České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačové grafiky a interakce

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Vojtěch Koukal**

Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Název tématu: **Komplexní informační systém neziskové organizace**

Pokyny pro vypracování:

Analyzujte potřeby zpracování a uchovávání dat (o zaměstnancích, dětech, projektech apod.) v neziskové organizaci Ekocentrum Podhoubí. Následně navrhněte a implementujte informační systém pro komplexní správu těchto dat. Systém bude umožňovat intuitivní vkládání a editaci těchto dat, stejně tak i získávání potřebných informací z nich. Systém nasaďte a otestujte na reálných uživatelích z cílové organizace. Systém implementujte na platformě Java EE.

Seznam odborné literatury:

Martin Fowler. 2002. Patterns of Enterprise Application Architecture. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Vedoucí: Ing. Tomáš Černý, MSc.

Platnost zadání: do konce letního semestru 2015/2016

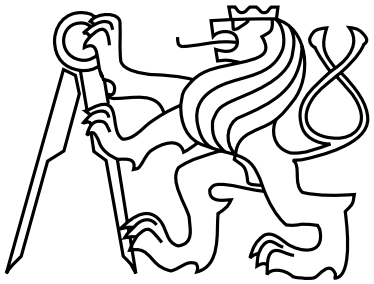
[redacted]
prof. Ing. Jiří Zára, CSc.
vedoucí katedry

L.S.

[redacted]
prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 1. 12. 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Diplomová práce

Komplexní informační systém neziskové organizace

Bc. Vojtěch Koukal

Vedoucí práce: Ing. Tomáš Černý MSc.

Studijní program: Otevřená informatika, Magisterský

Obor: Softwarové inženýrství

5. května 2015

Poděkování

Rád bych poděkoval svému vedoucímu diplomové práce Tomáši Černému za cenné rady při zpracování a za čas strávený nad touto prací. Dále děkuji celému kolektivu Ekocentra Podhoubí za možnost realizace této práce a za čas, který se mnou strávili.

Prohlášení

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

V Berouně dne 4. 5. 2015 .....

Abstract

The aim of this thesis is to design, create, deploy and test a new information system for non-profit organization Ekocentrum Podhoubí. Currently used system, composed from isolated and not uniformly-formatted Microsoft Excel sheets (or Google Docs), is unsustainable. This system also causes lot of troubles to orient in for newly incoming employees. Furthermore there is no uniform data storage for documents and so individual employees have those documents only on their computers and are sending them to other employees via e-mail. This situation doesn't allow for simple information sharing between employees and can cause serious lost of data (there is no backup available). The newly created system will allow complex data and information management.

Keywords: Java EE, information system, document storage

Abstrakt

Cílem této práce je navrhnout, vytvořit, nasadit a otestovat nový informační systém pro neziskovou organizaci Ekocentrum Podhoubí. Nyní používaný systém skládající se ze vzájemně izolovaných a nejednotně formátovaných tabulek v MS Excel nebo na Google Drive je dlouhodbě neudržitelný a nově příchozím pracovníkům působí velké problémy se v tabulkách zorientovat. Navíc neexistuje jednotné úložiště dat, dochází tedy k situacím, kdy jednotliví pracovnící mají všechna data uložena jen na svých počítačích a ostatním je posílají e-mailem. Tato situace neumožňuje efektivní sdílení informací mezi zaměstnanci a hrozí zde riziko ztráty dat, jelikož neexistuje žádný způsob jejich zálohování. Tento nově navrhnutý systém bude umožňovat komplexní správu dat a informací v této organizaci.

Klíčová slova: Java EE, informační systém, ukládání dokumentů

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Ekocentrum Podhoubí (dále jen Organizace), jakožto organizace, která bude výstup této práce využívat, je nezisková organizace zaměřující se na ekologické vzdělávání a osvětu.

Zajišťují také běh dvou poboček ekoškolky (Troja a Braník). V současnosti zaměstnává cca 20 pracovníků a v rámci Ekoškolky se stará o cca 30 dětí.

1.1 Ekocentrum

V ekocentru probíhá v průběhu roku několik programů a jiných aktivit, které je vhodné evidovat a sledovat. Zde krátce popíšu aktivity, které se budou týkat této práce.

1.1.1 Zaměstnanci

V Organizaci je pevné jádro cca 5-6 lidí, kteří jsou zaměstnáni na celý úvazek a zajišťují chod organizace. Jde zejména o ředitelku organizace, ředitelku školek, finančního manažera, administrátora EVP, marketingového manažera apod. Dále jsou v organizaci dvě provozní manažerky, které se starají o denní běh jednotlivých poboček.

Jako další zde vystupují lektoři EVP, což jsou dlouhodobí brigádníci zaměstnáni na Dohodu o provedení práce(DPP). Ti mají na starost přípravu a vedení EVP pro školy a školky.

Dále se v průběhu roku vystřídají v organizaci různí další brigádníci na pomocné práce, jako např. malování, stěhování apod. Ti bývají placeni hodinově na základě DPP.

1.1.2 Projekty

V organizaci probíhá současně několik projektů z dotačních programu města Prahy (magistrátní granty, OPPA), operačního programu vzdělávání a konkurenceschopnost(OPVK) a jiných. Ke každému projektu se vztahuje velké množství informací, které by bylo vhodné evidovat na jednom místě.

Každý projekt má svého projektového manažera z dotační organizace, projektového manažera z organizace a další kontakty. Ke každému projektu přísluší dokumenty (smlouva, výkazy, monitorovací zprávy), kalendář s upomínkami a indikátory. Indikátor je vlastnost,

která je dána při uzavření smlouvy na projekt a která musí být do ukončení projektu splněna. Jako příklad takového indikátoru ve vzdělávacím programu může být minimální počet úspěšných účastníků.

1.1.3 Interní záležitosti organizace

Jako každá jiná organizace i tato musí vést značné množství administrativy. Většinou se jedná o rutinní záležitosti ve kterých by mohl informační systém pomoci.

V tomto systému se bude zejména jednat o:

- Evidenci majetku.
- Evidenci dokumentů (zápisy z porad, interní směrnice a pokyny apod.).
- Evidenci kontaktů.
- Pokladní deník s příjmy a výdaji.
- Kalendář důležitých úkolů a upomínek.

1.1.4 Ekoškolka

Organizace založila a vede ekoškolku s dvěmi pobočkami - v Braníku a Troji. Do každé školky rodiče přihlašují své děti na různé druhy docházek. V základu jsou tři druhy docházky pro děti - dopolední, odpolední a celodenní. Rodiče mohou vytvářet kombinace docházky a dny docházení podle své potřeby, dítě tedy může docházet např. pouze dvakrát týdně.

Celkem ekoškolka nabízí tři jídla děnně - dopolední svačinu, oběd a odpolední svačinu. Na základě docházky (ze smlouvy) jsou dětem automaticky přihlášena jídla. Rodič má poté možnost si jídlo (den předem, do 11:00) odhlásit. Pokud ho nestihne včas odhlásit, bude za něj muset zaplatit a po dohodě s provozní je možné si jídlo odnést domů.

Docházka dětí je definována ve smlouvě, ale je nutné vést i reálnou docházku, evidenci, kdy dítě opravdu bylo ve školce. Učitelé tento požadavek plní tím, že mají předpřipravené docházkové archy do kterých ručně vyplňují docházku a poté je archivují v papírové podobě.

Ke každému dítěti jsou potřeba celkem dva základní dokumenty:

- Smlouva s rodičem
- Evidenční list dítěte

Dále se potom s rodiči uzavírají dodatky ke smlouvě, ve kterých se mění docházka dítětě (nebo prodlužuje).

1.2 Struktura práce

Druhá kapitola obsahuje popis problému. Je zde rešerše existujících řešení a seznam funkčních a nefunkčních požadavků které vyplynuly z konzultací s Organizací.

Třetí kapitola obsahuje analýzu systému. V této sekci jsou uvedeny případy užití a jejich diagramy. Druhou část tvoří doménový model systému.

Čtvrtá kapitola obsahuje návrh řešení. Zde je uvedena diskuze o použitém frameworku/-platformě pro vytvoření aplikace. Jsou zde také uvedeny sekvenční diagramy. Jako poslední část jsou zde diskutovány různé technologie pro vytvoření systému.

Pátá kapitola se zabývá popisem implementace celého systému a popisu částí ze kterých se systém sestává. Jsou zde také uvedeny problémy a zajímavosti, které se vyskytly v průběhu implementace.

Šestá kapitola se zabývá testováním systému. Jsou zde uvedeny postupy a výsledky unit testů, intergračního a zátěžového testování a testování uživatelského rozhraní s uživateli Organizace.

Sedmá kapitola popisuje budoucnost systému, kam by se měl vyvíjet.

Poslední, osmá, kapitola obsahuje zhodnocení a závěr celé práce.

Kapitola 2

Popis problému, specifikace cíle

2.1 Existující řešení

Existuje mnoho různých informačních systémů pro školy, ale žádný z nich není přímo orientovaný na školky. Nejznámnější jsou asi čtyři systémy, které jsou uvedeny v následujících podkapitolách.

2.1.1 Škola online

Jedná se o kompletní systém pro správu informací o škole a jejich žácích [45]. Umožňuje vést jejich evidenci, docházku a hodnocení. Dále nabízí možnost propojení se stravovacím systémem nebo elektronickou třídní knihou. Tyto části ovšem musí být zajištěny externím systémem.

U každého žáka je vedena elektronická žákovská knížka s jeho hodnocením a absencemi, poznámkami a případnými pochvalami. Systém nabízí také možnost správy školní družiny či klubu.

Také jsou zde obsaženy všechny rozvrhy a suplování učitelů. Nabízí také možnosti vytváření rozvrhů a kontroly kolize předmětů nebo učitelů v rozvrhu. Podporuje vytváření výkazů, informace o přijímacích řízeních nebo o zápisu, tisk vysvědčení, evidenci úrazů apod.

Po zakoupení je tato aplikace hostována na serverech poskytovatele a škole je přidělen administrátorský přístup, pomocí kterého se poté provádí správa (včetně vytváření účtů zaměstnancům a rodičům).

2.1.2 Bakaláři

Bakaláři[17] jsou informační systém, který je velmi podobný systému Škola online. Nabízejí také komponenty jako jsou:

- Evidence žáků a zaměstnanců
- Klasifikace
- Třídní kniha
- Tematické plány

- Knihovna
- Rozvrh hodin
- Rozpis maturit apod.

Zde se jedná o aplikaci, která je nasazena on-premise, tedy přímo ve škole. Aplikace tedy musí být nasazena na serveru ve škole a webová aplikace (která je součástí) slouží pouze pro předávání informací mezi školou a rodiči. Všechny softwarové komponenty ale musí běžet na strojích dodaných školou. Toto řešení vyžaduje také instalaci klientů na všechny počítače ve škole a jejich následnou údržbu a aktualizaci.

2.1.3 iŠkola

iŠkola[27] umožňuje škole vést elektronickou agendu a využívat moderní informační technologie. Rozsah iŠkoly sahá od malých vesnických škol až pod velké školské komplexy. Jedná se o webový nástroj, který nabízí pracovní prostředí pro všechny pracovníky školy a rodiče.

Kromě evidence žáků a rodičů nabízí systém mnoho modulů, které lze dle potřeby vypínat nebo zapínat. Je zde nástroj na zápise známek do elektronické žákovské knížky, prohlížení známek rodiči nebo oznámení nové známky rodiči na mobilní telefon. Další modul umožňuje tisk vysvědčení a dalších školních tiskopisů. Dále je možné vést elektronickou třídní knihu, tvořit rozvrhy hodin a suplování, exportovat data pro Ministerstvo školství, vytvářet vývěsky, e-learningové kurzy, zadávat domácí úkoly apod.

Jedná se o systém, který je hostovaný na serverech poskytovatele a uživatelé k němu přistupují pomocí webového prohlížeče. Aplikace je tak dostupná odkudkoliv s přípojením na internet.

2.1.4 edookit

edookit[21] je informační systém cíleny nejen na základní školy, ale i na mateřské školy, jazykové školy nebo i pro firemní e-learning.

Tento systém, obdobně jako ostatní, nabízí základní funkcionalitu evidence žáků a jejich hodnocení a docházky, zadávání domácích úkolů, komunikaci s rodiči apod. Narozdíl od ostatních systémů tento nabízí i evidenci plateb, které se ale musí do systému zadávat ručně (systém neumí automatické stahování plateb z banky). Systém nabízí také pro neformální komunikaci mezi školou a rodiči jakousi vlastní sociální síť.

Jde opět o aplikaci, která beží na serverech u poskytovatele a je přístupná přes webový prohlížeč, případně přes aplikaci pro Android OS. Uživatelské rozhraní je přizpůsobené pro zobrazení na mobilních zařízení a pro ovládání dotykem a gesty.

2.1.5 Shrnutí existujících řešení

V tabulce tab. 2.1.5 jsou zobrazeny základní vlastnosti tří výše uvedených systémů. Dále jsou v tabulce uvedeny vlastnosti, které jsou od nového systému očekávány a jejich pokrytí existujícími systémy.

Bohužel technologické detaily jednotlivých implementací nejsou veřejně dostupné. Většina webových produktů používá jako webový server Apache [15] a jako databázový server PostgreSQL[40]. Jakékoliv další detaily provozovatelé z důvodu bezpečnosti utajují.

Tabulka 2.1: Porovnání systémů

	ŠkolaOnline	Bakaláři	iŠkola	edookit
Místo nasazení	u poskytovatele	on-premise	u poskytovatele	u poskytovatele
Cena za rok v kč	4600	5700 + 1140	1200	1200 + 600
Evidence	ANO	ANO	ANO	ANO
Objednávky jídel	NE	NE	NE	NE
Pokladna/platby	NE	NE	NE	ANO
Docházka dětí	ANO	ANO	ANO	ANO
Projekty	NE	NE	NE	NE
Pobočky	ANO	NE	NE	NE
Kontakty	částečně	částečně	částečně	částečně
Docházka zam.	NE	NE	NE	NE

Komerčně dostupné systémy se orientují především na práci s dětmi a rodiči, na komunikaci mezi školou a rodiči, předávání hodnocení a informací o dětech apod. Tyto systémy však neumožňují mnoho funkcionality, kterou Organizace vyžaduje, jako je například evidence zaměstnanců a jejich docházky, evidence projektů a pokladny, objednávky jídel rodiči (bez využití externího systému), stahování a přiřazování plateb z banky, centrální databázi kontaktů apod.

Jako částečné řešení by mohlo být považováno využití jednoho systému na správu organizačních věcí (zaměstnanci, docházky, úvazky, evidence...), druhého systému na správu dětí ve školce (evidence, docházka, platby) a třetího systému na správu obědů. Tato situace by však byla pro tak malou Organizaci nanejvýš zbytečná a finančně velice náročná. Nevýhodou by také byla soustředěnost dat do více různých systémů a s tím spojená jejich údržba.

Z výše uvedené tabulky a dostupných materiálů vyplývá, že žádný z těchto systémů plně nevyhovuje požadavkům Organizace na informační systém.

2.2 Požadavky na systém

V této sekci jsou uvedeny požadavky, které systém musí splňovat. Jedná se o požadavky funkční, které definují chování systému, tak i o požadavky nefunkční, které definují platformu, na které systém bude nasazen. Pro přehlednost jsou funkční požadavky rozděleny na celky, kterých se konkrétně týkají.

2.2.1 Zaměstnanci

Funkční požadavky v sekci zaměstnanců zahrnují především úkoly týkající se správy uživatelů a jejich vlastností. Administráto má možnosti zaměstnance vytvářet, archivovat a editovat. U každého zaměstnance je také možnost vést evidenci dokumentů, které se daného zaměstnance týkají. Dále je také možnost přidávat úkoly zaměstnancům.

Dále zde také jednotliví zaměstnanci mají možnost vykazovat svou práci.

Kompletní seznam požadavků je uveden v C.2.

2.2.2 Projekty

Tato sekce zahrnuje činnosti, které smí administrátor provádět s projekty. Kromě vytváření, editace a odstraňování projektů smí administrátor také přidávat poznámky, dokumenty

nebo události s termínem splnění. Kromě těchto vlastností lze také přidávat k projektům indikátory splnění a k jednotlivým indikátorům přidávat poznámky.

Detailní seznam požadavků je uveden v C.3.

2.2.3 Provozní záležitosti

V této sekci jsou zařazeny požadavky, které se nehodí do žádné jiné, ale jsou nezbytné aby je systém podporoval. Spadají sem požadavky na správu databáze dokumentů, kontaktů, evidenci majetku, pokladní deník nebo kalendář důležitých událostí.

Kompletní seznam požadavků je uveden v příloze C.4.

2.2.4 Ekoškolka

Sekce ekoškolka obsahuje vše co se týká správy obou poboček ekoškolky. Je zde možné přidávat děti a jejich rodiče, editovat je, spravovat jídla v ekoškolce, zobrazovat a editovat platby od rodičů, zobrazovat dluhy rodičů. Ke každému dítěti je také možno přiřadit smluvní dokumenty, zobrazovat statistiky dětí nebo dítě archivovat. Ke každému dítěti je také přiřazena docházka.

Kompletní seznam požadavků na tuto sekci je uveden v příloze C.5.

Kapitola 3

Analýza

3.1 Případy užití

Podle požadavků od zadavatele byly vytvořeny modely případů užití. Pro přehlednost byly rozděleny do několika částí (stejně jako funkční požadavky), které jsou uvedeny v následujících podkapitolách. Zároveň je u každé podkapitoly uvedeno i mapování mezi funkčními požadavky a případy use-case.

Vzhledem k rozsáhlosti diagramů případů užití byly tyto umístěny do přílohy D a jsou odkazovány z textu.

3.1.1 Uživatelé systému

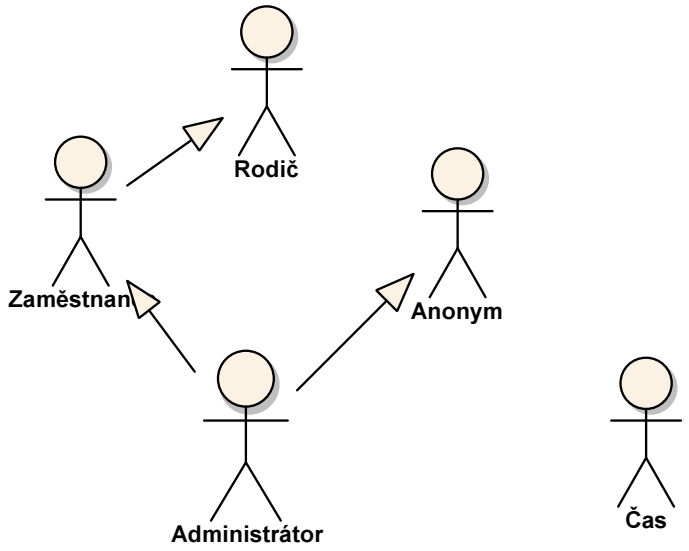
V systému bude vystupovat celkem 5 typů uživatelů:

- Anonym - nepřihlášený uživatel, který bude mít přístup pouze do veřejné sekce (přihlašování na semináře).
- Rodič - uživatel s přiděleným jménem a heslem, bude mít přístup do sekce pro rodiče.
- Zaměstnanec - uživatel s přiděleným jménem a heslem, bude mít přístup do části systému.
- Administrátor - uživatel s přiděleným jménem a heslem, bude mít kompletní přístup do všech částí systému.
- Systém - speciální uživatel, použit pro zobrazení případů užití, které vykonává systém automaticky.

Vztahy mezi těmito uživateli jsou znázorněny na obr. 3.1.

3.1.2 Zaměstnanci

V této sekci jsou zobrazeny akce, které může provádět Administrátor s uživateli jako např. vytvářet nebo upravovat. Dále jsou zde zachyceny úkony, které může nad svým uživatelským účtem vykonávat sám přihlášený uživatel.



Obrázek 3.1: Uživatelé systému

Diagram případů užití jsou uvedeny na obr. D.2 a obr. D.1. Tabulka tab. E.1 uvádí mapování mezi případy užití a funkčními požadavky. Je z ní vidět, že každý požadavek je realizován alespoň jedním případem use-case a naopak, každý případ use-case náleží k nějakému požadavku.

3.1.3 Projekty

Tato část popisuje akce, které je možné provádět v sekci Projektů. Administrátor zde může založit projekt a vést k němu evidenci dokumentů jako jsou například smlouvy, dodatky apod. Dále je možné Psát k projektu tzv. indikátory což jsou určité parametry, které musí být splněny, aby byl projekt uznán a proplacen dotační agenturou (občas jsou velice komplikované).

Důležitá je také možnost vedení kalendáře s upomínkama k projektu (datum odeslání monitorovací zprávy, datum vyúčtování apod.) a zasílání upomínek e-mailem zodpovědné osobě. K projektu bude také možné přidávat poznámky. U každého projektu je také evidence kontaktů (projektový manažer, zodpovědná osoba apod).

Diagram use-case pro Projekty je uveden na obr. D.3. Tabulka tab. E.2 uvádí mapování mezi případy use-case a funkčními požadavky.

3.1.4 Provozní záležitosti

Do této sekce patří požadavky, které nemohou být v systému vynechány, ale zároveň pro ně nebylo vhodné vytvořit samostatnou sekci.

Spadá sem evidence majetku, která je pro organizaci velice důležitá a povinná. Dále se zde soustředí vznikající dokumenty, jako jsou zápisy z porad, tak aby byly všechny na jednom místě a snadno dostupné všem.

Dále zde bude možné vytvářet kontakty a seznamy kontaktů s následnou možností zobrazení informací nebo hromadné rozesílky e-mailů.

Pro finanční řízení zde také bude vedení pokladního deníku a aktuálního stavu pokladny.

Diagram use-case pro Provozní záležitosti je uveden na obr. D.4. Tabulka tab. E.3 uvádí mapování mezi případy use-case a funkčními požadavky.

3.1.5 Ekoškolka

Tato sekce se zabývá správou informaci a dat v sekci Ekoškolka. V této části vystupují celkem 4 aktéři.

Je zde zanesena správa dětí a jejich rodičů, docházka, objednávky obědů a správa plateb od rodičů. Děti se ze systému nebudou odstraňovat trvale, ale budou pouze přesunuty do archivu, aby byly informace o nich i nadále přístupné.

