


# Informační systém pro správu budov

Petr Hlaváč

---

Bakalářská práce  
2014

 Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky

---

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Petr Hlaváč**  
Osobní číslo: **A11217**  
Studijní program: **B3902 Inženýrská informatika**  
Studijní obor: **Informační a řídicí technologie**  
Forma studia: **kombinovaná**

Téma práce: **Informační systém pro správu budovy**

## Zásady pro vypracování:

1. Seznamte se důkladně s databázovým prostředím MySQL.
2. Provedte podrobnou analýzu úlohy.
3. Vytvořte databázovou aplikaci pro správu budovy v MySQL.
4. Vytvořená aplikace bude plně funkční a bude evidovat podpůrné činnosti spojené se správou budovy, zahrnující řízení obslužných činností, jak interní tak externí, údržbu budov a potřebných statistik, včetně zabezpečení proti neoprávněnému nakládání se záznamy a objekty aplikace.
5. Ověřte funkčnost vytvořeného informačního systému na dostatečném množství testovacích dat.

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. Merz, H., Hansenmann, T., Hubner, C. **Automatizované systémy budov. GRADA, 2007. ISBN 978-80-247-2367-9.**
2. **ČSN EN 15 221, Facility management. ČSNi Praha, 2009**
3. Daniels, K. **Technika budov. Jaga group. Bratislava, 520 s., 2003, ISBN 80-88905-60-5**
4. Pankrác, M. **PHP a MySQL bez předchozích znalostí, Computer Press, a. s. 2007, ISBN 978-80-251-1758-3**
5. Gilmore, J., W. **Velká kniha PHP 5 a MySQL , Zoner Press, 3. Vydání, 2011, ISBN 978-80-7413-163-9**
6. Welling, L., Thompson, L. **PHP and MySQL Web Development, Sams Publishing, 2003, ISBN 0-672-32525-X**
7. **MariaDB Documentation [online]. Dostupné z WWW: [https://mariadb.com/kb/en/mariadb-documentation/], poslední revize: 30.1.2014.**
8. **Dokumentace frameworku Nette. [online]. Dostupné z WWW: [http://doc.nette.org/cs/], poslední revize: 30.1.2014.**

Vedoucí bakalářské práce: **prof. Ing. Dagmar Janáčová, CSc.**  
Ústav automatizace a řídicí techniky


Datum zadání bakalářské práce: **28. února 2014**

Termín odevzdání bakalářské práce: **13. června 2014**

Ve Zlíně dne 28. února 2014

  
prof. Ing. Vladimír Vašek, CSc.  
*děkan*



  
prof. Ing. Vladimír Vašek, CSc.  
*ředitel ústavu*

## ABSTRAKT

Tématem této práce je analýza, návrh a implementace informačního systému určeného pro správu budov. Vytvořený informační systém poskytuje podporu službám facility managementu a obsahuje funkce pro plánování úkolů, správu kontaktů a veškeré dokumentace. Systém je implementován formou webová aplikace v jazyce PHP, je založen na frameworku Nette a využívá databázový server MySQL. Uživatelské rozhraní aplikace je tvořeno pomocí frameworku Bootstrap.

### *Klíčová slova:*

Facility management, informační systém, PHP, Nette, správa budov, podpůrná činnost

## ABSTRACT

The aim of this work is the analysis, design and implementation of the information system intended for the building management. Created information system provides support to facility management services and includes functions for the task scheduling, contact management and documentation management.

The system is implemented in the form of a web application in PHP, Nette framework and MySQL database server are used. User interface is formed by a framework Bootstrap.

### *Keywords:*

Facility Management, Information system, PHP, Nette, building management, support of services

**Poděkování**

Jsem velmi rád, že na tomto místě mohu poděkovat vedoucí své práce, paní prof. Ing. Dagmar Janáčové, CSc., za obětavost, připomínky a cenné rady při vedení mé bakalářské práce. Dále bych chtěl poděkovat Veronice, Oláfkovi a Lilian za neocenitelnou pomoc a trpělivost.

**Prohlašuji, že**

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo - bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji**

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně dne 13. června 2014

.....  
podpis diplomanta

**OBSAH**

<b>ÚVOD</b> .....	<b>9</b>
<b>I TEORETICKÁ ČÁST</b> .....	<b>9</b>
<b>1 LITERÁRNÍ STUDIE</b> .....	<b>11</b>
1.1 FACILITY MANAGEMENT .....	11
1.1.1 Historie .....	13
1.1.2 Současná situace .....	13
1.1.3 Legislativa .....	13
1.1.4 Facility manager .....	15
1.1.5 Přínos .....	15
1.2 INFORMAČNÍ SYSTÉMY URČENÉ PRO FM .....	16
1.2.1 Základní rozdělení systémů .....	16
1.2.2 Přehled dostupných řešení .....	17
1.2.3 Společné prvky všech systémů .....	18
1.3 ZHODNOCENÍ .....	18
<b>2 TECHNOLOGIE PRO TVORBU APLIKACE</b> .....	<b>20</b>
2.1 ARCHITEKTURA MVC .....	20
2.1.1 Model .....	20
2.1.2 View .....	21
2.1.3 Controller .....	21
2.2 PHP .....	21
2.2.1 Historie .....	22
2.2.2 Výhody .....	22
2.3 MYSQL .....	22
2.3.1 Alternativní databázový systém MariaDB .....	23
2.4 MYSQL WORKBENCH .....	23
2.5 NETTE FRAMEWORK .....	24
2.5.1 Historie .....	24
2.5.2 Životní cyklus aplikace .....	25
2.5.3 Životní cyklus presenteru .....	25
2.5.4 Podpora přístupu k databázi .....	27
2.5.5 Šablonovací systém Latte .....	28
2.5.6 Konfigurace .....	29
2.6 TECHNOLOGIE PRO TVORBU FRONT-END ČÁSTI .....	29
2.7 NETTE TESTER .....	31
<b>II PROJEKTOVÁ ČÁST</b> .....	<b>33</b>

<b>3</b>	<b>POŽADAVKY NA APLIKACI</b> .....	<b>35</b>
3.1	NASTAVENÍ PROSTŘEDÍ .....	35
3.2	SPRÁVA DOKUMENTŮ .....	35
3.3	PLÁNOVÁNÍ ÚKOLŮ .....	36
3.4	ZÁZNAM O HAVÁRII.....	36
3.5	UŽIVATELE SYSTÉMU A JEJICH UŽIVATELSKÉ ROLE.....	37
3.6	UPOZORNĚNÍ NA ZAČÁTEK ÚKOLU .....	37
<b>4</b>	<b>NÁVRH ZÁKLADNÍCH ENTIT</b> .....	<b>38</b>
4.1	FM PRODUKT A JEHO UMÍSTĚNÍ.....	38
4.2	PROVÁDĚNÁ ČINNOST .....	38
4.3	KONTAKT.....	38
4.4	PRIORITA ÚKOLU .....	38
4.5	DOKUMENTY .....	39
4.6	PLÁNOVANÁ ÚLOHA .....	39
4.7	ZÁZNAM O PROVEDENÍ PLÁNOVANÉ ÚLOHY .....	40
4.8	PERIODICKÁ ÚLOHA.....	40
4.9	ZÁZNAM O PROVEDENÍ PERIODICKÉ ÚLOHY .....	40
4.10	ZÁZNAM O HAVÁRII.....	41
4.11	UŽIVATEL A UŽIVATELSKÉ ROLE.....	41
<b>5</b>	<b>EER MODEL DATABÁZE</b> .....	<b>42</b>
<b>6</b>	<b>NÁVRH UŽIVATELSKÉHO ROZHRANÍ APLIKACE</b> .....	<b>47</b>
<b>7</b>	<b>NÁVRH STRUKTURY APLIKACE</b> .....	<b>54</b>
7.1	MODELOVÁ VRSTVA .....	54
7.2	ŘÍDÍCÍ VRSTVA .....	55
7.3	UŽIVATELSKÉ ROLE.....	55
<b>8</b>	<b>TESTY</b> .....	<b>58</b>
8.1	TESTOVÁNÍ PŘIHLÁŠENÍ .....	58
8.2	TESTOVÁNÍ UŽIVATELSKÝCH PRÁV .....	59
8.3	TESTOVÁNÍ PERIODY ÚKOLU.....	59
	<b>ZÁVĚR</b> .....	<b>61</b>
	<b>SEZNAM POUŽITÉ LITERATURY</b> .....	<b>62</b>
	<b>SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK</b> .....	<b>67</b>
	<b>SEZNAM OBRÁZKŮ</b> .....	<b>68</b>
	<b>SEZNAM PŘÍLOH</b> .....	<b>69</b>

## ÚVOD

Téměř všechny firmy jsou dnes nuceny zabývat se otázkou: „Jak minimalizovat náklady spojené se zabezpečením správy a údržby nemovitostí?“. Motivací je především snaha ušetřit na nákladech spojených s údržbou nemovitostí a vnitřního vybavení. Vzniklé úspory mohou sloužit jako finanční rezerva při nepříznivé situaci na trhu, nebo také mohou být vhodně investovány do rozvoje společnosti, či mohou sloužit jako prostředky pro provádění nákladnějších oprav a celkové revitalizace budov. Revitalizovaná budova se vyznačuje nižší energetickou náročností, která se pozitivně projeví na snížení nákladů za energie a vytápění. Další oblast investic může představovat podpora technologií ohleduplných k životnímu prostředí. Ekologicky šetrné technologie snižují produkci odpadu, které se projeví na snížení nákladů za zpracování odpadů.

Vhodnou cestu k nalezení úspor představuje facility management, jehož úkolem je řídit podpůrné činnosti organizace s cílem dosáhnout optimálního řízení mezi procesy, pracovníky a prostorem. Umožňuje navrhnout podpůrné procesy tak, aby jako celek tvořily synergickou kombinaci. Sladěním podpůrných činností dochází ke snížení nákladovosti a ke zvýšení celkové efektivity. Zvyšuje se nejen životnost výrobních i nevýrobních zařízení, ale i celková produktivita pracovníků. Díky kvalitnějšímu využití plochy, sníženému opotřebení, menší poruchovosti a snížení energetické náročnosti, dochází ke vzniku významných úspor.

Rozpětí poskytovaných služeb facility managementu je ovšem velmi rozsáhlé. Poskytuje služby jak pracovníkům (lidské zdroje), tak i služby procesům (činnostem) a v neposlední řadě i služby prostorům (prostředí). Ve své práci se proto soustředím pouze na služby uplatněné při správě budovy. Cílem bude tvorba informačního systému pro správu budovy, který bude navržen s ohledem na podporu služeb facility managementu. Při návrhu aplikace budou použity moderní nástroje jako jsou skriptovací jazyk PHP, databázový server MySQL, frameworky Nette, Twitter Bootstrap a jQuery. Informační systém bude navržen s ohledem na architekturu MVC s podporou modularity. Úkolem informačního systému bude řízení podpůrných činností firmy a bude obsahovat nástroje pro inventarizaci zařízení, evidenci pracovníků, správu dokumentace, plánování prací s možností včasného připomenutí e-mailem a na obrazovku aplikace včetně možnosti tvorby statistik.

# I. TEORETICKÁ ČÁST

## 1 LITERÁRNÍ STUDIE

Společnosti jsou v dnešní době stále více nuceny zabývat se efektivou své výroby a nabídkou poskytovaných služeb. S rostoucí globalizací čelí sílícímu tlaku konkurence. Konkurenční společnosti pak zcela logicky tlačí na cenu výrobků i služeb a zároveň kladou větší požadavky na kvalitu poskytovaných služeb. Rostoucí kvalita životní úrovně zvedá nároky zaměstnanců na kvalitu pracovního prostředí. Společnosti se snaží vzniklou situaci vyhodnotit a hledají způsob, jak vhodně zareagovat. Optimalizovat pouze hlavní výrobní proces přitom nestačí. Nutné je podpořit všechny vedlejší činnosti podílející se na vlastním provozu firmy. Způsobem, jakým lze řešit tuto situaci, nabízí Facility Management (FM).

FM se zabývá způsobem jak podpořit a optimalizovat vedlejší činnosti ve společnosti. Jednu oblast služeb poskytovanou FM tvoří služby spojené se zabezpečením chodu spravované budovy. Každá společnost totiž zpravidla musí disponovat prostorem, ve kterém uskutečňuje svou výrobní či administrativní činnost. Při návrhu těchto budov se používají metody integrovaného plánování, kdy dochází k vysoké tepelné, akustické i vizuální pohodě. Důležitým faktorem je i elektromagnetická kompatibilita a ekologická zátěž. Správný návrh také zabraňuje vzniku syndromu nemocné budovy[1].

S velikostí spravovaných prostor roste náročnost zabezpečení efektivního chodu. Správu je pak nutné rozdělit mezi více zodpovědných osob. Tyto osoby však musí disponovat nástrojem, kterým by dokázali sladit své činnosti tak, aby nedocházelo ke konfliktům či haváriím, které by výrazně narušovali chod společnosti.

Odpovědní pracovníci mohou svěřenou oblast správy řešit různými nástroji. Lze si psát písemné poznámky, použít jednoduché úkolovníky či vést záznamy v tabulkovém procesoru („Excelu“). Tento způsob rozhodně nelze nazvat optimálním.

Optimální řešení spočívá ve využití vhodného informačního systému. Zvolený informační systém musí být schopen nabídnout takové funkce, kterými by bylo možné pokrýt všechnu požadovanou činnost. Měl by nabízet intuitivní a přehledné uživatelské rozhraní. Důležitým kritériem je i zabezpečení systému proti neoprávněné manipulaci. Mezi požadované vlastnosti aplikace patří schopnost zajišťovat evidenci a plánování úkolů, evidenci havarijních stavů, správu dokumentů a možnost sestavení přehledu o jednotlivých činnostech za dané období.

### 1.1 Facility management

Vyslovit přesnou definici FM není snadné. Jednotlivé země či regiony definují FM různými způsoby. Nejčastější definice však zní:

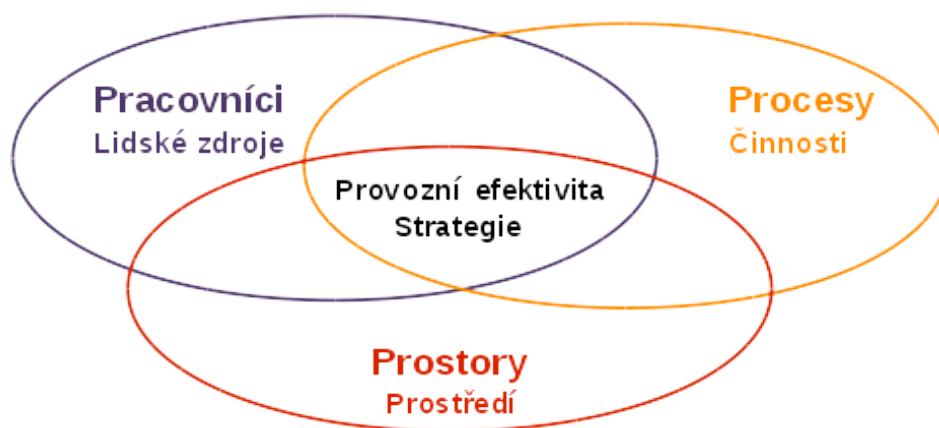
„Facility management je souborem navzájem integrovaných prvků, pokrývajících všechny činnosti, spojené s komplexní správou nemovitostí.“[2]

Facility management lze také popsat jako:

„Integrace činností v rámci organizace k zajištění a rozvoji sjednaných služeb, které podporují a zvyšují efektivitu vlastní základní činnosti“ [3]

FM je multioborová disciplína, zabývající se podpůrnými činnostmi v organizaci s cílem tyto činnosti v maximální možné míře zefektivnit. Zaměřuje se na tři hlavní oblasti:

- Pracovníci (Lidské zdroje)
- Procesy (Činnosti)
- Prostory (Prostředí)



Obr. 1. Facility management: 3P

Oblast „Prostory“ je typická pro FM. Jedním z cílů je totiž posílení procesů, pomocí nichž pracoviště a pracovníci podají nejlepší výkony, které se odrazí na pozitivním ekonomickém růstu a celkovému úspěchu organizace[4]. Typické uplatnění nachází při zajištění provozu, údržby, servisu výrobních a provozních zařízení, při zajištění odborné technické prohlídky včetně revizí, monitoringu, zajištění ochrany majetku (zabezpečovací zařízení, ostraha) a správa vozového parku.

Procesy FM se uplatňují už při návrhu nových a také při revitalizaci stávajících budov. Jedním z těchto procesů je například Energy management. Energy management se zabývá úsporou a řízením spotřeby energie. To je důležité zvláště při návrhu automatizace budov. Automatizované budovy představují další způsob ke zvyšování pohodlí zaměstnanců a ke snížení energetické náročnosti. Snížení energetické náročnosti představuje pozitivní krok s ohledem na ekologickou zátěž[5].

Správa budovy a činnosti s ní spojené představují jednu disciplínu v rámci FM. Objevují se zde potřeby jako je například evidence majetku, dodavatelů a dokumentace, potřeby plánování periodických kontrol a revizí, potřeby plánování rozsáhlejších oprav a investic, potřeby dokumentace havarijních stavů. Evidenci každé jednotlivé činnosti je vhodné provádět v jednotném centrálním registru. Výhodou jednotného systému

je přehlednost, unifikovanost ve smyslu uživatelského rozhraní, způsob zálohování a zabezpečení.

### 1.1.1 Historie

Kořeny FM lze hledat v USA. V druhé polovině sedmdesátých let, vyvstaly nové nároky na správce při budování nových typů kanceláří označovaných jako open-space. Pro tento typ řešení nebylo připraveno žádné řešení. Neexistovaly školy nebo organizace, kde by její členové mohli dostat odpovědi na své otázky. V květnu 1980 proto vznikla organizace pojmenovaná jako National Facility Management Association. Tato organizace se skládala z třicítky zájemců o FM. O rok později v reakci na vlnu kanadských zájemců došlo k rozšíření této organizace a jejímu přejmenování na International Facility Management Association. Pod tímto názvem ji lze nalézt i dnes[6].

Postupem času nabrala IFMA na síle i na velikosti. Došlo k rozšíření nejen v rámci USA, ale i celé Evropy. V současné době je členem organizace IFMA 18.000 členů z celkem 67 zemí světa ve kterých má 130 poboček[7].

### 1.1.2 Současná situace

V současné době má většina společností komplexně provázány zařízení, činnosti a systémy. Toto provázání však vykazuje značné rozpory a duplicity, které ve výsledku redukuje efektivitu provozu. Reálně je možné stávající systém přebudovat než kompletně zrušit a vytvořit nový. Tvorbu kompletně nového systému nejsou společnosti ochotny akceptovat[8].

### 1.1.3 Legislativa

Pro facility management je dostupná EU norma CEN/EN 15221 (ČSN EN 15221), která se skládá ze sedmi podskupin popisujících jednotlivé oblasti FM. Podskupiny jsou[3][9][10]:

- ČSN EN 15221-1 - Definice a terminologie

Představuje hlavní dokument norem facility managementu. Účelem je stanovit termíny v oblasti facility managementu, které mají za cíl zlepšit komunikaci mezi investory, zvýšit efektivitu procesů, rozvíjet nástroje a systémy.

- ČSN EN 15221-2 - Průvodce přípravou FM smluv

Stanovuje návod na přípravu efektivních FM smluv. Stanovuje vztah mezi organizací (FM klient) a poskytovatelem služby (FM poskytovatel).

