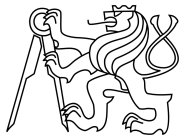
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Informační systém pro správu směn brigádníků
Student:	Martin Zákravský
Vedoucí:	Ing. Petra Pavlíčková, Ph.D.
Studijní program:	Informatika
Studijní obor:	Informační systémy a management
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

1. Analyzujte současná řešení informačních systémů spravování směn brigádníků ve firmách. Na základě analýzy proveďte sběr požadavků na systém od zaměstnavatelů.
2. Zkonzultujte použití vhodné architektury s vedoucím práce, proveďte analýzu a návrh vlastního řešení informačního systému. Zaměřte se na konkurenční výhody navržené aplikace.
3. Zvolte vhodnou technologii a architekturu. Implementujte prototyp informačního systému prostřednictvím standardních postupů softwarového inženýrství.
4. Zdokumentujte navržené řešení a popište pokyny k jeho nasazení. Zhodnoťte implementaci a navrhněte doporučení dalšího vývoje.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 25. ledna 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Informační systém pro správu směn brigádníků

Martin Zákravský

Vedoucí práce: Ing. Petra Pavlíčková, Ph.D.

15. května 2017

Poděkování

Rád bych vyjádřil své díky především vedoucí své práce za její obětavost a snahu. Dále pak svým přátelům a rodině za jejich podporu a konstruktivní připomínky.

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 15. května 2017 .....

Odkaz na tuto práci

Zákravský, Martin. *Informační systém pro správu směn brigádníků.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato bakalářská práce je zaměřena na řešení problému plánování směn a organizace časových možností brigádníků ve firmě. Motivací k výběru tohoto tématu pro mě byla vize příjemnějšího a přehlednějšího prostředí pro organizaci pracovní činnosti zaměstnanců i zaměstnavatele, jakožto i prevence problémů (např. kolize směn, překročení limitu zaměstnanců na směnu) a zabezpečení před neoprávněnými změnami (např. zasahování do směn cizích lidí, neohlášené rušení vlastních směn). Cílem této práce je podrobný návrh, analýza a implementace informačního systému, který bude tento problém řešit. Součástí analýzy bude detailní stanovení požadavků na životní cyklus směn, práva uživatelů a fungování interních procesů. Samotný informační systém bude realizován jako webová aplikace založená na PHP s použitím vhodného frameworku, která bude komunikovat s MySQL databází. Prototyp aplikace bude jakožto hlavní výstup této práce podporovat autentizaci jednotlivých uživatelů, rozlišení uživatelských rolí, manipulaci s jednotlivými směnami podle předepsaných pravidel stanovených v rámci analýzy a přehledné vyobrazení obsazenosti směn.

Klíčová slova plánování směn, brigády, organizace času, komunikace zaměstnanec-zaměstnavatel, webová aplikace

Abstract

This bachelor thesis is dedicated to resolving the issue of planning shifts and organizing time schedules of part-time workers in companies. The motivation for choosing this topic came from a vision of a more pleasant and better arranged environment for work, both for the employee and the employer, as well as for prevention of problems (ie. shift collision, exceeding the limit of workers on a certain shift) and as a security element against unauthorized changes (ie. interfering with other people's shifts, unreported cancelling of user's own shifts). The goal of this thesis is a descriptive design, analysis and implementation of an information system, which will resolve this problem. Part of the analysis will focus on detailed setting of requirements on the shifts' life-cycles, user permissions and internal processes requirements. The information system itself will be realized as a web application based on PHP and using an appropriate framework and it will communicate with a MySQL database. As the main result of this thesis, the prototype of the application will support authentization of individual users, differentiation of individual user roles, manipulation with shifts using the rules specified in the analysis and well-arranged display of the availability of shifts.

Keywords shift planning, part-time jobs, time management, employee-employer communication, web application

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Inspirací pro tuto bakalářskou práci byla zkušenost z pracovního prostředí manuálního testera pro jednu z větších firem působících v České republice v oblasti IT. Testování je pouze jednou z mnoha oblastí, kde se firmám vyplatí najímat brigádníky, a to především díky jejich časové flexibilitě. Oblasti, které z tohoto typu zaměstnanců mohou mít užitek, najdeme v podstatě v každé sféře osobního i profesionálního života.

Podstata tohoto typu pracovního poměru sama implikuje potenciální problémy, které mohou nastat. Toto platí zejména u brigád, kde očekáváme dlouhodobější trvání (nikoliv sezonní či jednorázové práce). Vyšší časová flexibilita implikuje nižší závazek ze strany zaměstnance i zaměstnavatele. Organizace času je tedy klíčová, neboť počty a časové možnosti našich zaměstnanců se mohou velmi rychle měnit. Často spatřujeme systém směnného provozu, který bude hlavním subjektem výzkumu této práce.

Ačkoliv je systém směnného provozu v České republice poměrně rozsáhlý, nalezení vhodného nástroje řešícího tyto problémy, kterým se detailně věnuji v oblasti sběru požadavků, může být velice obtížné. Mnoho nástrojů zanedbává nutnost rozdělení odpovědnosti a přináší oběma stanám - zaměstnanci i zaměstnavateli - možnosti snadné neoprávněné manipulace se směnami. Je také vhodné myslet na uživatelskou přívětivost, která usnadní práci oběma stranám. Vhodná volba nástroje přinese vyšší přehlednost a snadnou dohledatelnost dat - naproti tomu špatná volba se může vymstít nespokojeností zaměstnanců, neefektivitou jejich pracovní činnosti a nejasnostmi v oblasti přidělování zaměstnaneckých odměn. I přesto se zaměstnanci často uchylují k řešením, která jsou v tomto směru riziková.

Ve světle těchto pozorování jsem se rozhodl této problematice věnovat komplexněji. V rámci této práce jsem provedl analýzu běžných potřeb zaměstnavatele i zaměstance, které jsou součástí zmíněného směnného provozu. Tyto potřeby jsem se pokusil najít v již existujících systémech řešení a provedl jsem i návrh a implementaci prototypu vlastního informačního systému, který tyto potřeby bude naplňovat.

Kapitola 1

Cíl práce

První, podpůrná část této bakalářské práce popíše obecné metodiky zvolené pro návrh, analýzu a implementaci informačního systému. Nastíní postupy, kterými se budu v práci ubírat. Dále zde věnuji prostor analýze stávající situace. Identifikuji zde typické problémy liniových managerů - osob, jejiž pracovní náplň z většiny zahrnuje časové rozdělení zaměstnanců, řešení absencí apod. [1] - a sestavím z nich seznam požadavků pro informační systém, který je bude plnit. Následně shromáždím několik existujících řešení pro správu směn brigádníků a zjistím, zda tyto požadavky plní. Zaměřím se zde na prostory pro zlepšení, které pak využiji ve vlastní implementaci.

Ve druhé části práce se začnu věnovat implementaci vlastního informačního systému po teoretické stránce. S pomocí požadavků vycházejících z výsledků první části vypracuji návrh systému tak, aby tyto požadavky plnil. Stanovím přesná pravidla pro uživatele a definuji business procesy, které v takovém systému budou fungovat a probíhat. Popíšu zde některé případy užití.

Ve třetí, stěžejní části práce se budu věnovat dokumentaci samotné implementace po praktické stránce. Zvolím vhodnou technologii a architekturu pro realizaci a vypracuji v ní prototyp informačního systému, který bude organizovat směny brigádníků. Podrobně vysvětlím celou strukturu architektury a její konkrétní náležitosti navazující na plnění požadavků z předchozí sekce. Budu klást důraz na korektní postupy a rozšířitelnost aplikace.

Poslední, čtvrtou část dedikuji pokynům k nasazení a především zhodnocení celého procesu vývoje a jeho případné změny. Navrhnu zde i případná rozšíření a nové funkce informačního systému v budoucnu.

Kapitola 2

Popis problematiky

2.1 Analýza problému

Pro potřeby analýzy definuji několik stran, pro začátek dvě základní: koordinátora a pracovníka (v kontextu práce také „zaměstnance“). Koordinátor zde bude reprezentovat potřeby zaměstnavatele. Jeho povinnost je rozhodování o míře potřeby lidských zdrojů - kolik lidí bude potřeba na daný den a v jakých časových obdobích. Dále tyto potřeby musí naplnit z dobrovolníků, kteří se o daná místa hlásí a dát jim vědět, zda mají či nemají přijít do práce. Další jeho úlohou je řešení akutních a neočekávaných situací z obou stran - musí být připraven na omlouvání směn daných zaměstnanců (v rámci interní politiky firmy počítám s patřičným časovým předstihem a správnou formou) a také je povinnen zaměstnancům dát vědět, pokud se z akutních důvodů daná směna ruší. Má právo uznávat směny zaměstnanců podle vlastního uvážení.

Pracovník je osoba s předepsanou činností práce. Jeho povinností je seznámit se se systémem směn dané firmy, využívat ho a pravidelně sledovat jeho změny. Je vázán internímí pravidly pro omlouvání směn a je poučen o tom, že počet jeho uznaných směn není garantován. Má ovšem právo být delegován výlučně na pracovní činnosti, které jsou součástí jeho pracovní smlouvy. Má právo spravovat si své časové možnosti sám a má poskytnutou přiměřenou ochranu proti manipulaci ze strany koordinátora - nesmí dojít k přesunutí či zapsání nové směny bez jeho souhlasu a nesmí dojít ke zrušení jeho potvrzené směny bez upozornění. Koordinátor je jedinou stranou, která může do jeho směn zasahovat - ostatní pracovníci na to nemají nárok.

V případě větších firem můžeme pracovníky dále dělit dle logických celků. V reálném světě si toto můžeme připodobnit například na samoobsluze: pracovníci lze mohou plnit mimo jiné funkci pokladních či doplňovačů zboží. Můžeme mít několik zaměstnanců, kteří jsou schopni plnit více těchto činností, ovšem nemusí to být všichni (z důvodů právních, zkušenostních a jiných). Pokud máme několik takových celků, záleží na rozhodnutí firmy, jak bude fungovat hierarchie mezi pracovníky. Jednou alternativou je jeden koordiná-

tor (nebo skupina koordinátorů) pro všechny, druhou je konkrétní koordinátor pro konkrétní funkci. Můžeme přistoupit i k více komplexním řešením - koordinátor může působit jako výpomoc v jiné funkci apod.

Pokud plní funkci koordinátora více lidí, jejich pracovní povinnosti v rámci dané zóny působnosti se dělí. V rámci systému směn tak pracovník nekomunikuje s konkrétní osobou, ale podává své žádosti komukoliv z pověřené skupiny lidí.

Funkce koordinátora či administrátora může - ale nemusí - splývat s funkcí osoby, která má na starosti rozdělování finančních odměn. V reálném životě je toto ovšem často citlivým údajem. Dokumenty o výši platu jsou dle § 316 odst. 1 zákoníku práce obsahem osobního spisu, do kterého mohou nahlížet mj. nadřízení zaměstnance, ovšem celkově velmi úzký okruh osob [2]. Při kopírování těchto údajů do aplikace je nutné se ujistit, že citlivost údaje zachováváme, a protože koordinátor nemusí vždy nutně být přímý nadřízený (může se jednat o zaměstnance na stejné pozici, který byl pouze pověřen odpovědností za nějakou funkci), oddělím v rámci této práce tuto roli a nazvu ji rolí účetního.

2.2 Existující řešení

2.2.1 Nededikovaná řešení

Nededikovanými řešeními myslím v kontextu této bakalářské práce taková řešení, která jsou víceúčelová a jejichž primárním účelem není časová organizace brigádníků (nebo časová organizace vůbec). Neelektronickým příkladem mohou být různé papírové rozvrhy, tabulky, kalendáře apod. Ve sféře počítačů to mohou být například tabulkové procesory - Microsoft Excel, Google Sheets...

Tato řešení ač technologicky primitivní mohou být efektivnější než zavádění specializovaného systému, kde by taková změna přinesla spíše komplikace. Opírají se však o důvěru a zodpovědnost na pracovišti - stačí jediná chyba či špatný úmysl a systém selže. V závislosti na volbě řešení můžeme získat různé úrovně interakce - např. Google Sheets nabízí možnost sledování změn a archivace - nicméně vzhledem k nedostatku specializace tato řešení nejsou vhodná pro větší počty účastníků. Zpravidla nejsou schopna nabídnout žádný (nebo jen velmi omezený) systém organizace uživatelů a jejich práv, životního cyklu směn a ochranu před zásahy třetí strany. Funkce revize jsou minimální.

2.2.2 Dedikovaná řešení

Tato sekce se bude zabývat již existujícími informačními systémy, které slouží pro správu směn zaměstnanců.

Plánuj směny

Jedná se o velice jednoduché řešení pro pouhé generování rozpisů směn. Služba působí neaktualizovaně a chaoticky. Evidence docházky zde nefunguje. Tato služba není všepokrývajícím nástrojem a evidentně se nesnaží konkurovat jiným komplexním službám.

Humanity

Tato komerční služba, dříve známá jako ShiftPlanning [3], podporuje komplexní portál pro plánování směn. Podporuje funkce jako vykazování docházky, podporu mobilních zařízení a SMS notifikace o změnách. Směny používají vlastní systém životního cyklu, které jsou schopné rozlišit. Může se zdát, že se jedná o ideální systém, nicméně i toto řešení má své nedostatky.

Největší problém vidím v odlišnosti filozofie tohoto informačního systému. Směny jsou pracovníkům přidělovány, místo aby si je sami vybírali. Pracovník si může daný den označit jako nedostupný - tuto funkci jsem otestoval, a danému koordinátorovi se pak směna, kterou se pokusí přiřadit, červeně zvýrazní. I přesto se dostane pracovníkovi do rozpisu. Systém evidentně funguje na principu, který je neslučitelný s hlavním tématem řešení této bakalářské práce - oprávněná manipulace se směnami a rovnoměrné dělení rolí.

Humanity je distribuován jako služba - není určen pro přímou integraci se systémy firmy. Nabízí API, ale je zřejmé, že řešení není navrženo pro běh ve vlastním intranetu firmy. Řešení je po uplynutí zkušební doby zpoplatněné.

Směnomat

Česká služba podobná službě Humanity. Nabízí pestré množství funkcí, ale trpí i podobnými nedostatky. Podobně jako v Humanity se mi zde podařilo přiřadit směnu v době nedostupnosti daného uživatele pouze s upozorněním. Tento systém je stejně jako Humanity distribuován ve formě předplacené služby, jejíž nedostatky jsem naznačil v sekci o Humanity.

2.3 Přínosy vlastního řešení

Po analýze problematické domény a rešerši existujících řešení dále navrhnu vlastní řešení - interaktivní informační systém s následujícími výhodami:

- distribuce formou nasaditelné aplikace,
- otevřený zdrojový kód,

- striktní systém schvalování a přidělování směn,
- notifikace o každé změně provedené na uživatelových směnách.

Toto řešení bude skloubeno s funkcemi, které již existující systémy podporují - např. možnost zobrazení statistik za uživatele, počítání výplaty... Tím se můj systém stane plnohodnotnou alternativou a poskytne komplexní balíček služeb, který bude schopen pokrýt požadavky náročnějších zaměstnavatelů.

2.4 Vývoj informačního systému

Před samotným technickým řešením svého informačního systému provedu analýzu domény z pohledu funkčního a procesního. Funkční pohled člení podnik jako celek na několik dílčích funkčních oblastí, zatímco procesní pohled sleduje navazující činnosti probíhající ve firmě podle logiky jejich návazností [4]. Funkční pohled bude formován s přihlédnutím na jednotlivé uživatelské role - oddělené funkční celky s různými přístupovými právy. Procesní pohled pak bude zaměřen na proces komunikace mezi jednotlivými uživateli skrz dané role, především v rámci komunikace ohledně směn.