Objednávky jídel (dvě svačiny a oběd) budou vygenerovány na základě zadané docházky dítěte (ze smlouvy) a rodiče budou mít možnost si předem jednotlivá jídla odhlásit. Rodiče také budou vidět přehled těchto objednávek a cenu za jídla.

Systém si bude automaticky stahovat informace o platbách od rodičů z banky a přiřazovat je k dětem podle variabilního symbolu. Pokud platba nebude přesně odpovídat, systém informuje administrátora a ten ji bude moci ručně přiřadit a označit za vyřešenou.

Také bude možné hromadně rozesílat emaily rodičům, automaticky zasílat učitelům informaci o blížících se narozeninách dítěte apod.

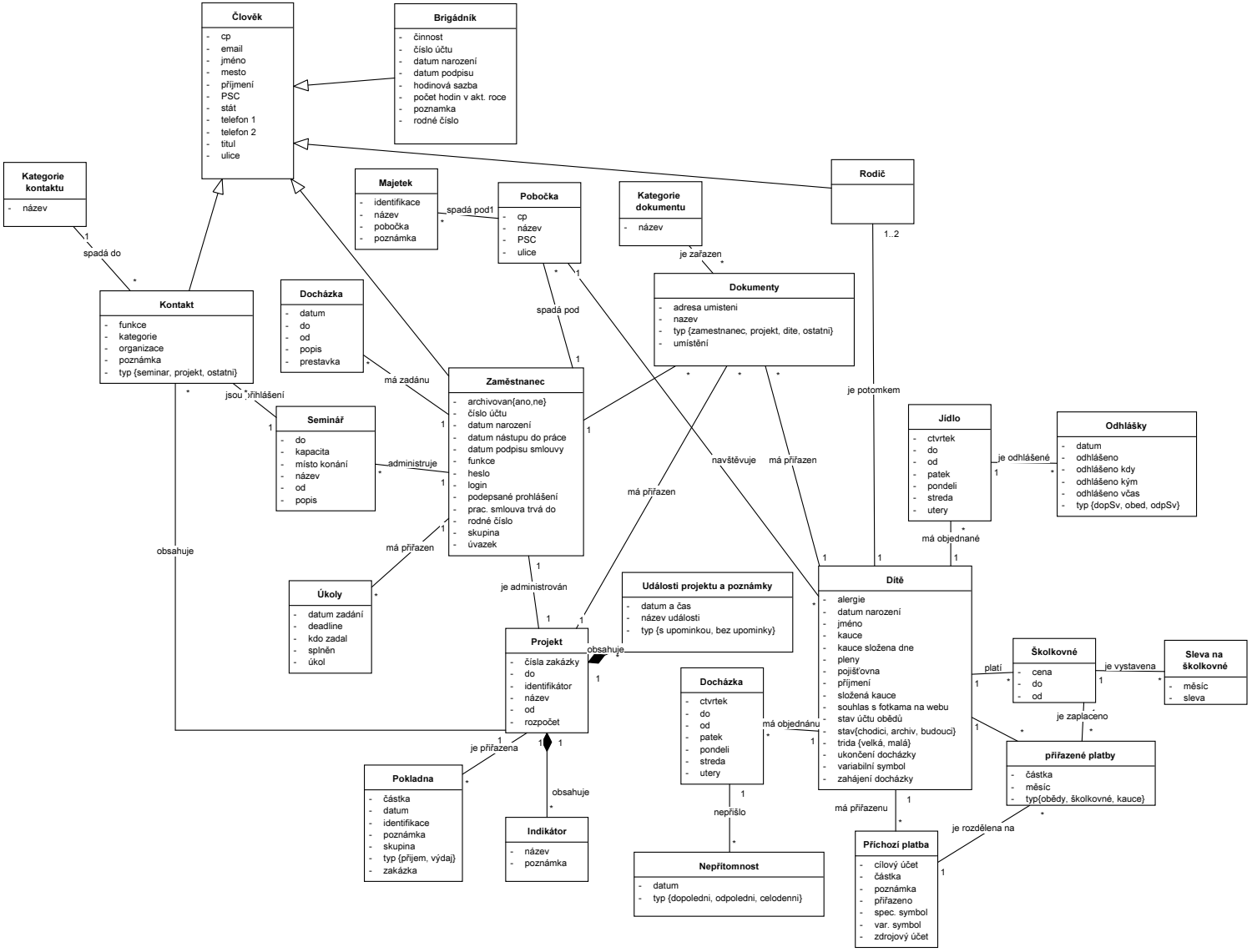
Jelikož se jedná o celkem rozsáhlou část, byla rozdělena na dvě části. Diagramy případů užití jsou uvedeny na obr. D.5, obr. D.6 a obr. D.7. Tab. E.4 a tab. E.5 uvádí mapování mezi případy use-case a funkčními požadavky.

3.2 Doménový model

Analýzou podstatných jmen a sloves z požadavků byl vytvořen doménový model aplikace[5], viz. obr 3.2.

Vztahy mezi entitami byly odvozeny také z požadavků (převážně ze sloves). Jednotlivé vlastnosti entit byly odvozeny jak z požadavků definovaných v sekci 2.2 tak i z konzultací s organizací.

Prvním celkem jsou entity týkající se Projektů, Zaměstnanců a jejich vlastností. Dalším celkem je Ekoškolka se správou dětí a obědů. Další celky jsou již menší jako je evidence dokumentů, majetku a událostí.



Obrázek 3.2: Doménový model aplikace

Kapitola 4

Návrh řešení

4.1 Framework

V současné době jsou pro implementaci a návrh aplikací v enterprise sféře používané zejména tři platformy/frameworky. Konkrétně se jedná o Java EE 7, Spring a Play. V následujích podkapitolách je krátce popíši a uvedu co bude použito pro tuto práci.

4.1.1 Java EE 7

Java Enterprise Edition ve verzi 7 [10] je poslední verze tohoto vysokoúrovňového objektově-orientované jazyka [10], vydávaného společností Oracle. Enterprise Edition staví na základech z Java SE(standard edition), kterou rozšiřuje o možnosti enterprise jazyka jako je bezpečnost, vrstevnatá architektura nebo webové služby.

Všechny aplikace v Java EE 7 by měly být nasazeny v Java EE 7 certifikovaném kontejneru. Tato verze Javy obsahuje mnohé specifikace, které usnadňují tvorbu a portabilitu aplikací - jako příklad můžeme uvést podporu RMI(Remote Method Invocation), JMS (Java Messaging Service) nebo Web services. Při dodržení specifikací je zaručena kompatibilita aplikace se všemi certifikovanými servery a kontejnery. Tyto aplikační servery se při běhu automaticky mohou starat např. o bezpečnost, transakce, rozšiřitelnost nebo paralelizaci.

Java EE 7 nabízí kromě plné verze vývojového kitu (JDK) i se zmenšenou verzí (web profile), který nabízí pouze funkcionalitu potřebnou pro vytváření webových aplikací[6]. Obsahuje tedy pouze podmnožinu funkcionalit plné verze. Tato skutečnost nabízí možnost zmenšení běžící instance aplikačního serveru. Tento profil obsahuje technologie jako jsou např. JSF 2.2, EL 3.0, Servlet 3.0, JPA 2.1 nebo Dependency injection (DI) [6].

Java EE 7 tak přichází s kompletní out-of-the-box možností vytváření webové aplikace bez nutnosti používání dalších externích technologií (i když to nevylučuje).

4.1.2 Spring

Spring[46] je open-source projekt, který staví na základech Java EE 7. Jedná se o rodinu projektů, které mají za cíl usnadnit vývojářum práci a vykonat za ně většinu rutinní práce.

Tento projekt není specifický pouze pro webové aplikace, ale lze samozřejmě využít i pro aplikace desktopové nebo serverové. Základním projektem z této rodiny je Spring framework [47].

Spring framework nabízí svou funkcionalitu v jednotlivých modulech. Několik základních modulů:

- Autentizace a autorizace
- AOS - Aspektově orientované programování
- Přístup k datům - správa a práce s relačními a NoSQL databázemi
- Inversion of Control (IoC) kontejner - stará se o automatickou konfiguraci a použití objektů s využitím Dependency injection
- MVC - podpora pro model-view-controller architekturu[7]
- Správa transakcí
- Testování

Základem celého Spring frameworku je právě IoC kontejner, který zajišťuje správu a konfiguraci jednotlivých objektů. K tomu využívá převážně postupy Dependency injection a reflexe. Kontejner zajišťuje a spravuje celý životní cyklus jednotlivých objektů, takže programátor toto vůbec nemusí řešit.

Spring framework sám o sobě neobsahuje žádnou přístupovou metodu k datům v databázi, ale nabízí podporu pro všechny standardně používáné frameworky, mezi něž patří např. hibernate, TopLink nebo JPA.

4.1.3 Play

Play[37] je framework, který je narozdíl od Spring napsaný převážně ve Scale(jádro) a také v Javě. Umožňuje programátorům využít k programování oba tyto jazyky. Při použití Scaly je navíc dostupná možnost využít existující Java knihovny. Scala má také výhodu v kvalitním paralelním zpracování, které má význam ve víceprocesorovém (vícejádrovém) prostředí.

Play přichází s instalačním prostředím TypeSafe Activator. Po stažení této platformy jsou automaticky nainstalovány tyto komponenty:

- Play framework
- Akka
- Scala
- Activator

Play framework je asynchronní a neblokující framework, který podporuje vývoj jak v Javě, tak i ve Scale. Standardně nabízí plnou podporu pro RESTful aplikace, čímž je značně usnadněna škálovatelnost projektu. Nabízí také silnou podporu pro paralelní zpracování informací.

Akka je systém pro podporu konkurenční běh a distribuci na Java Virtual Machine (JVM). K této činnosti používá aktory (actor system).

Scala je programovací jazyk, který je jak funkcionální, tak i objektový. Dokáže využít v projektu existující Java knihovny, není tedy potřeba je speciálně kompilovat pro Scalu. Scala se kompiluje do Java bytekódu, takže je spustitelná v JVM. Má podobnou syntaxi jako jazyk C, používá statický typový systém

Activator je platforma, kterou play používá nejen pro instalaci nutných závislostí, ale i ke kontrole jednotlivých projektů. Umožňuje sestavení, spouštění, debug jednotlivých projektů napsaných v Play. Obsahuje také embedde Netty server, na kterém se standardně programy spouštějí (popř. se dá vygenerovat WAR a nasadit na jiný aplikační server).

4.1.4 Shrnutí

Po vyzkoušení jednotlivých možností implementace a po diskuzi s vedoucím práce byla vybrána k implementaci platforma JAVA EE 7.

Tato platforma nabízí veškeré potřebné nástroje a prostředky k implementaci žádaného systému. Navíc tím, že se jedná o standard je zaručena kompatibilita s jakýmkoliv certifikovaným běhovým prostředím této platformy (různé aplikační servery).

4.2 Sekvenční diagramy

Sekvenční diagramy se používají po celou dobu vývoje aplikace a slouží k demonstraci interních interakcí mezi účastníky systému a objekty v systému[8]. Tyto diagramy se používají k popisu např. use case, subsystémů nebo protokolů. Já zde použiji sekvenční diagramy k popisu několika zajímavých use case.

4.2.1 Přihlášení se do systému

Tento diagram popisuje činnost, kterou systém vykoná v případě pokusu o přihlášení uživatele do systému. Zadané údaje jsou předány ze stránky s přihlašovacím formulářem do backend třídy (controller), který následně vytvoří objekt uživatele. Tento objekt je předán business třídě, která pomocí DAO třídy ověří uživatele v databázi.

Diagram je uveden na obr. 4.1.

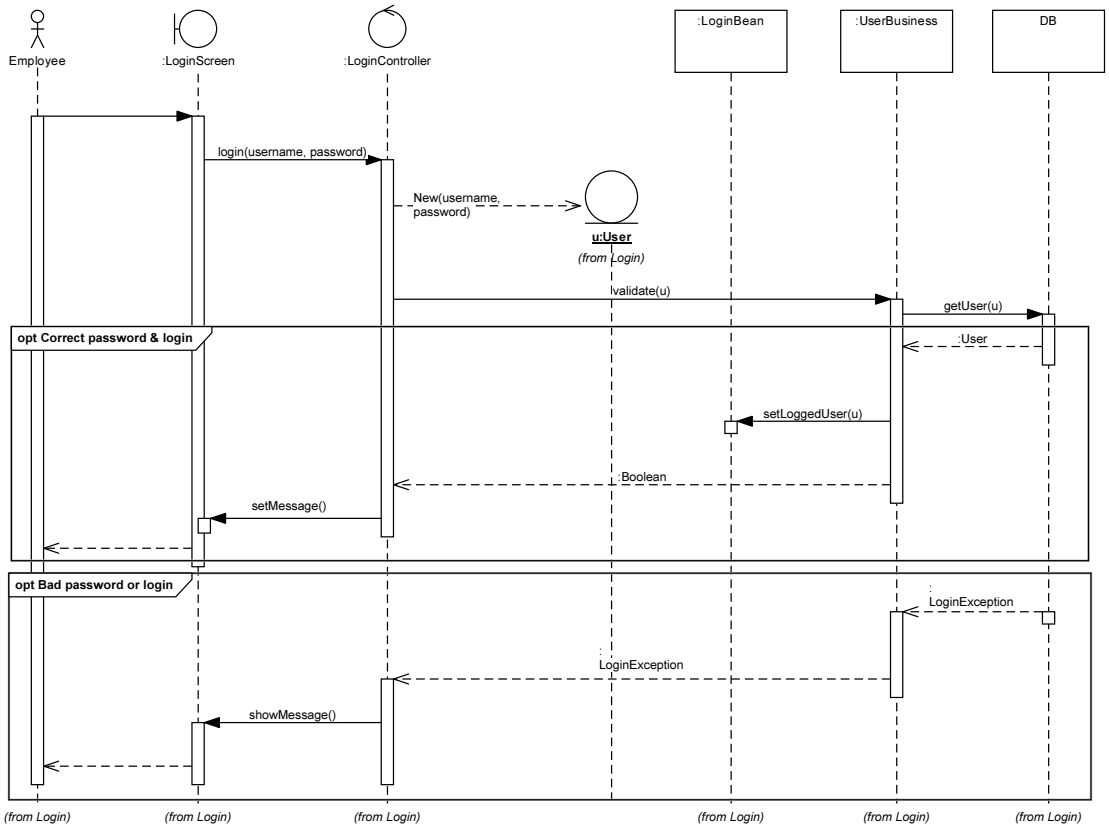
4.2.2 Přidání úkolu zaměstnanci

Diagram na obr. F.1 zachycuje přidání úkolu konkrétnímu zaměstnanci. Systém nejdříve nechá zadavatele vybrat konkrétního zaměstnance, kterému bude úkol zadán. Uživatel poté vytvoří samotný úkol - zadá text úkolu a datum do kdy má být úkol dokončen. Systém po odeslání informací vytvoří objekt úkolu a uloží ho do databáze.

4.2.3 Přidat dítě

Diagram na obr. F.2 ukazuje posloupnost akcí, které probíhají při přidání dítěte.

Každé dítě musí mít alespoň jednoho rodiče. Zde je možnost, že se přidává dítě k již existujícím rodičům nebo se rodiče v rámci tohoto scénáře vytvoří. Poté se již zadají údaje o dítěti a systém ho uloží.



Obrázek 4.1: SD login

4.2.4 Poslat upomínku zaměstnanci

Tento diagram zachycuje akci, kdy systém automaticky kontroluje všechny úkoly pro zaměstnance a posílá jim e-mail s blížícími se datumy splnění.

Diagram je zachycen na obr. F.3.

4.2.5 Manuálně přiřadit platbu

Tento diagram se použije v případě, kdy systém zjistí příchozí platbu, která sice variabilním symbolem odpovídá nějakému dítěti v databázi, ale částka neodpovídá dluhu na obědy ani školkovnému. Uživatel má poté možnost částku rozdělit na školkovné na jednotlivé měsíce nebo uložit na účet obědů. Diagram je na obr. F.4.

4.3 Technologie

V této sekci v krátkosti popíši používané technologie při vytváření systému.

4.3.1 Aplikační server

Jelikož práce bude psána na platformé Java EE 7 bude použit i aplikační server, který je pro tuto platformu certifikovaný. V současné době existují pouze 2 aplikační servery, které

mají certifikaci pro Web profil Java EE 7[9].

Konkrétně se jedná o server GlassFish 4.0[23] a WildFly 8[50]. Z těchto dvou byl zvolen server WildFly verze 8, se kterým je autor lépe seznámen a ze zkušenosti se jedná o paměťove méně náročnou aplikaci než je server GlassFish.

Wildfly ve verzi 8 je open-source aplikační server, který je vyvíjen společností Red Hat[42]. Samotný balík serveru obsahuje mimo samotného aplikačního serveru i jiné komponenty jako např. Hibernate (ORM framework), Weld (webserver), RESTEasy (REST provider) a mnoho dalších[13].

4.3.2 Databáze

Existuje dnes velké množství databází, které by byly použitelné pro tento projekt. Nejdříve byla vybrána databáze MySQL 5 [34] se kterou má autor bohaté zkušenosti. Na doporučení vedoucího práce byla však poté vybrána databáze PostgreSQL[40], která se bude na tuto práci hodit lépe.

4.3.3 ORM framework

Ze značného výběru ORM frameworků byl nakonec vybrán framework Hibernate [24]. Jedná se o velice pokročilý open-source framework, který je standardně distribuován s aplikačním serverem WildFly 8, čímž je zaručena jejich vzájemná kompatibilita. Hibernate je také plně kompatibilní s databází PostgreSQL.

4.3.4 View

Po diskusi s vedoucím práce a po prostudování používaných frameworků použitelných na vytvoření grafického rozhraní pro koncové uživatele byl vybrán framework AngularJS [14]. Jedná se o open-source framework o jehož vývoj se stará Google. Cílem tohoto frameworku je umožnit snadné vytváření převážně single-page aplikací. Základním principem je obohacování statického HTML speciálními tagy nebo atributy, které pocházejí z dílny AngularJS (nebo je možné vytvářet i vlastní), které jsou následně zkompilovány pomocí JavaScriptu a uživateli je vykreslen výstup.

Kapitola 5

Implementace

5.1 Úvod

V této sekci jsou popsány detaily implementace celého systému. Sekce je rozdělena do podsekcí podle jednotlivých částí systému - databáze, serverová část a klientská část.

5.2 Databáze

V databázi byly z návrhu vytvořeny všechny potřebné tabulky. Diagram je zachycen na obr. G.1. V databázi byly také vytvořeny vztahy mezi tabulkami aby byla zajištěna datová integrita i na této úrovni. Z této databázové vrstvy byly poté vytvořeny Entity v jazyce Java se kterými pracuje serverová aplikace. Všechny tabulky mají unikátní identifikátor ve formě sloupce 'id', který má nastaven datový typ 'serial', který zaručuje inkrementální nárust při každém přidání záznamu do tabulky. Tento identifikátor je také použit jako primární klíč v jednotlivých tabulkách a zároveň slouží jako cizí klíč v relacích s ostatními tabulkami.

5.3 Serverová část

Serverová část aplikace je implementována v jazyce Java EE 7 s využitím několika frameworků a pomocných knihovne. Dále následuje popis jednotlivých části této aplikace.

5.3.1 Hibernate

Hibernate[24] je zde použit jako ORM framework, který tvoří vrstvu mezi samotnou databází a logikou aplikace samotné. Tato část je obsažena v balíčku 'Entity', který obsahuje jednu entitu pro každou tabulku v databázi. Navíc je zde i rozhraní 'EntityInterface', které definuje co musí každá entita obsahovat - konkrétně se jedná o atribut 'id' a k němu setter a getter.

Balíček 'Entity' obsahuje ještě podbalíček 'Entity.enums' ve kterém jsou vloženy Enumy, které se používají v jednotlivých entitách. Například jde o výčet typů jídel nebo výčet typu kontaktu.

5.3.2 Jackson

Jackson[28] se nachází v procesu zpracování požadavku na druhé straně aplikace než Hibernate. Jedná o serializační knihovnu, která umožňuje převod mezi Java objekty (entitami) a textovým řetězcem, konkrétně se jedná o JSON [12].

Jackson sám o sobě není schopen serializovat objekty, které mají mezi sebou cirkulární reference (tj. objekt A odkazuje objekt B a ten opět odkazuje objekt A). Naštěstí modulární architektura Jacksonu umožňuje dodání vlastního serializeru. Tuto funkci vykonává serializer JSOG[31], který umožňuje serializaci i objektů s cirkulárnimi referencemi. Vyžaduje však použití této knihovny na straně klienta.

5.3.3 Joda time

Joda time[30] je projekt, který nahrazuje Java funkce pro práci s datem a časem. Umožňuje pracovat pouze s časem nebo datem, nabízí rozšířené a zjednodušené funkce pro práci s těmito objekty. Jelikož interně využívá milisekundy, stejně jako standardní funkce v JDK, není problém s interoperabilitou s klasickými funkcemi Javy.

5.3.4 PicketLink

PicketLink[35] je open-source projekt, který nabízí možnosti autorizace a autentizace uživatelů pro Java aplikace. Skládá se z mnoha modulů [36], které nabízí mnoho funkcionality. V tomto projektu je využita pouze část těchto modulů. Konkrétně se jedná o moduly umožňující základní práci s uživateli (CRUD operace), modul zajišťující generování tokenu, který je posílám jako autentizace klienta a modul umožňující zabezpečení jednotlivých Java tříd nebo funkcí pomocí anotací s definicí požadovaného přístupu.

5.3.5 Deltaspike

Apache Deltaspike[19] je kolekce CDI rozšíření, které je možno použít v Java aplikacích s podporou CDI. Projekt se opět skládá z mnoha modulů umožňujících rozšíření standardní knihovny funkcí jazyka Java EE.