- ČSN EN 15221-3 - Kvalita ve FM

Zavádí nový pojem „FM produkt“, který představuje přesně vymezenou a měřitelnou FM službu. Hlavním cílem je sledování výkonu a kontroly FM produktů.

- ČSN EN 15221-4 - Kategorizace FM

Zavádí kategorizaci FM produktů. FM produkty jsou zařazeny do skupin a následně jsou specifikovány jak procesně, tak i nákladově.

- ČSN EN 15221-5 - Procesy ve FM

Specifikuje procesní standardy se zdůrazněním jednotlivých etap procesu. Stanovuje rozdíly mezi účinností procesu a jeho efektivitou. Zdůrazňuje význam přesné definice vstupů a výstupů procesu.

- ČSN EN 15221-6 - Měření prostor ve FM

Zaměřuje se na popis standardů měření prostorů a ploch. Zavádí kategorizaci prostor dle užitečnosti pro uživatele.

- ČSN EN 15221-7 - Benchmarking ve FM

Zavádí pojem benchmarking (porovnávání). Umožňuje efektivně porovnávat údaje vzniklé z předchozích částí do podoby srovnatelné základny pro vyhodnocení kvality a efektivity služeb.

První skupina ČSN EN 15221-1 definuje celkem tři základní úrovně[8].

- **Strategická úroveň** stanovuje politiku a strategii FM v návaznosti na strategii a potřeby společnosti. Definuje jakým způsobem má být FM začleněno do systému řízení a jakým způsobem má být řízeno. Strategická úroveň má na starosti facility manažer.
- **Taktická úroveň** zajišťuje všechny potřebné standardy, předpisy, plány a pravidla. Určuje postupy při řešení havarijních a mimořádných stavů. Taktická úroveň stanovuje a vyhodnocuje systémy hlášení a jasně určuje povinnosti včetně kompetencí.
- **Provozní úroveň** zajišťuje monitorování a kontrolu dodávaných služeb. Operativně řeší případně zjištěné neshody. V rámci provozní úrovně jsou definovány dispečinky včetně call center. Provozní úroveň také zajišťuje styk a komunikaci s klienty.

Dle typu požadavků jsou základní FM procesy tříděny na dvě oblasti. Jedná se o takzvané „tvrdé a měkké“ služby[8].

- Mezi **tvrdé služby** řadíme prostor a infrastrukturu. Zde jsou definovány oblasti jako například ubytovací a prostorové služby, pracoviště, technická infrastruktura, čištění a úklid.
- Mezi **měkké služby** řadíme lidi a organizace. Zde jsou definovány oblasti jako zdraví, bezpečnost, ochrana, péče o uživatele objektu, interní logistika.

#### 1.1.4 Facility manager

Facility manager je řídicí pracovník zodpovědný za strategii FM. Zodpovídá tedy za tvorbu a dodržování postupů v organizaci z hlediska FM[6]. IFMA poskytuje facility managerům služby certifikace na základě jejich zkušeností a vědomostí ve třech kategoriích.

Jedná se o tyto kategorie[11]:

- FMP (Facility Management Professional)
- SFP (Sustainability Facility Professional)
- CFM (Certified Facility Manager)

#### 1.1.5 Přínos

Podporou vedlejších činností organizace se dosahuje výrazné podpory hlavním činností, které jsou předmětem zisku. Dochází ke snížení nákladovosti a ke zvýšení celkové efektivity. Podporou FM procesů dochází také ke zlepšení pracovních podmínek zaměstnanců. Spokojený zaměstnanec podává kvalitnější výkon a je lépe motivován ke své práci. Hlavním přínosem zavedení facility managementu pro organizaci jsou[8]:

- zvýšení efektivity hlavních činností organizace
- zvýšení životnosti výrobních i nevýrobních zařízení
- vyšší produktivita pracovníků
- přehled o nákladech a úrovni služeb
- efektivní využití prostorů
- zisk z nadstandardních služeb
- zhodnocení uvolněných prostorů

Neméně důležitým přínosem pro organizaci jsou úspory které vznikají díky[8]:

- kvalitnějšímu využití plochy

- snížení energetické náročnosti
- nižšímu opotřebování
- menší poruchovosti
- rychlejší reakční době

## 1.2 Informační systémy určené pro FM

S rostoucím zájmem společností o FM bez ohledu na jejich velikost, roste i nabídka firem zabývajících se problematikou vývoje, podpory, správy a implementace FM systémů. Tyto firmy zajišťují kompletní servis prací představující studie, analýzy, implementaci a školení. Samozřejmě pak bývá i následná podpora zajišťující zaškolení nových pracovníků i realizaci požadovaných úprav systému.

### 1.2.1 Základní rozdělení systémů

Pro jednotlivé typy systému se používají různé zkratky. Tyto zkratky vyjadřují, jakou problematiku je daný systém schopen řešit. Naší pozornosti by neměl uniknout software, označovaný zkratkami FM, CAFM nebo CMMS.

#### Systemy FM

Systemy označované jako FM, poskytují oporu při řešení problematiky řízení procesů zakázek a objednávek. S jejich pomocí je také zajištěna podpora správy majetku. Tyto systémy bývají většinou implementovány jako modul či nadstavba klasického ERP systému. ERP systémů je na trhu opravdu hodně a většinu z nich tvoří zavedené značky. Konkrétními představiteli těchto systémů jsou SAP R3, Microsoft Dynamics, Helios, Altus Vario[12].

#### Systemy CMMS

Systemy označované CMMS se zaměřují na problematiku FM z pohledu plánování údržby. Cílem těchto systémů je zajistit plynulý a bezpečný provoz budov. Nasazují se v podnicích, kde je rozhodující plynulost výrobního procesu. Vytváří podporu personálu provádějícímu činnosti spojené s údržbou výrobních zařízení a zařízení s nimi souvisejícími. Umožňují sledování zásob a prostředků určených na údržbu. Dokáží sledovat a následně vyhodnotit pracovní vytíženost jednotlivých pracovníků či pracovních skupin. Nabízejí kompletní pracovní postup údržby, který zahrnuje registry majetku, skladové systémy a objednávkové systémy[13].

## Systémy CAFM

Systémy CAFM navazují na aplikace CMMS. Nabízí kompletní řešení pro všechny oblasti působnosti organizace jako jsou správa a řízení úkolů, správa a rezervace místností, správa vozového parku.

Základem těchto systémů je integrace s GIS a CAD systémy. Výhodou této integrace je možnost zpracovávání grafických informací. Vizualizace údajů může mít někdy větší vypovídající hodnotu, než jejich textová interpretace. Tato vlastnost může být neocenitelná například při výpočtu plochy potřebné pro malování místnosti. Díky propojení s CAD systémem, může tento systém pracovat s daty přímo uloženými v CAFM. Výpočet plochy je pak triviální záležitostí, neboť každý CAD systém disponuje funkcí pro výpočet plochy v uzavřeném polygonu. CAFM systémy umožňují členění budov či pozemků na jednotlivé dílčí části. Bez problémů tak může být rozlišena například konkrétní místnost v rámci svého patra resp. celé budovy. Každý takto sledovaný subjekt, u sebe obsahuje všechny relevantní údaje včetně výkresové dokumentace. Jednoduchým pohledem tak lze získat jasný přehled o činnostech na úrovni těchto elementárních částí, až po přehled jako celku.

Klíčovou vlastností pro tyto systémy je způsob zpracování a správa dokumentů s následnou možností sdílení. Jednotná databáze umožňuje sjednotit značení a kódování. Externí uživatelé systému (včetně zákazníků) s patřičným oprávněním, mají možnost přes jednotné rozhraní přistupovat k dokumentaci. Pro externí pracovníky jsou takto dostupné všechny podklady důležité pro realizaci zakázky. Takto je i zajištěna možnost nahrávat do systému v průběhu realizace zakázky změnové dokumenty zaznamenávající aktuální stav prováděných prací. V případě pronájmu budov je takto umožněno nájemcům uvádět do systému aktuální údaje týkající se spotřeby vody či energií.

Velké systémy obsahují přímo podporu pro vyřizování a zpracování požadavků ve formě Helpdesku. Tento systém poskytuje prostředí pro zadávání problémů s možností následného sledování průběhu zpracování až po jeho úspěšné vyřízení. Samozřejmostí je možnost získání konkrétních přehledů a statistik. Toto rozhraní je využíváno nejen pro kontakt se zákazníky, ale i pro interní či externí pracovníky[14].

### 1.2.2 Přehled dostupných řešení

Průzkumem trhu jednoznačně vyplývá existence nepřeberného množství systémů. Na trhu se objevují jednak tuzemské, tak i zahraniční firmy zastupované svými zprostředkovateli.

## Zahraníční systémy

Dle informací tvůrce, představuje špičku mezi IS systémy TIFM systém Archibus.[15] Tento systém je svým rozsahem určen pro velké organizace. Dalším představitelem zahraničního systému pro velké organizace je Manhattan CenterStone, eFACiLiTY od společnosti SIERRA.

## Tuzemské systémy

Mezi tuzemské tvůrce patří systém od společnosti CadStudio, pit-FM, MaintPlan CMMS od společnosti EasySoft, Gisa-fm od společnosti AZIMUT CZ, Building Manager od společnosti IC Software a MaintPlan CMMS od společnosti EasySoft.

### 1.2.3 Společné prvky všech systémů

Společným znakem všech systémů je jejich modularita. Každý výrobce nabízí základní modul (jádro) obsahující sadu základních funkcí a následně rozšiřující sadu modulů. Každý z modulů řeší svou specifickou funkcionalitu. Díky této vlastnosti je možné systém volbou modulů přizpůsobit konkrétním potřebám. Druhým aspektem je samozřejmě cena. V případě zakoupení jen potřebných modulů je možné ovlivnit výslednou cenu řešení.

Dalším společným znakem je podpora webových technologií. Každý systém disponuje uživatelským rozhraním dostupným přímo přes internet za pomoci běžného webového prohlížeče, nebo podporuje připojení pomocí VPN.

Při pořízení kteréhokoliv systému je vždy nutné zaplatit licenční poplatek, který se liší v závislosti na velikosti poskytnutých služeb či objednaných modulů. Samotné pořízení systému tedy vždy představuje určitou finanční zátěž.

## 1.3 Zhodnocení

Problematika volby konkrétního komerčního IS není nikterak snadná záležitost. Existuje nepřehledné množství individuálních požadavků daných různými potřebami organizací. Specialisté v oboru při výběru více či méně inklinují k řešení, se kterými již mají své zkušenosti.

IS nabízené jednotlivými firmami obsahují řadu pokročilých funkcionalit. U velkých systému, jako je například Archibus, je zřetelné, že se jedná o vyladěný systém, který je vyvíjen už řadu let. Všechny nabízené IS dokážou díky modularitě splnit požadované funkce. Finanční zátěž spojená s pořízením těchto systémů ovšem není zanedbatelná.

Díky masovému rozšíření a dostupnosti sítě internet, lze zvolit strategii, kdy bude aplikace poskytována zcela zdarma. Hrazené budou pouze služby spojené s úpravou

aplikace dle individuálního přání zákazníka, či tvorbou nového modulu aplikace.

Vhodným krokem je vytvoření vlastní aplikace, která bude poskytovat podporu FM službám, potřebných pro efektivní správu budovy. Tato aplikace bude umožňovat plánování úkolů s podporou ukládání dokumentů, možnost vést záznamy o haváriích a notifikace uživatelů se zajištěným systémem řízení uživatelských rolí.

Jako nejvýhodnější se jeví po vzoru konkurence použít pro IS formát webové aplikace. Ze svých zkušeností při tvorbě aplikací založených na webových technologiích, navrhuji použít jako skriptovací jazyk, jazyk PHP. Pro zajištění perzistence dat volím databázový systém MySQL, respektive jeho „fork“ MariaDB. Použití PHP a MySQL představuje nejčastěji používané prostředky při tvorbě webových aplikací[16].

Při tvorbě aplikace lze postupovat způsobem, kdy veškeré kódy napíšu sám. Druhý způsob představuje použití stabilního frameworku, který bude zajišťovat podporu aplikaci v případě dalšího vývoje. Framework musí být dobře zabezpečen proti různým typům útoků a musí zajišťovat podporu moderním architekturám (například MVC). Dále by měl umožňovat tvorbu znovupoužitelných a dobře testovatelných kódů.

Všechny nároky kladené na framework splňuje Nette Framework. Tento framework volím i s přihlédnutím k jeho českému původu a osobním zkušenostem.

## 2 TECHNOLOGIE PRO TVORBU APLIKACE

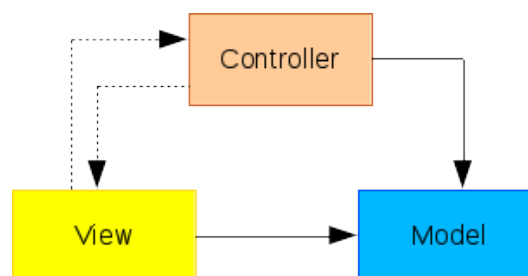
### 2.1 Architektura MVC

Architektura MVC popisuje způsob, jakým by měla být aplikace při návrhu rozdělena do logických částí tak, aby změna jedné části neměla vliv na chod zbylých částí. Nejedná se však o návrhový vzor (design pattern), ale o popis celé architektury.

Architekturu MVC (Model-View-Controller) popsal v roce 1979 Trygve Reenskaug. V dnešní době tato architektura nabírá velmi na popularitě[17]. Jak je již z názvu patrné, architektura MVC se skládá ze tří hlavních částí. Jedná se o:

- Model
- View
- Controller

Základní myšlenkou tohoto návrhu je dosáhnout oddělení logických částí aplikace jejich rozdělením. Aplikační logiku bude představovat model, obslužnou logiku controller a zobrazení dat obstarává view[18].



Obr. 2. MVC

Rozdělení na logické části přináší řadu výhod. Zvýší se přehlednost a znovupoužitelnost kódu. Každou dílčí část lze také testovat jednoduchými testy. Jednotlivé části lze upravovat nezávisle na ostatních. Výsledek tvoří pomyslnou stavebnici, ve které do sebe jednotlivé části přesně zapadají.

Ve vlastní implementaci MVC nemusí být jednotlivé vrstvy obsaženy pouze v rámci jedné třídy. Jednu vrstvu může reprezentovat více tříd a jedna třída může obsahovat i průnik více vrstev. Hranice mezi vrstvami je tedy dána jejich „významem“, nikoliv „fyzickou“ realizací.

#### 2.1.1 Model

Modelová vrstva zahrnuje data a doménovou (business) logiku. Pro tuto vrstvu je typické, že nemá absolutně žádnou přímou nebo nepřímou vazbu na View či Controller. Úkolem této vrstvy je pouze poskytnout abstrakci datům, se kterými aplikace pracuje.

Modelová vrstva v sobě může obsahovat rozhraní pro perzistenci dat. Toto rozhraní bývá u většiny webových aplikací reprezentováno databází (SŘBS).

### 2.1.2 View

Tato vrstva má za úkol data získané z modelové vrstvy vhodným způsobem interpretovat uživateli. U webových aplikací bývá výstupem aplikace HTML kód, obrázek či například PDF dokument.

U většiny webových aplikací, tvoří vrstvu View šablonovací systém HTML šablon. Šablonovacím systémem může být i sám skriptovací jazyk (např. PHP), nicméně tento způsob nebývá příliš efektivní (z důvodů přehlednosti, výkonově se jedná o nejlepší variantu). Pro efektivní práci se šablonami využívají moderní aplikace speciální šablonovací systémy. Pro jazyk PHP existuje řada knihoven jako jsou například TemplatePower, Smarty či Latte, které je součástí frameworku Nette. Většina moderních šablonovacích systémů ve skutečnosti zpracuje šablony do nativního PHP kódu.

### 2.1.3 Controller

Controller je řadič zajišťující reakci na vstupní události. Představuje vstupní bod aplikace. Jeho úkolem je na základě vstupní akce zajistit požadovanou odezvu. U webových aplikací je tato vstupní akce reprezentována URL adresou. Úkolem Controlleru je zavolat požadovanou akci, která například uloží nový stav do modelové vrstvy a zajistí zavolání správného View. Někdy controller zprostředkovává data z modelové vrstvy do vrstvy View.

## 2.2 PHP

PHP (celým názvem PHP: Hypertext Preprocessor) je serverově orientovaný skriptovací jazyk, navržený pro tvorbu webových aplikací, vyznačující se svou jednoduchostí a rozšířeností[19]. PHP je distribuováno jako open source projekt. Je tedy dostupné bez licenčních poplatků zcela zdarma. Díky otevřenosti kódu, jsou případné bezpečnostní chyby snáze nalezeny a poté odstraněny. Jádro a knihovny jazyka jsou napsány v jazyce C, díky tomu se dá říct, že je jazyk PHP je nezávislý na platformě (neboť pro většinu platforem existuje překladač jazyka C). Aplikace napsané v PHP lze tedy provozovat na všech dominujících systémech jako jsou UNIX, GNU Linux, Microsoft Windows.

Původní syntaxe jazyka čerpala z jazyka Perl. Proměnné se tedy stejně označují znakem \$. Objektový model je velmi podobný modelu, který používá jazyk Java. Polymorfismus je zajišťován rozhraními (*interface*) a od verze 5.3 jsou podporovány jmenné prostory (*namespace*).

### 2.2.1 Historie

Základy jazyka PHP položil v devadesátých letech minulého století Rasmus Lerdorf. PHP v té době sloužilo nejprve jako nástroj pro generování malé osobní stránky. Původní význam zkratky PHP zněl „Personal Home Page“ (někdy uváděno jako PHP Tools). Krátce na to Rasmus vydává sadu skriptů původně psanou v jazyce Perl a později přepsanou do jazyka C, označovanou jako PHP/FI.

Další milník v historii představovalo v roce 2000 vytvoření nového jádra označeného jako Zend. Díky tomuto jádru byl rapidně zvýšen výkon celého jazyka.

V roce 2004 přichází PHP ve verzi 5.0 s novým jádrem Zend II. Toto jádro přichází s podporou objektového programování a představuje další výrazný posun.

Podpora jmenných prostorů byla přidána v roce 2009 ve verzi PHP 5.3. Jednalo se o další výrazné přiblížení k trendu moderních jazyků.

### 2.2.2 Výhody

Jazyk PHP je určen především pro tvorbu webových aplikací. Dá se však použít i pro konzolové nebo i dektopové aplikace. Díky jeho masové rozšířenosti, je jeho podpora u hostingových společností takřka stoprocentní. Svobodná licence redukuje náklady spojené s náklady na licenční poplatky. PHP má velmi bohatou a propracovanou dokumentaci. Na webu existuje spousta knihoven či návodů na obecné i více specifické problémy. Jazyk PHP podporuje nativně (nebo přes rozhraní) práci s mnoha databázovými systémy.

## 2.3 MySQL

MySQL je velmi rychlý, silný a robustní relačně databázový systém[20]. Je dostupný pro většinu platforem a je licencován pod svobodnou licencí GPL, tak i pod komerční licencí[21]. Výhodou použití databáze MySQL je i jeho cena. Pro nekomerční použití je zdarma. Pro komerční využití se platí licenční poplatek. Tento poplatek je však mnohokrát menší, než u srovnatelných komerčních systémů.