Vývoj systému bude v návaznosti na předchozí analýzu pokračovat výčtem funkčních požadavků. Součástí analýzy budou i takzvané nefunkční požadavky, které se zabývají tím, jak bude aplikace poskytovat kýženou funkcionalitu, spíše než tím, co bude poskytovat [5]. Pro plné zdokumentování domény bude práce následně doplněna doménovým modelem a stavovými diagramy u entit, u kterých to bude zapotřebí.

Samotný vývoj se bude řídit filozofií objektového návrhu software pro snadné napojení na doménový model. Drobné změny v oblasti požadavků na systém ve fázi implementace nejsou vyloučeny. Budu se vyvarovat tzv. principu implementace pomocí „vodopádu“ - princip lineárního přístupu k vývoji software, kde je dodržována pevná sekvence postupů [6]. Má aplikace bude podpůrným business systémem, pro který je nejlepší praktikou agilní přístup [7].

V kontextu zařazení mezi podnikovými informačními systému řadím tuto aplikace mezi tzv. ERM - systém pro implementování technologií pro řízení lidských zdrojů [8].

Kapitola 3

Analýza a návrh systému

3.1 Organizace uživatelů

Vzhledem k interní povaze informací sdíleném v informačním systému bude přístup povolen pouze po přihlášení. Uživatelé budou děleni do projektů. Projekty definujeme jako oddělené, logické celky, které mezi sebou nebudou sdílet žádné informace (s výjimkou uživatelů). Projekty můžeme v praxi použít pro oddělení částí firmy, které spolu nesouvisí a z hlediska plánování směn na sobě nejsou závislé, např. jednotlivé pobočky prodejen. Nic ovšem nebrání tomu, aby jeden uživatel mohl být přiřazen na více projektů.

Další logické celky, na které budou uživatelé děleni, jsou pracovní činnosti. Jak bylo stanoveno v analýze problematiky, činnost je pracovní náplň, které se pracovník bude věnovat po celou dobu směny. Každá směna se tedy váže k jedné, konkrétní činnosti. U zaměstnanců evidujeme, které činnosti jsou oprávněni plnit - podle toho si budou moct zapisovat směny.

Během implementace jsem došel k názoru, že nejlepší řešení bude příslušnost zaměstnanců a koordinátorů k jednotlivým projektům řešit právě přes pracovní činnosti. O zaměstnanci tedy říkám, že patří do projektu A tehdy a jen tehdy, pokud je oprávněn vykonávat alespoň jednu činnost evidovanou pod projektem A (u koordinátora platí stejný princip). Původně mělo toto řešení vyřešit konceptuální smyčku uživatel-činnost-projekt, ovšem dále jsem v rámci implementace došel k potřebě přímé relace uživatel-projekt pro určení tzv. „preferovaného projektu“ (projekt, který uživatel momentálně používá a do kterého je uživatel po přihlášení automaticky přihlášen). Tato relace je ovšem velmi slabá - příslušnost k projektu se podle ní nikde neověřuje, od toho slouží pracovní činnosti.

Administrátor

Administrátor je role správce, který je pověřen technickou stránkou běhu aplikace. Jeho úkolem je správa jednotlivých uživatelů a projektů na abstraktní

úrovni, aniž by vyžadoval nebo využíval znalost interních podmínek jednotlivých projektů (s výjimkou jejich pracovních činností). Jedná se o roli, která může být přiřazena libovolnému uživateli a nevylučuje se s ostatními rolemi.

Administrátor má následující práva:

- správa uživatelů (přidávání, úprava a odebírání),
- správa projektů (přidávání, úprava a odebírání).

Účetní

Účetní je role uživatele oprávněného ke správě financí uživatelů v systému. Tato role není vázaná na konkrétní projekt. Účetní může být zároveň koordinátorem i zaměstnancem v libovolném projektu.

Účetní má následující práva:

- zobrazení a úprava hodinové odměny libovolného uživatele v projektu.

Koordinátor

Koordinátor je role uživatele oprávněného ke správě směn daných činností v daném projektu. Tato role je vázaná na konkrétní projekt a činnosti v daném projektu. Koordinátor může (ale nemusí) být v daném projektu zároveň zaměstnancem.

Pokud je koordinátor zároveň i zaměstnancem v daném projektu a chce spravovat směny týkající se jeho činnosti, pak změny jeho směn nespadají pod běžný proces schvalování a jsou automaticky schváleny. Pokud se naopak rozhodne vlastní směnu pod vlastní pracovní činností zrušit, může tak provést bez nutnosti schvalování.

Koordinátor má následující práva:

- vypisování časových rozvrhů směn,
- správa směn ostatních uživatelů v daném projektu pod jeho pracovními činnostmi - potvrzování, rušení, potvrzování žádostí o rušení a jejich zamítání,
- přepínání odpracovaných směn uživatelů v daném projektu pod jeho pracovními činnostmi mezi stavy odpracovaná a zmeškaná
- volná správa vlastních směn v daném projektu,
- správa uživatelů v daném projektu - přidávání a odebírání pracovních činností, za které je odpovědný.

Zaměstnanec

Zaměstnanec je pracovní jednotka v daném projektu. Má přidělené pracovní činnosti (v kontextu jen „činnosti“), které může vykonávat, a v rámci těchto činností může žádat o směny.

Zaměstnanec má následující práva:

- zobrazit a žádat o směny, které se týkají jeho činností,
- rušit vlastní nepotvrzené směny,
- žádat o zrušení svých potvrzených směn.

3.2 Organizace procesů

Tato sekce se bude zabývat průchodem informací napříč uživatelskými rolemi definovaných v předchozí sekci.

3.2.1 Správa časových možností

Časová možnost (v kontextu této práce jen „možnost“, nebo také „slot“) je pevně daný časový interval, v rámci kterého si zaměstnanci mohou zapisovat směny. Koordinátoři vypisují na pracovní činnosti, kterými jsou pověřeni, možnosti na jednotlivé dny. Každá časová možnost může být omezena shora kapacitou směn pro ni vypsaných. Možnosti se samy o sobě mohou překrývat, ovšem to, zda si zaměstnanci mohou zapisovat směny, které si spolu v rámci těchto možností budou odporovat (zaměstnanec si zapíše například jednu směnu od osmi hodin do dvanácti, a druhou od desíti hodin do čtrnácti), rozhoduje konfigurace daného projektu.

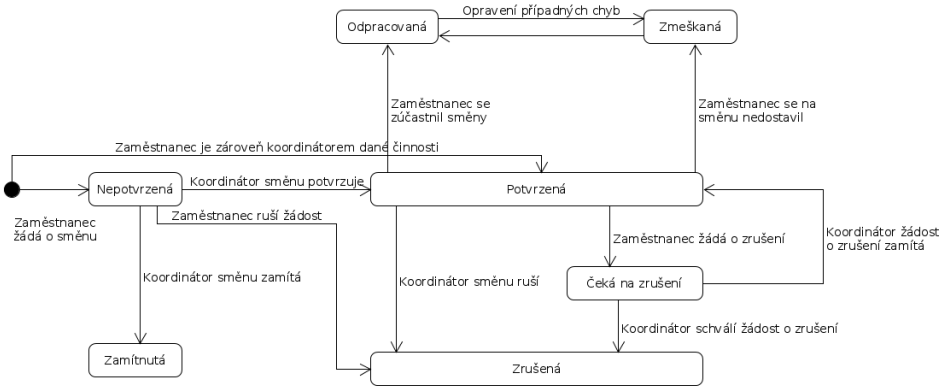
Směna je pracovní závazek mezi uživatelem a konkrétním časovým slotem, který se může nacházet v několika stavech. Pokud hovoříme o existenci směny, samo o sobě to neimplikuje, že daný pracovník bude daný den pracovat. Stavy různých směn a přechody mezi nimi podrobně popisuje životní cyklus směn.

3.2.2 Životní cyklus směn

Diagram 3.1 popisuje všechny možné stavy směn, přechody mezi nimi a důvody těchto přechodů. Aplikace bude obsahovat komponentu, která nedovolí jiné přechody, než které jsou v tomto diagramu definovány. Důvod je zřejmý - monitorování správného chování směn je jednou z klíčových vlastností této aplikace.

3.2.3 Hierarchie entit

Entitu v OOP označujeme jako jednotku - fyzický objekt či abstraktní ideu - která nemá v době návrhu jednoznačné jméno [9]. V naší aplikaci jí budou



Obrázek 3.1: Stavový diagram směny

definovat klasické třídy. Tyto entity jsou popsány v doménovém diagramu 3.2 i s patřičnými relacemi mezi sebou. Na modelu je relace mezi projektem a uživatelem zvýrazněna přerušovanou čarou - toto nespecifikuje žádnou technickou odlišnost relace, ale slouží ke zdůraznění slabé relace, která nedefinuje příslušnost uživatelů k projektům (jak bylo vysvětleno v sekci Organizace uživatelů).

Doménový model nedefinuje uživatelské role administrátora a účetního, neboť tyto role nebudou řešeny skrze atributy tříd.

V modelu bude použita notace uvedena v tabulce v 3.2.3.

M	Mandatory	Atribut je povinný.
O	Optional	Atribut je volitelný.
U	Unique	Atribut je unikátní v rámci entity.

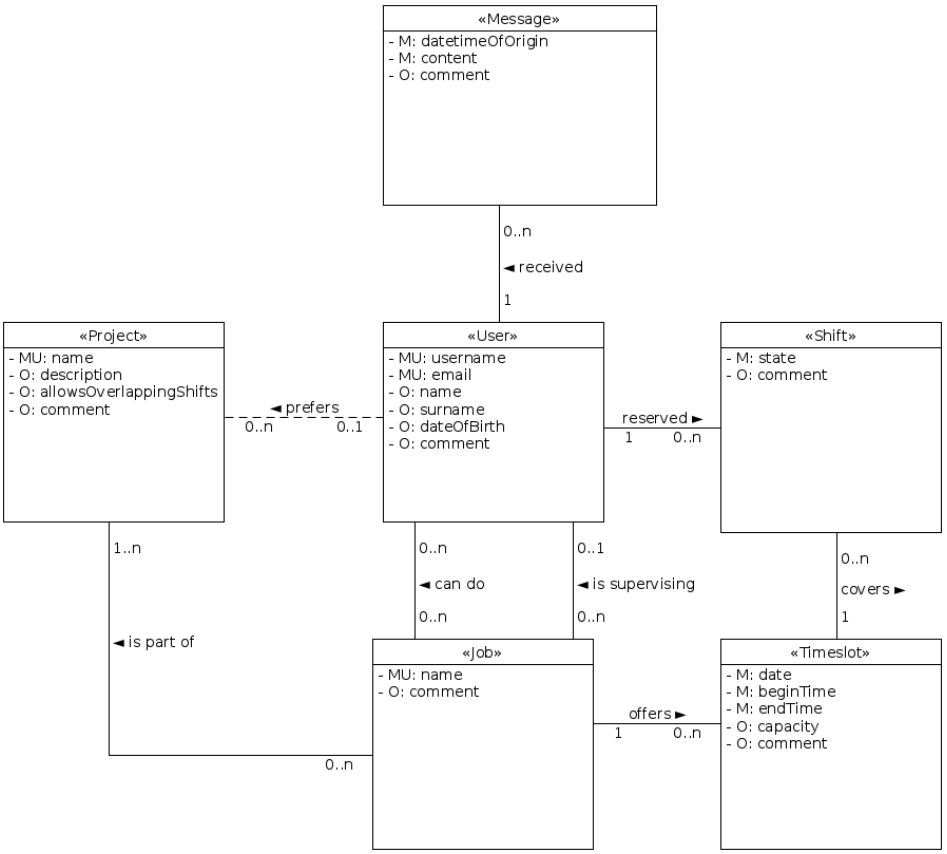
Tabulka 3.1: Atributy doménového modelu

3.2.4 Řešení kolizí entit

Tato sekce se věnuje stanovování pravidel pro závislost entit ve chvíli, kdy jsou na sobě dvě entity závislé a jedna z nich zanikne. Slouží ke zpřesnění analýzy a odstranění případných nedorozumění v částech programu, které nejsou popsány v případech užití.

Smazání uživatele

Uživatel je relačně propojen s notifikacemi, směnami a pracovními činnostmi. Pokud je smazán, obě jeho relace na pracovní činnosti nic nemění. V systému



Obrázek 3.2: Doménový model

zůstanou zachovány i jeho notifikace a směny, které budou nadále evidovány bez majitele - v rámci vedení historie mohou být informace o směnách již neaktivních zaměstnanců velice užitečné.

Smazání pracovní činnosti

Pracovní činnost se pojí na projekt, uživatele a časové sloty. Při jejím smazání to projekt ani uživatele žádným způsobem neovlivní, ovšem není možné ji smazat, dokud jsou na ní závislé nějaké časové sloty. Pro její smazání musí být veškeré tyto relace nejdříve zrušeny.

Smazání časové možnosti

Časová možnost neboli časový slot se pojí se směnami a pracovními činnostmi. Není možné jí smazat ve chvíli, kdy k ní jsou přirazeny nějaké směny - i ve

zrušeném stavu. Důvodem udržování zrušených je hlavní filozofie aplikace - poskytování transparentních informací zaměstnancům i koordinátorům. Mimo to se směna může nacházet i v odpracovaném stavu, což poskytuje základy pro počítání mzdy, a takové informace systém rozhodně musí vždy udržet. Směna si sama o sobě neudržuje informace o čase a datu a potřebuje k tomu daný časový slot - jedná se o informace, které jsou klíčové pro identifikaci směny.

Smazání směny

Jak naznačuje předchozí sekce, smazání směn v aplikaci zcela vylučuji. Pokud má být uvedena jako neaktuální, musí být uvedena do stavu zrušená, popř. zamítnutá. Důsledkem toho je, že po vzniku směny pod určitou pracovní činností tato pracovní činnost nepůjde smazat. Toto chování ovšem při běžném používání aplikace nebude způsobovat problémy - funkce smazání pracovních činností slouží především k eliminaci krátkodobých omylů (překlepy v názvech apod.).

Smazání projektu

Projekt se přímo pojí pouze s pracovními činnostmi. Zde již dovoluji smazání i při existenci pracovních činností, které smazat nelze - u daných pracovních činností se projekt jednoduše vymaže.

Obhajoba smyček

V mém konceptuálním návrhu se vyskytují dvě smyčky - jednak již zmíněná smyčka uživatel-činnost-projekt a jednak smyčka uživatel-činnost-slot-směna.

První smyčku obhájím takto: Vazba mezi uživatelem a projektem se bude vždy určovat podle jeho pracovních činností. Na základě této vazby potom může (ale nemusí) vzniknout pomocná vazba uživatel-preferovaný projekt. Pokud tato vazba zanikne, nic se nestane. Pokud zaniknou vazby uživatel-činnost tak, že je uživatel odebrán z projektu (z pozice zaměstnance i koordinátora), vazba uživatel-projekt bude násilně přerušena. Vzhledem k tomu, že pracovní činnost s přiřazenými uživateli smazat nelze, je tato smyčka obhájena.

