VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

IMPLEMENTACE SYSTÉMU PRO DYNAMICKÁ SKONTA

IMPLEMENTATION OF A SYSTEM OF DYNAMIC DISCOUNTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **STANISLAV BARTOŠ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20521/2017/xbarto87

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Bartoš Stanislav**
Obor: Informační technologie
Téma: **Implementace systému pro dynamická skonta**
Implementation of a System of Dynamic Discounts Implementation
Kategorie: Informační systémy

Pokyny:

1. Seznamte se s programovacím jazykem Python a s informačním systémem firmy Investiční aukce.
2. Prostudujte problematiku dynamických skont, díky kterým by si dodavatelé do velkých společností mohli transparentně a podle předem určených pravidel stanovovat dobu splatnosti svých pohledávek, dynamicky přesně podle svých potřeb.
3. Analyzujte možnosti implementace dynamických skont v rámci stávajícího IS a navrhněte takové rozšíření.
4. Navržené řešení implementujte.
5. Zhodnoťte dosažené výsledky a další možnosti pokračování tohoto projektu.

Literatura:

- Pilgrim, M.: Ponořme se do Python(u) 3. CZ.NIC, 2010. ISBN: 978-80-904248-2-1.
- Gelsomino, L. et al.: Supply Chain Finance: Modelling a Dynamic Discounting Programme. Journal of Advanced Management Science Vol. 4, No. 4, July 2016. ISSN: 2315-4462.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2, částečně bod 3.

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: **Bartík Vladimír, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

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

Tato práce se zabývá návrhem a implementací systému pro financování faktur na principu dynamických skont, což dodavatelům umožní získat prostředky dříve a odběratelům přinese zisk. Jedná se o firemní zadání bakalářské práce pro firmu *Platební instituce Roger a.s.* Webová aplikace je implementována v Pythonu a frameworku web2py. Součástí je analýza požadavků, návrh i popis samotné implementace.

Abstract

This thesis is about design and implementation of a system for financing invoices according to the principle of dynamic (Sliding scale) discounts, which allows suppliers to get money faster and customers can obtain a profit. It is company assignment for company *"Platební instituce Roger a.s."* Web based application was implemented in Python and framework web2py. This thesis includes analysis of requirements, design as well as description of an implementation itself.

Klíčová slova

informační systém, dynamická skonta, web, Python, web2py, JavaScript, Ajax, PostgreSQL

Keywords

information system, dynamic discounts, web, Python, web2py, JavaScript, Ajax, PostgreSQL

Citace

BARTOŠ, Stanislav. *Implementace systému pro dynamická skonta*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Implementace systému pro dynamická skonta

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vladimíra Bartíka, Ph.D. Další informace mi poskytli pan Ing. Tomáš Slobodník a pan Ing. Matěj Bednář z firmy Platební instituce Roger a.s. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Stanislav Bartoš
16. května 2018

Poděkování

Chtěl bych poděkovat panu Ing. Vladimíru Bartíkovi, Ph.D. za vedení mé bakalářské práce, panu Ing. Tomáši Slobodníkovi za užitečné rady a odborný dohled při řešení a panu Ing. Matěji Bednářovi za rady z oblasti teorie dynamických skont a celého systému po teoretické stránce.

Obsah

Kapitola 1

Úvod

Cílem mé bakalářské práce bylo navrhnout a implementovat systém pro dynamická skonta, který dodavatelům dodávajícím zboží do velkých firem na faktury s obvykle dlouhou dobou splatnosti (i 90 dnů) umožní získat prostředky rychleji. Tato velká firma – odběratel zaplatí za fakturu dříve, za což je jí poskytnuta sleva podle principu dynamických skont (Sliding scale discount), viz dále.

Zadavatelem mé bakalářské práce je firma *Platební instituce Roger a.s.* Tato firma se zabývá inovacemi ve finančních službách, zejména pak optimalizací splatnosti faktur, což je předmětem mé bakalářské práce. Jedná se o nový koncept této společnosti, který umožňuje dodavatelům do velkých společností transparentně a podle předem určených pravidel zkracovat dobu splatnosti svých pohledávek a zároveň usnadní získávání nových klientů, a tím povede k dosažení většího zisku.

V jednotlivých kapitolách bude věnována pozornost analýze požadavků, návrhu, informacím o použitých technologiích, implementaci atd. V rámci analýzy požadavků se budu zabývat zejména požadavky na systém, rolemi uživatelů a jejich konkrétními případy užití (use case diagram). V další kapitole, tj. Použité technologie, se zaměřím na technologie využívané při řešení včetně jejich stručného popisu. V kapitole Návrh je mimo jiné obsažen návrh databáze, tj. ER diagram. Vlastní implementace systému je pak popsána v kapitole Implementace.

Kapitola 2

Dynamická skonta

V této kapitole je podrobněji popsána teorie dynamických skont (*Sliding scale discount*), včetně výhod, které přináší. Faktické údaje obsažené v této sekci jsem čerpal zejména z článku o dynamických skontech [2] a z materiálu obdržených od firmy *Platební instituce Roger a.s.* [4]

2.1 Teorie skont

V posledních deseti letech vznikala zejména díky rozvoji moderních technologií řešení pro optimalizaci dodavatelského řetězce (anglicky *Supply Chain*), která zahrnují řešení z oblastí od distribuce a logistiky až po financování a tzv. Cash-Flow. Souhrnný název pro tato řešení je Supply Chain Management (SCM). Do těchto řešení, konkrétně do financování (Supply Chain Finance – SCF) a Cash-Flow, spadají i Dynamická skonta. [2]

Supply Chain Finance – SCF se dá obecně definovat jako sada integračních a optimalizačních řešení finančních procesů, která probíhají mezi společnostmi (odběratelem a jeho dodavateli), zaměřujícími se především na optimalizaci Cash-Flow a lepší kontrolu kapitálu v rámci dodavatelského řetězce.

Dynamická skonta jsou formou *obchodního úvěru* (anglicky *trade credit*). Obchodní úvěr je jedním z firmami nejpoužívanějších zdrojů likvidity a ve většině zemí (zejména v USA) se uplatňuje ve dvou základních formách, a to buď ve formě prostého pozdržení platby, nebo tzv. hotovostní slevy (cash discount policy), při níž buď dodavatel umožní odběrateli zaplatit v kratším časovém rozmezí (např. 10 dní) výměnou za slevu, anebo zaplatí plnou nominální hodnotu faktury v případě, že bude platit v běžné lhůtě.

Dynamická skonta jsou dnes považována za nejsofistikovanější metodu SCF a nejdůležitější trend v oblasti managementu dodavatelského řetězce. Jde o tzv. *three-corner* SCF model, neboť zahrnuje celkem tři strany – kromě odběratele a dodavatele je zapotřebí ještě třetí strana – většinou jde o finanční instituci nebo poskytovatele IT služeb (v našem případě v této roli vystupuje firma *Platební instituce Roger a.s.*).

Dynamická skonta se od základní varianty liší možností výměny tzv. *Early Payment Proposal* (EPP). EPP je požadavek na dřívější proplacení faktury, a to výměnou za procentuální slevu z nominální hodnoty faktury. Tato sleva závisí zejména na dvou proměnlivých hodnotách – na dni, kdy byla platba provedena, a na požadovaném skontu. Dynamické skonto je tedy závislé na datu zaplacení faktury, a to přímo úměrně – čím dřív je faktura zaplacena, tím vyšší je skonto (viz sekce 2.4 a obrázek 2.1). [2]

Pro účely propagace u dodavatelů je zde také možnost využít tzv. "One time early payment", což je možnost, kdy velký odběratel proplatí svému dodavateli faktury v termínu, který si dodavatel sám určí, a to bez požadavku skonta. Tato možnost slouží zejména pro zvýšení zájmu dodavatelů při implementaci dynamického skonta.
Shrnutí: Dynamická skonta jsou tedy obdobou běžných skont, avšak díky využití moderních technologií mohou být nastavovány jednotlivě pro každou fakturu zvlášť na základě potřeb dodavatele. To umožňuje dodavateli si určit, do kdy potřebuje dostat za fakturu zaplaceno, podle čehož se vypočítá výše skonta, a to pro každou fakturu jednotlivě. [2]

2.2 Výhody pro odběratele

Mezi nejvýznamnější výhody pro odběratele patří:

- Úspora peněz ve výši cca. 2 % z celkové fakturace (to se odvíjí od základní doby splatnosti a nastavení maximální výše skonta).
- Možnost financovat pohledávky z cizích peněz.
- Úspora nákladů, neboť zaměstnanci nemusí řešit žádosti dodavatelů, kteří chtějí proplatit fakturu dříve.
- Posílení dodavatelského řetězce – nižší riziko platební neschopnosti a tím i neschopnosti plnit objednávky.
- Jednodušší implementace EDI (Electronic Data Interchange, neboli česky Elektronická výměna dat) u dodavatelů.
- Možnost prodloužit si základní dobu splatnosti.
- Lepší vztahy v rámci dodavatelského řetězce.

2.3 Výhody pro dodavatele

Pro dodavatele pak plynou následující benefity:

- Mohou si sami určit, kdy dostanou zaplaceno.
- Větší přehled a kontrola nad finančními prostředky.
- Nižší riziko platební neschopnosti.
- Výhodnější než úvěry.

2.4 Technické řešení dynamických skont

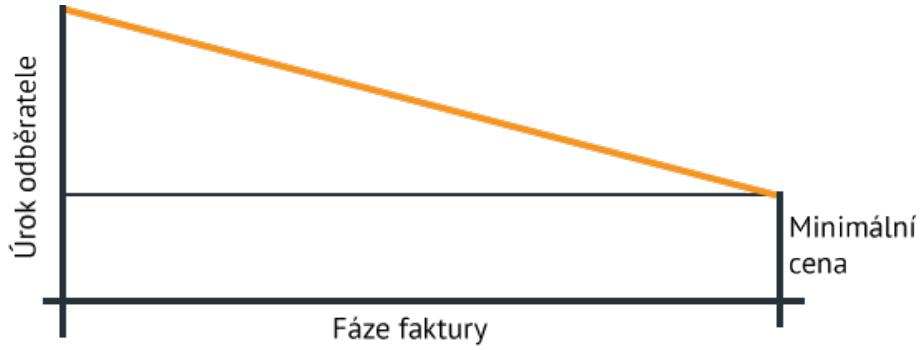
Výše skonta je určována dynamicky pomocí Sliding scale discount. Vstupními parametry jsou požadovaný úrok odběratele, minimální cena a systémem dopočtená fáze faktury, přičemž minimální cena bude fixní na 1 % z hodnoty faktury a požadovaný úrok (požadovaný výnos) si odběratel nastaví sám v rámci svého rozhraní. Vzorec pro výpočet výše skonta je pak následující 2.1:

$$\textit{Výše skonta} = (\textit{Úrok odběratele - minimální cena}) * \textit{Fáze faktury} + \textit{Minimální cena} \tag{2.1}$$

Přičemž fáze faktury se vypočítá takto 2.2

$$\textit{Fáze faktury} = \textit{Doba do splatnosti/doba splatnosti} \tag{2.2}$$

Grafické znázornění skonta je na obrázku 2.1.



Obrázek 2.1: Grafické znázornění dynamického skonta, zdroj: *Platební instituce Roger a.s.*

2.4.1 Fáze

Podle literatury [2] se referenční proces systému dá rozdělit na čtyři fáze práce s fakturou, přičemž z toho dvě fáze (nahrání faktury a přijetí faktury) jsou stejné jako v základním systému pro elektronické faktury. Projdeme si teď každou fázi jednotlivě:

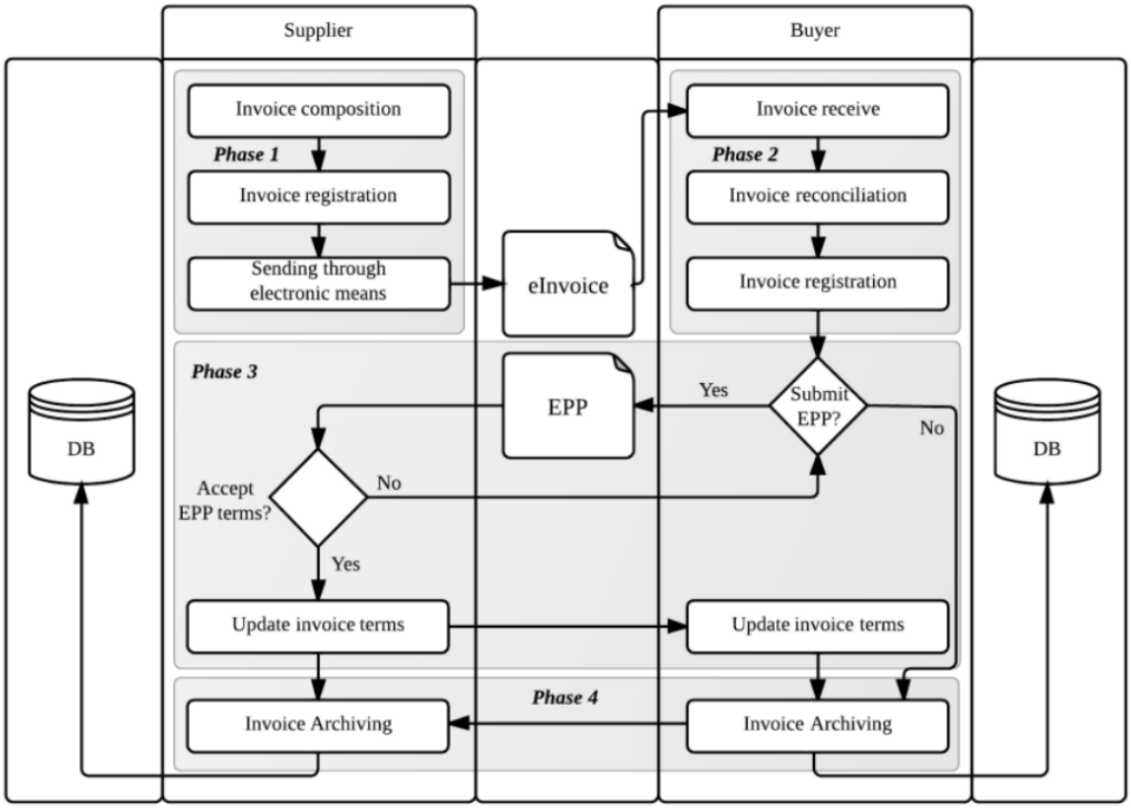
1. **Nahrání faktury**
   Tato fáze zahrnuje nahrání faktury do systému. Fáze začíná vystavením faktury a končí jejím odesláním odběrateli. Výměna mezi dodavatelem a odběratelem probíhá elektronicky, většinou přes komunikační systémy EDI (*Electronic Data Interchange*). Předpokládá se, že fakturované zboží bylo odesláno již před začátkem této fáze. Této fáze se účastní pouze dodavatel, který fakturu vystavuje.