MySQL podporuje více typů datových uložišť (storage engine). Každé toto uložisko vyniká v různých typech operací. Vhodným výběrem lze poměrně snadno dosáhnout navýšení výkonu. Nejpoužívanější uložiska jsou MyISAM a InnoDB[22].

Výchozí formát uložiska dat (storage engine) je od verze 5.5 InnoDB, které nahradilo dříve používaný formát MyISAM[23].

## Formát MyISAM

Formát MyISAM vyniká v rychlosti čtení. Za zmínku stojí podpora fulltextového vyhledávání za pomoci fulltext indexů. Při ukládání dat nezamyká pouze řádek, ale celou tabulku. MyISAM umožňuje automatickou nebo ruční kontrolu a opravu v případě havárie. Umožňuje také přímo kopírovat datové a indexové soubory mezi různými systémy. MyISAM však nepodporuje cizí klíče[22].

## Formát InnoDB

InnoDB je transakční datové uložisko kompatibilní s modelem ACID. ACID popisuje vlastnosti, které mají zajistit, že zpracovávané transakce budou vždy spolehlivě dokončeny. Mezi charakteristické rysy patří podpora cizích klíčů, uzamykání na úrovni řádků, transakce, body obnovení[24][22].

### 2.3.1 Alternativní databázový systém MariaDB

MariaDB je databázový systém od původních tvůrců MySQL, který vzniknul jako „fork“ MySQL. Důvodem jejího vzniku byla snaha o udržení open-source licence GNU GPL[25][26].

MariaDB je s MySQL binárně kompatibilní a do verze 5.5 následuje i číslování MySQL[27]. Na rozdíl od MySQL podporuje více datových uložišť, disponuje větším výkonem. Hlavní předností je open-source licence[28].

Velké linuxové distribuce jako Fedora (od verze 19), Red Hat Enterprise Linux, openSUSE, Arch Linux, nahradily MySQL za MariaDB a nabízejí MariaDB jako výchozí „MySQL kompatibilní“ systém při své instalaci[29].

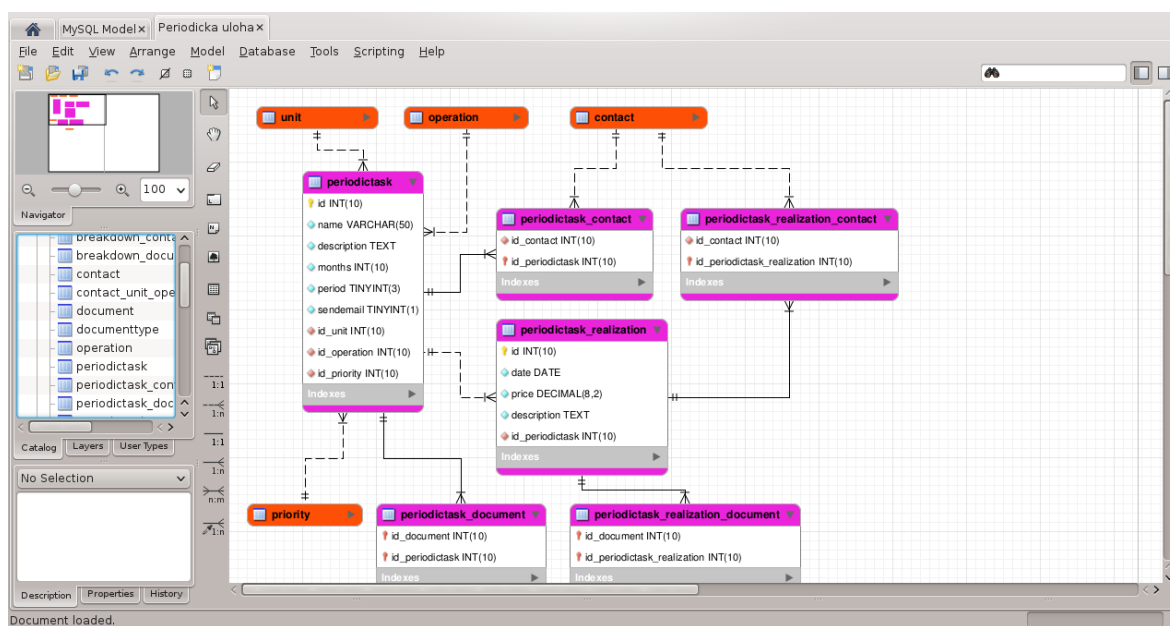
## 2.4 MySQL Workbench

MySQL Workbench je multiplatformní open source (licence GPL) nástroj tvůrců MySQL databáze[30]. Hlavní funkce MySQL Workbench lze rozdělit do třech oblastí. Jsou to:

- Vizuální návrh struktury databáze (EER modely)
- Tvorba, spouštění a následná optimalizace SQL dotazů
- Monitorování a analýza chodu databázového serveru

MySQL Workbench disponuje sadou funkcí pro podporu operací *import*, *export* a *diff* označované jako *forward* a *reverse engineering*. Díky těmto funkcím je možné existující model exportovat jako SQL skript. Porovnáním modelu a databáze lze vytvořit rozdílový SQL *ALTER* skript. Podporována je i možnost vytvoření modelu na základě již

vytvořené databáze. Tyto funkce jsou velmi užitečné při reflektování změn, provedených v modelu do *živé* databáze[31].



Obr. 3. Aplikace MySQL Workbench

## 2.5 Nette Framework

Nette Framework je open source PHP framework primárně určený pro tvorbu webových aplikací založených na MVC architektuře. Pro Nette Framework je velmi důležitá otázka bezpečnostních rizik. Zaměřuje se především na eliminaci rizik typu XSS, CSRF, session hijacking, session fixation[32].

Výhody použití jsou[33]:

- Podpora technologií AJAX / AJAJ, Dependency Injection, SEO, DRY, KISS
- Událostmi řízený běh aplikace
- Dokumentace a podpora komunity
- Open-source licence New BSD nebo GNU General Public License (GPL) verze 2 či 3

### 2.5.1 Historie

Původním tvůrcem Nette Frameworku je David Grudl, který patří i v současné době mezi hlavní vývojáře[34]. V létě roku 2009 vznikla za účelem seskupení vývojářů Nette organizace Nette Foundation. Nette Foundation si především klade za cíl[35]:

- Vývoj frameworku Nette
- Pořádat školení pro zájemce o Nette Framework
- Organizovat setkání vývojařů webových aplikací (Akce s názvem "Poslední sobota")
- Podporovat tvorbu dokumentace a článku o Nette Frameworku

### 2.5.2 Životní cyklus aplikace

Aplikace je spuštěna souborem *index.php* umístěného zpravidla v kořenovém adresáři webu. V souboru *index.php* se provádí požadavek na vložení a provedení obsahu souboru *bootstrap.php*. Soubor *bootstrap.php* zajišťuje tvorbu systémového kontejneru (DI kontejneru). Nad tímto kontejnerem je volána metoda startující vlastní aplikaci. V první fázi se HTTP požadavek pomocí routeru překládá na parametry identifikující konkrétní presenter a akci v rámci tohoto presenteru, která se má vykonat. Presenter tedy přijme požadavek a zajistí pro něj odpověď. Odpověď nemusí být pouze HTML kód, ale jakékoliv data jako obrázek, XML soubor, JSON, stavový kód.

### 2.5.3 Životní cyklus presenteru

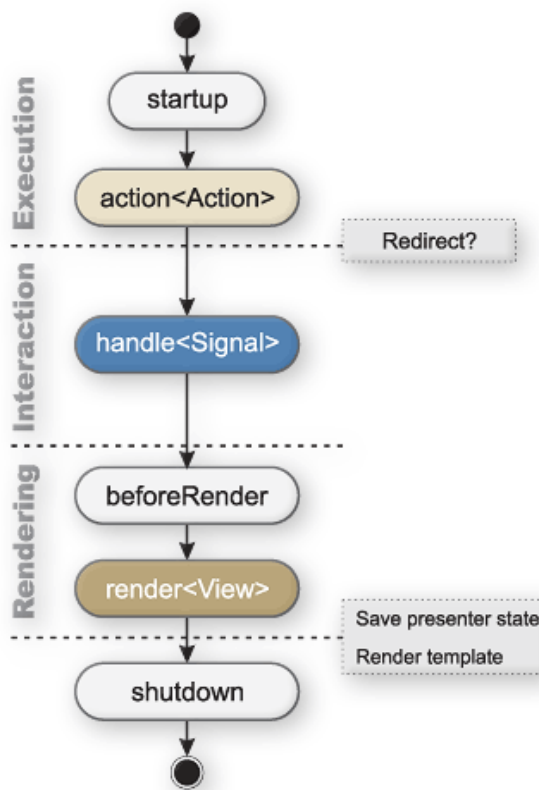
Životní cyklus presenteru je velmi důležitý z pohledu návrhnu i samotného běhu aplikace. Cyklus se skládá celkem z šesti kroků vykonávaných sekvenčně:

- startup
- action
- handle
- beforeRender
- render
- shutdown

Životní cyklus presenteru je možné kdykoliv v průběhu zpracování přerušit zavoláním speciálních metod jako jsou například *terminate*, *sendTemplate*, *sendPayload*, *sendResponse*.

Přerušení cyklu presenteru je vhodné zejména v případě výstupu obsahu libovolného souboru nebo například JSON objektu. Nedochozí pak k nechtěnému spuštění procesu vykreslení šablon.

Význam jednotlivých metod je následující:



Obr. 4. Životní cyklus presenteru[18]

- **Metoda startup** se volá ihned po vytvoření presenteru. Je společná pro všechny akce a využít se například k autorizaci uživatele.
- **Metoda action** je již různá pro každou akci. Pokud máme například akci edit, bude se příslušná metoda jmenovat actionEdit. Tato metoda je vhodná k provádění akcí, které mohou mít za následek požadavek na redirect.
- **Metoda handle** je využívána například pro zpracování signálů a používají ji například formuláře. Tuto metodu využívají také komponenty a AJAXové požadavky. Příkladem může být například signál delete pro smazání záznamu. Příslušný handle se bude jmenovat handleDelete.
- **Metoda beforeRender** je společná pro všechny akce a uplatňuje se například pro naplnění šablony daty, které jsou společné pro všechny akce.
- **Metoda render** slouží k naplnění šablony požadovanými daty. Pro akci edit se bude metoda jmenovat renderEdit.
- **Metoda shutdown** je volána těsně před ukončením presenteru. Představuje ideální místo například pro ukončení otevřeného spojení.

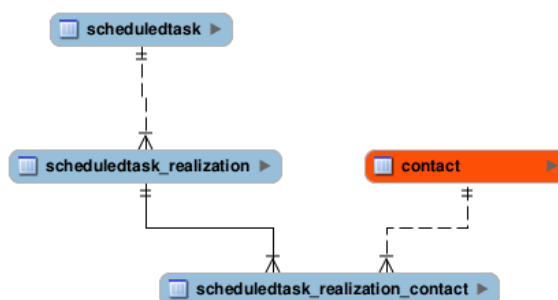
#### 2.5.4 Podpora přístupu k databázi

Framework Nette nijak nezobecňuje modelovou vrstvu architektury MVC. Nejsou pro ni vyhrazeny žádné speciální postupy ani metody. Záleží tedy na uvážení programátora, jakou cestu zvolí. Jednou z těchto cest je použití Nette Database. Nette Database umožňuje připojení k různým databázím (SŘBD) jako jsou MySQL, SQLite, MSSQL, PostgreSQL. Základem práce s Nette Database je objekt `Nette\Database\Connection` tvořící obálku nad PDO. PDO vytváří jednoduchou abstraktní vrstvu pro přístup k databázi. Abstraktní vrstva poskytuje výhody ve formě konzistentního kódování, flexibility, objektově orientovaného přístupu a výkonu. Pro různé typy databází se vždy používají stejné funkce pro zadávání příkazů a získávání dat z databáze[36].

Pro složitější výběry se využívá funkce poskytované objektem `Nette\Database\Context`, které vychází z knihovny NotORM od známého českého vývojáře Jakuba Vrány. Pro plné využití síly této knihovny jsou zapotřebí definované cizí klíče. Základní myšlenkou Nette Databáze je totiž to, že programátor se nemá o to, jakým způsobem se výběr dat provede, ale má psát případně pouze podmínky (proto například neexistuje metoda `join`). Díky vlastnosti cizích klíčů je Nette Databáze schopna určit vnitřní strukturu databáze (vazby mezi tabulkami). Tuto znalost pak využívá vnitřní parser při sestavování SQL dotazů, které jsou učené pro databázi ke zpracování. Pro práci s konkrétní tabulkou, je nutné nejprve vytvořit novou instanci objektu `Nette\Database\Table\Selection`. Nová instance se vytvoří zavoláním metody `table` na objektu `Nette\Database\Context`. Argumentem funkce `table` je název požadované tabulky. Nad objektem `Nette\Database\Table\Selection` je pak možné volat funkce pro výběr, vkládání či mazání záznamů[37][38].

Při přístupu k hodnotám sloupců z jiných tabulek svázaných libovolnou relací se využívá tzv. tečková respektive dvojtečková notace. Tečková notace se používá pro přístup k rodiči. Dvojtečková anotace se používá pro přístup k potomkovi.

#### Příklad tečkové a dvojtečkové anotace



Obr. 5. Příklad schématu databáze

Následujícím způsobem získáme výpis všech záznamů z tabulky *scheduledtask\_realization*, které mají jako jméno řešitele osobu jménem „Jan“.

```
1 $selection = $connection->table('scheduledtask_realization')
2   ->where(
3     ':scheduledtask_realization_contact.contact.firstname ?',
4     'Jan'
5   );
```

Následujícím způsobem získáme výpis všech záznamů z tabulky *scheduledtask\_realization* které mají jako název úkolu řetězec „Úkol“.

```
1 $selection = $connection->table('scheduledtask_realization')
2   ->where(
3     'scheduledtask_realization.scheduledtask.name ?',
4     'Úkol'
5   );
```

### 2.5.5 Šablonovací systém Latte

Šablony a šablonovací systém je prostředek, jak dosáhnout efektivního a znovupoužitelného psaní kódu, který je typický pro výstup webové aplikace (HTML kód). Šablonu si můžeme představit jako dokument obsahující jednak statická data, ale tak i proměnné, podmínky, cykly. Úkolem šablonovacího systému je tuto šablonu zpracovat a vrátit zpracovaný dokument. V Nette je použit šablonovací systém Latte[39].

Latte obsahuje sadu základních maker a helperů, které tvoří hlavní sílu tohoto systému[39]. Samozřejmostí je možnost použití vlastních maker a helperů, které lze do Latte jednoduše zaregistrovat.

#### Příklad makra

Příkladem makra je například *n:foreach*. Toto makro zajistí postupnou iteraci přes objekt *\$flashes*, kdy každý takto iterovaný prvek bude obsažen v HTML elementu *div*.

```
1 <div n:foreach="$flashes as $flash" class="alert">
2   {$flash}
3 </div>
```

#### Příklad helperu

Příkladem použití helperu uvádí následující kód. Je použit helper *upper*, způsobující převedení všech znaků uložených v proměnné *\$flash* na velká písmena.

```
1 <div n:foreach="$flashes as $flash" class="alert">
2   {$flash|upper}
3 </div>
```

## Ochrana proti XSS útoku

Velmi užitečnou vlastností šablonovacího systému Latte je skutečnost, že speciální řídicí znaky každé výpisované proměnné jsou automaticky nahrazeny jejich ekvivalentem v HTML. Všechny řídicí znaky HTML kódu jsou tedy překódovány do jejich bezpečné alternativy. Například řídicí znak `<` je nahrazen HTML ekvivalentem `&lt;`. Díky tomuto chování, je zabezpečena ochrana proti možnému XSS útoku. Nahrazování lze u konkrétní proměnné potlačit helperem `noescape`[39].

### 2.5.6 Konfigurace

Jak již bylo zmíněno v kapitole o životním cyklu presenteru, při startu aplikace se nejprve vytváří systémový DI kontejner. Tento kontejner je tvořen na základě konfiguračního souboru pouze jednou, respektive pak při každé změně konfigurace.

Konfigurační soubor tedy hraje ve frameworku velmi důležitou roli.

Konfigurace se v Nette Frameworku zapisuje do souboru umístěného v adresáři `config` adresářové struktury aplikace. Název souboru bývá `config.neon`. Cesta resp. název souboru se dá změnit v souboru `bootstrap.php`.

Konfigurační soubor obsahuje různé tematicky oddělené sekce. V databázové sekci se například uvádí údaje potřebné pro připojení k databázovému serveru. V sekci `services` se uvádí výčet tříd či komponent, pro které využíváme vlastností DI kontejneru.

## 2.6 Technologie pro tvorbu Front-end části

Výstup webové aplikace bývá převážně tvořen HTML kódem a je určen k interpretaci Internetovým prohlížečem. Internetový prohlížeč tedy tento kód zpracuje do grafické podoby a výsledek zobrazí uživateli. Požadavkem na webové aplikace tedy je, aby nabídla intuitivní a přátelské uživatelské rozhraní.

V dnešních dobách rozvoje chytrých mobilních zařízení, lze k webovým aplikacím přistupovat i přes tato zařízení. Výstup webové aplikace by se měl dokázat správně interpretovat i na těchto zařízeních, které mají pravidla menší dostupné rozlišení displaye, než display běžného PC.

Pro potřeby moderních webových aplikací jsou v současné době k dispozici různé CSS a JS frameworky, díky kterým je možno vytvářet v krátkém čase moderní aplikace. Výhodou frameworků je, že dokážou sjednotit rozdílnou podobu aplikace v různých typech prohlížečů. Dosáhne se tak stejného vzhledu na všech prohlížečích včetně rozdílných platforem.

Přestaviteli moderních frameworků pro tvorbu webu jsou pro HTML a CSS framework Twitter Bootstrap a pro JS je to framework JQuery a JQueryUI.

## Twitter Bootstrap

Twitter Bootstrap je moderní framework používaný pro tvorbu front-end části aplikace. Byl vytvořen vývojáři Twitteru jako požadavek na tvorbu univerzálního rozhraní, které by bylo možné použít opakovaně pro různé typy aplikací.

V roce 2011 byl Twitter Bootstrap uvolněn jako open-source. V dnešní době patří mezi nejúspěšnější frameworky pro tvorbu front-end částí webových aplikací[40].

Výhody použití tohoto frameworku:

- Stejné zobrazení na všech prohlížečích bez ohledu na platformu
- Rychlost nasazení
- Podpora responzivního web-designu
- Snadná udržitelnost kódu

## Knihovny jQuery a jQueryUI

Knihovna jQuery a framework jQueryUI představují oporu při práci s dynamickými prvky na straně klienta. Díky těmto knihovnám lze vytvořit velice sofistikované webové aplikace. Rozhraní těchto aplikací je pak srovnatelné s běžnými desktopovými aplikacemi. Obě knihovny jsou navíc poskytovány jako open-source. Je možné je tedy bezplatně použít[41].

### jQuery

jQuery je open-source knihovna používaná pro tvorbu skriptů určených pro zpracování JavaScriptem. Vznikla v roce 2005 s cílem, vytvoření jednotného rozhraní pro práci s JavaScriptem na různých typech prohlížečů[42].

V dnešní době existuje velké množství pluginů využívající tuto knihovnu. K dispozici jsou například pluginy pro práci s kalendářem (datepickers), s výběrem barvy (colorpickers), validační plugin pro kontrolu údajů zadaných do formuláře.