Druhá smyčka může vyvolat celou řadu problémů (např. uživateli bude přiřazena směna, která je slotem propojena s činností, na kterou nemá právo). Tyto problémy se dají ošetřit na úrovni interní logiky pečlivým sledováním uživatelských práv při každé komunikaci s aplikací - uživatel si nebude moct zapsat směnu na činnost, na kterou nemá právo, neuvidí sloty, které nepřísluší jeho pracovní činností apod. Veškeré kolize, které by mohly vzniknout přetržením některé relace po jejich vzniknutí jsou popsány v sekcích pro smazání jednotlivých entit výše.

3.3 Sběr požadavků

3.3.1 Funkční požadavky

V této sekci se budu věnovat požadavkům na konkrétní funkce systému.

- *Evidence uživatelů*
  - Aplikaci musí být schopná persistentně uchovávat uživatele a informace o nich. Dále musí umět poskytnout akce a zobrazovat detailní informace o uživatelích v závislosti na roli toho, kdo se o to pokouší.
- *Evidence projektů*
  - Systém musí evidovat projekty, poskytovat do nich přístup oprávněným uživatelům a manipulovat s nimi.
- *Personifikace dle pracovních schopností*
  - Systém musí umět přizpůsobovat data zobrazená uživatelům na základě pracovních činností, které jsou jim přiděleny. Musí být schopen zobrazit pouze taková data, která jsou pro zaměstnance provádějícího konkrétní pracovní činnost relevantní. Musí být také schopen tyto schopnosti na žádost oprávněných uživatelů měnit.
- *Zohlednění pravidel manipulace se směnami*
  - Systém musí dovolit pouze takové operace nad směnami, které jsou definovány uživatelskými oprávněními a životním cyklem směn. Pokud někdo operuje nad cizí směnou (zpravidla koordinátor nad směnou zaměstnance), musí o tom být pracovník dané směny systémem notifikován.
- *Správa finančních odměn*
  - V systému bude pro každého zaměstnance evidována příslušící finanční odměna za vykonanou práci udaná hodinově. Údaj a hodnoty z něj vypočítané může být přístupný pouze zaměstnanci či oprávněným osobám.

3.3.2 Nefunkční požadavky

Výkon a škálovatelnost

Systém neklade žádné zvláštní nároky na výpočetní výkonnost, ani nepředpokládá extrémní vytíženost ze strany uživatelů. Očekávaná zátěž se pohybuje v řádu desítek, maximálně stovek požadavků za hodinu. Maximální reakční doba serveru je v řádu jednotek sekund.

Spolehlivost

Data, která do systému uživatel zadá si musí zachovat strukturu i v případě výpadku. Systém musí být připraven na kolize požadavků uživatelů a používat princip transakcí, aby vyloučil logické nesrovnalosti v datech.

Dostupnost

Očekávaná dostupnost systému je 24 hodin denně 7 dní v týdnu mimo neočekávaných výpadků či pravidelných aktualizací.

Rozšířitelnost

Jakožto systém s otevřeným zdrojovým kódem musí aplikace být postavena na obecně známé struktuře, aby byla snadno rozšířitelná i jinými stranami, než samotným autorem. Musí využívat principu zapouzdření a oddělení jednotlivých komponent a klást důraz na jejich znovupoužitelnost (princip DRY). Měla by používat vícevrstvou architekturu a oddělovat data od samotného kódu. Pro grafické rozhraní by měla být použita obecně známá technologie, která není přímo závislá na samotné výkonnostní části kódu.

Spravovatelnost

Systém neklade žádné nároky na detailní monitoring či komplexní správu běhu. Je požadována manipulace s entitami v úrovni uživatelského prostředí a alternativní přístup k datům i mimo samotnou aplikaci - oddělení datové složky od zbytku aplikace vhodnou formou.

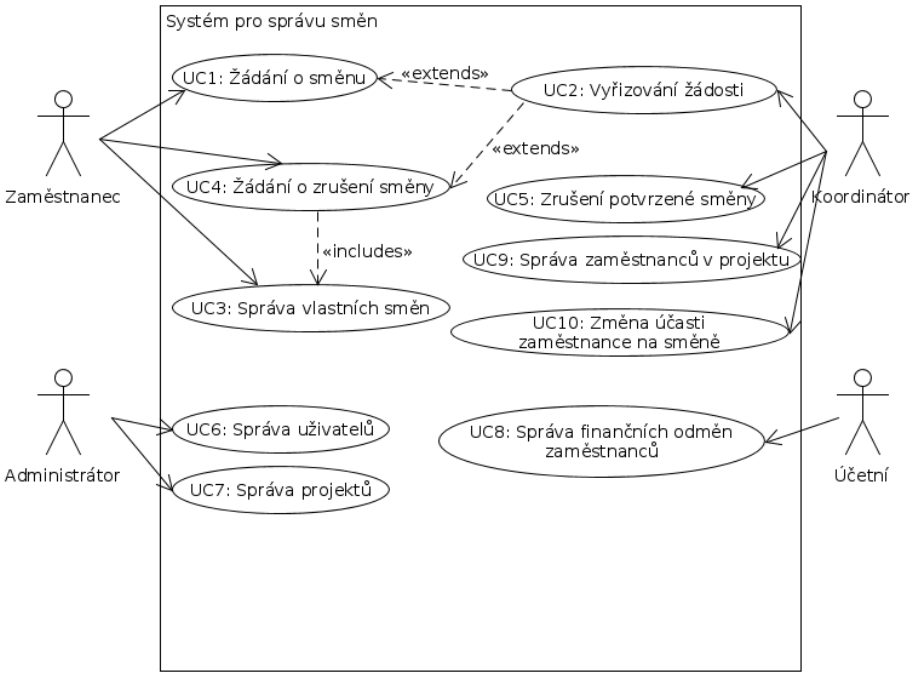
Bezpečnost

Systém musí zvládat základní principy autentizace a autorizace uživatelů, dodržovat konzistenci dat a respektovat uživatelská práva. Systém musí využívat technologii, která bude v případě zájmu uživatele konfigurovatelná pro šifrované spojení (ačkoliv samotná implementace takové bezpečnostní komponenty nebude součástí prototypu).

3.4 Případy užití

3.4.1 Tvorba případů užití

Případy užití neboli UC slouží v tvorbě informačních systémů k zachycení kontraktu zúčastněných stran o chování daného systému [10]. UC jsou klíčovou součástí analýzy. V této sekci je nejprve hierarchicky popíšu v tzv. UC diagramu neboli diagramu užití, který slouží ke shrnutí uživatelů, kteří používají daný systém, a toho, co s ním mohou dělat [11]. V něm budou sepsány případy užití implementované v základním prototypu, které budou nadále do detailů rozebrány.



Obrázek 3.3: Diagram případů užití

Jak je vidět na diagramu 3.3, jednotlivé případy užití popisují některé základní funkční požadavky, např. evidence zaměstnanců či projektů. Některé UC jsou na sobě závislé (relace `«includes»`) - např. uživatel nemůže zažádat o zrušení směny, aniž by prošel sekcí pro správu vlastních směn. Některé UC jsou dále fakultativními rozšířeními jiných případů (relace `«extends»`) - např. zaměstnancovo zažádání o směnu UC1 může (ale nemusí) být následováno vyřízením této žádosti (UC2).

UC1 - Žádání o směnu

Případ užití popisuje průběh vytvoření žádosti zaměstnance o konkrétní směnu.

- **Aktéři**
  - Uživatel
  - Systém
- **Cíle**
  - Cílem je umožnit uživateli vytvořit novou směnu v nepotvrzeném stavu a zaevidovat ji do systému.
- **Vstupní podmínky**
- Zaměstnanec musí být registrovaný a nezablokovaný uživatel v systému.
- K dispozici zaměstnance jsou otevřeny časové sloty pro jeho pracovní činnost s nenaplněnou kapacitou.
- **Pokud má zaměstnanec v daném projektu i roli koordinátora, pak není koordinátorem činnosti, ve které žádá o směnu**[1]
- **Výstupní podmínky**
  - Všechny údaje jsou perzistentně uloženy a pokud byla vytvořena směna, je nad ní možné provést další akce.

Běžná cesta

1. Uživatel se přihlásí do systému pomocí svých přihlašovacích údajů.
2. Systém zobrazí úvodní obrazovku.
3. Uživatel si vybere projekt a přejde do sekce pro zaměstnance
4. Systém zobrazí domovskou stránku sekce pro zaměstnance.
5. Uživatel přejde do sekce plánování směn.
6. Systém zobrazí časový plán.
7. Uživatel si vybere den a časový slot s požadovanou směnou a klikne na tlačítko pro zažádání.
8. Systém provede kontrolu uživatelských práv - přiřazený projekt a pracovní činnost, kontrola proti změnám v mezičase.

[1] Pokud by tomu tak bylo, UC není aplikovatelný. Zaměstnanec má právo na automatické uznání směny, pokud je zároveň koordinátorem dané činnosti.

9. Systém provede kontrolu přístupnosti směny - otevřený časový slot, nenaplněná kapacita a validní přechod mezi stavy směn.
10. Systém vytvoří novou směnu a uvede ji do stavu „nepotvrzená“.
11. Systém informuje uživatele o úspěšném procesu tak, že dané tlačítko pro žádání o směnu změní na tlačítko pro zrušení žádosti.

Alternativní cesty Alternativní cesty pro tento UC nejsou.

Výjimky

8.1 Uživateli byly v mezičase odebrány práva na pracovní činnost či projekt.

- Uživateli je zobrazena chybová hláška.

9.1 Směna byla v mezičase zaplněna nebo byl časový slot uzavřen.

- Uživateli je zobrazena chybová hláška.

UC2 - Vyřizování žádosti

Případ užití popisuje vyřizování žádosti týkající se směn od zaměstnanců na straně koordinátora (slučuje potvrzování a zamítání směn či žádosti o jejich zrušení). Tento UC předpokládá, že v minulosti byl alespoň jednou splněn UC1, případně následován UC4.

- **Aktéři**
  - Koordinátor
  - Systém
- **Cíle**
  - Cílem je umožnit koordinátorovi vyřídit žádost zaměstnance.
- **Vstupní podmínky**
  - Koordinátor musí být registrovaný a nezablokovaný uživatel v systému.
  - Koordinátor má žádosti k vyřízení (žádosti o směnu či o její zrušení).
- **Výstupní podmínky**
  - Všechny údaje jsou perzistentně uloženy a směny, které byly předmětem vyřizování, jsou k dispozici k revizi zaměstnanců.
  - Zúčastněný zaměstnanec byl notifikován o provedení změn.

Běžná cesta

1. Uživatel se přihlásí do systému pomocí svých přihlašovacích údajů.
2. Systém zobrazí úvodní obrazovku.
3. Uživatel si vybere projekt a přejde do sekce pro koordinátory.
4. Systém zobrazí domovskou stránku sekce pro koordinátory.
5. Uživatel přejde do sekce žádostí k vyřízení.
6. Systém zobrazí žádosti o směny a žádosti o zrušení směn spadajících pod činnosti, které uživatel koordinuje.
7. Uživatel si vybere žádost k vyřízení a pomocí tlačítek rozhodne, zda ji schválí nebo zamítne.
8. Systém provede kontrolu uživatelských práv - přiřazený projekt a pracovní činnost, kontrola proti změnám v mezičase.
9. Systém provede kontrolu přístupnosti směny - validní přechod mezi stavy směn.
10. Systém uvede směnu do kýženého stavu - přípustné stavy v závislosti na typu žádosti a rozhodnutí koordinátora jsou „potvrzená“ a „zrušená“.
11. Systém odešle notifikaci zaměstnanci, kterého se směna týká, o změně stavu směny.

Alternativní cesty

5.1.1 Koordinátor přejde do sekce plánování. UC pokračuje krokem 6.

5.1.2 Koordinátor vybere konkrétní den, o kterém ví, že tam jsou nevyřízené žádosti. UC pokračuje krokem 6.

Výjimky

8.1 Uživateli byly v mezičase odebrány práva na pracovní činnost či projekt.

- Uživateli bude zobrazena chybová hláška.

9.1 Směna byla v mezičase uvedena do jiného stavu jiným koordinátorem.

- Uživateli bude zobrazena chybová hláška.

UC3 - Správa vlastních směn

Případ užití zachycuje kroky potřebné k zobrazení informací o vlastních, nezrušených a nezamítnutých směnách pro zaměstnance. Případ užití je rozšiřitelný provedením dalších akcí nad směnami dle přání uživatele.

- **Aktéři**
  - Zaměstnanec
  - Systém
- **Cíle**
  - Cílem je poskytnout uživateli informace o jeho nadcházejících směnách a zprostředkovat rozhraní pro provádění dalších akcí nad nimi.
- **Vstupní podmínky**
  - Zaměstnanec musí být registrovaný a nezablokovaný uživatel v systému.
  - Zaměstnanec má v systému evidované nějaké směny v potvrzeném stavu.
- **Výstupní podmínky**
  - Všechny údaje, o které zaměstnanec zažádal, jsou zobrazeny na obrazovce.

Běžná cesta

1. Uživatel se přihlásí do systému pomocí svých přihlašovacích údajů.
2. Systém zobrazí úvodní obrazovku.
3. Uživatel si vybere projekt a přejde do sekce pro zaměstnance.
4. Systém zobrazí domovskou stránku sekce pro zaměstnance.
5. Uživatel přejde do sekce plánování.
6. Systém zobrazí časový plán a volby pro typy zobrazení - agenda, měsíční a denní. Také umožní zvolit mezi zobrazením časových slotů nebo směn.
7. Uživatel zvolí zobrazení směn ve formě agendy.
8. Systém zobrazí seznam všech směn v daném časovém období.

Alternativní cesty

7.1 Pokud uživatel chce zobrazení pro konkrétní den, zvolí pouze zobrazení směn.

7.1.1 Uživatel zvolí denní zobrazení a prokliká se na požadovaný den pomocí směrových tlačítek. UC pokračuje krokem 8.

7.1.2 Uživatel zvolí měsíční zobrazení, vybere si měsíc a konkrétní den. UC pokračuje krokem 8.

Výjimky V rámci běžného používání neočekávám v tomto UC žádné výjimky.

UC4 - Žádání o zrušení směny

Případ užití popisuje kroky zaměstnance, které musí zaměstnanec provést pro zažádání o zrušení již potvrzené směny. Případ předpokládá úspěšný průchod scénářem UC3 a navazuje na něj.

- **Aktéři**
  - Zaměstnanec
  - Systém
- **Cíle**
  - Cílem je vytvořit a zaevidovat žádost o zrušení směny.
- **Vstupní podmínky**
  - Zaměstnanec musí být registrovaný a nezablokovaný uživatel v systému.
  - Zaměstnanec má v systému evidované nějaké směny v potvrzeném stavu.
  - Zaměstnanec je na úspěšném konci scénáře UC3.
- **Výstupní podmínky**
  - Všechny údaje jsou perzistentně uloženy a pokud byla žádost vytvořena, je nad ní možné provést další akce.

Běžná cesta

1. Uživatel si vybere směnu, kterou chce zrušit a klikne na tlačítko pro zažádání o zrušení.
2. Systém provede kontrolu uživatelských práv - přiřazený projekt a pracovní činnost, kontrola proti změnám v mezičase.
3. Systém provede kontrolu přístupnosti směny - validní přechod mezi stavy směn.
4. Systém uvede směnu do kýženého stavu „čeká na zrušení“ a informuje uživatele o úspěšně provedené činnosti.

Alternativní cesty Alternativní cesty pro tento UC nejsou.

Výjimky

2.1 Uživateli byla v mezičase odebrána práva

- Uživateli je zobrazena chybová hláška.