2. **Přijetí faktury**
   Zahrnuje všechny aktivity uskutečněné odběratelem od obdržení faktury a končí schválením platby. Této fáze se účastní pouze odběratel.

3. **EPP**
   Jak již bylo řečeno výše EPP (Early Payment Proposal) je žádost o dřívější zaplacení faktury, výměnou za slevu z její nominální hodnoty. Tato fáze začíná podáním EPP a končí definicí o aktualizaci dat splatnosti v případě, že EPP schválily obě strany – odběratel i dodavatel. Jakmile odběratel odešle EPP a dodavatel to potvrdí, může dojít k aktualizaci splatností a faktura je připravena k archivaci.

4. **Archivace faktury**
   Archivování faktury může být zahájeno třemi událostmi – oboustranným schválením EPP, dosažením původního data splatnosti faktury, anebo případně speciálním nařízením k archivaci faktury v případě specifických regulací (například pevná doba od přijetí faktury, kdy odběratel chce financovat jenom nejnovější faktury s největším skontem). Dokud není faktura označena jako archivovaná, pak ji systém nabízí k EPP – je možnost požádat o její dřívější proplacení.

Celý proces je znázorněn na obrázku 2.2. Tento referenční proces se v praxi uplatňuje ve dvou konfiguracích. V první poskytovatel služby vyvíjí, poskytuje a udržuje IT platformu pro řešení dynamických skont a odběratel si své faktury financuje pouze ze svých vlastních prostředků. Ve druhé konfiguraci poskytovatel služby kromě IT platformy pro řešení skont nabízí odběrateli i finanční prostředky pro financování faktury – tudíž vystupuje jak v roli poskytovatele jak IT tak finančních služeb. Navíc tato možnost dále posiluje dodavatelský řetězec, neboť dodavatel může dostat za fakturu zaplaceno i v případě, že odběratel momentálně nemá dostatek volných finančních prostředků. [2]
Pozn.: v našem konkrétním případě se uplatňuje druhá konfigurace, neboť firma *Platební instituce Roger a.s.* nabízí odběratelům i možnost nechat si financovat faktury z externích zdrojů za určitých podmínek.



Obrázek 2.2: Referenční proces dynamického skonta [2]

Kapitola 3

Analýza požadavků

Tato kapitola obsahuje analýzu požadavků na systém. Analýza byla vytvořena na základě konzultací ve firmě *Platební instituce Roger a.s.* Systém musí obsahovat rozhraní jak pro odběratele, tak pro dodavatele.

3.1 Stávající systém

Platební instituce Roger a.s. je tzv. *FinTech* společnost. *FinTech* je zkratka pro *financial technology*, tedy, jak z názvu vyplývá, jde zejména o propojení finančnictví s moderními technologiemi. Do této kategorie tedy spadají i dynamická skonta.

Z toho vychází i produkty firmy, které cílí zejména na jednoduchost financování. Asi dvě nejdůležitější platformy jsou Platba a Aukce. Platba se zaměřuje především na financování faktur pro menší a střední podniky figurující jako dodavatelé do větších společností. Jde o to, že dodavatelé mohou obdržet až 75% z hodnoty faktury do tří dnů, zbytek po zaplacení faktury odběratelem. Oproti tomu Aukce cílí primárně na investory, kteří si vybírají, které faktury chtějí financovat, a poté v aukci soutěží o to, kdo nabídne lepší podmínky financování.[4]

Platformy firmy *Platební instituce Roger a.s.* jsou založeny na frameworku web2py pro jazyk Python. Jelikož mnou vytvářená aplikace má být nasazena v prostředí firmy *Platební instituce Roger a.s.*, pak bude nejjednodušší použití stejných technologií, a to jak pro vývoj, tak zejména pro nasazení a pro udržování, a v neposlední řadě pak také pro samotné zaměstnance, neboť nebudou muset zápolit s nějakou pro ně neznámou technologií. Další výhodou je, že aplikace mohou běžet na jednom serveru, jsou schopny spolu komunikovat přímo, neboť jednotlivé moduly mohou být přímo importovány apod.

Ve světle výše uvedeného jsem se rozhodl nevybočovat a použít osvědčené technologie – tedy Python s frameworkem web2py a databáze v postgreSQL. O těchto technologiích se lze dočíst více v kapitole 4. V rámci web2py bude systém implementován jako samostatná aplikace, a to zejména pro oddělení jednotlivých platforem, a tedy lepší čitelnost, navíc jednotlivé aplikace web2py mohou běžet současně i pro jednu instanci web2py.

Závěrem tedy je, že systém bude implementován ve web2py jako samostatná aplikace včetně samostatné databáze.

3.2 Systém a proces

V této podkapitole jsou uvedeny požadavky na systém z pohledu procesu, a dále jsou zde uvedeny části, ze kterých se bude systém skládat, a jejich funkce.

3.2.1 Dělení systému

Systém se dělí na následující části:

- backend,
- administrátorské rozhraní,
- frontend pro dodavatele,
- frontend pro odběratele.

3.2.2 Proces

1. **Stažení a fakturace do systému**
    - Administrátorské rozhraní
        (a) Nahrání faktur do systému ve formátu csv
2. **Vytvoření nabídky**
    - Frontend odběratele
        (a) Vytvoření nabídky financování
        (b) Schválení financování dodavatelů
3. **Zpracování a potvrzení nabídky**
    - Frontend dodavatele
        (a) Přehled faktur
        (b) Financování faktur
4. **Vytvoření dokumentace**
    - Backend
        (a) Vytvoření platebního avíza

3.2.3 Funkce frontendu dodavatele

Frontend pro dodavatele slouží primárně k výběru faktur pro financování, jelikož ty si vybírá dodavatel. Dále by měl mít možnost si zobrazit faktury financované v minulosti. Bude se tedy skládat z těchto částí:

Přehled faktur

V Přehledu faktur se má nacházet tabulka financovatelných faktur. Financovatelné faktury jsou ty, které ještě nebyly financovány, do splatnosti jim zbývá minimálně 5 dní a minimální hodnota faktury je 20 000 CZK. V tabulce se objevují pouze faktury importované v souboru formátu CSV. Kromě toho musí tabulka umožňovat filtraci podle odběratele a doby do splatnosti. Dále v ní musí být obsažena také funkcionalita pro výběr jednotlivých funkcí k profinancování – viz Financování faktur níže.

Přehled minulosti

V Přehledu minulosti by se měla nacházet tabulka obsahující faktury, které již byly v minulosti financovány, a to včetně údaje o výši poskytnutého skonta.

Financování faktur

V rámci Přehledu faktur si dodavatel bude moci vybrat jednu nebo více faktur k postoupení k financování.

3.2.4 Funkce frontendu odběratele

V rámci svého frontendu si odběratel bude primárně nastavovat podmínky, za kterých je ochoten faktury financovat, tedy zejména požadovaný výnos. Proto se frontend pro odběratele musí skládat minimálně z části pro vytvoření nabídky a dále části pro schválení dodavatelů.

Vytvoření nabídky financování

Předpokladem pro tento případ užití je, že odběratel má volný balík peněz a poté vypíše nabídku na poskytnutí skont. Zadá množství volných finančních prostředků a velikost skonta, které požaduje.

Schválení financování dodavatelů

Systém pro odběratele musí umožnit schvalování (i zrušení schválení) dodavatelů. Schválení dodavatelé si poté budou moci nechat financovat faktury s požadovaným skontem.

3.2.5 Backend

Backend představují ty části, které budou pracovat na serveru, tedy zejména databázi.

Databáze

V databázi musí být uloženy identifikační údaje jednotlivých společností a dále také veškeré faktury, naimportované do systému ze seznamu ve formátu csv.

Generování platebních přehledů

V rámci zkrácení splatnosti vytvoří systém přehled (pravděpodobně v excelové podobě) pro odběratele, jehož obsahem bude: seznam faktur, u kterých bylo schváleno skonto, výši ve které mají být zaplaceny, a datum, ke kterému mají být zaplaceny.

Archivace/Tvorba auditní stopy

Systém uchovává záznamy o následujících akcích uživatelů (ze strany dodavatele i odběratele): přidání odběratele, vytvoření nabídky financování, financování faktur, schválení financování dodavatelů.

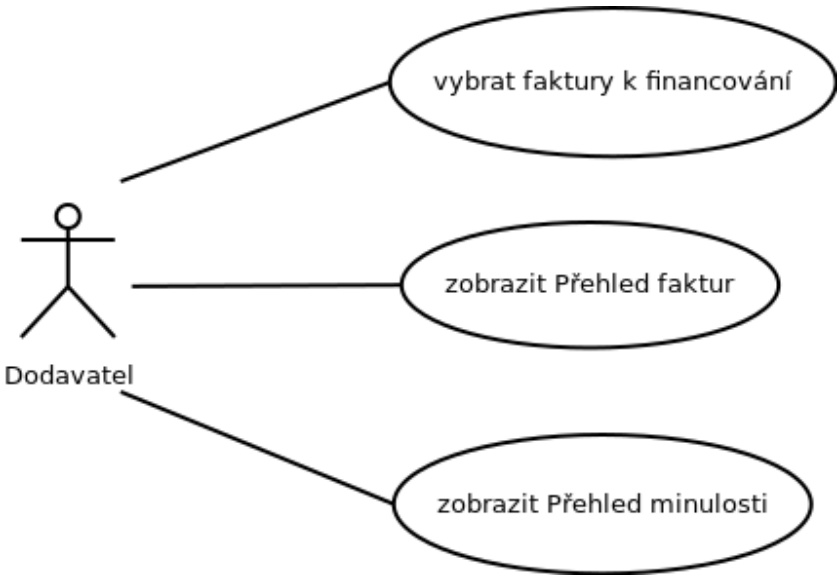
3.2.6 Administrační rozhraní

Administrační rozhraní umožňuje správu backendu, jmenovitě databáze dodavatelů a odběratelů, nastavení výše minimální ceny a uchovávání nahraných a vytvořených dokumentů. Kromě toho zde má jít také nahrávat faktury ve formátu csv.

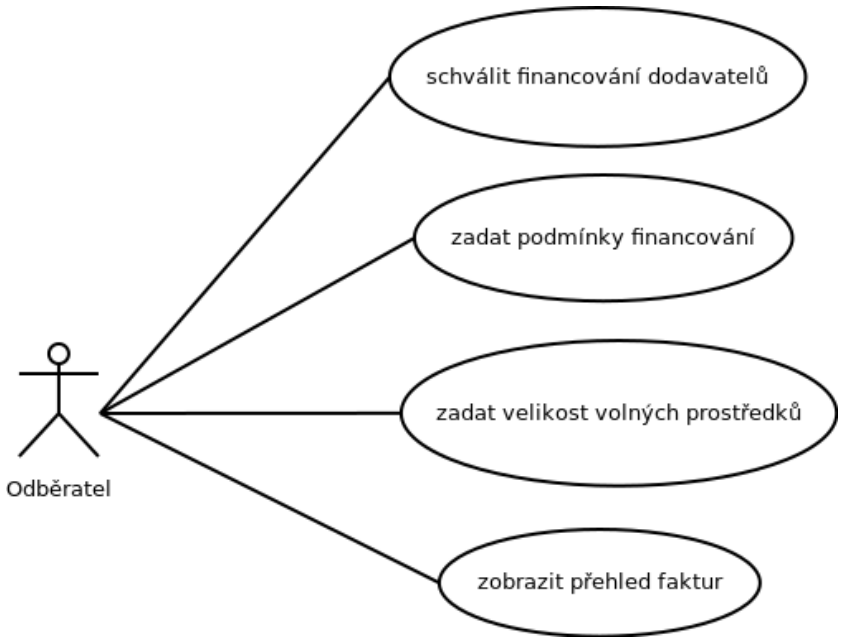
3.3 Diagram případů užití

Z výše uvedených požadavků je patrné, které skupiny uživatelů (role) se v systému budou vyskytovat a co bude konkrétnímu uživateli v konkrétní skupině umožněno vykonat. Kvůli přehlednosti jsem celý diagram případů užití (anglicky *Use Case Diagram*) rozdělil do tří částí podle rolí.

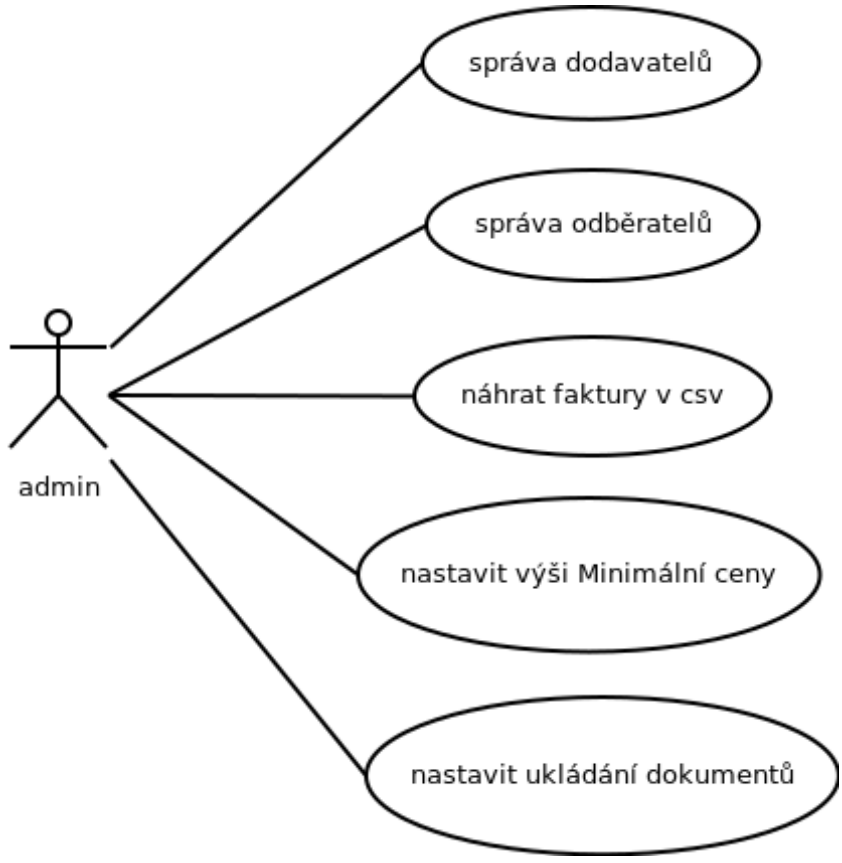
První ze tří rolí je dodavatel, viz obr. 3.1. Dodavatel si zobrazí aktuální faktury včetně nabízeného skonta a může si vybrat faktury, které si nechá profinancovat. Dále si musí umět zobrazit v minulosti financované faktury. Další rolí je odběratel – viz obr. 3.2. Odběratel nastavuje výši finančních prostředků a velikost požadovaného skonta (podmínky financování). Výše finančních prostředků představuje objem peněz, které je odběratel ochoten použít na dřívější proplacení faktur dodavateli a velikost požadovaného skonta, respektive požadovaný výnos představují výnos odběratele p.a., tedy za rok. Dále odběratel schvaluje dodavatele, kterým chce nabídnout financování a v neposlední řadě by měl mít možnost si zobrazit faktury. Poslední rolí je admin viz obr. 3.3. Jeho nejdůležitější rolí je správa uživatelů a nahrávání faktur ve formátu csv. Dále může nastavovat minimální cenu a ukládání dokumentů.