### jQueryUI

jQueryUI je rozsáhlý framework, určený pro JavaScript. Obsahuje velké množství pluginů, které usnadňují tvorbu aplikace a umožňují implementovat funkce, přispívající ke zvýšení komfortu uživatelské rozhraní aplikace.

Framework se skládá ze čtyřech částí. Jedná se o[43]:

- Interactions - interakce mezi uživatelem a uživatelským rozhraním aplikace

- Widgets - UI komponenty
- Effects - Efekty či animace
- Utilities - Nástroje pro pozicování UI komponent

## 2.7 Nette Tester

Testování aplikace a její částí představuje v současné době neopomenutelnou součást vývoje aplikace. Dobře navržená aplikace se vždy vyznačuje tím, že lze velmi dobře otestovat[44]. Testováním rozumíme tvorbu malých skriptů (unit testů), které mají za cíl konstatovat, zda výstup zvoleného objektu vyhovuje zadanému kritériu. Zvoleným objektem může být například funkce, metoda třídy či výraz. Zadané kritérium může být například číslo, řetězec či vyjímka.

Správně navržená aplikace je tedy složena z malých bloků (funkcí), kdy každý tento blok vykonává libovolnou operaci. Platí však, že pro tento malý blok lze napsat vlastní test. Při návrhu aplikace je tedy nutné zohlednit výše uvedené požadavky a tvořit nejlépe funkce tak, aby při popisu jejich funkčnosti nebylo nutné použít spojku „a“.

Nette Tester je framework pro tvorbu unit testů, který lze použít k testování jakýkoliv aplikací. Nette Framework používá Nette Tester k testování celého vlastního frameworku. Práce s Nette Testerem je velmi jednoduchá. K vyhodnocení testů slouží celkem devatenáct metod třídy *Assert*. Metody mají buď jeden parametr jako například *Assert::true*, nebo dva parametry jako například metoda *Assert::equal*. Samozřejmostí je i podpora pro testování zachycení vyjímek[45].

### Příklad

V příkladu budeme chtít otestovat dvě metody třídy *Foo*. Od první funkce očekáváme, že vrátí vždy desetinásobek vstupu. Od druhé funkce budeme očekávat návratovou hodnotu *true*.

Nejprve definujeme třídu, kterou použijeme pro testování.

```
1 class Foo {
2     /**
3     * Vraci desetinásobek vstupu
4     * @param int $input Vstupni hodnota
5     * @return int
6     */
7     function calculate($input) {
8         if ($input < 0) {
9             throw new Exception("Input value cannot be lower than zero.
10            $input was given.");
11        }
12        $return = 10 * $input;
13        return $return;
14    }
```

```

15     }
16
17     /**
18      * Vrací vždy true
19      * @return boolean
20      */
21     function isAny() {
22         return true;
23     }
24 }

```

Nyní napíšeme jednoduchý test, který porovná výsledek funkce *calculate* s očekávanou hodnotou. Pokud se hodnoty shodují, je test úspěšný.

```

1 use Tester\Assert;
2
3 $foo = new Foo;
4
5 Assert::equal(50, $foo->calculate(5));

```

Pokud bude na vstupu funkce *calculate* záporná hodnota, očekáváme, že funkce vyvolá výjimku. Tu zachytíme v testu.

```

1 use Tester\Assert;
2
3 $foo = new Foo;
4
5 Assert::exception(function() use ($foo) {
6     $foo->calculate(-5);
7 }, 'Exception');

```

Následující test očekává, že návratová hodnota funkce *isAny* bude *true*.

```

1 use Tester\Assert;
2
3 $foo = new Foo;
4
5 Assert::true($foo->isAny());

```

Jednotlivé testy testující určitou specifickou část je vhodné sloučit do jednoho souboru. Vyvoříme-li například soubor *Foo.create().phpt*, lze podle názvu očekávat, že soubor obsahuje testy pro testování vytvoření objektu *Foo*.

Pokud test proběhne úspěšně, zobrazí se hlášení ve tvaru:

```

|-----/-----) (-----/-----) |-----)
|_| \_____/____) |_| \_____/____) |_| \_____/____) v1.0.0
Note: No php.ini is used.
PHP 5.5.11 | 'php-cgi' | 1 threads
OK (1 tests, 0.0 seconds)

```

V případě selhání testu se zobrazí hlášení ve tvaru:

```

|-----/-----) (-----/-----) |-----)
|_| \_____/____) |_| \_____/____) |_| \_____/____) v1.0.0
Note: No php.ini is used.
PHP 5.5.11 | 'php-cgi' | 1 threads

```

```
F
-- FAILED: budovy/tests/Foo.create().phpt
  Failed: 50 should be equal to 5
    in Tester/Framework/Assert.php(370)
    in Tester/Framework/Assert.php(76) Tester\Assert::fail()
    in budovy/tests/Foo.create().phpt(32) Tester\Assert::equal()
FAILURES! (1 tests, 1 failures, 0.1 seconds)
```

## **II. PROJEKTOVÁ ČÁST**

### 3 POŽADAVKY NA APLIKACI

Aplikace je určena pro správu budov, kde si klade za cíl podpořit FM služby, které jsou potřebné pro optimální chod budovy a minimalizaci nákladů při její správě.

Postupuji tak, že celou problematiku služeb nejprve rozdělím na jednotlivé významově dílčí části. Analýzou každé části pak dostanu jasný přehled požadavků, které aplikace v dané oblasti musí splňovat. Za základě takto získaných poznatků, budu schopen navrhnout základní ER model databáze.

#### 3.1 Nastavení prostředí

V této části práce jsou postupně nalezeny a identifikovány všechny základní prvky tvořící jádro aplikace. Základní požadavek stanovuje schopnost popsat konkrétní objekt a jeho umístění v rámci spravované budovy či například celého areálu. Tento objekt tvoří FM produkt (sledovaný objekt), pro který budeme chtít evidovat veškeré činnosti, které při jeho správě nastanou. Sledovaný objekt může představovat například celá místnost či jeden konkrétní objekt jako je například elektrický rozvaděč.

Umístění sledovaného objektu v rámci spravovaného areálu je vhodné popsat samostatným objektem. Objekt bude reprezentovat například konkrétní místnost. Tímto postupem budeme schopni rozdělit spravované objekty na dílčí části ve kterých jsou umístěny příslušné objekty.

Další základní činností aplikace bude schopnost popsat úkon, který se realizoval, nebo se má realizovat s konkrétním objektem (sledovaný objekt). Definice úkonu jako elementární jednotky nám pomůže například při sestavování statistik. Realizaci úkolů zajišťují jak externí firmy či interní pracovníci. Aplikace musí být schopna rozeznat tyto jednotlivé osoby a musí k těmto osobám vést patřičné kontaktní údaje.

Může nastat situace, kdy konkrétní firma či pracovník zajišťuje určité práce pro určité zařízení jako výhradní dodavatel. Tuto situaci by bylo vhodné v systému uchovat a umožnit při vytváření úkolu tuto konkrétní osobu přiřadit ve výchozím stavu jako kontaktní osobu.

#### 3.2 Správa dokumentů

Správou dokumentů rozumíme všechny činnosti, které jsou zapotřebí k zajištění práce s libovolnými soubory. Jednotlivé soubory mohou představovat například dokumentaci k určitému zařízení či například fakturu za dodané práce. Díky schopnosti spravovat tyto dokumenty bude možné v aplikaci evidovat dokumentaci potřebnou k vykonávání obslužných prací při správě budovy.

Aplikace tedy musí umožnit nahrávat libovolné soubory, které pak lze vhodně seskupovat do skupin dle jejich významu. Skupiny bude možné spravovat uživatelem systému.

Výmazem skupiny nesmí zcela logicky dojít k odstranění dokumentů patřící do této skupiny.

Jednotlivé dokumenty pak bude možné přiřadit vytvořeným úkolům či jiným objektům, u kterých budeme potřebovat vazbu k dokumentu.

### 3.3 Plánování úkolů

Plánování úkolů tvoří stěžejní část celé aplikace. Při návrhu plánovacího systému je důležité důrazně dbát na jeho variabilitu a pružnost. Jednotlivé úkoly lze rozdělit do dvou skupin. První skupinu tvoří úlohy periodicky opakované a druhou skupinu tvoří úlohy spojené s konkrétním datem.

Jako periodicky se opakující úlohu označíme ty úlohy, které se vždy po daný časový úsek s určitou periodou opakují. Pro tyto úlohy bude charakteristická perioda opakování a časové rozmezí jejich platnosti. Jako časové rozmezí budeme uvažovat jednotlivé měsíce v roce. Jako periodu označíme konkrétní dny, které lze popsat jistým způsobem v rámci jednoho měsíce. Budeme tedy uvažovat o periodách jako jsou každý den, pracovní den, víkendový den, první pracovní den v měsíci, první pracovní den v týdnu.

Jako úlohu pevně spojenou s konkrétním datem označíme tu úlohu, kterou je nutné vykonat v určitý konkrétní den a která se nikdy neopakuje. Pro tuto úlohu bude charakteristické datum začátku úkolu. Plánovanou úlohu bude před provedením nutné nejprve schválit osobou s patřičným uživatelským oprávněním.

Úkolům je vhodné mít možnost přiřadit požadovanou prioritu úkolu. Může totiž nastat situace, kdy je nutné řešit dva úkoly ve stejném čase. Bez rozlišení priority by mohl být řešen jako první úkol s menší důležitostí.

Jednotlivé úkoly by měly mít možnost přiřazení neomezeného počtu dokumentů. Tyto dokumenty budou představovat například výkresová dokumentace k zařízení či jiné podklady nutné pro realizaci úkolu.

Důležitou částí správy úkolů bude schopnost vytvořit záznam o provedení úkolu. Záznam bude obsahovat datum provedení a částku, která byla v souvislosti s realizací úkolu naúčtována. Záznamu bude také možné přiřadit kontakt na osobu, která práce vykonala. Může totiž nastat situace, kdy práce vykonala jiná osoba než ta, která byla původně k úkolu přidělena. K jednotlivým záznamům o provedení lze také přiřadit dokument.

### 3.4 Záznam o havárii

V reálném světě nastávají situace, které nejsou dopředu plánovány a které je nutné evidovat pro pozdější vyhodnocení. Tyto stavy označíme jako havárie. Záznamy o havárii mohou být jasným vodítkem například při vyhodnocování poruchovosti konkrétního zařízení. Pokud z těchto záznamů vyplyne zvýšené množství servisních zásahů, bude

vhodné přistoupit k výměně zařízení. Sníží se tak riziko selhání a poklesnou náklady na toto zařízení. Záznamům o havárii by měla být možnost přiřazení neomezeného počtu dokumentů. Tyto dokumenty bude představovat například písemný zápis o havárii či faktura za provedené práce.

### 3.5 Uživatelé systému a jejich uživatelské role

Uživatelské účty a uživatelské role představují nástroj, jak zabránit neoprávněné manipulaci s aplikací. U aplikací dostupných v rámci sítě Internet hraje způsob zabezpečení obzvláště důležitou roli.

Autentifikace uživatelů bude probíhat na základě seznamu uživatelů vedených v systému. Každý uživatel bude disponovat svým uživatelským jménem a heslem. Uživatelské heslo však nesmí být nikde v systému uloženo v otevřené podobě. Tento požadavek je splněn použitím jednosměrné hash funkce *sha1*.

Uživatel systému bude disponovat jednou konkrétní uživatelskou rolí. Uživatelské role představují nástroj pro autorizaci všech prováděných operací uživatelem. Každá uživatelské role pak bude obsahovat seznam povolených operací, které může uživatel v systému vykonat.

### 3.6 Upozornění na začátek úkolu

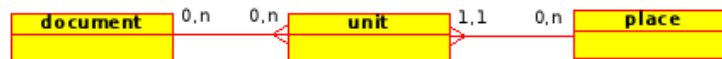
Důležitou vlastností systému musí být možnost k úkolu stanovit počet dnů před začátkem úkolu, kdy má být e-mailem upozorněna obsluha aplikace včetně kontaktní osoby přiřazená k tomuto úkolu na nutnost zahájení prací.

## 4 NÁVRH ZÁKLADNÍCH ENTIT

V předchozí části jsme definovali požadavky, které musí aplikace splňovat. V této části těchto znalostí využijeme k tvorbě ER schématu. Určíme základní entity a definujeme vazby mezi těmito entitami.

### 4.1 FM produkt a jeho umístění

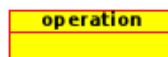
FM produkt (objekt zájmu) a jeho umístění v rámci prostoru je možné popsat dvěma entitami *unit* a *place*. Entita *place* představuje umístění a entita *unit* představuje konkrétní FM produkt. Vztah mezi entitami *unit* a *place* označíme jako  $(1, 1) : (0, n)$ . Entita *unit* bude mít dále vztah  $(0, n) : (0, n)$  s entitou *document*, která představuje dokument evidovaný v systému. Entita *document* je popsána v samostatné kapitole.



Obr. 6. Entity pro objekt zájmu a umístění

### 4.2 Prováděná činnost

Pro popis prováděné činnosti vytvoříme entitu *operation*. Tato entita bude popisovat jednotlivé prováděné úkony, které lze realizovat nad jednotlivými FM produkty.



Obr. 7. Entita pro prováděnou činnost

### 4.3 Kontakt

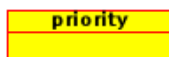
Entita *contact* představuje jednu konkrétní kontaktní osobu. Popisuje osobní a kontaktní údaje pracovníka či firmy, kteří zajišťují provedení přidělených prací.



Obr. 8. Entita pro kontakt

### 4.4 Priorita úkolu

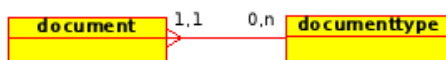
Prioritu úkolu popíšeme entitou *priority*. Tato entita bude představovat prioritu, s jakou mají být jednotlivé úkoly zpracovávány.



Obr. 9. Entita pro prioritu

#### 4.5 Dokumenty

Dokumenty evidované v systému bude představovat entita *document*. Dokumenty jak už bylo výše uvedeno, představují například záruční listy, uživatelské manuály, faktury za provedené práce. Pro rozlišení různých typů dokumentů, lze vytvořit samostatné skupiny, do níž bude každý jednotlivý dokument zařazen. Jednu skupinu můžou tvořit například smlouvy, druhou můžou být například faktury. Skupina dokumentů bude v systému popsána entitou *documenttype*. Vztah mezi entitami *document* a *documenttype* označíme jako  $(1, 1) : (0, n)$ .

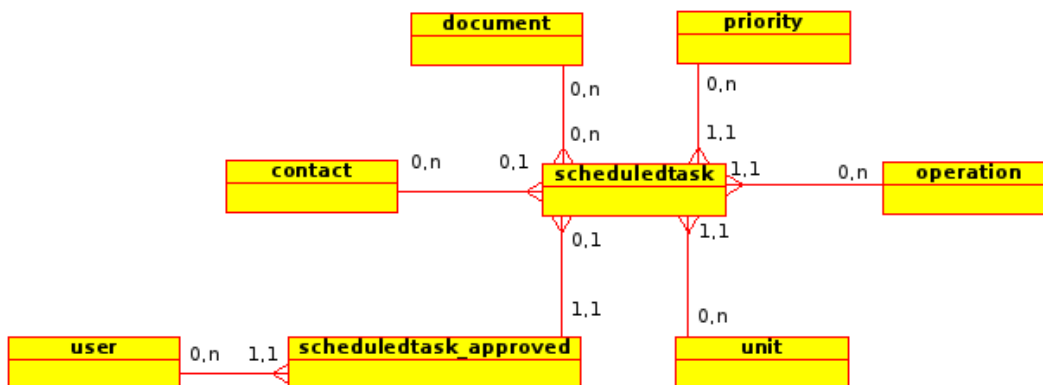


Obr. 10. Entity pro dokument a jeho typ

#### 4.6 Plánovaná úloha

Plánovanou úlohu bude popisovat entita *scheduledtask*. Plánovaná úloha představuje typ úlohy, který je svázán s konkrétním datem. Entita *scheduledtask* bude ve vztahu  $(1, 1) : (0, n)$  s entitami *unit*, *operation* a *priority*. Vztah mezi entitami *scheduledtask* a *contact* bude  $(0, 1) : (0, n)$ . Vztah mezi entitami *scheduledtask* a *document* bude  $(0, n) : (0, n)$ .

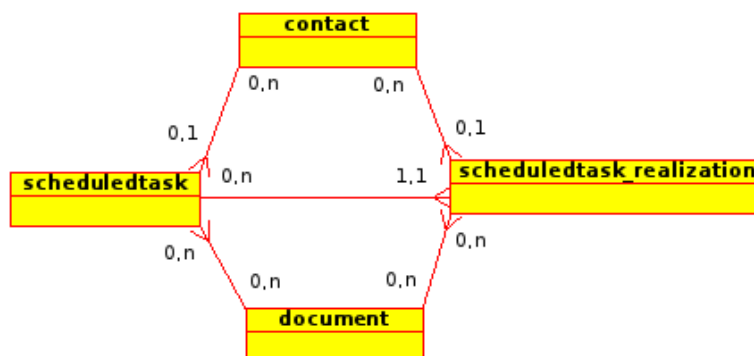
Entita *scheduledtask\_approved* představuje záznam o schválení úlohy odpovědnou osobou. Tato entita je ve vztahu  $(1, 1) : (0, n)$  s entitou *user* a ve vztahu  $(1, 1) : (0, 1)$  s entitou *scheduledtask*.



Obr. 11. Entita pro plánovanou úlohu

#### 4.7 Záznam o provedení plánované úlohy

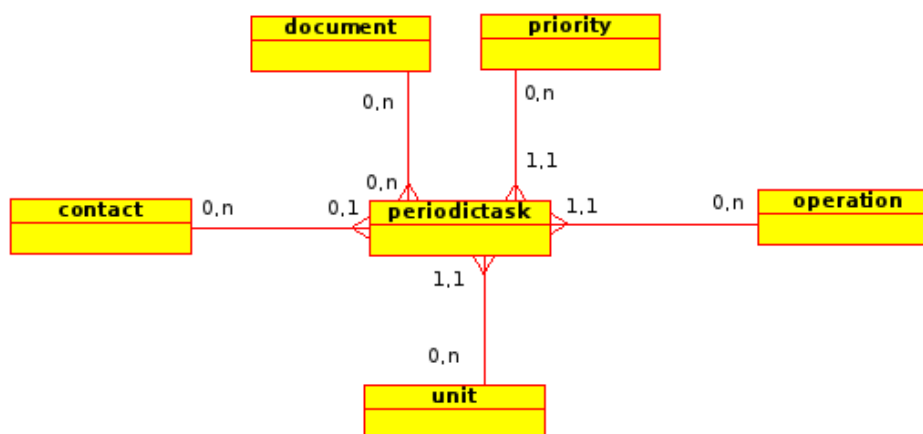
Záznam o provedení plánované úlohy představuje entita *scheduledtask\_realization*. Vztah mezi entitou *scheduledtask\_realization* a entitami *scheduledtask* a *contact* bude  $(0, 1) : (0, n)$ . Vztah mezi entitami *scheduledtask\_realization* a *document* bude  $(0, n) : (0, n)$ .



