FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Párování plateb a výpočet dlužné částky v UMŠ Lvíčata
Student:	Jakub Lacný
Vedoucí:	Ing. Jiří Smítka
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2019/20

Pokyny pro vypracování

Navrhněte a implementujte webovou aplikaci pro párování plateb za jednotlivé žáky a děti Univerzitní základní a mateřské školy Lvíčata.
Umožněte výpočet dlužné částky za školné, stravné a případně za další nabízené služby.
Program by měl párovat platby se jménem žáka po nahrání bankovního výpisu na základě variabilního symbolu.
Uživatel programu by měl mít možnost zobrazit, vytisknout a přesunout do tabulky v programu Excel různé relevantní sestavy.
Předpokládá se práce více uživatelů najednou. Vyřešte problém autentizace a autorizace.
Aplikaci otestujte v prostředí UMŠ Lvíčata.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 14. března 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Párování plateb a výpočet dlužné částky v UMŠ Lvíčata

Jakub Lacný

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Smítka

10. května 2018

Poděkování

Chtěl bych poděkovat panu Ing. Jiřímu Smítkovi za vedení mé bakalářské práce a paní Lucii Mauerové z UMŠ Lvíčata za trpělivost a ochotu. Velké poděkování patří mým rodičům a přátelům, kteří mě během celého studia podporovali.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 10. května 2018 ……

Odkaz na tuto práci

Lacný, Jakub. *Párování plateb a výpočet dlužné částky v UMŠ Lvíčata.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Cílem práce je zanalyzovat potřeby Univerzitní mateřské a základní školy Lvíčata a na jejich základě vytvořit webovou aplikaci určenou k evidenci plateb a jejich párování s jednotlivými žáky. Výsledkem je aplikace, která zautomatizuje aktuálně ručně prováděné administrativní úkony. Program je schopen číst bankovní výpisy formátu ABO a také vytvářet různé sestavy a exportovat je do formátu určeného pro běžné tabulkové editory (například Microsoft Excel). K implementaci je použit PHP framework Symfony s balíkem Sonata a dále knihovny pro práci s bankovními výpisy a tabulkami. Aplikace byla otestována přímo v prostředí UMŠ Lvíčata.

Klíčová slova Informační systém, UMŠ Lvíčata, Párování plateb, Webová aplikace, Bankovní výpisy, Evidence plateb

Abstract

The purpose of this thesis is to analyze the needs of UMŠ Lvíčata in order to implement a web application which significantly improves the work related to students' records, and with pairing the payments of the students. The main result of this thesis is a fully-functional application automatizing several administrative operations which are currently all done by hand. The application is capable of importing bank statements in ABO format, creating various spreadsheets, and exporting them into a common spreadsheets editor (such as Microsoft Excel). The application is implemented in PHP using the Symfony framework with the help of Sonata Admin Bundle, and bundles for ABO files and spreadsheets. The application was tested directly in UMŠ Lvíčata.

Keywords Information System, UMŠ Lvíčata, Payment pairing, Web application, Bank statements, Payment records

Obsah

Seznam obrázků

Seznam tabulek

Seznam ukázek kódu

Úvod

Seznámení s problematikou

Každé vzdělávací zařízení, ať už mateřská, základní, střední nebo i vysoká škola musí evidovat své studenty a jejich útraty za stravné, školné a případně jiné záležitosti. Pokud se jedná o zařízení s menším počtem žáků, není většinou problém vést evidenci ručně v libovolném nástroji jako například Microsoft Excel nebo podobnén tabulkovém editoru. S přibývajícími žáky se však tato evidence podstatně zvětšuje a je potřeba najít vhodné a udržitelné řešení.

S procesem párování plateb se setkáváme ve všech sférách, ve kterých dochází k finančním transakcím. Vždy máme službu, která je někomu účtována, a kód, podle kterého tyto platby párujeme. U finančního úřadu je to daňové identifikační číslo (DIČ), v našem případě to bude pro primární typ plateb, tedy stravné ve školní jídelně, rodné číslo žáka. Pro další typy plateb (například za školné) budou použity jiné, vhodně zvolené kódy.

Aktuální stav v UMŠ Lvíčata

V univerzitní mateřské a základní škole Lvíčata probíhá tato evidence stále ručně, ale nachází se aktuálně již na hranici, kdy citelně ubírá čas pro ostatní administrativní úkony. Evidenci vykonávají aktuálně pracovníci školy ručně za pomocí tabulkového editoru. Jedná se o celkově tři tabulky (jedna fyzická, dvě v tabulkovém editoru), mezi kterými probíhá párování (princip je rozebrán v rešeršní části).

Jelikož se v UMŠ Lvíčata plánuje rozšíření působnosti a s tím spojený nárůst žáků, je nutné najít vhodné řešení pro tuto evidenci. Vyřešit to informačním systémem se přímo vybízí, protože se jedná o poměrně jednoduchou operaci, kterou může rychle provést právě počítač.

Zvolené řešení

V UMŠ Lvíčata se rozhodli pro řešení této problematiky vlastním systémem přímo určeným pro jejich potřebu. Chtějí tedy aplikaci, která dokáže evidovat studenty, jejich účty, a k nim dané pohledávky a platby. Aplikace má umožňovat vkládání plateb ze souborů typu ABO (přípona .gpc), což je v České Republice a na Slovensku nejpoužívanější sjednocený typ souboru, ze kterého se importují data z bankovních účtů do různých účetních systémů a podobně.

Je také požadováno, aby se z programu daly různé seznamy žáků, plateb apod. vyexportovat ve formátu vhodném pro otevření v tabulkovém editoru (MS Excel).

Cíl práce

Cílem práce je zanalyzovat principy evidence plateb v UMŠ Lvíčata a na základě získaných informací vytvořit aplikaci, která pracovníkům zjednoduší evidenci plateb za stravné, školné a případně i jiné služby. Osobním cílem autora je pak zlepšit se v navrhování a implementaci webových aplikací a následně tyto dovednosti převést i mimo školu.

Struktura práce

V první kapitole autor popisuje stav evidence plateb za stravné a školné v UMŠ Lvíčata a důvody, proč potřebují nový systém. Dále jsou v ní zanalyzovány různá dostupná řešení a jejich zhodnocení v rámci daných požadavků. Obsahuje také analýzu datového formátu ABO, který je v aplikaci využíván.

Ve druhé kapitole jsou zanalyzovány požadavky pracovníků školy a sepsány ve funkčních a nefunkčních požadavcích. Jsou zde také sepsána možná architektonická řešení, důležité případy užití a návrh doménového modelu.

Ve třetí kapitole autor rozebírá a následně zhodnocuje dostupné technologie vhodné pro implementaci zadaného systému.

Čtvrtá kapitola se zaobírá samotnou implementací v technologiích zvolených ve třetí kapitole. Je zde popsáno databázové schéma, kostra aplikace, záležitosti ohledně zabezpečení a splnění funkčních požadavků.

Pátá kapitola obsahuje výsledky uživatelského testování a jeho zhodnocení.

Kapitola 1

Rešeršní část

1.1 Vymezení základních pojmů

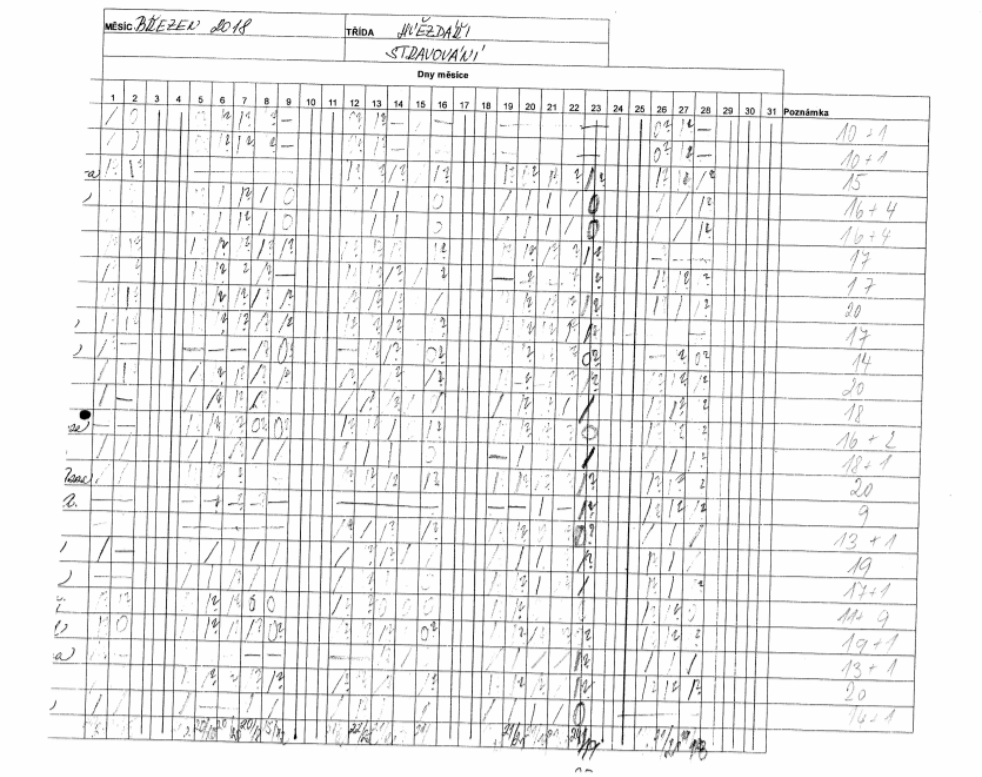
Následují některé pojmy používané v této práci a také ve výsledném systému, které se vyskytují velice často.

- **Zařazení** – skupina, do které může být zařazen žák. Například se může jednat o třídu základní školy.
- **Skupina** – skupinou se rozumí skupina pohledávek (např. skupina pohledávek obsahující všechny pohledávky za stravné za březen).
- **Výpis** – Výpis je seskupení plateb k určitému datu. Když například nahrajeme ABO soubor, všechny platby v něm obsažené budou patřit pod jeden výpis.
- **Škola** – školou je vždy myšlena UMŠ Lvíčata (Univerzitní mateřská a základní škola Lvíčata).
- **Tabulkový editor** – tabulkovým editorem rozumíme program určený k vytváření a editování tabulek. Jedná se například o programy Microsoft Excel nebo OpenOffice Calc.

1.2 Aktuální stav v UMŠ Lvíčata

Aktuálně musí pracovník ručně zadávat platby z bankovního výpisu k příslušnému žákovi v tabulkovém editoru. Je to řešení, které je jistě velmi používané, je však velice neefektivní. Moderní tabulkové editory sice umožňují používání mnoha pomocných funkcí, které tyto operace umí zjednodušit, ale vyloženě automatické nahrání výpisu z účtu a propárování plateb v něm obsažených neumožňuje.

Základem výpočtu dlužné částky je pro školu evidenční tabulka, která se ručně vyplňuje podle žákova stravování v daný den. Sloupce tabulky představují dny v měsící a řádky představují jednotlivé žáky. Do samotných buněk tabulky se vyplňují znaky podle účtované částky (za půldenní nebo celodenní stravné). Na konci měsíce se jednotlivé sazby v tabulce sečtou do separátního sloupce a mohou být přesunuty do další tabulky. Ukázka této tabulky se nachází na obrázku 1.1.



Obrázek 1.1: Tabulka na evidenci stravování

Výsledný sloupec z první tabulky (obrázek 1.1) je poté vstupem do druhé tabulky, která se již nachází v tabulkovém editoru na počítači. Zadávají se do ní počty celodenních a půldenních stravování a editor z těchto údajů vypočítá výslednou útratu. Tu poté porovná se zaplacenou zálohou a informuje v separátních sloupcích o přeplatku či nedoplatku. Příklad této tabulky se nachází na obrázku 1.2.

Vyúčtování stravného za měsíc leden 2018 - Hvězdáři

Jméno a příjmení	Půldenní menu	Celodenní menu	Skutečně proje	Zaplatil	Vratka	Doplatek	Předpis	Celkem oběd.
	45	55						
		13	715	1000	285		715	13
		13	715	1000	285		715	13
		16	880	1000	120		880	16
	3	18	1125	1000	0	125	1125	21
	3	17	1070	1000	0	70	1070	20
	1	20	1145	1000	0	145	1145	21
		9	495	1000	505		495	9
		21	1155	1000	0	155	1155	21
		21	1155	1000	0	155	1155	21
		13	715	1000	285		715	13
		20	1100	1000	0	100	1100	20
		20	1100	1000	0	100	1100	20
	1	20	1145	1000	0	145	1145	21
	1	15	870	1000	130		870	16
	2	17	1025	1000	0	25	1025	19
		11	605	605	0		605	11
	2	19	1135	1000	0	135	1135	21
		14	770	1000	230		770	14
		12	660	1000	340		660	12
	4	14	950	1000	50		950	18
		21	1155	1000	0	155	1155	21
	3	14	905	1000	95		905	17
		20	1100	1000	0	100	1100	20
		20	1100	1000	0	100	1100	20
Celkem	20	398	22790	23605	2325	1510	22790	418
	900	21890	22790		2325			
			0	0	815			

Obrázek 1.2: Tabulka s výpočtem dlužné částky

Celková částka za stravné pro jednotlivé žáky se poté vkládá do poslední tabulky, ve které probíhá párování s platbami na bankovním účtu. Páruje se na základě variabilního symbolu, který je přidělený každému žákovi. Výsledkem jsou informace o nutnosti přeúčtování přeplatků do dalších období nebo o nutnosti informovat rodiče o nedoplatku. Příklad této tabulky se nachází na obrázku 1.3.

Platba stravné –březen 2018

Lvíčata

Stravné

Jméno dítěte	Uhrazeno Kč	Datum	Hrazeno v jiný/z	Předpis	Variabilní symbol	Poznámka	
	1 000 Kč	12.3.18		1000	112160		0
	1 000 Kč	7.3.18		1000	112125		0
	1 000 Kč	7.3.18		1000	112124		0
	1 000 Kč	9.3.18		1000	112137		0
				1000	112187		-1000
				1000	112140		-1000
				1000	112139		-1000
	1 000 Kč	5.3.18		1000	112104		0
				1000	112165		-1000
				1000	112129		-1000
	1 000 Kč	6.3.18		1000	112169		-1000
				1000	112176		-1000
	1 000 Kč	9.3.18		1000	112180		0
	1 000 Kč	12.3.2018		1000	113128		-1000
	1 000 Kč	8.3.18		1000	112097		0
				1000	112194		-1000
	1 000 Kč	2.3.18		1000	112123		0
	1 000 Kč	7.3.18		1000	112159		0
	1 000 Kč	1.3.18		1000	112192		0

Obrázek 1.3: Tabulka sloužící k párování plateb

1.3 Aktuální stav řešení problematiky

Evidence plateb spojená s párováním plateb není tématem, které by v informatice nikdy nebylo rozebíráno. Existuje mnoho služeb, které určitou formu párování poskytují. Jedná se především o účetní systémy, které párují faktury s platbami na účtě podle variabilního symbolu (většinou číslo faktury) a komplexní řešení určené pro chod celého stravovacího zařízení (v našem případě školní jídelna). U těch je potřeba pořídit celý systém a to s sebou nese nemalé pořizovací náklady. V následující části autor představuje vybrané nástroje.

1.3.1 Fakturoid[1]

Jedná se o jednoduchou webovou aplikaci (k dispozici je i mobilní aplikace na Android a iOS) umožňující vydávání faktur a jejich evidenci.

Dle [1] v placené verzi umožňuje také například vyplnit daňové přiznání, ale především se umí napojit na bankovní účet a párovat vydané faktury podle variabilního symbolu u platby (číslo faktury). Aktuálně podporuje téměř všechny české banky (s vyjímkou mBank a České Spořitelny [1]).

Nejedná se však o typ systému, který by byl vhodný pro použití v rámci školy nebo například stravovacího zařízení.

1.3.2 Jídelna[2]

Jedná se o aplikaci vytvořenou společností Z-WARE[3] přímo za účelem evidence provozu různých typů jídelen. Umožňuje evidenci strávníků a propojení také s terminály sloužícími k obědnávání a placení obědů na základě čipů.

Tento systém dokáže zpracovávat ABO výpisy z bankovních účtů a propojovat z něj přijaté částky s reálnými útratami strávníků. Umí také rozlišovat sazby za různé typy stravování [2].

1.3.3 Stravné[4]

Podobnou aplikací přímo určenou pro provoz ve školní jídelně je program Stravné od společnosti s ručením omezeným Veřejná informační služba[5].

Tento program v základní konfiguraci párování plateb s bankovním účtem neumožňuje. Dle [3] se tato funkcionalita však nachází v modulu „Banka“ a Umožňuje načítání souboru s výpisem, formát však není specifikován. Následně také dokáže párovat platby na základě variabilních symbolů.

[1] Dostupné z: `https://www.fakturoid.cz`

[2] Dostupné z: `https://www.z-ware.cz/jidelna`

[3] Dostupné z: `https://www.z-ware.cz`

[4] Dostupné z: `http://web.visplzen.cz/produkty/stravovaci-systemy/program-stravne/`

[5] Dostupné z: `http://web.visplzen.cz`

1.4 Výpisy ve formátu ABO[6]

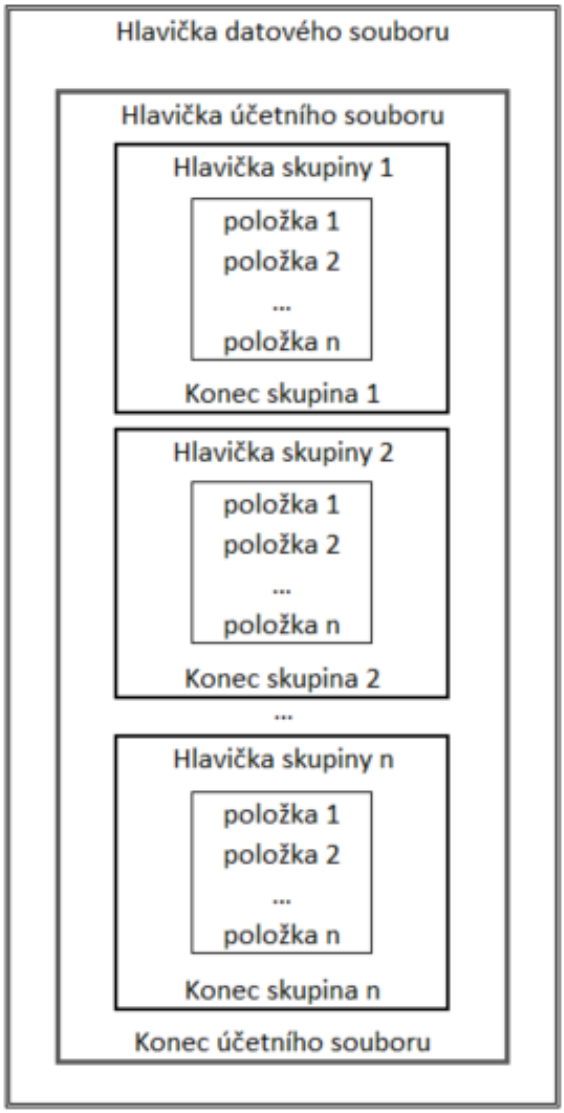
„Formát ABO se v České republice a na Slovensku běžně používá pro výměnu finančních zpráv. Jeho struktura je pevně definována, a to podle dále uvedeného přehledu.“[4] Přestože se jedná o standardní souborový formát, u některých bank dochází k určitým odlišnostem. Každá banka však poskytuje technický popis těchto souborů, podle kterého lze přizpůsobit program pro čtení těchto odlišných verzí souboru. Nejběžnějšími příponami těchto souborů jsou `.gpc`, `.kpc` nebo `.abo`.

Jedná se o textový soubor s pevnou strukturou. Specifikace umožňuje využití variabilní délky některých sloupců, všechny banky toho však nevyužívají a místo toho přidávají nuly (variabilní symbol `2563` je tedy v souboru uveden jako `2563000000`). Graficky znázorněná struktura celého souboru je znázorněna na obrázku 1.4. V tabulce 1.1 je pak příklad specifikace bloku s platbou.

Tabulka 1.1: Struktura řádku s platbou v ABO souboru

Poř. Č.	Název	Fixní	Min. délka	Max. deélka
1	Číslo účtu debet	ne	2	17
2	Separátor pole	ano	-	1
3	Číslo účtu kredit	ne	2	17
4	Separátor pole	ano	-	1
5	Částka	ne	1	12
6	Separátor pole	ano	-	1
7	Variabilní symbol	ne	1	10
8	Separátor pole	ano	-	1
9	Konstantní symbol	ne	8	10
10	Separátor pole	ano	-	1
11	Specifický symbol	ne	0	10
12	Separátor pole	ano	-	1
13	Zpráva pro příjemce	ne	0	35
	Koncový znak zprávy	ano		2

[6] Automatické bankovní operace



Obrázek 1.4: Struktura souboru ABO

Kapitola 2

Analytická a návrhová část

2.1 Možná architektonická řešení

Nabízí se několik modelů, které by bylo možné na tuto aplikaci použít. V této kapitole autor rozebírá tři možné modely vhodné pro implementaci a vybírá nejvhodnější model k implementaci požadované aplikace.

2.1.1 Aplikace běžící na jednom osobním počítači

Jednalo by se o aplikaci, která se nainstaluje na PC[7] a program bude používán pouze z tohoto jednoho počítače. Výhodami by byly jednoduchost nasazení a nevyžadování internetového připojení (či alespoň připojení do lokální sítě), ovšem nevýhodou by byla například nemožnost přistupovat z více pracovišť. Hrozila by také ztráta dat při poškození daného počítače, protože data by byla ukládána lokálně. Nevýhodou je také nutnost vždy aplikaci instalovat.

2.1.2 Aplikace běžící lokálně se sdíleným úložištěm

Stejně jako v předchozím případě by se jednalo o nainstalovanou aplikaci na osobním počítači, ovšem s tím rozdílem, že by byla napojena na externí databázi a bylo by tak možné ji nainstalovat a používat z více pracovišť najednou. Nevýhodou je nutnost připojení k databázovému serveru a také nutnost vždy ručně instalovat aplikaci na daný osobní počítač.

[7] Personal computer - osobní počítač

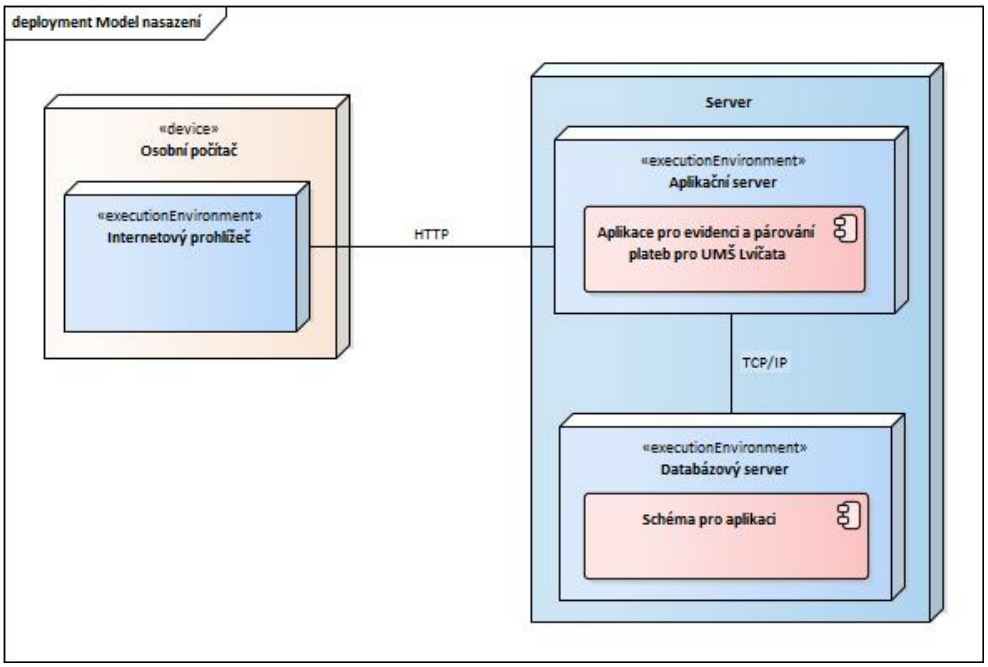
2.1.3 Webová aplikace dostupná z internetového prohlížeče

Aplikace v tomto případě běží na separátním serveru a pracovníci k němu přistupují skrze internetový prohlížeč. Na serveru (lokálním či vzdáleném) běží také databáze určená pro ukládání dat. Výhodou je možnost přistupovat z jakéhokoliv pracoviště bez nutnosti instalovat aplikaci a možnost pracovat simultánně z více pracovišť, pokud s tím aplikace počítá. Nevýhodou je složitější nasazení aplikace a nutnost připojení pracoviště do sítě.

2.2 Zvolené architektonické řešení

Zvoleným řešením je webová aplikace. Nebude ji nutné instalovat na každý počítač, ze kterého se k ní bude přistupovat a s připojením k lokální síti, případně přímo k internetu aktuálně už není problém a jedná se o standard.

Na obrázku 2.1 je model nasazení zvoleného architektonického řešení. Modelem nasazení se rozumí diagram, který rozpoznává architektonicky významné komponenty a mapuje je na fyzický hardware [5].



Obrázek 2.1: Zvolený model nasazení

2.3 Funkční požadavky na systém

Hlavním účelem tvorby programu je nahrávání plateb z ABO výpisů a jejich párování s reálnými útratami žáků. K této funkcionalitě se ale váže nutnost zrealizovat také evidenci mnoha dalších záležitostí. Následuje seznam funkčních požadavků na systém.