Obrázek 3.1: Diagram případů užití pro dodavatele



Obrázek 3.2: Diagram případů užití pro odběratele

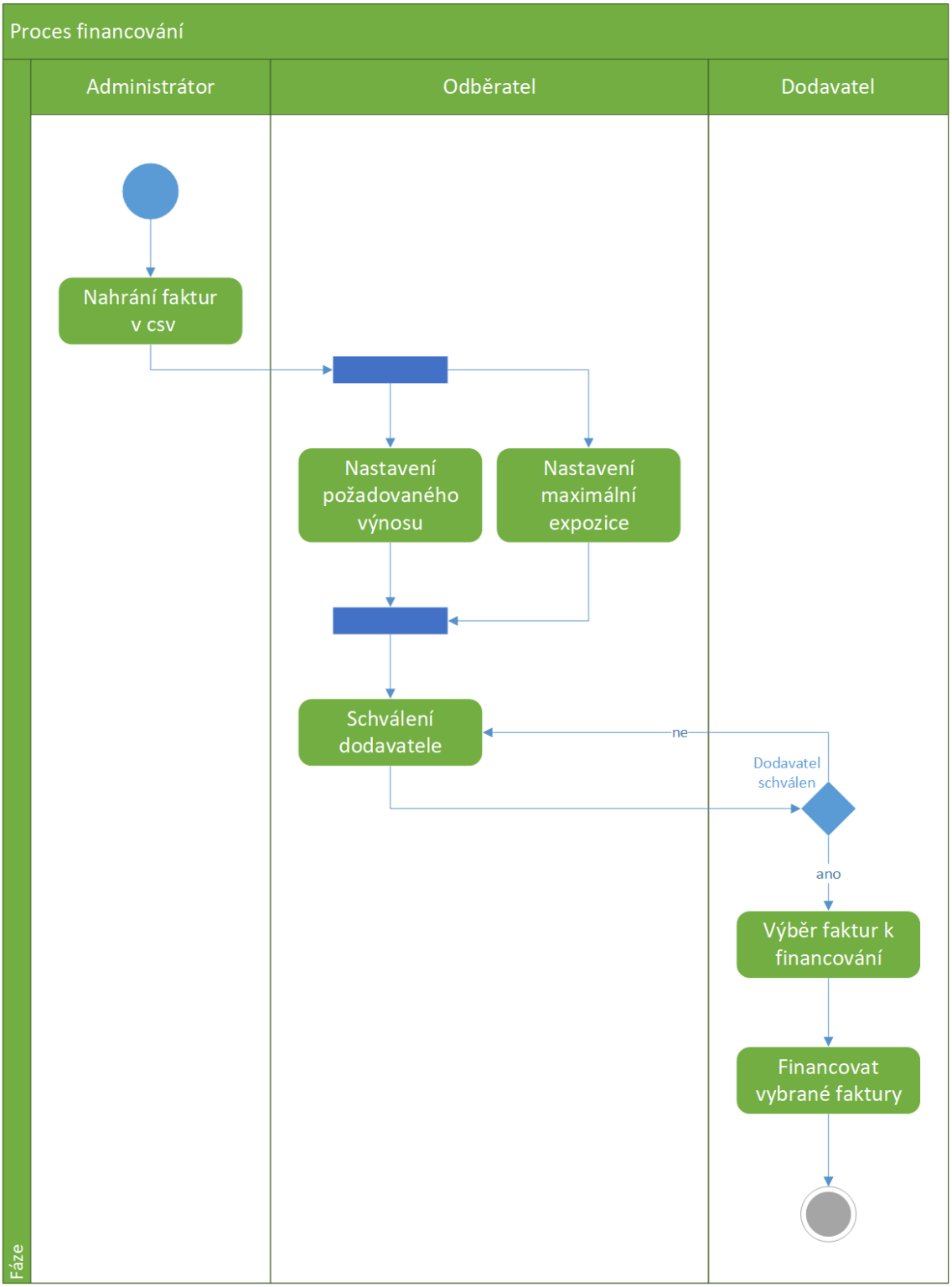


Obrázek 3.3: Diagram případů užití pro administrátora

3.4 Diagram aktivit

Pro názornost jsem se rozhodl vytvořit i zjednodušený diagram aktivit 3.4, na kterém je zachycena předpokládaná posloupnost uživatelských akcí tak, jak následují po sobě. Jedná se o zjednodušenou variantu, takže neobsahuje akce, které provádí systém, ale pouze uživatelské aktivity.

Nejprve administrátor musí nahrát faktury do systému, což je realizováno nahráním pravděpodobně csv souboru, který je většinou vygenerován z interního ERP systému odběratele. Následně odběratel nastaví všechny potřebné podmínky financování, což zahrnuje jak nastavení požadovaného výnosu, tak i nastavení maximálních volných prostředků (expozice), které je odběratel ochoten financovat. Ostatní prostředky pak může poskytnout firma *Platební instituce Roger a.s.* Následně odběratel vybere dodavatele, kterým umožní nechat si faktury profinancovat dříve se skontem. Tito vybraní dodavatelé si pak budou moci v systému vybrat jednotlivé faktury k profinancování. Postup je zjednodušeně znázorněn v diagramu aktivit 3.4.



Obrázek 3.4: Zjednodušený diagram aktivit

Kapitola 4

Použité technologie

V této kapitole budou stručně rozebrány technologie použité při řešení. Výběr použitých technologií je založen především na požadavcích a doporučeních od firmy *Platební instituce Roger a.s.* Důležité informace nezbytné pro pochopení implementace systému, tedy zejména týkající se frameworku web2py, jsou uvedeny dále v kapitole 6. V této kapitole se tedy omezím spíše na výčet jednotlivých technologií a jejich stručný popis.

4.1 Python

Python je vysokoúrovňový, interpretovaný programovací jazyk, jehož autorem je Guido van Rossum. K jeho typickým znakům patří dynamická kontrola datových typů, podporuje více paradigmat – zejména objektově orientované a procedurální, ale i např. funkcionální. K jeho dalším velkým výhodám patří efektivnost, alespoň tedy mezi interpretovanými jazyky, protože v případě standardní a nejvíce používané implementace – CPython – je Python napsán v jazyce C, stejně jako většina standardních modulů. [3]

Dnes jsou používány dvě navzájem nekompatibilní verze Pythonu – verze 2 a verze 3. Pro implementaci jsem použil verzi 2, konkrétně 2.7, neboť jsou v ní psány aplikace firmy *Platební instituce Roger a.s.* a jedná se o doporučenou verzi pro web2py. Navíc na web2py podpoře Pythonu 3 se sice pracuje, ovšem stále se vyskytují nějaké chyby. Z hlediska využitelnosti do budoucnosti to představuje celkem problém, neboť bylo oznámeno, že do roku 2020 bude ukončena podpora pro verzi 2, a proto předpokládám útlum této verze.

Na přepis kódu z Pythonu 2 do Pythonu 3 existuje skript *2to3*, který se dodává spolu s Pythonem 3. Tento skript umí automaticky převést většinu věcí, které jsou v těchto verzích odlišné – jedná se především o změny v syntaxi, ostatní je však třeba převést manuálně. [3]

4.1.1 PEP

Další z pojmů, se kterými je třeba se seznámit, je *Python Enhancement Proposal*, zkráceně PEP[5]. Jak již název napovídá, jedná se o návrhy komunity, jak Python vylepšit, případně doporučení pro vývoj v Pythonu. Návrhy jsou poté schvalovány jednak komunitou, a jednak autorem Pythonu – Guido van Rossumem. V praxi je jedním z nejdůležitějších PEP 8, jelikož obsahuje doporučení, jak psát v Pythonu přehledně a čitelně. To je velmi důležité, neboť jak je známo, tak při vývoji software se nejvíce času nestráví psaním zdrojového kódu, ale jeho čtením. Zmiňuji se o tom zejména proto, neboť jsem se snažil při implementaci systému tato doporučení co nejvíce dodržovat. Všechna tato pravidla lze nalézt zde [6], namátkou uvedu jen některá z nich:

- Odsazení o velikosti 4 mezer.
- Maximální počet znaků na řádek je 79.
- Funkce jsou odděleny dvěma prázdnými řádky.
- Importy by měly být na samostatných řádcích.
- Binární operátory by měly být z obou stran odděleny mezerou.
- Operátor přiřazení by měl být oddělen z obou stran mezerou (x = 5), pokud se nejedná o přiřazení do parametru v rámci volání funkce, v takovém případě by neměl být oddělen mezerami (function(x=5));
- a další viz [6].

4.2 web2py

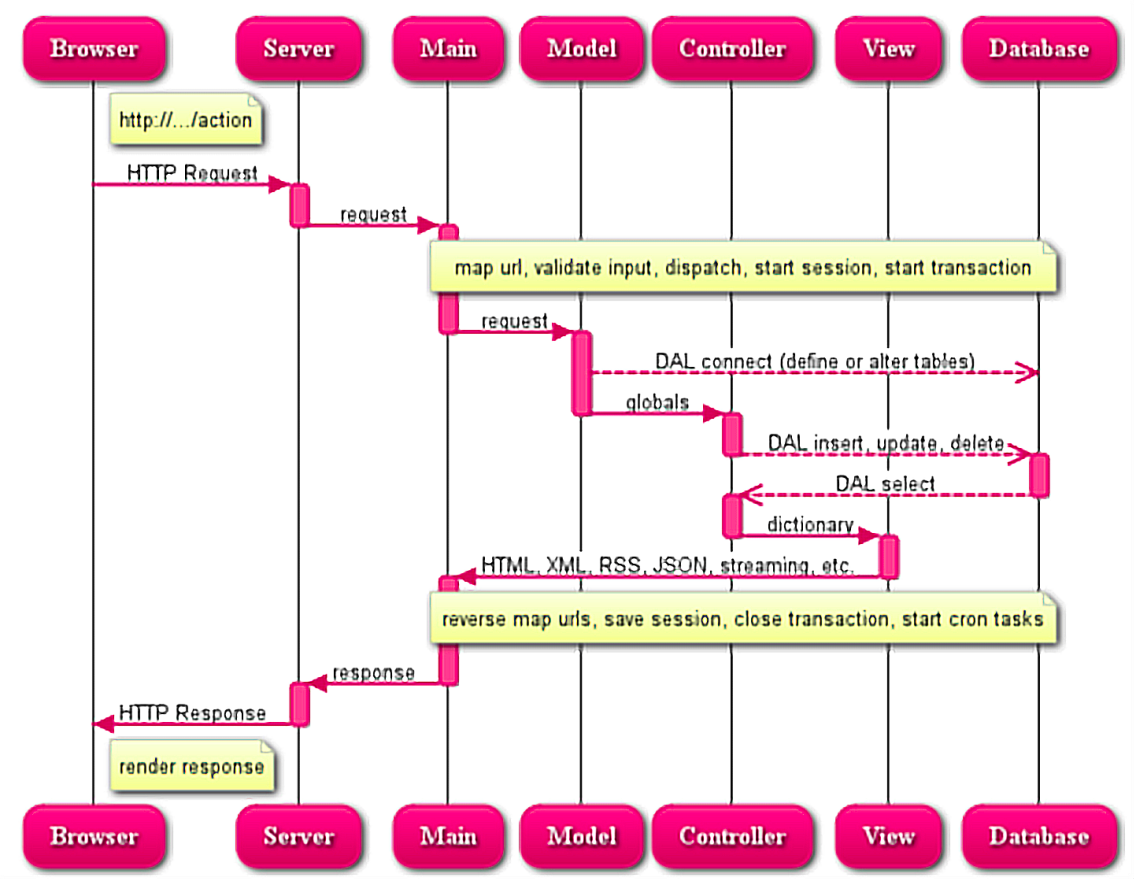
Jak již bylo popsáno výše 3.1, tak pro implementaci byl použit framework web2py – konkrétně jeho verze 2.16.1 (stable) z roku 2017.

Web2py je open-source webový aplikační framework. Je implementován v jazyce Python, viz. 4.1. Původním a hlavním autorem je profesor Massimo Di Pierro, který web2py začal vyvíjet v roce 2007, a to především pro vzdělávací účely. Hlavní výhodou je jednoduchost používání – web2py není potřeba instalovat, pracuje ihned po rozbalení a spolupracuje s většinou webových serverů, databází a všemi knihovnami Pythonu. Web2py získalo v minulosti několik ocenění v oblasti open-source softwaru a full-stack webových frameworků, založených na Pythonu. Full-stack webový framework znamená, že obsahuje všechny komponenty, které jsou zapotřebí pro vytvoření kompletní webové aplikace.

Vývoj ve web2py je založen na dodržování návrhového vzoru Model-View-Controller (MVC), a to díky tomu, že web2py odděluje data (model) od řídící logiky (controller) a prezentace (view). Obecně platí, že model obsahuje definici dat, tudíž většinou definici nějaké databáze, případně globálních proměnných. V rámci definice databáze je zahrnuto jak definování tabulek, tak případných dalších funkcí, triggerů a podobně. View, neboli pohled, se pak stará o prezentaci dat uživateli. V případě webové aplikace je k tomu prakticky vždy využit jazyk HTML, v případě desktopových aplikací se využívá jazyk, který je typicky postaven na XML, ať už QML v případě Qt, XAML v rámci WPF aplikací a další. Controller, neboli řadič, případně kontrolér, se stará o funkcionalitu aplikace, k čemuž se typicky používá standardní programovací jazyk (např. v našem případě Python). Kontrolér má typicky na starosti vytváření databázových dotazů a posílání jimi získaných dat do pohledu. Princip MVC ve web2py je znázorněn na obrázku 4.1.

Jako server se může použít buďto vestavěný web2py server, anebo klidně i server třetí strany (third-party), např. Apache. Main je jádro WSGI aplikace. WSGI – Web Server Gateway Interface je Python standard pro komunikaci mezi serverem a aplikací v Pythonu. Model, controller a view pak představují vlastní uživatelskou aplikaci, v jedné instanci web2py jich může být spuštěno i více.