Obr. 12. Entita pro záznam o provedení plánované úlohy

#### 4.8 Periodická úloha

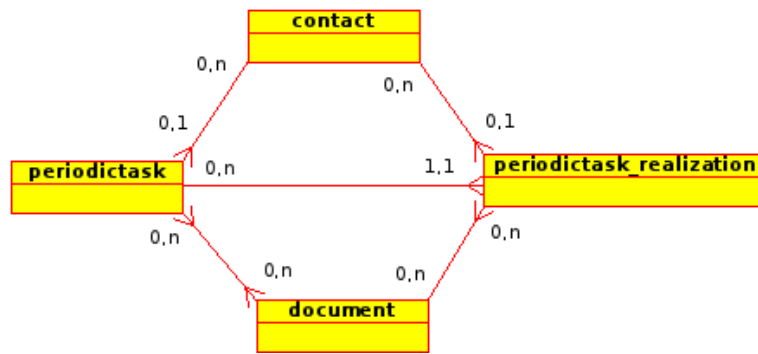
Periodickou úlohu popisuje entita *periodictask*. Plánovaná úloha představuje typ úlohy, který je zadána periodou opakování. Entita *periodictask* bude ve vztahu  $(1, 1) : (0, n)$  s entitami *unit*, *operation* a *priority*. Vztah mezi entitami *periodictask* a *contact* bude  $(0, 1) : (0, n)$ . Vztah mezi entitami *periodictask* a *document* bude  $(0, n) : (0, n)$ .



Obr. 13. Entita pro periodickou úlohu

#### 4.9 Záznam o provedení periodické úlohy

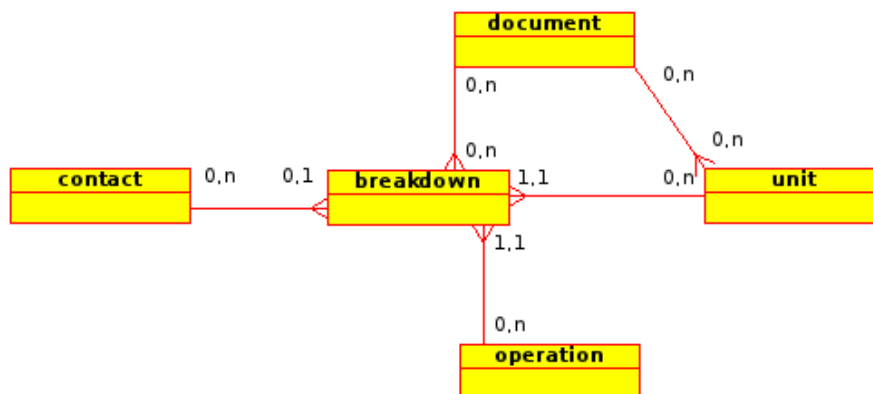
Záznam o provedení periodické úlohy představuje entita *periodictask\_realization*. Vztah mezi entitou *periodictask\_realization* a entitami *periodictask* a *contact* bude  $(0, 1) : (0, n)$ . Vztah mezi entitami *periodictask\_realization* a *document* bude  $(0, n) : (0, n)$ .



Obr. 14. Entita pro záznam o provedení periodické úlohy

#### 4.10 Záznam o havárii

Pro evidenci havárií bude sloužit entita *breakdown*. Havárií rozumíme záznam o provedení činnosti, která nebyla dopředu nijak plánována. Entita *breakdown* bude ve vztahu (1, 1) : (0, n) s entitami *unit* a *operation*. Vztah mezi entitou *breakdown* a *contact* bude (0, 1) : (0, n). Vztah mezi entitou *breakdown* a *document* bude (0, n) : (0, n).



Obr. 15. Entita pro záznam o havárii

#### 4.11 Uživatel a uživatelské role

Uživatele lze popsat entitou *user* která je ve vztahu (1, 1) : (0, n) s entitou *role*. Entita *role* představuje uživatelské oprávnění.



Obr. 16. Entity pro uživatele a uživatelskou roli

## 5 EER MODEL DATABÁZE

Při tvorbě EER modelu vycházím z poznatků, získaných při tvorbě ER modelu. EER model oproti ER modelu má již relace mezi entitami fyzicky realizovány za pomoci cizích klíčů a příslušných vazebních tabulek. Jednotlivé entity jsou také doplněny o své atributy a primární klíče. Schémata jednotlivých logických celků jsou uvedena na obrázcích. Celkové schéma databáze je uvedeno v příloze.

Primární klíč většiny entit tvoří atribut *id*. Atribut *id* je bezznaménkové celé číslo a disponuje příznakem *auto\_increment* pro automatické navyšování identifikátoru.

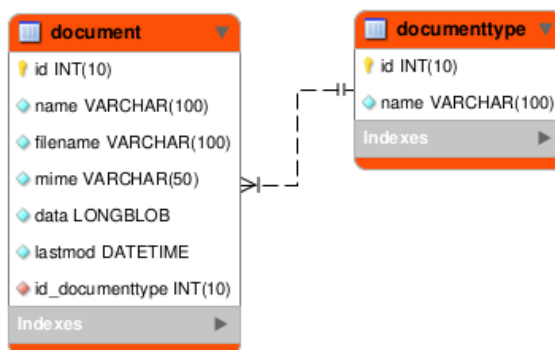
Primární klíč entity *role* tvoří kód *role*. Tento kód je pak v aplikaci použit pro kontrolu uživatelských oprávnění.

Relace (0, n) : (0, n) mezi entitami jsou realizovány za pomoci vazební tabulky obsahující dva sloupečky. První atribut je cizím klíčem do cílové tabulky, druhý atribut je cizím klíčem do zdrojové tabulky. Oba atributy tvoří složený primární klíč. Například pro relaci mezi entitami *unit* a *document* jsem tedy vytvořil tabulku *unit\_document* která má jako první atribut (*id\_unit*) cizí klíč do tabulky *unit*. Druhý atribut (*id\_document*) tvoří cizí klíč do tabulky *document*.

Relace (0, 1) : (0, n) jsou řešeny obdobně jako relace (0, n) : (0, n) pouze s tím rozdílem, že primární klíč vazební tabulky tvoří pouze atribut s cizím klíčem zdrojové entity.

Relace (1, 1) : (0, n) jsou realizovány tak, že zdrojová tabulka přímo obsahuje cizí klíč cílové entity.

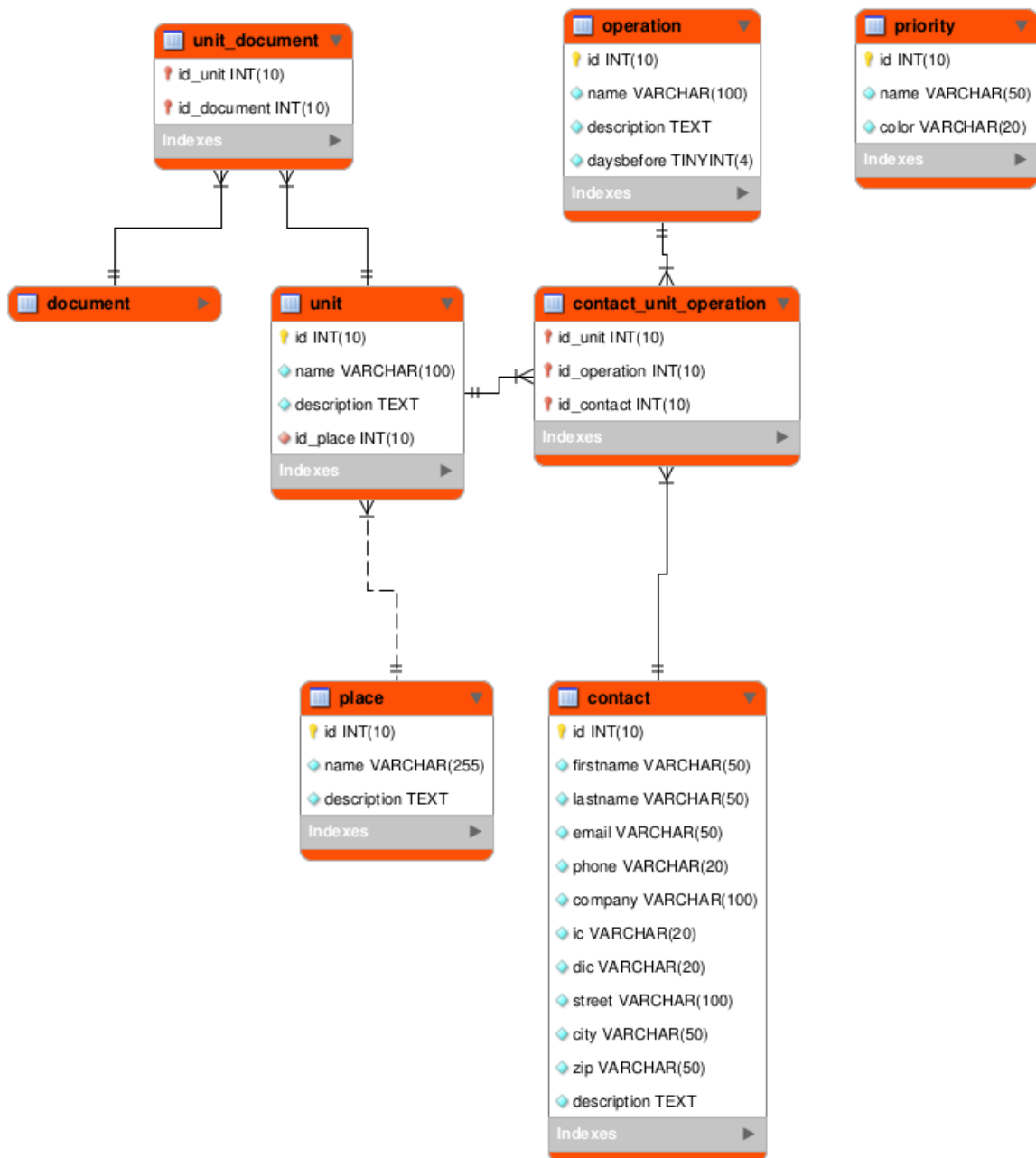
Evidované dokumenty jsou ukládány přímo do databáze, konkrétně do tabulky *document*. Tabulka *document* pro tyto účely obsahuje tři atributy. Atribut *filename* představuje název souboru, atribut *data* obsahuje zdrojová data souboru a atribut *mime* udává MIME typ dokumentu. MIME se používá pro rozlišení typu souboru. Například MIME typ *image/jpeg* odpovídá obrázku ve formátu JPEG, MIME typ *application/pdf* odpovídá dokumentu ve formátu PDF.



Obr. 17. EER schéma dokumentů

Na schématu základních entit lze nalézt všechny entity tvořící jádro aplikace. Ze sché-

matu je jasně zřetelná vazba mezi jednotkou *unit* (FM produkt) a jeho umístěním ve spravovaném prostoru *place*.

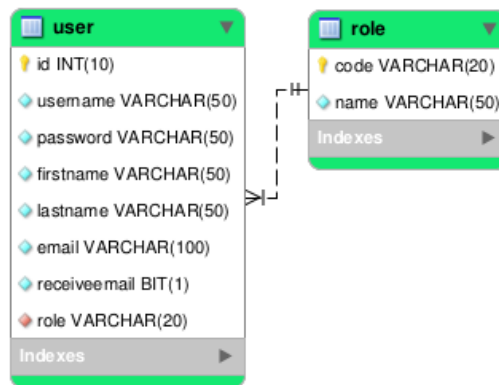


Obr. 18. EER schéma základních entit

Na schématu uživatelů je viditelná vazba mezi uživatelem a jeho uživatelskou rolí. Uživatel disponuje atributem *receiveemail*, který udává, zda daný uživatel má obdržet e-mailové upozornění na blížící se termín naplánovaných prací.

Ze schématu plánovaných úloh jsou zřetelné dva atributy *start* a *end* typu *DATE*. Tyto atributy slouží k naplánování prací na konkrétní období zadané rozpětím datumů.

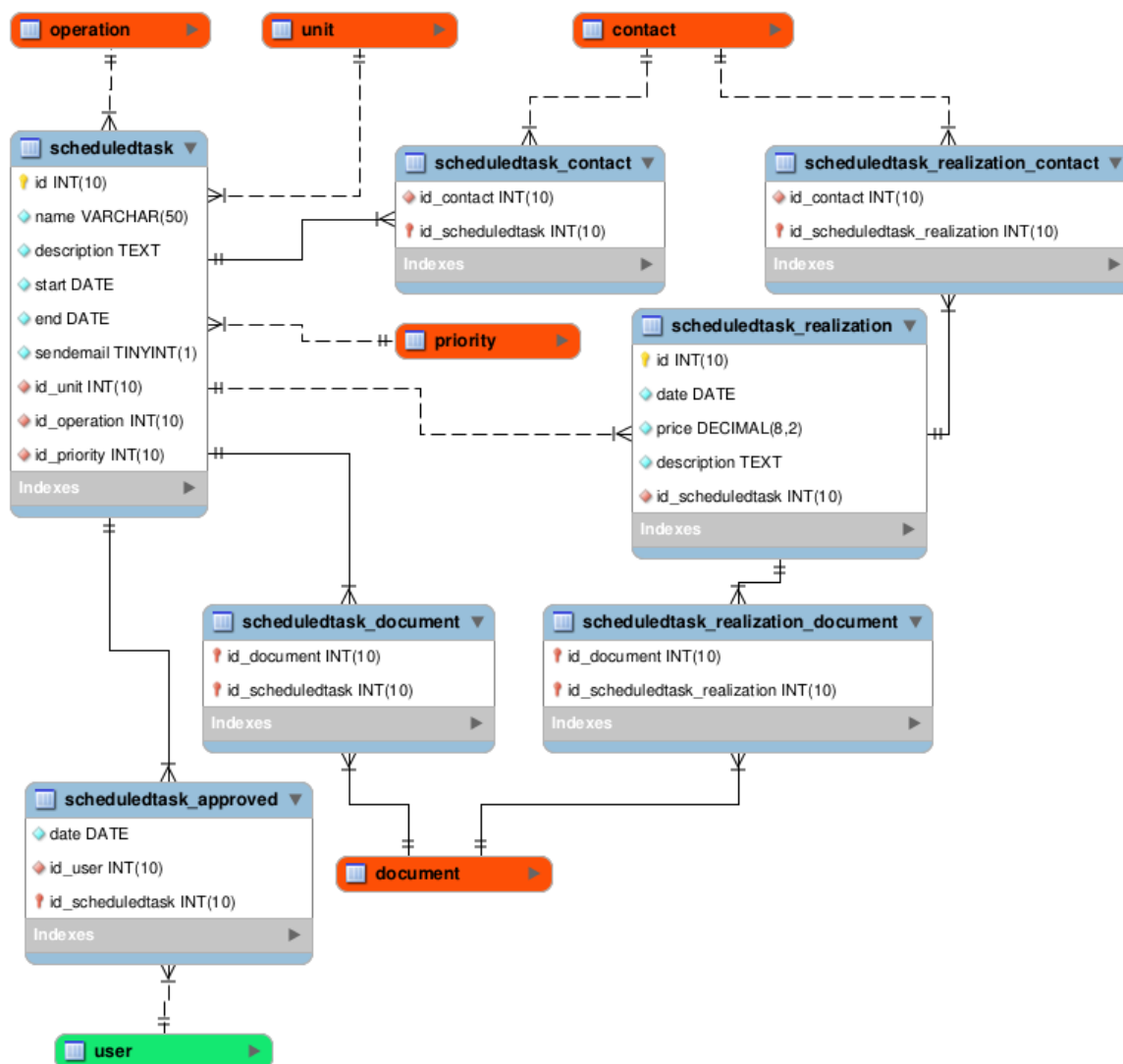
Pro periodické úlohy je charakteristická perioda opakování. Ta je patrná ze schématu



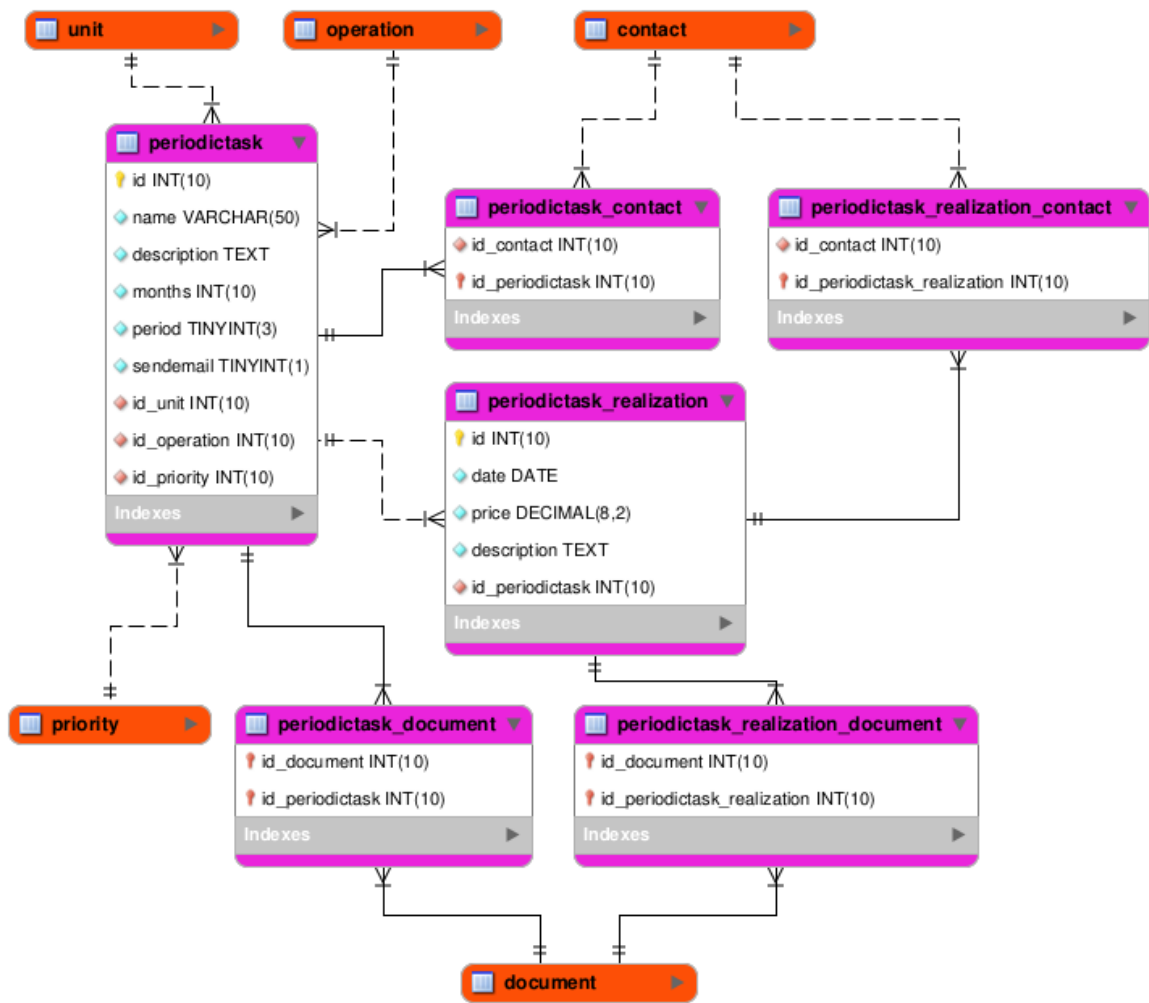
Obr. 19. EER schéma uživatelů

periodických úloh. Jedná se o atributy *period* a *months*, které jsou celočíselné a bezznaménkové. Atribut *period* udává periodu opakování, kdy každému číslu je přiřazena různá perioda opakování. Hodnota 0 představuje periodu každý den, 1 představuje každý pracovní den atd. Atribut *months* je využit pro uložení měsíců, ve kterých je zvolený úkol aktivní. Způsob uložení hodnot odpovídá bitové masce. Každému měsíci je vyhrazen jeden bit, kdy lednu odpovídá LSB a pro prosinec je vyhrazen bit na pozici 12.