- **Evidence žáků** – Systém musí umět vkládání, editaci a mazání žáků, u kterých je potřeba párovat platby za služby školy. Žák může mít v systému zadaného také rodiče, u kterého jsou kontaktní informace (email, telefonní číslo). Jeden rodič může mít v systému libovolné množství žáků. Rodiče v systému slouží pouze k přehlednému ochovávání kontaktních údajů. Dále může mít žák také libovolné zařazení, které reprezentuje například třídu, kterou žák navštěvuje.

- **Evidence účtů** – Ke každému žákovi může být vytvořeno libovolné množství účtů pro různé typy plateb (stravné, školné, kroužky apod.). Účty jsou rozlišeny variabilním symbolem používaným i u plateb.

- **Evidence pohledávek za studenty** – Studentům se v systému evidují pohledávky za účtované služby.

- **Evidence plateb z bankovního účtu** – Aplikace eviduje platby rodičů za žáky. Platby je možné vytvářet, editovat i mazat.

- **Vkládání pohledávek souborem** – Bude možné nahráním souboru do systému vytvořit pohledávky za stravné nebo školné. Soubory jsou ve formátu editoru MS Excel (přípona `.xlsx`) nebo ve formátu OpenDocument (přípona `.ods`).

- **Vkládání ABO souborů** – Systém dokáže zpracovat bankovní výpisy ve formátu ABO a vytvořit na jejich základě platby.

- **Párování plateb a pohledávek** – Při nahrávání pohledávek souborem či plateb ABO souborem se systém pokouší je podle variabilního symbolu také spárovat s existujícími účty v systému. Pokud se spárování nepodaří, je možné položku spárovat ručně, případně v případě chyby je možné ji „odpárovat“.

- **Export do tabulkového editoru** – Program je schopný exportovat relevantní sestavy ve formátu vhodném pro tabulkový editor.

- **Tisk sestav** – Uživatel má možnost vytisknout různé relevantní sestavy.

2.4 Nefunkční požadavky

Nefunkční požadavky nepopisují co má systém umět, ale spíše jakým způsobem to vykonávat. Nejčastěji se jedná o požadované technické specifikace, dále také o požadavek na použití určité technologie (například aby byl výsledek použitelnější pro více potencionálních zákazníků i za cenu práce navíc) nebo požadavky na kvalitu výsledného produktu [6].

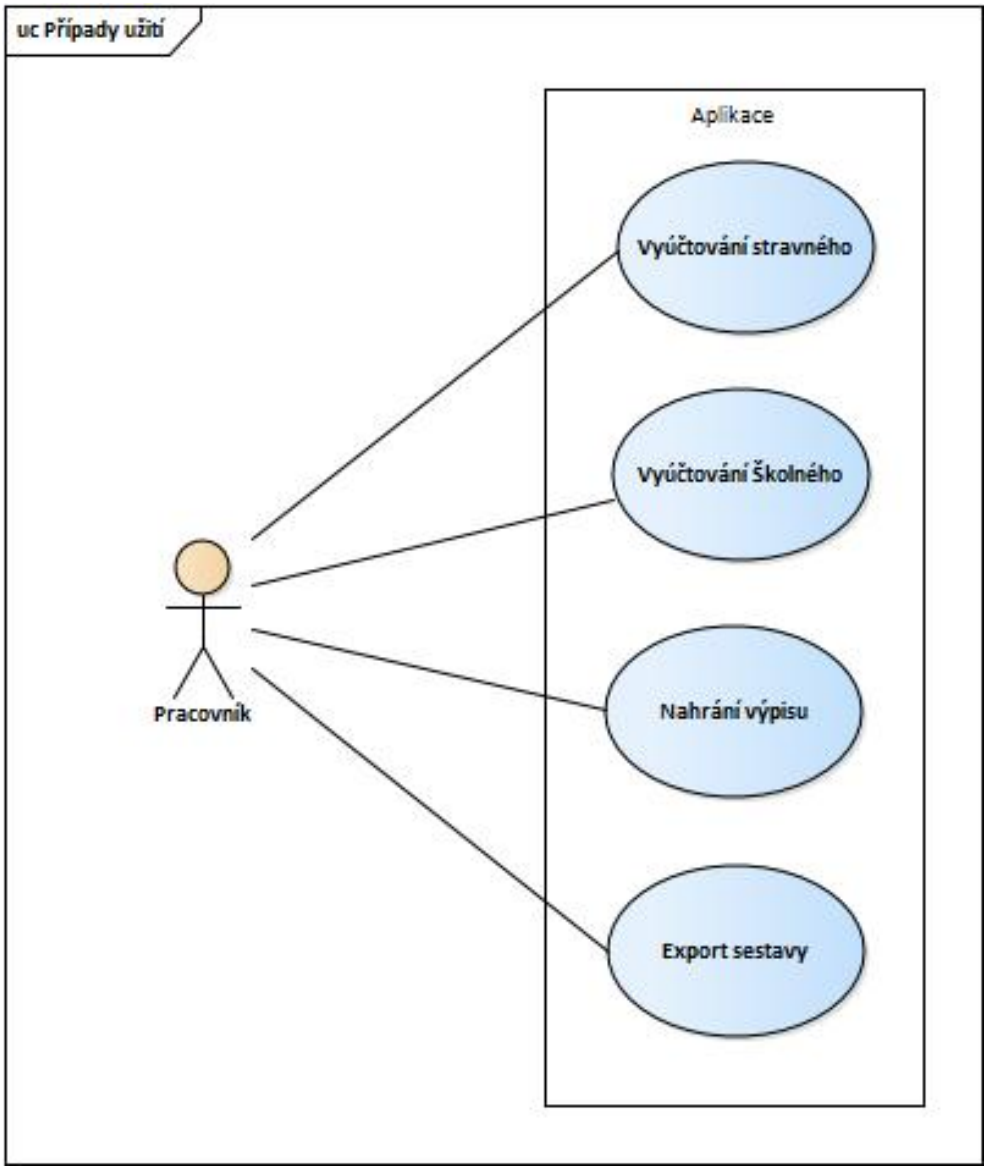
- **Provoz na osobním počítači** – Ke spuštění aplikace nebude potřeba speciálně upravený počítač, minimální požadavky by měly být také v souladu s aktuálními počítači.
- **Možný přístup z více pracovišť najednou** – Bude možné, aby do systému přistupovaly dvě pracoviště (dva počítače) najednou.
- **Grafické uživatelské prostředí** – Aplikace nebude vyžadovat znalost pokročilé práce s počítačem (např. práce pomocí konzole).
- **Kompatibilita s běžnými prohlížeči** – Ke spuštění nebude potřeba žádného speciálního programu, ale bude jej možné používat z běžného internetového prohlížeče (např. Google Chrome[8], Mozilla Firefox[9]).

[8]Dostupné z: `https://www.google.com/chrome/`
[9]Dostupné z: `https://www.mozilla.org/cs/firefox/new/`

2.5 Důležité případy užití

V této části autor popisuje důležité případy užití, které bude zpracovávat vyvíjená aplikace. Nejdůležitějšími jsou ty, které umožňují samotné párování plateb, což je nejdůležitější funkcionalita programu. Tyto případy užití jsou vyzobrazeny v obrázku 2.2.

- **Vyúčtování stravného** – Pracovník nahraje do systému soubor z tabulkového editoru. Soubor obsahuje údaje o počtech celodenního nebo půldenního stravného daných žáků a systém tyto útraty přiřadí podle variabilních symbolů k daným účtům.
- **Vyúčtování školného** – Podobně jako u stravného se nahraje soubor z tabulkového editoru s identifikátory účtů a s částkami za školné, které se k nim následně přidají.
- **Nahrání výpisu** – Pracovník nahraje do systému ABO soubor, ze kterého jsou vyextrahovány platby a následně zaevidovány v systému. Systém také automaticky páruje platby podle variabilního symbolu
- **Export sestavy** – V jakémkoliv seznamu lze stáhnout aktuální výběr ve formátu vhodném pro tabulkové editory. Pracovník vybere požadovaný seznam, připadně si jej přizpůsobí vhodným seřazením či filtrem a stiskne tlačítko stáhnout. Objeví se mu nabídka možných formátů a po vybrání se data ve vybraném formátu stáhnou.



Obrázek 2.2: Důležité případy užití

2.6 Doménový model

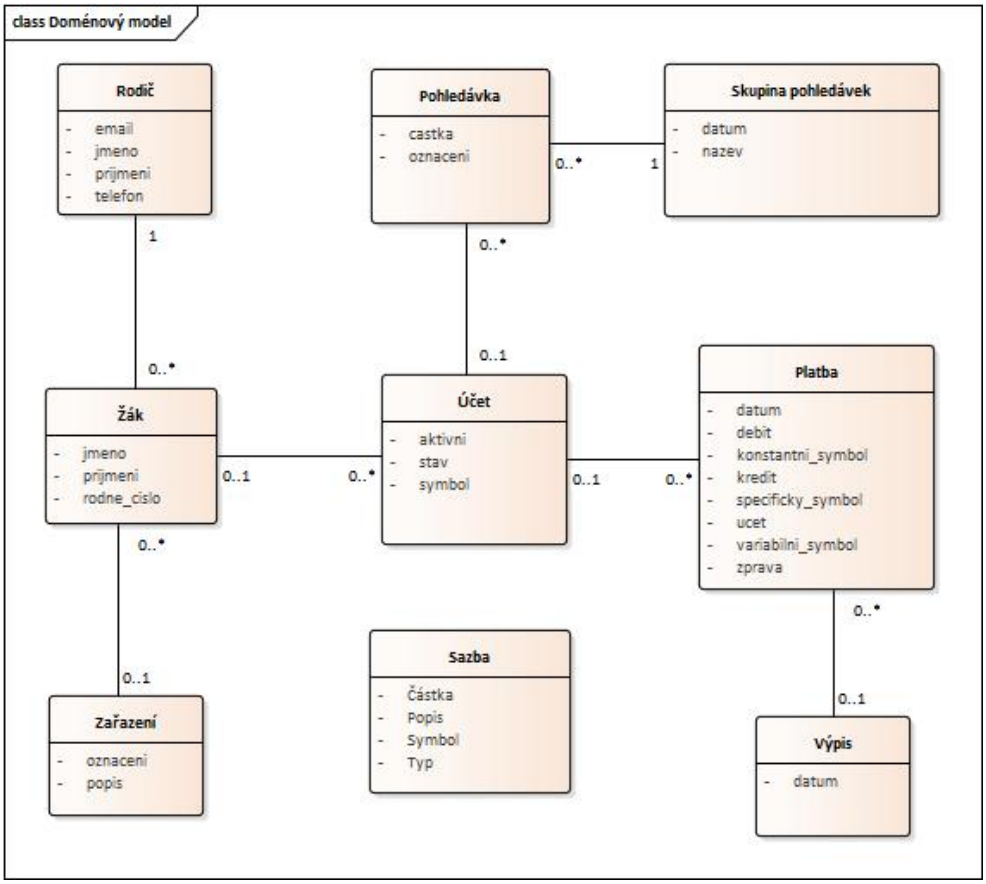
Doménový model zobrazuje důležité objekty, se kterými systém pracuje. Jedná se o zjednodušený pohled na základní logiku systému. Třídy v něm jsou podstatně zjednodušené a jedná se o náčrt budoucího databázového schématu.

Dle [7] je to model, který stejně jako model případů užití vzniká v raných fázích vývoje. Jedná se o formu diagramu tříd. Třídy v něm jsou ale značně zjednodušené, neobsahují finální jména či názvy atributů a neobsahují žádné metody.

Doménový model je tedy náčrtem základních entit systému a vztahů mezi nimi, který není závislý na platformě ani použité technologii a atributy v něm nemají určené datové typy [7].

Popis domén implementovaného systému se nachází společně s jejich grafickým znázorněním včetně vazeb na následujících dvou stranách (obrázek 2.3).

- **Žák** – Jedná se o žáka navštěvujícího UMŠ Lvíčata. Evidujeme u něj jméno, příjmení a kvůli identifikaci také rodné číslo.
- **Rodič** – Rodič slouží jako kontaktní osoba žáka a také jako jedna z metod filtrování žáků. Evidujeme u něj jméno, příjmení a kontaktní informace (email a telefon). Vazba je typu 1:N, tedy jeden rodič může mít více žáků na škole, ale ke každému žákovi je evidován pouze jeden rodič.
- **Zařazení** – Tato entita slouží především k filtrování žáků podle tříd v základní škole nebo skupin v mateřské škole. Každá skupina má svůj název a může mít také popis.
- **Účet** – Účet je centrální entitou celého informačního systému. Je identifikován variabilním symbolem a patří vždy k maximálně jednomu žákovi (nebo žádnému), naopak jeden žák může mít vedených více účtů pro účely rozlišení plateb, například za stravné či školné. Obsahuje také údaj o tom, jestli je účet aktivní. Nedílnou součástí je stav účtu, který udává aktuální peněžní situaci na účtu, tedy souhrn všech pohledávek a plateb k danému účtu.
- **Skupina pohledávek** -– Je skupinou pro pohledávky týkající se jedné události (například vyúčtování stravného za jeden konkrétní měsíc). Obsahuje pouze datum přidání a její popis.
- **Pohledávka** – Je to částka, která je účtována danému žákovi za určité služby (stravné, školné, apod.) na jeden z jeho účtů. Obsahuje pouze částku a označení dané pohledávky.
- **Výpis** – Jedná se o entitu sdružující platby nahrané v jednom ABO souboru. Obsahuje pouze datum a opět je zde především k filtrovacím účelům.
- **Platba** – Je to hlavní výstup z importování ABO souboru. Obsahuje všechny běžné údaje k danému pohybu na účtu.
- **Sazba** – Jedná se o jedinou třídou, která nemá jedinou asociaci. Představuje sazbu za určitou „službu“, například za celodenní nebo půldenní stravné. Symbol je název sazby, který je zároveň vyhledáván v nahraných tabulkách a pomocí něj je tedy identifikován. Typ udává, jestli se jedná o sazbu nebo přímo o hodnotu, je tedy možné vložit tabulku, ve které bude sloupec „Školné“, ze kterého se bude přímo brát částka a nebude se násobit částkou uvedenou u sazby. Obsahuje také popis sazby pro přehlednost.



Obrázek 2.3: Doménový model

Kapitola 3

Výběr vhodných technologií

3.1 Možné programovací jazyky

Pro vývoj webových aplikací se využívá mnoho různých technologií. Každý jazyk s sebou nese určité nároky na nasazení (požadovaný typ serveru), kompatibilní doplňky (například databáze) či pomocné knihovny.

V této kapitole autor rozebírá vybrané vhodné jazyky pro implementaci.

3.1.1 Java[10]

Java je stále jeden z nejpoužívanějších jazyků používaných pro vývoj informačních systémů [8], případně i pro jiné účely (např. mobilní aplikace). Jedná se o objektově orientovaný jazyk, jehož syntaxe vychází z jazyků C++ a C [9].

Hlavním rysem jazyka Java je to, že narozdíl od vysokoúrovňových jazyků jako například C++ není kompilovaným jazykem. Kompilace je překlad zdrojového kódu do binárního, spustitelného souboru. Kompilované jazyky jsou z tohoto důvodu závislé na platformě. Java narozdíl od toho je jazykem interpretovaným. Zdrojový kód se tedy překládá do takzvaného „bajtkódu“, který se poté spouští pomocí JVM, který je nahrán v procesoru [10].

Jedná se o hojně využívaný jazyk pro tvorbu informačních systémů, a to především kvůli vysoké bezpečnosti a spolehlivosti. Nevýhodou je však nutnost zajištění specializovanějšího hostingu pro výsledný systém, protože tento jazyk není tak často podporován jako některé alternativy.

[10] Dostupné z: `https://java.com/`

3.1.2 PHP[11]

PHP je populární skriptovací jazyk uzpůsobený k vývoji webových aplikací [11]. Jeho největšími výhodami je bezproblémové nasazování na většinu hostingů a fakt, že se jedná, stejně jako v případě Javy, o jazyk interpretovaný. U PHP ovšem nedochází ani k překladu do „bajtkódu“, ale na server se nasazují přímo PHP skripty, což zjednodušuje nasazování nových funkcí.

Pro tento jazyk existují také desítky různých frameworků, které je možné využít. Jedná se také o velice rozšířenou technologii [8] a díky tomu je dostupné velké množství materiálu na internetu.

Nejnovější verzí PHP je verze 7.1. Ve verzi 7 přibyla například omezené možnosti statického typování (původně se jedná o čistě dynamicky typovaný jazyk) a samotný jazyk byl podstatně zrychlen [12].

3.1.3 Ruby[12]

Ruby je dynamický, otevřený programovací jazyk zaměřený na jednoduchost a produktivitu [13]. Také se jedná o interpretovaný a je objektově orientovaný. Má také opravdu jednoduchou syntaxi a proto je snadný k naučení, ale i přesto se jedná o výkonný jazyk.

Hlavním důvodem zařazení tohoto jazyka je velice populární MVC[13] framework Ruby on Rails[14]. Na tomto frameworku běží hodně populárních stránek (např. SoundCloud[15] a Twitch[16]).

3.2 Možné databáze

Databází je aktuálně na výběr velké množství. Je k dispozici velké množství placených řešení, ale také open-source řešení. Některé databáze se také liší principy, na kterých jsou postavené. V této kapitole autor popisuje vybrané databáze.

3.2.1 MySQL[17]

„MySQL je nejoblíbenější open source databází na světě. Databáze MySQL se díky osvědčené výkonnosti, spolehlivosti a snadnému používání stala volbou číslo jedna pro webové aplikace. Využívají ji významné webové atributy včetně Facebooku, Twitteru či YouTube i všech pět nejlepších webových stránek.“ [14]

[11] Dostupné z: `http://php.net`
[12] Dostupné z: `https://www.ruby-lang.org/en/`
[13] Model-View-Controller
[14] Dostupné z: `http://rubyonrails.org`
[15] `https://soundcloud.com`
[16] `https://www.twitch.tv`
[17] Dostupné z: `https://www.mysql.com`

Výhodou MySQL je její velká rozšířenost u většiny poskytovatelů hostingu. Je také podporována většinou jazyků a také ORM[18] frameworky (například Doctrine 2[19] pro PHP nebo Hibernate[20] pro Javu).

3.2.2 PostgreSQL[21]

Také se jedná o databázový systém s otevřeným zdrojovým kódem. Vznikl na konci devadesátých let a má vynikající pověst pro svou spolehlivost a bezpečnost. Je použitelný na všech rozšířených operačních systémech a umožňuje také moderní datové typy jako například JSON[22] nebo XML[23] [15].

3.2.3 MongoDB[24]

Jedná se o modernější přístup k ukládání dat do databází. Narozdíl od MySQL i PostgreSQL se nejedná o relační databázi, ale o takzvanou dokumentovou databázi. Řadí se také do skupiny databází nazývané „noSQL“, to se používá pro takové databáze, které nevyužívají standardní tabulková schémata, ale umožňují vkládat data dynamicky. V praxi to znamená, že se do databáze vkládají data většinou ve formátu JSON[25]. Oproti tradičním databázím se tedy hodí například pro jazyk JavaScript, ale nehodí se pro složitější databázová schémata [16].

[18] Object-relational Mapping
[19] Dostupné z: `https://www.doctrine-project.org`
[20] Dostupné z: `http://hibernate.org`
[21] Dostupné z: `https://www.postgresql.org`
[22] JavaScript Object Notation
[23] eXtensible Markup Language
[24] Dostupné z: `https://www.mongodb.com`
[25] JavaScript Object Notation

3.3 Možné aplikační servery

Narozdíl od databázových serverů se aplikačních serverů používá poměrně malé množství. V této kapitole autor popisuje dva vybrané možné servery.

3.3.1 Apache[26]

Apache je vyvíjen již od devadesátých let a stále patří k nejpoužívanějším HTTP serverům [17] [18]. Je často volen z důvodu rozsáhlé dokumentace, dobré podpoře a přizpůsobitelnosti systému, na kterém běží [19].

3.3.2 NGINX[27]

NGINX je novější než Apache a za poslední roky jeho obliba velmi stoupla [17]. Jeho výhodami jsou lepší výkon a zároveň menší náročnost na hardware [19]. Největším rozdílem oproti Apache je, že NGINX nevytváří pro každého uživatele nový proces. To u menších projektů snižuje náročnost, protože ve výchozím stavu vytváří NGINX jeden proces na každé dostupné procesorové jádro [20].

3.4 Zvolené technologie

Při výběru především programovacího jazyka hrála roli rozšiřitelnost výsledné aplikace, dostupnost balíčků vhodných k účelům bakalářské práce a také sympatie autora. Aplikace bude implementována v jazyce PHP za použití některého MVC[28] frameworku (viz dále).

Stejně jako při vývoji bude i při nasazení využit aplikační server Apache a databáze MySQL. Jedná se o běžnou kombinaci a o spolehlivé nástroje [21].

[26] Dostupné z: `https://httpd.apache.org`
[27] Dostupné z: `https://www.nginx.com`
[28] Model-view-controller

3.5 Vhodné frameworky a balíčky

K samotnému programovacímu jazyku je možné použít již existující knihovny a nástroje. Pro jazyk PHP je výběr velice široký a nabízí se mnoho vhodných nástrojů.

3.5.1 Vhodné PHP frameworky

- **Symfony**[29] – Jedná se o velice robustní MVC[30] framework. Samotný framework je (dle [22]) implementován nad takzvanými Symfony komponentami, což jsou nástroje, které lze použít i samostatně. Standardní verze toho frameworku také obsahuje ORM[31] framework Doctrine 2, kterému se autor dále věnuje v kapitole 3.5.3. Využívá šablonovací systém Twig[32].

- **Nette**[33] – Stejně jako u Symfony se jedná o sbírku mnoha komponent, které tvoří framework, ale je možné některé využít i samostatně. Výhodou Nette je, že se nejedná až o tak robustní řešení jako Symfony. Na svých stránkách se pyšní vysokou produktivitou, čistým kódem a vysokou bezpečností [23]. Nevýhodou je nutnost doplňovat mnoho funkcionalit, které jsou například v Symfony přímo vestavěné (především se jedná o nepřítomnost jakéhokoliv ORM frameworku).

- **Laravel**[34] – Dle [24] se jedná aktuálně o nejpoužívanější framework. Tento framework je postavený, stejně jako Symfony, nad Symfony komponentami, jádro je tedy identické. Rozdíl je především ve stylu syntaxe a psaní kódu obecně. Základní myšlenkou Laravelu je šetření kódu a zahrnování obecně využívaných procedur přímo do frameworku. Tohle je však zároveň i nevýhodou, protože to způsobuje jeho menší přizpůsobitelnost [25].

[29] Dostupné z: `https://symfony.com`
[30] Model-view-controller
[31] Object-relational mapping
[32] Dostupné z: `https://twig.symfony.com`
[33] Dostupné z: `https://nette.org/cs/`
[34] Dostupné z: `https://laravel.com`

3.5.2 Zvolený PHP framework

Aplikace bude implementována ve frameworku Symfony. K tomuto rozhodnutí došel autor po zhodnocení všech zmíněných frameworků v souvislosti s požadavky na systém (viz 2.3) a také v souvislosti s dostupnými nástroji vázanými k jednotlivým frameworkům.

Symfony umožní zjednodušit implementaci použitím balíku Sonata (viz 3.5.3.3) a také do něj nebude potřeba ručně přidávat ORM framework (to by bylo potřeba při použití frameworku Nette). Oproti frameworku Laravel se také jedná o univerzálnější nástroj.

3.5.3 Dodatečné nástroje a balíčky

Pro jazyk PHP existuje nezanedbatelné množství knihoven a balíků, které je možné použít. Není tedy potřeba ručně implementovat čtení souborů z tabulkových editorů, ani bankovních výpisů v ABO formátu (viz 1.4). Díky frameworku Symfony je možné také použít administrační balík Sonata pro zjednodušení celé implementace.

3.5.3.1 Doctrine 2[35]

Doctrine je přímo součástí Symfony frameworku, ale je možné jej použít i samostatně (je možnost jej zakomponovat i do Nette frameworku [26]).

Jedná se o čistý ORM framework pro PHP. Umožňuje tedy anotacemi mapovat třídy na tabulky v databázi (Doctrine využívá PDO[36], má tedy kompatibilitu s většinou používaných databází [27]).Jedná se o velmi vhodný nástroj pro naše účely, protože je kompatibilní s balíkem Sonata, který je využit na tvorbu celého systému [28].

3.5.3.2 Composer[37]

Je to nástroj sloužící k instalaci balíčků a jejich závislostí pro PHP. Podobným nástrojem je například npm[38] pro jazyk javascript [29].

Pomocí tohoto nástroje je také možné vytvořit základní strukturu frameworku Symfony a zároveň nainstalovat jeho základní závislosti.

[35] Dostupné z: `https://www.doctrine-project.org`
[36] PHP Data Objects
[37] Dostupné z: `https://getcomposer.org`
[38] Dostupné z: `https://www.npmjs.com`

3.5.3.3 Sonata Admin Bundle[39]

Sonata je soubor balíčků, které rozšiřují funkcionalitu Symfony frameworku o mnoho dalších funkcí a dokáže výrazně zjednodušit tvorbu určitých často opakovaných funkcionalit v různých aplikacích. Pro účely této aplikace je použit především její administrační balík, který zjednodušuje tvorbu standardních administračních panelů. Spolupracuje přímo s Doctrine 2 a po vytvoření a nakonfigurování administračních tříd (viz 4.3) vygeneruje v podstatě kompletní administraci [30].

3.5.3.4 PhpSpreadsheet[40]

Jedná se o nástupce balíku PhpOffice[41], který byl přizpůsoben pro instalaci pomocí nástroje Composer. Je to knihovna napsaná v čistém PHP (nepoužívá tedy žádný externí nástroj), která umožňuje čtení a vytváření souborů pro tabulkové editory (například MS Excel nebo LibreOffice Calc) [31].

3.5.3.5 Bank Statements[42]

Tento balíček umožňuje jednoduché parsování[43] a následné zpracování bankovních výpisu v ABO formátu. Tento balík podporuje i ABO soubory bank, které je mají lehce upraveny (například ČSOB) [32].

3.5.3.6 Další použité nástroje

Pro účely verzování a správy celého projektu je použit verzovací nástroj Git[44]. Jako vývojové prostředí je použité IDE PhpStorm[45] od firmy JetBrains.

[39] Dostupné z: `https://sonata-project.org/bundles/`
[40] Dostupné z: `https://phpspreadsheet.readthedocs.io/en/develop/`
[41] Dostupné z: `https://github.com/PHPOffice`
[42] Dostupné z: `https://github.com/jakubzapletal/bank-statements`
[43] Zpracování surových dat na strukturovaná data umožňující další použití
[44] Dostupné z: `https://git-scm.com`
[45] Dostupné z: `https://www.jetbrains.com/phpstorm/`

3.6 Popis zvoleného frameworku

3.6.1 MVC architektura

Symfony je frameworkem využívajícím architekturu MVC[46] [33]. Z této zkratky přímo vycházejí hlavní tři části takovéhoto frameworku. Tyto části je ideální držet oddělené kvůli přehlednosti a zachování tohoto paradigmatu. Následuje popis jednotlivých částí.

- **Controller** – *„Kontroler je část, se kterou komunikuje uživatel. Předá jí parametry a ona mu vrátí HTML stránku.“* [33] Typicky tedy kontroler dostane od uživatele adresu (URL[47]), případně i nějaké parametry (např. GET, POST). Podle těchto parametrů získá od modelu potřebná data, ty předá pohledu (v případě Symfony se jedná o Twig šablony (viz 3.6.2.2)), který vygeneruje HTML soubor který se odešle uživateli jako odpověď [33].
- **Model** – *„Obsahují logiku aplikace, jako např. práci s databází nebo výpočty. Každá datová entita má většinou svůj model (uživatel, článek, komentář, …).“*[33].
- **View** – V rámci Symfony jsou výchozím pohledem Twig šablony (viz 3.6.2.2). Ty se přegenerovávají do HTML kódu který se posílá uživateli [33]. Jedná se tedy o část, která prezentuje data získaná z modelu v podobě vhodné pro zobrazení.

3.6.2 Důležité části frameworku Symfony

3.6.2.1 Controller

Kontrolery představují řídící vrstvu v Symfony. Zpracovávají objekty typu `Request` a vracejí objekty typu `Response`. Odpovědí může být například HTML, JSON, XML, přesměrování nebo stažení souboru [34]. Příklad jednoduchého Kontroleru je na ukázce 3.1.

[46] Model-View-Controller
[47] Uniform Resource Locator

Ukázka kódu 3.1: Příklad Symfony kontroleru

```
        namespace App\Controller;
...
class MyController {

    /**
      * @Route("/hello/{name}", name="app_hello")
      */
    public function helloAction($name) {

        return new Response (
            '<html><body>Hello␣'.$name$.'</body></html>'
        );
,,,
```

3.6.2.2 Twig

Twig je šablonovací systém pro PHP. Vyznačuje se rychlostí (uvádí minimální overhead[48]), bezpečností a flexibilitou [35]. Oproti čistému PHP umožňuje využívat zjednodušenou syntaxi včetně filtrů. Příklad některých Twig kódů je na ukázce 3.2.

Ukázka kódu 3.2: Příklad Twig šablony

```
// priklad for cyklu, else se vykonává pokud je pole users prazdne
{% for user in users %}
    * {{ user.name }}
{% else %}
    Nebyli nalezeni zadni uzivatele...
{% endfor %}

// priklad rozsirovani sablony layout.html
{% extends "layout.html" %}

{% block content %}
    Obsah stranky...
{% endblock %}
```

[48] výkon a prostředky požadované navíc oproti čistému PHP

3.6.2.3 konfigurace

Nejběžnějšími konfiguračními soubory frameworku Symfony jsou soubory `YAML`. Jedná se o jednoduché konfigurační soubory. Alternativně lze použít také XML soubory nebo PHP kódy [36]. Ukázky YAML konfigurací jsou v následujících kapitolách (například ukázky 4.7 a 4.9).

3.6.2.4 ORM

V Symfony se již v základu nachází ORM framework Doctrine 2. Tento nástroj je rozebrán v části 4.1.

3.6.2.5 Form Types

Symfony obsahuje také vytváření formulářů včetně jejich validací. `Form Type` se zkládá z `Fields` (polí), které reprezentují HTML formulářové vstupy. Validace je možné přidávat přímo ve `Form Type`, nebo pomocí validací přímo v entitách [37]. Na ukázce 3.3 je předvedena tvorba formuláře pro nahrání tabulky s pohledávkami.

Ukázka kódu 3.3: Ukázka vytvoření formuláře v Symfony frameworku

```
public function buildForm(FormBuilderInterface $builder,
                          array $options) {
    $builder->add('description', TextType::class, [
        'constraints' => [
            new Length([
                'min' => '3'
            ])
        ]
    ])
    ->add('sheet', FileType::class, [
        'constraints' => [
            new File([
                ...
            ])
        ]
    ]);
}
```

Kapitola 4

Realizace

Následující kapitola popisuje implementaci navrhnutého systému za použití zvolených technologií. Veškeré zdrojové kódy včetne skriptu na vytvoření databázového schématu se nachází na přiloženém CD C

4.1 Databázové schéma

Databázové schéma vygenerované z oanotovaných entit (pomocí Doctrine 2) odpovídá doménovému modelu. Nebylo potřeba dekomponovat žádnou vazbu a přibyly pouze cizí klíče reprezentující vztahy (relace) mezi entitami. Jako demonstrace tohoto mapování anotacemi je použita vazba mezi studentem a účtem (lze ji vidět na doménovém modelu v obrázku 2.3) a databázovém schématu v obrázku 4.1). Na ukázce kódu 4.1 je vidět anotace na straně studenta, na ukázce 4.2 pak na straně účtu.

Ukázka kódu 4.1: Anotace na straně studenta

```
/**
  * @ORM\Table(name="student")
  * @ORM\Entity(repositoryClass="App\Repository\StudentRepository")
  */
class Student {

    ...

    /**
     * @ORM\OneToMany(targetEntity="Account", mappedBy="student")
     */
    private $accounts;

    ...
```

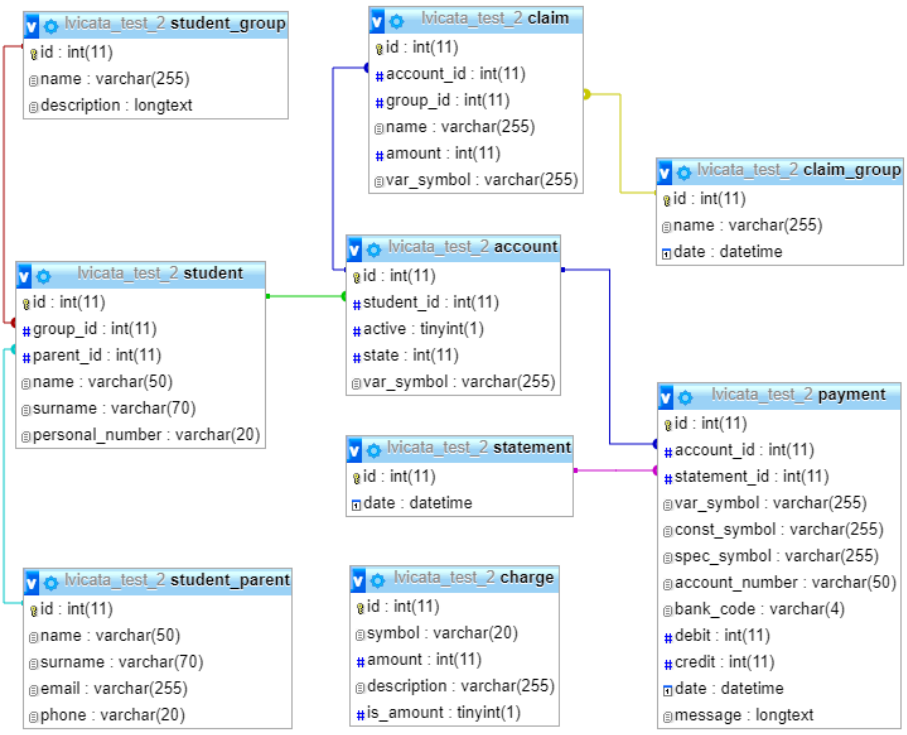
Ukázka kódu 4.2: Anotace na straně účtu

```
/**
  * @ORM\Table(name="account")
  * @ORM\Entity(repositoryClass="App\Repository\AccounttRepository")
  */
class Account {

    ...

    /**
     * @ORM\ManyToOne(targetEntity="Student", inversedBy="accounts")
     * @ORM\JoinColumn(name="student_id", referencedColumnName="id")
     */
    private $student$;

    ...
```

Výsledkem je tedy velice podobné schéma (na obrázku 4.1) jako v případě doménového modelu (na obrázku 2.3). Pouze proběhlo přejmenování všech názvů do angličtiny a přidání cizích klíčů. Žádnou vazbu ani nebylo potřeba dekomponovat, protože všechny vazby jsou typu 1:N (například jeden rodič může mít více studentů, ale jeden student může mít pouze jednoho rodiče). Přeložení názvů z doménového modelu se nachází v tabulce 4.1.

Tabulka 4.1: Překlad názvů z doménového modelu

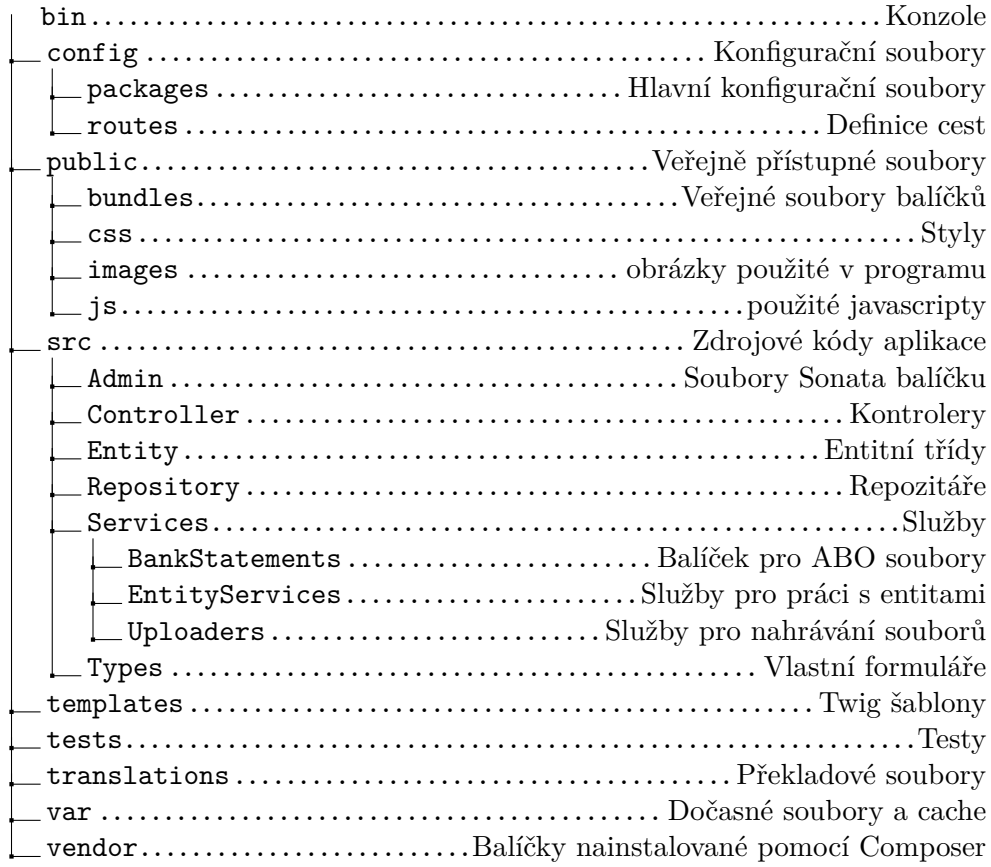
Původní název	Přeložený název
Rodič	student_parent
Žák	student
Zařazení	student_group
Účet	account
Pohledávka	claim
Skupina pohledávek	claim_group
Platba	payment
Výpis	statement
Sazba	charge



Obrázek 4.1: Databázové schéma

4.2 Struktura aplikace

V nejnovější verzi frameworku Symfony se lehce změnila a dresářová struktura. Těmito změnami se autor nezaobírá, namísto toho popisuje strukturu implementované aplikace (struktura je v adresářovém stromu 4.2).



Obrázek 4.2: Adresářová struktura projektu

- **Konzole** – Obsahuje především PHP soubor Console.php, který slouží k vykonávání příkazů skrze terminál. Je možné například vytvářet databázová schémata, zašifrovávat hesla v konfiguračních souborech nebo spouštět testovací server.
- **Konfigurační soubory** – Konfigurace se v Symfony frameworku dají ukládat buď v XML formátu, nebo YAML formátu. Naše aplikace využívá druhou, častěji využívanou možnost.
- **Hlavní konfigurační soubory** – Obsahuje jednotlivé konfigurační soubory pro všechny komponenty systému, které konfiguraci vyžadují. Jedná se například o soubory `framework.yml` nebo `sonata-admin.yml`.

- **Definice cest** – V tomto adresáři jsou přítomné všechny definice URL[49] cest k daným kontrolerům a jejich akcím a také například definice odhlašovací adresy.

- **Veřejně přístupné soubory** – Nachází se zde skripty, styly a ostatní soubory, které jsou veřejně dostupné skrze `HTTP` protokol. Obsahuje také soubor `index.php`, přes který probíhá veškerá komunikace se systémem.

- **Zdrojové kódy aplikace** – Obsahuje samotné zdrojové kódy celého systému.

- **Soubory Sonata balíčku** – Obsahuje třídy, pomocí kterých se konfigurují jednotlivé stránky systému. Tyto třídy jsou potommky třídy `AbstractAdmin` z balíčku Sonata a jsou podrobněji rozebrány v kapitole 4.3.

- **Kontrolery** – Obsahuje všechny kontrolery naší aplikace. Běžné kontrolery (potomky abstraktní třídy Cotroller) jsou používány pouze na nahrávání souborů (bankovní výpisy, tabulky s pohledávkami). Dále se zde nachází i specializované kontrolery balíčku Sonata (potomky abstraktní třídy CRUDController), které slouží k doplnění některých funkcionalit do administračních souborů.

- **Entitní třídy** – Jedná se o třídy, které obsahují anotace ORM[50] frameworku Doctrine, který obstarává ukládání těchto tříd do relační databáze.

- **Repozitáře** – Jsou to třídy, které se vážou k entitním třídám a obstarávají získávání daných entit z databáze. V těchto třídách jsou již vygenerované základní možnosti filtrování a je do nich možné přidávat i vlastní metody.

- **Služby** – Obsahuje pomocné třídy obsahující vlastní logiku v aplikaci. Jedná se především o práci s entitami (vytváření) a o zpracovávání souborů (výpisy a tabulky). Obsahuje také balíček Bank Statements pro zpracovávání ABO výpisů.

- **Balíček pro ABO soubory** – Jedná se o balíček jakubzapletal/bank-statements. Je určený pro instalaci pomocí nástroje Composer 3.5.3.2, vyžaduje ovšem starší verzi Symfony frameworku a tak byla provedena extrakce potřebných funkcionalit z balíčku a jeho zakomponování přímo do služeb aplikace.

[49] Uniform Resource Locator
[50] Object-relational mapping

- **Služby pro práci s entitami** – Při nahrávání souborů je nutné vytvářet některé nové entity a zajistit jejich perzistenci v databázi. Pro tyto účely existují tyto třídy.
- **Vlastní formuláře** – Obsahuje třídy pro generování nahrávacích formulářů pro bankovní výpisy a pohledávky v tabulkách.
- **Twig šablony** – Obsahuje šablony, ze kterých se generují výsledné HTML stránky.
- **Testy** – Obsahuje soubory s jednotkovými testy.
- **Překladové soubory** – Obsahuje soubory ve formátu YAML, které obsahují překlady zpráv ze Sonata balíčku z angličtiny do češtiny. Dále obsahuje také zprávy formulářových validátorů.
- **Dočasné soubory a cache** – Obsahuje dočasně nahrané soubory, vygenerované cache pro aplikaci a logy.
- **Balíčky nainstalované pomocí Composer** – Jedná se o vygenerovanou složku, která vznikne spuštěním příkazu `composer install`. Obsahuje všechny balíčky vyžadované aplikací a soubor `autoload.php`, který je do aplikace přidá.

4.3 Kostra aplikace

Pro vytvoření hlavního skeletonu aplikace, tedy v podstatě administrace nad danými entitami, je využit administrační balík Sonata (viz 3.5.3.3). Základem tohoto balíku jsou třídy odvozené od abstraktní třídy `Admin`. Tato třída obsahuje metody, ve kterých se definují nastavení části administrace pro danou entitu.

Seznam objektů v systému se definuje v metodě `configureListFields`. Ta jako parametr dostává instanci třídy `ListMapper`, pomocí které se nastavují jednotlivé sloupce zobrazené v seznamu a také akce k daným entitám (např. editace). Nastavení seznamu objektů je předvedeno na účtech na ukázce 4.3.

Ukázka kódu 4.3: Nastavení seznamu účtů ve třídě AccountAdmin

```
...
protected function configureListFields(ListMapper $list) {
    $list->add('var_symbol')
        ->add('student.fullname')
        ->add('state', 'currency', [
            'currency' => 'CZK'
        ])
        ->add('active')
        ->add('_action', 'actions', array(
            'actions' => array(
                'edit' => array(),
                'delete' => array()
            )
        ));
}
...
```

Možnosti filtrace se definují v metodě `configureDatagridFilters`. Ta také dostává parametrem instanci třídy, tentokrát však třídy `DatagridMapper`. Pomocí tohoto objektu se možnosti filtrace editují, přičemž Sonata má dostatečné množství různých typů filtrů [38]. Nastavení filtrace je předvedeno na ukázce 4.4.

Ukázka kódu 4.4: Nastavení možností filtrace ve třídě AccountAdmin

```
...
protected function configureDatagridFilters(DatagridMapper $filter) {
    $filter->add('student', 'doctrine_orm_model', [
        'mapping_type' => ClassMetadataInfo::MANY_TO_ONE
    ], null, [
        'class' => Student::class,
        'choice_label' => 'fullname',
        'multiple' => true
    ])
    ->add('state', 'doctrine_orm_number', array(), MoneyType::class, [
        'currency' => 'CZK'
    ])
    ->add('active');
}
...
```

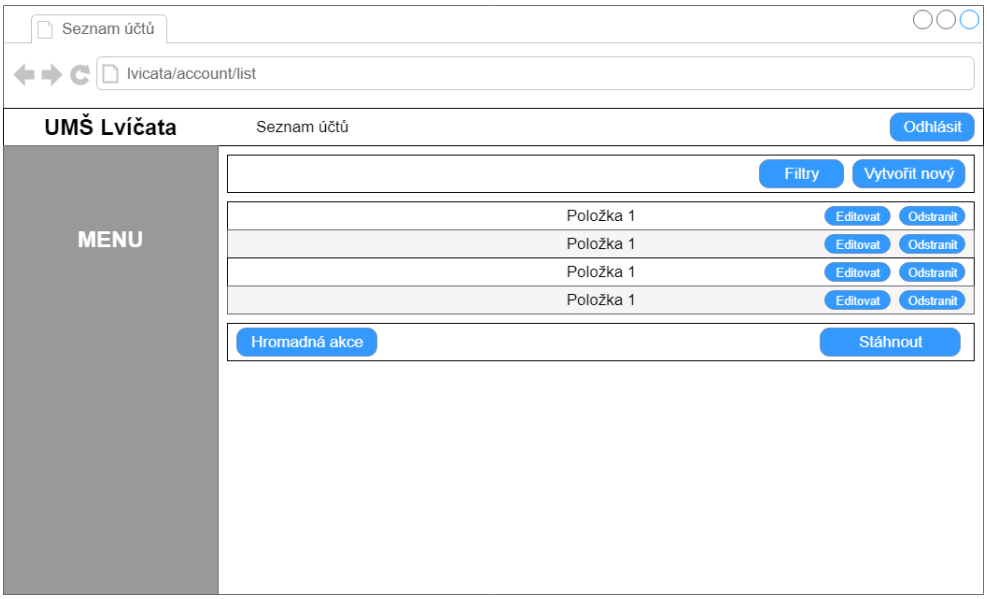
Poslední povinnou částí každého potomka třídy `Admin` je nastavení formuláře na vytváření nových entit a editaci těch stávajících. Opět se jedná o metodu, tentokrát `configureFormFields`. Parametrem je instance třídy `FormMapper`, pomocí které se formulář nastavuje. Nastavení formuláře je předvedeno opět na entitě `Account` (účet v doménovém modelu) na ukázce 4.5.

Ukázka kódu 4.5: Nastavení formuláře ve třídě AccountAdmin

```
...
protected function configureFormFields(FormMapper $form) {
    $form->add('var_symbol', TextType::class)
        ->add('student', 'sonata_type_model', [
                'property' => 'fullname',
                'required' => false
            ])
        ->add('active', CheckboxType::class, [
            'required' => false
        ]);
}
...
```

V těchto třídách se dají nastavit ještě mnohé další funkce, například formát dat při exportu nebo dostupné formáty pro export. Pro všechny entity jsou však tyto metody většinou stejné a proto byly nastaveny společně pro všechny najednou pomocí jedné `Trait`. `Trait` je mechanismus rozšiřování funkcionalit tříd bez nutnosti dědit od nějaké třídy. Jsou v PHP od verze 5.4 a byly přidány z toho důvodu, že PHP je jazyk umožňující dědit pouze od jedné třídy (což neplatí například u Javy) [39].

Vzhled vygenerované aplikace odpovídá běžným systémům. Byly však provedeny některé úpravy aby výsledek odpovídal požadavkům. Návrh obrazovky je demonstrován na seznamu účtů na obrázku 4.3.



Obrázek 4.3: Standardní obrazovka seznamu entit

4.4 Zabezpečení

4.4.1 Nejčastější útoky na webové aplikace

Dle [40] jsou nejčastějšími útoky na aplikace útoky typu SQL[51] Injection a XSS[52]. Následuje popis jednotlivých útoků.

- **SQL Injection** – Podle [41] se jedná o vložení vlastního SQL dotazu do datového vstupu na webu, například skrze formulář nebo aplikační API[53]. Pokud je aplikace zranitelná vůči těmto útokům, může útočník způsobit ztrátu dat, jejich získání nebo modifikaci a v krajních případech je možné aplikaci úplně odstavit nebo nad ní převzít kontrolu [40].
- **XSS** – Podobně jako SQL Injection se jedná o „propašování“ vlastního kódu do cílové aplikace. Narozdíl od SQL Injection se u XSS ale nejedná o SQL dotaz, nýbrž o kus vlastního kódu, který se v cílové aplikaci spustí. Například se tedy může jednat o kus javascriptového kódu nebo o kus kódu v jazyce šablon využitých v aplikaci (například Twig, Latte, JSP). Výsledkem tohoto útoku může být krádež uživatelských dat z internetového prohlížeče nebo přesměrování na jiné, většinou nebezpečné, stránky [42][40].

4.4.2 Ošetření zmíněných hrozeb

Oběma zmíněným útokům se dá zábránit escapováním všech částí kódu, které mohou být modifikovány uživateli (formuláře, výpisy uživatelských vstupů z databáze, apod.). Escapování je postup, kdy před přidáním tohoto zranitelného kusu kódu se jeho obsah pošle do dané funkce a ta tento obsah zpracuje tak, aby daný kus kódu nebyl nebezpečný [43]. U čistého PHP se jedná o funkce `addSlashes` (používaná při ukládání do databáze) a `htmlspecialchars` (používána při vypisování dat do šablony).

Framework Symfony využívá šablonovací systém Twig[54], který dle [44] automaticke escapuje veškeré výstupy z aplikace a ošetřuje také vstupy ve formulářích. Není tedy potřeba přidávat další vrstvy ochrany proti těmto útokům.

[51] Structured Query Language
[52] Cross-site scripting
[53] Application Programming Interface
[54] Dostupné z: `https://twig.symfony.com`

4.5 Autentizace a autorizace

4.5.1 Autentizace

Autentizace je proces, při kterém ověřujeme identitu osoby, která se připojuje do systému. Ve webových aplikacích se využívá nejčastěji kombinace přihlašovacího jména a hesla. V jiných oblastech je možné se setkat také s autentifikací pomocí biometrie (otisk prstu, sken oční duhovky) nebo například pomocí fyzických prostředků (čipová karta, hardwarový klíč).

Do požadované aplikace budou mít přístup pouze pracovníci školy, není tedy potřeba řešit autentizaci nijak složitě a bude vytvořen pouze jeden přístupový „účet“ pro pracovníky školy. V budoucnu však nebude žádný problém s přidáním funkcionality uživatelských účtů pro případ, že by se aplikace dále rozvíjela.

Framework Symfony obsahuje již hotové komplexní řešení pro tyto účely. Jedná se o balík security [45], který je možné využít také zvlášť, tedy mimo tento framework. Veškeré nastavení je uloženo v souboru `security.yaml`. V tom se nachází nastavení metod autentizace, metody odhlášení a je možné do něj přidat také pevné uživatelské účty, čehož je v aplikaci využito. Příklad tohoto konfiguračního souboru je zobrazen na ukázce 4.6.

Ukázka kódu 4.6: Ukázka nastavení autentizace v souboru security.yaml

```
# následuje definice uzivatele lvicata
security:
    providers:
        in_memory:
            memory:
                users:
                    lvicata:
                      password: # zde je ulozen hash hesla
                      roles: 'ROLE_ADMIN'

# následuje nastavení hashovaciho algoritmu, v
# nasem pripade se jedna o bcrypt
encoders:
        Symfony\Component\Security\Core\User\User:
            algorithm: bcrypt
            cost: 12
```

4.5.2 Autorizace

Po úspěšně provedené autentizaci nastává proces autorizace. Jedná se o přidělení práv k přístupu k určitým částem systému. V praxi to u webových aplikací znamená, že autentizovanému uživateli je přidělena uživatelská role, podle které může poté provádět určité operace v systému.

V Symfony se i autorizace konfiguruje v souboru `security.yaml`, příklad nastavení v konfiguračním souboru je zobrazen na ukázce 4.7 [45].

Ukázka kódu 4.7: Ukázka nastavení autorizace v konfiguračním souboru

```
# nasledujici kod znamena, ze pouze uzivatel s pridelenou
# roli ROLE_ADMIN muze pristupovat do vsech casti systemu
# ve ktere zacina adresa za domenou /admin
access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
```

Kontrolovat přístup k určitým funkcionalitám je možné také přímo v kontrolerech. To je umožněno jak anotacemi, tak i zavoláním metody [45]. Oba tyto typy kontroly přístupu jsou ukázány na ukázce 4.8.

Ukázka kódu 4.8: Ukázka nastavení autorizace pomocí anotace

```
class AdminController extends Controller {

    ...

    /**
      * @Route("/admin/edit", name="admin.edit")
      * nasleduje ukazka kontroly pomoci anotace
      * @Security("has_role('ROLE_ADMIN')")
      */
    public function EditAction( ... ) {

    // nasleduje ukazka kontroly prav zavolanim metody
    $this->denyAccessUnlessGranted('ROLE_ADMIN');

    ...
```

V aplikaci bude autorizace velice jednoduchá. Nejedná se o systém do kterého by měli přístup například rodiče (alespoň zatím). V aplikaci tedy bude pouze jedna uživatelská role (administrátor) a po úspěšné autentizaci bude přístupný celý systém.

Je tedy potřeba, aby se každý dostal pouze na přihlašovací obrazovku (cesta „/login“) a do všech ostatních částí systému byl možný přístup až po úspěšném přihlášení. Výsledná konfigurace je předvedena na ukázce 4.9.

Ukázka kódu 4.9: Nastavení autorizace v aplikaci

```
access_control:
    - { path: ^/$, roles: ROLE_ADMIN }
    - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/, role: IS_AUTHENTICATED_FULLY }
```

4.6 Splnění zbývajících funkčních požadavků

V této kapitole je rozebráno splnění funkčních požadavků, které se již netýkají samotné evidence, ale funkcí, které jsou do systému přidány právě na požadavek vedení školy. Jedná se o možnost importu pohledávek tabulkou, možnost importu ABO výpisu, možnost exportování do tabulkového editoru a možnost tisknout sestavy (viz 2.3)

4.6.1 Import tabulky s pohledávkami

Ve skupině „Pohledávky“ v navigačním menu se nachází možnost „Nahrát tabulku pohledávek“. Tato možnost otevře jednoduchý formulář, ve kterém se nachází pole pro název budoucí skupiny pohledávek a výběr souboru na nahrání (návrh obrazovky 4.4).

Po nahrání a úspěšné validaci formuláře se použije balík phpspreadsheet (viz 3.5.3.4). Tento balík rozpozná formát souboru (po úspěšné validaci je jisté, že se jedná o kompatibilní formát) a začne jej číst. Na začátku algoritmu se vytvoří nová skupina pohledávek a poté se postupně vytváří jednotlivé pohledávky dle tabulky.

Na začátku algoritmu dojde k identifikaci sloupců podle prvního řádku (podle symbolů u sazeb, viz 2.6) podle kterého se poté data v nich zpracovávají. Pokud se jedná o sazbu za stravné, hodnota ve sloupci se vždy násobí příslušnou sazbou. Pokud se jedná o sazbu za školné (a ostatní), jako pohledávka se vytvoří přímo částka uvedená ve sloupci.

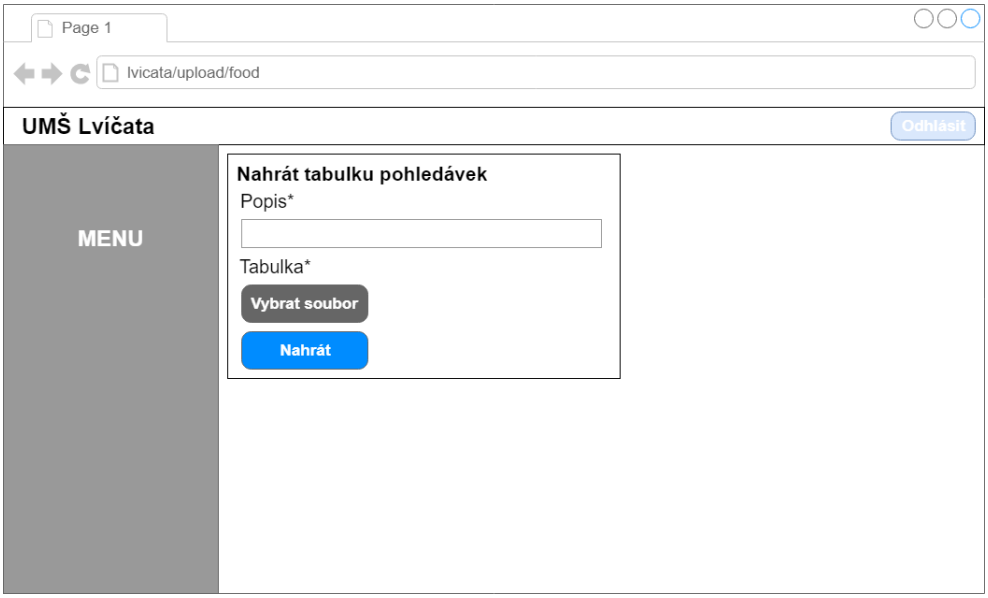
Uživatel je následně přesměrován na seznam skupin pohledávek. Může ručně zpárovat nespárované položky, případně je vymazat a v případě rozsáhlejšího problému vymazat celou skupinu včetně všech pohledávek.

4.6.2 Import ABO souboru

Z uživatelského hlediska je situace a řešení stejné jako u nahrání tabulky s pohledávkami. Ve skupinq „Platby“ v navigačním menu je možnost „Nahrát výpis“. Při této volbě se uživateli zobrazí podobný formulář jako na obrázku 4.4 s tím rozdílem, že chybí možnost přidat název skupiny.

Po nahrání ABO souboru se pomocí balíku Bank Statements (viz 3.5.3.5) tento soubor zpracuje, vytvoří se výpis a do něj se přidají jednotlivé platby nacházející se v souboru. Systém zároveň zpáruje pomocí variabilního symbolu všechny platby, u kterých je to možné.

Po dokončení je uživatel přesměrován na obrazovku s výpisy a stejně jako u pohledávek může importované platby mazat, ručně párovat či vymazat celý výpis včetně všech plateb.



Obrázek 4.4: Návrh obrazovky pro nahrání tabulky s pohledávkami

4.6.3 Export do tabulkového editoru

Tato funkce je již vestavěná v balíku Sonata Admin Bundle (viz 3.5.3.3), není tedy potřeba ji implementovat ručně.

V praxi tato funkce funguje tím způsobem, že si uživatel vyfiltruje libovolná data na kterémkoliv seznamu, vybere možnost „Stáhnout“ a dostane na výběr dostupné formáty. Pro účely této aplikace jsou dostupné pouze formáty `XLS` a `CSV`.

4.6.4 Tisk sestav

S pracovníky školy bylo domluveno, že tisk sestav bude probíhat pomocí již existujícího exportu do tabulkového editoru. Pracovník tedy exportuje vybranou sestavu postupem z předchozí kapitoly a v tabulkovém editoru má poté možnost tuto tabulku vytisknout.

Kapitola 5

Testování

Testování aplikace proběho dle zadání v prostředí UMŠ Lvíčata. Testování se zhostila paní Lucie Mauerová, která je hospodářkou školy a bude tedy s programem také pracovat. Pro účely testování byla identifikována jedna persona a vytvořeny dva základní scénáře.

5.1 Persony

Jedinou personou je pracovník UMŠ Lvíčata. Jedná se o člověka, který již ve škole pracuje a ví tedy, jakým způsobem se aktuálně platby za stravné a školné evidují. Zároveň se jedná o člověka, který byl v kontaktu s autorem po celou dobu vývoje a autor uživatelské prostředí programu této osobě přizpůsoboval. Jedná se o osobu, která má průměrnou znalost práce s počítačem.

5.2 Scénáře

Byly identifikovány dvě základní operace v systému. Jedná se o procesy, které probíhají vícekrát za měsíc a jedná se tak o nejčastější operace v rámci systému, které pracovníci vykonávají. Následuje jejich popis.

- **Nahrání tabulky se stravným** – Pracovník v tabulkovém editoru sepíše útraty žáků za stravování za daný měsíc. Následně se přihlásí do systému, vybere příslušnou položku v navigačním menu a vyplní formulář. Zadá do něj popis daných pohledávek a vybere připravenou tabulku. Formulář odešle a je informován o počtu vložených pohledávek.

- **Nahrání ABO souboru** – Pracovník stáhne z internetového bankovnictví výpis za určité časové období ve formátu ABO. Přihlásí se do aplikace, v navigačním menu vybere příslušnou položku a do formuláře vloží připravený ABO soubor. Po odeslání je pracovník informován o počtu vložených plateb.

5.3 Vyhodnocení

Během testování bylo identifikováno několik drobných problémů. Jednalo se buď o menší chyby, které bránili například editacím některých entit, nebo o špatné překlady některých položek. Tyto drobnosti byly okamžitě při odhalení opravovány.

Oba scénáře byly pracovníkem úspěšně provedeny a systém pracoval dle očekávání.

Celkové dojmy ze systému během testování byly pozitivní. Na základě testování budou doladěny některé překlady a drobnosti týkající se především uživatelského rozhraní. Prototyp aplikace je tedy otestován a nic nebrání nasazení aplikace na produkční server a následnému zahájení testování v ostrém provozu.

Závěr

Závěrem by autor rád zdůraznil, že se podařilo splnit všechny požadavky na systém dané pracovníky školy. Během testování bylo objeveno několik závad na funkčnosti a vzhledu aplikace, které však byly hned opraveny a je tak naplánováno nasazení do ostrého provozu. Zároveň je aplikace připravena na budoucí vývoj, se kterým počítají i v UMŠ Lvíčata

Zhodnocení

V rámci této práce byly analyzovány potřeby Univerzitní základní a mateřské školy Lvíčata a na jejich základě navrhnut a implementován program sloužící k evidenci plateb za stravné a školné. Aplikace umožňuje evidovat žáky, jejich třídy (zařazení), rodiče a především účty, ke kterým je možné přidávat pohledávky a platby.

Hlavním požadavkem na aplikaci byla možnost nahrávat soubory ve formátu ABO. Tento požadavek byl splněn společně s umožněním nahrávání souborů z tabulkového editoru. Tyto tabulky mohou obsahovat jak počty celodenního či půldenního stravného k danému žákovi, nebo sazby za školné. Všechny sazby za stravné je možné editovat a případně je možné i přidat nové. Jak platby, tak i pohledávky se po nahrání daných souborů párují s účty na základě variabilních symbolů.

K implementaci samotné byl využit framework Symfony s několika dalšími dodatečnými balíčky. Tento framework byl zvolen z důvodu jeho rozšířenosti, množství dodatečných balíků a v neposlední řadě také kvůli možnostem budoucího vývoje aplikace. Zvolený framework také řeší zabezpečení vůči nejpoužívanějším útokům a umožňuje práci více pracovníků najednou. Vyřešena byla také autentizace a autorizace pomocí základních možností frameworku.

Po implementaci byl prototyp aplikace otestován přímo v prostředí UMŠ Lvíčata. Na základě testování bylo upraveno několik drobností, ale celkové dojmy byly velice pozitivní. Nic tedy nebrání nasazení aplikace do ostrého provozu, což je také v plánu během léta z důvodu menšího počtu dětí ve škole.

Přínos pro autora

Autor u všech částí této práce aplikoval postupy, které se naučil během studia a vyvinul funkční aplikaci, která bude nasazena do reálného provozu. Práce tak pro něj znamenala nabití mnoha zkušeností z praxe a zdokonalení se v mnoha metodikách a technologiích.

Možnosti budoucího vývoje

Díky použití mnoha standardních a aktuálně využívaných nástrojů není problém do aplikace časem přidávat další funkcionality. Možností rozšíření je velké množství. V případě úspěšného spuštění standardu PSD2, který má za úkol sjednotit rozhranní bankovních API je možné jej implementovat. Dále je možné také vytvořit veřejnou část pro rodiče určenou pro kontrolu útrat žáka.

Rozšiřování tedy nic nebrání. Autor na možné rozšiřování myslel také a kód je psán tak, aby nebylo příliš složité jej rozšířit o nové funkcionality.

Literatura

1. Párování plateb s bankou. *Fakturoid - jednoduchý fakturační program* [online]. ©2009-2018 [cit. 2018-04-11]. Dostupné z: `https://www.fakturoid.cz/podpora/automatizace/parovani-plateb-s-bankou`.
2. JIW - Jídelna pro Windows. *Z-WARE.cz – stravovací, docházkové a přístupové systémy* [online]. ©2016–2017 [cit. 2018-04-13]. Dostupné z: `https://www.z-ware.cz/sites/default/files/2017-01/sw-jiw.pdf`.
3. Program Stravné. *Veřejná informační služba, s.r.o. Plzeň* [online]. ©2014 [cit. 2018-04-13]. Dostupné z: `http://uloziste.visplzen.cz/obchodni_listy/Stravovaci_systemy/Program_Stravne/Program_Stravne.pdf`.
4. Technický popis struktury ABO formátu pro programátory. *Česká spořitelna, a.s.* [online]. 2015 [cit. 2018-04-30]. Dostupné z: `https://www.csas.cz/content/dam/cz/csas/www_csas_cz/dokumenty/produkty/internetove-bankovnictvi/servis-24/ABO_format.pdf`.
5. NEUSTADT, Ila; ARLOW, Jim. *UML 2 a unifikovaný proces vývoje aplikací*. Computer Press, Albatros Media as, 2007. ISBN 978-80-251-1503-9.
6. GORTON, Ian. Essential software architecture. In: Springer Science & Business Media, 2006, s. 6. ISBN 3-540-28713-2.
7. ČÁPKA, David. 4. díl - UML - Doménový model. *itnetwork.cz - Ajťácká sociální síť* [online]. ©2018 [cit. 2018-05-06]. Dostupné z: `https://www.itnetwork.cz/navrh/uml/uml-domenovy-model-diagram`.
8. GitHub Octoverse 2017. *GitHub* [online]. ©2018 [cit. 2018-04-24]. Dostupné z: `https://octoverse.github.com`.
9. ALVES-FOSS, Jim. *Formal syntax and semantics of Java*. Springer Science & Business Media, 1999. ISBN 3-540-66158-1.

10. KEOGH, Jim. *Java bez předchozích znalostí*. Computer Press, Albatros Media as, 2016. ISBN 978-80-251-0839-0.

11. PHP: Hypertext Preprocessor. *PHP Manual* [online]. ©2001-2018 [cit. 2018-05-01]. Dostupné z: `http://php.net`.

12. PHP: New features. *PHP Manual* [online]. ©2001-2018 [cit. 2018-04-15]. Dostupné z: `http://php.net/manual/en/migration70.new-features.php`.

13. Ruby Programming Language. *Ruby Programming Language* [online]. ©2018 [cit. 2018-05-01]. Dostupné z: `https://www.ruby-lang.org/en/`.

14. MySQL | Nejoblíbenější databáze s otevřeným zdrojovým kódem. *Oracle Česká republika* [online]. ©2018 [cit. 2018-05-01]. Dostupné z: `https://www.oracle.com/cz/mysql/index.html`.

15. PostgreSQL. *PostgreSQL* [online]. ©2018 [cit. 2018-05-01]. Dostupné z: `https://postgres.cz/wiki/PostgreSQL`.

16. MONGODB VS MYSQL COMPARISON: WHICH DATABASE IS BETTER? *DA-14 - Web and Mobile App Development Company* [online]. ©2018 [cit. 2018-05-07]. Dostupné z: `https://da-14.com/blog/mongodb-vs-mysql-comparison-which-database-better`.

17. April 2018 Web Server Survey. *Netcraft* [online]. ©2018 [cit. 2018-05-01]. Dostupné z: `https://news.netcraft.com/archives/2018/04/26/april-2018-web-server-survey.html`.

18. About the Apache HTTP Server Project. *The Apache HTTP Server Project* [online]. ©1997-2018 [cit. 2018-05-01]. Dostupné z: `https://httpd.apache.org/ABOUT_APACHE.html`.

19. About the Apache HTTP Server Project. *The Apache HTTP Server Project* [online]. ©1997-2018 [cit. 2018-05-01]. Dostupné z: `https://httpd.apache.org/ABOUT_APACHE.html`.

20. LESLIE, Alexandra. NGINX vs. Apache (Pro/Con Review, Uses, Hosting for Each). *Hosting Advice* [online]. 2018 [cit. 2018-05-07]. Dostupné z: `http://www.hostingadvice.com/how-to/nginx-vs-apache/`.

21. NARAMORE, Elizabeth. *Formal syntax and semantics of Java*. Computer Press, 2006. ISBN 80-251-1073-7.

22. About Symfony Project. *Symfony, High Performance PHP Framework for Web Development* [online]. ©2018 [cit. 2018-04-26]. Dostupné z: `https://symfony.com/about`.

23. Rychlý a pohodlný vývoj webových aplikací v PHP. *Nette Framework* [online]. ©2008-2018 [cit. 2018-04-30]. Dostupné z: `https://nette.org/cs/`.

24. The State of PHP MVC Frameworks in 2017. *SitePoint – Learn HTML, CSS, JavaScript, PHP, Ruby & Responsive Design* [online]. 2017 [cit. 2018-04-30]. Dostupné z: `https://www.sitepoint.com/the-state-of-php-mvc-frameworks-in-2017/`.

25. OTWELL, Taylor. Love beautiful code? We do too. *Laravel - PHP Framework for Web Artisans* [online]. ©2018 [cit. 2018-04-30]. Dostupné z: `https://laravel.com`.

26. KONEČNÝ, Martin. 1. díl - Úvod do Doctrine 2 v Nette frameworku. *itnetwork.cz - Ajťácká sociální síť* [online]. ©2018 [cit. 2018-04-27]. Dostupné z: `https://www.itnetwork.cz/php/nette/doctrine/tutorial-uvod-do-doctrine-2-v-nette-frameworku`.

27. PHP Data Objects. *PHP: Hypertext Preprocessor* [online]. ©2001-2018 [cit. 2018-04-27]. Dostupné z: `http://php.net/manual/en/book.pdo.php`.

28. TICHÝ, Jan. Doctrine 2: úvod do systému. *Zdroják - o tvorbě webových stránek a aplikací* [online]. ©2018 [cit. 2018-04-27]. Dostupné z: `https://www.zdrojak.cz/clanky/doctrine-2-uvod-do-systemu/`.

29. Apache vs Nginx: Practical Considerations. *DigitalOcean* [online]. ©2018 [cit. 2018-05-01]. Dostupné z: `https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations`.

30. Admin Bundle (Symfony Bundles Docs). *Symfony, High Performance PHP Framework for Web Development* [online]. ©2018 [cit. 2018-04-27]. Dostupné z: `https://symfony.com/doc/master/bundles/SonataAdminBundle/index.html`.

31. Welcome to PhpSpreadsheet's documentation. *PhpSpreadsheet* [online]. ©2018 [cit. 2018-04-27]. Dostupné z: `https://phpspreadsheet.readthedocs.io/en/develop/`.

32. Bank Statements. *GitHub* [online]. ©2018 [cit. 2018-04-27]. Dostupné z: `https://github.com/jakubzapletal/bank-statements`.

33. MÁCA, Jindřich. 1. díl - Úvod do Symfony frameworku pro PHP. *itnetwork.cz - Ajťácká sociální síť* [online]. ©2018 [cit. 2018-05-05]. Dostupné z: `https://www.itnetwork.cz/php/symfony/zaklady/uvod-do-symfony-frameworku-pro-phpu`.

34. Controller. *Symfony, High Performance PHP Framework for Web Development* [online]. ©2018 [cit. 2018-05-05]. Dostupné z: `https://symfony.com/doc/current/controller.html`.

35. Twig. *Twig - The flexible, fast, and secure template engine for PHP* [online]. ©2018 [cit. 2018-05-05]. Dostupné z: `https://twig.symfony.com`.

36. Defining and Processing Configuration Values. *Symfony, High Performance PHP Framework for Web Development* [online]. ©2018 [cit. 2018-04-26]. Dostupné z: http://symfony.com/doc/current/components/config/definition.html.

37. Form Types Reference. *Symfony, High Performance PHP Framework for Web Development* [online]. ©2018 [cit. 2018-05-07]. Dostupné z: https://symfony.com/doc/current/reference/forms/types.html.

38. 5. FILTER FIELD DEFINITION. *Sonata Project* [online]. ©2010-2018 [cit. 2018-05-01]. Dostupné z: https://sonata-project.org/bundles/doctrine-orm-admin/master/doc/reference/filter_field_definition.html.

39. Traits. *PHP Manual* [online]. ©2001-2018 [cit. 2018-05-01]. Dostupné z: http://php.net/manual/en/language.oop5.traits.php.

40. OWASP Top 10 - 2017. *OWASP* [online]. ©2018 [cit. 2018-04-28]. Dostupné z: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf.

41. SQL Injection. *OWASP* [online]. ©2016 [cit. 2018-04-28]. Dostupné z: https://www.owasp.org/index.php/SQL_Injection.

42. Cross-site Scripting (XSS). *OWASP* [online]. ©2018 [cit. 2018-04-28]. Dostupné z: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS).

43. VRÁNA, Jakub. PHP okénko: Escapování. *Root.cz - informace nejen ze světa Linuxu* [online]. ©2005 [cit. 2018-04-28]. Dostupné z: https://www.root.cz/clanky/php-escapovani/.

44. Creating and Using Templates. *Symfony, High Performance PHP Framework for Web Development* [online]. ©2018 [cit. 2018-04-28]. Dostupné z: http://symfony.com/doc/current/templating.html.

45. Security. *Symfony, High Performance PHP Framework for Web Development* [online]. ©2018 [cit. 2018-04-29]. Dostupné z: https://symfony.com/doc/current/security.html.

Dodatek A

Seznam použitých zkratek

ABO Automatizované bankovní operace

CRUD Create, read, update, delete

XLSX Formát specifikace Office open XML používaný editorem Microsoft Excel

XML Extensible markup language

PC Personal computer (osobní počítač)

TCP/IP Transmission Control Protocol/Internet Protocol

HTTP Hypertext Transfer Protocol

GPL GNU General Public License

URL Uniform Resource Locator

ORM Object-relational mapping

MVC Model-view-controller

SQL Structured Query Language

XSS Cross-site scripting

API Application Programming Interface

JSON JavaScript Object Notation

HTML HyperText Markup Language

Dodatek B

Instalační příručka

Zdrojové kódy aplikace se nacházejí na přiloženém CD. Je na něm přítomný SQL skript pro vytvoření databáze potřebné k běhu a archiv se zdrojovými kódy aplikace.

Pro spuštění na již připraveném aplikačním serveru je potřeba archiv se zdrojovými kódy rozbalit do příslušného adresáře. Archiv již obsahuje potřebné `.htaccess` soubory pro spuštění na aplikačním serveru Apache. Po rozbalení je potřeba aplikaci naistalovat do produkční konfigurace pomocí následujícího příkazu:

```
composer install --no-dev --optimize-autoloader
```

Následně je potřeba vytvořit pro aplikaci požadované databázové schéma v databázi. To je možné provedením následujících dvou příkazů:

```
php bin/console doctrine:database:create --if-not-exists
php bin/console doctrine:schema:update --force
```

Případně je možné databázi přidat spuštěním přiloženého SQL skriptu přímo v databázovém serveru.

Po vytvoření databáze je nutné do souboru `.env` nastavit správné přihlašovací údaje pro databázi. Jedná se o následující řádek:

```
DATABASE_URL=mysql://username:password@adresa:port/jmeno_databaze
```

Pro spuštění ve vývojářském režimu není potřeba mít nainstalován aplikační server. Stačí provést instalaci včetně vývojářských balíčků následujícím příkazem:

```
composer install
```

Následně je možné spustit testovací server příkazem:

```
php bin/console server:run
```

Dodatek C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `aplikace.zip` ..... zdrojové kódy implementace
  - `BP_Lacny_Jakub_2018.zip` ... zdrojová forma práce ve formátu LaTeX
  - `create-script.sql` ..... SQL skript na vytvoření databáze
- `text` ..... text práce
  - `BP_Lacny_Jakub_2018.pdf` ..... text práce ve formátu PDF