Další z důležitých vlastností je důraz na zabezpečení, ať už jde o ochranu proti SQL injekci, nebo ochranu proti Cross Site Scripting. Hesla jsou vždy ukládána ve formě hash a session je uloženo na straně serveru. [9]



Obrázek 4.1: Průběh zpracování požadavku ve web2py [9]

4.3 PostgreSQL

Všechna data, jmenovitě např. faktury a údaje o firmách, je nutno někde ukládat. K tomu slouží *Systém řízení báze dat* (*database management system*). Mnou použitý systém řízení báze dat je PostgreSQL, který je objektově-relační a je k dispozici pod licencí typu MIT, tudíž se jedná o open-source. Primární platformou je GNU/Linux, případně obecně UN*X. Pro komunikaci s databází slouží jazyk SQL, a v případě, kdy je požadováno provedení procedurálních operací (funkce, větvení,...), lze použít i procedurální nadstavbu PL/pgSQL, ne nepodobnou PL/SQL od firmy Oracle.

4.4 HTML a CSS

Jelikož se jedná o webovou aplikaci, tak pohled (view) je implementován v HTML. HTML (*HyperText Markup Language*) je značkovací jazyk, odtud Markup Language, který se používá pro popis webových stránek. K tomu se v tomto jazyce používají tzv. tagy, ty navíc mohou obsahovat své atributy. Tag poznáme tak, že je uzavřen mezi úhlové závorky (<, >). Rozpoznáváme otevírací a uzavírací tagy – uzavírací tag navíc obsahuje lomítko za první úhlovou závorkou. Otevírací a uzavírací tag spolu tvoří element, ve kterém mohou být obsaženy další elementy, čímž se dosáhne jakési kaskádové struktury.

CSS (*Cascading Style Sheets*) se používá k definici stylu, resp. vzhledu webové stránky. Obsahuje tzv. pravidla, kterými lze určovat vzhled jednotlivých elementů. Mezi jeho výhody patří zejména fakt, že CSS soubor je možné vložit do více HTML souborů, a dále také to, že je oddělena deklarace elementů od definice jejich vzhledu.[7]

4.5 JavaScript

JavaScript je objektově orientovaný skriptovací jazyk, používaný v dnešní době zejména pro ovládání různých událostí a interaktivních prvků v rámci webové stránky na straně klienta. Syntaxí se řadí do rodiny jazyka C, sémanticky je však zcela a naprosto odlišný. K tomu, aby mohl být JavaScript použit, je samozřejmě nezbytné, aby jej podporoval a uměl interpretovat webový prohlížeč, což je však v dnešní době téměř samozřejmostí.

Využití JavaScriptu v rámci webu je poměrně široké – od již zmíněných interaktivních prvků GUI přes zpracování událostí až po validaci uživatelských vstupů.

Dalším rozšířením využitelnosti JavaScriptu v rámci webových aplikací je knihovna jQuery. Tato knihovna se zaměřuje na další usnadnění interakce mezi JavaScriptem a HTML. Umožňuje například jednoduchý přístup k jednotlivým DOM elementům, manipulaci s atributy, různé efekty a animace, a dále také AJAX.

AJAX (*Asynchronous JavaScript and XML*) je, jak už anglický název napovídá, technologie založená na využití JavaScriptu a XML. Použije se JavaScriptový kód přiložený k webové stránce, který provádí další asynchronní volání serveru. Server obvykle vrátí odpověď v XML, kterou musí klientský JavaScript dále zpracovat. To celé umožňuje měnit jen některé DOM elementy, aniž by se musela znovu načítat celá webová stránka.[1]

Kapitola 5

Návrh

V této kapitole se budu zabývat návrhem databáze, a to jak teorií návrhu, tak konkrétním návrhem řešení pro implementaci dynamických skont.

5.1 Fáze návrhu

Podle knihy *Database system concepts*[8] je v první fázi návrhu nejdůležitější si dobře promyslet, jaká data budou potřebovat budoucí uživatelé ukládat, tedy je potřeba získat tzv. user requirements, tj. požadavky uživatelů.

Dalším krokem je zvolit datový model a za použití konceptu tohoto modelu se převedou požadavky uživatelů na konceptuální schéma. Typicky bývá použitým modelem *entity-relationship model* (ER). V něm konceptuální schéma představuje entity, které jsou reprezentovány v databázi, atributy entit a vztahy mezi jednotlivými entitami. Obvykle také fáze konceptuálního návrhu vyústí ve vytvoření ER diagramu (*entity-relationship diagram*), který poskytuje grafické znázornění schématu. V další fázi je poté nutné toto schéma přezkoumat a odhalit případné nedostatky v návrhu.

Nakonec následuje finální fáze, tedy převedení abstraktního modelu na implementaci databáze. To zahrnuje především převod abstraktního datového modelu na implementační model databázového systému, což je typicky například relační databázový model. Poté následuje samotná implementace podle vybraného implementačního modelu.[8]

5.2 Entity-Relationship model

Entity-Relationship model byl vytvořen k usnadnění návrhu databáze. Jelikož jsem tento model při návrhu také využil, tak se o něm stručně zmíním.

Základem ER modelu jsou entity a vztahy mezi nimi. Entita je objekt z reálného světa, který je rozlišitelný od ostatních. Například každá osoba na univerzitě je entita. Každá entita obsahuje množinu svých vlastností, přičemž hodnoty některých vlastností mohou danou entitu identifikovat. Těmto vlastnostem se říká atributy. Entity stejného typu, které obsahují stejné atributy (pozor – ne hodnoty atributů), se dají sdružit do množiny – této množině se pak říká entitní množina. Atributy, pomocí kterých lze entity jednoznačně identifikovat, se nazývají primární klíče.

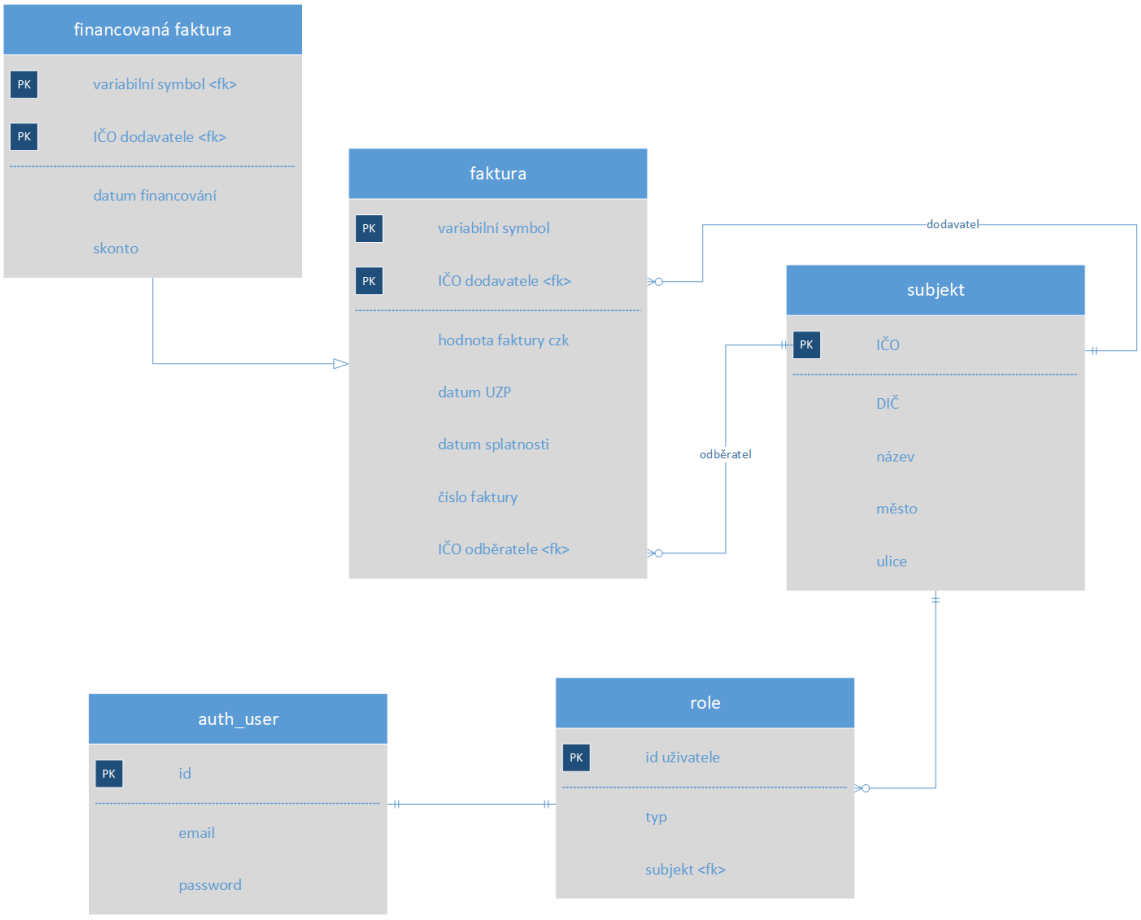
Vztahem se rozumí jakési spojení mezi entitami. Stejně jako entity i vztahy tvoří množiny, a to na základě toho, do kterých entitních množin patří entity, mezi kterými je daný vztah.[8]

5.3 Vlastní návrh

V této podkapitole se budu věnovat vlastnímu návrhu. V první fázi návrhu byl vytvořen ER diagram, který byl následně převeden na schéma databáze.

5.3.1 ER diagram

Prvotní návrh sestával zejména z definice těch nejdůležitějších dat, které bude nutné ukládat. V této fázi jsem se ještě kvůli větší přehlednosti nezabýval mimo jiné nastavovanými hodnotami odběratelů (např. výše požadovaného skonta), ukládáním všech hodnot z faktury atd. Výsledný ER diagram z této fáze je na obrázku 5.1. Jak je na něm vidět, tak tento návrh sice neobsahuje všechny entity, jež se nakonec v databázi vyskytují, ale pro pochopení funkcionality systému a základů vnitřní logiky navržené databáze je podle mého názoru optimální. Základ systému tvoří faktury a subjekty (firmy), které vystupují na faktuře v roli buď odběratele, nebo dodavatele. Dále je specializace financovaná faktura, která značí, že faktura byla financována odběratelem dříve v rámci dynamických skont. Auth_user představuje uživatele, který se přihlašuje do systému, role pak mapuje daného uživatele na příslušný subjekt (firmu) a navíc rozhoduje, zda daný účet bude sloužit pro přihlášení buď do dodavatelské, nebo odběratelské části, protože z logiky věci vyplývá, že uživatel nemůže vystupovat zároveň jako odběratel i dodavatel. Subjekt (firma) je jednoznačně identifikovatelná svým identifikačním číslem osoby (IČO), které tudíž poslouží jako primární klíč. Fakturu je pak nutné identifikovat jednak variabilním symbolem a jednak i identifikačním číslem osoby, která danou fakturu vystavuje, tudíž se použije IČO dodavatele – jde tedy o složený primární klíč. Je to dáno tím, že faktury jsou vystavovány dodavateli, a proto může nastat situace, kdy dva rozdílní dodavatelé nastaví stejný variabilní symbol pro dvě různé faktury, takže samotný variabilní symbol není dostačující.



Obrázek 5.1: ER diagram základních entit návrhu

5.3.2 Schéma databáze

V druhé fázi, po zapracování všech entit nezbytných pro chod systému a připomínek od firmy *Platební instituce Roger a.s.*, jsem návrh upravil tak, jak je znázorněn na schématu databáze 5.2.

Názvy entit byly upraveny na anglické názvy tak, jak budou následně i pojmenovány tabulky v databázi. Kromě toho bylo rovněž požadováno, aby byly veškeré hodnoty na faktuře ukládány v databázi, tedy v našem případě hodnoty, které mohou být potenciálně přítomné v csv souboru od odběratele. Rozdělení atributů faktury je kvůli tomu, že k některým atributům se v praxi bude přistupovat častěji, k některým zase prakticky vůbec. Dále byly do návrhu zakomponovány proměnlivé hodnoty, které si nastavuje odběratel v rámci svého systému, tj. požadovaný výnos (`required_profit`) a maximální expozice (`maximum_exposition`). Co se týká primárního klíče *id*, který se vyskytuje u některých entit, tak ten je vytvořen automaticky web2py jako autoinkrement. Dále tento návrh oproti předcházejícímu obsahuje také veškeré atributy v tabulce `auth_user`. Je to kvůli tomu, že je tato tabulka implementována ve web2py a všechny tyto atributy jsou v ní přítomny, proto by bylo dobré, aby návrh obsahoval celou tabulku, tak jak bude následně v databázi. Na druhou stranu budou uživatelé využívat k přihlášení pouze email a heslo. S tím souvisí i to, že samotné web2py definuje některé své tabulky, které reálně budou v databázi přítomny – jmenovitě např. `auth_event`, která obsahuje log událostí – zejména přihlášení uživatelů. Tyto tabulky však není nezbytně nutné v návrhu znázorňovat. Nyní si projdeme jednotlivé tabulky:

- `subject` představuje fyzické a právnické osoby. Primárním klíčem je atribut `ico` – IČO není přeloženo do angličtiny ze dvou hlavních důvodů, totiž že je to zažitá zkratka v českém prostředí a překlad například jako ID by mohl být mylně interpretován jako autoinkrementální ID používané web2py. Zároveň na nepřekládání této zkratky v názvu tabulky databáze byl zájem firmy *Platební instituce Roger a.s.* Dalšími atributy jsou `dic` – DIČ, `name` – jméno, `place` – bydliště a `street` – ulice.

- `invoice` je klíčovou tabulkou celého systému. V této tabulce jsou ukládány ty nejdůležitější údaje z jednotlivých faktur, ostatní hodnoty jsou pak ukládány v tabulce `invoice_other_values`. Primární klíč je v tomto případě složený, jak již bylo popsáno u ER diagramu, tvoří jej jak variabilní symbol – `variable_symbol`, tak IČO dodavatele – `supplier_ico`, které je zároveň cizí klíč. Další atributy jsou `subtype` – subtyp, `invoice_number` – číslo faktury, `matter` – věc, `taxable_supply_date` – datum UZP (uskutečnění zdanitelného plnění, tedy začátek platnosti faktury), `due_date` – datum splatnosti, `invoice_value_czk` – hodnota faktury převedena na koruny, `invoice_value` – hodnota faktury, `currency` – měna, `payment_state` – stav platby, `setoff_state` – stav zápočtu, `request` – požadavek, `order_number` – číslo objednávky, `order_name` – název objednávky, `bank_account` – číslo účtu a konečně `customer_ico` – číslo odběratele, což je cizí klíč, tedy reference na tabulku `subject`.

- `invoice_other_values` obsahuje údaje z faktury, které buď vůbec nemusí být přítomny, nebo nejsou relevantní pro chod systému, ale přesto je nezbytné je ukládat. Jedná se o atributy `book` – kniha, `type` – typ, `account_transaction_date` – datum uskutečnění účetního případu, `obligation` – závazek, `obligation_czk` – závazek v korunách, `tax_base` – základ daně, `exchange_rate` – kurs, `last_payment_date` – datum poslední úhrady, `payment_form` – forma úhrady a `alt_supplier_name`, tedy alternativní název dodavatele.