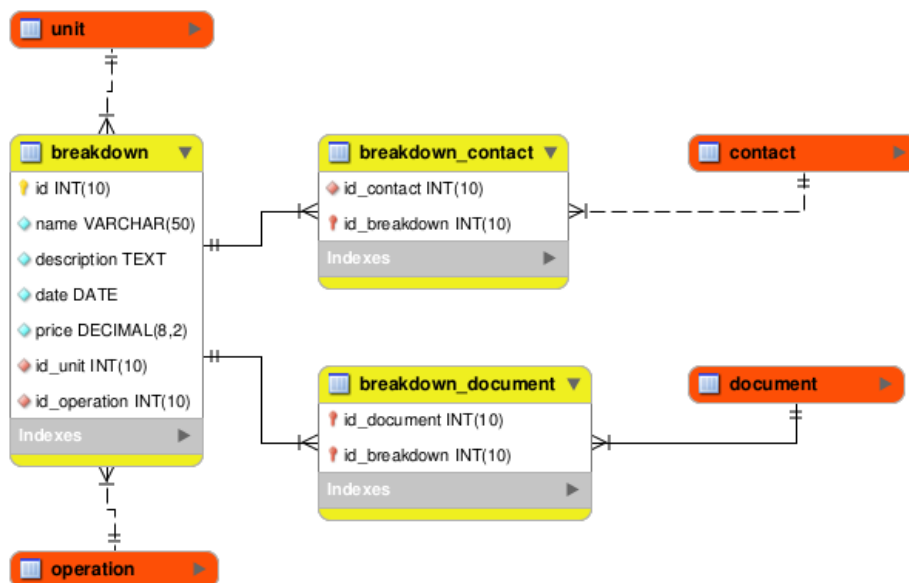
Ze schéma havárií je patrné, že entita neobsahuje atribut pro periodu či atributy pro zadání rozpětí datumu potřebných pro plánování úkolů. Obsahuje pouze atribut *date* typu *DATE*, který slouží k označení dne, kdy k havárii došlo. Havárie neslouží k plánování, ale pouze ke zaznamenání činností, které již nastaly.



Obr. 20. EER schéma plánované úlohy



Obr. 21. EER schéma periodické úlohy



Obr. 22. EER schéma havárií

## 6 NÁVRH UŽIVATELSKÉHO ROZHRAŇÍ APLIKACE

Návrh uživatelského rozhraní představuje při tvorbě aplikace velmi obtížný krok. Každý uživatel preferuje jiný rozsah informací obsažených na jedné obrazovce aplikace a také preferuje jiný způsob ovládání. Někdo ocení malé jednoduché formuláře, jiný zase formuláře komplexní, složené z více sekcí. Nalezení kompromisu nebývá ve většině případů snadné.

Uživatelské rozhraní aplikace musí být proto navrženo s ohledem na intuitivnost a přehlednost pro průměrně zdatného uživatele. Musí disponovat podporou všech moderních internetových prohlížečů. Rozložení klíčových prvků v různých částech aplikace by mělo být kvůli přehlednosti vždy co nejpodobnější. Vhodně musí být zvolena i barevná kombinace prvků aplikace.

Při návrhu rozložení uživatelského rozhraní se inspiroji v nespočetném množství šablon dostupných na portále TemplateMonster [46]. Volím klasické rozložení složené z hlavičky a levé části tvořené přehledným menu. Zbytek volné plochy tvoří prostor pro vlastní obsah. Při tvorbě uživatelského rozhraní využívám také Twitter Bootstrap šablony, které definují vzhled jednotlivých prvků na stránce. Jako podporu při tvorbě dynamických prvků a zpracování událostí na straně klienta, využívám frameworku jQueryUI a knihovnu jQuery.

Druhým krokem návrhu, je volba způsobu ovládání aplikace. Vzhledem k tomu, že aplikace bude určena pro náročné prostředí, je nutné zvolit takový způsob ovládání, aby uživatel měl vždy jasný přehled o tom, kde se aktuálně nachází a jakou činnost může provést. Každá dílčí činnost bude tedy rozdělena kvůli přehlednosti do samostatného formuláře či okna. Toto chování bude co nejvíce sjednoceno tak, aby vždy konkrétní činnost byla v různých částech aplikace prováděná stejně.

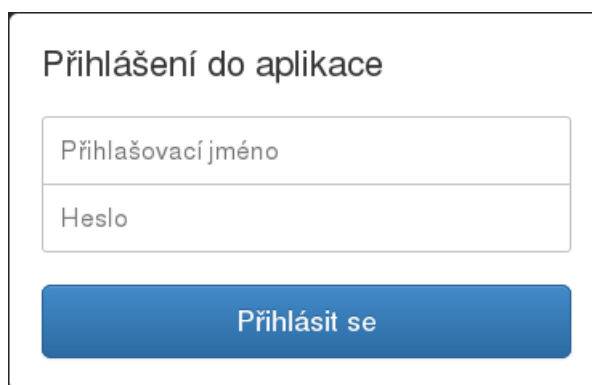
Hlavní menu aplikace bude tvořeno skupinami, ve kterých budou seskupeny odkazy na části aplikace. Každá skupina bude obsahovat odkazy na významově stejné části aplikace. Například skupina „Plánovaná úloha“ bude seskupovat odkazy na správu plánovaných úkolů, autorizaci a záznamy o realizaci.

Výpis položek (například úkolů) bude vždy realizován formou datagridu. Tento datagrid umožňuje ucelený pohled včetně možnosti rychlého filtrování. V každém datagridu bude poslední sloupec tvořit seznam akcí, které lze s příslušným záznamem vykonat. Množství zobrazených akcí závisí na konkrétních uživatelských právech. Může se tedy stát, že seznam akcí bude prázdný. Tento stav nastane, pokud uživatel nebude disponovat vhodným uživatelským oprávněním.

Pro splnění stanovených cílů, je aplikace tvořena sadou komponent, které mají vždy stejný charakter ovládání a v co největší míře i stejný vzhled. Význam jednotlivých komponent je vhodné rozebrat podrobně.

## Přihlašovací okno

Po připojení k aplikaci (také po vypršení platnosti přihlášení z důvodu uzavření okna aplikace či dlouhé neaktivity), se zobrazí přihlašovací okno. Toto okno je tvořeno přihlašovacím formulářem s prvky pro zadání přihlašovacích údajů. Po vyplnění správných přihlašovacích údajů, dochází k přesměrování na hlavní stránku aplikace.



The image shows a login form with the title "Přihlášení do aplikace". It features two input fields: "Přihlašovací jméno" and "Heslo". Below the fields is a blue button with the text "Přihlásit se".

Obr. 23. Přihlášení do aplikace

## Hlavní strana

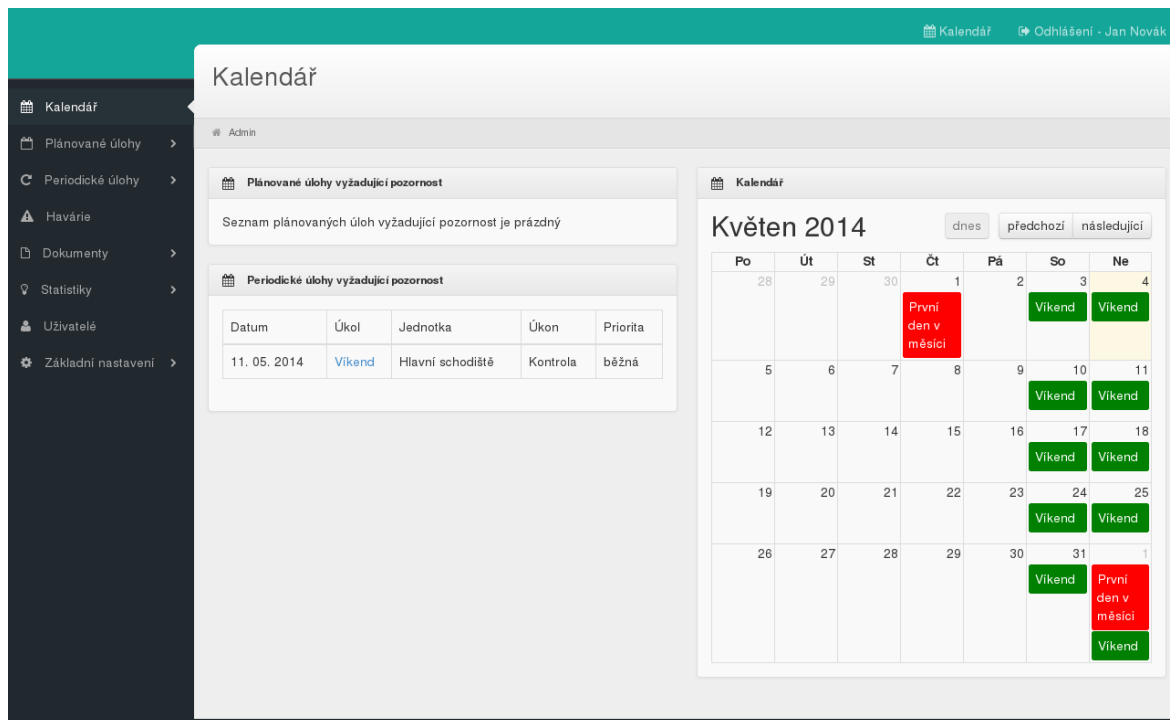
Hlavní strana aplikace je složena ze dvou částí. Levou část hlavní plochy tvoří seznam úkolů a pravá část je vyhrazena pro plánovací kalendář.

V seznamu úkolů se zobrazují periodické a plánované úlohy pro které platí, že termín provedení této úlohy odpovídá dnešnímu dni plus počet dnů nastavený u příslušného úkonu (editační pole „Upozornit před“).

Kalendář obsahuje přehled úkolů ve zvoleném aktuálním měsíci. Ve výchozím stavu je zobrazen vždy aktuální měsíc. Jiný měsíc lze samozřejmě zvolit po stisknutí tlačítek „předchozí“ či „následující“. Po stisknutí tlačítka „dnes“, se opět zobrazí aktuální měsíc. Jednotlivé úkoly tvoří odkazy umístěné v buňkách příslušných dnů. Po kliknutí na tento odkaz se zobrazí detailní stránka přehledu úkolu. Barva odkazu je závislá na prioritě úkolu. Tuto barvu lze nastavit v editačním formuláři příslušné priority úkolu.

## Hlavní menu

Hlavní menu tvoří hlavní navigační prvek aplikace. Menu je vždy umístěno v levé části okna. Některé položky hlavního menu tvoří přímo odkaz na konkrétní stránku, jiné seskupují více odkazů do skupin. Podskupina je symbolizována šipkou umístěnou v pravé části odkazu. Po stisknutí tohoto odkazu se rozbalí podmenu, které je tvořeno odkazy na jednotlivé stránky.



Obr. 24. Hlavní strana aplikace

Aktuální pozice v menu je symbolizována jinou barvou pozadí odkazu a odlišnou barvou textu.

### Vedlejší menu

Vedlejší menu je umístěno v pravém horním rohu okna aplikace a slouží primárně k zobrazení aktuálně přihlášeného uživatele a nabízí možnost odhlášení ze systému.

První položkou menu je odkaz na hlavní stranu aplikace, kde je umístěn plánovací kalendář.

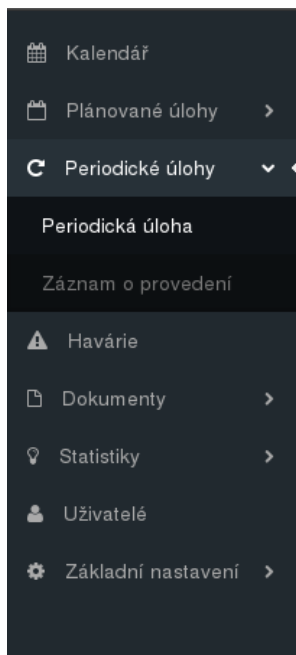
Druhou položku menu tvoří link pro odhlášení uživatele ze systému. Součástí tohoto odkazu je i jméno a příjmení aktuálně přihlášené osoby.

### Drobečková navigace

Aktuální pozice v aplikaci je vždy patrná z hlavního nadpisu a pod ním umístěné drobečkové navigace. Položky drobečkové navigace jsou zároveň odkazy na příslušné části aplikace.

### Datagrid

Výpis položek je v každé části aplikace tvořen datagridem, který skládá z tabulky, filtrů a sady operací, které se dají nad jednotlivým záznamem datagridu vykonat.





Obr. 25. Hlavní menu aplikace



Obr. 26. Vedlejší menu aplikace

Filtrování slouží k rychlému vyhledání záznamů. Filtry jsou tvořeny formulářovými prvky a jsou umístěny v hlavičce datagridu. Po vyplnění textu příslušného textového formulářového prvku a stisknutí klávesy ENTER či výběru položky z výčtového typu formulářového dojde k vyhledání příslušných záznamů.

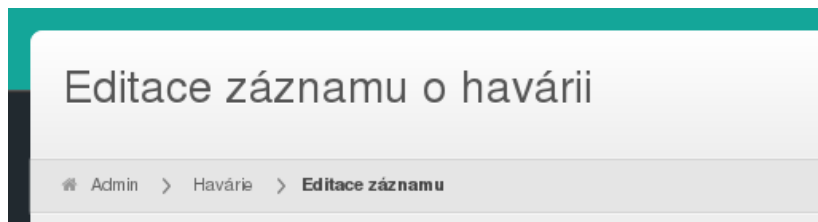
Pro práci s filtrováním slouží i tlačítka umístěná v pravé části hlavičky datagridu. Tlačítko  spouští hledání a tlačítko  (zobrazuje se pouze, pokud je filtr aktivní) zruší zadaný filtr.

Počet záznamů v datagridu je omezen na maximálně 20 zobrazených položek. Pro přístup k ostatním položkám slouží navigační prvky, umístěné v patičce datagridu.

K dispozici jsou tlačítka pro posun o jednu stránku (20 položek), pro přesun na začátek a konec seznamu.

Aktuální pozice ve výpisu je zobrazena mezi tlačítky pro posun a je složena z dvou hodnot oddělených lomítkem. První udává aktuální stránku a druhá udává počet dostupných stránek.







Akce, které lze nad položkou vykonat, jsou umístěny v pravé části řádku s položkou v okně datagridu. Jednotlivé akce zde představují ikony. Počet zobrazených ikon se liší v závislosti počtu dostupných akcí a také na uživatelských právech. Pokud uživatel vlastní pouze oprávnění na zobrazení datagridu, nebude ve sloupečku s akcemi žádná



















Obr. 27. Drobečková navigace

ikona.

Význam jednotlivých ikon je následující:

-  Úprava položky
-  Přehled úlohy
-  Smazat položky
-  Zobrazit dostupné dokumenty
-  Vytvořit záznam o schválení
-  Schválit úlohu

Název úkolu	Perioda	Umístění	Jednotka	Operace	Kontakt	Priorita	Upozornit e-mailem	
Víkend	každý víkendový den	Luh 1805	Hlavní schodiště	Kontrola		běžná	Ano	 /  /  / 
První den v měsíci	první den v měsíci	Luh 1805	Elektrický rozvaděč	Kontrola		běžná	Ano	 /  /  / 
Každý únorový den	každý den	Luh 1805	Hlavní schodiště	Úklid		běžná	Ne	 /  /  / 
Kontrola kotle - netěsní	každý den	Garáž	Plynový kotel	Kontrola		vysoká	Ne	 /  /  / 

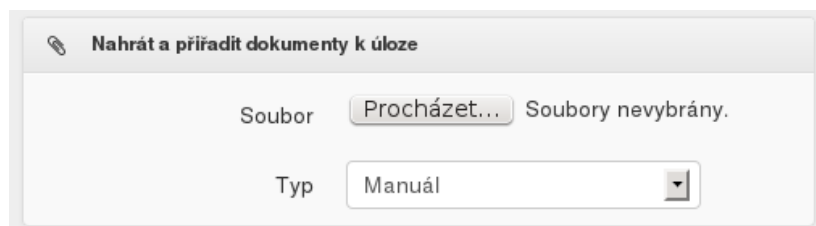
Obr. 28. Datagrid

## Editace záznamu

Každý záznam, ať už se jedná o úlohu, úkon nebo jednotku, se vytváří a upravuje v editačním okně. Editační okno je zobrazeno po stisknutí příslušného tlačítka „editovat“ v datagridu či po stisknutí tlačítka „vytvořit nový záznam“.

Editace okna je tvořena formulářem, kde v levé dolní straně formuláře jsou umístěny tlačítka pro uložení záznamu a pro návrat zpět na předchozí stránku.

Editace formulář může obsahovat část určenou pro nahrávání dokumentů. Vybrané soubory se po potvrzení formuláře uloží do systému a přiřadí se k aktuálně editovanému záznamu. Typ nahrávaných souborů se určí volbou příslušné hodnoty v seznamu umístěném v bloku pro nahrávání souborů.



Obr. 29. Výběr dokumentů

### Přehled záznamu / úlohy

Přehled úlohy je dostupný po stisknutí příslušného tlačítka v datagridu či po kliknutí na úlohu v plánovacím kalendáři. Na stránce s přehledem se zobrazují nejdůležitější informace, které se vážou ke konkrétnímu záznamu.

Pokud je vyvolán přehled úlohy, zobrazují se informace o zadání úkolu, přiřazené kontaktní osobě a přiřazené dokumenty.

V případě kontaktní osoby se zobrazuje seznam všech úkolů, pro které je tato kontaktní osoba vybrána jako řešitel.

Pokud je zvolen dokument, zobrazují se všechny záznamy, ke kterým je zvolený dokument přiřazen.

Ve spodní části stránky s přehledem je vždy umístěno tlačítko pro editaci záznamu a pro návrat na předchozí stránku.

### Správa dokumentů položky

Okno správy dokumentů se skládá z datagridu, ve kterém se zobrazují všechny dokumenty evidované v systému. V pravé části každé položky v datagridu je dostupná funkce přiřazení resp. odřazení dokumentu k záznamu.

Akce provedená po stisknutí tlačítek:

- Odřadit dokument
- Přiřadit dokument

Ve spodní části obrazovky jsou dostupné tlačítka pro úpravu záznamu a pro návrat na předchozí stranu.

### **Statistiky**

Statistiky jsou dostupné v hlavním menu aplikace pod položkou „Statistiky“. Okno statistik se skládá z formuláře obsahujícího prvky pro volbu datumu a umístění. Po stisknutí tlačítka „Zobraz“, se zobrazí pod formulářem podrobný přehled úkolů, rozdělený do čtyřech boxů. Boxy reprezentují shrnutí nákladů na jednotku, kontakt, úkon a celek. Úkoly jsou v rámci boxu seskupeny a seřazeny dle měsíce realizace.

## 7 NÁVRH STRUKTURY APLIKACE

### 7.1 Modelová vrstva

Nette Framework je určen k tvorbě webových aplikací s architekturou MVC. Podporuje všechny jednotlivé vrstvy, nicméně jak jsem uváděl v teoretické části, modelová vrstva není nijak zobecněna. Prvním krokem při realizaci bude tedy příprava modelové vrstvy. Tato vrstva bude sloužit k jednotnému přístupu k datům uložených v databázi.