3.1 Koordinátor v mezičase směnu zrušil.

- Uživateli je zobrazena chybová hláška.

UC5 - Zrušení potvrzené směny

- **Aktéři**
  - Koordinátor
  - Systém
- **Cíle**
  - Cílem případu je zrušení již předtím potvrzené zaměstnanecké směny.
- **Vstupní podmínky**
  - Koordinátor musí být registrovaný a nezablokovaný uživatel v systému.
  - Koordinátorovy pracovní činnosti evidují nějaké směny v potvrzeném stavu.
- **Výstupní podmínky**
  - Všechny údaje jsou perzistentně uloženy a pokud byla směna zrušena, je k dispozici zaměstnanci k revizi.
  - Zúčastněný zaměstnanec byl notifikován o provedení změn.

Běžná cesta

1. Uživatel se přihlásí do systému pomocí svých přihlašovacích údajů.
2. Systém zobrazí úvodní obrazovku.
3. Uživatel si vybere projekt a přejde do sekce pro koordinátory.
4. Systém zobrazí domovskou stránku sekce pro koordinátory.
5. Uživatel si zobrazí plán, přepne do zobrazení směn a pomocí navigačních prvků se odkáže na den, ve kterém se nachází směna, kterou chce zrušit.
6. Systém zobrazí všechny směny evidované pro daný den.
7. Uživatel klikne na tlačítko pro zrušení směny.
8. Systém provede kontrolu uživatelských práv - přiřazený projekt a pracovní činnost, kontrola proti změnám v mezičase.
9. Systém provede kontrolu přístupnosti směny - validní přechod mezi stavy směn.
10. Systém uvede směnu do stavu „zrušená“
11. Systém odešle notifikaci zaměstnanci, kterého se směna týká, o změně stavu směny.

Alternativní cesty Alternativní cesty pro tento UC nejsou.

Výjimky

8.1 Uživateli byly v mezičase odebrány práva na pracovní činnost či projekt.

- Uživateli bude zobrazena chybová hláška.

9.1 Směna byla v mezičase zrušena jiným koordinátorem.

- Uživateli bude zobrazena chybová hláška.

UC6 - Správa uživatelů

Případ užití popisuje kroky potřebné k administračním funkcím uživatele určeným pro správu uživatelů.

- **Aktéři**
  - Administrátor
  - Systém

- **Cíle**
  - Cílem tohoto případu je zpřístupnit administrátorovi informace o uživatelích systému a poskytnout rozhraní pro úpravu jejich údajů, přiřazení k pracovním činnostem a jejich smazání ze systému.
- **Vstupní podmínky**
  - Administrátor musí být registrovaný a nezablokovaný uživatel v systému.
- **Výstupní podmínky**
  - Uživateli byly předány potřebné informace spolu s příslušným rozhraním.

Běžná cesta

1. Uživatel se přihlásí do systému pomocí svých přihlašovacích údajů.
2. Systém zobrazí úvodní obrazovku.
3. Uživatel přejde do sekce pro administrátory - volba projektu není zapotřebí, neboť role administrátora není vázaná na konkrétní projekt.
4. Systém zobrazí domovskou stránku sekce pro administrátory.
5. Uživatel vybere možnost správy uživatelů.
6. Systém zobrazí stránkovaný seznam všech uživatelů systému. K dispozici jsou základní informace, včetně uživatelského jména, reálného jména a příjmení. U každého uživatele administrátor najde rozhraní pro jejich úpravu (která zahrnuje i přiřazování pracovních činností) a smazání.

Alternativní cesty Alternativní cesty pro tento UC nejsou.

Výjimky V rámci běžného používání neočekávám v tomto UC žádné výjimky.

UC7 - Správa projektů

Případ užití popisuje kroky potřebné k administračním funkcím uživatele určeným pro správu projektů.

- **Aktéři**
  - Administrátor
  - Systém

- **Cíle**
  - Cílem tohoto případu je zpřístupnit administrátorovi informace o projektech v systému a poskytnout rozhraní pro úpravu jejich údajů a smazání ze systému.
- **Vstupní podmínky**
  - Administrátor musí být registrovaný a nezablokovaný uživatel v systému.
- **Výstupní podmínky**
  - Uživateli byly předány potřebné informace spolu s příslušným rozhraním.

Běžná cesta

1. Uživatel se přihlásí do systému pomocí svých přihlašovacích údajů.
2. Systém zobrazí úvodní obrazovku.
3. Uživatel přejde do sekce pro administrátory - volba projektu není zapotřebí, neboť role administrátora není vázaná na konkrétní projekt.
4. Systém zobrazí domovskou stránku sekce pro administrátory.
5. Uživatel vybere možnost správy projektů.
6. Systém zobrazí stránkovaný seznam všech projektů v systému. K dispozici jsou základní informace, včetně názvu projektu a popisu. U každého projektu administrátor najde rozhraní pro jejich úpravu a smazání.

Alternativní cesty Alternativní cesty pro tento UC nejsou.

Výjimky V rámci běžného používání neočekávám v tomto UC žádné výjimky.

UC8 - Správa finančních odměn zaměstnanců

Případ užití popisuje kroky, které vykonává účetní pro plnění své funkce správce finančních odměn.

- **Aktéři**
  - Účetní
  - Systém

- **Cíle**
  - Cílem případu je poskytnout účetnímu informace o finančních odměnách zaměstnanců a rozhraní pro jejich úpravu.
- **Vstupní podmínky**
  - Administrátor musí být registrovaný a nezablokovaný uživatel v systému.
- **Výstupní podmínky**
  - Uživateli byly předány potřebné informace spolu s příslušným rozhraním.

Běžná cesta

1. Uživatel se přihlásí do systému pomocí svých přihlašovacích údajů.
2. Systém zobrazí úvodní obrazovku.
3. Uživatel přejde do sekce pro účetní - volba projektu není zapotřebí, neboť role účetního není vázaná na konkrétní projekt.
4. Systém zobrazí soupis všech uživatelů v systému spolu s jejich finančními odměnami, které jsou cenzurovány překrývacím okénkem (mizícím na kliknutí) a tlačítkem pro odeslání jejich úprav.

Alternativní cesty Alternativní cesty pro tento UC nejsou.

Výjimky V rámci běžného používání neočekávám v tomto UC žádné výjimky.

UC9 - Správa zaměstnanců v projektu

Případ užití zachycuje kroky potřebné ke správě uživatelských činností v rámci projektu.

- **Aktéři**
  - Koordinátor
  - Systém
- **Cíle**
  - Cílem je zprostředkovat koordinátorovi daného projektu informace a rozhraní potřebné k přerozdělení pracovních činností zaměstnanců v projektu.

- **Vstupní podmínky**
  - Koordinátor musí být registrovaný a nezablokovaný uživatel v systému.
- **Výstupní podmínky**
  - Uživateli byly předány potřebné informace spolu s příslušným rozhraním.

UC10 - Změna účasti zaměstnance na směně

Případ užití zachycuje kroky potřebné k přepnutí odpracované zaměstnancovy směny do zmeškaného stavu.

- **Aktéři**
  - Koordinátor
  - Systém
- **Cíle**
  - Cílem případu je označení odpracované zaměstnancovy směny jako zmeškané.
- **Vstupní podmínky**
  - Koordinátor musí být registrovaný a nezablokovaný uživatel v systému.
  - Koordinátorovy pracovní činnosti evidují nějaké odpracované směny v potvrzeném stavu.
- **Výstupní podmínky**
  - Všechny údaje jsou perzistentně uloženy a pokud byla směna přepnuta, je k dispozici zaměstnanci k revizi.
  - Zúčastněný zaměstnanec byl notifikován o provedení změn.

Běžná cesta

1. Uživatel se přihlásí do systému pomocí svých přihlašovacích údajů.
2. Systém zobrazí úvodní obrazovku.
3. Uživatel si vybere projekt a přejde do sekce pro koordinátory.
4. Systém zobrazí domovskou stránku sekce pro koordinátory.

5. Uživatel si zobrazí plán, přepne do zobrazení směn a pomocí navigačních prvků se odkáže na den, ve kterém se nachází směna, kterou chce označit jako zmeškanou.
6. Systém zobrazí všechny směny evidované pro daný den.
7. Uživatel klikne na tlačítko pro přepnutí směny do zmeškaného stavu.
8. Systém provede kontrolu uživatelských práv - přiřazený projekt a pracovní činnost, kontrola proti změnám v mezičase.
9. Systém provede kontrolu přístupnosti směny - validní přechod mezi stavy směn.
10. Systém uvede směnu do stavu „zmeškaná“
11. Systém odešle notifikaci zaměstnanci, kterého se směna týká, o změně stavu směny.

Alternativní cesty Alternativní cesty pro tento UC nejsou.

Výjimky

8.1 Uživateli byly v mezičase odebrány práva na pracovní činnost či projekt.

- Uživateli bude zobrazena chybová hláška.

9.1 Směna byla v mezičase označena jako zmeškaná jiným koordinátorem.

- Uživateli bude zobrazena chybová hláška.

Běžná cesta

1. Uživatel se přihlásí do systému pomocí svých přihlašovacích údajů.
2. Systém zobrazí úvodní obrazovku.
3. Uživatel si vybere projekt a přejde do sekce pro koordinátory.
4. Systém zobrazí domovskou stránku sekce pro koordinátory.
5. Uživatel zvolí možnost správy uživatelů.
6. Systém zobrazí seznam všech zaměstnanců přiřazených k pracovním činnostem v projektu. U každého uživatele jsou zobrazeny základní údaje (uživatelské jméno, reálné jméno a příjmení...) a tlačítko pro úpravu.
7. Uživatel si zvolí uživatele a klikne na tlačítko pro úpravu.
8. Systém zobrazí formulář pro přidání a odebrání pracovních činností, které je na projektu oprávněn spravovat.

Alternativní cesty Alternativní cesty pro tento UC nejsou.

Výjimky V rámci běžného používání neočekávám v tomto UC žádné výjimky.

Kapitola 4

Implementace

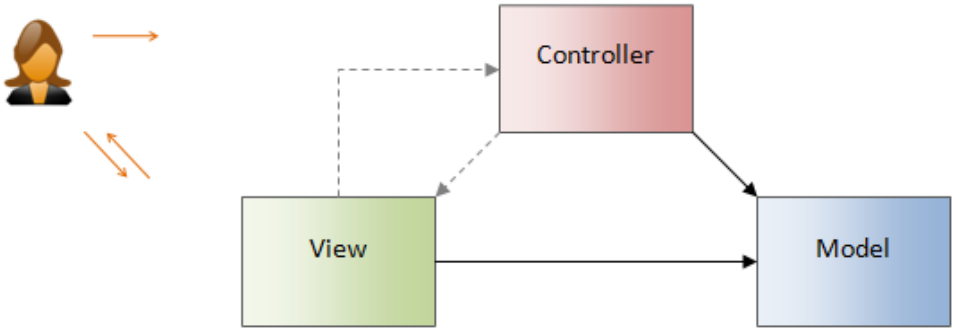
4.1 Volba technického řešení

Rozhodl jsem se systém koncipovat jako webovou aplikaci - systém bude běžet jako serverový program a bude nabízet rozhraní přístupné z webového prohlížeče. To, zda bude aplikace přístupná volně nebo pouze z intranetu bude záviset na rozhodnutí firmy, která se rozhodne daný systém využívat. Toto řešení přinese několik výhod:

- Aplikace bude ovladatelná téměř z libovolného počítače připojeného k internetu (popř. intranetové síti, kde daný systém bude funovat).
- Aplikace bude pro své ovládání vyžadovat pouze webový prohlížeč (bližší nároky jsou uvedeny v konkrétně použitých nástrojích).
- Pro nasazení v produkci bude snadné implementovat HTTPS protokol, který umožní bezpečnou komunikaci (toto prototyp nebude aplikovat).

Aplikace bude navržena ve třech vrstvách pomocí MVC architektury. Samotný kód bude tak rozdělen na tři logické celky - jeden z nich bude odpovědný za ukládání dat, druhý za poskytování rozhraní pro uživatele a třetí za logiku interních operací. Uživatel provede žádost pomocí komponenty View, která jí odešle ke zpracování komponentě Controller. Odpověď systému vyhodnotí komponenta Controller ve spolupráci s komponentou Model a vrátí data v podobě View.

V následujících sekcích podrobně popíšu nástroje, které budu využívat v jednotlivých komponentách. Některé komponenty vyžadují zachování licencí, které jsou v projektu přiloženy v souboru `licence.txt`.



Obrázek 4.1: Ilustrace MVC architektury [12]

4.1.1 Komponenta Model

MySQL

MySQL je velmi rozšířený open-source DBMS systém vlastněný společností Sun Microsystems. Je poskytován zdarma. Používá relační databázový model, se kterým se komunikuje pomocí jazyka SQL. V interakci s PHP se jedná o nejpopulárnější používaný databázový systém. [13]

Tento databázový model sám o sobě nám zaručí standardní přístup k datům (SQL), možnost používání transakcí a zámků, které budou klíčové při zaručování konzistence dat, a možnost jednoduchého zálohování.

Samotná databáze nebude součástí prototypu, ale prototyp a pokyny k nasazení budou navržen tak, aby spolupracoval právě s tímto typem DBMS.

Doctrine

Doctrine je skupina několika PHP knihoven zaměřených na ORM, které umožňuje vytvořit prostředníka mezi relační databází a objektově orientovanou aplikací. Využívá koncepty dalšího populárního ORM software Hibernate určeného pro jazyk Java a adaptuje je pro jazyk PHP. Je poskytován zdarma a dobře spolupracuje s frameworkem Symfony, kterému se budu věnovat níže. [14]

4.1.2 Komponenta View

Twig

Twig je šablonovací nástroj, který slouží jako nadstavba pro PHP. Je vydáván jako open-source produkt pod BSD licencí.[15]

Použitím tohoto nástroje dosáhneme vysoké znovupoužitelnosti jednotlivých komponent GUI naší aplikace - umožní nám lépe oddělit složky, které mají některé části systému společné a dosáhnout tak plnění principu DRY.

JavaScript, jQuery a AJAX

Pro interakci s uživatelem budeme využívat JavaScriptu - dynamického programovacího jazyka pro HTML a web [16]. Tento programovací jazyk nám umožňí dynamické změny v aplikaci bez toho, aniž by uživatel musel obnovovat stránku. To zaručí větší pohodlí a interaktivitu. Zejména pro účely kompatibility mezi prohlížeči a zjednodušení kódu využijeme knihovnu jQuery - populární nadstavbu JavaScriptu používanou i v mezinárodních společnostech, jako je Google, Microsoft či IBM [17].

AJAX není sám o sobě programovacím jazykem - je to název pro technologii zpřístupňující serverová data již načtené webové stránce [18]. Bez této technologie bychom mohli JavaScriptem pouze upravovat lokální rozhraní uživatele, ale s jeho použitím budeme moct ovlivňovat data uložená na serveru, aniž by uživatel odešel z aktuální stránky.

Má aplikace vyžaduje mít v prohlížeči zapnutý Javascript. Pokud to tak není, uživateli se zobrazí varovná hláška. Jedná se o jeden z mála požadavků na klientský prohlížeč.

HTML a CSS

HTML a CSS (spolu s JavaScriptem) tvoří základní kameny technologií pro webové vývojáře [19]. Jedná se o standardní technologie používané pro zobrazování webových stránek. Pro účely této aplikace budou použity standardy HTML 5 a CSS 3.