- `financed_invoice` obsahuje pouze faktury, které si dodavatel nechal profinancovat dříve se skontem. O těchto fakturách je nutno uložit kdy a s jakým skontem byly profinancovány. Kromě složeného primárního klíče, který je zároveň cizí klíč jako reference na tabulku `invoice`, tedy tabulka obsahuje atributy `finance_date` – datum financování, a `discount` – dynamické skonto, za které byla faktura financována.

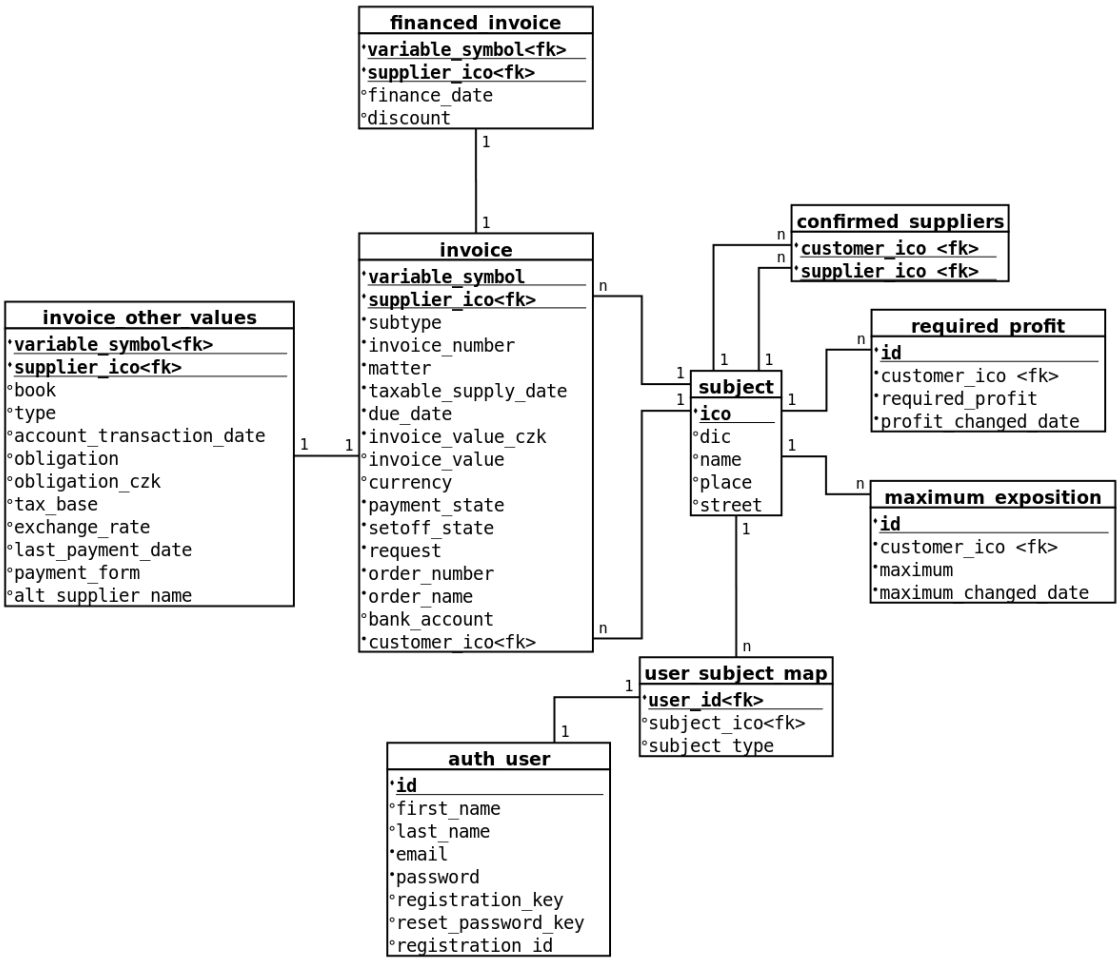
- `confirmed_suppliers` představuje schválené dodavatele, tedy dodavatele, kterým odběratel dovolí nechat si profinancovat faktury se skontem. Jeden odběratel tedy může mít více schválených dodavatelů, podobně jeden dodavatel může figurovat u více odběratelů. Jedná se tedy o vztah M:N mezi subjekty, který je nutno implementovat jako samostatnou tabulku – přesně tento účel tedy plní tabulka `confirmed_suppliers`. V mapovací tabulce se musí vyskytnout primární klíče zúčastněných entit, což v tomto případě představují atributy `customer_ico` a `supplier_ico`.

- `auth_user` je tabulka, kterou vytváří web2py. Obsahuje zejména přihlašovací údaje a další důležité informace o uživatelích. Za zmínku rovněž stojí i fakt, že hesla v této tabulce jsou automaticky ukládána v zašifrované podobě – *hash*.

- `user_subject_map` je tabulka, která obsahuje mapovaní jednotlivých uživatelů na subjekty. V podstatě se tedy jedná o tabulku pro vztah mezi tabulkami `auth_user` a `subject`. Důvod, proč je toto řešeno samostatnou tabulkou, je ten, že je třeba rozlišit odběratele od dodavatelů, ale firmy (subjekty) mohou teoreticky vystupovat jak v roli odběratele, tak v roli dodavatele. Proto je tento údaj nutný přiřadit uživatelům, přičemž není doporučeno zasahovat do tabulky `auth_user`, kterou vytváří web2py, a to zejména z bezpečnostních důvodů, neboť web2py obsahuje i zabezpečené přihlašovací formuláře, které automaticky pracují s touto tabulkou. Atribut `subject_type` tedy obsahuje rozlišení, zda se daný uživatel přihlásí do systému jako odběratel, nebo dodavatel.

- `required_profit` ukládá odběratelem nastavený požadovaný výnos. Jelikož je nutno ukládat i historii změn kvůli tomu, aby se daly spočítat průměrné výnosy za určité časové období, je zapotřebí ukládat i datum a nestačí pouze přepisovat hodnotu v jednom atributu. Z toho důvodu je zapotřebí změny ukládat v samostatné tabulce. Atribut `required_profit` obsahuje vlastní nastavený požadovaný výnos v procentech, zatímco atribut `profit_changed_date` představuje datum, kdy došlo ke změně.

- `maximum_exposition` ukládá odběratelem nastavovanou maximální expozici, tedy v podstatě objem volných prostředků, který je odběratel ochoten financovat. Obdobně jako v tabulce `required_profit` je i zde nutno ukládat historii změn kvůli výpočtům průměrného výnosu a průměrné míry utilizace, neboli využití prostředků za určité časové období. Obdobně atribut `maximum` je vlastní nastavovaná expozice a atribut `maximum_changed_date` je datum změny.



Obrázek 5.2: Finální schéma databáze

5.3.3 Uživatelské rozhraní

Grafický návrh a HTML šablonu uživatelského rozhraní jsem obdržel od firmy *Platební instituce Roger a.s.*, mým úkolem pak bylo zejména tuto šablonu upravit a převést do view ve web2py, tudíž se o návrhu uživatelského rozhraní zmíním jen stručně. Jak již bylo řečeno, tak frontend bude rozdělen na dvě části, a to část pro odběratele a část pro dodavatele. V původním návrhu byly dvě přihlašovací stránky pro každou část zvlášť, což jsem však upravil tak, že je jedna přihlašovací stránka společná pro obě části a po zadání emailu a hesla se rozhodne, do které skupiny (tedy odběratel, nebo dodavatel) daný uživatel patří, viz návrh databáze. Je to z toho důvodu, že web2py obsahuje pouze jednu tabulku pro uživatele – `auth_user`, což by znamenalo, že v případě, že by se například uživatel pro dodavatele pokoušel přihlásit na stránce pro odběratele, tak by ho web2py automaticky přihlásilo, neboť se email i heslo vyskytují v tabulce `auth_user`, následně by mu byl odepřen přístup na stránku a musel by být dodatečně odhlášen. S jednou přihlašovací stránkou tato starost odpadá a navíc přináší uživateli zjednodušení v tom, že nemusí řešit, zda je na správné stránce, prostě zadá své přihlašovací údaje, načež je přihlášen, a systém sám rozhodne, do které části frontendu. Další věci, které bylo nutno rovněž předělat, pak odhalila samotná implementace a zejména uživatelské testování, o tom už ale více v následujících kapitolách.

Kapitola 6

Implementace

Tato kapitola bude pojednávat zejména o implementační části. Jelikož bylo pro tvorbu aplikace využito web2py, tak bude nezbytné si popsat proces fungování aplikace a další.

Typická webová aplikace je založena na modelu klient-server, kde server poskytuje klientům nějakou službu. Server přijímá od klientů požadavky a odesílá jim odpovědi. Serverové část se tudíž obvykle označuje jako *backend*, neboť má na starosti vnitřní logiku, databázi a podobně, na rozdíl od klientské části (označované jako *frontend*), která vytváří uživatelské rozhraní. Klient s webovým serverem komunikuje zasíláním HTTP požadavků, načež server klientovi odpoví odesláním HTTP odpovědí, tedy typicky stránky v HTML. Dá se tedy říci, že v případě webové aplikace je serverová část většinou implementována v některém programovacím jazyce – v našem případě Python, jinak typicky např. Java, C#, PHP atd., zatímco klientská část je prakticky vždy tvořena HTML stránkou spolu s CSS a JavaScriptem, alespoň co se webových aplikací týká.

Jak již bylo řečeno, tak backend je implementován v Pythonu s frameworkem web2py. Proto bude nejdříve nutné si uvést některé principy a pravidla, které tento framework vyžaduje ke správnému chodu aplikace, a poté se zaměřím na konkrétní záležitosti řešeného systému.

6.1 Architektura

Nejprve několik slov k architektuře, na které je postavena každá web2py aplikace, tedy i tato. Jak již bylo uvedeno 4.2, tak web2py aplikace jsou založeny na použití návrhového vzoru MVC, což je zkratka pro model-view-controller.

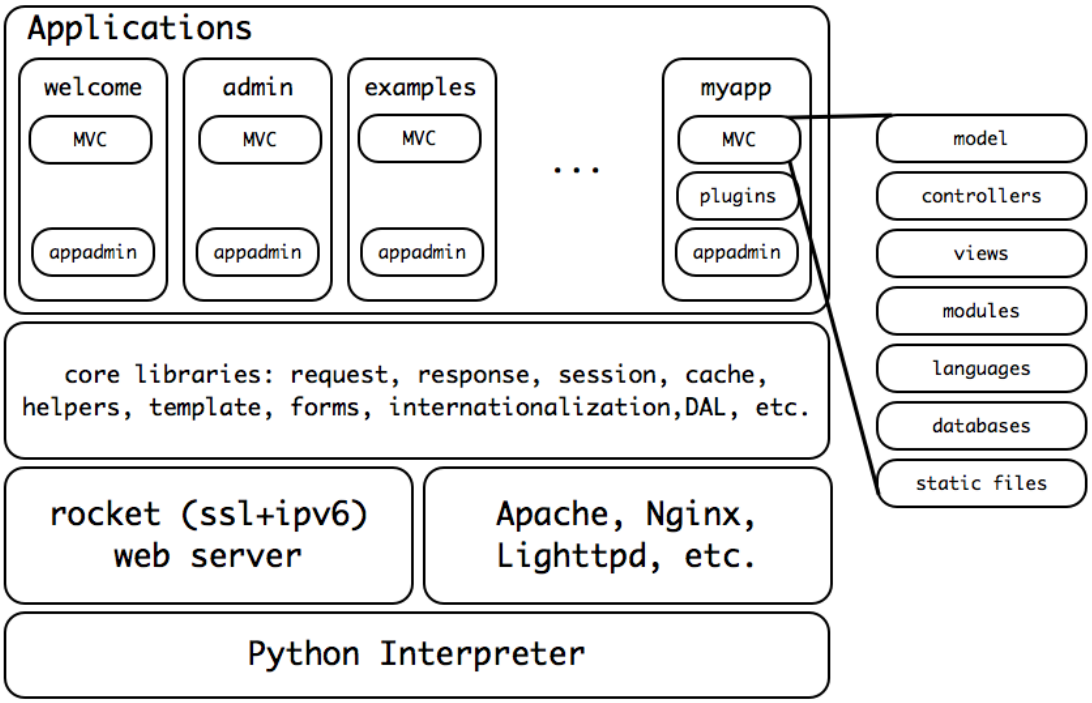
Ve web2py aplikaci model obsahuje zejména definici databázových tabulek, konfigurační nastavení, případně další globální proměnné. View, neboli pohled, se v podstatě stará o dynamické vytvoření HTML. Proto obsahuje typicky šablonu v HTML, do které jsou data vkládána dynamicky za použití šablonovacího jazyka, který je v podstatě modifikovaný Python, o tom už ale více dále. A konečně controller (řadič) se stará o celkovou funkčnost aplikace, tj. v případě nutnosti pokládá databázové dotazy, naplňuje pohled daty, obvykle za použití slovníku (dictionary) jazyka Python a další.[9]

6.2 Struktura aplikace

Celková struktura web2py je znázorněna na obrázku 6.1. Již jsem zmínil, že ve web2py může současně běžet více aplikací, přičemž každá aplikace má svou vlastní strukturu. Struktura aplikace ve web2py je složena z několika částí, od nichž se potom odvíjí i adresářová struktura každé web2py aplikace. Zde jsou uvedeny v abecedním pořadí:

- **cache** ukládá data (cache) pro rychlejší zpracování,
- **controllers** popisují aplikační logiku a procesy,
- **cron** umožňuje vykonávat předem naplánované úlohy nezávisle na platformě,
- **databases** ukládají metadata o tabulkách, případně přímo databáze SQLite,
- **errors** ukládají údaje o chybách, ke kterým došlo za běhu aplikace,
- **languages** popisují překlad řetězců v aplikaci do jiných jazyků,
- **models** popisují reprezentaci dat ve formě databázových tabulek,
- **modules** jsou případně další Python moduly,
- **privates** jsou soubory, ke kterým má přístup kontrolér, ale nikoliv vývojář,
- **sessions** ukládají informace vztahující se k jednotlivým uživatelům,
- **static** jsou sobory, které nejsou tvořeny dynamicky (např. CSS, JavaScript soubory atd.),
- **tests** obsahuje testy,
- **uploads** jsou soubory, ke kterým má přístup model, ale nikoliv vývojář (např. soubory nahrané uživateli),
- **views** popisují, jakým způsobem mají být data prezentovány uživateli pomocí HTML a JavaScriptu.

Dále typická aplikace obsahuje prázdný soubor `__init__.py`, který zajišťuje, aby Python mohl najít a importovat moduly z modules. Jedná se o typický vícesouborový modul. [3] Navíc by měly být přítomny soubory ABOUT a README, jejichž funkce je myslím zřejmá.[9]



Obrázek 6.1: Celková struktura web2py [9]

6.3 Proces zpracování požadavku

Zde bude popsán proces zpracování požadavku na server pro typickou web2py aplikaci, a to od obdržení požadavku až po odeslání odpovědi.