Pro přístup k databázi použijí objekt *Nette\Database\Context*.

Modelovou vrstvu bude tvořit sada objektů pojmenovaných „repozitář“ a umístěných do adresáře *Repository* v kořenovém adresáři aplikace. Každý jednotlivý repozitář bude mít v názvu postfix „Repository“ a bude dědit od abstraktního objektu *Repository*. Objekt *Repository* bude obsahovat společné metody pro všechny repozitáře.

Zvláštní význam má metoda *getTable*, vracějící klonovanou instanci objektu z proměnné *\$table*.

```

1 /**
2  * Vrací objekt reprezentující databázovou tabulku.
3  *
4  * @return Nette\Database\Table\Selection
5  */
6 protected function getTable() {
7     return clone $this->table;
8 }
```

V proměnné *\$table* je totiž uložena instance objektu *Nette\Database\Table\Selection*, která se do každého repozitáře vkládá v jejím konstruktoru.

Pokud nad instancí objektu *Nette\Database\Table\Selection* zavoláme například metodu *where* s libovolným argumentem, stává zadaný argument součástí této instance. Po opětovném volání metody *where* by došlo pouze k rozšíření již dříve zadané podmínky. Následující příklad demonstruje toto chování.

```

1 $table = $connection->table('user');
2
3 // SQL dotaz bude SELECT * FROM 'user' WHERE 'username' = 'jan'
4 $selection = $table->where('username', 'jan');
5
6 // SQL dotaz bude SELECT * FROM 'user' WHERE 'username' = 'jan' AND
7 // 'username' = 'tomas'
7 $selection = $table->where('username', 'tomas');
```

Některé metody umístěné v rámci repozitáře svým významem přímo do modelové vrstvy nepatří. Jedná se například o metody použité pro obsluhu datagridu. Modelová vrstva by totiž neměla zpřístupňovat data, dle konkrétních požadavků prezentační vrstvy, nicméně hranici modelové vrstvy netvoří konkrétní objekt, ale význam. Umístění těchto metod přímo do repozitáře jsem volil z ohledu na praktičnost tohoto řešení.

Pro vyhodnocení platnosti úkolu v rámci periodických úloh, obsahuje repozitář *Peri-*

*odicTaskRepository* metodu *checkPeriod* s parametry *\$period* a *\$date*. Tato metoda vrátí *true* v případě, že zadané datum odpovídá zadané periodě.

## 7.2 Řídící vrstva

Řídící člen aplikace představují presentery. Každá url adresa je ve Frameworku Nette přeložena jako volání konkrétního presenteru a příslušné akce nad tímto presenterem. Životní cyklus presenteru byl popsán v teoretické části.

Presenter v Nette frameworku představuje třída, která dědí od *Nette\Application\UI\Presenter*. Pro potřeby aplikace vytvářím vlastní abstraktní třídu *BasePresenter*. Abstraktní třída *BasePresenter* bude použita jako rodič pro všechny presentery, které budou v aplikaci použity. Důvodem této abstrakce je například zajištění zabezpečení aplikace. Třída *BasePresenter* implementuje v metodě *startup* kontrolu přihlášeného uživatele a ověřuje, zda uživatel disponuje dostatečnými právy k vykonání požadované akce.

Třída *BasePresenter* také obsahuje kód pro vytvoření komponenty *WebLoader*, která slouží ke spojování a minimalizaci CSS a JavaScriptových souborů.

## 7.3 Uživatelské role

Pro práci s uživatelskými rolemi využívám dostupných funkcí, které jsou součástí Nette frameworku. Tyto funkce poskytují veškerou podporu, kterou požadujeme pro zabezpečení aplikace před neoprávněným přístupem a manipulací s daty.

Proces autentifikace je zajištěn metodou *authenticate* v třídě *UserManager* implementující rozhraní *Nette\Security\IA Authenticator*. Implementací rozhraní *IA Authenticator* říkáme Nette Frameworku, že má použít tuto třídu jako výchozí autentifikátor. Toto chování je v případě nutnosti možné potlačit manuálním nastavením autentifikátoru v konfiguračním souboru.

Metoda *authenticate* v případě úspěšného přihlášení vrací instanci třídy *Identity*, implementující rozhraní *Nette\Security\IIdentity*. Rozhraní *IIdentity* představuje uživatelovu identitu. Instanci této třídy byla při vytvoření předaná uživatelská role. Tato role se následně vyhodnocuje použitým autorizátorem v procesu autorizace.

Jako autorizátor se využívá třída *Nette\Security\Permission*, implementující rozhraní *Nette\Security\IAuthorizator*. Inicializace autorizátoru probíhá v konfiguračním souboru aplikace.

Vyhodnocení uživatelských oprávnění se provádí metodou *isAllowed* nad instancí třídy *Nette\Security\User*. Metoda *isAllowed* vyžaduje dva parametry. Prvním parametrem je název oblasti. Druhým parametrem je název činnosti. Objekt *User* představuje v systému uživatele, který právě pracuje s aplikací. Objekt *User* obsahuje všechny výše

uvedené instance tříd.

Definice uživatelského oprávnění se skládá ze tří částí. Jsou to:

- role
- oblast
- činnost

Tímto způsobem lze konkrétnímu uživateli disponujícímu konkrétní uživatelskou rolí povolit vstup do konkrétní oblasti a povolit v této oblasti provedení konkrétní činnosti.

### **Definice rolí**

- Uživatel bez oprávnění (guest)
- Pomocník (maintenance)
- Facility manager (manager)
- Správce bez omezení uživatelských oprávnění (admin)

### **Definice oblastí**

Výčet oblastí volím dle jednotlivých logických celků aplikace tak, aby mohla být vždy zpřístupněna pouze určitá část aplikace.

- Úkoly (task)
- Statistiky (statistic)
- Nastavení úkonů, jednotek, umístění (setup)
- Dokumenty (document)
- Uživatelé systému (user)

### **Definice činností**

Výčet činností odráží všechny možnosti, které lze v systému s jednotlivými záznamy provést.

- Prohlížení (view)
- Vytvoření nového záznamu (create)
- Vytvořit záznam o realizaci (realize)
- Vymazat záznam (delete)

## Sestavení kompletního modelu

Na základě výše uvedených definic je seskládán výsledný kód, který je umístěn v konfiguračním souboru aplikace.

```
1 authorizator:
2   class: Nette\Security\Permission
3   setup:
4     - addRole('guest')
5     - addRole('maintenance', 'guest')
6     - addRole('manager', 'maintenance')
7     - addRole('admin', 'manager')
8     - addResource('task')
9     - addResource('statistic')
10    - addResource('setup')
11    - addResource('document')
12    - addResource('user')
13    - allow('guest', 'task', 'view')
14    - allow('maintenance', 'task', 'edit')
15    - allow('maintenance', 'setup', 'view')
16    - allow('maintenance', 'document', 'view')
17    - allow('manager', 'task', ['create', 'realize'])
18    - allow('manager', 'statistic', 'view')
19    - allow('admin', Nette\Security\Permission::ALL,
  Nette\Security\Permission::ALL)
```

## 8 TESTY

U vývoje aplikace dochází velmi často k situacím, kdy je nutné dříve napsaný kód refaktorovat. Refaktorování je přitom velmi důležitou částí procesu realizace aplikace. Při refaktorování je ovšem možné udělat chybu v částí kódu, kterou jsme již dříve odladili.

Automatizované testy představují způsob, jakým lze kdykoliv otestovat již napsaný kód a určit, zda pracuje jak očekáváme. Při návrhu aplikace a hlavně při vlastním psaní kódu, musíme neustále myslet na to, že pro každou část kódu by jsme mohli potřebovat kdykoliv vytvořit test. Je tedy vhodné vždy rozdělit řešenou problematiku na dílčí části, kdy jednotlivou část začleníme do jedné funkce. Každá funkce bude tedy vykonávat jen dílčí operaci. Složením těchto funkcí dohromady získáme řešení zpracovávané úlohy. Pro každou dílčí operaci je pak možné napsat jednoduchý test, který jasně prokáže, že funkce pracuje v rámci stanovených parametrů. Pokud bychom stejný úkol zpracovávali v rámci jedné metody, test by nám prokázal, že se v metodě vyskytla chyba, nicméně metoda je tak dlouhá a komplikovaná, že nalezení chyby by trvalo podstatně déle.

Velmi vhodné bude vytvořit automatizované testy, které pokryjí problematiku zabezpečení aplikace. Tyto testy například při modifikaci uživatelských oprávnění jasně prokážou, že nebyla narušena bezpečnost aplikace, tedy že neoprávněná osoba bude disponovat chybnými uživatelskými právy.

### 8.1 Testování přihlášení

Test pro přihlášení uživatelů se skládá ze tří částí. Test č. 1 testuje, zda se správně přihlásí uživatel *admin*. Test č. 2 testuje, zda se správně vyhodnotí, že zadané heslo pro uživatele *admin* je chybné. Test č. 3 ověřuje, že uživatel *admin* disponuje správnou uživatelskou rolí.

Uživatelé a role byly naplněny tímto SQL skriptem.

```
1 INSERT INTO 'role' VALUES ('superadmin', 'Admin'), ('guest', 'Guest');
2 INSERT INTO 'user' VALUES (0, 'admin', SHA1('password'), '', '', '', 0,
  'superadmin');
```

#### Test 1

```
1 Assert::type('Nette\Security\Identity',
  $this->userManager->authenticate(array('admin', 'password')));
```

#### Test 2

```
1 Assert::exception(function () {
2     $this->userManager->authenticate(array('jan', 'somepassword'));
3 }, 'Nette\Security\AuthenticationException');
```

### Test 3

```
1 $user = $this->userManager->authenticate(array('admin', 'password'));
2 Assert::same(['superadmin'], $user->getRoles());
```

## 8.2 Testování uživatelských práv

Pro testování uživatelských oprávnění je nutné nejprve sestavit jejich model. Model sestavím do následující podoby.

```
1 $this->permission = new Nette\Security\Permission();
2 $this->permission->addRole('guest');
3 $this->permission->addRole('maintenance', 'guest');
4 $this->permission->addRole('manager', 'maintenance');
5 $this->permission->addRole('admin', 'manager');
6 $this->permission->addResource('task');
7 $this->permission->addResource('statistic');
8 $this->permission->addResource('setup');
9 $this->permission->addResource('document');
10 $this->permission->addResource('user');
11 $this->permission->allow('guest', 'task', 'view');
12 $this->permission->allow('maintenance', 'task', 'edit');
13 $this->permission->allow('maintenance', 'setup', 'view');
14 $this->permission->allow('maintenance', 'document', 'view');
15 $this->permission->allow('manager', 'task', ['create', 'realize']);
16 $this->permission->allow('manager', 'statistic', 'view');
17 $this->permission->allow('admin', Nette\Security\Permission::ALL,
    Nette\Security\Permission::ALL);
```

Testování bude probíhat tak, že budeme volat metodu *isAllowed* a kontrolovat návratovou hodnotu dle očekávané hodnoty.

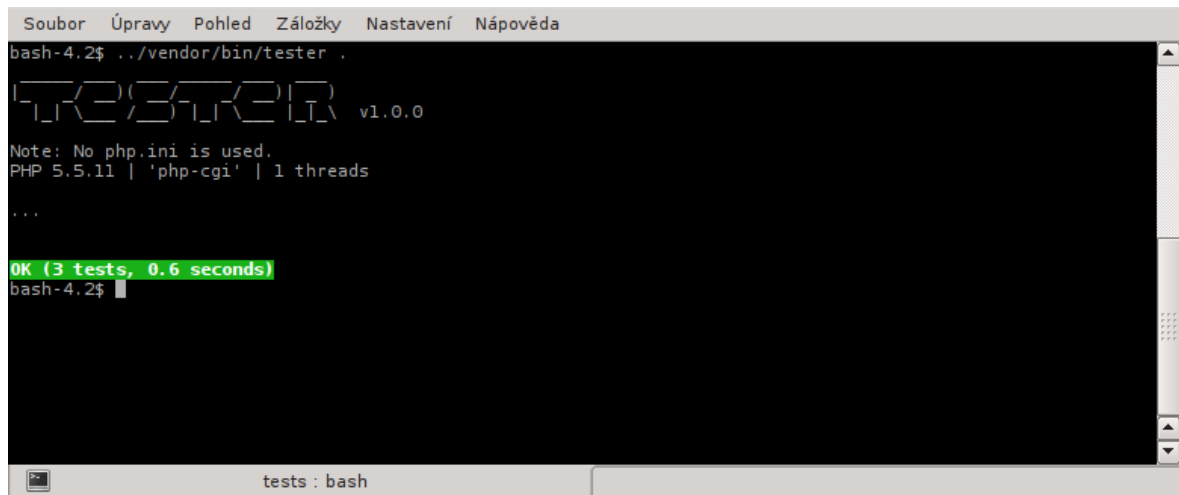
```
1 Assert::true($this->permission->isAllowed('guest', 'task', 'view'));
2 Assert::false($this->permission->isAllowed('guest', 'task', 'edit'));
3 Assert::true($this->permission->isAllowed('maintenance', 'task',
    'view'));
4 Assert::true($this->permission->isAllowed('maintenance', 'task',
    'edit'));
5 Assert::false($this->permission->isAllowed('maintenance', 'user',
    'edit'));
```

## 8.3 Testování periody úkolu

Testování periody úkolu spočívá ve volání metody *checkPeriod* (metoda je uvedena v příloze), kdy kontrolujeme očekávanou návratovou hodnotu po zadání patřičných parametrů na vstup funkce.

```
1 for ($n = 0; $n < 60; $n++) {
2     $date = new \DateTime();
3     $date->setTimestamp(rand(0, time()));
4     Assert::true($this->periodicTaskRepository->checkPeriod(0, $date));
5 }
6
7 Assert::false($this->periodicTaskRepository->checkPeriod(1, new
    \DateTime('2014-04-06')));
```

```
8 Assert::true($this->periodicTaskRepository->checkPeriod(1, new
  \DateTime('2014-04-07')));
9 Assert::true($this->periodicTaskRepository->checkPeriod(2, new
  \DateTime('2014-04-06')));
10 Assert::false($this->periodicTaskRepository->checkPeriod(2, new
  \DateTime('2014-04-07')));
```



```
Soubor  Úpravy  Pohled  Záložky  Nastavení  Nápověda
bash-4.2$ ./vendor/bin/tester .
TREBARIER v1.0.0
Note: No php.ini is used.
PHP 5.5.11 | 'php-cgi' | 1 threads
...
OK (3 tests, 0.6 seconds)
bash-4.2$
```

Obr. 30. Výsledek spuštěných testů. Všechny testy proběhly správně.

## ZÁVĚR

Cílem této práce bylo vytvoření návrhu informačního systému určeného pro správu budov. Pro úspěšné splnění cíle bylo nutné se nejdříve seznámit se službami facility managementu, které jsou potřebné pro zajištění maximálně efektivní podpory při správě budov. Po seznámení se s rozsahem a charakterem FM služeb, jsem mohl navrhnout funkce informačního systému tak, aby těmto službám zajišťovaly co nejvyšší podporu. Získáním přehledu o funkcích a vlastnostech, které má informační systém obsahovat, bylo vhodné zaměřit se na hledání hotových řešení nabízených na trhu s informačními systémy. Bohatá nabídka hotových řešení ukázala, že správa budovy s podporou služeb facility managementu, představují jasnou cestu, kudy by se měly organizace vydat při cestě k optimalizaci a úsporám. Finanční náklady spojené s pořízením těchto aplikací převyšující náklady na tvorbu vlastního systému, mě motivovaly k tvorbě vlastního systému, který se na rozdíl od konkurence vyznačuje nulovými, či velmi nízkými pořizovacími a provozními náklady. Další nezanedbatelnou oblast tvoří možnost snadného doplnění programových funkcí, či možnost přizpůsobení systému na míru. Díky použitému konceptu webové aplikace, nejsou požadavky na výkon či programové vybavení klientské stanice nikterak vysoké. Postačí běžný internetový prohlížeč a připojení k síti Internet.

U návrhu a tvorby informačního systému, jsem se řídil zkušenostmi ze své praxe tvorby webových aplikací. Díky široké dostupnosti sítě Internet, jsem zvolil pro informační systém formát webové aplikace a použil kombinaci skriptovacího jazyka PHP a databázového systému MySQL, které jsou pro webové aplikace nejpoužívanější. Aplikace je vytvořena za použití českého PHP frameworku Nette a je navržena s ohledem na charakter architektury MVC.

Prvním krokem návrhu aplikace představovalo slovní pojmenování všech objektů a funkcí, které budou v systému použity. Tyto funkce vychází ze znalostí služeb facility managementu. Následným krokem byla tvorba ER modelu databáze. Aplikace totiž využívá k zajištění perzistence dat relační databázi, a tak model databáze tvoří při vývoji nezbytnou součást. V ER modelu jsou jasně zřetelné všechny entity a také příslušné relace. Získáním přehledu o struktuře databáze, jsem mohl přistoupit k tvorbě EER modelu. Tento model již obsahuje všechny potřebné atributy a stanovuje primární klíče. EER model má také vyřešeny relace mezi entitami za pomoci vazebních tabulek. Ze získaných znalostí kompletního modelu databáze, jsem mohl přistoupit k návrhu uživatelského rozhraní. Cílem při tvorbě rozhraní byla jednoduchost a uživatelská přívětivost. Vytyčeného cíle se mi podařilo dosáhnout za použití moderního UI frameworku Twitter Bootstrap a skriptovacího frameworku JQuery. Posledním krokem při realizaci aplikace bylo vlastní programování zdrojového kódu. Při návrhu struktury kódu aplikace jsem kladl důraz na modulárnost aplikace a především na možnou tes-

tovatelnost.

Vytvořený informační systém umožňuje evidenci a správu služeb potřebných při správě budovy. Mezi tyto služby se řadí plánování úkolů, evidence havárií, evidence kontaktů na interní i externí pracovníky a evidence dokumentace ve formě souborů. Plánování úkolů je rozděleno na dva typy. Prvním typem jsou periodické úlohy. Pro tyto úlohy charakteristická pevná perioda opakování. Druhým typem jsou úkoly, které jsou plánované pro konkrétní datum. Tyto úkoly jsou určeny pro jednorázové operace. Součástí aplikace je i evidence havárií. Havárie jsou zde chápány jako neplánované činnosti, které je nutné z důvodů zpětné vazby evidovat. Aplikace umožňuje sestavit přehled (statistiku) za zvolené období, který znázorňuje náklady na jednotlivé zařízení a provedené úkony. Tyto přehledy jsou užitečné například pro stanovení poruchovosti zařízení, kdy včasnou výměnou je možné snížit náklady na servis tohoto zařízení.

Po naplnění aplikace zkušebními záznamy, se povedlo odstranit některé drobné nedostatky aplikace. Simulací reálné práce také vznikly nové požadavky, které byly do aplikace úspěšně začleněny. Na zkušebních záznamech se prokázala funkčnost aplikace a schopnost jejího nasazení do reálného provozu.