Má aplikace bude využívat některé komponenty HTML 5 pro validaci polí, které se nemusí ve starších prohlížečích správně validovat. To ovšem není problém, protože aplikace bude veškerý uživatelský vstup validovat i na serveru.

Ikony

V rámci uživatelského prostředí jsem na některých místech využíval ikony z balíčku Iconic - responzivní SVG ikony pro webové vývojáře používané na tisících stránkách již pět let [20].

4.1.3 Komponenta Controller

PHP

PHP (prehypertextový procesor) je serverový, skriptovací jazyk, který je běžně používaný a nabízený zdarma jako alternativa k proprietárním technologiím, jako je např. Microsoft ASP [21]. V mém informačním systému bude hlavním používaným programovacím jazykem.

Symfony

Symfony je sada znovupoužitelných PHP komponentů a zároveň framework pro tvorbu PHP aplikací [22]. Jeho volba přináší vysokou kompatibilitu s ostatními zvolenými nástroji.

V rámci Symfony budu používat také nástroj FOSUserBundle - pomocnou knihovnu pro autentizaci a spravování uživatelů. Tento nástroj není přímým dílem autorů frameworku Symfony.

4.2 Persistence dat

4.2.1 Návrh a mapování

V souladu s principy objektového programování a zároveň s principy relační databáze bude ORM realizováno následujícím způsobem: Jednolivé enkapsulované entity budou definovány prostřednictvím standardních PHP tříd s Doctrine anotacemi, které pak vytvoří podklady pro schéma, které Doctrine nahraje a namapuje do samotné databáze ve formě tabulek. V těchto třídách budou také definovány metody pro získávání dat jednotlivých entit (tzv. gettery a settery), některá integritní omezení a pomocné anotace pro výchozí nastavení řazení.

Definovány budou následující entity:

- *User* - entita označující uživatele - jeho práva, osobní údaje…,
- *Project* - entita označující projekt - jeho název, popis, kapacita…,
- *Timeslot* -jednotlivé časové možnosti sloužící k pokrývání směn,
- *Shift* - jednotlivé směny v libovolném stavu,
- *Job* - pracovní činnosti, které jsou uživatelé schopni provádět nebo za které jsou odpovědni v rámci koordinátorské činnosti,
- *Message* - pomocná entita pro zobrazení notifikací o posledních změnách směn uživatele - v prototypu použita pouze pro potřeby zaměstnance, nikoliv koordinátora.

Mimo entit jsou součástí modelu také tzv. repozitáře. Repozitáře jsou třídy zpřístupňující data z databáze při požadavku na složitější dotazy. Pokud budeme chtít např. získat nefiltrovaný seznam všech směn jednoho zaměstnance, můžeme toho snadno dosáhnout pomocí metod třídy entity *User*. Pokud ovšem budeme chtít aplikovat složité filtrování a řazení (např. pouze směny v budoucnosti v určitém stavu týkající se určitých pracovních činností seřazené dle času…), je vhodné tuto činnost delegovat právě na model. Nabízí se zde použití nástroje DQL - dotazovacího jazyka určeného speciálně pro Doctrine, které se soustředí spíše na objektové schéma, než schéma relační [23]. Svým

vzhledem připomíná SQL a umožňuje vytvářet komplexní dotazy, které jsou pak následně interně převedeny na SQL optimalizovány. To přináší výhodu z hlediska efektivity - SQL a jeho vestavěný optimalizér jsou schopny vybrat nejefektivnější způsob sběru dat výběrem několika možných způsobů a porovnáním jejich nároků na prostředky [24]. PHP nativně nenabízí příliš mnoho způsobů jak zacházet s kolekcemi dat a implementace vlastní kolekce či použití nástrojů třetí strany by bylo zbytečným krokem.

4.2.2 Databázový plánovač

Většina operací, které jsou na směnách v aplikaci prováděny, vzniká na straně uživatele, ale jedna konkrétní není závislá na uživatelské činnosti - přepínání směn do odpracovaného stavu. Pro koordinátora by mohlo být manuální přepínání odpracovaných směn uživatelů zbytečnou prací, které za něj aplikace dokáže udělat sama (to ovšem koordinátorovi nebere možnost následně toto rozhodnutí aplikace změnit).

Databázi jsem nastavil tak, aby jednou za určitou časovou jednotku (ve výchozím nastavení každých 10 sekund) přepnula všechny směny, které se již nachází v minulosti a zároveň jsou označeny jako „potvrzené“ do stavu „odpracované“. Pokud se zaměstnanec na směnu nedostaví, koordinátor může danou konkrétní směnu přepnout manuálně do stavu „zmeškaná“ a plánovač ji dále bude ignorovat.

Automatické přepínání se netýká směn, u kterých bylo zažádáno o zrušení - v realitě mohlo k zažádání dojít brzy před začátkem směny a takové chování by mohlo mylně způsobit uznání směny, která měla být následně zrušena. Pokud je směna zpětně uvedena do potvrzeného stavu, plánovač jí automaticky přepne spolu s ostatními.

4.3 Interní logika

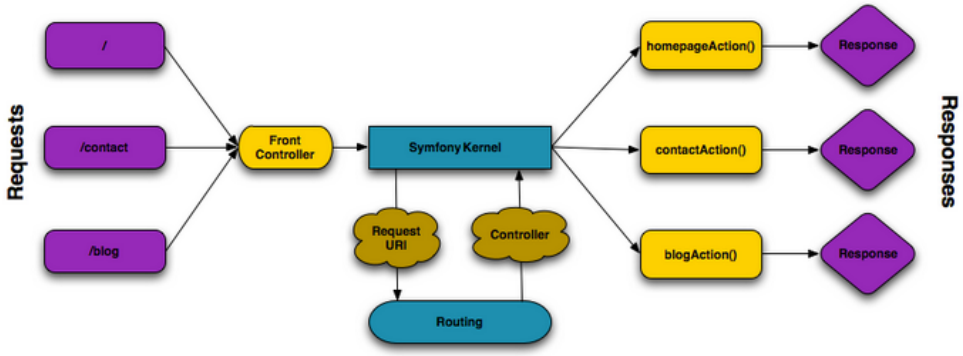
4.3.1 Systém požadavků

Pro zpracování uživatelských požadavků má Symfony předepsaný systém: Akci vyvolává uživatel odesláním HTTP žádosti (při běžném používání zpravidla GET či POST) na server. Komponenty Symfony žádost vyhodnotí a pokusí se namapovat programátorem stanovené kontrolery (viz níže) na základě tzv. „routes“, které nejsou ničím jiným, než URL cestami. Zde si programátor může nadefinovat jakékoliv chování - tento přístup tedy stíní běžný přístup webového serveru - vyhodnocení URL jako cesty.

Níže demonstruji chování na příkladu URL, které uživatel může zadat.

```
http://www.example.com/admin/user_overview
```

Na běžném Apache serveru s povoleným procházením adresářů by se server



Obrázek 4.2: Systém zpracování požadavků v Symfony [25]

pokusil vyhledat v kořenovém adresáři složku `admin` a v ní podsložku či soubor bez přípony `user_overview`. Symfony ovšem tuto cestu vezme tak, jak byla obdržena od uživatele, a následně se pokusí v definovaných kontrolerech vyhledat odpovídající cestu `/admin/user_overview`. Toto chování má dvě výhody - citlivý obsah zdrojového kódu Symfony bude chráněn před zraky uživatele a zároveň budou všechny URL adresy použite v aplikaci přehledné a čitelné.

V konfiguračním souboru aplikace `security.yml` je možné nadefinovat pevný přístup rolí k jednotlivým cestám. Toto bude využito pro zabezpečení některých oblastí aplikace.

V aplikaci bude použita následující hierarchie cest:

- `/`
  - Zastřešuje přihlašovací formulář a veřejně přístupné údaje.
  - Není zabezpečeno.
- `/admin`
  - Zastřešuje prostředí pro administrátora.
  - Zabezpečeno rolí v konfiguračním souboru.
- `/accountant`
  - Zastřešuje prostředí pro účetního.
  - Zabezpečeno rolí v konfiguračním souboru.
- `/supervisor`
  - Zastřešuje prostředí pro koordinátora.
  - Zabezpečeno logikou kontrolerů.

- `/employee`
  - Zastřešuje prostředí pro zaměstnance.
  - Zabezpečeno logikou kontrolerů.

Vzhledem k tomu, že role zaměstnance a koordinátora je vázaná na projekty, bude zde aplikována jiná strategie, než u administrátora a účetního. Kontrolery budou rozhodovat o přístupu těchto oblastí ve spolupráci s databází.

4.3.2 Správa uživatelů

Pro správu uživatelů používám nadstavbu FOSUserBundle, která se vyplatí, pokud ukládáme uživatele do databáze. Poskytuje šablonu pro uživatelskou entitu, kterou rozšiřuji ve svém modelu. Tato šablona umožňuje pohodlnou a bezpečnou manipulaci s uživatelským heslem - při jeho úpravě volám speciální komponentu (která se v aplikaci dá vyhledat jako služba pod názvem `fos_user.user_manager`), která se sama stará o hashování hesla. Kromě toho využívá informací z konfiguračního souboru a odstiňuje většinu aplikace za přihlašovací formulář.

4.3.3 Kontrolery

Symfony poskytuje PHP třídu *Controller* [2]. Tato třída umožňuje jednotlivým metodám přiřadit anotace, pomocí kterých je mapujeme přímo na URL cesty. Tyto anotace potom využívá router, metodám poskytne HTTP požadavek formou argumentu a návratovou hodnotu využije jako odpověď prohlížeči.

Kontrolery ve své aplikaci dělím dle následující hierarchie:

Kontrolery pro operace s daty

V aplikaci je řada kontrolerů, jejichž primární funkci je zpracovávání uživatelského vstupu (nejčastěji ve formě formulářů, fakultativně pomocí GET požadavků). Tyto kontrolery - narozdíl od kontrolerů pro prostředí rolí - nemusí nutně renderovat vlastní obrazovku a mohou uživatele pouze odkázat na jeho předchozí pomocí poskytnuté HTTP hlavičky.

Tyto kontrolery jsou dedikované pro zpracování uživatelského vstupu formou Symfony formulářů. Formuláře se definují pomocí formulářových tříd, které konvenčně používají příponu „Type“ - např. formulář pro modifikaci uživatele by se tedy jmenoval „UserType“. Obsahem těchto tříd jsou pole, která využíváme a která se automaticky mapují na atributy dané třídy definované v entitě.

[2] nezaměňovat s komponentou *Controller* v MVC

Největší výhodou Symfony formulářů je automatická validace na straně serveru i klienta. Tato validace je založená na:

- Doctrine anotacích v entitě,
- PHP deklaracích typu v argumentech funkcí,
- callback funkcích definovaných v entitě.

Doctrine anotace jsou základní podmínky definované na úrovni databáze. Definovat se zde dá např. nenulovost, unikátnost apod. I vlastnosti jazyka PHP jsou použít pro validaci - PHP je velice slabě typovaný jazyk, ale umožňuje specifikovat konkrétní třídy jako příchozí argumenty metod. Pro jakékoliv kontroly se složitější logikou lze využít tzv. callback validační funkce, v jejichž těle si můžeme definovat libovolné podmínky (ve své aplikaci toto využívám například pro kontrolu časových slotů, kde musí počáteční čas být nižší nebo stejný s koncovým časem).

Symfony formuláře nabídnou dvouúrovňovou kontrolu uživatelského vstupu - na straně klienta a serveru. Kontrola na straně serveru se provádí pomocí HTML 5 formulářových tagů - ta lze ale snadno obejít a nepokrývá vše. Po odeslání formuláře na server Symfony automaticky zkontroluje všechny tři výše uvedené podmínky a v případě neshod vygeneruje odpovídající výjimku.

CSRF ochrana CSRF je technika útoku hackera na aplikaci, kde škodlivá stránka vytvoří žádost pro jinou stránku použitím existující uživatelské session [26]. Příkladem může být například škodlivý link, na který uživatel klikne v době, kdy je zároveň do aplikace přihlášen. Tento link může uživatele navést k nechtěnému smazání či změně dat.

Pro potřeby formulářů Symfony implementuje CSRF ochranu automaticky pomocí generovaných tokenů v ukrytých polích formuláře - žádost se nezpracuje, pokud nebyl požadovaný token přijat. Má aplikace ovšem pro účely funkcí smazání uživatele či projektu nevyužívá Symfony formuláře (funkce je realizována jednoduchou URL cestou, kterou zpracovává příslušný kontroler). Proto jsem tuto oblast zabezpečil manuálně - každý formulář pro odeslání žádosti o smazání dostává od kontroleru přidělený token (k čemuž v symfony slouží služba „CSRF token manager“), který se následně validuje při zpracovávání žádosti.

Kontrolery pro obsluhu prostředí rolí

Kontrolery pro obsluhu prostředí jsou pojmenovány podle role, kterou obsluhují (např. „AdminController, SupervisorController...“ a jejich funkcí je zkontrolovat uživatelská práva - což se týká rolí vázaných na konkrétní projekty - a přesměrovat ho na správnou obrazovku. V rámci kontroleru se získávají data potřebná pro zobrazení obrazovky, avšak komplexní operace nad databází jsou zde v souladu s principy MVC minimální.

Kontroler pro správu směn

Kontroler pro operace nad směnami slouží především pro operace s databází, podobně jako kontrolery pro operace s formuláři. Narozdíl od nich zde ale formuláře použity nejsou. V zájmu uživatelského pohodlí jsem se totiž rozhodl operace nad směnami realizovat jako žádosti AJAX. Pro žádání o směnu, potvrzení směny apod. tedy není nutné obnovovat celou stránku, a uživatel tak může na jedné stránce provést několik operací rychle za sebou.

Mechanismus uvedu na příkladu: Uživatel si chce zapsat směnu. Pomocí uživatelského prostředí se dostane na příslušnou stránku, vybere si časový slot a vedle něj vidí tlačítko „Zažádat o směnu“. Tlačítko je realizováno jako klasický odkaz, který ovšem nevede nikam (toho dosáhneme pomocí pseudoprotokolu `javascript:`, které zavolá funkci `void(0)`). Pomocí jQuery se ovšem o této události dozví handler implementovaný v externím skriptu - tlačítko má přiřazenou třídu `dynamic_requester`, na kterou čeká jQuery selektor. Skript následně:

- provede parsing atributů tlačítka, které ho informují o tom, jakou žádost má zkonstruovat,
- odešle žádost na server pomocí jQuery a její funkce `$.ajax();`,
- upraví atributy tlačítka na další logickou operaci se směnou (v uvedeném příkladu poskytne údaje pro transformaci tlačítka na zrušení žádosti),
- upraví vzhled tlačítka, aby dalo uživateli najevo, že žádost se zpracovává,
- callback funkce počká na odpověď serveru,
  - v případě úspěchu transformuje tlačítko na tlačítko další logické operace se směnou,
  - v případě selhání (např. slot byl v mezičase zrušen, kapacita byla naplněna...) tlačítko zablokuje a zobrazí chybovou hlášku.