HTTP požadavky dorazí na webový server, který je zpracovává paralelně ve více vláknech. Zde je HTTP hlavička požadavku rozebrána a předána do tzv. *dispatcheru*, načež dispatcher určí, které z instalovaných aplikací je požadavek určen (ve web2py může současně běžet více aplikací). Následně namapuje cestu v URL na volání funkce – každému URL je tedy přiřazeno volání funkce. Například v případě, že je požadována stránka localhost/app/default/index, se zavolá funkce index z kontroléru default (soubor default.py) v aplikaci app. To je výchozí způsob směrování stránek, který lze ale samozřejmě změnit použitím směrovacích pravidel. Požadavky na soubory ve složce static jsou přímo vyřízeny, ostatní požadavky jsou mapovány na akce, tedy funkce v kontrolérech dané aplikace. Před vykonáním funkce jsou ještě vytvořeny, případně načteny, session objekty, a dále jsou vykonány modely, přičemž posloupnost vykonávání modelů je dána abecedním pořadím názvů souborů. Následně je spuštěna volaná funkce v příslušném kontroléru.

Důležitým aspektem funkcí v kontrolérech je jejich návratová hodnota, neboť podle návratových hodnot, respektive jejich typů, se určuje, co se má dít dál:

- **string** – pakliže je typem návratové hodnoty string, je přímo vložen do odpovědi bez použíti view (pohledu). Toho lze využít zejména při použití AJAX callback, kdy lze takto vrátit např. volání javascriptové funkce, případně přímo HTML element. Viz

příklad, kde je volána JavaScript funkce, která překreslí koláčový graf.

```
return 'pie_chart(' + str(round(pie_chart_value)) + ');'
```

- **dict** – pokud je návratovým typem dict, neboli slovník, pak je využit view, kde se hodnoty ze slovníku mohou vkládat na příslušná místa v HTML za pomoci šablonovacího jazyka. Toto je tedy asi nejběžnější typ návratové hodnoty funkce v kontroléru. Jako příklad zde uvádím návratovou hodnotu funkce *index* z kontroléru *default*, tedy obsahující přihlašovací formulář.

```
return dict(form=auth(), title_name='Login')
```

Obdobou tohoto návratového typu je také možnost vrátit ve slovníku všechny lokální proměnné ve funkci:

```
return locals()
```

- **HTML helper** – HTML helper je funkcionalita, která nabízí generování HTML tagů, jedná se tedy o obdobu návratové hodnoty string. Využití je stejně jako u stringu zejména při použití AJAX callbacků.
- Další možností pak je přesměrování na jinou stránku, což se neřeší návratovou hodnotou, ale zavoláním funkce redirect:

```
redirect(URL('customer', 'dashboard'))
```

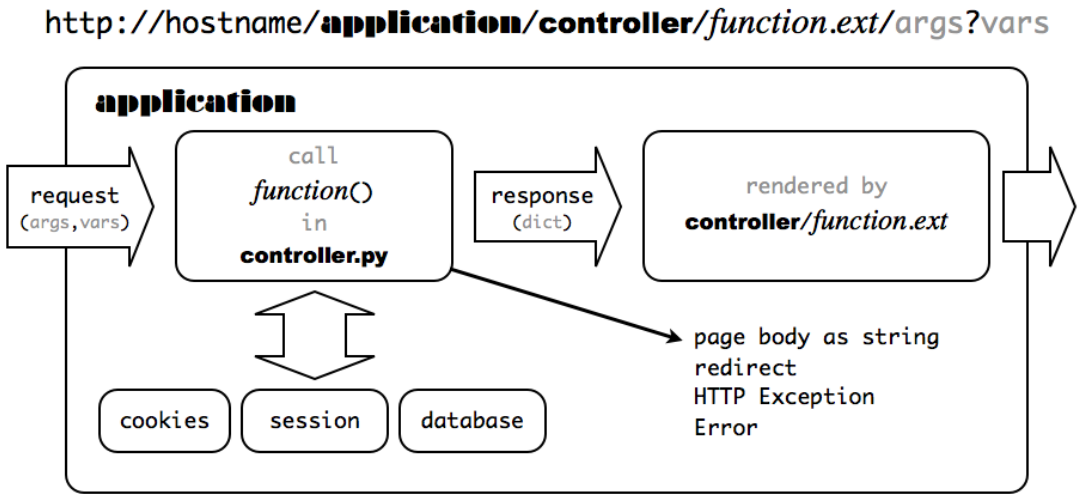
- Vrácení HTTP chybové stránky se zase řeší takto:

```
raise HTTP(404)
```

Pokud je tedy návratovou hodnotou funkce slovník, pak je použit pohled, do kterého jsou hodnoty ze slovníku vkládány za pomoci šablonovacího jazyka. Použití views je tedy podmíněno právě tím, že typem návratové hodnoty funkce v kontroléru je slovník. Z toho logicky vyplývá, že zatímco přítomnost příslušného kontroléru je nezbytná, view přítomen být vůbec nemusí, ovšem je zejména pro vracení celých HTML stránek opravdu silně doporučen, ne-li přímo přikázán.[9]

Následně je takto vytvořená stránka vrácena klientovi v HTTP odpovědi. Na obrázku 6.2 je znázorněn jak princip mapování URL na volání funkce, tak možnosti, které mohou nastat po jejím volání v závislosti na typu návratové hodnoty. Příslušný pohled (view) je pak umístěn v rámci adresářové struktury v adresáři *views* a dále v adresáři pojmenovaném podle kontroléru. Samotný název pohledu je pak shodný s názvem volané funkce. Function.ext tedy představuje pohled, přičemž .ext je přípona, která nemusí být přítomna, v tom případě se předpokládá html. Koncovka pak určuje koncovku použitého pohledu, který bude vykreslovat hodnoty vrácené z příslušného kontroléru. To umožňuje mít pro jeden kontrolér několik způsobu zobrazení (html, xml, json, rss atd.).

Celý kód je pak prováděn jako transakce, a pokud v průběhu vykonávání dojde k chybě, je transakce revertována a údaje o chybě jsou uloženy do tzv. ticketu, které jsou ukládány v adresáři errors. [9]



Obrázek 6.2: Postup zpracování požadavku [9]

6.4 Model

Model aplikace se nachází v adresáři *models*, konkrétně jde o soubor *db.py*. Obsahuje zejména konfiguraci autentizace, to znamená například, která stránka slouží jako přihlašovací, dále která stránka se má načíst, respektive která funkce z kontroléru se má zavolat po úspěšném nebo neúspěšném pokusu o přihlášení a podobně. Velmi důležitou části je samozřejmě také definice tabulek v databázi, viz dále. V modelech jsou definice veškerých globálních proměnných. Jelikož jsou soubory modelů zpracovávány v abecedním pořadí, platí, že definovaná globální proměnná je viditelná ve všech následujících modelech a v úplně každém controlleru a view.

6.4.1 Databáze

Web2py umožňuje definovat tabulky v databázi přímo v Pythonu, čímž poskytuje jakousi abstrakci od konkrétní implementace v nějakém SQL jazyce (případně NoSQL) a umožňuje přenositelnost aplikace na různé databázové stroje pouhou změnou řetězce připojení (anglicky *connection string*). Této abstrakci se říká *databázová abstrakční vrstva* – DAL. DAL kromě definic tabulek poskytuje i možnost vytváření dotazů, záznamů a dalších, a to vše v syntaxi Pythonu, načež dynamicky sestaví dialekt SQL jazyka pro zvolený databázový stroj.[9]

Co se týká konkrétních tabulek mnou použitých při řešení, tak ty je možno vyčíst z výše uvedeného ER diagramu 5.2. Dále databáze obsahuje tabulky vytvořené web2py automaticky, zejména se jedná o tabulku ukládající session – `web2py_session_infin` (kde infin je název aplikace), nebo například `auth_event`, která ukládá autentizační události. Pro názornost zde 6.1 uvádím konkrétní příklad definice tabulky ve web2py – jedná se

o tabulku subject (neboli firma). Malá poznámka k příkladu – ve web2py je implicitní datový typ atributu string.

Při zpracování modelu jsou aktuální definice tabulek porovnány s metadaty (migracemi) v adresáři *databases* a v případě, že se liší, jsou provedeny změny i v databázi.

```
db.define_table(
    'subject',
    Field('ico', 'bigint', notnull=True),
    Field('dic'),
    Field('name', notnull=True),
    Field('place'),
    Field('street'),
    primarykey=['ico']
)
```

Zdrojový kód 6.1: Příklad definice tabulky ve web2py.

6.5 Controller

V této podkapitole bude rozebrán podrobněji kontrolér. O view se lze dočíst o něco dále v klientské části.

V rámci mého řešení jsou kontroléry rozděleny do čtyř souborů podle uživatelských rolí, tj. kontrolér s názvem default, který se načítá jako první, obsahuje autentizační mechanismy, customer obsahuje logiku pro stránky odběratele, obdobně supplier pro dodavatele a admin pro administrátora. Od názvů kontrolérů se následně odvíjí i názvy použitých pohledů, viz dále.

Jak již bylo řečeno, tak jednotlivé funkce obsahují primárně logiku, tedy dotazy na databázi, následné zpracování výsledku, např. provedením nějakého výpočtu atd., a také samozřejmě vkládání hodnot do databáze, případně aktualizace, či jejich odstranění.

Dalším důležitým aspektem je zabezpečení funkce, zejména se tedy jedná o kontrolu, zda ten, kdo se snaží danou funkci zavolat, respektive načíst danou stránku, k tomu má dostatečné oprávnění. K ověření tohoto web2py používá dekorátory jazyka Python. Dekorátory umožňují pozměnit funkcionalitu funkce bez toho, aby ji přímo modifikovaly. [9] V kontextu zabezpečení funkce je tedy před jejím samotným voláním nejprve ověřeno, jestli k tomu má daný uživatel dostatečné oprávnění.

6.5.1 Moduly

Ke kontrolérům patří také moduly, neboť funkce importované z modulů budou volány zejména v kontrolérech. Jak již bylo uvedeno, tak moduly jsou umístěny v adresáři *modules*. Tento adresář je přidán implicitně do procházených adresářů pro importování modulů, tudíž při importu stačí uvés t standardně název souboru bez koncovky.

Nespornou výhodou použití modulů je fakt, že jsou zkompilovány při spuštění web2py na rozdíl od kontrolérů, které jsou kompilovány až po obdržení požadavku od klienta. Tedy stejný kód poběží pravděpodobně rychleji v modulu nežli v kontroléru. Jak je ovšem zřejmé, tak po startu web2py se změny v modulech neprojeví. K tomu, aby se projevily, je třeba web2py restartovat, což může být menší nevýhoda při vývoji, ovšem při nasazení už se

stejně nepočítá s tím, že by se kód v modulech často měnil, tudíž je z dlouhodobého hlediska výhodnější umístit kritické úseky kódu do modulů.

6.6 View

Jak již bylo zmíněno, tak view, neboli pohled, slouží k prezentaci, tedy zobrazení dat uživateli, což je většinou realizováno pomocí HTML. K tomu, aby mohly být do HTML dokumentu vložen kód v Pythonu, který má na starost dynamické generování, je třeba jej nějak odlišit. Ve web2py, stejně jako například v Djangu, je to realizováno za pomocí dvojice složených závorek. Kód v Pythonu mezi složenými závorkami nemusí být odsazovaný kvůli tomu, aby nebyla narušena pravidla pro odsazování HTML. Problém, který ale vzniká, je ten, že jelikož v Pythonu záleží na odsazování z důvodu rozpoznání ukončení bloku, je nutno bloky ukončit nějak jinak. K tomu používá web2py klíčové slovo Pythonu `pass`. Pak tedy lze nejen přímo do HTML kódu vkládat jednotlivé proměnné ze slovníku předaného do pohledu z kontroléru, ale i používat podmínky a cykly, což je typicky využitelné například u tabulek.[9]

6.7 Autentizace a autorizace

V této podkapitole se zaměřím na autentizaci a autorizaci, respektive obecně na řízení přístupu. To zahrnuje všechny tři části MVC aplikace, tedy modul, kontrolér i pohled – v modelu jsou definovány tabulky a konfigurace, kontrolér má na starosti zejména autorizaci (viz výše použití dekorátorů Pythonu) a přihlášení, a v pohledu jsou pak řešeny přihlašovací, případně registrační formuláře.

O autentizaci se ve web2py stará třída `Auth` z modulu `gluon.tools`. Pro využití této třídy je nejprve nutné vytvořit její instanci a nakonfigurovat ji, což se dělá v modelu. Tato třída poté automaticky vytvoří všechny potřebné tabulky v databázi, které jsou pro autentizaci, respektive autorizaci, nezbytné. Konstruktoru třídy je tedy nutno předat minimálně DAL objekt, tedy databázi, ve které budou potřebné tabulky vytvořeny. Samotná definice tabulek pak proběhne zavoláním metody `define_tables`, u které je možno za pomocí argumentu `username` stanovit, zda bude přihlašování probíhat pomocí uživatelského jména (hodnota True), nebo emailu (hodnota False), tedy zda vůbec bude uživatelské jméno vytvořeno. Asi nejdůležitější tabulkou je již zmíněná tabulka `auth_user`, viz schéma databáze 5.2, která obsahuje zejména email a heslo, přičemž heslo je zde ukládáno v zašifrované podobě. Konkrétně je výchozím způsobem 1000 iterací algoritmu PBKDF2 spolu s SHA512 a délkou výsledného hash 20 bytů. Konfigurace pak zahrnuje například definice toho, která stránka se má načíst, respektive která funkce kontroléru se má zavolat po různých událostech, například přihlášení, odhlášení, případně kam přesměrovat za předpokladu, že je již někdo přihlášen a podobně.

Co se týká autorizace, tak ta je ve web2py realizována zejména v kontrolérech za použití již zmíněných dekorátorů jazyka Python. Celé to funguje tak, že pokud není splněna podmínka předaná dekorátoru jako argument, pak je přístup k dané funkci zamítnut. Důležitá je proměnná `auth.user_id`, která obsahuje ID aktuálně přihlášeného uživatele, pod ID je zde myšlen primární klíč `id` v tabulce `auth_user`. Dekorátorů ověřujících splnění nějaké podmínky je samozřejmě mnohem více, přičemž `auth.requires(condition)` je ten nejobecnější, další je například `auth.requires_login()`, který pouze ověří, zda je uživatel přihlášen. Požadavky v dekorátorech je samozřejmě možno i kombinovat.

V kontrolérech pak bude, kromě již zmíněné autorizace v podobě dekorátorů u každé funkce, například vytvoření přihlašovacího formuláře, případně funkce `user`, která umožní další akce, jako odhlášení, změnu hesla a podobně. Vzhled formulářů pak lze zcela změnit v pohledech, což je i můj případ.