Kromě toho, že v této práci použitý security framework PicketLink vyžaduje DeltaSpike jako nutnou závislost má zde také další využití. Konkrétně se jedná o využití modulu Scheduler[20], který nabízí možnost plánování spouštění úloh pomocí implementace rozhraní 'Job' a anotací '@Scheduled', která nabízí možnost definice času spouštění úlohy pomocí syntaxe velmi podobné linuxovému CRONu. Ke své práci vyžaduje přítomnost knihovny, která umožňuje plánování úloh. V této práci se jedná o knihovnu Quartz[41]. Velkou výhodou použití knihovny Quartz ve spojení s modulem DeltaSpike Scheduler spočívá v tom, že v takto definovaných třídách je možné využívat CDI. Pokud by byl použit pouze framework Quartz, bez DeltaSpike Scheduler modulu, CDI by nebylo funkční a výrazně by se tak zhoršila kvalita aplikace kvůli zvýšené vazbě mezi třídami.

5.3.6 Resteasy

Resteasy[43] je JAX-RS provider pomocí kterého lze definovat a kontrolovat REST zdroje v Java EE aplikaci. Stará se o automatickou delegaci serializace a deserializace, vytváření REST endpointů, definici metod zdrojů (POST,GET, PUT, DELETE apod).

5.3.7 Balíčky

V této sekci jsou krátce popsány balíčky, které obsahuje serverová část a jejich funkce.

- dao - obsahuje třídy DAO, které umožňují přístup k databázi (CRUD operace s entitama)
- entity - obsahuje třídy, které definují doménový model aplikace
- entity.bank - obsahuje entity, které se vážou na část aplikace, která automaticky stahuje platby z banky. Nemají svůj odpovídající obraz v databázi, ale jsou využívány jen pro parsování dat z banky a prezentaci uživateli
- entity.enums - obsahuje výčtové typy, které jsou používané v entitách
- helper - balíček obsahuje pomocné třídy, které jsou používány především v servisní vrstvě(komparátory, nastavení, práce s datem a časem apod.)
- security - obsahuje balíčky vztahující se k bezpečnosti aplikace (definice tokenu, security manager apod.)
- service - obsahuje servisní třídy, které vykonávají vlastní business logiku aplikace
- service.scheduler - obsahuje třídy, které jsou spouštěny automaticky v nastavený čas (stahování plateb z banky apod.)
- resource - obsahuje REST endpointy
- test - obsahuje třídy s JUnit testy

5.3.8 Implementační problémy a zajímavosti

5.3.8.1 Zabezpečení

Zabezpečení serverové části je řešeno pomocí frameworku PicketLink[35], který ve spolupráci s Apache DeltaSpike[19] nabízí možnosti deklarativního zabezpečení funkcí a celých tříd. Zároveň nabízí i možnosti autentizace uživatelů a jejich správu.

Pro tuto práci důležitou části tohoto bezpečnostního frameworku je možnost autentizace REST klientů. Tato autentizace funguje tím způsobem, že po úspěšné autorizaci uživatele jménem a heslem je tomuto uživateli vygenerován a zaslán token. Tento token je následně uložen lokálně u uživatele na klientovi a musí být obsažen v každém požadavku, který klient na server odešle. Token je uložen v hlavičce každého HTTP požadavku, jak je ukázáno na obr. 5.1.

Server si pomocí tohoto tokenu(který má nastavenu určitou platnost) získá uživatele kterému token náleží a podle něj dále rozhoduje jestli uživatel bude vpuštěn do sekce do které se snaží dostat.

Autentizace na úrovni serveru je tvořena anotacemi, které jsou vloženy k jednotlivým funkcím nebo na celou třídu (tím jsou aplikovány na všechny funkce ve třídě). Příklad aplikace takové anotace je na výpisu. 1. Tyto dvě anotace konkrétně zajišťují, že přístup bude povolen pouze zalogovanému uživateli, který má nastavenu jednu z rolí ADMINISTRATOR,

Obrázek 5.1: Token uložený v hlavičce HTTP požadavku

EMPLOYEE nebo TEACHER. Kromě RBAC (Role-based access control), který přiděluje práva na základě přiřazené role k uživateli je ještě možné využít povolování přístupu dle přiřazené skupiny nebo partition.

Interně funguje toto zabezpečení tak, že díky anotaci se vytvoří Proxy [3], která odchytí příchozí požadavek na metodu třídy a ten je ověřen - podle výsledku tohoto ověření je poté povolen vstup do metody nebo zakázán a je vytvořena vyjímka.

Výpis 1: Aplikace anotace na REST resource

```
@RequestScoped
@Path("/private/children")
@LoggedIn
@RolesAllowed({"EMPLOYEE","TEACHER"})
public class ChildResource extends GenericResource...
```

5.3.8.2 Periodické spouštění úloh

Na serveru existuje několik úloh, které je nutno pouštět pravidelně (např. stahování plateb z banky, archivace dětí apod.). K tomuto účelu je využita funkcionalita z frameworku Apache DeltaSpike[19], konkrétně jeho část Scheduler[20]. Tento modul spolupracuje s plánovacím frameworkem Quartz 2[41]. Pomocí implementace rozhraní Job (z balíčku org.quartz), které předepisuje metodu 'execute', je možné definovat úkoly, které budou spouštěny periodicky. Kromě periodického spouštění nabízí framework Quartz i další možnosti spouštění jako je jednorázové spuštění, opakované spuštění v zadaném intervalu nebo je také možnost definovat počet opakovaných spuštění.

Ukázka anotace třídy, která se stará o stahování plateb z banky a o jejich přiřazení dětem, je na obr.2. Konkrétně se tato úloha spouští každý den ve 3:00 ráno.

Výpis 2: Anotace umožňující naplánované spouštění úlohy

```
@RequestScoped
@Scheduled(cronExpression = "0 0 3 1/1 * ? *")

public class DownloadAccountStatements implements Job {
...
}
```

5.3.8.3 Vytvoření entit, DAO a servisních tříd

Na počátku projektu byla vytvořena nejdříve databáze se všemi potřebnými omezeními (relace, složené klíče, constrainty apod.) a teprve poté bylo přikročeno k vytvoření entitních tříd. Pro ulehčení jinak rutinní práce bylo využitova nástroje z Hibernate Tools, který umožňuje pomocí reverzního inženýrství vytvořit entitní třídy automaticky. Tento nástroj dokáže nejen vytvořit jednotlivé entity s jejich atributy, ale dokáže i přidat základní relace a omezení mezi entitami. Pokud je potřeba využít složitější konstrukce (jako je M:N relace nebo složitější hierarchické struktury) je nutné do kódu již zasáhnout ručně. Celkově tento nástroj ulehčí a urychlí vývoj aplikace a zabrání vzniku chyb, které by mohly vzniknout překlepem.

Při vytváření DAO(také nazývány Table Data Gateway[7]) tříd (nejnižší vrstva aplikace, která komunikuje s databází) byla využita možnost generického programování. Byla vytvořena generická třída *GenericDAO<T, I extends Serializable>*, která je předkem všech DAO tříd v projektu. Tato třída je parametrizovatelná Objektem, který bude představovat (např. dítě nebo zaměstnanec) a typem identifikátoru objektu (např. Integer). V této třídě jsou vytvořeny základní funkce, které jsou potřeba v každé DAO třídě - vytvoření, změna, odstranění a získání záznamu. Dále je zde také funkce na získání Hibernate Session, která je poté používána na vytvoření vyhledávacích kritérií. Poslední zajímavou funkcí v této třídě je funkce *List<T> searchByExample(T ex, Class<T> cls)*, která umožňuje vyhledávat v databázi podle příkladného objektu - tj. pokud bude potřeba vyhledat všechny děti, které jsou archivovány, stačí pouze vytvořit prázdný objekt Dítě, nastavit mu vlastnost 'archivováno' a Hibernate udělá vše ostatní potřebné.

Generické programování bylo použito také pro vytvoření servisní vrstvy, která navíc využívá služeb generické vrstvy DAO. Tímto přístupem se výrazně zjednodušil návrh celé aplikace, její správá a zvýšila se rychlost vývinu. Samozřejmě pokud generické metody nevyhovovaly, byly přepsány jinou implementací v konkrétní třídě.

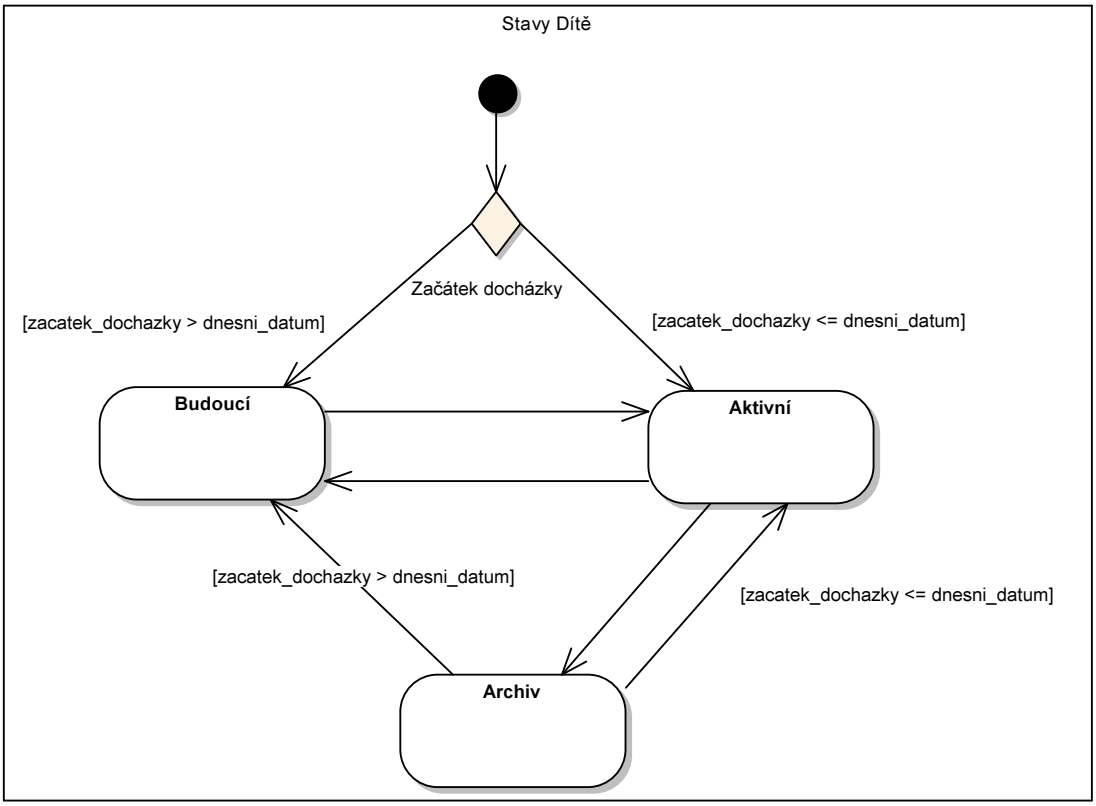
5.3.9 Stavové diagramy

5.3.9.1 Dítě

Stavový diagram pro objekt dítěte je zobrazen na obr.5.2. Po založení dítěte v aplikace se může dostat do dvou stavů, v závislosti na začátku docházky:

- Budoucí - začátek docházky dítěte je v budoucnosti oproti aktuálnímu dni
- Aktuální - začátek docházky je v aktuální den nebo v minulosti

Mezi těmito stavy se může volně přecházet v závislosti na nastavení docházky dítěte. Ze stavu aktivní může dítě přejit do stavu Archiv, který indikuje to, že dítě do školky již nedochází. Pokud si to rodiče rozmyslí a dítě opět do školky přihlásí, dítě přejde do stavu Aktivní nebo Budoucí podle nastavení nové docházky.



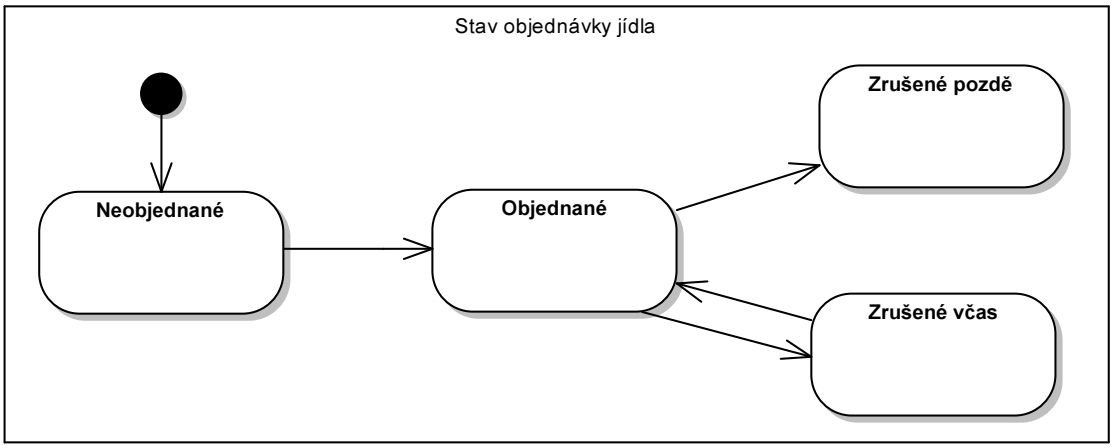
Obrázek 5.2: Stavový digram Dítě

5.3.9.2 Jídlo

Jídlo se může nacházet ve stavech, které jsou zobrazené na obr.5.3. Diagram se vyathuje k jednomu konkrétnímu jídlu, ať už se jedná o dopolední svačinku, oběd nebo odpolední svačinku. Ve výchozím stavu jídlo objednané není. Poté může přejít do stavu Objednáno, tj. s jídlem se počítá a pokud nebude včas odhlášeno bude objednáno i u dodavatele. Rodič (nebo zaměstnanec) může jídlo odhlásit. Při odhlášení jsou dvě možnosti:

- Odhlášeno včas - rodič jídlo nemusí platit, u dodavatele nebude objednáno
- Odhlášeno pozdě - jídlo bylo u dodavatele již objednáno a rodič ho musí zaplatit (ale může si ho vyzvednout ve školce)

Včas zrušené jídlo je možné ještě doobjednat, ale pouze v určitém čase (před realizací objednávky u dodavatele).



Obrázek 5.3: Stavový digram Jídlo

5.4 Uživatelské rozhraní

Jak bylo napsáno již dříve, uživatelské rozhraní bylo vytvořeno pomocí HTML a MVC[7] frameworku AngularJS.

Pro snadnější stylování webového rozhraní byl použit framework Bootstrap[18]. Jedná se mobile-first CSS a JS framework, který byl původně vyvinut firmou Twitter. V projektu byl navíc použit i projekt UI Bootstrap[48], který nabízí UI komponenty, které jsou založené na Bootstrapu, ale jsou psané v AngularJS. Obsahuje komponenty jako je například dialog, alert(barevně zvýrazněné upozornění) nebo datepicker(pop-up kalendář).

Uživatelské rozhraní je rozdělené na dvě části. První je určena pro zaměstnance Organizace, druhé je určeno pro rodiče dětí. Zaměstnanci mají přidělena práva podle jejich role a mohou tak administrovat data uložená v databázi. Rodiče dětí mají pouze právo na editaci jídel u svých dětí.

5.4.1 Adresářová struktura

Projekt byl rozdělen do několika složek pro lepší přehlednost (obdoba balíčků v Javě). Následuje krátký popis jednotlivých složek.

- css - obsahuje kaskádové styly použité v aplikaci
- js - složka obsahující následující podsložky a soubory:
  - app - obsahuje definici AngularJS aplikace, routing a HTTP security interceptor
  - calendar - obsahuje definici použité komponenty Kalendář
  - controllers - obsahuje kontroléry pro jednotlivé části aplikace
  - core - zde jsou umístěny soubory, které musí být přítomny vždy, tj. samotný AngularJS a jeho závislosti.

  - directives - obsahuje direktivy, které byly vytvořeny a použity při vytváření rozhraní.
  - filters - obsahuje filtry
  - services - obsahuje odkazy na REST zdroje a služby, které vykonávají samotnou business logiku aplikace
  - JSOG.js - knihovna potřebná pro zpracování serializovaných objektů zaslaných serverem obsahujících cirkulární závislosti

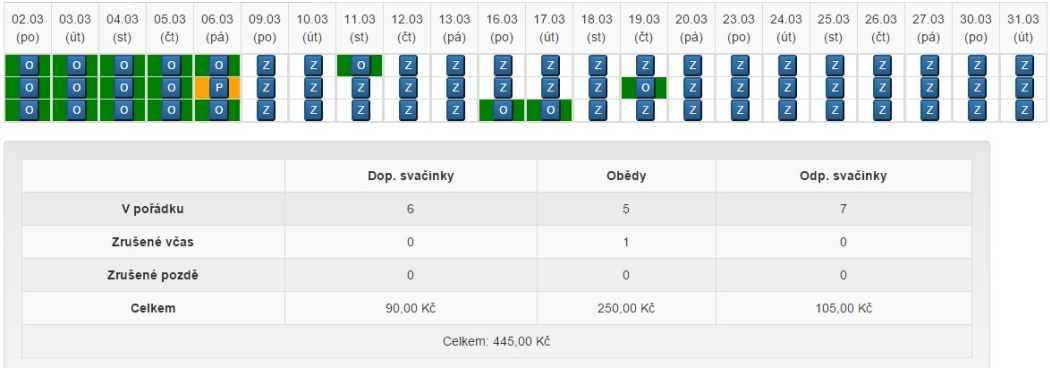
- partials - obsahuje HTML soubory, které netvoří celou stránku, ale jsou nahrávány do předpřipravené šablony.
- login.html - stránka s přihlašovacím formulářem do systému
- index.html - hlavní šablona aplikace do které jsou následně nahrávány soubory ze složky /partials

5.4.2 Implementační problémy a zajímavosti

5.4.2.1 Direktiva pro administraci jídel dětí

Administrace jídel dětí je jednou z klíčových částí systému. Nejen proto, že se díky této části objednávají reálné počty obědů pro děti, ale počítá se z ní i cena obědů, které rodiče následně platí. Rodiče mají buď možnost dát vědět pracovníkovi organizace nebo si mohou jídlo odhlásit sami prostřednictvím jejich jména a hesla.

Pro tuto práci byla vytvořena direktiva 'meals'. Umožňuje listování v měsících a grafické zobrazení objednaných jídel, jejich stav (objednáno, zrušeno včasm, zrušeno pozdě, neobjednáno) a změnu jejich stavu. Direktiva dále také zobrazuje počty a ceny objednaných jídel. Ukázka direktivy v prohlížeči je na obr. 5.4. Vytvoření stránky, která by nabízela stejnou funkcionalitu by vyžadovalo množstvá místa a kódu a navíc by nebyla možná znovupoužitelnost. Díky direktivám je možné celý tento kód přesunout mimo stránku a poté jen direktivu vložit do stránky jako tag, viz. obr.3.



Obrázek 5.4: Grafické zpracování direktivy meals

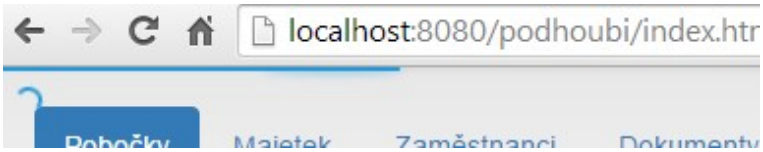
Výpis 3: HTML tag pro vložení direktivy

```
<meal child="selectedChild" admin="true"></meals>
```

5.4.2.2 Zobrazení stavu HTTP požadavku

Jelikož AngularJS pracuje s REST endpointy asynchronně je potřeba zobrazit stav a výsledek HTTP požadavku uživateli. Načítání informací ze serveru může také nějakou dobu trvat, je tedy vhodné zobrazit informaci o tom, že data se načítají a informovat ho po dokončení této operace.

Ideálním prostředkem na tuto práci je status bar, který zobrazuje stav načítání stránky. Ukázka tohoto statusu je na obr. 5.5. Jde o projekt psaný pro AngularJS, který po jeho importu automaticky odchytává (pomocí HTTP Interceptoru) všechny požadavky procházející skrz službu $http (která se využívá pro všechny HTTP požadavky) a graficky zobrazuje jejich stav pomocí animovaného ukazatele v horní části stránky. Uživatel je tak informován o tom, že probíhá nějaká akce.



Obrázek 5.5: Loading bar

5.4.2.3 Zabezpečení

Zabezpečení na straně klienta je velmi diskutabilní a rozhodně ho nelze používat samostatně. Vždy musí být použito ve spojení se server-side zabezpečením.

Zabezpečení na klientovi se sestává ze tří částí. První část definuje samotné ověření příchozího klienta, druhá část se zabývá zobrazením pouze těch částí, které může uživatel vidět a poslední část se zabývá předáním informací o přihlášeném uživateli na server (REST je stateless služba a neuchovává informace o aktuálně přihlášeném uživateli).

Ověření přihlášeného klienta se setává ze zadání jména a hesla, které má klient přidělené. Tyto údaje jsou předány na server (po nasazení komunikace bude šifrovaná), který údaje ověří a v případě úspěchu předá klientovi jedinečný Token, kterým se bude ověřovat.