Metoda, kterou AJAX žádost odešle, je typu GET. V rámci URL cesty je i zde implementována CSRF ochrana. Při zpracovávání žádosti jsem vzal v potaz, že nevyužitím konceptu formulářů jsem se připravil o automatickou validaci a implementoval svou vlastní. Server tedy zpracuje přijatou žádost následujícím způsobem:

- ověří CSRF token,
- zvaliduje ID směny a v případě úspěchu ji nahraje z databáze (metoda `retrieveAndValidateShift()`),
- ověří, zda má uživatel právo nad směnou provést žádanou operaci (metoda `checkPermissions()`),

- ověří, zda je požadovaná změna stavu validní (z životního cyklu směn je znát, že některé přechody jsou nepřípustné - např. ze stavu „zrušená“ na stav „odpracovaná“) a případně přechod uskuteční a notifikuje zaměstnance o změně, pokud operaci provedl koordinátor (metoda `switchShift()`),
- v případě nové směny zkontroluje kapacitu (metoda `checkCapacity()`),
- v případě nové směny a vypnutého nastavení projektu `allowsOverlappingShifts` zkontroluje překrývání směn (metoda `checkOverlapping()`),
- potvrdí transakci a uloží změny do databáze,
- jako odpověď prohlížeči odešle odpověď s novým CSRF tokenem pro odeslání dalších požadavků - prohlížeč není nikam přesměrován.

4.3.4 Služby

Služby jsou v Symfony nezávislé komponenty, které jsou určeny k tomu, aby byly volány z libovolné části aplikace. V této sekci okomentuji služby, které jsem zavedl v rámci své aplikace.

Služba pro odesílání zpráv

V budoucích rozšířeních aplikace se předpokládá komplexnější využívání zpráv, ovšem pro potřeby prototypu budou zprávy sloužit pouze k notifikaci zaměstnanců o změnách nad jejich směnami (potvrzení, zrušení...). Jakákoliv změna směny ze strany koordinátora je v kódu vázána na odeslání notifikace zaměstnanci, jemuž je směna přiřazena (blíže popsáno v sekci kontrolerů - metoda `switchShift()`).

Aplikace definuje metodu `shiftChangeNotify()` speciálně pro odesílání zpráv o změnách směn. V budoucnu se předpokládá vývoj dalších, univerzálnějších metod, např. pro přímou komunikaci mezi uživateli.

Služba pro stránkování dat

Při rozsáhlejším používání aplikace a zvětšování množství uložených dat naskytne problém se zobrazováním informací (např. přehled uživatelů), kdy se na jedné stránce objeví příliš dlouhé seznamy nepohodlné na navigaci. K řešení tohoto problému slouží třída `PaginationWidget`, která slouží k „porcování“ dat na stránky. Díky použití služeb bude tato komponenta přístupná ve všech kontrolerech. Je nastavená pro ukládání libovolných dat ve formě `ArrayCollection` - třída, kterou Symfony využívá z API Doctrine, která slouží k objektovému zapouzdření běžného PHP pole s přidáním některých užitečných funkcí, jako například filtrování a iterování. Tyto data je pak schopna rozdělit na stránky dle uživatelsky definovaných parametrů a poskytovat je dle čísla stránky, kterou uživatel zadá.

4.4 Uživatelské prostředí

Pro uživatelské prostředí jsem použil několik vzájemně zanořených Twig šablon. Pro co největší přehlednost kódu jsou znovupoužitelné části odděleny. V této sekci popíši strukturu těchto šablon.

4.4.1 Základní principy

Po přihlášení do aplikace uživatel uvidí hlavní stránku, ze které se může nechat přesměrovat do jednotlivých oblastí. Možnosti nabízené jednotlivými oblastmi závisí na projektu který má vybraný jako preferovaný. Preferovaný projekt se uživateli vždy zobrazuje v hlavním menu a uživatel ho může kdykoliv změnit. Zbytek menu poskytne dle uživatelských oprávnění přístup do administrátorské, účetnické, koordinátorské a zaměstnanecké sekce.

V administrátorské sekci uživatel dostane možnost zobrazení a správy všech uživatelů a projektů. Libovolnou entitu může upravit pomocí formuláře, nebo smazat (ukázka na obrázku B.4).

V účetnické sekci uživatel vidí všechny ostatní uživatele a má přístup k jejim hodinovým mzdám. S ohledem na citlivost dat je zde implementována speciální funkce - po načtení stránky jsou pole překryta cenzurovacím okénkem, které při kliknutí zmizí a umožní editaci daného pole. Myšlenka za tímto opatřením minimalizuje množství citlivých údajů na obrazovce a zmenšuje šanci, že je uvidí třetí osoba např. „přes rameno“.

V sekci pro koordinátory uvidí uživatel všechny časové sloty založené pod pracovními činnostmi, za které je zodpovědný, a všechny směny zapsané v těchto činnostech. Obě tyto entity si může zobrazit v denním zobrazení a v kalendáři (po měsících). Krom toho zde má možnost přístupu do sekce žádostí k vyřízení, kde uvidí všechny směny v jeho spravovaných činnostech, které jsou v nevyřízeném stavu - čekají na potvrzení nebo žádají o zrušení (ukázka na obrázku B.3).

Sekce pro zaměstnance umožní uživateli zobrazit notifikace o posledních změnách nad jeho směnami spolu s informací kdy k nim došlo a kdo je provedl (ukázka na obrázku B.2). Také zde dostanou možnost zobrazit si své finanční ohodnocení a přepočítaný výdělek za poslední měsíc. V plánovací části vidí časové sloty, ke kterým jsou oprávněni se přihlásit, a své vlastní směny. K dispozici jsou tři zobrazení - agenda (seznam relevantních slotů/směn v budoucnu), kalendář (zobrazení po měsících) a denní (zobrazení slotů/směn po konkrétních dnech).

4.4.2 Kořenové šablony

V aplikaci používám dvě kořenové šablony - `base.html.twig` a v ní vložená `frame.html.twig`. Z těchto dvou navzájem vnořených šablon se konstruují všechny ostatní obrazovky. První šablona má na starosti poskytovat kostru

HTML dokumentu - hlavičku s potřebnými meta tagy, vložení potřebných CSS šablon (na což Twig umožňuje vynucovací funkci `asset`), jQuery knihovny a Javascriptových externích skriptů relevantních pro konkrétní obrazovku (což je realizováno pomocí Twig bloku `additional_headers`, kde si pro každou konkrétní stránku mohu zvolit, které skripty chci mít k dispozici). Druhá šablona slouží pro renderování GUI, které uživatel najde na všech obrazovkách po přihlášení - hlavní menu, výběr projektu, logo aplikace a uživatelský panel pro odhlášení.

4.4.3 Widgety

Malé sekce použité na více místech prostředí označuji jako widgety. Jedná se o obyčejné Twig šablony, které se dokáží přizpůsobit dle vstupních parametrů.

Kalendář

Pro měsíční zobrazení směn a časových slotů jsem implementoval šablonu kalendáře (ukázka na obrázku B.1). Pro její vykreslení je nutné předat vstupní argumenty `year` a `month`. V kalendáři je pak každý den reprezentován buňkou, která slouží jako odkaz pro přechod k dennímu zobrazení. Pro plné přizpůsobení kalendáře každá buňka vykresluje blok `inside_cell`, což umožní, aby se v každé z nich zobrazovaly zkrácené informace ohledně daného dne (např. počet směn).

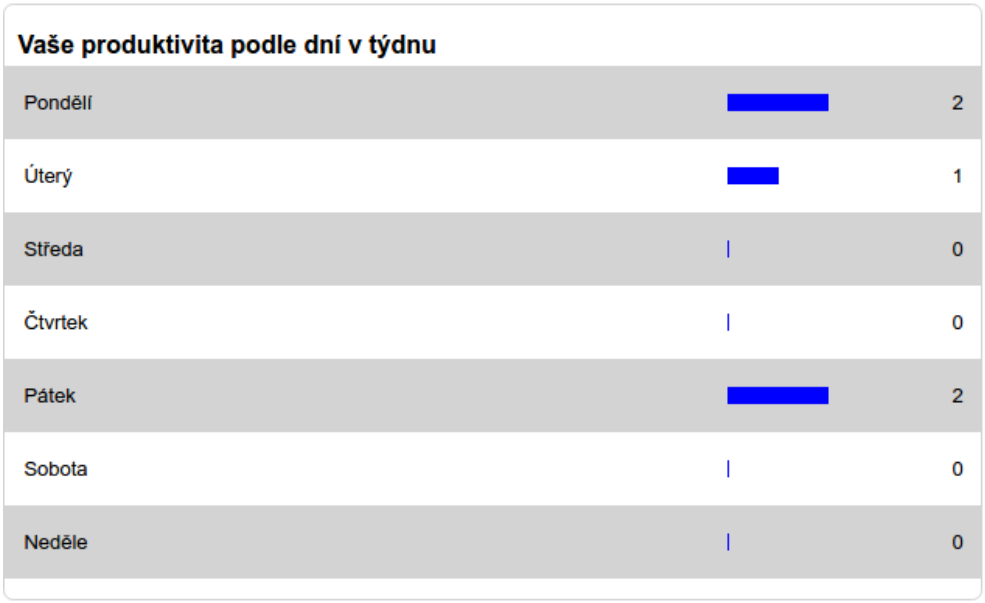
Widgety produktivity

Widgety produktivity jsou widgety, které slouží k zobrazení rychlého přehledu informací. Nejedná se o stěžejní funkci, ale poskytne uživateli větší pohodlí, přehlednost a zajímavé informace. V prototypu implementuji tři widgety produktivity, dva pro zaměstnance a jeden pro koordinátora. Widgety implementované v rámci prototypu slouží především k nastínění možností dalších rozšíření aplikace.

Mnoho funkcí pro získávání informací pro widgety produktivity využívají nativní SQL namísto preferovaného DQL. To proto, že DQL není schopné dokonale emulovat všechny SQL funkce (např. funkce `WEEKDAY()`).

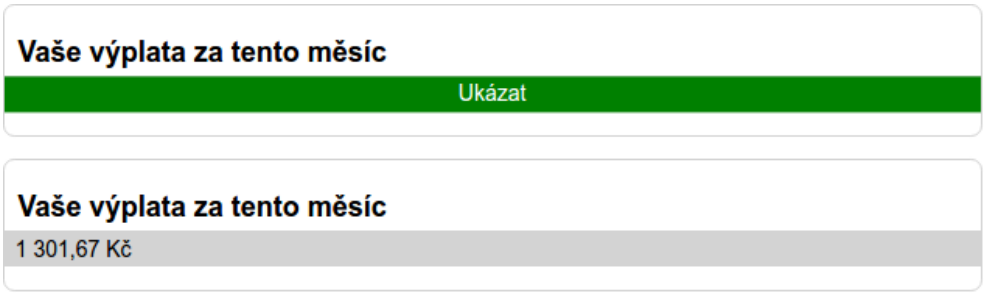
Týdenní přehled Týdenní přehled se nachází na domovské obrazovce zaměstnanců a slouží pro zobrazení tyčového grafu, který ukazuje poměr odpracovaných směn podle dní v týdnu. Jedná se čistě o informaci pro zajímavost pracovníka.

Finanční přehled Finanční přehled se nachází na domovské obrazovce zaměstnanců a slouží zaměstnanci pro zobrazení dosavadní výplaty za aktuální



Obrázek 4.3: Týdenní přehled pro zaměstnance

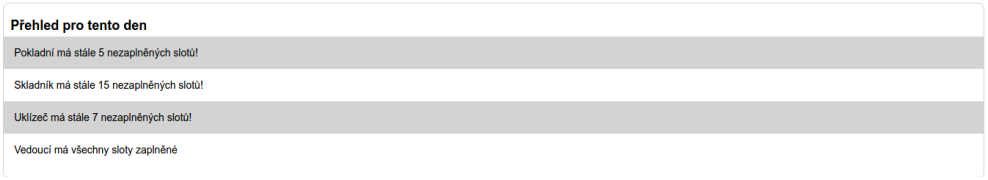
měsíc. Výplata se počítá ze mzdy, kterou má uživatel přidělen (pokud je vyplněna) a to pomocí směn ve stavu „odpracovaná“. Widget implementuje překrývací okénko, podobně jako sekce pro účetního (výdělek zaměstnance je citlivá informace a toto opatření ji ochrání před neoprávněnými přihlížejícími, kteří by se uživateli mohli koukat přes rameno ve chvíli, kdy se nachází na domovské stránce).



Obrázek 4.4: Finanční přehled pro zaměstnance - zakrytý a odkrytý

Přehled obsazenosti směn Pro koordinátory je na obrazovkách denního přehledu směn a časových slotů připraven widget ukazující obsazenost jednotlivých pracovních činností na daný den, což mu pomůže rychle zjistit, kolik lidí

na daný den ještě musí sehnat. Widget zobrazuje všechny pracovní činnosti, za které je koordinátor odpovědný.



Obrázek 4.5: Denní přehled pro koordinátory

4.4.4 Překlady

Aby byla aplikace rozšířitelnější o další jazyky, psal jsem většinu textací v aplikaci anglicky a textace jsem obalil pomocí speciálního Twig filtru pro automatický překlad. Samotná funkcionalita pro přepínání jazyků implementována nebyla, ale pokud se v budoucnu aplikace bude o tuto funkci rozšiřovat, nebude potřeba přepisovat veškeré texty. Jelikož se některé texty nahrávají dynamicky pomocí Javascriptu, bylo nutné implementovat speciální cestu `translate` výchozího kontroleru `DefaultController`, která na požádání volá stejnou Symfony komponentu pro překlad, kterou využívá Twig. Aby však server nebyl zbytečně zatěžován, ukládají se již přeložené výrazy do lokálního uložiště prohlížeče a kontroler se volá jen, pokud prohlížeč narazí na dosud neznámou textaci.

```
function translate(element, val) {
    var data = {
        original: val,
        translated: null
    };

    if (localStorage.getItem(data.original) == null) {
        $.post("/translate/", data, function (response) {

            // pro nizsi zatez serveru si prohlizec pamatuje jiz prelozene vyrazy
            localStorage.setItem(data.original, response.translated);
            element.html(response.translated);
        });
    }
    else {
        element.html(localStorage.getItem(data.original));
    }
}
```

Obrázek 4.6: Skript pro dynamické překlady

Kapitola 5

Testování

5.1 Testy podle případů užití

Dle případů užití jsem vypracoval řadu manuálních testů pro ověření komplexní funkčnosti aplikace, které jsem následně provedl a úspěšně jimi prošel. Mimo pozitivních scénářů jsem ověřil také správné chování negativních scénářů - takových situací, kdy scénář není možné úspěšně dokončit. Pro důkladné otestování jsem prošel i všechny alternativní cesty a výjimky a ověřil jejich správné chování.

5.2 Unit testy

V OOP mluvíme o unit testech jako o automatických testech, které slouží pro testování jednotlivých tříd a metod [27]. V rámci své aplikace jsem se rozhodl aplikovat tuto metodu testování pro třídy reprezentující jednotlivé entity v aplikaci. Tyto metody mají celou řadu metod, které jsou vygenerované pomocí Doctrine - jedná se o výchozí konstruktory, gettery a settery. Tyto metody nebudou předmětem testování - mimo jiné i proto, že jsou velice primitivní. Pozornost budu věnovat pomocným metodám, které jsem si v rámci entit musel vytvořit sám.