6.8 Výpočet skonta

Ke klíčovým aspektům systému patří výpočet dynamického skonta pro každou fakturu. Teorie výpočtu skonta již byla uvedena výše 2.4, zde se zaměřím především na implementaci. Jelikož výpočet skonta patří ke klíčovým prvkům, je lepší zvolit možnost jej umístit do modulu než jej ponechat přímo v kontroléru. To zejména ze dvou důvodů: jak již bylo řečeno, tak kód v modulech je překládán při spuštění web2py, tudíž při vzniku požadavku je již pouze interpretován, pracuje tedy rychleji. Dalším důvodem je testovatelnost, neboť testovat přímo kontrolér, který vrací většinou slovník, není úplně tím nejlepším řešením.

Ve zdrojovém kódu 6.2 se nachází funkce `calculate_discount`, která řeší právě výpočet dynamického skonta. Proměnná `days_to_due_date` představuje počet dní, které zbývají do data splatnosti, `invoice_phase` pak je fáze faktury, tedy počet dní do splatnosti dělená celkovou dobou splatnosti, viz 2.4. Z databáze se následně získá aktuální požadovaný výnos odběratele, a pokud si ho odběratel dosud nenastavil, pak je výchozí hodnota 1%. Požadovaný výnos odběratel ovšem zadává ve formě úroku p.a., je tedy nezbytné jej ještě přepočítat na dobu do splatnosti, což je realizováno výrazem: počet dní do splatnosti děleno počet dní v roce, který je, alespoň co se týká běžně používané praxe při počítání úroků, kvůli zjednodušení výpočtů stanoven dohodou na 360, což byl i požadavek firmy *Platební instituce Roger a.s.* Celkový vzorec, který již byl popsán výše 2.4, je vidět ve zdrojovém kódu, jen je nutno ještě ošetřit případ, kdy při nastavení malého požadovaného úroku a malého počtu zbývajících dní vyjde dynamické skonto na hodnotu menší než jedna. Minimální cena však musí být fixní na 1%, což bylo požadováno firmou. Dále je výsledek zaokrouhlen na dvě desetinná místa a pro jednotnost převeden na datový typ *Decimal*.

```
def calculate_discount(db, invoice):
    """
        Calculates dynamic discount for invoice given in function parameter

        :param db: DAL object - database which required data will be read
    from
        :param invoice: concrete row in the table invoice

        :return dynamic discount for invoice rounded to 2 decimal places
        """
    days_to_due_date = (invoice.due_date - datetime.date.today()).days

    invoice_phase = (Decimal(days_to_due_date)
                     / (invoice.due_date
                        - invoice.taxable_supply_date
                        ).days
                     )

    required_profit_row = db(
        (db.required_profit.customer_ico == invoice.customer_ico
         )).select(db.required_profit.required_profit,
                   orderby=~db.required_profit.id
                   ).first()

    if required_profit_row is not None:
        required_profit = required_profit_row.required_profit
    else:
        required_profit = 1

    discount = ((required_profit * (Decimal(days_to_due_date) / 360) - 1)
                * invoice_phase
                ) + 1

    if discount < 1:
        discount = 1

    return Decimal(round(discount, 2))
```

Zdrojový kód 6.2: Implementace výpočtu dynamického skonta

6.9 Nahrávání faktur

Jak již bylo uvedeno, tak nahrávání faktur probíhá prostřednictvím csv souboru. Je to dáno tím, že faktur nahrávaných do systému bude velké množství, řádově tisíce. Pro zpracování csv souborů byl použit modul `csv`, který je součástí jazyka Python.[3]

Problémem, se kterým jsem se zde potýkal, byly časté změny struktury csv souborů, což následně vedlo i na časté změny implementace. Použití csv souboru pak není jediná možnost, další zahrnují například XML nebo JSON, do budoucna bude nezbytné zavést jednotnou strukturu a zavést podporu pro různé formáty.

Nahrávání faktur do systému pak bude mít na starosti odběratel v rámci své části frontendu.

6.10 Frontend

V této podkapitole se zaměřím na implementovaný systém z pohledu uživatele, tedy primárně na frontend. Jelikož se jedná o webovou aplikaci, tak frontend tvoří primárně HTML, zde ve formě view, viz výše. Další neméně důležitou součástí je JavaScript, ve kterém jsou implementovány především zpracování událostí uživatelského rozhraní, grafy v odběratelské části a různé další animace. Pro zpracování událostí, pro které je potřeba požadavek na server, je ve většině případů použit AJAX.

6.10.1 Přihlášení

Jak již bylo uvedeno, existuje pouze jedna stránka pro přihlášení, a po zadání hesla je uživatel automaticky přesměrován do té které části systému (tedy odběratel, dodavatel, případně administrátor), podle toho, kam má přístup.

6.10.2 Frontend pro odběratele

Frontend pro odběratele představuje tu část systému, do které se přihlašuje uživatel pod účtem odběratele. Zde se tudíž nachází nastavování podmínek financování, výse maximální expozice, schvalování dodavatelů a podobně, viz dále.

Dashboard

Po přihlášení pod účtem odběratele se zobrazí dashboard s přehledem nejdůležitějších informací a údajů. Dashboard je rozdělen do čtyř částí – Expozice, Podmínky financování, Faktury a Dodavatelé. Část Expozice obsahuje údaje o množství aktuálně financovaných prostředků, maximální množství volných prostředků a aktuální míru využití maximální sumy. Podmínky financování pak obsahují zejména údaje o výnosech a aktuální hodnotu požadovaného výnosu, od kterého se pak odvíjí hodnota dynamického skonta. Faktury obsahují informace o počtu aktuálně platných faktur a celkovém počtu faktur, které má daný odběratel v systému. Část dashboardu Dodavatelé pak obsahuje počty schválených a neschválených dodavatelů. Pro každou z výše jmenovaných částí dashboardu existuje stránka s detailními informacemi obsahující podrobnější informace nebo grafy a umožňující změny hodnot.

Expozice

Stránka Expozice je rozdělena na dvě části, přičemž horní obsahuje číselné údaje, spodní pak grafy. V horní části je kromě hodnot zobrazených už v dashboardu, tedy množství aktuálně financovaných prostředků, maximální expozice (volných prostředků) a aktuální utilizace (poměr využití expozice), i údaj o průměrné utilizaci v současném roce. Údaj o maximální expozici je nastavitelný – zde si tedy odběratel určí, jaké množství prostředků je ochoten financovat.

Ve spodní části jsou pak grafy: koláčový graf obsahující poměr zastoupení odběratelovy expozice na celkovém financování dodavatelů a sloupcový graf obsahující přehled průměrných měsíčních utilizací, tedy procentuální hodnoty využití nastavované maximální expozice. Grafy jsou implementovány pomocí JavaScriptu.

Podmínky financování

Obdobně jako u Expozice, i zde je stránka rozdělena na dvě části – horní obsahuje číselné údaje o podmínkách financování, spodní pak graf s měsíčními výnosy. Údaje v horní části jsou požadovaný výnos v procentech p.a., který lze měnit, a tím i určovat požadovanou výši skonta. Dále Celkový výnos za aktuální rok, tedy výnos ze všech financovaných faktur, a to i z faktur financovaných externími prostředky. Oproti tomu Výnos naší expozice obsahuje výnos pouze z prostředků přímo financovaných odběratelem, tedy z nastavované maximální expozice.

Sloupcový graf ve spodní části pak znázorňuje měsíční přehled výnosů, a to jak výnosu z expozice odběratele, tak celkový výnos.

Faktury

Zde se nachází primárně tabulka faktur. Je možné přepínat mezi tabulkou všech faktur v systému a tabulkami pouze aktuálních faktur, tedy faktur, jejichž doba splatnosti ještě nenastala. Obě tabulky je možno filtrovat podle názvu dodavatele, nominální hodnoty faktury, statusu faktury, tedy jestli byla, nebo nebyla financována, a data splatnosti. Samozřejmostí je rovněž nastavení počtu řádků v tabulce na stránku. Za zmínku stojí fakt, že počet řádků je tím nastaven pro tabulku dodavatelů a obráceně, což je zařízeno pomocí tzv. *cookies*.

Dodavatelé

Zde lze zobrazit všechny dodavatele, ti jsou rozděleni na schválené a neschválené. Schváleným dodavatelům je na rozdíl od neschválených umožněno nechat si faktury proplatit dříve s dynamickým skontem. Schvalování dodavatelů probíhá přímo zde, a to takovým způsobem, že odběratel si vybere dodavatele, případně i více dodavatelů, pomocí tzv. *checkboxu*, a následně klikne na tlačítko Potvrdit. Obdobným způsobem pak probíhá i ve druhé tabulce rušení schválených dodavatelů. V tabulce je možno vyhledávat pomocí názvu dodavatele.

6.10.3 Frontend pro dodavatele

Nejdůležitějším aspektem frontendu pro dodavatele je možnost výběru a následného financování faktur, což tvoří základ cele této části aplikace.

Přehled faktur

Po přihlášení pod účtem dodavatele se zobrazí jako první Přehled faktur. V přehledu faktur vidí dodavatel faktury, které si může nechat financovat, tedy faktury, které splňují všechny potřebné podmínky, jmenovitě například to, že dodavatel je mezi schválenými, dále samozřejmě faktura musí byt aktuální, tedy datum splatnosti ještě nenastalo, a další. Výběr faktur probíhá obdobně jako v odběratelské části při schvalování dodavatelů – tabulka přímo obsahuje *checkbox*, pomocí čehož si dodavatel může vybrat jednu, nebo i více faktur, a to podle toho, kolik prostředků zrovna potřebuje, a podle skonta, které je pro něj nejvýhodnější. Za zmínku stojí i fakt, že okamžitě po výběru faktur se dodavateli objeví informace o součtu vybraných faktur a skontu, které z nich bude strženo, což je implementováno pomocí AJAXu.

Přehled minulosti

V této části si dodavatel může zobrazit faktury, které si nechal v minulosti financovat se skontem. Záznamy je možno filtrovat podle nominální hodnoty a období faktury.

6.10.4 Frontend pro administrátora

O frontendu pro administrátora se zmíním spíše stručně. Frontend pro administrátora obsahuje přidávání subjektů, přidávání faktur ve formátu csv a registraci uživatelů.

U přidávání faktur ve formátu csv je nutno kromě csv souboru zadat i IČO odběratele, od kterého byl csv soubor obdržen, neboť v některých případech se nemusí číslo odběratele vyskytovat přímo v csv souboru, protože tento soubor je ve většině případů generován z interních systémů odběratele. To je velký problém, neboť v podstatě co odběratel, to jinak strukturovaný csv soubor. Obecně řečeno je současné nahrávání faktury spíše provizorní řešení, v budoucnu bude snaha o to, aby se faktury od odběratelů získávaly nějakým jednotným, předem dohodnutým způsobem pomocí EDI. K tomu však bude zapotřebí řekněme zaběhnutí systému a samozřejmě také větší zapojení jednotlivých odběratelů.

Kapitola 7

Testování

V této kapitole se zaměřím na testování systému. Pod pojmem testování se rozumí zkoumání, zda systém splňuje požadavky, cílem pak je odhalit co nejvíce chyb a nedostatků systému.

7.1 Unit testy

Unit testy, neboli jednotkové testy, jsou takové testy, které se, jak již název napovídá, zaměřují na testování určitých částí systému, tedy jednotek, přičemž onu jednotku většinou představuje funkce, případně třída a její metody. Jednotkový test tedy testuje funkčnost pouze této jednotky.

Pro Unit testy v jazyce Python existuje modul `unittest`, viz [3], který jsem využil i já. Základem je třída `TestCase`, představující třídu pro testovací případy, ze které lze dědit a tím definovat konkrétní testovací případy. Jednotlivé testy pak představují uživatelem definované metody, jejichž názvy začínají slovem `test`, tedy např. `testCalculatingDiscount`, což je nezbytně nutné k tomu, aby byly testy rozpoznány. Dalšími metodami této třídy, o kterých je dobré něco vědět, jsou metody `setUp`, která je volána automaticky před samotným testováním, a `tearDown`, která je volána po skončení všech testů. Důležitými metodami pak jsou samozřejmě metody pro ověření platnosti dané podmínky (`assertTrue, assertEqual atd.`).[3]

Při testování web2py aplikace lze využít metody používané pro testování běžné aplikace napsané v jazyce Python, jen je nutno přidat pár drobností – zejména co se týká spouštění testů, jelikož web2py aplikace nebude interpretována přímo, ale prostřednictvím web2py. Na rozdíl od běžné aplikace, kde se pro spuštění všech testů používá zavolání funkce `unittest.main`, to při testování web2py aplikace tímto způsobem nepůjde, neboť je nutno testy spouštět v prostředí web2py a navíc je nutno použít další argumenty, například pro načtení modelů. Proto je nezbytné vytvořit samostatnou testovací sadu, tedy instanci třídy `TestSuite`, do ní přidat všechny požadované testovací případy, a konečně tuto sadu spustit, tak jak je znázorněno ve zdrojovém kódu 7.1. Spuštění testů ve web2py prostředí se pak provádí obdobně, jako by se provedlo spouštění samostatného web2py skriptu, tedy konkrétně příkazem:

```
python2.7 web2py.py -S infin -M -R tests.py
```

Zde `-S infin` představuje název aplikace, `-M` zajistí načtení všech modelů a `-R tests.py` představuje cestu k samotnému skriptu, tedy v tomto případě testům. [9]

Při jednotkovém testování jsem se zaměřil zejména na testování nejdůležitějších výpočtů, které jsou umístěny v samostatných modulech. To ovšem neznamená, že by se kód umístěný v kontrolérech nedal testovat, jen je nutno z něj dané funkce načíst například prostřednictvím `execfile` a nastavit hodnoty v požadavku (objekt třídy `Request`). Oproti tomu kód v modulu lze importovat přímo a není nutné simulovat požadavek, dá se tedy říct, že je to o něco jednodušší.

```
suite = unittest.TestSuite()
suite.addTest(unittest.makeSuite(TestCalculations))
unittest.TextTestRunner(verbosity=2).run(suite)
```

Zdrojový kód 7.1: Spouštění testovacího případu pro web2py aplikaci

7.2 Uživatelské testování

Pod pojmem Uživatelské testování se rozumí testování systému, do kterého jsou zapojeni i jeho budoucí uživatelé. V mém konkrétním případě se jednalo zejména o zaměstnance firmy *Platební instituce Roger a.s.*, případně zástupci dalších firem, které projevily o systém zájem. Od uživatelů systému se očekává alespoň základní znalost dané problematiky, tedy zejména znalost názvosloví používaného ve vztahu k fakturám a účetnictví a dále základní znalost principu fungování dynamických skont.