Vytyčený úkol, představující tvorbu informačního systému určeného pro správu budov, se podařilo úspěšně naplnit. Při tvorbě aplikace jsem nemusel řešit žádné překážky, které by měly na aplikaci omezující charakter či bránily jejímu úspěšnému dokončení. Tento fakt je důležitý pro následný další vývoj aplikace. Po nasazení systému do ostrého provozu je očekávána řada podmětů ze strany uživatelů na zpříjemnění uživatelského ovládání či na úpravu jednotlivých funkčních bloků. Díky použití moderního frameworku s podporou architektury MVC, lze pružně reagovat na požadavky uživatelů a zajistit úpravu či tvorbu nových programových funkcí aplikace.

## SEZNAM POUŽITÉ LITERATURY

- [1] DANIELS, Klaus. *Technika budov: příručka pro architekty a projektanty*. 1. čes. vyd. Bratislava: Jaga group, 2003. ISBN 80-889-0563-X
- [2] KUDA, František a Eva BERÁNKOVÁ. *Facility management v technické správě a údržbě budov*. 1. vyd. Praha: Professional Publishing, 266 s. ISBN 978-807-4311-147
- [3] ČSN EN 15221-1. *Facility management: Část 1: Termíny a definice*. ČESKÝ NORMALIZAČNÍ INSTITUT, Praha, 2007
- [4] Facility Management. *IFMA CZ* [online]. [cit. 2014-03-19]. Dostupné z: <http://www.ifma.cz/index.php/facility-management/co-je-facility-management/166-facility-management?start=12>
- [5] MERZ, Hermann, Thomas HANSEMANN a Christof HÜBNER. *Automatizované systémy budov: sdělovací systémy KNX/EIB, LON a BACnet*. 1. vyd. Praha: Grada, 2008. ISBN 978-80-247-2367-9
- [6] Kořeny Facility Managementu. *IFMA CZ* [online]. [cit. 2014-03-18]. Dostupné z: <http://www.ifma.cz/index.php/facility-management/historie-fm/164-koeny-facility-managementu>
- [7] Stručná historie IFMA ve světě. *IFMA CZ* [online]. [cit. 2014-03-18]. Dostupné z: <http://www.ifma.cz/index.php/facility-management/historie-fm/163-stru4n8-historie-ifma-ve-svt>
- [8] ŠTRUP, Ondřej a Viera SOMOROVÁ. *Definice a terminologie FM* [prezentace PowerPoint]. 2009 [cit. 2014-05-03]
- [9] ČSN EN 15221-2. *Facility management: Část 2: Průvodce přípravou smluv o facility managementu*. ČESKÝ NORMALIZAČNÍ INSTITUT, Praha, 2007
- [10] Nové čtyři díly evropské normy EN 15221 jsou na světě. *E15.cz / magazín* [online]. 2011 [cit. 2014-05-20]. Dostupné z: <http://magazin.e15.cz/bydleni/nove-ctyri-dily-evropske-normy-en-15221-jsou-na-svete-976948>
- [11] Credentials for Facility Managers. [online]. [cit. 2014-05-02]. Dostupné z: <http://www.ifma.org/professional-development/credentials>
- [12] APFELTHALER, Jan. *Facility management v ČR - Trh a podpora nástrojů IS/ICT 2007*. Diplomová práce. Vysoká škola ekonomická v Praze

- [13] JANSSEN, Cory. Computerized Maintenance Management System (CMMS). [online]. [cit. 2014-04-27]. Dostupné z: <http://www.techopedia.com/definition/25281/computerized-maintenance-management-system-cmms>
- [14] HAMPL, Milan a Ondřej ŠTRUP. CAFM systémy - IT podpora facility managementu. [online]. [cit. 2014-04-27]. Dostupné z: <http://www.cad.cz/pdmp/m/7-2007/1311-cafm-systemy-it-podpora-facility-managementu.html>
- [15] Company: Information. *Archibus* [online]. [cit. 2014-04-27]. Dostupné z: <http://www.archibus.com/Information>
- [16] Top 5 Web Application Languages. *Red27 Consulting* [online]. 2011 [cit. 2014-05-24]. Dostupné z: <http://www.red27.net/2011/01/26/top-5-web-application-languages/>
- [17] BERNARD, Borek. Úvod do architektury MVC. [online]. 2009 [cit. 2014-03-19]. Dostupné z: <http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/>
- [18] MVC aplikace & presentery. NETTE [online]. [cit. 2014-03-19]. Dostupné z: <http://doc.nette.org/cs/2.1/presenters>
- [19] PANKRÁČ, Miloslav. *PHP a MySQL: bez předchozích znalostí : [průvodce pro samouky]*. Vyd. 1. Brno, 2007, 221 s. ISBN 978-80-251-1758-3
- [20] WELLING, Luke a Laura THOMSON. *PHP and MySQL Web development*. 2nd ed. Indianapolis, Ind.: Sams, c2003. ISBN 06-723-2525-X
- [21] KOFLER, Michael. Using MySQL licensing: Open source license vs. commercial license. [online]. [cit. 2014-03-20]. Dostupné z: <http://searchitchannel.techtarget.com/feature/Using-MySQL-licensing-Open-source-license-vs-commercial-license>
- [22] An Introduction To MySQL Storage Engines. In: *Linux.org* [online]. 2013 [cit. 2014-03-23]. Dostupné z: <http://www.linux.org/threads/an-introduction-to-mysql-storage-engines.4220/>
- [23] InnoDB as the Default MySQL Storage Engine. *MySQL* [online]. [cit. 2014-03-23]. Dostupné z: <http://dev.mysql.com/doc/refman/5.5/en/innodb-default-se.html>
- [24] MySQL and the ACID Model. *MySQL* [online]. [cit. 2014-03-23]. Dostupné z: <https://dev.mysql.com/doc/refman/5.6/en/mysql-acid.html>
- [25] MariaDB Documentation. *MariaDB* [online]. [cit. 2014-03-23]. Dostupné z: <https://mariadb.com/kb/en/mariadb-documentation/>

- [26] PEARCE, Rohan. Dead database walking: MySQL's creator on why the future belongs to MariaDB. *Computerworld* [online]. roč. 2013 [cit. 2014-03-23]. Dostupné z: [http://www.computerworld.com.au/article/457551/dead\\_database\\\_walking\\\_mysql\\\_creator\\\_why\\\_future\\\_belongs\\\_mariadb/](http://www.computerworld.com.au/article/457551/dead_database\_walking\_mysql\_creator\_why\_future\_belongs\_mariadb/)
- [27] MariaDB versus MySQL - Compatibility. *MariaDB* [online]. [cit. 2014-03-23]. Dostupné z: <https://mariadb.com/kb/en/mariadb-versus-mysql-compatibility/>
- [28] MariaDB versus MySQL - Features. *MariaDB* [online]. [cit. 2014-03-23]. Dostupné z: <https://mariadb.com/kb/en/mariadb-vs-mysql-features/>
- [29] Distributions Which Include MariaDB. *MariaDB* [online]. [cit. 2014-03-23]. Dostupné z: <https://mariadb.com/kb/en/distributions-which-include-mariadb/>
- [30] ŠVIRGA, Václav. Modelování MySQL databáze pomocí MySQL Workbench. [online]. 2013 [cit. 2014-03-22]. Dostupné z: <http://www.abclinuxu.cz/clanky/modelovani-mysql-databaze-pomoci-mysql-workbench>
- [31] MySQL Workbench. *MySQL* [online]. [cit. 2014-03-22]. Dostupné z: <http://www.mysql.com/products/workbench/>
- [32] Rychlý a pohodlný vývoj webových aplikací v PHP — Nette Framework. *Nette* [online]. [cit. 2014-03-22]. Dostupné z: <http://nette.org/cs/>
- [33] Licenční politika. *Nette* [online]. [cit. 2014-03-22]. Dostupné z: <http://nette.org/cs/license>
- [34] Nette Framework. *Wikipedia: the free encyclopedia*. [online]. 2001- [cit. 2014-03-22]. Dostupné z: [http://cs.wikipedia.org/wiki/Nette\\_Framework](http://cs.wikipedia.org/wiki/Nette_Framework)
- [35] *Nette Foundation: About the Foundation* [online]. [cit. 2014-03-22]. Dostupné z: <http://nettefoundation.com/>
- [36] GILMORE, W. *Velká kniha PHP 5 a MySQL: kompendium znalostí pro začátečníky i profesionály*. Nové, 3. vyd. Překlad Jan Pokorný. Brno: Zoner Press, 2011, 736 s. Encyklopedie Zoner Press. ISBN 978-80-7413-163-9
- [37] Databáze & ORM. *Nette* [online]. [cit. 2014-03-22]. Dostupné z: <http://doc.nette.org/cs/2.1/database>
- [38] GRUDL, David. Dibi vs. Nette Database story. *phpFashion* [online]. [cit. 2014-03-22]. Dostupné z: <http://phpfashion.com/dibi-vs-nette-database-story>

- [39] Šablony. *Nette* [online]. [cit. 2014-03-22]. Dostupné z: <http://doc.nette.org/cs/2.1/templating>
- [40] About: History. *Bootstrap* [online]. [cit. 2014-03-22]. Dostupné z: <http://getbootstrap.com/about/>
- [41] Getting started: License FAQs. *Bootstrap* [online]. [cit. 2014-05-24]. Dostupné z: <http://getbootstrap.com/getting-started/#license-faqs>
- [42] About jQuery. *jQuery* [online]. [cit. 2014-05-24]. Dostupné z: <http://learn.jquery.com/about-jquery/>
- [43] About jQuery UI. *jQuery UI* [online]. [cit. 2014-05-24]. Dostupné z: <http://jqueryui.com/about/>
- [44] ZAMRZLA, Josef. Testování a tvorba testovatelného kódu v PHP. *Zdroják.cz* [online]. 2012 [cit. 2014-03-22]. Dostupné z: <http://www.zdrojak.cz/clanky/testovani-a-tvorba-testovatelného-kódu-v-php/>
- [45] Nette Tester - pohodové testování. *Nette* [online]. [cit. 2014-05-08]. Dostupné z: <http://tester.nette.org/cs/>
- [46] TEMPLATEMONSTER.COM, Inc. *TemplateMonster* [online]. [cit. 2014-05-03]. Dostupné z: <http://www.templatemonster.com/>

**SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK**

CAD	Computer-aided design
CAFM	Computer-Aided Facility Management
CMMS	Computerized maintenance management system
CSS	Cascading Style Sheets
DI	Dependency Injection
ERP	Enterprise Resource Planning
FM	Facility management
GIS	Geographic information system
GPL	General Public Licence
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IS	Informační systém
JS	JavaScript
JSON	JavaScript Object Notation
LSB	Least Significant Bit
MIME	Multipurpose Internet Mail Extensions
MVC	Architektura Model - View - Controller
PDO	PHP Data Objects
SŘBS	Systém řízení báze dat
VPN	Virtual Private Network
XML	Extensible Markup Language
XSS	Cross-site scripting

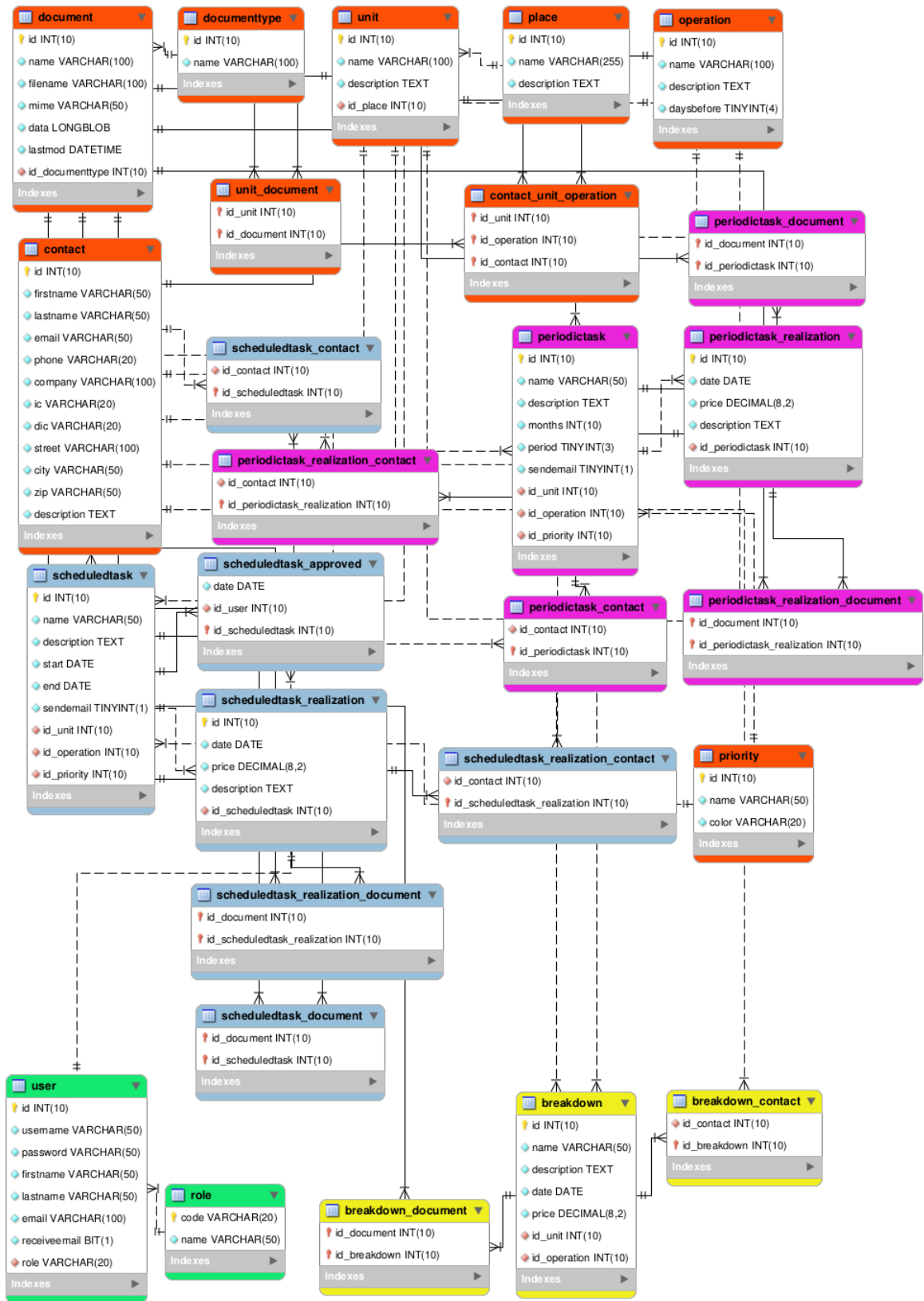
**SEZNAM OBRÁZKŮ**

Obr. 1. Facility management: 3P .....	12
Obr. 2. MVC .....	20
Obr. 3. Aplikace MySQL Workbench .....	24
Obr. 4. Životní cyklus presenteru[18] .....	26
Obr. 5. Příklad schématu databáze .....	27
Obr. 6. Entity pro objekt zájmu a umístění .....	38
Obr. 7. Entita pro prováděnou činnost .....	38
Obr. 8. Entita pro kontakt .....	38
Obr. 9. Entita pro prioritu .....	39
Obr. 10. Entity pro dokument a jeho typ .....	39
Obr. 11. Entita pro plánovanou úlohu .....	39
Obr. 12. Entita pro záznam o provedení plánované úlohy .....	40
Obr. 13. Entita pro periodickou úlohu .....	40
Obr. 14. Entita pro záznam o provedení periodické úlohy .....	41
Obr. 15. Entita pro záznam o havárii .....	41
Obr. 16. Entity pro uživatele a uživatelskou roli .....	41
Obr. 17. EER schéma dokumentů .....	42
Obr. 18. EER schéma základních entit .....	43
Obr. 19. EER schéma uživatelů .....	44
Obr. 20. EER schéma plánované úlohy .....	45
Obr. 21. EER schéma periodické úlohy .....	46
Obr. 22. EER schéma havárií .....	46
Obr. 23. Přihlášení do aplikace .....	48
Obr. 24. Hlavní strana aplikace .....	49
Obr. 25. Hlavní menu aplikace .....	50
Obr. 26. Vedlejší menu aplikace .....	50
Obr. 27. Drobečková navigace .....	51
Obr. 28. Datagrid .....	51
Obr. 29. Výběr dokumentů .....	52
Obr. 30. Výsledek spuštěných testů. Všechny testy proběhly správně. ....	60

## SEZNAM PŘÍLOH

- P I. Celkové EER schéma databáze
- P II. Funkce pro vyhodnocení datumu

# PŘÍLOHA P I. CELKOVÉ EER SCHEMA DATABÁZE



## PŘÍLOHA P II. FUNKCE PRO VYHODNOCENÍ DATUMU

```
1 /**
2  * Pro zadanou periodu otestuj datum, zda je platne
3  *
4  * @param string $period Perioda
5  * @param \DateTime $date Datum
6  * @throws \UnknownPeriod
7  * @return boolean
8  */
9 function checkPeriod($period, $date) {
10     $datetime = clone $date;
11
12     /*
13      *      0 'každý den',
14      *      1 'každý pracovní den',
15      *      2 'každý víkendový den',
16      *      3 'první den v měsíci',
17      *      4 'první pracovní den v měsíci',
18      *      5 'první pracovní den v týdnu',
19      *      6 'první víkendový den v týdnu',
20      *      7 'poslední pracovní den v týdnu',
21      *      8 'poslední den v měsíci',
22      *      9 'poslední pracovní den v měsíci'
23      *     10 'první víkendový den v měsíci'
24     */
25
26     switch ($period) {
27         case 0:
28             return true;
29
30         case 1:
31             return (int)$datetime->format('N') < 6;
32
33         case 2:
34             return (int)$datetime->format('N') > 5;
35
36         case 3:
37             return (int)$datetime->format('j') == 1;
38
39         case 4:
40             $datetime->modify('+ ' . (1 - $datetime->format('j')) . ' day');
41
42             switch ($datetime->format('N')) {
43                 case 6:
44                     $datetime->modify('+2 day');
45                     break;
46
47                 case 7:
48                     $datetime->modify('+1 day');
49                     break;
50             }
51
52             return $datetime->diff($date)->days == 0;
53
54         break;
55
56         case 5:
57             return (int)$datetime->format('N') == 1;
58
59         case 6:
60             return (int)$datetime->format('N') == 6;
61
62         case 7:
63             return (int)$datetime->format('N') == 5;
64
65         case 8:
66             $datetime->modify('+ ' . ($datetime->format('t') -
67 $datetime->format('j')) . ' day');
68
69             return $datetime->diff($date)->days == 0;
70
71         case 9:
72             $datetime->modify('+ ' . ($datetime->format('t') -
73 $datetime->format('j')) . ' day');
```

```

72
73     switch ($datetime->format('N')) {
74         case 6:
75             $datetime->modify('-1 day');
76             break;
77         case 7:
78             $datetime->modify('-2 day');
79             break;
80     }
81
82     return $datetime->diff($date)->days == 0;
83
84 case 10:
85     $datetime->modify('+ ' . (1 - $datetime->format('j')) . ' day');
86
87     switch ($datetime->format('N')) {
88         case 1:
89         case 2:
90         case 3:
91         case 4:
92         case 5:
93             $datetime->modify('+ ' . (6 - $datetime->format('N')) . '
94 day');
95             break;
96         case 7:
97             $datetime->modify('+6 day');
98             break;
99     }
100
101     return $datetime->diff($date)->days == 0;
102
103 default:
104     throw new \UnknownPeriod("Unknown period: " . $period);
105 }
106 }
107 }

```