Pro Unit testy budu používat PHPUnit, testovací framework pro PHP založený na architektuře xUnit [28]. PHPUnit automaticky zpracuje všechny třídy ve složce `/tests`, jejichž název končí na `Test.php`. Samotná aplikace PHPUnit nebude součástí samotné práce, ale testy ano. Pro samotné testování je nutné použít verzi PHPUnit 5.4 a vygenerovat konfigurační XML soubor.

```
wget https://phar.phpunit.de/phpunit-5.4.phar
chmod +x phpunit-5.4.phar
./phpunit-5.4.phar --generate-configuration
./phpunit-5.4.phar
```

```
/**
* Add shift
*
* @param \AppBundle\Entity\Shift $shift
*
* @return Timeslot
*/
public function addShift(\AppBundle\Entity\Shift $shift)
{
    $this->shifts[] = $shift;

    return $this;
}
```

Obrázek 5.1: Ukázka Doctrine vygenerovaného setteru nad entitou `Timeslot`

V rámci provádění Unit testů nad objekty v operační paměti je důležité poznamenat chování Doctrine entit. Na první pohled je z primitivních getterů a setterů vidět, že žádné relace, které vytvoříme nad objekty, nebudou oboustranné. Příklad je uveden v kódu 5.2, kde při přiřazování nové směny časovému slotu setter pouze přidá daný slot do kolekce[3]. Druhou stranu neřeší, a getter `getTimeslot()` na straně dané směny tak bude vracet nesprávné informace.

Důvod tohoto chování je prostý: Doctrine vygenerované gettery a settery oprávněně očekávají, že entity budou buďto přebrány z databáze (pomocí repozitářů), nebo později uloženy do databáze (metoda `persist()` komponenty `EntityManager`). Při jakýchkoliv těchto akcích se chování synchronizuje a informace se předají oběma stranám - gettery pak již budou daleko užitečnější. V rámci Unit testů ale s databází pracovat nebudu, což implikuje nutnost myslet na obě strany, pokud budu používat relace.

5.2.1 UserTest

Test pomocných getterů na projekty

Pro jednoduchost a přehlednost jsem definoval pomocné gettery na projekty uživatelů. Jelikož se uživatelé nepárují přímo se samotnými projekty, ale s pracovními činnostmi, které jim náleží, je nutné tento vztah definovat zvlášť. V testu zkouším správné chování následujících metod:

- `getAttendingProjects()` - projekty, kde je uživatel zaměstnán,

[3]Nejedná se o pole, jak by se mohlo na první pohled ze syntaxe zdát, ale o kolekci typu `ArrayCollection`, která implementuje přetěžování operátoru `[]`.

- `getSupervisingProjects()` - projekty, kde je uživatel koordinátorem,
- `getAvailableProjects()` - všechny projekty, kde plní uživatel nějakou roli (implementováno speciálně pro přepínání projektů v GUI).

```
// inicializace

$user = new User();

$jobA1 = new Job();
$jobS1 = new Job();
$jobX1 = new Job();

$projectA1 = new Project(); // uzivatel pracuje v tomto projektu jako zamestnanec
$projectS1 = new Project(); // uzivatel pracuje v~tomto projektu jako koordinator
$projectX1 = new Project(); // uzivatel nema s timto projektem nic spolecneho

$jobA1 -> setProject($projectA1);
$jobS1 -> setProject($projectS1);

$user -> addAttending($jobA1);
$user -> addSupervising($jobS1);

// pozitivni scenare

$this -> assertTrue($user -> getAttendingProjects() -> contains($projectA1));
$this -> assertTrue($user -> getSupervisingProjects() -> contains($projectS1));
$this -> assertEquals(new ArrayCollection([$projectA1, $projectS1]),
$user -> getAvailableProjects());

// negativni scenare

$this -> assertFalse($user -> getAvailableProjects() -> contains($projectX1));
$this -> assertFalse($user -> getAttendingProjects() -> contains($projectS1));
$this -> assertFalse($user -> getSupervisingProjects() -> contains($projectA1));
```

Test getteru směn podle časového slotu

V interní logice jsem se setkával s potřebou získávat informace o směnách uživatele v závislosti na jejich časovém slotu, proto jsem si vytvořil speciální getter. Ten testuje tento test.

```
$user = new User();

$timeslot1 = new Timeslot();
```

```
$timeslot2 = new Timeslot();

$shift1 = new Shift($user, $timeslot1);
$shift2 = new Shift($user, $timeslot1);
$shift3 = new Shift($user, $timeslot2);

$user -> addShift($shift1);
$user -> addShift($shift2);

$this -> assertEquals(new ArrayCollection([$shift1, $shift2]),
$user -> getShiftsByTimeslot($timeslot1));
```

5.2.2 TimeslotTest

Test překrývání časových slotů

Pokud si to uživatel zvolí v nastavení projektu, projekt nedovolí, aby uživatelé měli vzájemně se překrývající směny (mimo stavy „zrušená“ a „zamítnutá“, kde by tato kontrola nedávala smysl). Proto má každý časový slot metodu, kde se může porovnat s jiným časovým slotem a zjistit, zda se překrývají. Tento test kontroluje její správné chování.

```
$data = [
    // [zacatek prvniho, konec prvniho, zacatek druheho, konec druheho,
    prekryvaji se]
    ['00:00', '08:00', '12:00', '18:00', false], // sloty jsou zcela odlisne
    ['00:00', '08:00', '07:00', '18:00', true],  // sloty se prekryvaji o
    jednu hodinu
    ['00:00', '08:00', '00:00', '08:00', true],  // sloty jsou stejne
    ['00:00', '08:00', '08:00', '18:00', false], // test meznich hodnot 1
    ['00:00', '08:00', '07:59', '23:59', true],  // test meznich hodnot 2
];

foreach ($data as $item) {
    $timeslot1 = new Timeslot();
    $timeslot2 = new Timeslot();

    $timeslot1 -> setBeginTime($item[0]);
    $timeslot1 -> setEndTime($item[1]);

    $timeslot2 -> setBeginTime($item[2]);
    $timeslot2 -> setEndTime($item[3]);

    $this -> assertEquals($item[4], $timeslot1 -> overlapsWith($timeslot2));
    $this -> assertEquals($item[4], $timeslot2 -> overlapsWith($timeslot1));
```

```
}
```

Test naplnění kapacit

Každý časový slot si udržuje informace o své maximální kapacitě. Pokud je nastavena na nulu, nebude kontrolována. V opačném případě nedovolí zapsání nových směn, pokud počet směn mimo stavy „zamítnutá“ a „zrušená“ překračuje danou hodnotu. Chování této metody ověřuje tento test.

```
$user = new User();
$timeslot = new Timeslot();

// test omezene kapacity

$timeslot -> setCapacity(5);

for ($i = 0; $i < 4; $i++) {
    $shift = new Shift($user, $timeslot);
    $timeslot -> addShift($shift); // nutné kvuli Doctrine getterum a setterum
    $shift -> setState(Shift::CONFIRMED);
}

$this -> assertFalse($timeslot -> isFull());

// smeny ve stavec "zrusena" nebo "odmitnuta" nemaji na kapacitu vliv

$shift1 = new Shift($user, $timeslot);
$shift1 -> setState(Shift::CANCELLED);
$timeslot -> addShift($shift1);

$shift2 = new Shift($user, $timeslot);
$shift2 -> setState(Shift::REJECTED);
$timeslot -> addShift($shift2);

$this -> assertFalse($timeslot -> isFull());

// prekroceni kapacity

$shift1 = new Shift($user, $timeslot);
$shift1 -> setState(Shift::CONFIRMED);
$timeslot -> addShift($shift1);

$this -> assertTrue($timeslot -> isFull());
```

```
// test neomezene kapacity

$timeslot -> setCapacity(0);

$this -> assertFalse($timeslot -> isFull());
```

5.2.3 ShiftTest

Testování automatického potvrzení koordinátorových směn

Zakládání nové směny má definované speciální chování v kontroleru `ShiftController` - protože instance daného objektu na začátku provádění kontrol neexistuje, má smysl provádět je v samotném konstruktoru. Proto testuji automatické potvrzování koordinátorových směn, ale automatické potvrzování žádostí o zrušení již ne - tam se již nejedná o vlastnost entity ale o povinnost kontroleru.

```
// inicializace

$user = new User();

$timeslot = new Timeslot();

$job = new Job();

// test automatickeho potvrzeni koordinatorske smeny

$timeslot -> setJob($job);

$user -> addSupervising($job);

$shift = new Shift($user, $timeslot);

$this -> assertEquals(Shift::CONFIRMED, $shift -> getState());

$user -> removeSupervising($job);

// test spravneho chovani po odebrani koordinatorskych prav

$shift2 = new Shift($user, $timeslot);

$this -> assertEquals(Shift::UNCONFIRMED, $shift2 -> getState());
```

Testování chování metody ověřování finálního stavu

Finální stav je interní pojem, který z hlediska celkového pohledu implementace není stěžejní, ale pomůže ušetřit práci se směnami. Směna ve finálním stavu je směna ve stavu „zamítnutá“ a „zrušená“. Toto pomáhá aplikaci se rozhodnout, kdy např. zobrazit tlačítko pro zažádání o novou směnu. Tento jednoduchý test ověřuje chování metody k tomu určené.

```
$user = new User();
$timeslot = new Timeslot();

$shifts = new ArrayCollection();
$shifts[] = (new Shift($user, $timeslot)) -> setState(Shift::UNCONFIRMED);
$shifts[] = (new Shift($user, $timeslot)) -> setState(Shift::CONFIRMED);
$shifts[] = (new Shift($user, $timeslot)) -> setState(Shift::REJECTED);
$shifts[] = (new Shift($user, $timeslot)) -> setState(Shift::CANCELLED);
$shifts[] = (new Shift($user, $timeslot))
-> setState(Shift::REQUIRES_CANCELLATION);
$shifts[] = (new Shift($user, $timeslot)) -> setState(Shift::MISSED);
$shifts[] = (new Shift($user, $timeslot)) -> setState(Shift::FINISHED);

$this -> assertEquals(2, $shifts -> filter(function($item) {
    return $item -> isInFinalState() == true;
}) -> count());
```

Kapitola 6

Pokyny k nasazení

Tento manuál popisuje nasazení celé aplikace. Ačkoliv je napsán pro konkrétní platformu, zkušený uživatel může některé komponenty nahradit (jiný operační systém, jiná databáze) - pokyny jsou obecné, pouze příkazy jsou specifické pro dané prostředí (tabulka v 6.1). Požadavkem je však minimálně podpora symbolických linků v daném OS.

Operační systém	Ubuntu 16.04 (64 bit)
Databáze	MySQL
Příkazový řádek	bash

Tabulka 6.1: Prostředí pro pokyny k nasazení

6.1 Předpoklady

- Systém je správně nakonfigurován a zabezpečen jako server pro webovou aplikaci.
- Databázový systém je nainstalován a připraven - buďto na stejném systému jako server, nebo přístupná ze sítě. Jsou připravené přístupové údaje, adresa serveru a port. Je připravena databáze speciálně pro naši aplikaci.
- V systému máme administrátorská práva.

6.2 Postup

- V aplikaci je potřebné vyplnit údaje pro přístup k databázi - v konfiguračním souboru `app/config/config.yml` a mnou přidaným pomocným instalátorem `set_up.sh`. Je potřeba vyplnit jméno databáze, přihlašovací jméno, heslo, adresu serveru a port.

- Nainstalujeme PHP a spolupracující komponenty pro propojení PHP a databáze.

```
sudo apt-get install php php7.0-xml php7.0-mysql
```

- Nainstalujeme aplikaci `composer`, která nám umožní instalovat všechny potřebné závislosti.

```
curl -sS https://getcomposer.org/installer | sudo php -- --install-dir=/usr/lo
```

- Informujeme systém, že chceme používat *produkční* verzi prostředí (nepovinný krok, pokud chceme pokračovat s vývojem).

```
export SYMFONY_ENV=prod
```

- Nainstalujeme potřebné závislosti (parametr `-no-dev` je opět relevantní pouze pro neprodukční verze).

```
php /usr/local/bin/install --no-dev --optimize-autoloader
```

  - Pokud by předchozí příkaz vracel chybu, vyčistíme vyrovnávací paměť.

```
php bin/console cache:clear --env=prod
```

- Spustíme mnou přidaný instalátor `set_up.sh`. Tento skript:
  - importuje Doctrine schéma do databáze, vytvoří všechny tabulky a propojí je s entitami,
  - vytvoří výchozího uživatele `admin` s heslem `password`, **kterého je nutné před síťovým zpřístupněným vymazat nebo mu změnit heslo**,
  - importuje SQL skript na přepínání směn.

V tuto chvíli je aplikace připravena. Pro vývojové prostředí je možné jí spustit příkazem `php bin/console server:run <IP>:<PORT>`. Toto spustí vestavěný PHP server, který aplikaci spustí. Není ovšem doporučený pro produkční prostředí a měl by se používat spíše pro vývoj [29]. Aplikaci tedy můžeme nahrát na náš již nakonfigurovaný preferovaný webserver (např. Apache), kde se jako kořenový adresář používá složka `/web`.

Kapitola 7

Zhodnocení a výhled do budoucna

Celkový postup vývoje se díky předpřípravě základu frameworku obešel bez větších potíží. Během implementace bylo provedeno několik zásahů do analýzy, které se během návrhu jevily jako bezproblémové - konkrétně problém relace mezi uživatelem, projektem a pracovní činností. Také stavový diagram směn byl oproti prvotní myšlence doplněn o další stavy pro poskytnutí detailnější informací pro uživatele. Spolupráce zavedených technologií byla na velmi kvalitní úrovni. Drobné nedostatky v dokumentaci plně nahradila diskusní vlákna na internetu a tutoriály na oficiálních stránkách Symfony.

Požadavky na základní funkce byly naplněny a aplikace nabízí i díky svému otevřenému zdrojovému kódu velký prostor pro další rozšíření. Těm se budu věnovat v této kapitole.

7.1 Rozšíření systému notifikací

V prototypu byla nastíněna důležitost notifikací jakožto hlavního prvku kontroly, kterou zaměstnanec dostává nad svými směnami. Jejich potenciál je ale daleko vyšší - vhodnou úpravou by bylo možné získat efektivní systém obousměrné komunikace mezi koordinátorem a zaměstnancem (popř. i dalšími rolemi). Za zvážení také stojí koncept systému chatu. Aby nedošlo ke zmeškání žádných důležitých informací, notifikace by také šly volitelně odesílat e-mailem, popřípadě i na telefonní čísla formou SMS.

7.2 Widgety produktivity

Tato funkce demonstrovaná několika jednoduchými příklady nabízí potenciálním vývojářům mnoho inspirace. Vhodně napsané widgety mohou ušetřit uživatelům mnoho času při interpretaci jednotlivých dat, ve kterých je pro

každou roli důležitá jiná informace. Ačkoliv se nejedná o stěžejní funkci informačního systému, drobná personalizace dokáže zdokonalit uživatelský prožitek bez jakýchkoliv extra nároků na samotného uživatele [30].

7.3 API

API jakožto množina rutin, protokolu a nástrojů pro tvorbu softwarových aplikací specifikuje způsob, jakým spolu různé softwarové komponenty mohou spolupracovat [31]. U informačního systému jako je tento se nabízí například:

- propojení s účetním programem,
- propojení s existující sociální sítí,
- propojení se systémy osobního hodnocení,
- propojení s interaktivními kalendáři.

7.4 Náhrada dalších podnikových systémů

Předchozí sekce naznačuje potenciální propojení s existující sociální sítí na pracovišti a systémem osobního hodnocení - pokud ovšem tyto alternativy na pracovišti nejsou, nic nebrání tomu, aby byl využit softwarový podklad této aplikace pro tvorbu těchto komponent na míru dané společnosti.

7.5 Vykazovací systém

Aplikace momentálně nabízí jednoduchý systém vykazování - automatické, které předpokládá perfektní docházku a lze ho jednoduše měnit dodatečně. Případné rozšíření by mohlo integrovat systém vykazování a chození na směny. Zde záleží na implementaci dané aplikace - pokud by byla přístupná i mimo síť pracoviště - systém je momentálně připraven na spravování směn z pohodlí domova - bylo by nutné implementovat bezpečnostní prvek pro ověřování fyzické přítomnosti zaměstnance na pracovišti.