Při uživatelském testování byla odhalena řada chyb a nedokonalostí především v uživatelském rozhraní, neboť se ukázalo, že budoucí uživatelé nerozumí některým názvům. Z toho důvodu bylo mnoho názvů změněno, či jinak poupraveno, a dále přibyla funkcionalita tzv. *hoover*, tedy že se po najetí kurzoru myši na určitý text, případně oblast, objeví textová nápověda vysvětlující daný text. Také došlo k úpravě grafů v odběratelské části frontendu, která zahrnovala zejména přidání popisků k jednotlivým osám grafu, dále byly přidány legendy a podobně.

Kapitola 8

Závěr

Cílem mé bakalářské práce bylo vytvořit systém dynamických skont podle požadavků zadavatele – firmy *Platební instituce Roger a.s.* Řešení tedy zahrnovalo fáze od analýzy požadavků přes návrh až po implementaci a testování. K tomu bylo samozřejmě nezbytné nastudovat i teoretické znalosti technologií, zejména pak frameworku web2py.

8.1 Zhodnocení

První fází bylo nastudování principů fungování dynamický skont a analýza požadavků firmy *Platební instituce Roger a.s.* Výsledkem této fáze bylo vytvoření diagramu aktivit a diagramů případů užití, které, díky grafickému znázornění fungování systému, přispěly k pochopení celého systému jako celku a následnému ověření validity.

Poté jsem se zaměřil na studium technologií, které budou využity, zejména tedy na framework web2py. Po získání nezbytných znalostí a provedení analýzy požadavků pak bylo možné přistoupit k samotnému návrhu.

V rámci návrhu bylo nezbytné provést zejména návrh databáze, což čítá vytvoření ER diagramu, který umožňuje abstraktní znázornění databáze pomocí entit a vztahů, čímž usnadňuje pochopení, a schématu databáze, které přináší grafické znázornění databáze. Návrh databáze, zejména pak schéma databáze, bylo následně konzultováno se zaměstnanci firmy *Platební instituce Roger a.s.*, čímž byla zajištěna validita. Následující fázi pak tvořila implementace, která představovala nejdelší úsek. První byla vytvořena databáze podle předcházejícího návrhu, jako další pak zbylé části systému počínaje odběratelskou částí. Dosažené výsledky byly opět průběžně konzultovány ve firmě *Platební instituce Roger a.s.*, čímž byla zajištěna zpětná vazba. Od firmy byla obdržena i testovací data v podobě konkrétních faktur ve formátu csv. Právě nahrávání faktur do systému představovalo pravděpodobně ten největší problém, neboť nebyla definována požadovaná struktura, což je vzhledem k širokému spektru budoucích uživatelů a jejich systémů poměrně obtížné zajistit, do budoucna je to však nezbytně nutné.

V rámci testování pak bylo nutné ověřit zejména funkcionalitu výpočtu dynamických skont a dalších klíčových výpočtů.

Výsledný systém bude po sjednocení nahrávání faktur a dalších drobných úpravách, zahrnujících například notifikaci odběratelů a dodavatelů, připraven k nasazení.

8.2 Možnosti pokračování

Pokračování práce systému zahrnuje několik důležitých i méně důležitých věcí. K těm důležitým patří zejména sjednocení struktury nahrávání faktur, případně i zavedení podpory pro více formátů, tedy nejen csv. Navíc by bylo dobrým rozšířením využití EDI, automatický export faktur z účetních systému odběratelů, a jejich následný automatický import do systému, což však vyžaduje řekněme zaběhnutí systému, zejména kvůli nezbytnosti zapojení odběratelů. Důležité rozšíření by pak mohla být i notifikace emailem, což bude pravděpodobně nezbytné zejména kvůli zasílání dobropisů odběratelům.

Dalším užitečným rozšířením, které pravděpodobně bude v budoucnosti implementováno, je napojení na bankovní účty jednotlivých subjektů, což je umožněno díky nové evropské směrnici *PSD2*, a na základě čehož pak bude možné přečíst informace o výši volných prostředků odběratele, a podle toho následně určit i výši maximální expozice, ale mohla by být díky tomu i přidána možnost odběratele uskutečňovat platby přímo v rámci systému pro dynamická skonta, což mu usnadní práci. Dodavateli by se pak mohl zobrazovat aktuální stav jeho účtu, podle čehož by se mohl rozhodovat, zda přistoupí k využití dřívějšího proplacení faktury výměnou za skonto v případě hrozící platební neschopnosti.

Další rozšíření, které by mohla vést ke zvýšení uživatelského komfortu, pak představuje například zavedení lokalizace, tedy podpora více jazyků, podpora převodu měn a podobně. Spíše kosmetickými úpravami ve frontendu by pak mohlo být přidání různých nahrávacích animací (při použití AJAX callback), které by uživatele informovaly o probíhající akci, případně potvrzujících oken pro kontrolu provedení akcí a podobně.

Literatura

[1] Doernhoefer, M.: JavaScript. *ACM SIGSOFT Software Engineering Notes*, ročník 31, č. 4, 2006: s. 16–24, ISSN 0163-5948, doi:10.1145/1142958.1142972.

[2] Gelsomino, L. M.; Mangiaracina, R.; Perego, A.; aj.: Supply Chain Finance: Modelling a Dynamic Discounting Programme. *Journal of Advanced Management Science*, ročník 4, č. 4, 2016: s. 283–291, doi:10.12720/joams.4.4.283-291.

[3] Pilgrim, M.: *Ponořme se do Python(u) 3*. CZ.NIC, 2010, ISBN 978-80-904248-2-1.

[4] *Platební instituce Roger*. [Online; navštíveno 26.3.2018].
URL https://roger.cz/

[5] *PEP 0 – Index of Python Enhancement Proposals (PEPs)*. [Online; navštíveno 27.3.2018].
URL https://www.python.org/dev/peps/

[6] *PEP 8 – Style Guide for Python Code*. [Online; navštíveno 27.3.2018].
URL https://www.python.org/dev/peps/pep-0008/

[7] Písek, S.: *HTML: Začínáme programovat*. Grada, 2014, ISBN 978-80-247-5059-0.

[8] Silberschatz, A.; Korth, F. H.: *Database system concepts*. McGraw-Hill, 2006, ISBN 0-07-295886-3.

[9] *web2py - book*. [Online; navštíveno 27.3.2018].
URL http://www.web2py.com/book

Příloha A

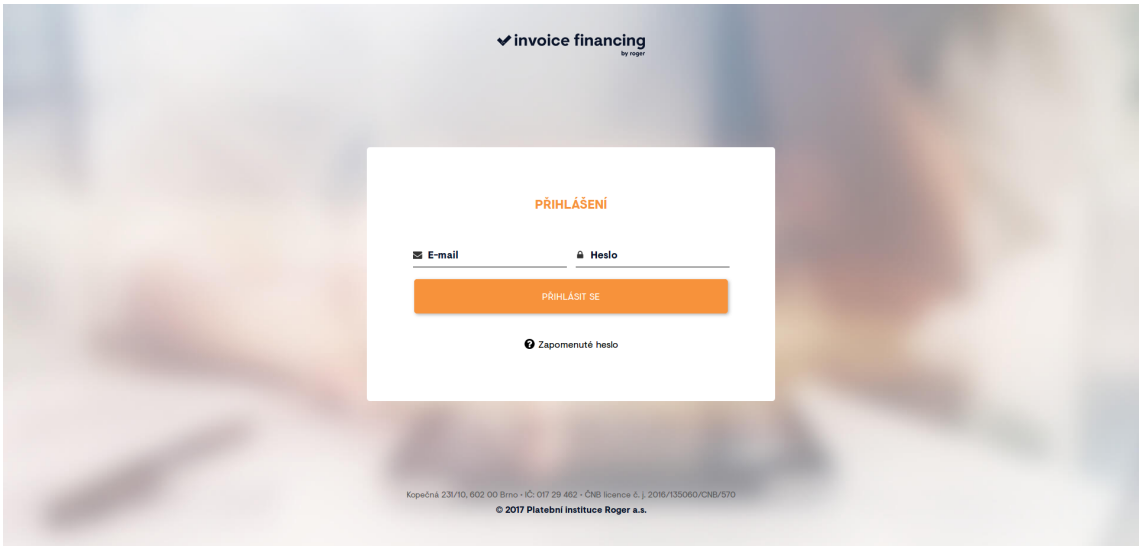
Obsah přiloženého CD

Přiložené CD obsahuje:

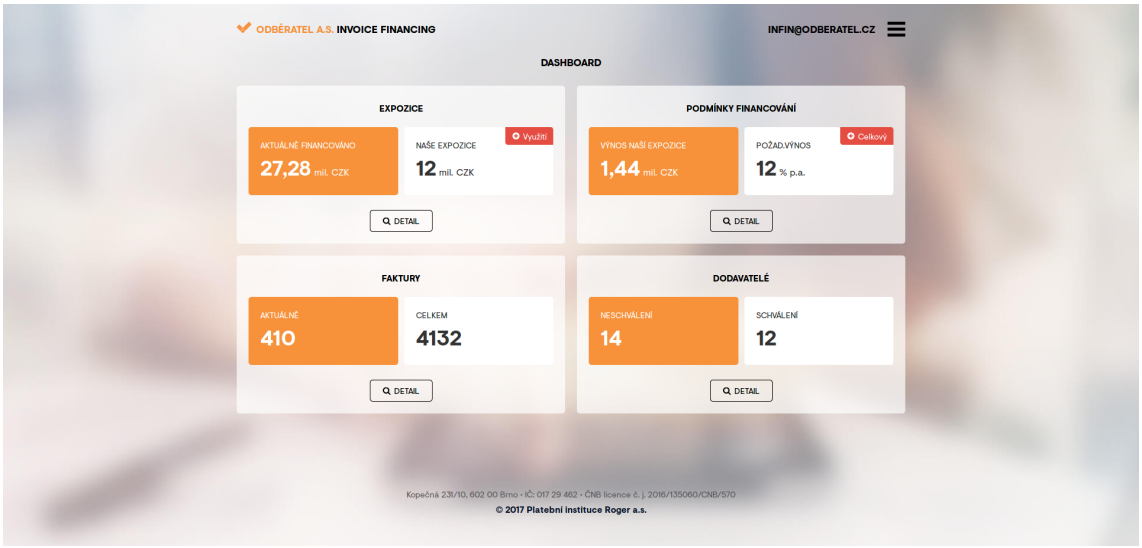
- písemnou zprávu ve formátu PDF,
- zdrojový tvar písemné zprávy v systému LaTeX,
- návod k instalaci, README,
- zdrojové texty informačního systému.

Příloha B

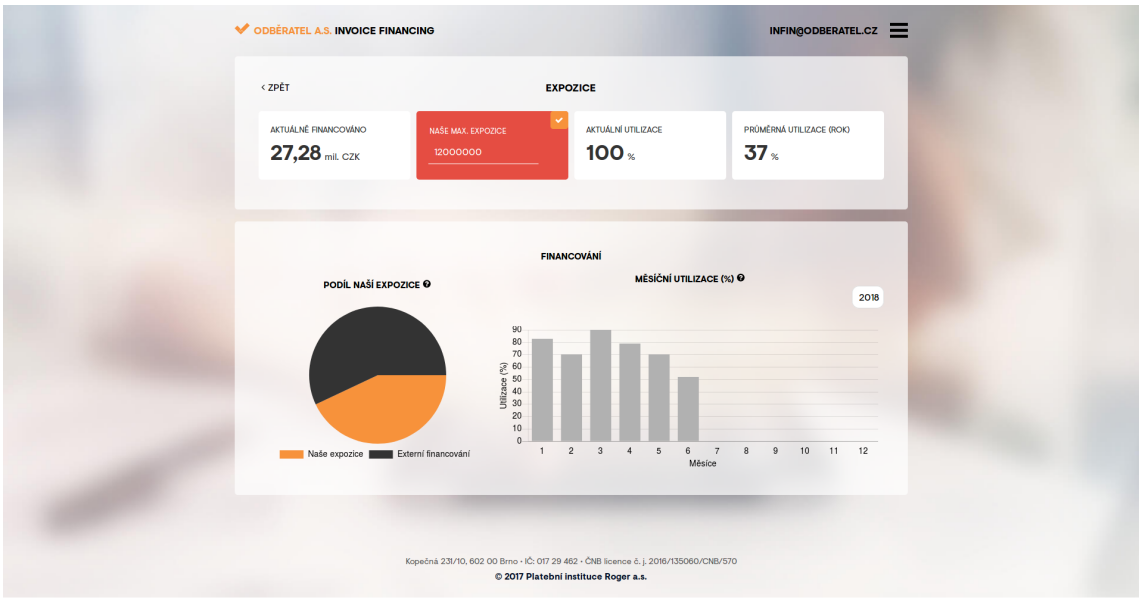
Snímky z aplikace



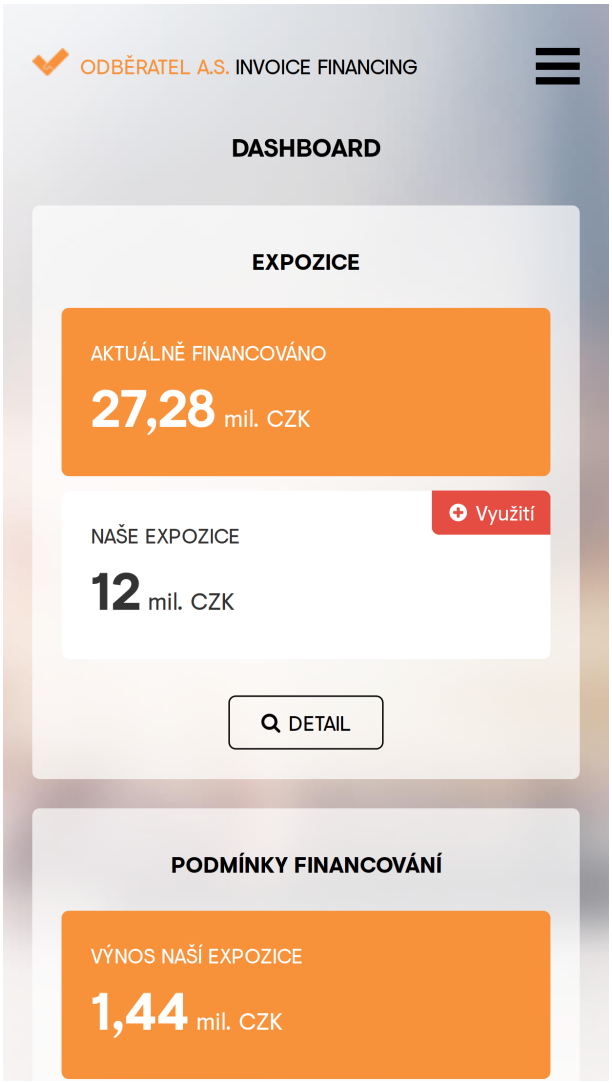
Obrázek B.1: Přihlášení



Obrázek B.2: Dashboard



Obrázek B.3: Expozice – při editaci maximální expozice



Obrázek B.4: Dashboard – responsivní design