Zobrazení pouze těch částí webu na které má uživatel právo je řešeno pomocí direktivy 'access'. Tato direktiva je omzena pouze na to aby mohla být použita jako atribut u jakéhokoliv tagu a obsahuje uživatelské skupiny, které mají k danému tagu mít přístup. Při vykreslování stránky je skupina aktuálního uživatele porovnána se seznamem skupin v direktivě 'access' a pokud vyhovuje, je daný tag vykreslen (včetně obsahu a vnořených tagů). V opačném případě tag ani jeho obsah vykreslen není. Druhou částí zde je zabezpečení stránek aby uživatel bez oprávnění nemohl na tyto části webu vstoupit když by zadal ručně adresu

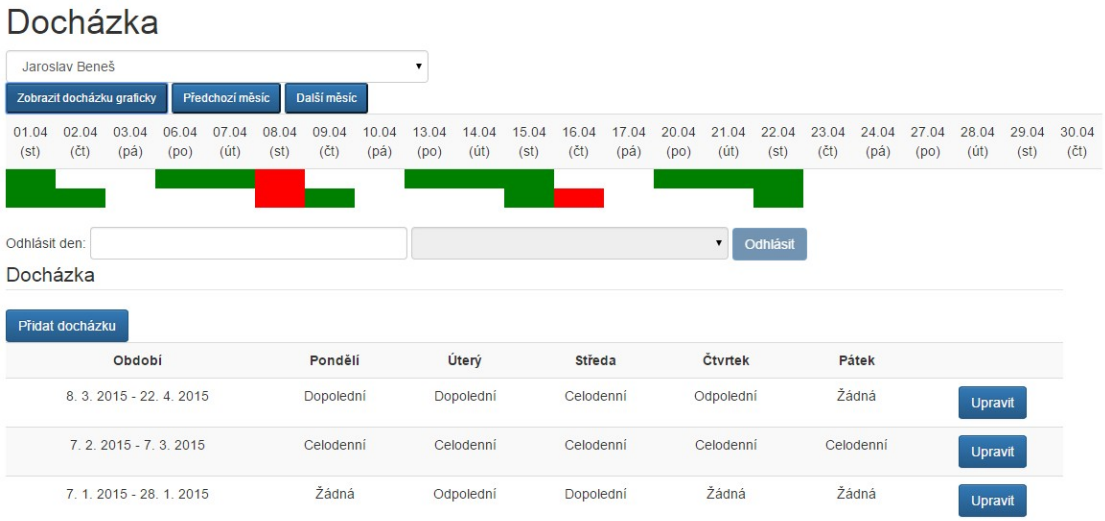
do prohlížeče. Každá z rout, která definuje jednu ze stránek obsahuje direktivu 'access', která je při každé změně routy vyhodnocena a podle výsledku je uživatel vpuštěn na stránku nebo je přesměrován na přihlašovací stránku.

Poslední částí je oveření uživatelských požadavků po jeho přihlášení. V případě, že je uživatel úspěšně autentizován je mu serverem přidělen jedinečný token, který klient musí vložit do každého požadavku, který na server odesílá. Server si podle tohoto tokenu získá právě přihlášeného uživatele a jeho práva. Pokud uživatel odešle požadavek bez tohoto tokenu (jinam než na Login) je mu odepřen přístup. Přidávání tokenu do všech odchozích HTTP požadavků je řešeno pomocí HTTP Interceptoru, který odchytává všechny HTTP požadavky a přidává k nim token, který je uložen u klienta.

5.4.3 Ukázka UI

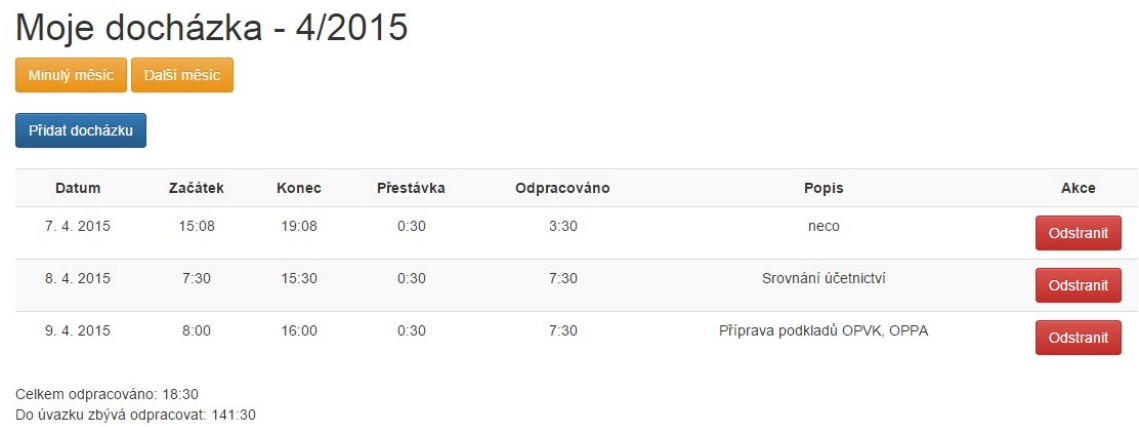
V této podsekci jsou uvedeny některé ukázky grafického rozhraní aplikace.

Obrázek 5.6 ukazuje zpracování sekce docházky dítěte. Ve vrchní části je graficky zobrazena docházka v aktuálním měsíci - zeleně objednaná, červeně absence a bíle části dnů bez docházky. Je zde možnost zapsat absenci na danou část dne. Ve spodní části stránky jsou poté zobrazeny aktuálně vedené docházky v textové podobě a je zde možnost docházku přidat či změnit existující.



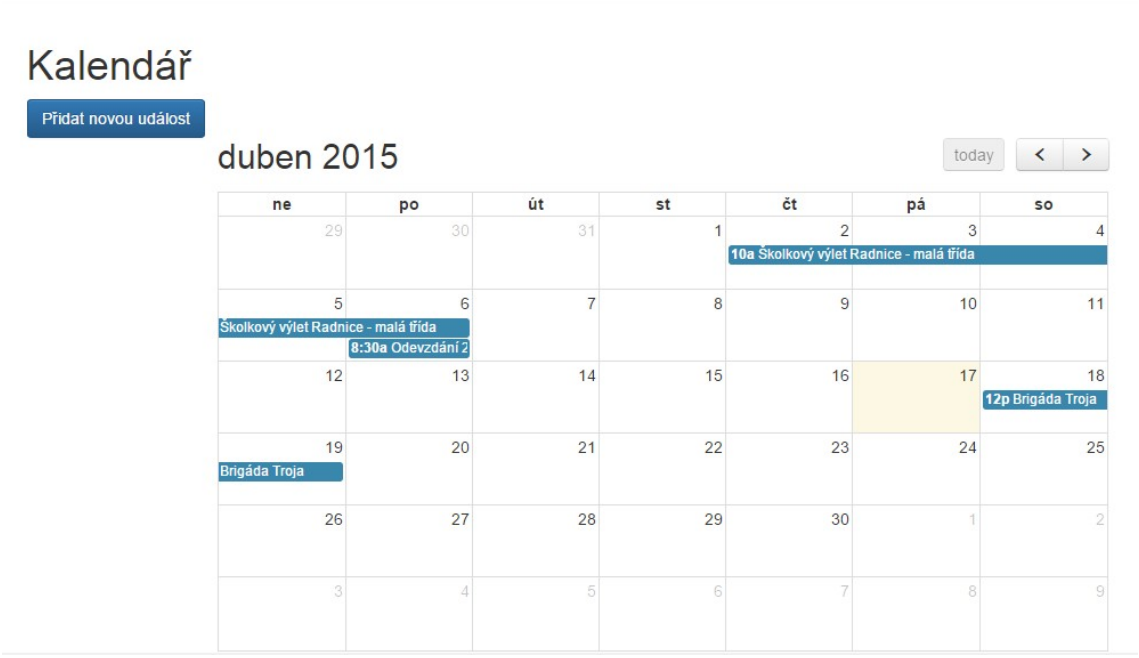
Obrázek 5.6: Docházka dítěte

Na obr.5.7 je sekce docházky aktuálně přihlášeného zaměstnance. Zaměstnanec zde může manipulovat s docházkou za aktuální a předchozí (maximálně pět dnů po jeho skončení) měsíc nebo si prohlížet svou docházku za měsíce minulé. V doní části stránky je zobrazen souhrnná doba, kterou zaměstnanec daný měsíc odpracoval a vztah této sumy k úvazku zaměstnance - tj. jaké má podčasy nebo přesčasy.



Obrázek 5.7: Odpracované hodiny zaměstnance

Obr. 5.8 ukazuje zobrazení kalendáře. Je zde možnost přidat novou událost s popisem a listovat mezi jednotlivými měsíci. V samotném kalendáři jsou zobrazeny již zadané akce - jednodenní nebo vícedenní.



Obrázek 5.8: Kalendář

Grafické zpracování sekce objednávek obědů dětí je na obr. 5.9. V horní části je rolovací menu pro výběr dítěte. Dále jsou zde tlačítka na zobrazení grafického znázornění objednávek obědů. Vlastní tabulka s obědy obsahuje několik druhů tlačítek:

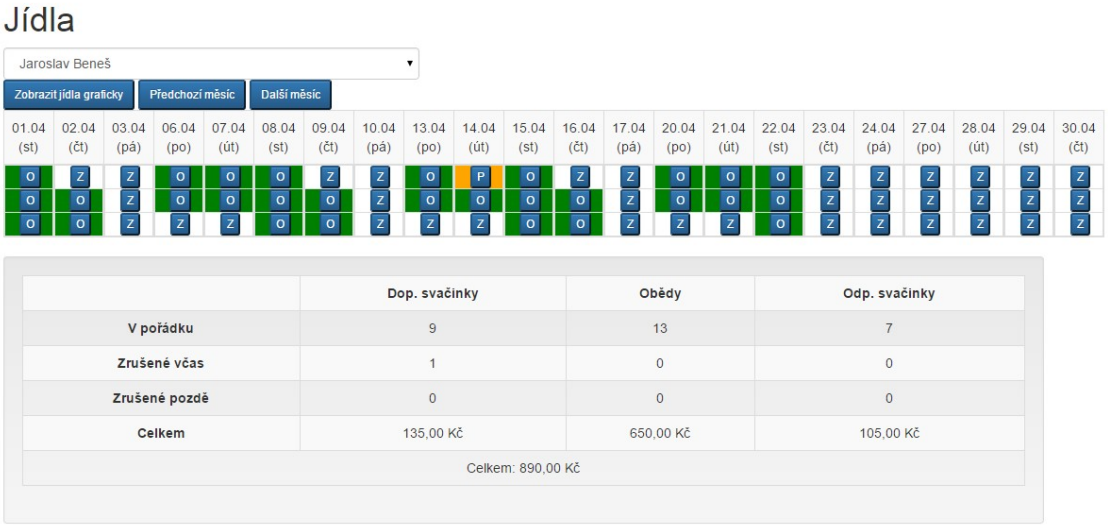
- O - odhlásí přihlášený oběd

- P - znovupříhlásí odhlášený oběd
- Z - založí nový oběd v čase na který nebyla udělána objednávka

Políčka jednotlivých jídel mají různé barevné značení:

- zelená - oběd je přihlášen
- žlutá - oběd je odhlášen včas
- červená - oběd je odhlášen pozdě
- bílá - oběd není přihlášen

Ve spodní části je poté zobrazený souhrn objednávek jídel a jejich cena pro vyúčtování.



Obrázek 5.9: Objednávky obědů dítěte

5.5 Nasazení

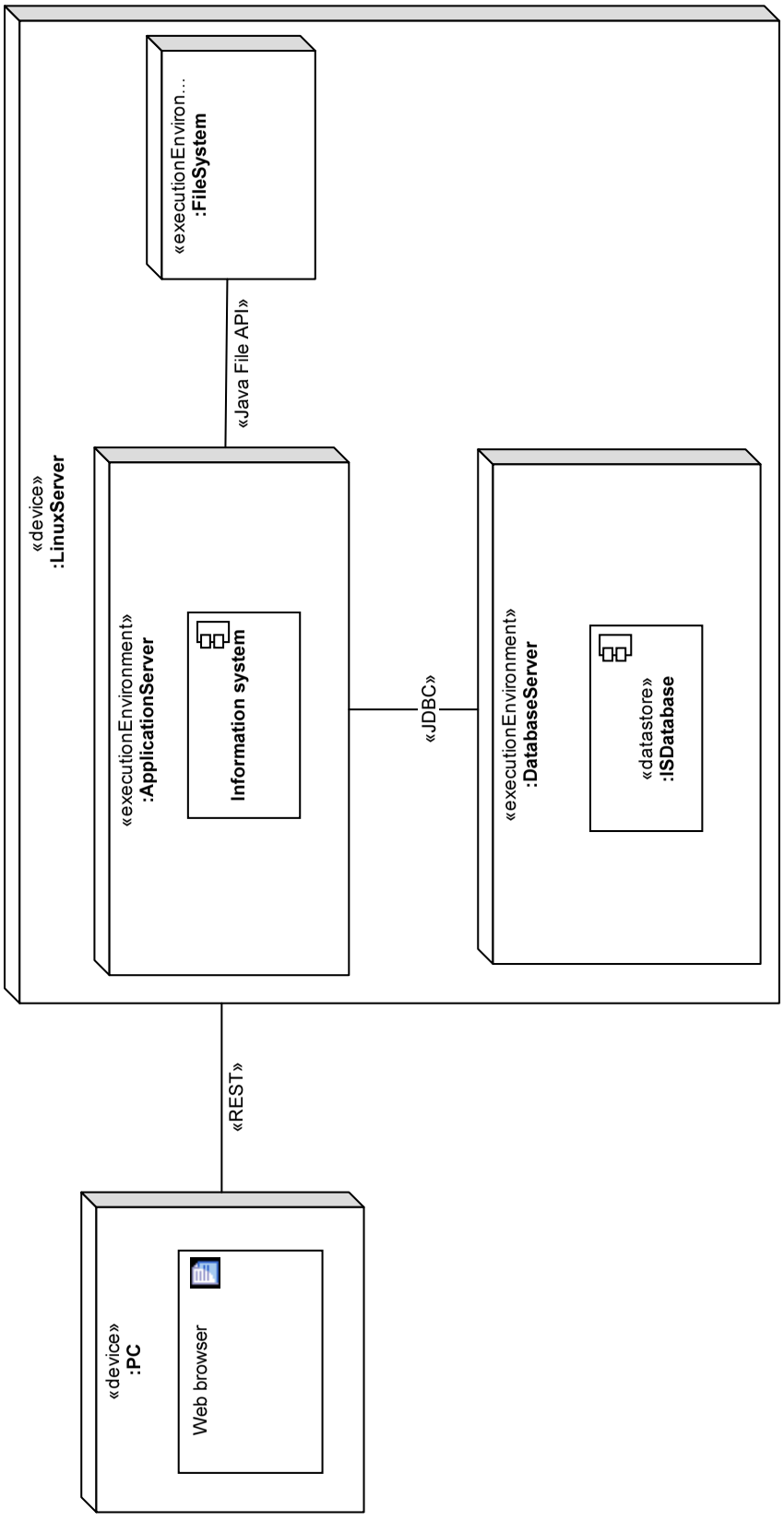
Diagram nasazení je na obr. 5.10. Celá serverová část běží na operačním systému GNU/Linux, konkrétně se jedná o distribuci Debian. Celý server běží na virtuálním stroji v pražském datacentru. Virtualizace stroje zajišťuje nízkou cenu provozu a zároveň nabízí široké možnosti rozšíření hardwarových prostředků v případě potřeby. Na serveru samotném jsou používány tři hlavní komponenty:

- Aplikační server - samotný server na kterém je aplikace nasazena
- Souborový systém - používán k ukládání uživatelských dat (převážně obrázků a dokumentů)

- Databázový server - slouží k ukládání dat z aplikace

Souborový systém a databázový server jsou obsluhovány prostřednictvím aplikačního serveru.

Klientská číst se skládá jen z webového prohlížeče s podporou JavaScriptu, který skrze HTTP protokol komunikuje s REST zdroji serveru a tím spravuje data na něm uložená. Na klientovi jsou uložena i drobná data, převážně kvůli zabezpečení a identifikaci uživateli, která se poté používají při komunikaci.



Obrázek 5.10: Diagram nasazení (deployment diagram)

Kapitola 6

Testování

Testování je nedílnou součástí každého softwarového projektu. Pro otestování tohoto projektu byly použity celkem tři různé postupy. Pro otestování nejmenších jednotek systému bez návaznosti na další části bylo použito unit testování. Pro otestování větších celků, a to hlavně pro třídy pracující s databází, bylo použito integračního testování. Dále bylo využito systémové testování, aby se prozkoumalo pokrytí případů use-case. Jako poslední bylo použito black-box testování, tedy testování grafického uživatelského rozhraní s budoucími uživateli systému.

6.1 Unit testování

Nazývané také jako jednotkové testování. Unit testování se zakládá na testování nejmenších částí systému nezávisle na ostatních částech[4]. Existuje několik frameworků na podporu testování. Pro tento projekt jsem vybral nejrozšířenější a nejlépe podporovaný framework a to JUnit[33] ve verzi 4.12. Testy v JUnit vypadají velmi podobně jako obyčejné java třídy. Hlavním nástrojem pro testování jsou assert metody. Pro splnění testu, musí všechny asserty v testovací metodě bezproblémově projít. Pokud i jediný selže, selhal celý test. Existuje několik druhů assertů, jak naznačuje tab.6.1

Další nedílnou součástí test case jsou anotace JUnit. Ty naznačují JUnit frameworku jak má zacházet s danými metodami. Nejdůležitější anotace shrnuje tabulka 6.2

Jelikož třídy modelu byly vytvořeny pomocí reverse engineeringu pomocí Hibernate tools[25] nemá smysl je testovat. Tento nástroj umožňuje pomocí přístupů reverse engineeringu projít zadané databázové tabulky a vytvořit z nich doménové třídy. Sám dokáže rozpoznat jednotlivé vztahy (a jejich druhy) a ty zanést do vytvořených tříd. Pokud je potřeba použít složitější vztahy (MappedSuperclass, SingleTable inheritance apod.) je nutno doménové objekty upravit manuálně.

V Unit testování byly samostatně otestovány třídy, které se nachází v těchto balíčcích:

- entity.converters
- entity.helper
- service.scheduler.tasks

Metoda	Účel
assertTrue(boolean condition)	Vyhodnotí se jako správná, pokud se condition vyhodnotí jako true.
assertFalse(boolean condition)	Vyhodnotí se jako správná, pokud se condition vyhodnotí jako false.
assertArrayEquals(Object[] expecteds, Object[] actuals)	Slouží k porovnání dvou polí objektů. Vyhodnotí se jako správná pokud jsou obě pole shodná.
assertEquals(Object expected, Object actual)	Porovnává dva libovolné objekty. Pokud jsou oba objekty shodné, vyhodnotí se jako správná.
assertNotNull(Object object)	Pokud objekt není null, vyhodnotí se jako správná.
assertThat(T actual, Matcher<T> matcher)	Do této metody lze dodat vlastní implementace porovnácího algoritmu (Matcher).

Tabulka 6.1: Důležité assert metody

Anotace	Význam
@Test	Definuje metodu, která je brána jako test.Může obsahovat více assertů, které musí být splněny najednou aby byl celý test splněn. Jedna třída může obsahovat více metod s touto anotací.
@Before	Anotace používaná k označení metody, která se zavolá před započetím testu. Je možné ji využít pro vytvoření kontextu nebo např. k získání původních dat.
@After	Označuje metodu, která se zavolá po dokončení testů. Vhodné např. k uklizení kontextu nebo navrácení původních hodnot.

Tabulka 6.2: Důležité anotace JUnit

- helper
- security

6.2 Integrační testování

V integračním testování již nejde o testování malých samostatných jednotek, které jsou nezávislé na okolí, ale přechází se na testování již větších celků. V základu se vychází z jednotek otestovaných unit testingem. U nich se ví, že jako samostatné jednotky fungují. Zde se poskládají do větších celků a ty jsou podrobeny testování. Existují tři základní způsoby testování[26]:

- Top-down - nejdříve se otestují nejsvrchnější části systému. Tento přístup dovoluje rychlé otestování vysokoúrovňové logiky a toku informací. Minimalizuje nutnost použití nadřazených objektů. Vyžaduje vytvoření objektů, které napodobují spodní vrstvy.
- Bottom-up - jako první se otestují nejnižší vrstvy a postupuje se výš. Nevýhodou je vytváření tříd emulujících nadřazené vrstvy.
- Umbrella approach - bere si z něco z každého předešlých přístupů. Zaměřuje se hlavně na testování modulů, se kterými je spojen vysoký stupeň uživatelské interakce.

Většina testovacích příruček a návodů [1] [4] doporučuje vytvářet tzv. Mock objekty, které se podstrčí testům a které simulují reálné prostředí. Použitý testovací framework v této práci nabízí podporu automatického rollbacku transakcí, které byly započaty v testu, a tím se databáze vrátí do původního stavu. Tato možnost byla využita a integrační testování tak proběhlo nad reálnými daty a reálnou databází.

V této práci byl použit testovací framework Arqullian[16]. Tento framework nabízí možnost vytvořit micro-deployment, který obsahuje pouze potřebné testované třídy a třídy a jejich závislosti. Takto miniaturizovaný archiv je poté nasazen na server. Tento server může mít tři podoby:

- embedded - server je obsažen v Arquillianu a ten se stará o celý jeho životní cyklus.
- managed - server je jako externí entita a Arquillian se stará pouze o životní cyklus (spuštění, vypnutí, deploy).
- remote - server je externí entita a Arquillian ho vzužívá pouze na deploy aplikace. O životní cyklus server se tedy musí starat jiná entita.

V této práci byly testy nasazovány na reálný aplikační server nad kterým aplikace poběží - WildFly 8.1.0. Po nasazení je automaticky spuštěn a vyhodnocen zadaný test, který běží ve stejném kontejneru jako reálná aplikace.

Testovací třída je strukturou velmi podobná třídám unit testů. Ukázka takové třídy je na obrázku 1.

Výpis 1: Ukázka integračního testu

```
@RunWith(Arquillian.class)
public class ChildrenIT {
@Inject
private ChildService service;

@Deployment
public static Archive<?> createDeployment() {
        WebArchive w = ShrinkWrap.create(WebArchive.class).addClass(ChildrenIT.
            class)
            .addAsResource("META-INF/persistence.xml", "META-INF/persistence.
                xml")
                        .addPackages(true, "service", "dao", "entity", "
                            resources", "security")
                        .deletePackage("service.scheduler.tasks")
                        .addAsLibraries(
                                        Maven.resolver()
                                        .loadPomFromFile("pom.xml")
                                        .importRuntimeAndTestDependencies()
                                        .asFile()
                        )
            .addAsWebInfResource(EmptyAsset.INSTANCE, "beans.xml");
        return w;
}

@Test
@Transactional(TransactionMode.ROLLBACK)
public void createChild() {
        int origCount = service.getAll(Child.class).size();
        Child c = new Child();
        c.setName("Pepa");
        c.setLastname("Novak");

        service.add(c);

        int newCount = service.getAll(Child.class).size();

        Assert.assertTrue(origCount == (newCount-1));
}
}
```

Testovací třída obsahuje tři hlavní anotace:

- @RunWith(Arquillian.class) - anotace označující tuto třídu jako Arquillian test
- @Deployment - anotace označující statickou funkci, která je použita pro vytvoření archivu (typicky JAR, WAR nebo EAR) který je následně nasazen na server
- @Test - anotace, která definuje samotnou testovací metodu

Statická funkce, která je označena anotací @Deployment, má v testu speciální význam, je v ní definován obsah budoucího testovacího archivu. V tomto konkrétním případě (obr.1) se do archivu (WAR) přidávají třídy z balíčků ‘service‘, ‘dao‘, ‘entity‘, ‘resources‘ a ‘security‘. Dále je také přidána samotná testovací třída ‘ChildrenIT‘. Dále je do archivu přidán soubor ‘persistence.xml‘ a prázdný soubor ‘beans.xml‘ (nutný pro správnou funkcni CDI). Dále jsou pomocí Maven resolveru dynamicky přidány potřebné knihovny, definované pro reálnou aplikaci. Nakonec je celý tento balík zabalen do WAR archivu pomocí nástroje ShrinkWrap[44] (součást balíku Arquillian) a nasazen na server.

Testovací metoda, označená anotací @Test, obsahuje samotný kód testu. Stejně jako v unit testech se k vyhodnocení testu využívají assert metody. V případě, že taková assert metoda selže, je vyvolána vyjímka a celý test selhal. Assert nabízí statické metody, které dovolují porovnávat různé datové struktury, vyhodnocovat boolean podmínky apod. Na metodu byla dále aplikována anotace @Transactional s hodnotou 'TransactionMode.ROLLBACK', která zaručuje, že všechny změny v databázi nebudou trvalé.

6.3 Systémové testování

Systémové testování staví na modulech otestovaných pomocí integračního testu. Ty jsou zase postaveny na unit testech. Tímto přístupem se snažíme eliminovat a zjednodušit hledání případné chyby.

Systémové testování má u každého projektu jiný rozsah. Někdy se může jednat jen o letmé odzkoušení funkčnosti. Jindy se může jednat o robustní testy, které se snaží napodobit všechny možné situace a prostředí do kterých se aplikace může dostat. Cílem testování v tomto projektu je otestování nejdůležitějších use case scénářů použitých při návrhu projektu.

Ke každému systémovému testu je vytvořen Test Case. Test case je jakýsi návod jak má daný test proběhnout. Dále obsahuje výchozí požadavky, které musí být splněny před spuštěním testu a očekávaný výstup [11]. Jako poslední, ale neméně důležitou, část obsahuje také jedinečný identifikátor.

Vzhledem k rozsáhlosti projektu a počtu testovacích scénářu jsou vybrané scénáře uvedené v příloze H.

6.4 Testy uživatelského rozhraní

Pro toto testování byla aplikace nasazena na reálný produkční server, stejně jako má být nasazena po jejím vývoji. Testování probíhalo s reálným budoucím uživatelem. Interakce se systémem probíhala na jeho počítači, stejně tak jako bude probíhat po dokončení vývoje a v ostrém provozu.

Testerovi byly předkládány různé testy, které musel v programu splnit. Testy, které tester podstoupil jsou uvedeny v tab.I.1 i s jejich stručným popisem a výsledkem testu.

Vývoj grafického rozhraní byl pravidelně konzultován se zaměstnanci Organizace, s testy tedy nebyl žádný výrazný problém. Objevili se pouze drobné problémy, které byly bez problémů nalezeny a odstraněny.

6.5 Výkonnostní testování

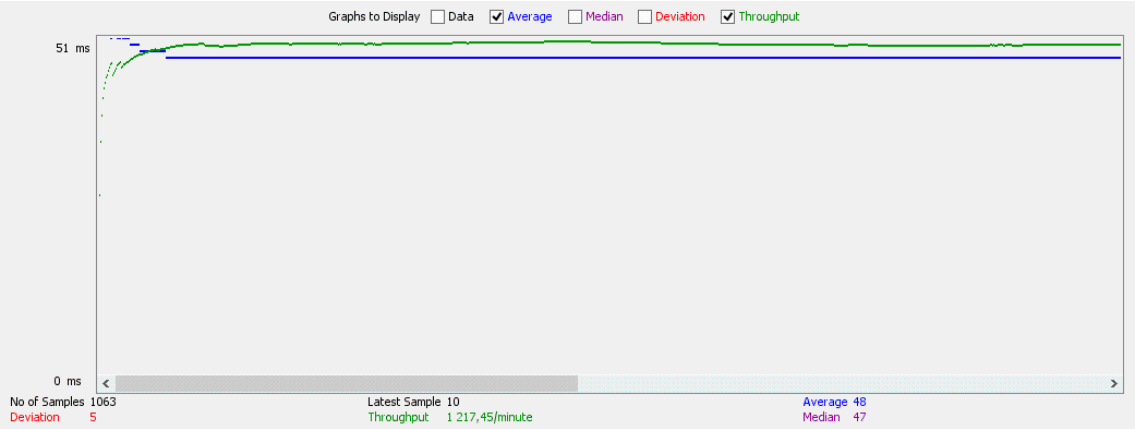
Výkonnostní testování (stress testing) se snaží napodobit přístup mnoha uživatelů k systému a zkoumá chování systému - hlavně jeho odezvu.

K testování byla využita aplikace Apache JMeter[29]. Tato open-source aplikace je určená pro simulaci přístupu nastaveného počtu uživatelů na nastavený zdroj a sledování odezvy od tohoto zdroje. Kromě možnosti testování REST zdrojů, jak je uvedeno dále, umožňuje

testovat i FTP servery, SOAP endpointy nebo aplikace napsané ve webových jazycích (Java, PHP, ASP.NET apod).

Základní nastavení pro testování v této práci se sestává z objektu Thread Group, který obsahuje konfiguraci počtu uživatelů, doby za kterou se začnou postupně připojovat a počet opakování testu. Dalším objektem je HTTP Request Defaults, kde se definují základní požadavky pro všechny testy týkající se HTTP requestu (adresa serveru, port, základní cesta). Dále je, kvůli ověření uživatele vůči serveru, nutno přidat objekt HTTP Header Manager ve kterém je definováno zasílání bezpečnostního tokenu v každé hlavičce HTTP dotazu. Samotné testování probíhá pomocí sampleru HTTP Request, kde se nastaví již jen cesta ke zdroji, který se má testovat a metoda přístupu (POST, GET...). Poslední objekt nese označení Graph Results. Tento objekt vytváří a zobrazuje graf celého testu. Je v něm vidět jak propustnost (Throughput, tj. kolik požadavků za minutu probíhá), tak také statistiky ohledně doby odezvy (aktuální, median, průměr a odchylka).

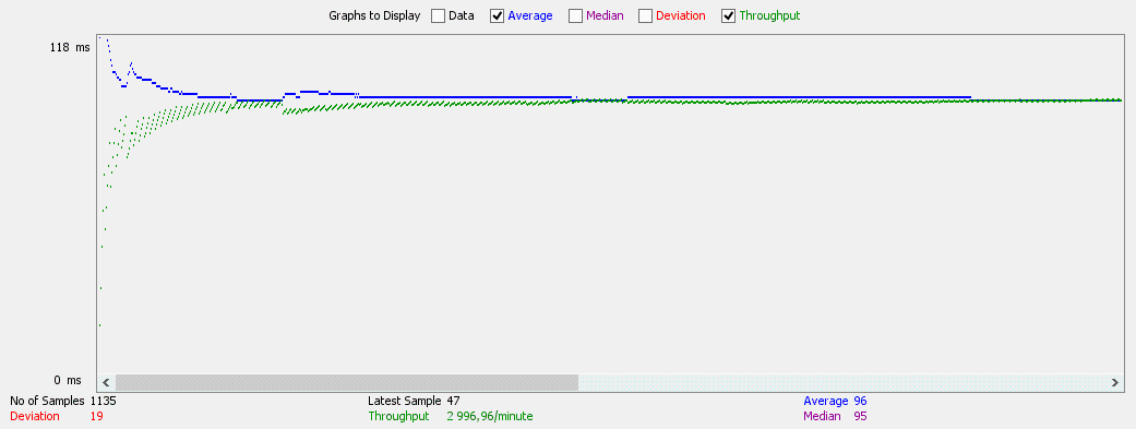
Základní měření bylo provedenou pouze pro jednoho uživatele, který opakovaně zasílal požadavek na REST zdroj children - jedná se o zdroj, ve kterém je nejvíce dat a lze tedy předpokládat, že jeho získání bude trvat nejdéle a zatíží server nejvíce. Výsledku testu jsou zobrazeny na obr.6.1. Z grafu vyplývá, že jeden uživatel dosáhl propustnosti 1217 úspěšných požadavku za minutu (tj. cca 20 požadavků za sekundu). Přičemž průměrná doba odezvy serveru byla 48ms.



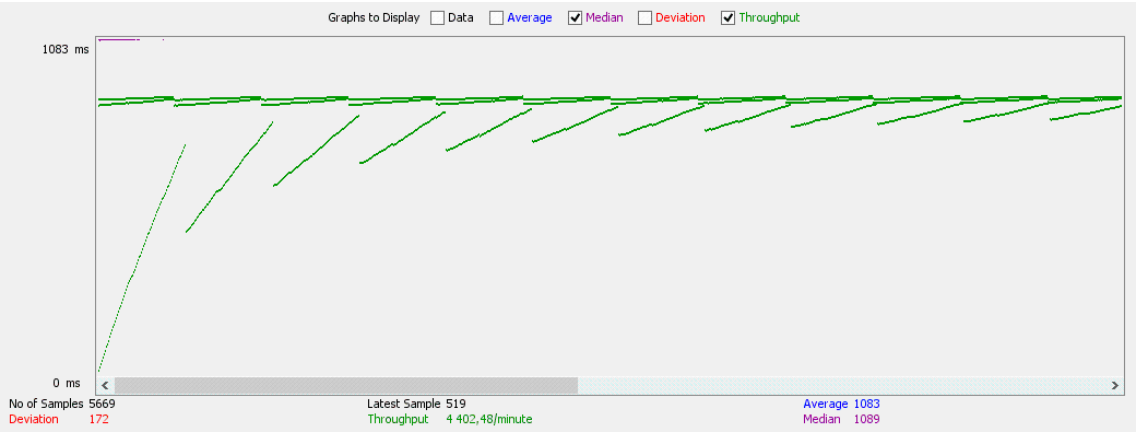
Obrázek 6.1: Zátěžový test s jedním uživatelem

Další měření bylo provedeno s pěti uživateli zároveň. Tento počet uživatelů je reálný odhad, kterým by mohl být systém zatížen. Graf tohoto měření je zobrazen na obr.6.2. Průměrná odezva serveru při tomto zatížení byla 96 ms. Tato odezva je pro práci s aplikací naprosto vyhovující a nebude uživatelům činit žádné problémy.

Pro zajímavost byl proveden i test s 80 paralelně přistupujícími uživateli ke zdrojům serveru. Výsledky jsou zobrazeny na obr.6.3. Z grafu vyplývá, že odezva od serveru výrazně stoupla, průměrně na 1077 ms. Vezmeme-li v potaz, že testy probíhaly proti nejrozsáhlejšímu zdroji, který je na serveru k dispozici není výsledná hodnota vůbec špatná. I při odezvách kolem sekundy lze provádět většinu operací bez výraznějšího omezení.

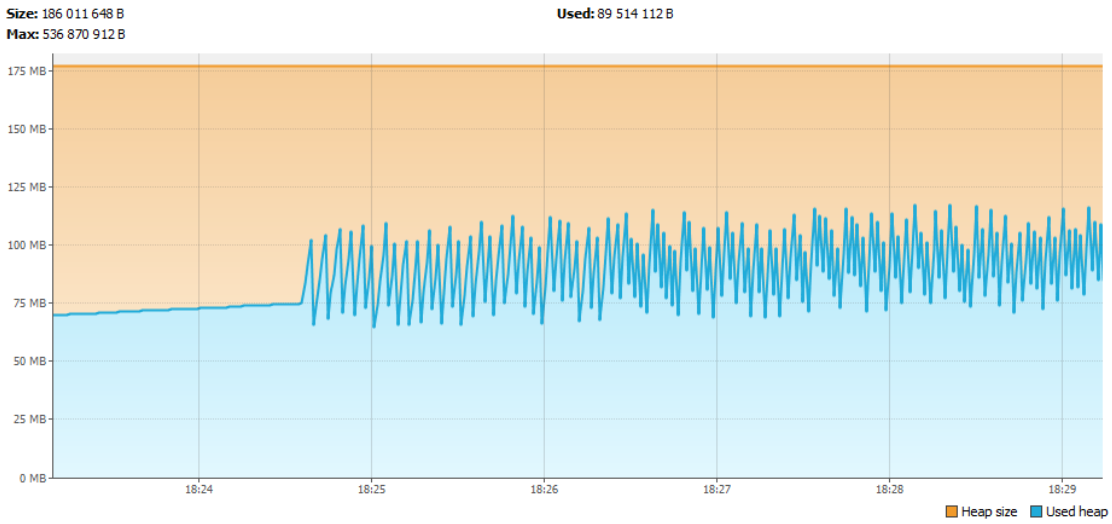


Obrázek 6.2: Zátěžový test s pěti uživateli



Obrázek 6.3: Zátěžový test s osmdesáti uživateli

Společně při testování výkonosti aplikace byla pozorována i paměťová náročnost aplikace nasazené na produkčním serveru. K tomu účelu byla využita aplikace jstatd[32], která umožňuje publikovat rozhraní pro monitoring běžících JVM strojů. Na toto rozhraní byla vzdáleně napojena aplikace VisualVM[49]. Tato aplikace umožňuje monitorování paměťové náročnosti java aplikací. Výstup z této aplikace je na obr.6.4. Minimální paměť, přidělená aplikačnímu serveru je nastavena na 64MB, maximální poté na 768MB. Jak z grafu vyplývá, při zátěži deseti uživateli paměťová náročnost nepřekročí 125MB. Na grafu je také vidět funkce garbage collectoru, který způsobuje 'zubatost' grafu využití haldy.



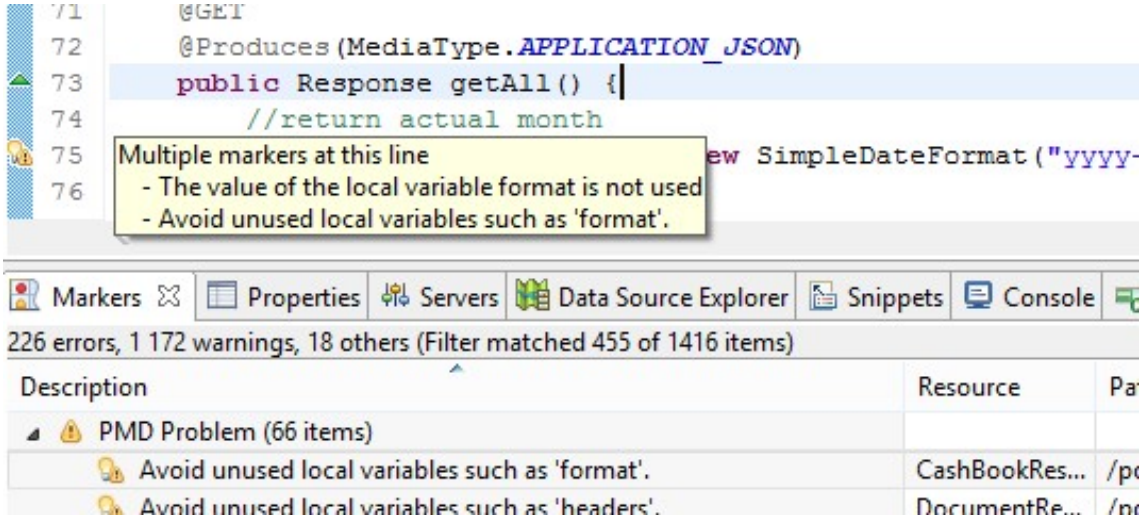
Obrázek 6.4: Měření paměťové náročnosti aplikace

6.6 Statická analýza kódu

Statická analýza kódu je disciplína testování při které je zkoumán zdrojový kód programu bez toho aby byl spouštěn [2]. Tato metoda se zabývá detekcí chyb, kterých se programátor mohl dopustit při vytváření zdrojového kódu. V této práci byly využity nástroje, které nabízí 'style checking' a 'bug finding' [2].

Style checking je metoda, kdy se automatizovaný nástroj snaží vyhledat ve zdrojovém textu programu chyby ve stylu. Jedná se například o použití zastaralých metod, špatných jmenných konvencí, komentářů, struktury programu apod. Celkově se jedná o chyby, které se projevují v horší čitelnosti zdrojového kódu a jeho udržitelnosti. Na vyhledání takovýchto chyb v kódu byl využit program PMD [38]. Jedná se o open-source nástroj, který podporuje analýzu kódů v množství jazyků, včetně Javy a JavaScriptu. Eclipse nabízí plugin, eclipse-PMD [39], který integruje funkcionalitu PMD do jeho prostředí a výstup PMD je tak přístupný přímo v IDE. Ukázka zobrazení chyby v prostředí Eclipse je na obr. 6.5

Bug finding je metoda, kdy se nástroj snaží vyhledat místa, kde se bude program chovat jinak než programátor očekával při jejích vytváření. Funguje na základě předdefinovaných pravidel, který se snaží vyhledat. Některé nástroje také odvozují nová pravidla z právě analyzovaného kódu. Pokud například na 99 místech kódu je použit stejný synchronizační zámek pro přístup k jedné proměnné, je pravděpodobné, že stejný zámek by měl být použit i na 100 místě. K identifikaci těchto problémů byla využita aplikace FindBugs[22], která nabízí možnost inspekce bytecodu aplikace a výpis nalezených chyb. Při přiřazení zdrojového kódu je aplikace schopna i zobrazit místo v kódu, kde chyba vznikla.



Obrázek 6.5: Zobrazení chyby PMD v prostředí Eclipse

Kapitola 7

Budoucnost

Současný stav systému je vyhovující pro jeho postuné nasazení a pro přechod Organizace na jeho využívání. Do budoucna bude nutné systém rozšířit o další funkcionalitu.

Architektura tohoto systému je připravena na vytvoření a nasazení nativní mobilní aplikace, která nebude využívat webové rozhraní, ale bude přímo přistupovat na REST rozhraní, ze kterého může přímo získávat (a odesílat) potřebná data.

Dalším prvkem, který bude nutné dodělat, je rozhraní pro vytváření a editaci ekologických výukových programů. Jde o databázi ve které budou zanesena data o EVP jako je název, počet účastníků, lektor, místo konání apod. Tyto údaje budou následně použity pro vykazování práce Organizace v rámci projektu, který sponzoruje hlavní město Praha.

Pro organizaci důležitým prvkem bude také vytvoření evidence HR požadavků jako jsou dovolené, nemocenské, výjezdy zaměstnanců apod. Každá dovolená musí být také schválena ředitelkou Organizace, bude tedy nutné vytvořit rozhraní i pro tuto funkcionalitu.

Dalším možným rozšířením je napojení na externí systém pro rozesílání hromadných e-mailů nebo SMS zpráv s informacemi o platbách rodičů (ať o potvrzení příchozí platby nebo výzva k úhradě).

Kapitola 8

Závěr

Cílem této práce bylo seznámení se s chodem neziskové organizace a na základě těchto informací navrhnout informační systém, který by ulehčil práci s informacemi v této organizaci. Po tomto seznámení bylo shledáno, že způsob práce s informacemi v rámci organizace je nedostatečný, komplikovaný a necentralizovaný. Neexistovalo žádné zálohování dat, což způsobovalo problémy při selhání zaměstnaneckých počítačů, které obsahovaly pouze jednu kopii dat.

Na základě těchto zjištění proběhla analýza požadavků organizace na informační systémy pro správu dat. Následně proběhl průzkum trhu, kdy byly prozkoumány běžně dostupné a používané informační systém. Žádný z těchto systémů nesplňoval všechny kritické požadavky na systém Orgfanizace. Následně byla tedy z těchto požadavků vytvořena analytická dokumentace budoucí aplikace, která obsahovala soupis všech požadavků na systém, jednotlivé aktéry systému, doménový model a všechny ostatní potřebné dokumenty. Dalším krokem bylo vytvoření návrhové dokumentace, která obsahuje diskusi nad výběrem technologie, návrh systému ve formě sekvenčních diagramů a popisu technologie. Z těchto dokumentací byla vytvořena implementace serverové a klientské části aplikace, která byla následně otestována. Testování probíhalo ve více úrovních (od unit testů až po testování s uživateli) tak, aby bylo zajištěno co možná největší pokrytí testy.

Při návrhu a implementaci byly využity moderní technologie a postupy, tak aby byla implementace dlouhodbě udržitelná a snadno rozšiřitelná o další funkcionalitu.

Při následných konzultacích s Organizací při implementaci a testování práce byly objeveny další funkcionality, které by Organizace mohla využít. Nabízí se tak další možnosti rozšíření stávajícího systému tak, aby přinášel Organizaci co největší užitek a usnadnění stávající práce.

Literatura

[1] Adam Bien. Integration testing. <http://www.oracle.com/technetwork/articles/java/integrationtesting-487452.html>, stav z 6. 4. 2015.

[2] Jacob West Brian Chess. *Secure Programming with Static Analysis*, volume 1. Addison-Wesley, 75 Arlington Street Suite 300 Boston MA 02116 USA, 1st edition, 2007.

[3] Ralph Johnson Erich Gamma, Richard Helm and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, volume 1. Addison-Wesley Professional, Boston, MA, USA, 1st edition, 1995.

[4] Steve Loughran Erik Hatcher. *Java Development with Ant*, volume 1. Manning, 209 Bruce Park Avenue, Greenwich, CT 06830, 1st edition, 2003.

[5] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process (2nd Edition)*. Prentice Hall PTR, 2nd edition, 2001.

[6] Bill Shannon Linda DeMichiel. Java^{TM} platform, enterprise edition 7 (java ee 7) web profile specification. <https://java.net/downloads/javaee-spec/WebProfile.pdf>, stav z 10. 9. 2014.

[7] Matthew Foemmel Edward Hieatt Robert Mee Randy Stafford Martin Fowler, David Rice. *Patterns of Enterprise Application Architecture*, volume 1. Addison-Wesley Professional, Boston, MA, USA, 1st edition, 2002.

[8] Granville Miller. Java modeling: A uml workbook, part 1. <http://www.ibm.com/developerworks/java/library/j-jmod0508/>, stav z 11. 9. 2014.

[9] Oracle. Java ee compatibility. <http://www.oracle.com/technetwork/java/javaee/overview/compatibility-jsp-136984.html>, stav z 10. 9. 2014.

[10] Oracle. Java platform, enterprise edition (ee). <http://www.oracle.com/technetwork/java/javaee/overview/index.html>, stav z 10. 9. 2014.

[11] Ron Patton. *Software Testing*, volume 1. Sams Publishing, 800 East 96th Street, Indianapolis, Indiana 46240, 2nd edition, 2005.

[12] Ecma 404 the json data interchange format. <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>, stav z 12. 3. 2015.

[13] About - wildfly. <http://wildfly.org/about/>, stav z 16. 4. 2015.

[14] Angularjs — superheroic javascript mvw framework. <https://angularjs.org/>, stav z 10. 3. 2015.

[15] The apache http server project. <http://httpd.apache.org/>, stav z 30. 3. 2015.

[16] Arquillian - write real tests. <http://arquillian.org/>, stav z 6. 4. 2015.

[17] Bakaláři. <http://www.bakalari.cz/>, stav z 11. 9. 2014.

[18] Bootstrap - the world's most popular mobile-first and responsive front-end framework. <http://getbootstrap.com/>, stav z 12. 3. 2015.

[19] Apache deltaspike. <https://deltaspike.apache.org/>, stav z 12. 3. 2015.

[20] Scheduler module. <https://deltaspike.apache.org/documentation/scheduler.html>, stav z 12. 3. 2015.

[21] edookit - školní informační systém. <http://www.edookit.cz/>, stav z 29. 3. 2015.

[22] Findbugs - find bugs in java programs. <http://findbugs.sourceforge.net/>, stav z 20. 4. 2015.

[23] Glassfish server. <https://glassfish.java.net/>, stav z 11. 9. 2014.

[24] Hibernate. <http://hibernate.org/>, stav z 11. 9. 2014.

[25] Hibernate tools. <http://hibernate.org/tools/>, stav z 3. 4. 2015.

[26] Integration testing. <http://msdn.microsoft.com/en-us/library/aa292128%28v=vs.71%29.aspx>, stav z 6. 4. 2015.

[27] iŠkola. <https://www.iskola.cz/>, stav z 11. 9. 2014.

[28] Jackson json processor. <http://jackson.codehaus.org/>, stav z 12. 3. 2015.

[29] Apache jmeter. <http://jmeter.apache.org/>, stav z 13. 3. 2015.

[30] Joda-time. <http://www.joda.org/joda-time/>, stav z 12. 3. 2015.

[31] jsog/jsog. <https://github.com/jsog/jsog>, stav z 12. 3. 2015.

[32] jstatd - virtual machine jstat daemon. <http://docs.oracle.com/javase/7/docs/technotes/tools/share/jstatd.html>, stav z 13. 3. 2015.

[33] Junit. <http://junit.org/>, stav z 3. 4. 2015.

[34] Mysql. <http://www.mysql.com/>, stav z 11. 9. 2014.

[35] Picketlink. <http://picketlink.org/>, stav z 12. 3. 2015.

[36] Picketlink about. <http://picketlink.org/about/>, stav z 12. 3. 2015.

[37] Play framework. <https://www.playframework.com/>, stav z 11. 9. 2014.

[38] Pmd. <http://pmd.sourceforge.net/>, stav z 20. 4. 2015.

[39] Pmd for eclipse. <http://pmd.sourceforge.net/eclipse/>, stav z 20. 4. 2015.

[40] Postgresql. <http://www.postgresql.org/>, stav z 11. 9. 2014.

[41] Quartz scheduler. <http://quartz-scheduler.org/>, stav z 12. 3. 2015.

[42] Red hat. <http://www.redhat.com/en>, stav z 16. 4. 2015.

[43] Resteasy - jboss community. <http://resteasy.jboss.org/>, stav z 12. 3. 2015.

[44] shrinkwrap. <https://github.com/shrinkwrap/shrinkwrap>, stav z 6. 4. 2015.

[45] Škola online. <http://www.skolaonline.cz/>, stav z 11. 9. 2014.

[46] Spring. <http://spring.io/>, stav z 11. 9. 2014.

[47] Spring framework. <http://projects.spring.io/spring-framework/>, stav z 11. 9. 2014.

[48] Angular directives for bootstrap. <http://angular-ui.github.io/bootstrap/>, stav z 12. 3. 2015.

[49] Visualvm. <http://visualvm.java.net/>, stav z 16. 3. 2015.

[50] Wildfly. <http://wildfly.org/>, stav z 11. 9. 2014.

Příloha A

Obsah CD

Součástí této práce je i přiložené CD s tímto obsahem:

- Složka Text
  - koukavoj_DIP.pdf - tisknutelná verze této práce ve formátu PDF
  - koukavoj_DIP - složka obsahující teXlipse projekt včetně obrázků v plném rozlišení
- Složka Source
  - Složka src - zdrojové soubory serverové části
  - Složka WebContent - zdrojové soubory klientské části

Příloha B

Seznam použitých zkratek

HTTP Hypertext Transfer Protocol

RBAC Role-Based Access Control

CRUD Create, Read, Update, Delete

CDI Contexts and Dependency Injection

REST Representational State Transfer

JAX-RS Java API for RESTful Web Services

DAO Data Access Object

MVC Model-View-Controller

SD Sequence Diagram

CSS Cascading Style Sheet

JS JavaScript

ORM Object-relational Mapping

WAR WebArchive

JAR Java Archive

JPA Java Persistence API

EAR Enterprise Archive

HTML Hypertext Markup Language

SOAP Simple Object Access Protocol

PHP PHP: Hypertext Preprocessor (původně Personal Home Page)

FTP File Transfer Protocol

JVM Java Virtual Machine

IDE Integrated Development Environment

HR Human Resources

IoC Inversion of Control

EVP Environmentální Výukový Program

UI User Interface

DPP Dohoda o Provedení Práce

Příloha C

Funkční požadavky

C.1 Uživatelé

V systému budou následující uživatelské role:

- Administrátor
- Zaměstnanec
- Učitel
- Rodič
- Anonym

C.2 Zaměstnanci

C.2.1 Evidence údajů o zaměstnancích

Systém bude umožňovat vložení údajů o zaměstnancích a jejich změnu.

C.2.2 Evidence úvazků

Systém bude umožňovat u každého zaměstnance uvést jiný úvazek na který je zaměstnán v organizaci.

C.2.3 Seznam úkolů

Systém bude umožňovat vytvářet a přiřazovat úkoly jednotlivým zaměstnancům a přiřadit k nim datum splnění.

C.2.4 Upomínky úkolů

Systém bude umožňovat zasílat zaměstnancům e-maily s blížícími se termíny úkolů, které mají splnit.

C.2.5 Evidence dokumentů

Systém bude umožňovat uložit k zaměstnanci důležité dokumenty (smlouvy, dodatky apod).

C.2.6 Archivace zaměstnanců

Systém bude umožňovat zaměstnance archivovat.

C.2.7 Evidence brigádníků

Systém bude umožňovat evidenci brigádníků.

C.3 Projekty

C.3.1 Evidence projektů

Systém bude umožňovat vést evidenci projektů probíhajících v rámci organizace.

C.3.2 Evidence dokumentů

Systém bude umožňovat vést evidenci základních dokumentů ke každému projektu.

C.3.3 Kalendář projektu

Systém bude umožňovat přiřadit ke každému projektu upomínky s důležitými datumy (odevzdání žádosti, odeslání monitorovací zprávy apod.).

C.3.4 Odeslání upomínky na mail

Systém bude umožňovat odeslání upomínky na e-mail administrátorovi projektu při blížícím se termínu nějaké akce v kalendáři projektu.

C.3.5 Poznámky k projektu

Systém bude umožňovat přidávat poznámky k projektu s aktuálním datem.

C.3.6 Indikátory splnění projektu

Systém bude umožňovat vést evidenci indikátorů projektu (tj. např. jestli účast na projektu byla dostatečná, jestli počet studentů byl dostatečný apod.) a poznámky k nim.

C.3.7 Evidence kontaktů k projektu

Systém bude umožňovat vést evidenci kontaktů k projektu (projektový manažer, správce projektu apod.).

C.4 Provozní záležitosti

C.4.1 Evidence majetku

Systém bude umožňovat vést evidenci majetku v organizaci.

C.4.2 Databáze dokumentů

Systém bude umožňovat vést evidenci různých dokumentů(např. zápisy z porad) a odkazů na dokumenty (google drive apod.) a dělit je do kategorií.

C.4.3 Seznam kontaktů

Systém bude umožňovat vést databázi kontaktů rozdělenou podle kategorií (učitelé, rodiče, příznivci apod).

C.4.4 Hromadné e-maily

Systém bude umožňovat rozeslat hromadné e-maily na kontakty z databáze kontaktů (pozvánky na akce, důležitá upozornění apod).

C.4.5 Pokladní deník

Systém bude umožňovat vést pokladní deník s příjmy a výdaji.

C.4.6 Kalendář

Systém bude umožňovat vést evidenci různých událostí a jejich datumů.

C.5 Ekoškolka

C.5.1 Evidence dětí

Systém bude umožňovat vést evidenci dětí přihlášených do ekoškolky a jejich změnu.

C.5.2 Evidence speciálních informací o dítětí

Systém bude umožňovat vést u dítětě kromě základních informací i další (např. alergie, požadavek na pleny, zdravotní omezení apod.)

C.5.3 Evidence rodičů dětí

Systém bude umožňovat vést evidenci rodičů dětí a jejich změnu.

C.5.4 Evidence docházky děti

Systém bude umožňovat vést evidenci docházky(dopolední, odpolední, celodenní) dětí a její změnu.

C.5.5 Generování smluvních dokumentů

Systém bude umožňovat vygenerovat smluvní dokumenty pro rodiče (smlouva, evidenční list) podle zadaného vzoru.

C.5.6 Hromadná korespondence

Systém bude umožňovat posílat hromadně e-maily na adresy rodičů(dluhy, změny ve školce apod.).

C.5.7 Úprava informací o dítěti rodičem

Systém bude umožňovat rodiči změnit základní údaje o dítěti (kontakt, jméno apod.).

C.5.8 Statistiky dětí

Systém bude umožňovat zobrazit různé statistiky o dětěch(dluhy rodičů, kauce, předpoklad zisku apod.).

C.5.9 Upozornění na narozeniny

Systém bude umožňovat zaslat e-mail s upozorněním na narozeniny dítěte ve školce.

C.5.10 Archivace dítěte

Systém bude umožňovat archivovat dítě.

C.5.11 Generování archu s docházkou

Systém bude umožňovat vygenerovat PDF s dokumentem pro ruční zápis docházky dětí učitelem.

C.5.12 Automatická archivace dítěte

Systém bude umožňovat automatickou archivaci dítěte v přednastavené datum ukončení docházky dítěte.

C.5.13 Evidence dluhů

Systém bude umožňovat vést evidenci dluhů rodičů - za obědy a docházku.

C.5.14 Evidence plateb

Systém bude umožňovat stahování plateb z banky organizace a jejich automatické přiřazení k dětem.

C.5.15 Manuální párování plateb

Systém bude umožňovat manuální párování plateb k dětem.

C.5.16 Zrušení objednávky obědů rodiči a zaměstnanci

Systém bude umožňovat rodičům a zaměstnancům rušit objednané obědy (obědy budou objednány na základě docházky).

C.5.17 Statistiky obědů

Systém bude umožňovat zobrazit statistiky obědů(výnosnost, počty apod.).

C.5.18 Upomínky rodičům

Systém bude umožňovat zasílat rodičům na e-mail upozornění o blížícím se termínu zaplacení školného.

C.5.19 Přehled obědů pro rodiče

Systém bude umožňovat rodičům zobrazit přehled objednávek obědů.

C.5.20 Přehled plateb pro rodiče

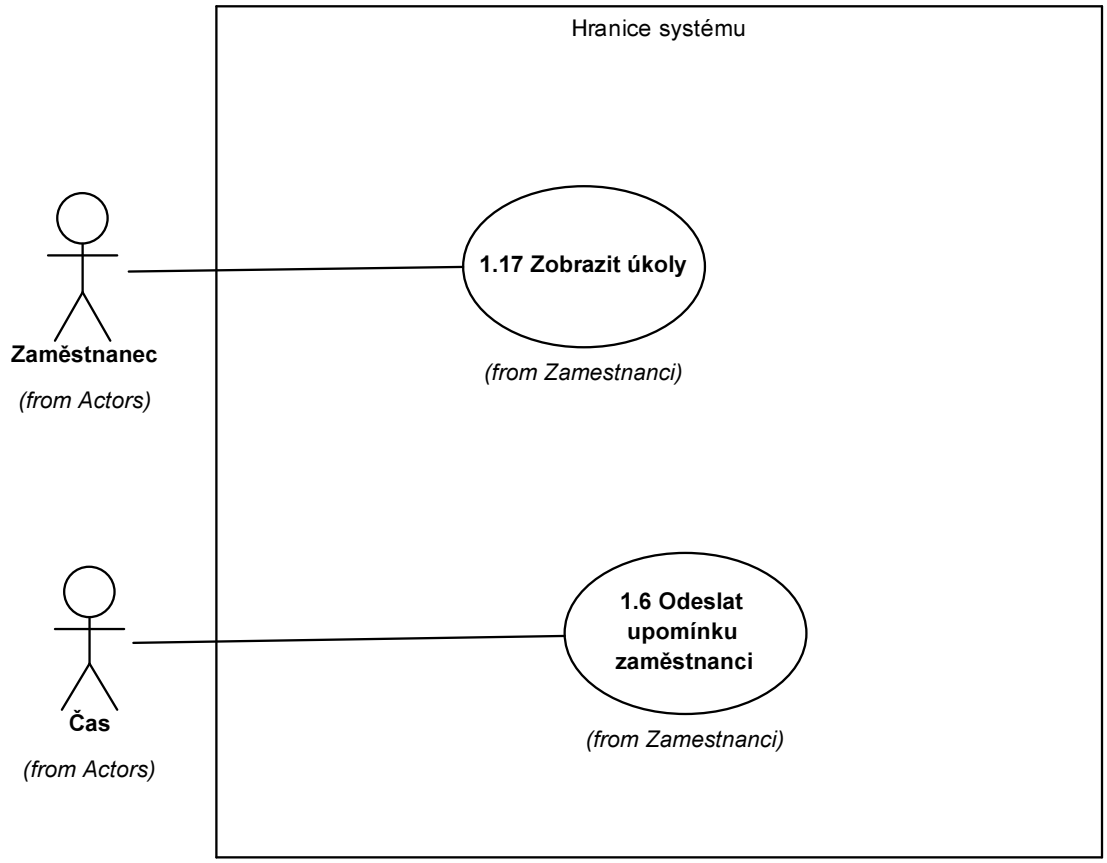
Systém bude umožňovat zobrazit přehled plateb za děti pro rodiče.

C.5.21 Nefunkční požadavky

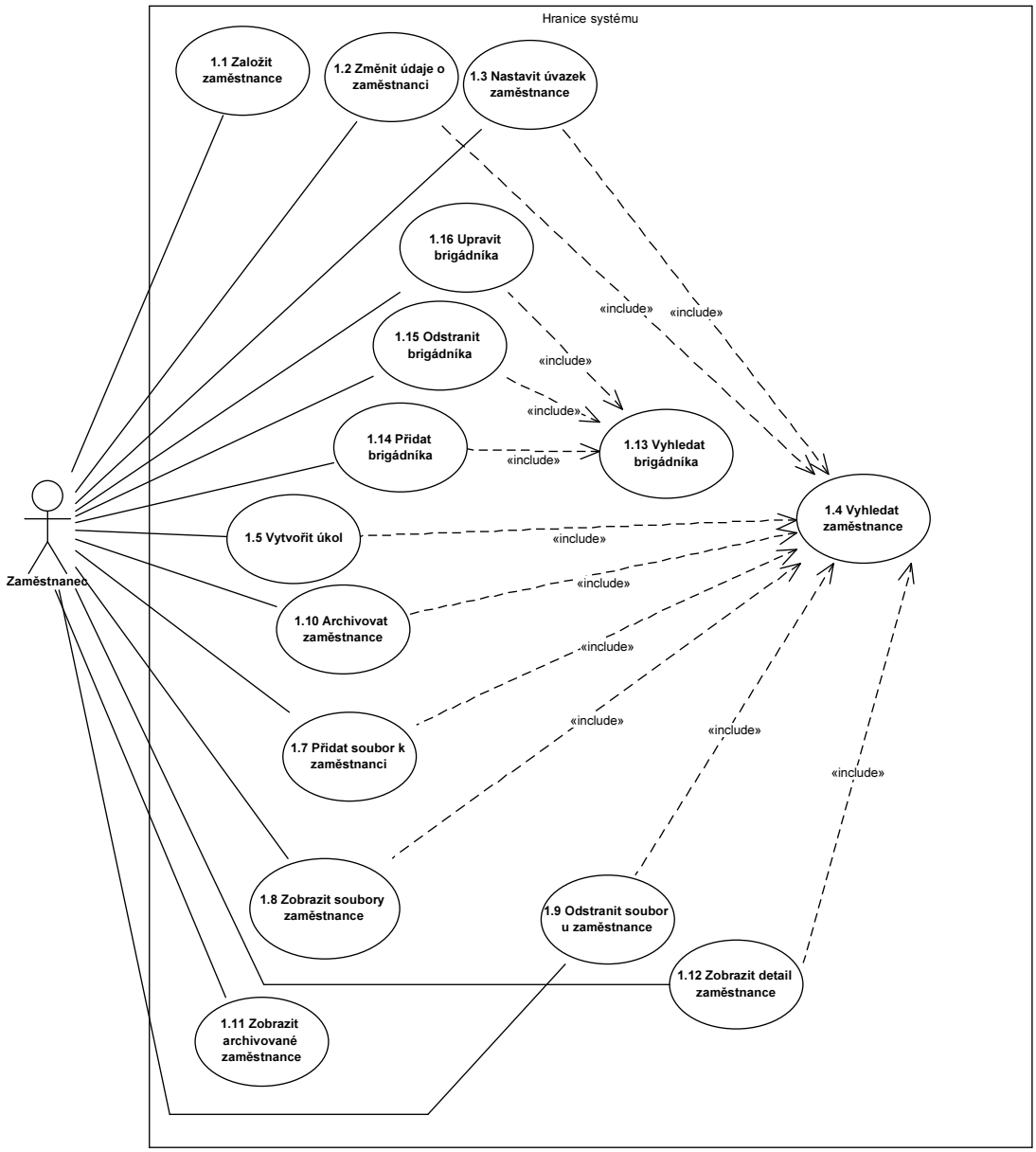
- Systém bude implementován v jazyce Java EE.
- Systém bude pro ukládání dat používat databázi PostgreSQL.
- Databáze systému bude pravidelně zálohována na externí úložiště.
- Uživatelé ve skupině Zaměstnanec budou mít přístup pouze do definovaných sekcí systému.
- Uživatelé ve skupině Rodič budou mít přístup pouze do sekce pro ně určené.
- Uživatelé s rolí administrátor budou mít kompletní přístup k celému systému, včetně jeho nastavení.
- Každý uživatel ze skupiny Zaměstnanec bude mít individuálně nastavené sekce, do kterých bude mít přístup.

Příloha D

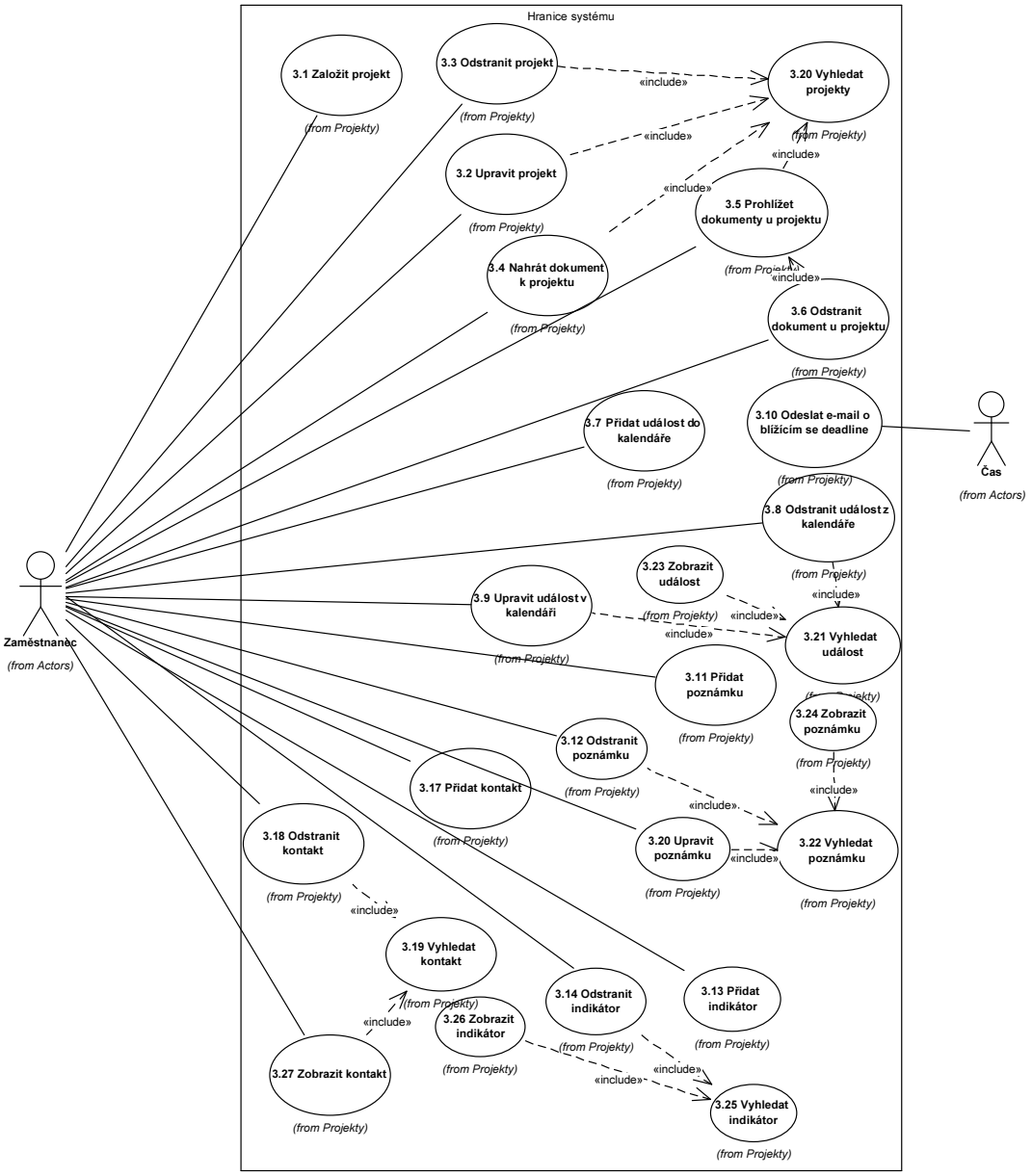
Diagramy případů užití



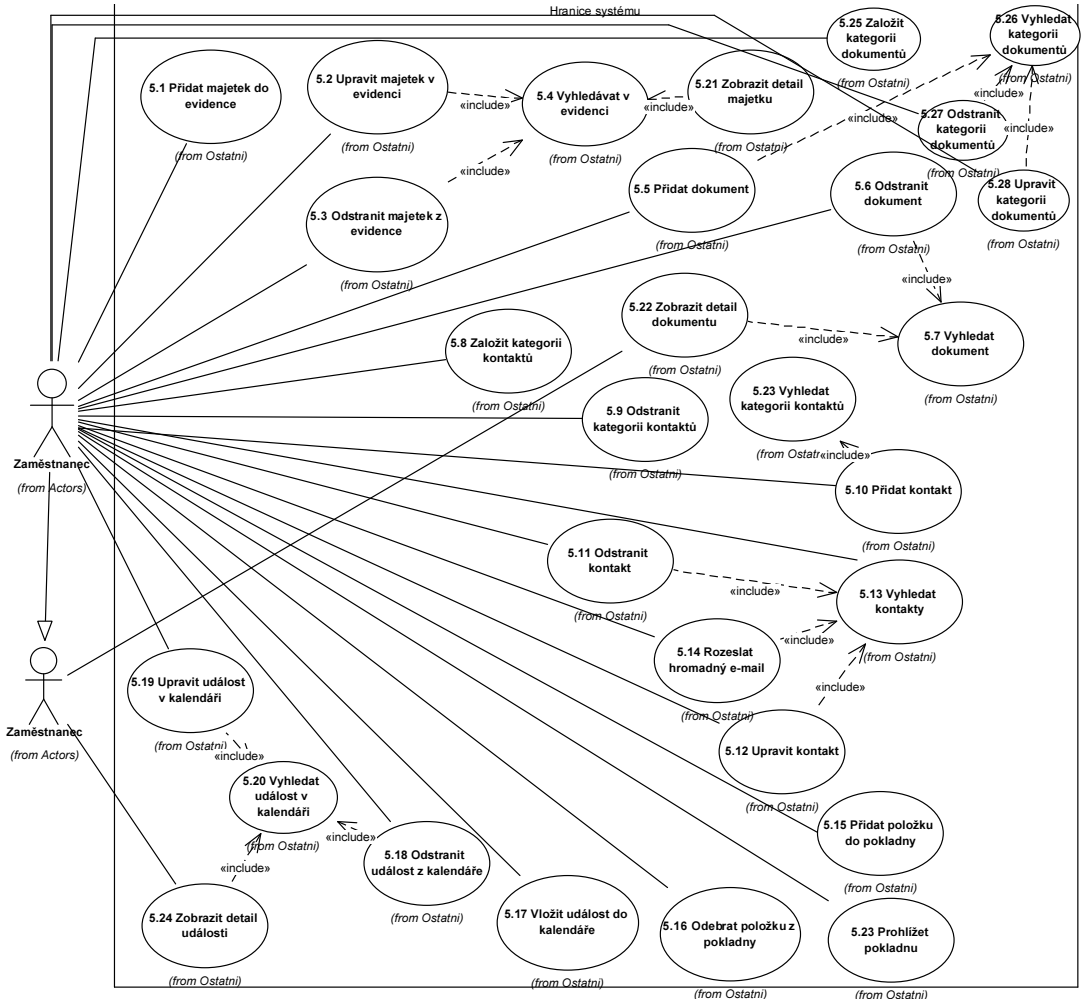
Obrázek D.1: Use case model zaměstnanci č.1



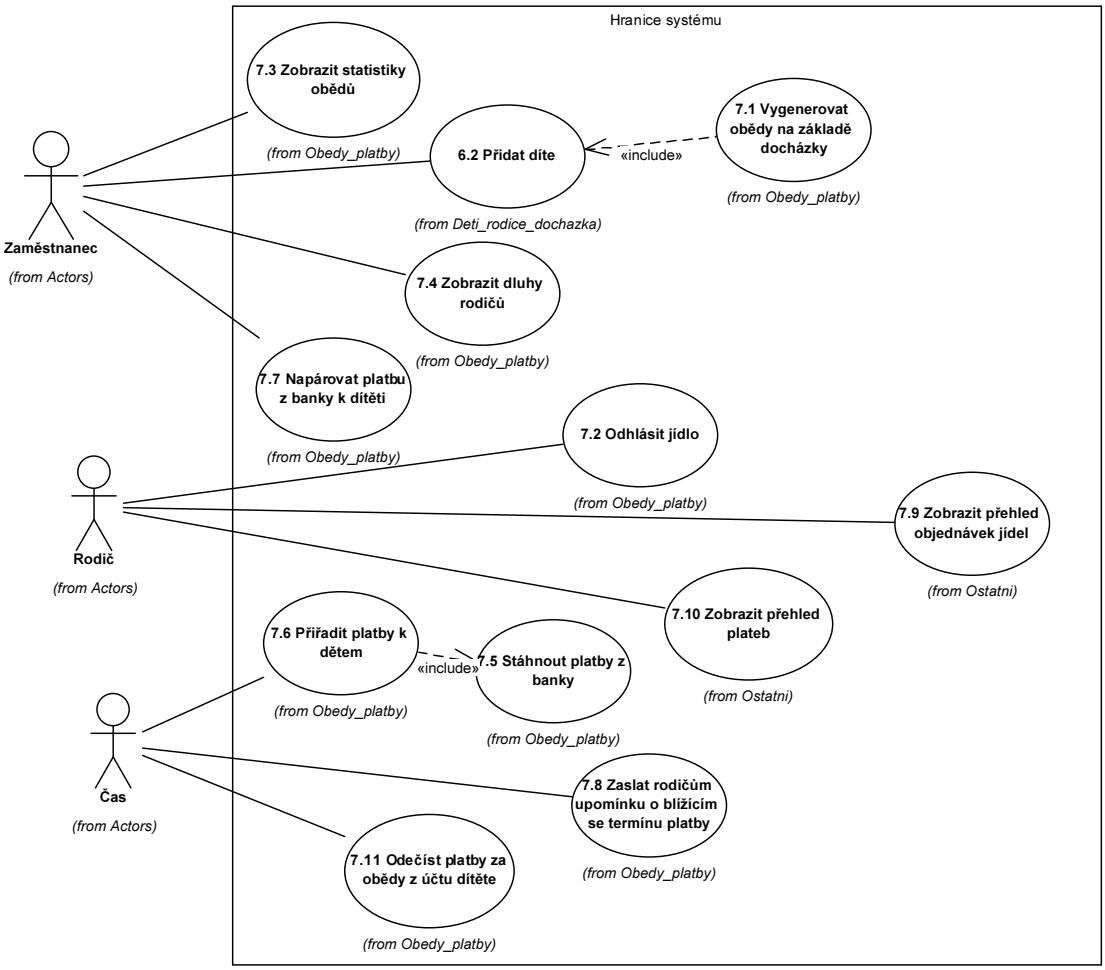
Obrázek D.2: Use case model zaměstnanci č.2



Obrázek D.3: Use case model Projekty

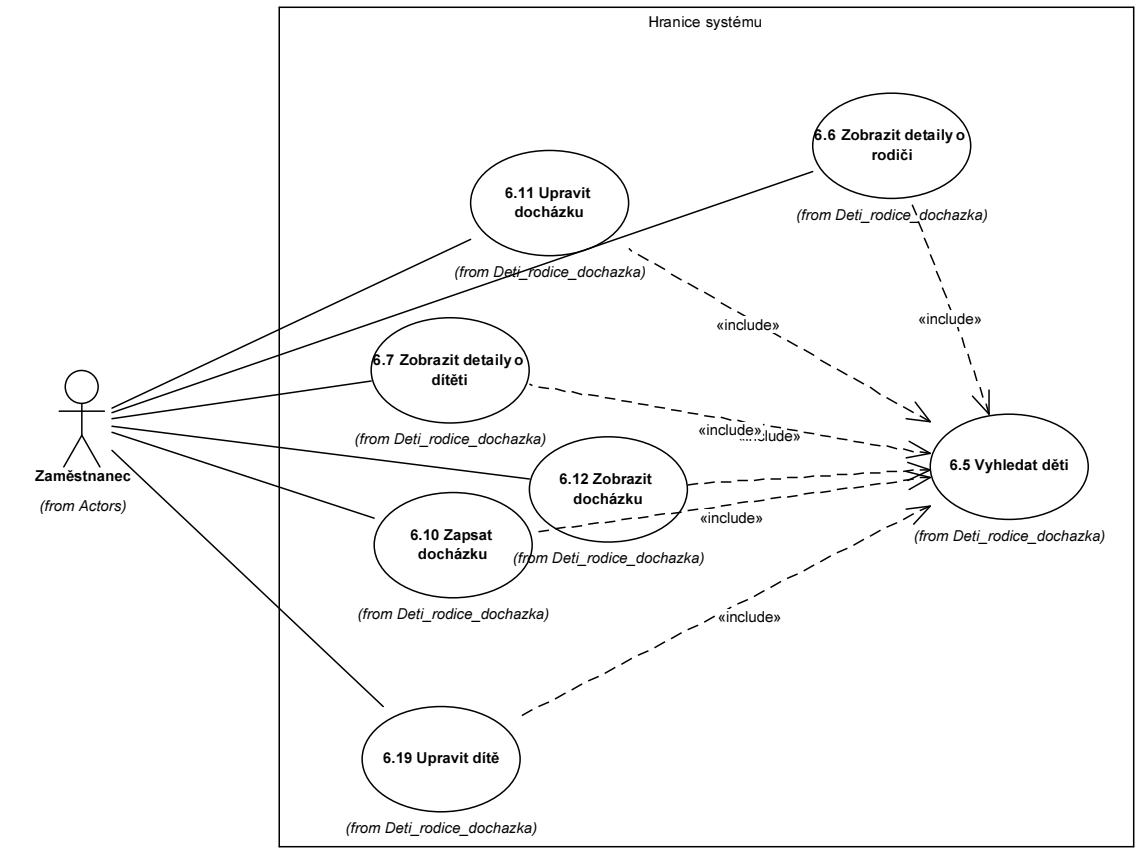


Obrázek D.4: Use case model Provozní záležitosti



Obrázek D.5: Use case model Ekoškolka, část 1.

Obrázek D.6: Use case model Ekoškolka, část 2.



Obrázek D.7: Use case model Ekoškolka, část 3.

Příloha E

Tabulky mapování mezi případy užití a funkčními požadavky

Tabulka E.1: Mapování mezi požadavky a případy use-case v sekci Zaměstnanci

	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	1.10	1.11	1.12	1.13	1.14	1.15	1.16	1.17
2.1	X	X		X								X					
2.2			X														
2.3					X												X
2.4						X											
2.5				X			X	X	X								
2.6										X	X						
2.7													X	X	X	X	

Tabulka E.2: Mapování mezi požadavky a případy use-case v sekci Projekty

	3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9	3.10	3.11	3.12	3.13	3.14	3.17	3.18	3.19	3.20	3.21	3.22	3.23	3.24	3.25	3.26	3.27
3.1	X	X	X															X							
3.2				X	X	X												X							
3.3							X	X	X										X						
3.4										X															
3.5											X	X						X		X		X			
3.6													X	X									X	X	
3.7															X	X	X								X

Tabulka E.3: Mapování mezi požadavky a případy use-case v sekci Provozní záležitosti

	5.1	5.2	5.3	5.4	5.5	5.6	5.7	5.8	5.9	5.10	5.11	5.12	5.13	5.14	5.15	5.16	5.17	5.18	5.19	5.20	5.21	5.22	5.23	5.24	5.25	5.26	5.27	5.28
4.1	X	X	X	X																	X							
4.2					X	X	X															X			X	X	X	X
4.3								X	X	X	X	X	X										X					
4.4														X														
4.5															X	X							X					
4.6																	X	X	X	X				X				

Tabulka E.4: Mapování mezi požadavky a případy use-case v sekci Ekoškolka, část 1.

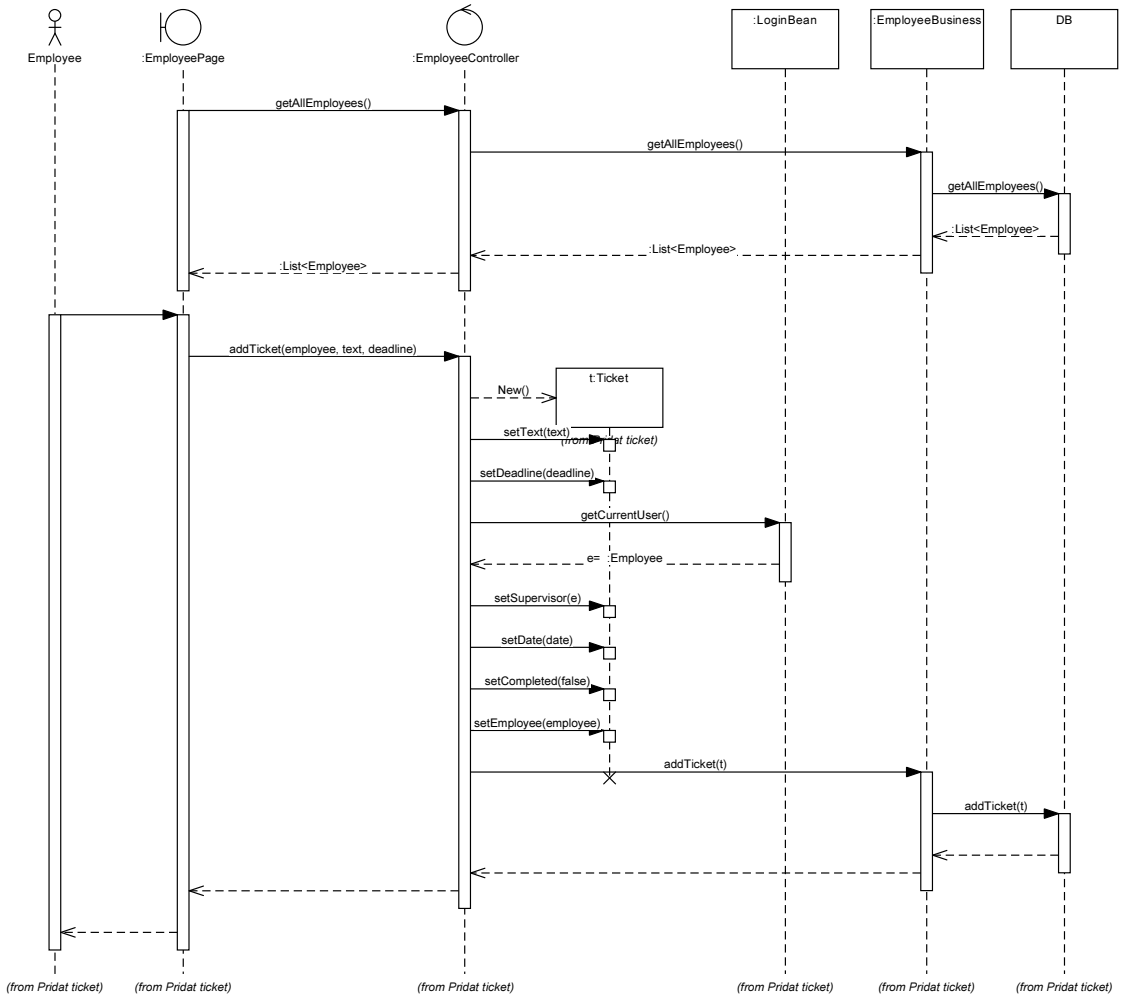
	6.1	6.2	6.3	6.4	6.5	6.6	6.7	6.8	6.9	6.10	6.11	6.12	6.13	6.14	6.15	6.16	6.17	6.19	6.20	6.21
5.1		X			X		X													
5.2							X													
5.3	X		X	X		X		X												
5.4										X	X	X								
5.5																X				
5.6																	X			
5.7													X							
5.8														X						
5.9															X					
5.10									X										X	
5.11																		X		
5.12																		X		X

Tabulka E.5: Mapování mezi požadavky a případy use-case v sekci Ekoškolka, část 2.

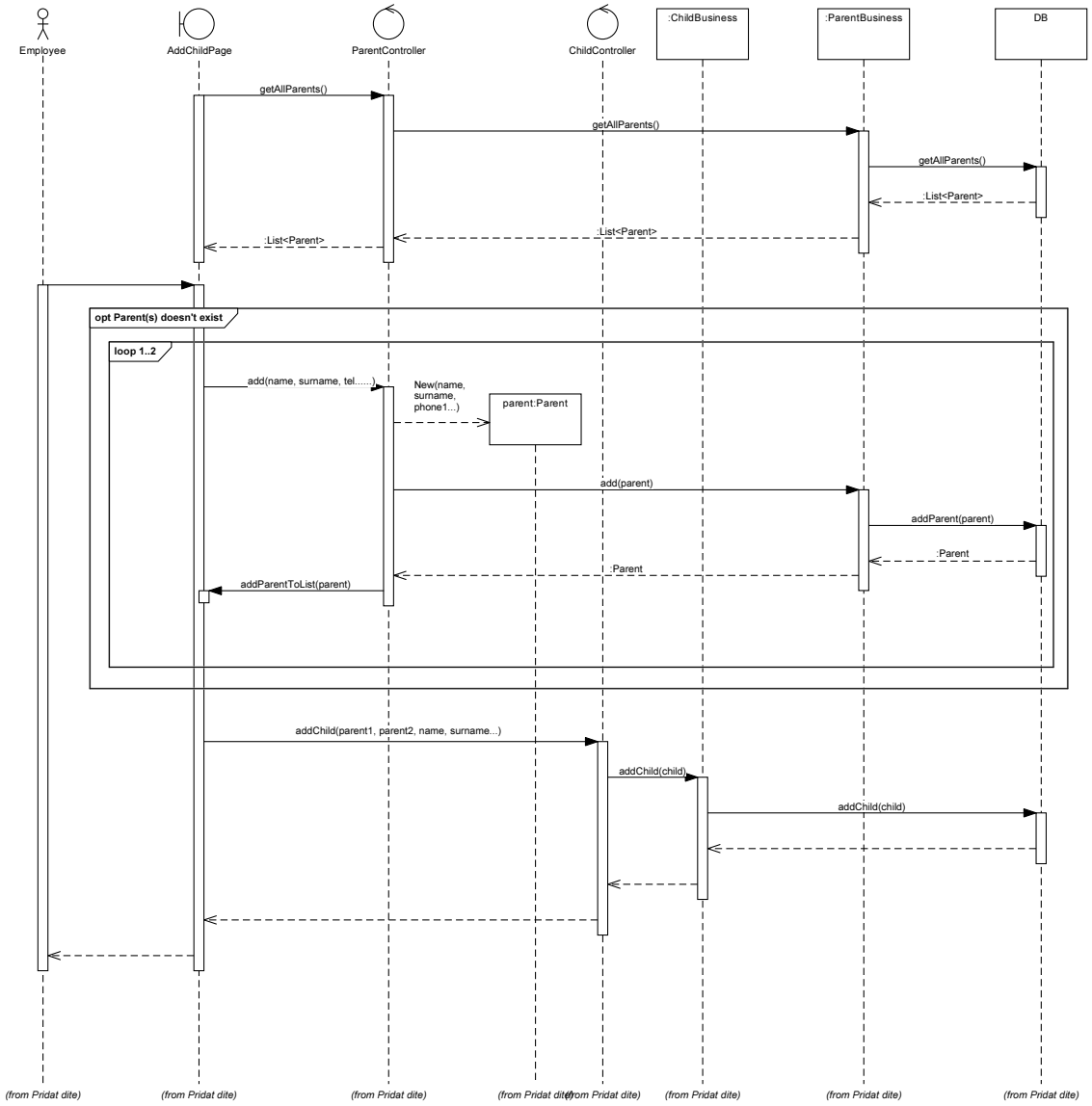
	7.1	7.2	7.3	7.4	7.5	7.6	7.7	7.8	7.9	7.10
5.13				X						
5.14					X	X				
5.15							X			
5.16	X	X								
5.17			X							
5.18								X		
5.19									X	
5.20										X

Příloha F

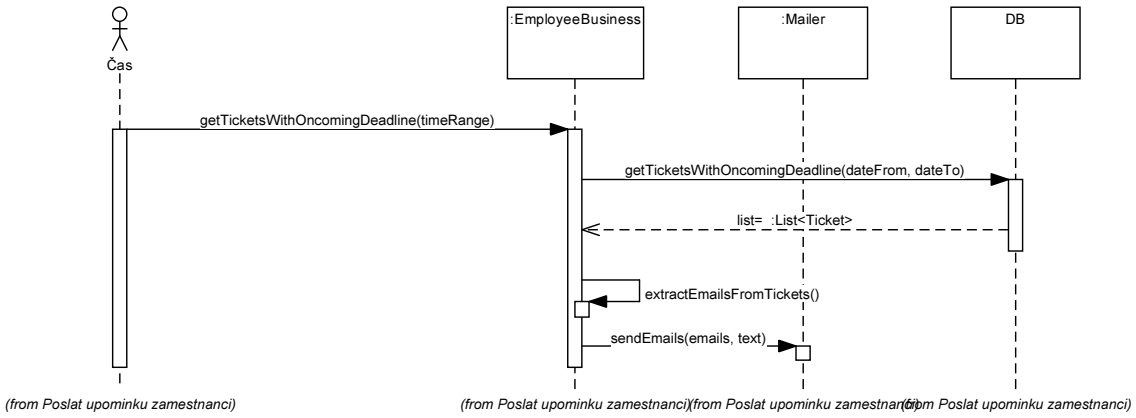
Sekvenční diagramy



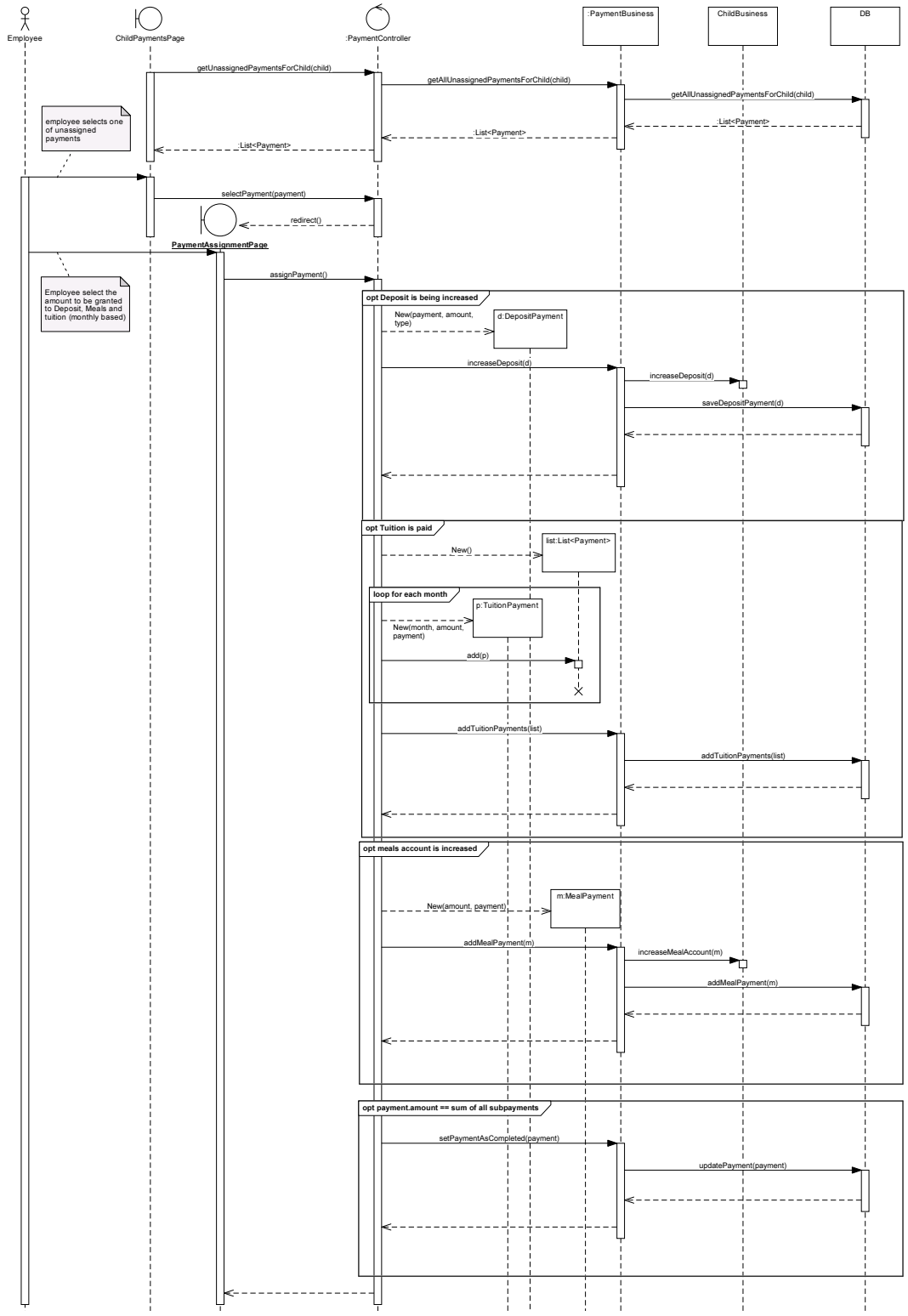
Obrázek F.1: SD přidat úkol zaměstnanci



Obrázek F.2: SD přidat dítě



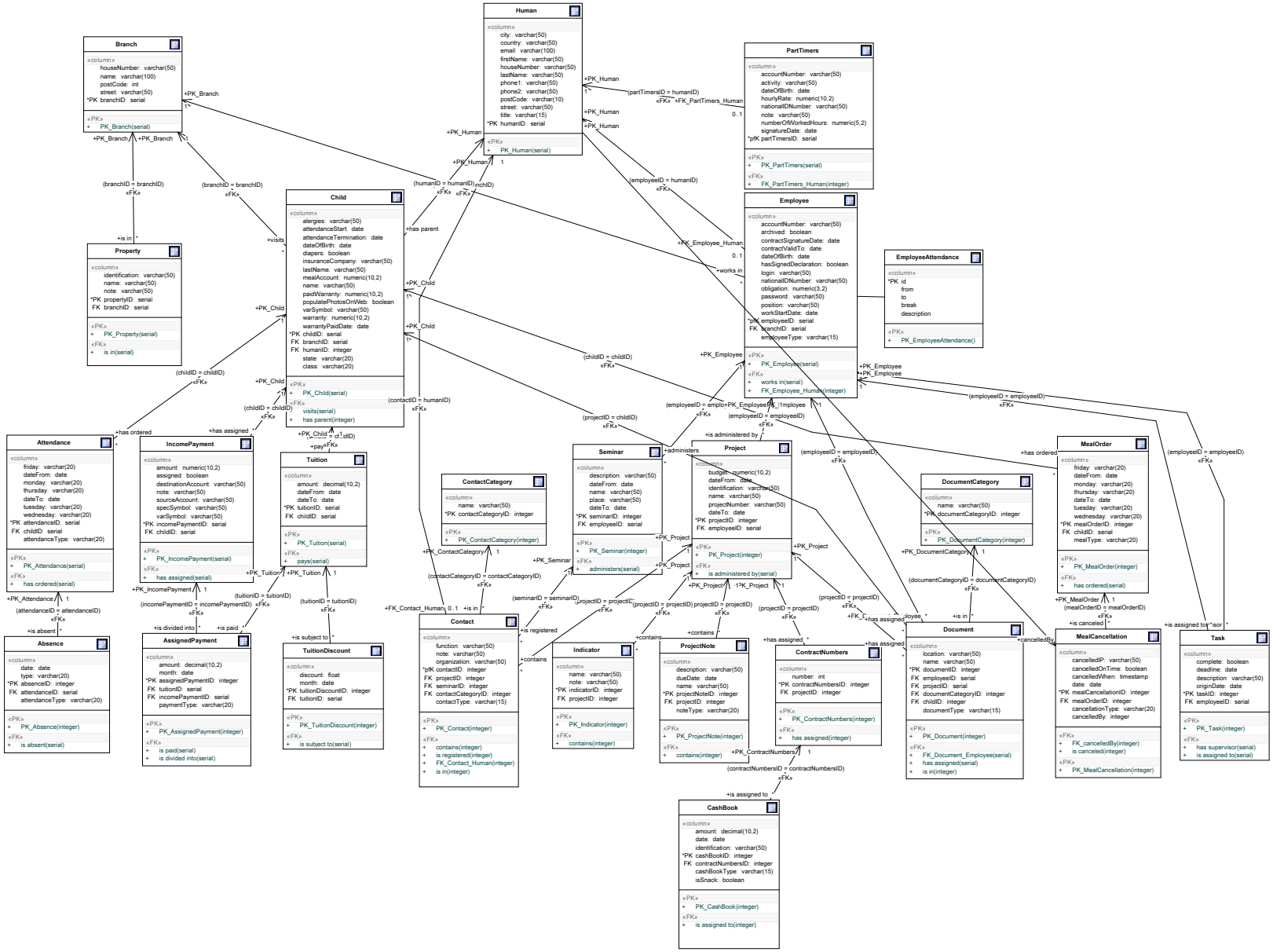
Obrázek F.3: SD poslat upomínku zaměstnanci



Obrázek F.4: SD přiřadit manuálně platbu

Příloha G

Databázový diagram



Obrázek G.1: Databázový diagram

Příloha H

Systémové testování - scénáře

H.1 Zaměstnanci

Založit zaměstnance

- ID: 1
- Vstupní požadavky: Přihlášený uživatel. Zobrazený seznam zaměstnanců.
- Postup:
  1. Kliknout na tlačítko Přidat zaměstnance
  2. Vložit údaje do formuláře.
  3. Kliknout na Odeslat.
- Očekávaný výstup: Systém zobrazí informativní hlášení o úspěchu a zobrazí seznam zaměstnanců včetně právě vloženého.

Upravit zaměstnance

- ID: 2
- Vstupní požadavky: Přihlášený uživatel. Zobrazený seznam zaměstnanců.
- Postup:
  1. Kliknout na tlačítko Změnit v řádku s daným zaměstnancem
  2. Upravit údaje ve formuláři.
  3. Kliknout na Odeslat.
- Očekávaný výstup: Systém zobrazí informativní hlášení o úspěchu a zobrazí seznam zaměstnanců včetně právě upraveného.

Upravit úvazek zaměstnanci

- ID: 3

- Vstupní požadavky: Přihlášený uživatel. Zobrazený seznam zaměstnanců.
- Postup:
  1. Kliknout na tlačítko Přidat zaměstnance
  2. upravit hodnotu v poli úvazek.
  3. Kliknout na Odeslat.
- Očekávaný výstup: Systém zobrazí informativní hlášení o úspšchu a zobrazí seznam zaměstnanců.

Vyhledat zaměstnance

- ID: 4
- Vstupní požadavky: Přihlášený uživatel. Zobrazený seznam zaměstnanců.
- Postup:
  1. Do pole Hledat vložit část příjmení zaměstnance
- Očekávaný výstup: Systém zobrazí řádky se zaměstnanci jejichž příjmení obsahuje zadaný text.

Zobrazit detail zaměstnance

- ID: 5
- Vstupní požadavky: Přihlášený uživatel. Zobrazený seznam zaměstnanců.
- Postup:
  1. Kliknout na jméno zaměstnance
- Očekávaný výstup: Systém zobrazí dialog s informacemi o zaměstnanci.

H.2 Projekty

Založit projekt

- ID: 6
- Vstupní požadavky: Přihlášený uživatel. Zobrazený seznam projektů.
- Postup:
  1. Kliknout na tlačítko Přidat projekt
  2. Vložit údaje do formuláře.
  3. Kliknout na Odeslat.

- Očekávaný výstup: Systém zobrazí informativní hlášení o úspěchu a zobrazí seznam projektů včetně právě vloženého.

Odstranit projekt

- ID: 7
- Vstupní požadavky: Přihlášený uživatel. Zobrazený seznam projektů.
- Postup:
  1. Kliknout na tlačítko Odstranit u vybraného projektu
  2. Potvrdit záměr odstranit projekt
- Očekávaný výstup: Systém odstraní projekt a zobrazí seznam zbývajících projektů.

Přidat indikátor k projektu

- ID: 8
- Vstupní požadavky: Přihlášený uživatel. Zobrazený seznam projektů.
- Postup:
  1. Kliknout na tlačítko Indikátory u vybraného projeku
  2. Kliknout na tlačítko přidat indikátor
  3. Vyplnit název indikátoru
  4. Kliknout na tlačítko Přidat
- Očekávaný výstup: Systém zobrazí informativní hlášení o úspěchu a zobrazí seznam projektů včetně právě vloženého.

Označit indikátor jako splněný

- ID: 9
- Vstupní požadavky: Přihlášený uživatel. Zobrazený seznam projektů.
- Postup:
  1. Kliknout na tlačítko Indikátory u vybraného projeku
  2. Kliknout na tlačítko Hotovo u vybraného indikátoru
- Očekávaný výstup: Systém změní stav u indikátoru na splněno a zobrazí ho uživateli.

H.3 Provozní záležitosti

Přidat majetek do evidence

- ID: 10
- Vstupní požadavky: Přihlášený uživatel. Zobrazený seznam majetku.
- Postup:
  1. Kliknout na tlačítko Přidat majetek
  2. Vyplnit potřebné údaje ve formuláři
  3. Kliknout na tlačítko Odeslat
- Očekávaný výstup: Systém zobrazí seznam majetku i s nově přidaným kusem.

Přidat kontakt

- ID: 11
- Vstupní požadavky: Přihlášený uživatel. Zobrazený seznam kontaktů.
- Postup:
  1. Kliknout na tlačítko Přidat kontakt
  2. Vyplnit potřebné údaje ve formuláři
  3. Kliknout na tlačítko Odeslat
- Očekávaný výstup: Systém zobrazí seznam kontaktů i s nově přidaným kontaktem.

Přidat kategorii kontaktů při přidávání kontaktu

- ID: 12
- Vstupní požadavky: Přihlášený uživatel. Zobrazený seznam kontaktů.
- Postup:
  1. Kliknout na tlačítko Přidat kontakt
  2. Vyplnit potřebné údaje ve formuláři
  3. Kliknout na tlačítko Přidat u položky kategorie
  4. Vyplnit název kategorie
  5. Kliknout na tlačítko Odeslat
- Očekávaný výstup: Systém zobrazí v rolovacím menu Kategorie i nově přidanou kategorii.

Vyhledat kontakty podle kategorie

- ID: 13
- Vstupní požadavky: Přihlášený uživatel. Zobrazený seznam kontaktů.
- Postup:
  1. Z rolovacího menu Kategorie vybrat požadovanou kategorii
- Očekávaný výstup: Systém zobrazí kontakty ze zadané kategorie.

Přidat položku do pokladny

- ID: 14
- Vstupní požadavky: Přihlášený uživatel. Zobrazena pokladna.
- Postup:
  1. Kliknout na tlačítko Přidat platbu
  2. Vyplnit požadované údaje
  3. Kliknout na odeslat
- Očekávaný výstup: Systém zobrazí zadané platby včetně nově přidané.

Přidat akci do kalendáře

- ID: 15
- Vstupní požadavky: Přihlášený uživatel. Zobrazený kalendář.
- Postup:
  1. Kliknout na tlačítko Přidat událost
  2. Vyplnit potřebné údaje ve formuláři
  3. Kliknout na tlačítko Přidat
- Očekávaný výstup: Systém zobrazí kalendář s nově přidanou událostí.

Zobrazit detail události

- ID: 16
- Vstupní požadavky: Přihlášený uživatel. Zobrazený kalendář.
- Postup:
  1. Kliknout na akci v kalendáři.
- Očekávaný výstup: Systém zobrazí dialogové okno s detaily o vybrané akci.

Přidat akci do kalendáře

- ID: 15
- Vstupní požadavky: Přihlášený uživatel. Zobrazený kalendář.
- Postup:
  1. Kliknout na tlačítko Přidat událost
  2. Vyplnit potřebné údaje ve formuláři
  3. Kliknout na tlačítko Přidat
- Očekávaný výstup: Systém zobrazí kalendář s nově přidanou událostí.

H.4 Ekoškolka

Vyhledat dítě a zobrazit informace

- ID: 16
- Vstupní požadavky: Přihlášený uživatel. Zobrazený seznam dětí.
- Postup:
  1. Do políčka Hledat zadat jméno dítěte
  2. Kliknout na jméno vyhledaného dítěte
- Očekávaný výstup: Systém zobrazí dialog s informacemi o dítěti.

Přidat dítě

- ID: 17
- Vstupní požadavky: Přihlášený uživatel. Zobrazený seznam dětí.
- Postup:
  1. Kliknout na tlačítko Přidat dítě.
  2. Vyplnit potřebné údaje ve formuláři
  3. Kliknout na tlačítko Přidat
- Očekávaný výstup: Systém zobrazí seznam dětí s nově přidaným dítětem.

Přidat docházku dítěti

- ID: 18
- Vstupní požadavky: Přihlášený uživatel. Zobrazená sekce docházky.
- Postup:
  1. Z rolovacího menu vybrat dítě

2. Kliknout na tlačítko Přidat docházku
3. Vyplnit potřebné údaje
4. Kliknout na tlačítko Přidat

- Očekávaný výstup: Systém zobrazí seznam docházek dítěte včetně nově přidané.

Zobrazit docházku dítěte

- ID: 19
- Vstupní požadavky: Přihlášený uživatel. Zobrazená sekce docházky.
- Postup:
  1. Z rolovacího menu vybrat dítě
  2. Kliknout na tlačítko Zobrazit docházku graficky
- Očekávaný výstup: Systém zobrazí grafické znázornění docházky dítěte.

Zapsat absenci dítěte

- ID: 20
- Vstupní požadavky: Přihlášený uživatel. Zobrazená sekce docházky.
- Postup:
  1. Z rolovacího menu vybrat dítě
  2. Kliknout do pole Odhlásit den
  3. Z pop-up kalendáře vybrat den
  4. Z rolovacího menu vybrat absentovanou část dne
  5. Kliknout na odhlásit
- Očekávaný výstup: Systém přidá absenci a zobrazí ji v grafickém zobrazení docházky.

Přidat rodiče při přidávání dítěte

- ID: 21
- Vstupní požadavky: Přihlášený uživatel. Zobrazená sekce děti.
- Postup:
  1. Kliknout na tlačítko Přidat dítě
  2. U rolovacího seznamu Rodič kliknout na tlačítko Přidat
  3. Vyplnit potřebné údaje
  4. Kliknout na tlačítko Odeslat
- Očekávaný výstup: Systém přidá rodiče a zobrazí ho v rolovacím seznamu.

Archivovat dítě

- ID: 22
- Vstupní požadavky: Přihlášený uživatel. Zobrazená sekce děti.
- Postup:
  1. Kliknout na tlačítko Odstranit u vybraného dítěte
  2. Potvrdit odstranění dítěte
- Očekávaný výstup: Systém archivuje dítě a odstraní ho ze zobrazeného seznamu aktivních dětí.

Napárovat platbu z banky k dítěti

- ID: 23
- Vstupní požadavky: Přihlášený uživatel. Zobrazená sekce Nepřiřazené platby.
- Postup:
  1. Kliknout na tlačítko Párovat u vybrané platby
  2. Z rolovacího menu vybrat dítě
  3. Ve formuláři rozložit platby dle potřeby (až do vyčerpání celé částky)
  4. Kliknout na Odeslat
- Očekávaný výstup: Systém přiřadí platby dítěti a zobrazí informaci o úspěšné akci.

Zobrazit přehled objednávek jídel

- ID: 24
- Vstupní požadavky: Přihlášený uživatel. Zobrazená sekce Jídla dětí.
- Postup:
  1. Z rolovacího menu vybrat dítě
  2. Kliknout na tlačítko Zobrazit jídla graficky
- Očekávaný výstup: Systém zobrazí grafické znázornění objednávky jídel v tabulce a zároveň zobrazí souhrnné informace o objednávkách jídel daného dítěte.

Odhlásit jídlo

- ID: 25
- Vstupní požadavky: Přihlášený uživatel. Zobrazená sekce Jídla dětí.
- Postup:

1. Z rolovacího menu vybrat dítě
2. Kliknout na tlačítko Zobrazit jídla graficky
3. V grafickém znázornění objednávek vybrat požadovaný měsíc
4. Kliknout na tlačítko 'O'

- Očekávaný výstup: Systém odhlásí oběd a změní grafické znázornění objednávky.

Zobrazit přehled plateb k dítětí

- ID: 26
- Vstupní požadavky: Přihlášený uživatel. Zobrazená sekce Platby dětí.
- Postup:
  1. Z rolovacího menu vybrat dítě
- Očekávaný výstup: Systém zobrazí seznam přiřazených plateb k dítěti.

Příloha I

Tabulka testů uživatelského rozhraní

Tabulka I.1: Tabulka testů uživatelského rozhraní

Název	**Problémy**	**Opraveno**	**Stav**
Vytvořit nového zaměstnance, změnit mu heslo a poté smazat	Žádné		OK
Přidat majetek do evidence, smazat majetek z evidence	Tabulka majektu nebyla automaticky obnovena	ANO	OK
Vyhledat majetek v evidenci podle názvu	Žádné		OK
Přidat příjem a výdej do pokladny	Žádné		OK
Zjistit aktuální stav pokladny	Žádné		OK
Zobrazit docházku zaměstnance (jiného než přihlášený uživatel) a přidat záznam	Žádné		OK
Přidat projekt, k projektu přidat zakázku a indikátor. Indikátor označit jako splněný Zobrazit deprojektu	Žádné Žádné		OK OK
Změnit adresu pobočky	Žádné		OK
Přidat dítě, zároveň přidat nového rodiče	Při přidávání dítěte nebyly označeny povinné položky a formulář nešel odeslat	ANO	OK
Odstranit dítě	Žádné		OK
Zobrazit nepřiřazené platby a napárovat platbu k dítěti	Žádné		OK
Zobrazit přiřazené platby k dítěti	Žádné		OK
Přidat dokument do nové kategorie dokumentů	Žádné		OK
Přidat jednodenní a vícedenní akci do kalendáře	Žádné		OK
Přidat kontakt do databáze kontaktů	Nebyly označeny povinné položky	ANO	OK
Zobrazit docházku dítěte	Žádné		OK
Přidat docházku dítěte	Po přidání docházky nebyla obnovena data v tabulce	ANO	OK
Přidat absenci dítěte	Po přidání absence nebyla tato změna obnovena v tabulce	ANO	OK
Zobrazit objednávky jídel dítěte	Žádné		OK
Přidat objednávku jídel	Žádné		OK
Odhlásit jídlo, přidat extra jídlo	Žádné		OK
Zobrazit informaci o ceně objednaných jídel	Žádné		OK
Zobrazit svou docházku, přidat záznam	Žádné		OK