Závěr

V práci se mi podařilo sestavit profil požadavků zaměstnavatele na informační systém pro plánování směn brigádníků. Tento profil byl následně porovnán s existujícími řešeními na trhu v této oblasti a byly identifikovány jejich silné stránky i nedostatky. Na těchto nedostatcích byla následně postavena analýza a návrh vlastního řešení informačního systému.

V rámci analýzy jsem postupoval dle standardních metodik vývoje informačních systémů - definoval jsem potřebné role, procesy a uvedl jsem příklad některých případů užití. V souladu s touto analýzou jsem provedl návrh architektury a výběr technologií pro vlastní implementaci prototypu informačního systému.

V implementační části jsem popsal postupy, které jsem použil při programování prototypu aplikace strukturovanou v souladu s vybranou architekturou. Na implementaci následně navazovalo testování prototypu a dokumentace týkající se nasazení produkční verze daného systému. Prototyp byl uveden do funkčního stavu (posouzeno z testování uvedeného v této práci a volného testování během implementace).

Závěrem jsem zhodnotil celý proces a průběh tvorby výstupů z této práce, v rámci kterého jsem identifikoval možný prostor pro zlepšení a rozšíření daného informačního systému. Všechny cíle definované v této práci byly úspěšně splněny.

Literatura

[1] Armstrong, M.: *Řízení lidských zdrojů*. Praha: Grada Publishing a.s., 10 vydání, ISBN 978-80-247-1407-3.

[2] Zpracování informací o mzdě a platu. *Mzdová praxe*, 2012, [cit. 2017-04-20]. Dostupné z: `http://www.mzdovapraxe.cz/archiv/dokument/doc-d38646v48599-zpracovani-informaci-o-mzde-a-platu/`

[3] ShiftPlanning becomes humanity. *Humanity.com [online]*, březen 2017, [cit. 2017-03-07]. Dostupné z: `https://www.humanity.com/blog/shiftplanning-now-humanity.html`

[4] Tomáš Brucker, A. B. I. S. D. C. V. , Jiří Voříšek: *Tvorba informačních systémů*. Praha: Grada Publishing a.s., první vydání, ISBN 978-80-247-7902-7.

[5] Gorton, I.: *Essential Software Architecture*. Richland, USA: Springer, druhé vydání, ISBN 978-3-642-19176-3.

[6] Lotz, M.: Waterfall vs. Agile: Which is the Right Development Methodology for Your Project? *Segue Technologies Inc.*, 2013, [cit. 2017-05-09]. Dostupné z: `http://www.seguetech.com/waterfall-vs-agile-methodology/`

[7] McConnell, S.: *Code complete*. Washington, USA: Microsoft Press, druhé vydání, ISBN 0-7356-1967-0.

[8] Pilou, J.: Employee Relationship Management (ERM). *CCM*, 2017, [cit. 2017-05-09]. Dostupné z: `http://ccm.net/contents/211-employee-relationship-management-erm`

[9] Rouse, M.: What is entity? *TechTarget*, 2005, [cit. 2017-05-09]. Dostupné z: `http://whatis.techtarget.com/definition/entity`

[10] Cockburn, A.: *Writing Effective Use Cases*. Addison-Wesley, třetí vydání, ISBN 9780558312602.

[11] Microsoft: *UML Use Case Diagrams: Guidelines*. 2015, [cit. 2017-04-27]. Dostupné z: https://msdn.microsoft.com/en-us/library/dd409432.aspx

[12] Bernard, B.: Úvod do architektury MVC. [cit. 2017-05-03]. Dostupné z: https://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[13] PHP: MySQL Database. *W3Schools.com [online]*, 2015, [cit. 2017-03-23]. Dostupné z: https://www.w3schools.com/php/php_mysql_intro.asp

[14] Welcome to the Doctrine project. *Doctrine-project.org [online]*, 2016, [cit. 2017-03-23]. Dostupné z: http://www.doctrine-project.org/

[15] GitHub: *Twig, the flexible, fast, and secure template language for PHP*. 2013, [cit. 2017-03-23]. Dostupné z: https://github.com/twigphp/Twig/blob/2.x/README.rst

[16] W3Schools: *JavaScript tutorial*. 2015, [cit. 2017-03-23]. Dostupné z: https://www.w3schools.com/js/default.asp

[17] W3Schools: *jQuery Introduction*. 2015, [cit. 2017-03-23]. Dostupné z: https://www.w3schools.com/jquery/jquery_intro.asp

[18] W3Schools: *AJAX Introduction*. 2015, [cit. 2017-03-23]. Dostupné z: https://www.w3schools.com/js/js_ajax_intro.asp

[19] Flanagan, D.: *JavaScript: The Definitive Guide: Activate Your Web Pages (Definitive Guides)*. O'Reilly Media, 6 vydání, ISBN 978-0-596-80552-4.

[20] Waybury: *About Iconic*. [cit. 2017-05-14]. Dostupné z: https://useiconic.com/about/

[21] W3Schools: *PHP 5 Tutorial*. 2015, [cit. 2017-03-23]. Dostupné z: https://www.w3schools.com/php/default.asp

[22] SensioLabs: *What is Symfony*. [cit. 2017-03-23]. Dostupné z: https://symfony.com/what-is-symfony

[23] Doctrine-project: *Doctrine Query Language - Doctrine 2 ORM 2 documentation*. [cit. 2017-04-13]. Dostupné z: http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/dql-doctrine-query-language.html

[24] The SQL Server Query Optimizer. *Simple talk [online]*, 2011, [cit. 2017-04-13]. Dostupné z: https://www.simple-talk.com/sql/sql-training/the-sql-server-query-optimizer/

[25] SensioLabs: The Symfony Application Flow. [cit. 2017-04-20]. Dostupné z: http://symfony.com/doc/current/introduction/http_fundamentals.html

[26] Ristic, I.: *Apache Security*. O'Reilly Media, první vydání, ISBN 0-596-00724-8.

[27] Fáze a úrovně provádění testů. *Testování softwaru*, 2011, [cit. 2017-04-27]. Dostupné z: http://testovanisoftwaru.cz/tag/unit-testing/

[28] Sebastian Bergmann and contributors: *PHPUnit*. 2017, [cit. 2017-04-27]. Dostupné z: https://phpunit.de/index.html

[29] SensioLabs: *How to Use PHP's built-in Web Server*. [cit. 2017-05-13]. Dostupné z: http://symfony.com/doc/current/setup/built_in_web_server.html

[30] Customization vs. Personalization in the User Experience. *Nielsen Norman Group*, 2016, [cit. 2017-05-09]. Dostupné z: https://www.nngroup.com/articles/customization-personalization/

[31] API - application program interface. *Vangie Beal*, [cit. 2017-05-09]. Dostupné z: http://www.webopedia.com/TERM/A/API.html

Příloha **A**

Seznam použitých zkratek

DRY Don't repeat yourself

GUI Graphical User Interface

MVC Model, View, Controller

DBMS Database Management System

SQL Structured Query Language

ORM Objektově-relační mapování

CSRF Cross-site request forgery

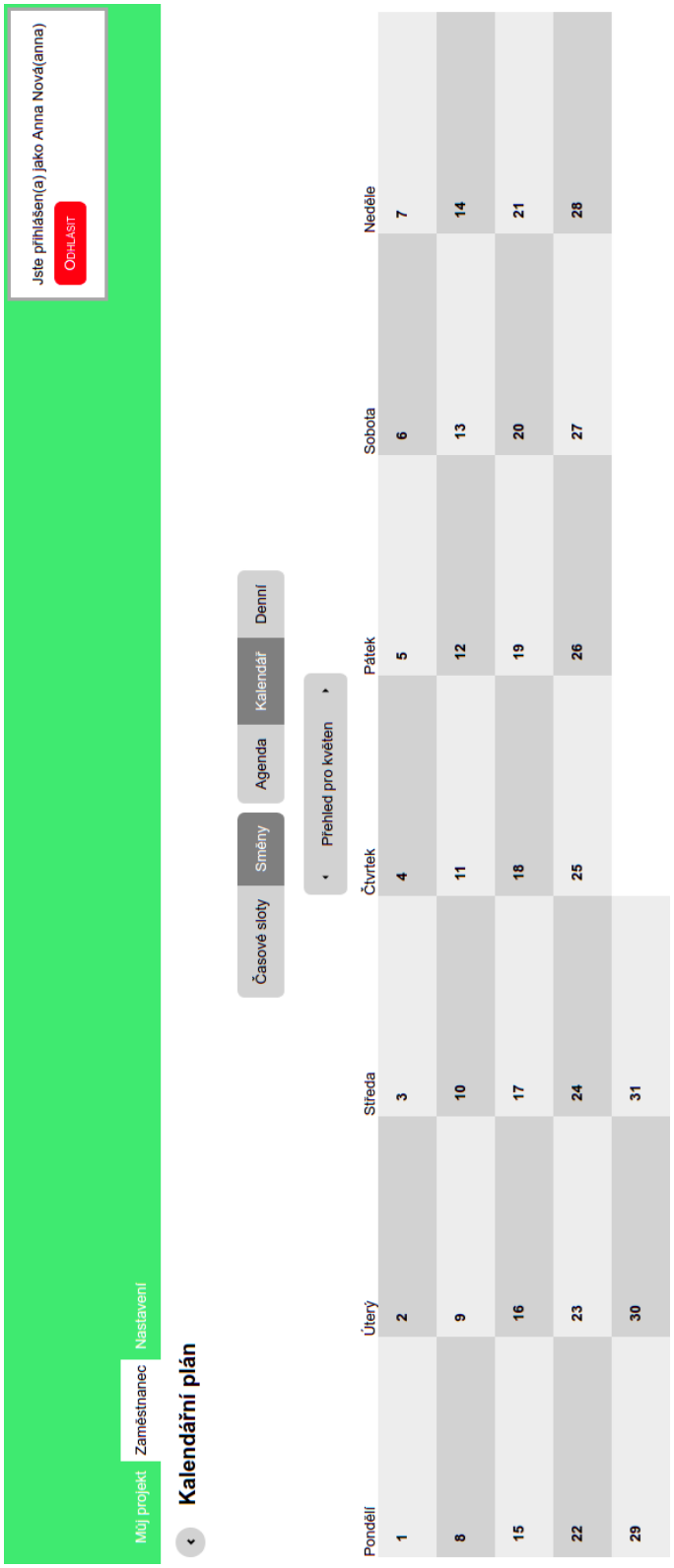
UC Use case

OOP Objektově orientované programování

ERM Employee relationship management

Příloha B

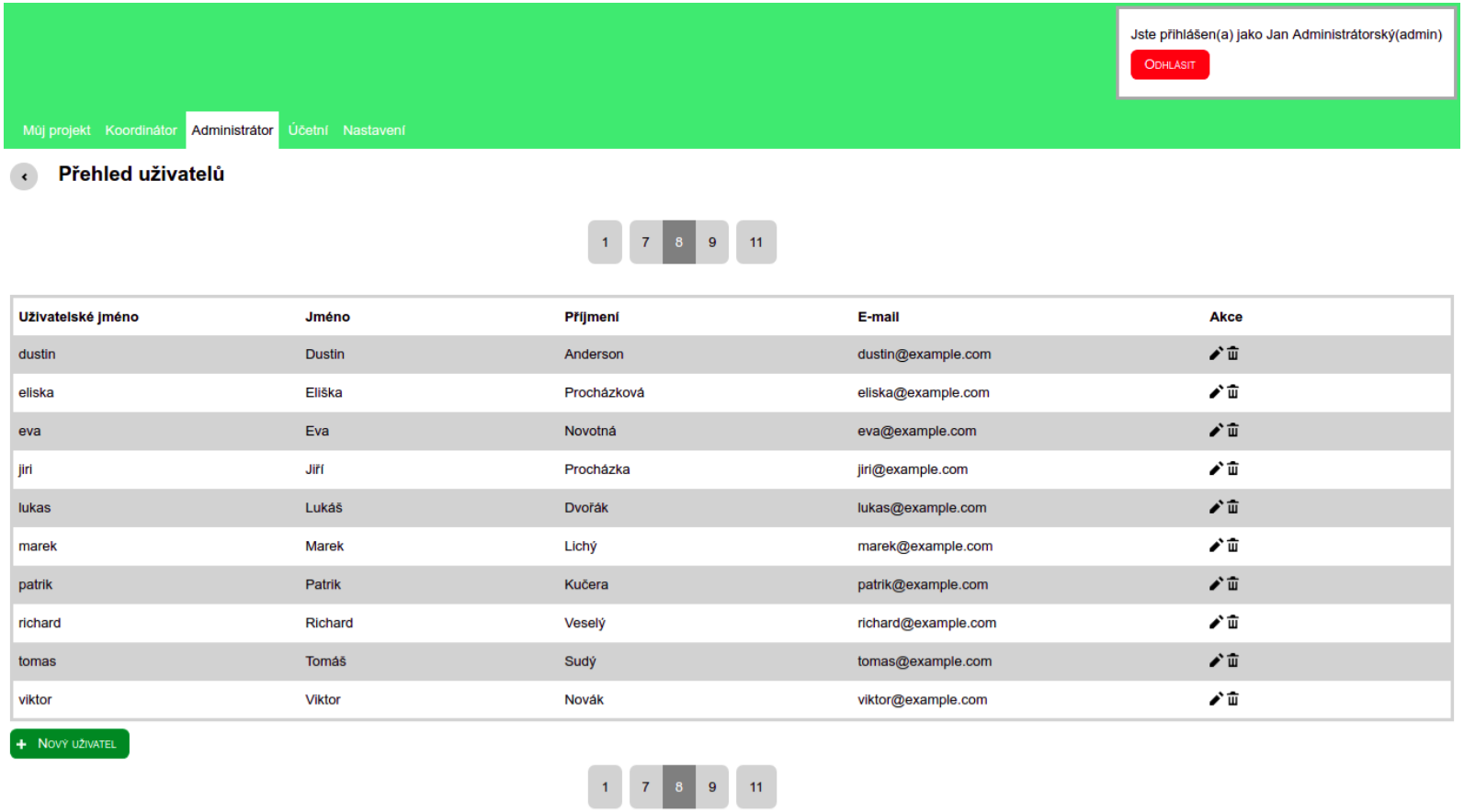
Ukázky aplikace



Obrázek B.1: Ukázka kalendáře

Obrázek B.2: Sekce zaměstnance

Obrázek B.3: Žádosti k vyřízení



Obrázek B.4: Přehled uživatelů

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... (prázdný, zachován kvůli zachování struktury)
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `impl` ..... ZIP soubor (z důvodu zachování symlinků)
      - `app` ..... konfigurační soubory a Twig šablony
      - `bin` ..... spustitelné binární soubory
      - `src` ..... zdrojové soubory aplikace
        - `AppBundle`
          - `Entity` ..... Doctrine entity
          - `Controller` ..... kontrolery
          - `Exception` ..... Výjimky
          - `Form` ..... třídy reprezentující formuláře
          - `Repository` ..... repozitáře entit
          - `Resources` ..... symbolický link pro /web
          - `Service` ..... služby
          - `Twig`
      - `tests` ..... PHPUnit testy
      - `web` ..... veřejně přístupné soubory
        - `bundles`
          - `app`
            - `css` ..... kaskádové styly
            - `js` ..... JavaScriptové skripty
            - `img` ..... obrázky
- `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF