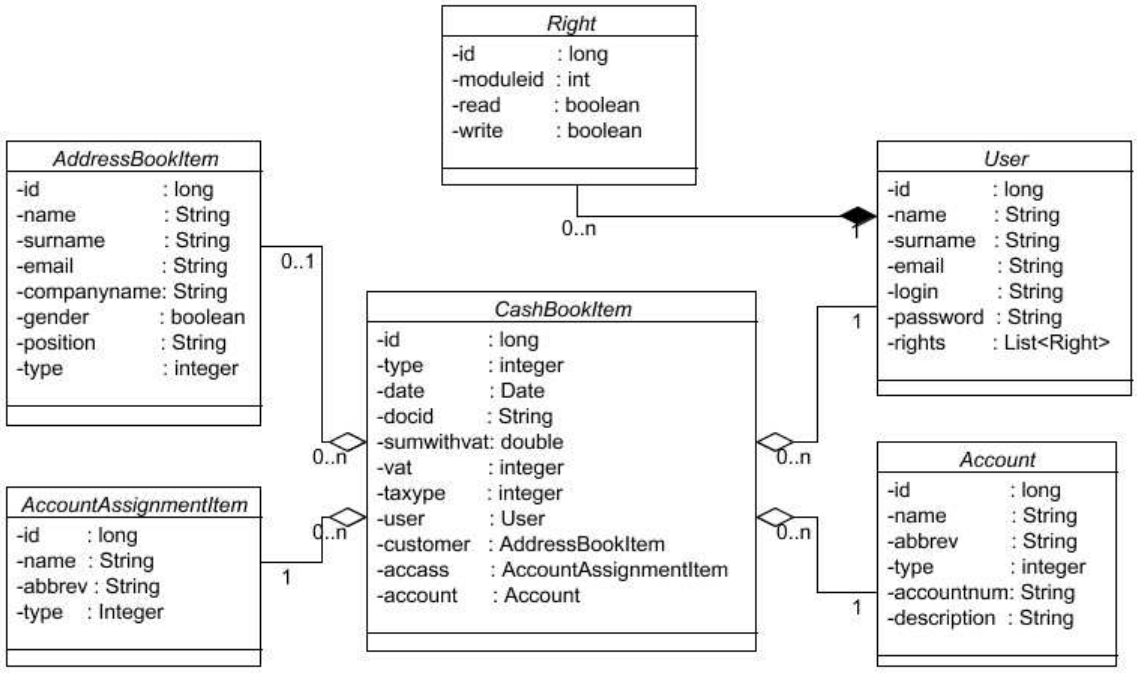
Doklad (CashBookItem)

Popis: Objekt reprezentující položku peněžního deníku, obsahuje v sobě všechny výše zmíněné entity

Atributy:

- typ (příjmový/výdajový)

- datum zaúčtování
- číslo uhrazovaného dokladu (číslo faktury, nepovinné)
- částka vč. DPH
- DPH (%)
- typ příjmu/výdaje (daňový/nedaňový)
- uživatel který položku vložil (*User*)
- dodavatel/odběratel (*AddressBookItem*)
- předkontace (*AccountAssignment*)
- účet (*Account*)



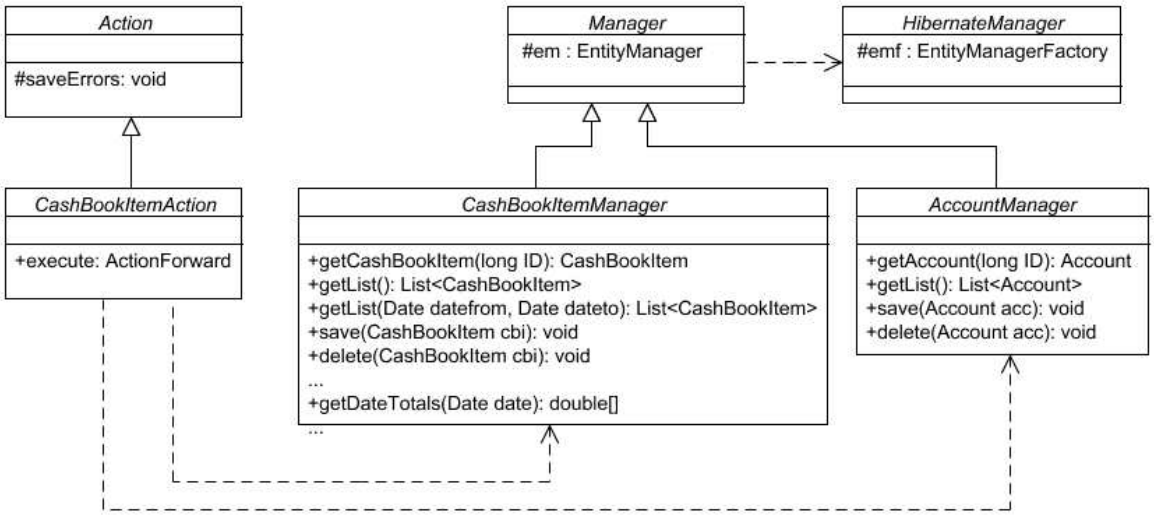
Obr. 7 Diagram persistentních tříd

7.3 Aplikační vrstva

Aplikační vrstvou tedy budeme rozumět rozhraní pro získávání uložených objektů z databázové vrstvy, manipulace s nimi a jejich opětovné uložení. Zde je třeba zmínit, že framework *Struts* do této vrstvy také zasahuje jednou ze svých tří částí (MVC), zvanou Model. Tato slouží jako rozhraní pro komunikaci s obecně logickou částí aplikace a zprostředkovává data zadaná uživatelem. Tyto data pochází například z webových formulářů, které jsou součástí webové prezentace, která je, dá se říct, výstupním produktem prezentační vrstvy.

Součástí této kapitoly tedy bude návrh správců entit. Jelikož tito sdílejí stejný přístup k persistentním objektům, budou všechny dílčí třídy správců vycházet z bázové třídy *Manager*, která komunikuje s nástrojem hibernate a má možnost získat tak prostředky pro přístup na úroveň báze dat. Tyto třídy obsahují souhrn metod, které spravují entity formou načítání jednotek, celých kolekcí těchto objektů, ukládání a mazání. Tento souhrn je společný pro všechny správce, avšak každý z nich může zacházet s daty po svém, adekvátně jejich obsahu. Dále pak jsou zde individuální metody pro ověřování a validaci jednotlivých informací a dat, které mají svým způsobem ovlivnit vlastnosti daného objektu a to povoleným způsobem.

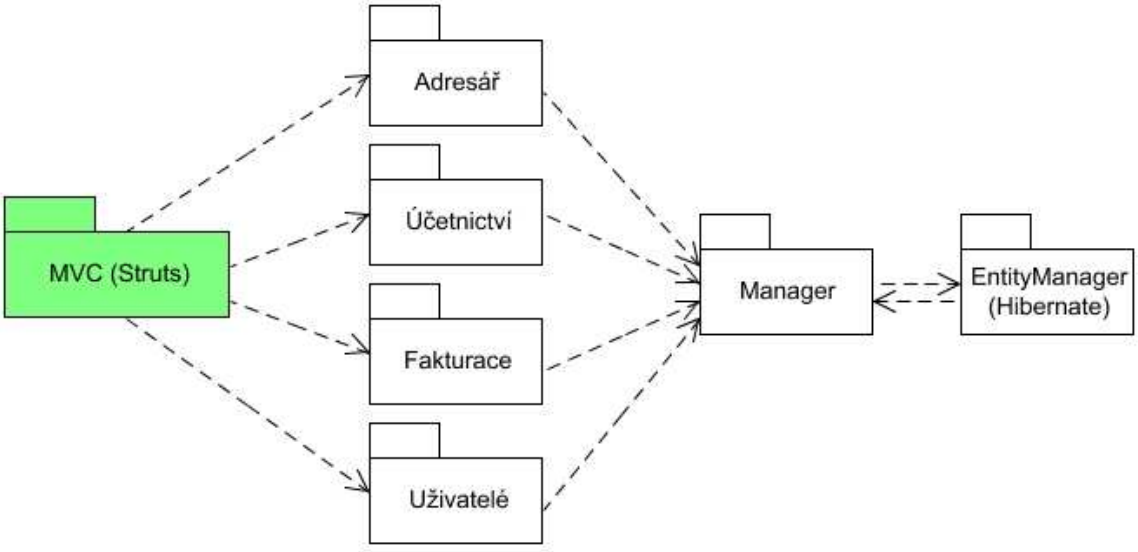
Tyto třídy se ve své podstatě opakují, nemělo by tedy smysl popisovat správce pro každou entitu zvlášť. Diagram na obrázku *Obr. 8* prezentuje hierarchii tříd správců s ukázkou návaznosti na třídu *org.apache.struts.action.Action*, která je součástí frameworku *Struts*. Ta obsahuje pouze metodu *execute*, která je volána právě na základě definované akce v nastavení. A právě metoda execute je bodem, odkud se volají správci entit, provádí se manipulace s daty a vrací se výsledek pro stránku, která je definovaná jako návratová. Těchto návratových stránek může být definováno libovolné množství a je tedy na výsledku prováděné operace, na které místo systému je požadováno vrácení výsledků. Jelikož se jedná o přesměrování stránky na stránku výstupní dle výsledku, je nutné uchovat případné návratové hodnoty (chybové hlášení apod.), tedy udržet kontext mezi stránkami. Zde je možno využít metodu *saveErrors* třídy *Action*, která umožňuje toto dočasné uchování chybových hlášení formou serializací a přenosu mezi stránkami.



Obr. 8 Diagram části aplikační logiky systému

Na výše zobrazeném diagramu je vidět výhoda použití podobného rozhraní, neboť návrháři resp. vývojáři usnadňuje práci s implementací součástí systému, které závisí na procesech, jež jsou již

optimalizované letitými zkušenostmi. Je vhodným základem pro další rozšíření a umožňuje přistupovat k systému obecně jako množině balíčků, které je možno jednoduchým způsobem doplňovat či naopak odebírat. Stylizovaný pohled na aplikaci umožňuje *Obr. 9*, kde je možno vidět tzv. package diagram jednotlivých modulů aplikace.



Obr. 9 Package diagram modulů aplikace

Jednotlivé moduly jsou řekněme sumarizujícím balíkem, obalem, který v sobě může obsahovat libovolné množství dalších modulů. To se týká balíku *účetnictví*, který se skládá z dalších podčástí, jako je samotný *peněžní deník*, dále pak správa nastavení *předkontace* a v neposlední řadě správa *účtů* a *pokladen*. Balíček *uživatelé* obsahuje logiku nejen pro správu uživatelů jako takových, ale také kontrolu zaslaných přihlašovacích údajů a jejich vyhodnocení.

Je vhodné umístit části systémů do tématických balíčků, využijeme tak jedny z hlavních výhod objektově-orientovaného přístupu k návrhu a to je skutečné fyzické i logické oddělení jednotlivých částí systému. Otevírá se tak možnost teoretické spolupráce v týmu, což je v našem případě čistě hypotetickou úvahou. Především je však připraveno pole pro budoucí rozšíření systému o další případné agendy, či libovolné moduly ku zákazníkově spokojenosti.

Návrh aplikační logiky taktéž počítá s odchytem výjimek generovaných napříč jednotlivými vrstvami systému. Zpracovává výjimky odchycené z nižší, tedy databázové vrstvy a transformuje je do uživatelsky příjemnější formy, detekuje chyby vstupního charakteru, tedy nekompletnost či nekorektnost zadaných informací, které jsou zaslány z front-endu.

7.4 Prezentační vrstva

Pro uživatele nejdůležitější součást systému, jediný styčný bod pro komunikaci se systémem. Jelikož při použití frameworku *Struts* odpadá návrh tříd a obecně objektů pro zpracování a rozřazování vstupních požadavků, pozornost bude zaměřena především na návrh uživatelského rozhraní, dle specifikace. Jak vyplývá již ze samotného zadání, důraz bude kladen především na logickou stavbu zaručující uživateli snadnou orientaci v systému a intuitivní přechody mezi jednotlivými agendami či obecně částmi aplikace. V této kapitole bude taktéž znázorněno modelové použití rozhraní a jeho možností formou abstrahovaného diagramu přechodů mezi jednotlivými obrazovkami i v závislosti na zvoleném vstupu i v případě vzniku nějaké chybové situace. Výše zmíněný framework umožňuje definovat libovolné množství návratových stránek rozlišených dle vůle implementátora, proto je tak možné konstruovat rozsáhlé a provázané modelové použití systému. V našem případě se bude jednat o zjednodušenou verzi systému pro modelovou názornost použití.

7.4.1 Uživatelské rozhraní

Návrh uživatelského rozhraní bude pojat ze dvou úhlů pohledů. Funkčního a implementačního. Implementační pohled nebude příliž rozsáhlý, neboť dle předpokladu využijeme pro realizaci značkovací jazyk HTML, případně jeho normované modifikace. Zmíníme jej tedy na prvním místě.

7.4.1.1 Implementační pohled

Jak již bylo řečeno, pro implementaci uživatelského rozraní máme k dispozici značkovací jazyk HTML. Ten však sám o sobě příliš mnoho zajímavých komponent nenabízí. Komponentou jsou míněny například prvky formuláře, tlačítka a podobně. Oproti možnostem implementace GUI v nějakém vyšším programovacím jazyce, máme k dispozici poněkud omezené spektrum prvků. Proto se může na první pohled zdát, že interaktivita systému notně ztrácí na dynamičnosti. Je však možné tento zdánlivý nedostatek využít v náš prospěch a to ve smyslu jednoduchosti a přímočarosti. Příliš mnoho prvků různých funkčností může na uživatele působit naopak negativním způsobem a ve snaze práci se systémem ulehčit ji naopak představuje jako předmět obav z nekorektního zacházení a stává se tak uživatelsky nepřívětivá. Některé nedostatky tohoto druhu byly zmíněny v kapitole analýzy již existujících produktů podobného zaměření na našem trhu. Proto se v návrhu budeme snažit aplikaci po uživatelské stránce co nejvíce zjednodušit a provázat tak, aby si uživatel mohl být naprosto jist svým počínáním bez nutnosti odborného školení a bez obav ze ztráty dat či jejich nechtěného poškození.

Tato tématika však již bude podrobněji rozebrána v následující kapitole zabývající se funkčním pohledem na návrh uživatelského rozhraní.

7.4.1.2 Funkční pohled

Více méně teoretické pojednání o předmětu navrhovaného uživatelského rozhraní zmíněném v předcházející kapitole se nyní budeme snažit převést do reálné praxe. Pro jednoduchost modelu návrhu a tím pádem menšího zatěžování čtenáře zde bude demonstrován návrh rozhraní pro evidování dokladů v peněžním deníku, což je sama o sobě podstata podobného systému. Návrh proběhne definováním jednotlivých obrazovek seřazených za sebou tak, jak je uživatel uvidí v při práci se systémem.

Rámec (pracovní plocha)

Nejprve by bylo vhodné charakterizovat obecný rámec, tedy prostředí, ve kterém se bude uživatel pohybovat. Mělo by se jednat o standardní rozvržení pracovní plochy, kde bude pokud možno vše tak říkajíc „na dosah“. Systém bude obsahovat dva typy uživatelského menu.

- *Hlavním menu*
    - přímé odkazy do jednotlivých agend, resp. balíčků agend, které jsou zastřešeny souhrnným pojmenováním. Toto menu je viditelné pro přihlášeného uživatele za všech okolností všude v systému, není tedy problém rychlý přesun kamkoli po aplikaci.
    - položky menu jsou zobrazovány v závislosti na povolených modulech správcem systému.

- *Kontextové menu*
    - položky kontextového menu jsou individuální pro jednotlivé oblasti resp. balíčky aplikace
    - jsou zobrazeny v rámci vnitřního panelu podobným stylem jako hlavní menu
    - jejich zobrazení je taktéž podmíněno oprávněním pro každý z modulů v daném balíčku

Hlavní částí pracovní plochy je tedy hlavní panel situovaný v samotném centu pracovní plochy, je tedy k dispozici celá šířka plochy důležitá pro výpis dat podobného charakteru, kde budeme chtít vměstnat co nejvíce dílčích informací k nahlédnutí na první pohled. Dále se budeme zabývat jednotlivými obrazovkami, tedy možným obsahem hlavního panelu.

Přihlašovací obrazovka

Pokud uživatel vstupuje do systému v daném sezení poprvé, je vyzván k zadání svých přihlašovacích informací nutných k ověření jeho systémové totožnosti a nabídnutí možností práce dle přidělených oprávnění. Jelikož se jedná o internetovou aplikaci a informace evidované v systému jsou přísně soukromého charakteru, je úvodní přihlášení do systému bezpochyby nutností. Zabezpečení vstupu aplikace tedy nabízí, ovšem ošetření rizik uživatelského/personálního charakteru už bude na bedrech administrátora. Přihlášení tedy vyžaduje tzv. loginu a hesla. Prvky formuláře tedy budou:

- *Login* – pole pro přihlašovací jméno uživatele
- *Heslo* – pole pro zadání hesla, jeho čitelnost však nebude z bezpečnostních důvodů umožněna
- *Přihlásit* – jednoduché tlačítko pro odeslání informací do systému

Dalším důležitým prvkem je výpis případných chybových hlášení uživateli, tím se zaručí aktuální informovanost uživatele. Chyboví hlášení budou pravděpodobně typu „nepravdivých informací“ nebo nevyplnění „povinných polí“.

Úvodní obrazovka (statistiky)

Obrazovka, kterou uživatel uvidí vždy po přihlášení. V případě, že do systému vstupuje v rámci jednoho sezení, bude na tuto obrazovku automaticky přesměrován. Zobrazí se i uživatelům, kteří by neměli oprávnění pro žádné další moduly. Měla by obsahovat tyto souhrnné informace:

- *Souhrnná bilance*
    - informace pro zákazníka o aktuálních příjmech a výdajích rozdělená do tří období
        - dnes – bilance pro aktuální den
        - měsíc – souhrn za zvolený měsíc, přednastaven aktuální
        - rok – příjmy a výdaje za celé zúčtovací období
    - pro každé období rozepsány celkové příjmy, výdaje a jejich součet
    - uživatel bude mít možnost zvolit si jednoduchým způsobem měsíc i rok pro daný výpis, tyto 2 položky jsou provázány. Pouze první položka *dnes* zůstává statická

- *Stav financí*
    - informace o aktuálním stavu financí na hotovostních pokladnách
    - informace o aktuálním stavu financí na bankovních účtech
    - tyto informace také podléhají aktuálně zvolenému roku

- *Osobní informace*
    - datum a čas posledního přihlášení, pro možnost kontroly
    - IP adresa, ze které bylo provedeno poslední přihlášení, pro zkušenější uživatele umožňuje kontrolu přístupu do svého účtu

Účetnictví

Balíček zaštiťující obecně peněžní deník, správu pokladních a bankovních dokladu, ze kterých se peněžní deník primárně skládá. Obsahuje kontextové menu pro navigaci mezi danými subpoložkami. Pokud má uživatel oprávnění pro tento modul, zobrazí se mu implicitně deník. Ten slouží pro rychlou orientaci v dokladech a sumarizuje příjmy a výdaje v něm vedené. Rozhraní by mělo obsahovat možnost filtraci položek v deníku dle data, např. výpis od-do apod. Samotný peněžní deník by nemělo být možno přímo editovat. Uživatel bude moci kliknutím na ikonku reprezentující tisk vytisknout aktuální i filtrovaný přehled. Zobrazí se stránka pouze s položkami deníku bez ostatních funkčních i designových prvků.

Peněžní deník

Výpis dokladů evidovaných v pokladním a bankovním modulu.

- nebude možnost zasahovat do evidence, pouze její filtrace a případně tisk.
- výpis bude obsahovat souhrnné sumy příjmů a výdajů za zvolené období
- zobrazení vypočteného základu daně z příjmu z daňových příjmů a výdajů
- tabulka evidovaných dokladů by měla mít tyto sloupce:
    - datum zavedení dokladu do systému
    - číslo uhrazovaného dokladu
    - popisný text
    - příjmy a výdaje bez DPH
    - částku DPH
    - celkový příjem/výdaj s DPH
    - zkratka účtu/pokladny, která eviduje doklad, zde bude umožněn uživateli, který má oprávnění editovat evidenci bankovních účtů a pokladen, proklik do této evidence a její případná editace
    - výpis typu příjmového/výdajového dokladu (daňový, nedaňový)
    - zkratka typu zaúčtování, zde bude umožněn uživateli, který má oprávnění editovat předkontace, proklik do tohoto nastavení a jeho případná editace

Banka/Pokladna

Tyto moduly se v zásadě liší pouze v použitém typu dokladu, tedy pokladní doklad, nebo bankovní výpis, proto bude jejich návrh společný. Pro jednodušší kontrolu při zpracování bude možnost přímo zvolit, zda si uživatel přeje vložit příjmový, nebo výdajový doklad. Tím se otevře formulář který bude obsahovat pole a hodnoty právě pro zvolený typ dokladu.

- formulář pro vložení příjmu/výdaje bude obsahovat tyto prvky:
    - číslo dokladu, nebo možnost zvolení jedné z přijatých/vydaných faktur. Tato volba je vzájemně výlučná, proto bude možno vyplnit pouze jednu z možností
    - volba účtu/pokladny
    - datum, ke kterému se doklad vztahuje
    - typ zaúčtování/předkontace
    - doplňkový text, popisující informace
    - volba odběratele/dodavatele. Tato volba bude přístupná pouze v případě, že se nebude vkládat uhrazovaná faktura, dodavatel/zákazník se pak převezme právě z dané faktury
    - částka s DPH a současně zadat DPH, ke kterému se částka vztahuje
    - typ – daňový/nedaňový příjem/výdaj

- tabulka s výpisem již evidovaných dokladů bude mít stejné sloupce jako samotný peněžní denník, navíc však bude obsahovat ikonky pro editaci a smazání záznamu v případě, že má uživatel k těmto úkonům oprávnění
- pro filtraci záznamů v těchto agendách bude k dispozici stejný formulář jako je v peněžním deníku.

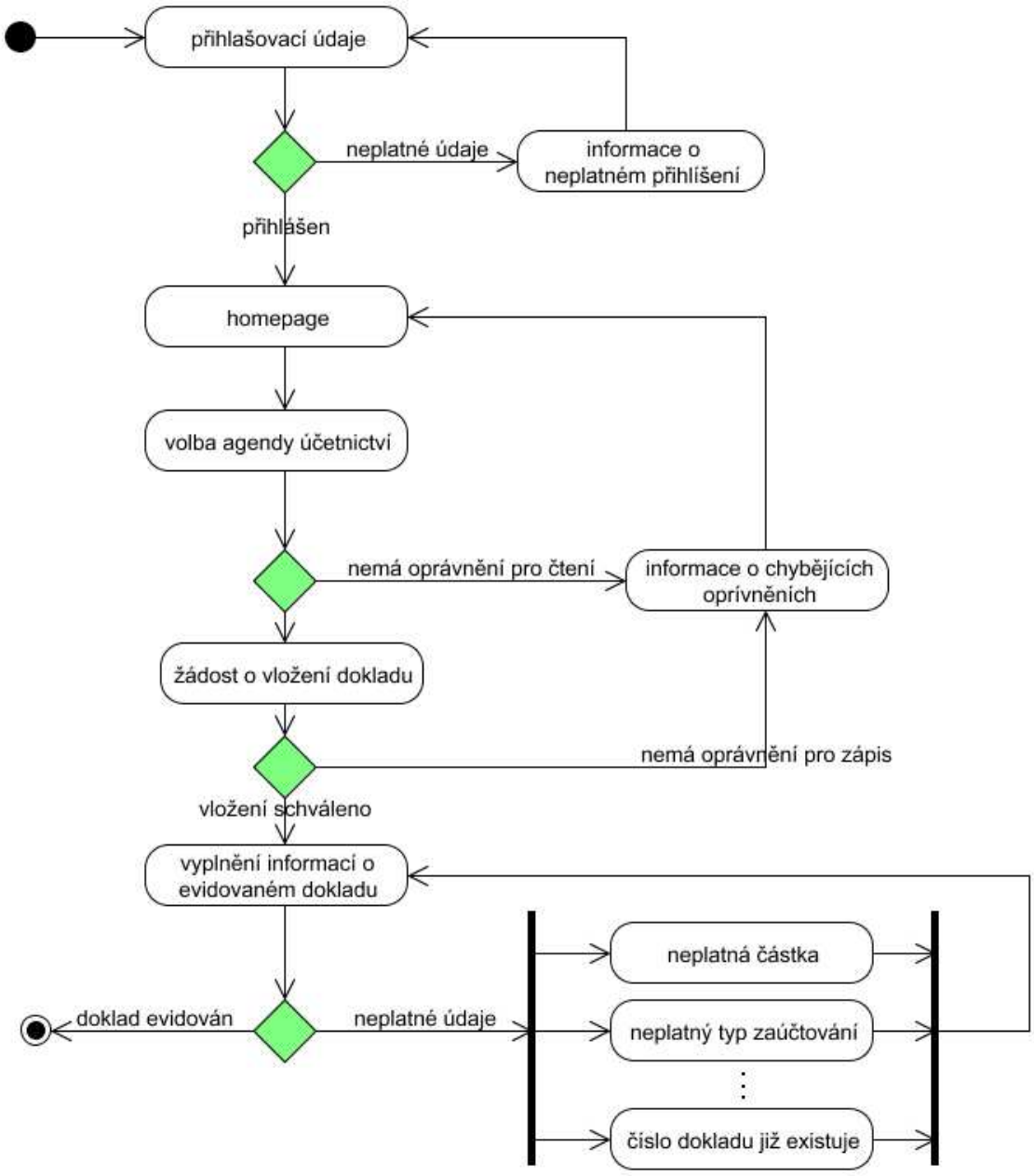
Další obrazovky obsahující formuláře a výpisy pro ostatní agendy není nutné zmiňovat, neboť jsou ve své podstatě podobné těm, co již byly prezentovány. V kapitole Případy použití bude popis zpracování jednotlivých formulář v interakci s uživatelem, demonstrace proběhne na případu vložení dokladu do peněžního deníku.

Uživatelé

Modul pro správu uživatelů systému můžeme v návrhu zmínit, neboť se svým způsobem liší od ostatních agend, zastává úlohu spíše systémového charakteru. Formulář pro vkládání a editaci bude obsahovat typické povinné informace včetně přihlašovacího jména a hesla. Navíc pak bude součástí i tabulka s výpisem jednotlivých modulů a možností zatrhnout typ oprávnění k nim se vztahující (čtení/zápis).

7.4.2 Modelový případ použití

Diagram návaznosti obrazovek bude hlavním předmětem této kapitoly. Jako modelový příklad interakce uživatelského rozhraní bude použit postup uživatele aplikací až po úspěšné zaevidování dokladu do peněžního deníku. Diagram na *Obr. 10* tedy modeluje stavový diagram případu použití pro zaevidování dokladu.



Obr. 10 Stavový diagram

Následuje popis jednotlivých částí diagramu. Jedná se o modelový příklad, proto bylo zadávání zobecněno, nezmiňujeme se o příjmech, výdajích, pokladně, bance a pod, tím by se samotný diagram rozrostl na dvojnásobek.

Prvním krokem je tedy samotné přihlášení uživatele do systému. Pokud by chtěl uživatel přistoupit na libovolnou interní stránku bez přihlášení, bude automaticky přesměrován do přihlašovacího formuláře. Následuje tedy odeslání vyplněných dat a validace. V případě nekorektně zadaných údajů bude uživatel přesměrován zpět na formulář a informován o neplatnosti jména nebo hesla. Pokud přihlášení proběhne úspěšně, bude přesměrován na úvodní stránku systému.

Úvodní stránkou (*homepage*) jsou míněny statistiky definované v kapitole [7.4.1.2]. Můžeme ji nazvat jednoduchým rozcestníkem. Volbou položky *Účetnictví* z hlavního menu bude uživatel přesměrován do agendy peněžní deník. Zde si z kontextového menu vybere jednu z agend *Pokladna*/*Banka*. Pokud nemá oprávnění na *čtení* zvolené agendy, bude informován a přesměrován na úvodní stránku.

Při pokusu o otevření formuláře pro vložení dokladu je proveden test na platnost oprávnění k zápisu pro danou agendu. Pokud uživatel toto oprávnění nemá (pokud nemá oprávnění, nezobrazí se odkaz na vložení dokladu, modelujeme případ zadání URL ručně), dojde k přesměrování na úvodní stránku a výpisu informačního textu o neexistenci oprávnění pro danou akci. Po úspěšném ověření práv se uživateli zobrazí formulář pro definování dokladu k evidenci.

Po odeslání dat z formuláře dojde k jejich ověření. Pokud data neprojdou ověřovacím testem, bude uživatel přesměrován zpět do formuláře a vyzván k jejich opravě tam, kde se vyskytla nějaká nesrovnalost. Jakmile data projdou validací, proběhne uložení, přesměrování uživatele do výpisu dané agendy a informaci o korektním zápisu do evidence.

Přibližně stejný postup je aplikován také v ostatních agendách systému, není tedy nutné se o nich významněji zmiňovat. Uživatel bude kontrolován při vstupu do každé sekce aplikace, v případě, že nemá oprávnění pro zápis pro některý z modulů, nebudou v tomto zobrazeny tlačítka ani odkazy k provedení jakékoli akce, která by měla za následek změnu v datech. Dalším bezpečnostním omezením bude automatické odhlášení uživatele při nečinnosti delší než několik minut. Pokud bude chtít takto automaticky odhlášený uživatel opět pokračovat v práci v systému, zvolením kterékoli z agend či úvodní stránky dojde k přesměrování na přihlašovací formulář. Také při pokusu o smazání jakéhokoli záznamu bude uživatel nejprve vyzván k potvrzení dané akce, aby nedošlo k nechtěnému odstranění důležitých informací.

8 Implementace

8.1 Úvodem

V kapitole nazvané implementace se budeme podrobněji zabývat procesem výstavby aplikace jež je předmětem projektu. Odhalíme některé vlastnosti použitých technologií s ohledem na jejich využitelnost obecně pro vývoj aplikací a pak s praktickým příměrem pro naše účely. Jelikož jsou zajímavé technologie na obou okrajových vrstvách sytému, nezbývá než se jimi zabývat odděleně a jejich popis opět rozdělit do tří oblastí odpovídajících rozložení jednotlivých vrstev aplikace, podobně jako tomu bylo při návrhu. Než se pustíme do řešení implementace pro konkrétní části, byla by škoda, nezmínit se obecně o prostředí, ve kterém byla aplikace postupně realizována.

Jak již vyplývá ze samotného zadání, implementačním jazykem byla zvolena Java. A to hned z několika důvodů. Jedním z nich je její obrovská univerzálnost se širokým spektrem možného využití nejen pro účely realizace samotného projektu, ale také v dalším pokračování a jeho případného rozšíření. Dává implementátorovi skutečně volný prostor k řešení dílčích problémů, je tedy opravdu na libovůli uživatele, jakým způsobem se daného úkolu zhostí. Forma webová aplikace, jež je předmětem práce, je jen jedním z možných způsobů využití, které nabízí. Dalším z nich je pochopitelně masová rozšířenost s velkou uživatelskou základnou, která mimo jiné zaručuje dostatečnou podporu a dostupnost studijních materiálů ve formě publikací či nejrůznějších dokumentů a tím pádem i případné pomoci při řešení problému spojených se samotnou implementací. Od toho se odvíjí též velké zázemí pro realizaci všemožných podpůrných nástrojů s oblastí zaměření od návrhu až po konkrétní běh systémů. V neposlední řadě hlavních důvodů je platforma předmětem požadavku na rozšíření vzdělání od samotného autora této práce.

Jedním z nejpoužívanějších nástrojů pro realizaci libovolné aplikace v jazyce Java je vývojové prostředí NetBeans. Bylo použito i pro tento projekt. Jedná se o klasické IDE (Integrated Development Environment), což je balíček nástrojů integrovaných v jedné aplikaci, která maximalizuje usnadnění vývoje podobných systému a umožňuje snadnou orientaci nejen v souborovém systému daného projektu, ale hlavně ve zdrojových kódech samotných. Urychluje psaní zdrojových textů pomocí tzv. *code invocation*, tedy nabízením možných objektů nejrůznějších knihoven pro jednotlivé části především díky vynikajícímu *class browser* modulu, který udržuje v povědomí třídy, jejich vlastnosti a metody včetně případné dokumentace a nabízí je na vhodných místech. Naproti tomu je z těchto důvodů relativně náročný na systémové prostředky, hlavně operační paměť, které ke svému provozu a způsobu pozvolného uvolňování paměti ve formě *garbage collector*u za obecné neexistence destruktorů, spotřebuje značné množství. Není možné opomenout zmínit se o tom, že celý systém je i přes svoji robustnost volně šířitelným produktem s neomezenými možnostmi využití i pro komerční účely. Pro vývoj tohoto projektu bylo prostření *NetBeans* zvoleno

především kvůli možnosti podpory pro celou platformu J2EE (Enterprise Edition), která je nutná pro vývoj webových aplikací v jazyce Java. V rámci tohoto balíčku rozšíření obsahuje vývojové prostředí i vlastní webový server *Tomcat* na kterém je pak výsledná aplikace snadno testovatelná jak v reálném tak i debug módu a veškeré systémové hlášení jsou přesměrovány zpět do vývojového prostředí a při detekci např. neošetřených výjimek je uživatel nasměrován přímo na místo vzniku s návrhem na jejich odstranění. Jelikož jsou JSP dokumenty (Java Server Pages) ve skutečnosti skriptovacím jazykem, který je při kompilaci konvertovaný na Servlet, je někdy třeba tyto kompilované jsp dokumenty kontrolovat pro snadnější odhalení chyb. Ovšem ty jsou uloženy v hloubi adresářové struktury webového serveru, proto k nim vývojové prostředí umožňuje snadný přístup. Proces od kompilace přes umístění aplikace na server je kompletně automatizované, aplikace se i spustí v přednastaveném webovém prohlížeči. Obecně nemohu toto prostředí než doporučit.

Adresářová struktura odpovídá jednotlivým balíčkům, které Java přímo podporuje a jsou tedy jednoduchou generalizující abstrakcí pro moduly implementované v systému.

8.2 Databázová vrstva

Prostor této kapitoly bude využit pro popis implementace a využití nástroje *Hibernate* pro persistenci entit. Hibernate umožňují několik možností nastavení. Všechna jsou realizována formou XML dokumentů. V našem případě se jedná o dokument *persistence.xml*, definice tzv. *persistence-unit*, ve kterém jsou definovány jednotlivé Entity, tedy objekty, pro které bude generována a spravována databázová struktura. Dále obsahuje nastavení připojení a určení ovladačů pro připojení k databázovému serveru. Systém je tedy plně nezávislý na použitém databázovém enginu a naprosto tak splňuje kritéria pro oddělení systémových vrstev. V našem případě se jedná o server *MySQL* , taktéž volně šířitelný engine bez dílčích omezení pro jeho použití. Avšak krom vytvoření samotné databáze není díky hibernate nutné se dále o databázi starat. Vše je zprostředkováno právě tímto nástrojem.

Celé rozhraní pro komunikaci s persistenčním nástrojem je umístěno ve třídě *danev.hibernate.HibernateManager* (definováno i se strukturou balíčků). Zde je prostředek pro vytvoření spojení na základě definičního xml dokumentu *persistence.xml* kde je definována persistentní jednotka použitá právě pro provoz dané aplikace. Těchto jednotek je možno vytvořit neomezené množství a přistupovat tak současně k několika databázovým serverům současně bez nutnosti ukončování spojení předchozích.

Persistentní entity, tedy objekty jež budou ukládány a spravovány nástrojem hibernate, mají svojí specifickou syntaxi. Není zde prostor pro ukázku konkrétního zdrojového kódu, ten je možné ve zkrácené verzi vidět v příloze [č. 1]. Syntaxí je jsou míněny speciální značky uvedené znakem @, které definují konkrétní vlastnosti a účel jednotlivých atributů a způsob zacházení s nimi. Jedná se o

tzv. mapování na relační model. *@Entity* například upozorňuje, že se jedná o persistentní objekt, je to jedna z mála povinných *Entity Beans*, což je souhrnný název pro všechna podobná značení. Můžeme definovat název tabulky, do níž bude objekt uložen, není to však povinné pole, hibernate v případě neuvedení této informace přidělí vygenerovaný systémový název. Jednotlivé sloupce, tedy atributy objektu se také mapují na konkrétní názvy s parametry např. definující podmínky integritních omezení, např. že pole nesmí být prázdné, nebo jeho hodnota musí být jedinečná apod. Speciální definice se týká také sloupce s identifikátorem, pro které je možno nastavit hodnoty pro vlastnosti primárního klíče, nastavení číselné řady indexů, autoinkrementace apod. Datové typy sloupců odpovídají možným datovým typům jež jsou k dispozici např. ve standardních knihovnách, nebo je možno použít typy reprezentované vlastními objekty. Pokud se jedná o objekty jiných persistentních entit, je nutné definování vazeb. Vazby jsou čtyř druhů (*OneToOne, OneToMany, ManyToOne, ManyToMany*) a každý z typů má svoje specifické nastavení. V projektu byly použity vazby 1:N, pro realizaci propojení tabulek uživatelů s jejich oprávněními a N:1 pro ukládání vazeb jednotky peněžního deníku na objekty s ním provázané (typ předkontace, účet, apod.). Více informací je možno nalézt ve zmíněné příloze.

Celkem je tedy v projektu realizováno 7 persistentních objektů, které jsou mezi sebou provázány na podobném principu, jak je možné vidět na diagramu *Obr.7* [7.2]. Na diagramu není vidět objekt pro fakturaci (InvoiceItem), který využívá některé z objektů podobně jako *CashBookItem.* Pro získávání jednotlivých objektů z databáze je možné použít vestavěných metod, které pracují na základě znalosti primárního klíče záznamu a je tedy jednoduché se na tento objekt odkázat přímo. Pro případ, že neznáme OID, je zde k dispozici HQL (*Hibernate Query Language*), což je jazyk ne nepodobný klasickému SQL, ale podporuje objektový přístup již na úrovní formulace dotazu. Je zde určitá analogie ve způsobu dotazování, avšak pro získání hodnot zde odpadají procedury a operace pro spojování tabulek apod. neboť formulujeme dotaz jako bychom se dotazovali přímo na konkrétní objekt, kterého vlastnost je objekt jiný. Pro názornost uvedu ukázku HQL dotazu použitého právě pro získání objektu CashBookItem na základě hodnoty atributu objektu Account.

```
select i from CashBookItem as i where i.account.atype = '2'
```

Dotaz je použit pro získání těch evidovaných záznamů z peněžního deníku, které jsou přiřazeny účtu typu *Banka*. Vlastnost *account* je objekt Account a ten obsahuje vlastnost *atype* reprezentující typ účtu.

8.3 Aplikační vrstva

Implementace prostřední části tří-vrstvého systému není v našem případě příliš rozsáhlá. Jak již z návrhu vyplývá, jedná se především o implementaci potomků třídy *Manager*. Ti mají specifické vlastnosti dle charakteru dat, jež mají na starost. Zajímavostí je způsob implementace jednotlivých managerů. Jelikož se předpokládá jejich vzájemné využití kdekoli v prezentační vrstvě, docházelo by tak k vytváření nových instancí ve všech servletech, které zrovna poběží, což by způsobovalo zbytečný nárůst paměťových nároků a server by byl zbytečně zatěžován. Z těchto důvodů jsou všichni správci implementováni jako tzv. *Singleton* objekty, což ve skutečnosti znamená, že jedna z vlastností těchto objektů je statická instance sebe sama, kterou je možno získat voláním konkrétní metody, jež tuto instanci vrací. Konstruktor je tedy volán jen jedou při prvním požadavku na objekt. Pro ošetření rizika souběžného prvního požadavku na objekt z více servletů je tato metoda definována jako *synchronized*, což ve skutečnosti při přístupu objekt uzamče a zabezpečí tak atomičnost této operace.

Následuje výčet základních vlastností univerzálního objektu *Manager*, které jsou reprezentovány metodami implementovanými v jeho potomcích.

- získání instance objektu
- získání jednoho objektu dle požadovaných vlastností. Vlastnostmi může být několik možných parametrů, které lze získat z objektů vyšší vrstvy
- získání celého seznamu objektů dle požadovaných kritérií
- uložení objektu, parametrem instance persistentní třídy
- smazání objektu, parametrem může být instance persistentní třídy, oid, či atribut společný pro záznamy, které si přejeme vymazat

Pro jednotlivé metody, které mohou mít vícero parametrů byl zvolen dvojí přístup. V místě, kde není možné předem odhadnout počet parametrů metody byla zvolena abstrakce v podobě pole s těmito atributy. Ty jsou pak sekvenčně přidány k HQL dotazu. Druhou variantou je využití možností jazyka Java a tou je přetěžování metod. Tato vlastnost nám tedy umožňuje deklarovat metody se stejným názvem, ale jiným počtem jiných parametrů. Počet těchto „stejných“ metod není omezen jinak než případným nedostatkem paměti. Tento přístup je využit například pro získávání seznamu objektů, mazání na základě oid, instance entity apod.

Pro manipulaci s daty v DB je navíc také použito transakčního přístupu, tedy zaručení, že vkládání či editace dat bude atomické. To znamená, že systém neumožní například vložit záznam, pokud se nepodařila nějaká jiná operace v rámci stejné transakce a došlo by tak k nekonzistentnosti dat. V praxi to znamená, že uložení například oprávnění pro uživatele nebude možné, pokud by operace uložení samotného uživatele skončila chybou.

8.4 Prezentační vrstva

Prakticky celá prezentační vrstva je pod správou rámce *Struts*. Samotná vrstva by se dala charakterizovat ze dvou pohledů. Ty se vztahují k MVC modelu rámce. Webová část (*View*) je reprezentována JSP dokumenty, skripty, které obsahují kód HTML prezentace a část logická (*Model*), která zabezpečuje zpracování požadavků a dat zaslaných od uživatele. Jelikož jsou tyto části významně odlišné, přistoupíme k jejich oddělenému popisu. V podkapitole *view* bude podrobněji rozebrána mimo jiné i technika použití JSP a jejich rozvržení v projektu.

8.4.1 Webová část (View)

Jsp skripty mohou obsahovat kromě HTML kódu pro formát výstupu ještě několik různých druhů značení. Toto značení má syntaxi značkovacích jazyků podobnou např. XML a je obsaženo v definičních souborech, které je nutno specifikovat ve skriptech, kde hodláme tyto značky použít. Značky mohou zajišťovat logiku, práci se šablonami apod. Pro možnost vkládat zdrojový kód v jazyce Java kdekoli v těle skriptu slouží vymezovací značky `<% %>` , ve kterých není uživatel nijak omezen. Pokud požadujeme v kódu využívat některé třídy, je nutné použít direktiv pro jejich import do servletu, který je výsledkem prekompilace jsp. Logické značky jsou co do použitelnosti relativně omezené, jsou vhodné například pro jednoduchý výpis obsahu seznamu. Použití je znázorněno ve skriptu *userList.jsp* jež má na starost výpis načteného seznamu existujících uživatelů pro jejich případnou editaci. Značka `<logic:iterate>` má na starost průchod celým seznamem záznam po záznamu. Její atributy definují, v jakém prostoru (*scope*) se nachází proměnná (*name*) , kterou je možno iterovat a objekt (*type*), jež je jednotkou seznamu. Je možnost definovat libovolnou třídu. Dále jsou zde značky reprezentující JavaBean komponenty, které například umějí přistupovat do iterovaného seznamu na pozici aktuálního kurzoru, nebo například přistupovat do souborů zdrojů ve smyslu předdefinovaných konstantních proměnných, které obsahují textové řetězce (*ApplicationResources*). Jsou zde definovaná jak chybová hlášení, tak i například *label* formulářových prvků, tlačítek, sloupců tabulek atp. Těchto souborů zdrojů je možno definovat libovolné množství, například pro vícejazyčnou podporu. V projektu tato podpora implementována přímo není, je však připraven pro toto rozšíření. Pro většinu výpisů tabulek je použito kódu v jazyce Java, neboť bylo třeba podmiňovat velké množství informací a pro tyto potřeby funkčnost logických značek nepostačuje.

Rozložení dokumentů je optimalizováno pro co největší možnou univerzálnost. Je použito standardních JSP značek `<jsp:include />` , jejichž atribut *page* definuje stránku, která se má na dané místo vložit. Takto je do každého z hlavních dokumentů, např. s postfixem „List“ (*cashbookList*, atp.) vložen *header.jsp* a *footer.jsp* , které intuitivně vkládají standardní hlavičku a

patičku HTML dokumentů a informace s nimi spojené. Dokument *header* navíc obsahuje hlavní panel aplikace, zobrazuje v sobě dokument *menu.jsp*, jež má na starosti vykreslování hlavního menu a deklarované cesty k souboru s CSS styly a kalendáři implementovaném v jazyce JavaScript. Kalendář je využit pro usnadnění uživateli formulovat požadované datum a nám tím odpadá kontrola korektního formátu data. Dále kromě hlavičky a patičky vkládáme dle požadavku například dokumenty obsahující formuláře. Ty jsou umístěny v podadresáři *templates*. Ve zmíněném hlavním dokumentu se většinou nachází tabulky pro výpis položek jednotlivých evidencí.

8.4.2 Logická část (Model)

Logická část prezentační vrstvy má za úkol komunikovat s vrstvou aplikační. Jsou to objekty odvozené od třídy *Action*, které přebírají implementaci metody *execute* která je volána řadičem (*controller*) po výběru z definovaných akcí. Do této metody vstupují dle pořadí tyto objekty:

- *ActionMapping* – mapování pro *ActionForward*, tedy objekt definující přesměrování na výstupní stránku dle výsledku provádění metody execute. V projektu využito pro směrování výstupu zpět na formulář v případě výskytu chybového stavu
- *ActionForm* – objekt reprezentující data odeslaného formuláře. Ten je také uživatelsky definován v odvozené třídě, popis bude následovat níže.
- *HttpServletRequest* – kontext aktivního servletu, který akci vyvolal. Přes tento objekt je možno získat přístup do SESSION apod.
- *HttpServletResponse* – kontext servletu pro zaslání okamžité odpovědi klientovi

Metoda *execute* tedy nabízí prostor ke zpracování dat z formuláře. Dle nastavení atributu *validate* v definici akcí je v objektu formuláře volána metoda *validate* dříve, než je spuštěn tento kód. Pokud je atribut *validate* nastaven na *false*, je tato metoda volána přímo a je tedy na uživateli aby explicitně volal validační metodu objektu formuláře. Tento případ se týká většiny definovaných akcí v projektu, neboť bylo třeba provést komplexnější kontroly informací. Třída formuláře je odvozena od *ActionForm*, takže při předání do metody *execute* je nutné přetypování na uživatelský objekt formuláře. Výsledkem volání *validate* je instance třídy *ActionMessages*, což je řekněme obecně kolekce možných chybových zpráv třídy *ActionMessage*, které jsou definovány na základě detekované chyby. Samotný objekt formuláře má počet a charakteristiku vlastnosti definovaných ve webovém formuláři a k nim odpovídající *get/set* metody. Ty tedy odpovídají více méně vlastnostem persistentních objektů, které máme definovány v databázové vrstvě. Metody *validate a reset* jsou převzaty z nadřazené třídy *ActionForm* a jejich implementace je libovolná. *ActionMessages* je možno dále rozšířit případnými uživatelskými chybovými hlášeními a předat je pomocí metody *saveErrors*

(uloží zprávy do pro kontext v budoucnu přeposlaného servletu). Jako jednotlivá chybová hlášení jsou využity opět textové konstanty definované v *ApplicationResources* pro danou jazykovou mutaci.

Následuje rozhodnutí na základě ověřených informací, na kterou stránku bude kontext přesměrován. Ta pak pomocí objektu typu JavaBean `<html:errors/>` vypíše případné zprávy. Nejedná se tedy o přesměrování, které vyžaduje komunikaci s klientským webovým prohlížečem, ale pouze o přeposlání, tzn. předání kontextu mezi různými servlety na straně serveru bez vědomí klienta.

Implementace pomocí rámce Struts je do značné míry příjemnou záležitostí v případě, že nemáte zájem se zabývat samotným zpracováním požadavků. V našem případě bylo vhodné jeho použití pro usnadnění dalších možných rozšíření, implementátor je tedy odstíněn od náročných úprav a připojení dalších modulů se tedy stává záležitostí značně jednoduchou, v podobě úprav konfiguračního souboru. Fyzické připojení modulu je realizováno drobnými úpravami pro provázání s ostatními balíčky systému.

9 Závěr

Cílem diplomového projektu bylo seznámit se s teoretickým zázemím nutným pro úspěšnou realizaci webového informačního systému pro vedení daňové evidence a jeho následná implementace. Součástí práce jsou kapitoly zabývající se obecnou osvětou problematiky daňové evidence v podobě vhodné snad i pro čtenáře, jemuž se nedostalo alespoň základního ekonomického vzdělání v dané oblasti. Další část projektu se zabývala analýzou již existujících komerčních produktů určených pro danou oblast. Přínosem bylo seznámení se s nutnými podmínkami, jež musí splňovat každá aplikace s ohledem na její zaměření a různými metodami řešení pro zpracování souvisejících dokumentů. Tato analýza pak sloužila jako výchozí bod pro stanovení požadavků a předběžný návrh. Samotný návrh systému je pojat z pohledu soustředěného na možnosti uživatele a jeho snadnou orientaci ve funkčnosti aplikace. Plán projektu definoval tři fáze vývoje, v rámci diplomového projektu byly uskutečněny první dvě. Třetí je tedy sumarizací možných rozšíření. Kapitoly návrhu a následně implementace se zabývají rozborem systému z pohledu tří-vrstvé architektury v niž je projekt realizován. Aplikaci v prostředí JSP/Servlet můžeme kladně hodnotit jako velice rychlou i při relativně komplexní stavbě a velké provázanosti. Technologie je vhodná pro implementaci náročných a rozsáhlých systému.

Aplikace pro vedení daňové evidence v pravdě nepatří mezi rozsahem největší možné systémy, účely využitelnosti, studia a demonstrace použití však splňuje beze zbytku. Jednou z klíčových vlastností je však relativně snadná rozšiřitelnost o další funkční moduly. Vhodnými moduly by mohly být např. evidence dlouhodobého majetku, evidence přijatých a vydaných objednávek s navazující správou skladových zásob, nebo mzdy a personalistika.

Literatura

[1] Ekonomický portál Euroekonom.cz – daňová evidence
Dokument dostupný na http://www.euroekonom.cz/podnikani/evidence2.html

[2] Ing. Marek Běhálek: Hibernate framework.
Dokument dostupný na http://www.cs.vsb.cz/behalek/frvs/2005/java/hibernate/hibernate.html

[3] Lukáš Zapletal: Úvod do problematiky Apache Struts. (Květen 2003)
Dokument dostupný na URL http://www.root.cz/clanky/apache-struts-1-1

[4] MySQL AB: MySQL Manual. (Únor 2004)
Dokument dostupný na URL http://www.mysql.com/documentation.

[5] Bollinger, Gary, Natarajan, Bharathi: JSP – Java Server Pages.
Grada Publishing, Praha, 2003.

[6] James Turner: MySQL™ and JSP™ Web Applications: Data-Driven Programming Using Tomcat and MySQL, Sams Publishing, 2002.

[7] Hall, M.: Java servlety a JSP
Neocortex, 2001, ISBN 8086330060

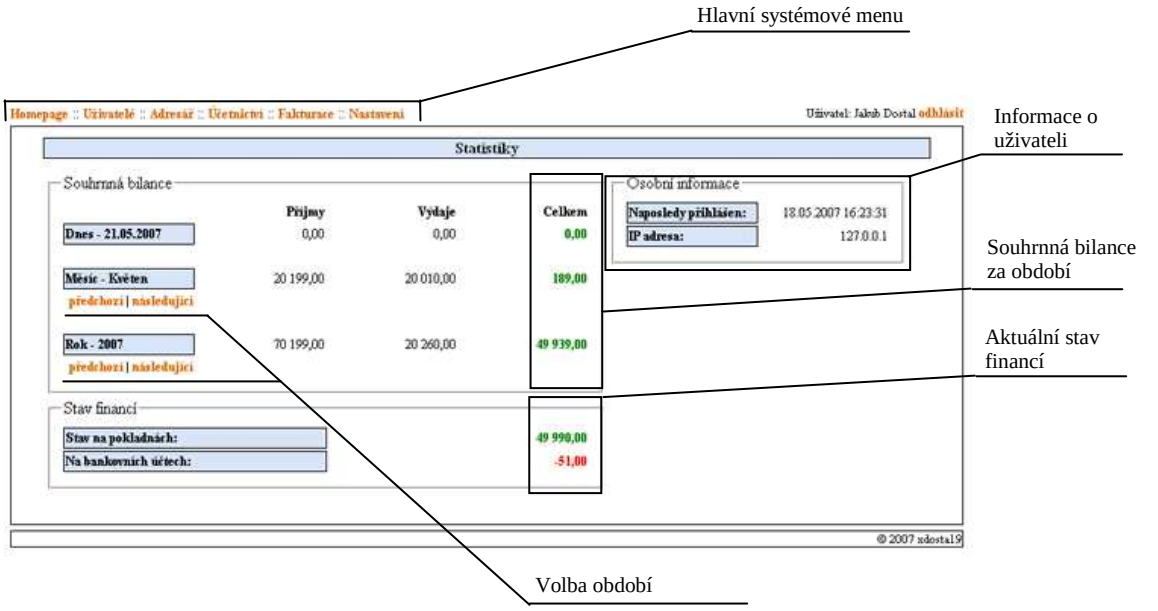
[8] Dokumentace API pro platformu J2EE v 1.4 – java.sun.com
Dokument dostupný na URL http://java.sun.com/j2ee/1.4/docs/api

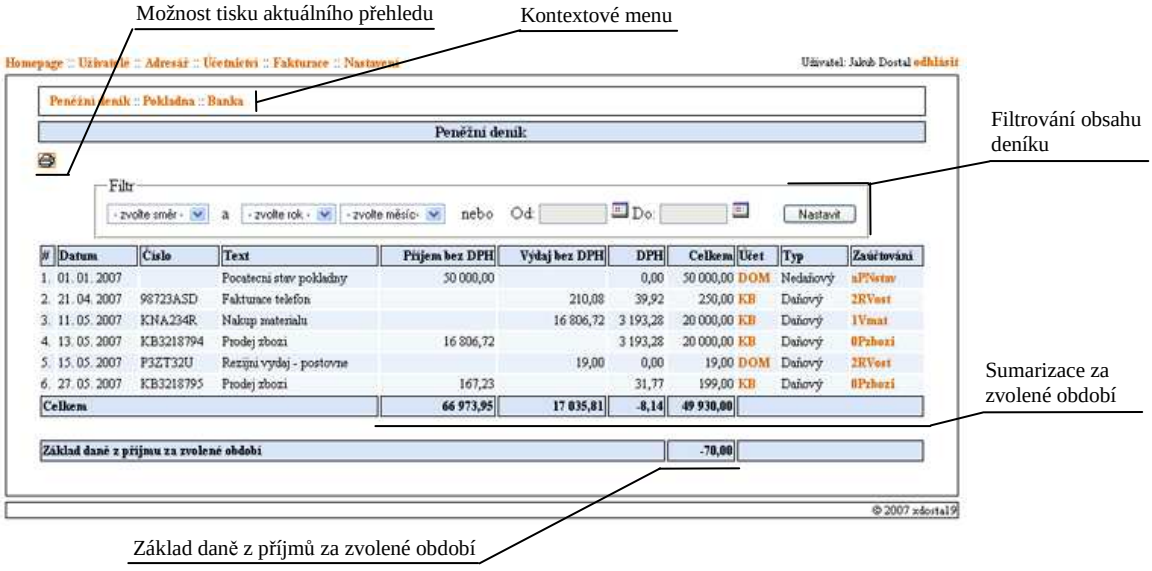
Seznam příloh

Příloha 1

```
@Entity @Table(name="tbl_cashbookitem")
public class CashBookItem implements Serializable
{
     private long id;
     private String idnum;           // cislo
     private Date dateaccount;       // datum
     private int vat;                // DPH %
     private User sysuser;           // vlozil uzivatel
     private AddrItem cust;          // dodavatel/odberatel
     private Account account;        // ucet/pokladna

    public CashbookItem() {}

    @Id @GeneratedValue(strategy=GenerationType.AUTO)
    public long getId() {return id;}
    public void setId(long id) {this.id = id;}

    @Column(name="idnum", nullable=true, unique=false)
    public String getIdnum() {return idnum;}
    public void setIdnum(String idnum) {this.idnum = idnum;}

    @Column(name="dateaccount", nullable=false, unique=false)
    public Date getDateaccount() {return dateaccount;}
    public void setDateaccount(Date dateaccount) {this.dateaccount = dateaccount;}

    @Column(name="vat", nullable=true, unique=false)
    public int getVat() {return vat;}
    public void setVat(int vat) {this.vat = vat;}

    @ManyToOne( cascade = {CascadeType.PERSIST, CascadeType.MERGE} )
    public User getSysuser() {return sysuser;}
    public void setSysuser(User sysuser) {this.sysuser = sysuser;}

    @ManyToOne( cascade = {CascadeType.PERSIST, CascadeType.MERGE} )
    public AddrItem getCust() {return cust;}
    public void setCust(AddrItem cust) {this.cust = cust;}

    @ManyToOne( cascade = {CascadeType.PERSIST, CascadeType.MERGE} )
    public Account getAccount() {return account;}
    public void setAccount(Account account) {this.account = account;}
}
```

Příloha 2





Vysvětlení ostatních použitých ikonek

vyvolání kalendáře pro selekci data

vložení nového záznamu obecně

vložení příjmu do evidence

vložení výdaje do evidence

úprava záznamu v evidenci

smazání záznamu z evidence